НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ

Інститут кібернетики ім. В. М. Глушкова

Кваліфікаційна наукова
праця на правах рукопису

ЛІТВІН АННА АНДРІЇВНА

УДК 681.3.06

ДИСЕРТАЦІЯ

МЕТОДИ ТА ЗАСОБИ АНАЛІЗУ І СИНТЕЗУ ПРИРОДНОМОВНИХ ТЕКСТІВ НА ОСНОВІ ОНТОЛОГІЧНОЇ МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ В АСПЕКТІ СТВОРЕННЯ ДІАЛОГОВИХ СИСТЕМ

Спеціальність 122 – комп’ютерні науки

Подається на здобуття ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Науковий керівник
Палагін Олександр Васильович
д.т.н., проф., академік НАН України

Київ – 2023

АНОТАЦІЯ

Літвін А. А. Методи та засоби аналізу і синтезу природномовних текстів на основі онтологічної моделі предметної області в аспекті створення діалогових систем. — Рукопис.

Дисертація на здобуття вченого ступеня доктора філософії за спеціальністю 122 – комп'ютерні науки – Інститут кібернетики ім. В. М. Глушкова НАН України, Київ, 2023.

Дисертаційна робота присвячена розробці методів та засобів автоматизованого аналізу та синтезу природномовних текстів на основі онтологічної моделі в аспекті створення діалогових систем.

У роботі створено метод автоматизованого формування запитів до онтології на основі семантичного аналізу природномовних текстів. Особливість підходу полягає в тому, що формальний запит автоматично будується із блоків шаблонів, що є такими, що налаштовуються, відповідно семантичних категорій визначених у аналізованому тексті.

Розвинено методи автоматичної побудови онтологій на основі глибокого синтактико-семантичного аналізу природномовного тексту. Створено методи для генерації відповідей за допомогою шаблонів гнучкої структури на основі сукупності набору понять, отриманих через виконання формального запиту до бази знань та відібраних із вхідної фрази.

Створено підхід, що базується на застосуванні структурованих інструкцій-підказок для великих мовних моделей, що утворюються з використанням мета-онтології, для вирішення широкого спектру задач аналізу природномовного тексту, в тому числі отримання відповідей на основі контекстів, відібраних за допомогою відповідних формальних запитів із онтологічної бази знань.

Ключові слова: *онтологія, онтологічна інженерія, графова база знань, автоматизований семантичний аналіз, машинна обробка природної мови, синтез приодномовного тексту.*

ABSTRACT

Litvin A. A. Methods and means of natural language texts analysis and synthesis based on the ontological model of the subject area in the aspect of dialogue systems development. — Manuscript.

A dissertation for the philosophy doctor degree of technical sciences in specialty 122 – computer science – V. M. Hlushkov Institute of Cybernetics of National Academy of Sciences of Ukraine, Kyiv, 2023.

The dissertation work is devoted to the development of methods and means of automated analysis and synthesis of natural language texts based on the ontological model in the aspect of dialogue systems development.

In the work a method was created for an automated generation of queries addressed to ontology which is based on semantic analysis of natural language texts. The peculiarity of the approach is that the formal query is automatically built from automatically customizable templates' blocks according to the semantic categories found in the analyzed text.

Methods of automatic construction of ontologies based on deep syntactic-semantic analysis of natural language text have been developed. Techniueqes were improved for responses generation using flexible structure templates based on a set of concepts obtained through a formal queries addressed to a knowledge base and selected from an input phrase.

An approach based on the application of structured prompts for large language models, which are formed using meta-ontology, was created to solve a wide range of problems of natural language text analysis, including obtaining answers based on contexts selected by means of formal queries from the ontological knowledge base.

Keywords: *ontology, ontological engineering, graph knowledge base, automated semantic analysis, machine processing of natural language, synthesis of monolingual text.*

СПИСОК ОПУБЛІКОВАНИХ РОБІТ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. ***Литвин А.А.**, Величко В. Ю., Каверинский В.В.* Метод получения информации из онтологии на основе анализа фразы на естественном языке. Проблеми програмування. 2020, № 2-3, С. 322 – 330.
2. ***A. A. Litvin**, Kaverinsky V. V, Velychko V. Yu.* Tree-based semantic analysis method for natural language phrase to formal query conversion. Radio Electronics, Computer Science, Control, № 2, 2021, P. 105 – 113.
3. *Palagin O., Malakhov K., Kaverinsky V. V.,* ***Litvin A. A.*** OntoChatGPT Information System: Ontology-Driven Structured Prompts for ChatGPT Meta-Learnin. International Journal of Computing, V. 22, No. 2, 2023, P. 170 – 183.
4. *Palagin O., Malakhov K., KaverinskyV. V.,* ***Litvin A. A.*** Ontology-driven development of dialogue systems. South African Computer Journal, V. 35, No. 1, 2023, P. 37 – 62.
5. *Velychko V. Yu., Kaverinsky V. V.,* ***Litvin A. A.*** A New Approach to Automatic Ontology Generation from the Natural Language Texts with Complex Inflection Structures in the Dialogue Systems Development. CEUR Workshop Proceedings, V. 3501, 2023, P. 172–185.
6. ***Литвин А.А.**, Величко В. Ю., Каверинский В.В.*Разработка архитектуры интеллектуального чат-бота. Information Content and Processing, V. 8, No. 2, 2019, P. 168 – 199.
7. ***A. A. Litvin**, Kaverinsky V. V, Velychko V. Yu.* Synthesis of chat-bot responses in the natural language of the flexive type based on the results of formal questions to ontology and semantic analysis of the initial phrase. International Journal "Information Content and Processing", Bulgaria, V. 7, No. 1, 2020.

Наукові праці, які засвідчують апробацію материалів дисертації:

8. ***A. A. Litvin**, Kaverinsky V. V, Velychko V. Yu.* Agent system for intellectual chatbot. International conference on software engineering “Soft Engine 2020”, Kyiv, NAU, 2020, P. 39 – 42.
9. ***A. A. Litvin**, Kaverinsky V. V, Velychko V. Yu.* Automatic ontology creation from natural language text. International conference on software engineering “Soft Engine 2021”, Kyiv, NAU, 2021, P. 21 – 25.
10. ***A. A. Litvin**, Kaverinsky V. V, Velychko V. Yu.* A new approach to automatic ontology creation from untagged text on natural language of inflective type. International conference on software engineering “Soft Engine 2022”, Kyiv, NAU, 2022, P. 37 – 45.
11. ***Литвин А.А.*** Общие принципы и задачи при создании современных чат-ботов. Збірник наукових праць конференції «Управление знаниями и конкурентная разведка», Харків, 2019.
12. ***Литвин А.А.*** Метод построения запросов к онтологии на основе анализа фразы на флективном языке. Збірник наукових праць конференції «Управление знаниями и конкурентная разведка», Харків, 2020.

ЗМІСТ

ВСТУП

Актуальність теми. Одним з важливих і актуальних напрямків розвитку комп’ютерних наук у наш час є розроблення методів автоматизованої обробки осмислених текстів на природних мовах. У наших дослідженнях увага приділялася процесам семантичного аналізу і синтезу природномовних текстів з орієнтацією на мови флективного типу, до яких належить, зокрема, українська мова. Методологія автоматизованої обробки таких мов взагалі і української зокрема розроблена на наш час недостатньо. Основні причини – граматична складність таких мов і не досить висока поширеність їх використання у світі у порівнянні, наприклад, з англійською мовою.

Безпосередньо пов'язаним з семантичною обробкою природномовних текстів є напрям онтологічної інженерії. У названому контексті актуальним є такі його аспекти, як автоматизована побудова онтологій на базі семантичного аналізу природномовних текстів, побудова формальних запитів до онтології виходячи з природномовних фраз користувача, синтез осмислених природномовних відповідей на базі результатів виконання формальних запитів до онтології.

Зв'язок роботи із науковими програмами, планами, темами. Дисертація пов’язана із тематикою, що виконувалась у відділі 205 Інституту кібернетики НАН України, а саме: проект № 159/01/0245 “Трансдисциплінарна інтелектуальна інформаційно-аналітична система супроводження процесів реабілітації при пандемії (TISP)”, 2020 – 2021 рр., ДКР «Сфера» (Державний контракт № 181/18/3, 2019 р.), проект «Створення методів та технологічних засад формування інтерактивних баз знань» (№ 0117U000005, 2021 р.).

Цілі і задачі дослідження. Метою дослідження є створення методології побудови природномовних інформаційних діалогових і довідкових систем, що працюють на базі онтологічної бази знань, яка

створюється автоматично на базі семантичного аналізу природномовних текстів.

Для досягнення вказаної мети потрібно було вирішити наступні задачі:

1. Розробити методи аналізу природномовного тексту прийнятні для мов флективного типу, зокрема, української, у двох основних аспектах призначення:

- автоматичне створення графових баз знань онтологічного типу на базі природномовного тексту або набору таких текстів;

- аналіз коротких фраз і запитів користувачів для визначення намірів і іменованих сутностей, що їх конкретизують, висловлених у них, необхідних для побудови формальних запитів до бази знань.

2. Створення гнучких шаблонів формальних запитів на мовах SPARQL та Cypher і програмних засобів для роботи з ними, необхідних для отримання релевантної інформації з бази знань.

3. Розробка методів генерації осмислених природномовних відповідей і їх представлення користувачеві із залучанням як підходів, заснованих на правилах, так і з залученням великих мовних моделей.

4. Практичне створення природномовних діалогових і довідкових систем, які втілюють розроблені підходи до аналізу і синтезу тексту із залученням баз знань онтологічного типу.

5. Проведення апробації і тестування розроблених діалогових систем, визначення формальних критеріїв якості їх роботи.

Об’єкт дослідження: автоматизований аналіз і синтез природномовних текстів.

Предмет дослідження: способи автоматизованого семантичного аналізу і синтезу природномовного тексту мовою флективного типу в аспекті онтологічної інженерії і створення довідкових і діалогових систем.

Методи дослідження. Математичне моделювання. Створення і тестування програмних систем. Інженерія інструкцій-підказок для великих

мовних моделей. Визначення критеріїв якості роботи систем за метриками: Accuracy, Recall, Precision та F1. Аналіз літературних джерел.

Наукова новизна отриманих результатів.

1. Отримала подальший розвиток теорія системи семантичних відношень, основною відмінністю якої є виведення значного розмаїття семантичних категорій з аксіоматичного набору понять верхнього рівня. Отримання нових семантичних категорій при цьому відбувається шляхом комбінації двох інших категорій, кожна з яких може бути первинною, або похідною будь-якого рівня. Метод дозволяє створити обґрунтовану OWL-онтологію семантичних категорій необхідної для вирішуваної задачі глибини і повноти відповідно предметної області шляхом виділення підграфів із отриманого зазначеним способом графу виведених семантичних відношень. Отримана таким способом OWL-онтологія може слугувати основою при створенні прикладних графових баз знань.

Раніше, подібної концепції виведення графу семантичних відношень не існувало. Запропоновані попередниками системи семантичних категорій мали суттєві відмінності між собою і не мали механізмів зіставлення і взаємного поєднання. Підходи запропоновані у роботах В. В. Величка, О.В. Палагіна і М. Г Петренка до побудови онтології верхнього рівня передбачали деревоподібну багаторівневу систему семантичних категорій, проте не передбачали концепцію взаємодії категорій різних рівнів для утворення у такий спосіб похідних семантичних відношень.

2. Вперше створено метод отримання формальних запитів мовами SPARQL і Cypher на основі результатів аналізу природномовних текстів, адаптованого до мов флективного типу, зокрема, української. Створення запитів відбувається з використанням гнучких шаблонів. Особливість підходу полягає в тому, що формальний запит автоматично будується із блоків шаблонів (основних і допоміжних), що є такими, що налаштовуються, відповідно визначених семантичних категорій визначених у аналізованому тексті, та сутностей, що їх конкретизують.

Раніше концепції запропонованого методу гнучких шаблонів, що збираються і налаштовуються автоматично у такому вигляді не існувало, а для української мови взагалі процеси автоматизованого створення формальних запитів до баз знань і відповідні практичні реалізації були розвинені вкрай недостатньо. Автоматизоване створення формальних запитів на мові Cypher на основі природномовних висловлювань лишалося мало дослідженим: існували лише окремі роботи, що розглядали досить прості підходи і лише для англійської мови.

3. Отримали подальший розвиток методи автоматичної побудови онтологій на основі природномовного тексту, в тому числі не розміченого, на базі глибокого синтактико-семантичного аналізу. Запропоновані способи є ефективними як для мов флективного типу, зокрема, української, так і для англійської мови.

Створені попередниками, зокрема, за участі В.В. Величка, семантичні аналізатори природномовних текстів українською мовою були здатні сформувати формальне представлення відношень сутностей аналізованого тексту у вигляді синтактико-семантичного графу. Проте розроблені інструменти не виконували представлення і зберігання опису семантичного графу у вигляді OWL-онтології, окрім того мали досить обмежену кількість визначуваних семантичних категорій. Наявні програмні інструменти було вдосконалено за рахунок введення значно більшого набору правил для аналізу фраз українською мовою і виявлення більш ніж 80 категорій семантичних відношень, кожна з яких може мати до більш ніж двох десятків підкатегорій. Також було створено програмну реалізацію для представлення графів розбору текстів у форматі RDF/XML, що є стандартом для семантичної павутини і підтримується рядом відомих СУБД. Окрім того, були розроблені методи і їх програмні реалізації для створення RDF/XML представлень OWL-онтологій на основі наборів текстів із визначеною регулярною структурою. На основі первинного розбору виконується формальне переставлення розбору такого тексту у вигляді визначеної JSON

структури, потім із набору таких структур формується RDF/XML представлення опису контекстної OWL-онтології. Хоча сучасні нейронно-мережеві великі мовні моделі здатні створювати OWL-онтології та RDF/XML представлення текстів, в тому числі і українською мовою, але семантична структура таких онтологій не є визначеною і часто не відповідає вимогам до поставленої прикладної задачі.

4. Отримали подальший розвиток методи для генерації відповідей за допомогою шаблонів гнучкої структури на основі набору понять, отриманих шляхом виконання формального запиту до бази знань і слів, відібраних із вхідної фрази користувача. Представлений метод адаптовано для мов флективного типу, зокрема, української. Шаблон визначає місця підстановки окремих слів та їх груп з вихідного речення та їх словоформи, додаткові текстові вставки з можливістю їх варіювання, місця підстановки результатів запиту та обмеження, що накладаються на них.

Раніше таких підходів до формування відповідей мовами флективного типу не існувало. Шаблони (не слід плутати із шаблонами формальних запитів) мають гнучку структуру і автоматично адаптуються згідно результатів як аналізу вхідного тексту, так і результатів виконання запиту. Великі мовні моделі добре справляються з задачею генерації тексту, але вони мало пристосовані саме до обробки і представлення результатів формальних запитів до бази знань онтологічного типу. Це підтверджено наведеними у роботі експериментальними даними зі зворотного синтезу природномовних речень на основі представлених у OWL-онтології (отриманої із тексту) понять і семантичних відношень між ними.

5. Вперше розроблений підхід, який базується на застосуванні структурованих інструкцій-підказок, що утворюються з використанням мета-онтології. Інструкції передаються великій мовній моделі, для виконання задач аналізу вхідного природномовного тексту (визначення висловлених намірів і іменованих сутностей). Далі з використанням інструкції, сформованої відповідно визначеним намірам і іменованим сутностям,

великій мовній моделі передаються контексти, відібрані із бази знань предметної області за допомогою відповідних формальних запитів для отримання релевантних відповідей.

Раніше такі підходи не були відомі. Хоча застосування інструкцій-підказок для великих мовних моделей зараз набуває широкого використання, їх створення не було онтологокерованим. Також наразі невідомі інші розробки щодо отримання відповідей і висновків за допомогою великих мовних моделей на базі контекстів, отриманих з онтологічних баз знань.

Практична цінність отриманих результатів.

Розроблені методи створення онтології як на базі структурованих, так і неструктурованих природномовних текстів дозволили створити бази знань з різних переметних областей, зокрема: за онтологією листування, реабілітаційна медицина, а також ряд тестових онтологій. Створені таким чином онтології є валідними, відповідають стандартам та можуть бути представлені для роботи з ними з допомогою графових СУБД, таких як Apache Jena Fuseki, Neo4J, RDFlib. Вони можуть слугувати основою для бази знань природномовних діалогових і довідкових систем.

На базі розроблених підходів до семантичного аналізу і синтезу природномовних текстів, а також генерації формальних запитів створено ряд діалогових і довідкових систем, що мають онтологію у якості своєї бази знань, зокрема з питань: фінансів і інвестицій (норвезька мова), змінення клімату і екології (англійська мова), листування (українська мова), педагогічних поглядів Василя Сухомлинського (українська мова), економіки сталого стану (українська мова), «Біла книга з ФРМ» (українська мова), реабілітаційної медицини (англійська мова).

Розроблено комбіновану систему OntoChatGPT, що використовує як великі мовні моделі, зокрема API ChatGPT, у якості одного з інструментів, так і онтологічну базу знань, що слугує для мета-навчання і доповнення такої моделі. Також особливістю системи є наявність мета-онтології, що слугує для створення структурованих інструкцій-підказок для великої мовної

моделі. Застосування розробленого підходу значно розширює можливості сильної мовної моделі, надаючи їй на вхід обрані з контекстної онтології фрагменти спеціалізованої інформації з вузької предметної галузі, поглиблюючи тим самим змістовність і релевантність відповіді, що повертається на виході. Наявність структурованих інструкцій-підказок дозволяє, з одного боку, реалізувати чітке визначення висловлених намірів у вхідному тексті, і понять, що їх конкретизують, з іншого боку, виконати згідно них аналіз інформації представленої із контекстної онтології та синтезувати природномовну відповідь заданої форми і структури.

Розроблено за стосунок для конвертації формату зберігання онтології із Graph Editor до стандарту OWL у вигляді RDF/XML та у зворотному напрямку. Це дозволило практично використовувати створені вручну у програмі Graph Editor (внутрішня розробка) онтології у графових СУБД таких як Jena Fuseki та Neo4J.

Особистий внесок здобувача. Всі основні положення й результати дисертаційної роботи, що виносяться на захист, отримано автором самостійно. У роботах, які опубліковані в співавторстві, здобувачеві належать наступні результати: [97, 104] – розроблення методів аналізу природномовного тексту для побудови формальних запитів до бази знань на мовах SPARQL і Cypher; [109] – використання елементів онтологокерованого підходу зокрема автоматизованого створення інструкцій-підказок для великих мовних моделей при побудові діалогових систем; [110] – загальні принципи побудови природномовних діалогових систем, що працюють з онтологічною базою знань (типи систем, мультиагентна архітектура, принципи взаємодії сервісів); [111] – методи генерації природномовних текстових відповідей на базі результатів виконання запиту до онтології.

Апробація результатів дисертації. Робота виконувалася в Інституті кібернетики ім. В. М. Глушкова НАН України. Основні наукові засади дисертації доповідалися на міжнародних науково-технічних конференціях: «Управління знаннями і конкурентна розвідка» (Харків, 2019), «Information

Content and Processing» (Варна, Болгарія, 2019), «Управління знаннями і конкурентна розвідка» (Харків, 2020), «УкрПрог-2020» (Київ, 2020), «Information Content and Processing» (Варна, Болгарія, 2020), “Soft Engine 2020” (Київ, 2020), “Soft Engine 2021” (Київ, 2021), “Soft Engine 2022” (Київ, 2022), «УкрПрог-2022» (Київ, 2022).

Публікації. Основні наукові результати дисертаційної роботи у повній мірі викладено в 13 публікаціях з яких: 4 статті у фахових журналах України, 5 у виданнях, включених до міжнародних наукометричних баз даних (Scopus або Web of Science).

Структура і об’єм роботи. Дисертація складається із вступу, 5 розділів, загальних висновків, списку літератури зі 111 найменувань. Загальний обсяг роботи складає 255 сторінок, зокрема 16 рисунків, 7 таблиць, 7 додатків.

РОЗДІЛ 1
АНАЛІЗ ПРИНЦИПІВ І ПІДХОДІВ ДО АВТОМАТИЗОВАНОГО СЕМАНТИЧНОГО АНАЛІЗУ І СИНТЕЗУ ПРИРОДНОМОВНИХ ТЕКСТІВ

1.1 Автоматизований семантичний аналіз природномовних текстів – принципи, підходи, задачі

1.1.1 Загальна характеристика проблематики автоматизованого аналізу природномовних текстів

Семантичний аналіз є важливим напрямком у автоматизованій обробці природномовного тексту. Основною його задачею є встановлення контексту словосполучення, речення або більшого фрагменту тексту в форматі зручному для машинної обробки. Так, використана лексика, синтаксис і словоформи передають роль об’єкта (сутності) через взаємозв’язок між лінгвістичними класами [1]. Методи семантичного аналізу можуть визначати наміри, настрої, інші смислові характеристики що подаються у фрагменті тексту та мовні сутності, що їх конкретизують.

В останні часи однією з важливих тенденцій, що посприяли розвитку автоматизованого семантичного аналізу став Інтернет та, зокрема соціальні мережі і месенджери. Сьогодні люди звертаються до соціальних медіа прагнучи ділитися своїми точками зору та висловлювати свої думки. При цьому схильні робити це дуже невимушеним способом, часто припускаючись граматичних, синтаксичних, орфографічних помилок [1, 2]. Поширеною і часто прешочерговою задачею при цьому є встановлення настрою і наміру, висловлених в уривку такого тексту (повідомленні) [3]. Також стає задача встановлення і конкретизації значень слів і словосполучень у даному контексті.

Проте, як зазначається, наприклад, у роботах [1, 4], більшість систем семантичного аналізу розроблено лише для англійської, рідше для інших поширених європейських мов [4].

Тому як природна мова є найпотужнішою формою спілкування, то не дивно, що існує, створюється і вдосконалюється широкий спектр відповідних алгоритмів, що включають такі методи як сортування, кластеризація, видобуток тексту і так далі [5]. Автоматизована обробка природної мови є поширеним напрямком досліджень і прикладних розробок в області інформатики у наш час.

Обробка і аналіз природної мови вважається однією з найскладніших технік у світі штучного інтелекту [17, 18]. В основі методів лежить перетворення вихідних даних природної мови (усної чи письмової) у придатну для подальшої машинної обробки форму. Обробка природномовного тексту є захоплюючим викликом, а для його реалізації потрібна взаємодія комп’ютера та людини [15]. Це область комп'ютерних досліджень, яка займається вивченням і розумінням зв'язку між комп'ютером і людською мовою [19]. Ці інструменти допомагають розробникам створювати прикладні програми. У рамках обробки природної мови встановлено кілька сфер інтересів. Таким чином, найважливіша діяльність основних напрямків зосереджена на наступному: пошук імен і назв; витягування знань із текстів; переклад текстів між мовами; узагальнення письмових робіт; виведення відповідей за допомогою алгоритмів логічного висновку; класифікація та кластеризаціїя документів [20]. Машинна обробка природної мови – це підмножина науки про дані, яка використовує в тому числі динамічні математичні обчислення та статистику, серед яких «наївний Байєс», k-найближчих сусідів, приховані марковські структури, умовні випадкові поля (CRF), дерева рішень, випадкові «ліси» і опорні векторні машини, що зарекомендували себе як чудові інструменти у задачах машинного навчання [21].

1.1.2 Огляд сучасних методологій та підходів до машинної обробки і аналізу природномовних текстів

Комбінація NLP та word2vec для оптимізованої моделі LDA. У 2020 в роботі [6] повідомлялося про створення комбінації natural language

processing (NLP) і word2vec для оптимізованої моделі LDA, яка посилається на теорію вибірки значущості для виділення предметних слів і використання косинусної подібності для оцінки частоти повторів. NLP і text2vec разом придатні для обробки текстів і використовують приховану модель призначення Діріхле для досягнення найкращих результатів. Вибірка значень, яка використовується для навчання моделі підвищує точність і запам'ятовуваність моделі. Косинусна подібність використовується для покращення результатів вимірювань шляхом прискорення операцій.

OWL/XML онтологія у задачах машинного перекладу. У 2017 році Казар Окба та ін. запропонували у роботі [10] методи конвертації з використанням різних джерел знань, сподіваючись підвищити якість перекладу. Унікальність запропонованих підходів полягає в тому, що вихідна мова може бути використана для розуміння контексту мови приймача. Порівняння результатів запропонованого методу із точними даними виявило, що система дає обнадійливі результати. Вони застосували прогресивний і системний підхід до машинного природномовного перекладу. Щоб усунути неоднозначність, вони розробили парадигму перекладу, засновану на семантичному аналізі. Для чого використана OWL/XML онтологія, що містить граматичні, термінологічні та анотаційні деталі. Таким чином, вони створили алгоритми придатні для перекладу невідредагованого тексту за допомогою власних інструментів.

Застосування графової бази даних для визначення подібності між текстами. У 2020 році А. Максутов та ін. у своїй роботі [18] запропонували підхід, при якому уточнюються нестандартні деталі, а потім ці знання використовуються для створення більш повного опису. У статті досліджується, як графова база даних може допомогти у визначенні подібності між текстами. Створений таким чином конспект допомагає визначити зв'язки між виразами, словами та реченнями. Проте якість роботи алгоритму залежать від правильного введення вихідної інформації. Чистий вихідний текст має вирішальне значення. Граф показує як помилки, так і

втрату інформації. У центрі системи лежить алгоритм аналізу залежностей. Конкретний аналізатор визначається залежно від складності речення на вході. Техніка аналізу була використана для пояснення подібності між сутностями, знайденими в тексті.

Ансамблева система для класифікації текстів. У роботі [26] автори запропонували систему, яка використовує методи розпізнавання тексту та інші технології для очищення наборів даних. Для формування прогнозної моделі застосовувалися методи ансамблю. Після аналізу отримані вектори ознак класифікуються відповідно до моделі. Програма виконує аналіз зібраних даних, обробку даних, а також шаблонів планування, класифікації та прогнозування. Центральна теорія запропонованого підходу полягає в тому, щоб підвищити точність класифікації за допомогою додаткових навчальних даних. Комп'ютер, про який йде мова, налаштований на збір даних із «Твіттера» та аналіз знайдених повідомлень. Текст перевіряється на емоційність за допомогою оригінальної системи класифікації. Аналізуються різні методи оцінки, щоб визначити, чи є репліка точною, шкідливою чи нейтральною. Ансамблева система класифікує набір даних краще, ніж традиційні методи. Серед цих ансамблевих підходів найуспішнішими виявилися випадкові дерева.

Rich Semantic Graph to Text. У статті [27] пояснюються методи вирішення з модулю «Rich Semantic Graph to Text». Вони ілюструють вжиті кроки та механізм, використані для створення цього модуля. Використання математичного граматичного аналізу є піонерським підходом до резюмування абстрактного тексту арабською мовою, а основною сферою дослідження є оцінка різних систем. Дослідники розробили модуль зменшення RSG, а також розробляють робочий прототип для своєї схеми.

Застосування онтології для автоматизованого тлумачення неоднозначних текстів. У роботі [28] автори розробили підхід до вирішення проблеми відсутності повної семантичної інформації для оптимального семантичного розуміння. Підхід полягає у виявленні значущості програмних

представлень вимог, а потім у визначенні ефекту семантичного аналізу. Вони використовували семантичну структуру (онтологію), спеціально створену для тлумачення та усунення неоднозначності текстів. Архітектура системи базується на семантичній технології, яка може бути включена в програмну документацію та реалізацію. Представлена методологія показує шляхи адаптації існуючих необхідних онтологій переметної області та використання їх для побудови і виконання експериментів і методів, які допомагають в управлінні знаннями у програмній системі.

Застосування векторної відстані між словами та tf-idf для дедуплікації веб-сторінок. У роботі [30] автори представили методологію, яка дозволила провести ретельний аналіз технологій вилучення семантичної інформації. Вони запропонували новий вид алгоритму дедуплікації веб-сторінок, який використовує векторну відстань між словами та підхід tf-idf. У даній роботі запропоновано модель побудови семантичної хмари. Запропоновано новий алгоритм де-дупликації веб-сторінок, який значно ефективніший, ніж стандартні рекомендації щодо сторінок на основі TF-IDF і відстані вектора слова. Показано, що запропонований метод відфільтровує веб-сторінки, що не повторюються, на основі схожості веб-сторінок у корпусі з Hbase.

Семантичний підхід до відповіді на запит за допомогою DBpedia та WordNet. У роботі [32] автори застосували семантичний підхід до відповіді на запит за допомогою DBpedia та WordNet. Дослідження мало на меті визначити найкращі методи вирішення питань. Документ містив рішення щодо того, як класифікуються іменовані сутності, як прирівнюються іменовані сутності та як можна звертатися до іменованих сутностей. Дослідження перевіряло точність, з якою багато хто відповідав на запитання. Використовуючи колекцію питань TREC, DBpedia та запропоноване рішення, метод отримав загальну оцінку F-міри 93,43%, середнє запам’ятовування 94,15% і середню точність 92,73%.

Підхід до семантичного упорядкування юридичних метаданих. У 2018 році Амін Слеймі представив у своїй роботі [33] підхід до семантичного упорядкування юридичних метаданих, щоб читач міг краще зрозуміти та інтерпретувати юридичні положення. Метадані важливі для ідентифікації єдиних правових вимог. Не вистачає літератури про те, як визначити узгодженість метаданих для аналізу формальних специфікацій. Крім того, наша здатність автоматизувати пошук правових семантичних метаданих не використовується оптимально; ми не використовуємо всі переваги обробки природної мови, які вони застосовують для перевірки кожного із запропонованих типів семантичних метаданих і вирішення будь-яких суперечок, що виникають. В першу чергу, виконується якісний аналіз, щоб класифікувати типи метаданих, які найкраще фіксуються інструментами відстеження. Модель рекомендує відповідну дослідницьку модель для юридичних вимог і пропонує точні правила вилучення застосовних метаданих. Дослідники проаналізували методи вилучення для різних окремих випадків. Аналіз показує, що метадані належні. Автори оцінюють узгодженість між 87,4 і 97,2 відсотками, а показник Recall – від 85,5 до 94,9 відсотків.

Міжмовна модель схожості слів. У роботі [34] було запропоновано підхід, що ґрунтується на новому методі вбудовування слів для побудови моделі схожості слів серед 76 різних мов світу. Дослідники використовуватимуть атлас, щоб допомогти їм звузити цільовий клас слів, які їх цікавлять. Порівняльний аналіз показує, що просторова подібність між двома мовами є подібністю у значеннях слів. Було проведено дослідження міжмовного порівняльного аналізу 76 різних мов. Результати показують, що англійська мова дещо схожа на інші мови з точки зору семантичного значення, але англійська мова не є тісно пов’язаною з іншими мовами.

Застосування семантичної мережі для виявлення неоднозначності у вимогах. У роботі [35] було використано семантичну мережу та методи обробки природної мови для виявлення неоднозначності у вимогах. Тому

глядач, швидше за все, матиме більше варіантів і зможе оцінити неоднозначні фрази. Вони створили рішення, щоб пояснити потенційно нечіткі критерії та підвищити обізнаність про те, як має працювати система. Розроблений POS-тегер може усунути неоднозначність семантичних понять у межах вимог користувача та те, як ці поняття пов’язані з іншими.

Семантичне сортування для ідентифікації документів. У роботі [37] було запропоновано новий алгоритм вирішення питань. POS-маркування відповідей користувачів і семантичне сортування використовувалися для ідентифікації документів. Запропонований підхід є логічною структурою, яка використовується для сортування речень на основі логічних принципів. Інформація збирається шляхом порівняння важливих характеристик у спробі знайти правильну відповідь. У порівнянні з традиційними синтаксичними підходами, запропонований підхід покращує вивчення синтаксису за допомогою лямбда-числення. Запропонована система при тестуванні досягла середньої точності 83%, перевершуючи поточний підхід приблизно на 11%, і досягаючи 95% точності для питань «Так/Ні». Цей метод простий, елегантний і більш природний, ніж системи, які передбачають використання обчислень. Крім того, підхід до аналізу забезпечує вищий ступінь точності відповідей на такі питання.

Задачі автоматизованого розуміння природної мови при створенні чат-ботів. У 2019 році Л. Гунасекара та ін. [38] представили підходи з використанням семантичних технологій, таких як NLG, і дослідили переваги та недоліки таких систем. Дослідження базується на використанні семантичної генерації природної мови за допомогою чат-ботів з меншими обчислювальними витратами та на потенціалі повторного використання її для подібних доменів із меншим кодуванням у частині генерації природної мови для доменів малого рівня. Дослідники створюють нову архітектуру для підтримки систем чат-ботів і вказують на цінність розуміння природної мови. Для побудови синтетичних мов можна створити онтологічну структуру. Платформа, яку вони надають, діє як фреймворк на основі API, який

пропонує інтерфейс для текстових відповідей. Програмний API, який можна легко інтегрувати в будь-яку систему, може стати основою, яка буде використовуватися в багатьох інших областях.

У роботі [40] розроблено техніку для уточнення алгоритму розуміння семантики природної мови та огляду, що підходить для попередньої обробки технологічних відповідей на роботизовані запитання, розбіжність термінів у смислі, семантичне дослідження репутації тощо, порівняльні оцінки та оцінка продуктивності. Вони попередньо обробили текст за допомогою потужних алгоритмів, які ілюструють те, як слова співвідносяться одне з одним у тексті. Однак пристрій має обмеження щодо точності та гнучкості [1].

Автори роботи [42] запропонували наступну концепцію. NLP використовує дані, які надходять із різних типів медіа. Обробка мови використовується для класифікації послідовності відповідей на поставлене запитання. Логічна система демонструє, як ефективно поставити запитання. Для проведення аналізу було використано SQUAD. Точність результатів тестування показала, що точність пошуку уривків за допомогою TFIDF склала 69,69%, тоді як люди (студенти) в середньому показували результат 69,93%.

1.1.3 Аналіз настроїв та емоцій у природномовному тексті

Аналіз настроїв – це за своє сутністю тип класифікації тексту, де визначається суб’єктивність висловлювань. В англомовній термінології існує поняття opinion mining – витягнення оцінки думок із великої кількості текстових джерел [7, 8]. Аналіз настроїв описується як виявлення настроїв людей щодо певної теми та її особливостей. Подібні проекти з пошуку думок популярні, оскільки люди хочуть використовувати поради інших у своїй діяльності [9]. Суб’єктивні дослідження великих даних є важливою проблемою в інтелектуальному аналізі даних і задачах обробки природномовного тексту [10].

У рамках напрямку машинної обробки природномовного тексту вивчається, які стратегії і як використовувати для визначення міркувань і емоцій автора тексту і вирішувати, що він виражає через них [11]. Культура може по-різному впливати на цю сферу, що може приводити до невірного тлумачення, якщо сприймати текст надто буквально. Так, у роботі [12] наводиться приклад фрази с коментарів у Інтернет ресурсі: «This new gadget is bad!» («Цей новий пристрій поганий!»). На перший погляд очевидно, що фраза натякає на неприязнь користувача до пристрою, проте для певної вікової групи спільноти дана фраза могла б мати на увазі і схвальну характеристику даного пристрою [12]. Таким чином для аналізу настроїв визначними є такі обставини як час місце і особа, що висловлюєте свою думку [13].

Для перевірки гіпотези дослідника використовуються зібрані дані. З появою Інтернету багато сфер досліджень вирішили збирати дані з нього [15]. Такі компанії, як Google, YouTube і Amazon, досліджують, як налаштувати контент відповідно до інтересів клієнта [16]. Очевидними є такі показники як «вподобайки» в соціальних мережах, кількість споживачів і продажі. Проте одних цих даних недостатньо. Тому підключається також автоматизований аналіз текстового матеріалу, що створюється користувачами. Але ця концепція ставить свої проблеми і складнощі, серед яких: використання різних мов в одній темі чи блозі; використання нестандартних слів, в тому числі таких, що неможливо знайти в словнику; використання нетекстових символів [17].

У роботі [29] описано можливі застосування аналізу настроїв. Зазначається, що у даній галузі найактуальнішою проблемою є відсутність повністю гнучких і відтворюваних баз даних і методів вимірювання. Вони припустили, що можна було б покращити словниковий запас таких систем, щоб мати можливість вихопити більше з того, що говорять і пишуть люди. Індустрія видобутку знань покриває багато областей і сфер. Аналіз настроїв – це широко використовуваний лінгвістичний інструмент, який

використовують рекламодавці та компанії соціальних медіа. Для вирішення проблеми автори використовували машинне навчання. Вони погодилися, що NLP має кілька питань без відповіді, які вирішити їм не під силу. Зазначені додаткові нюанси ставлять перед розробниками методів машинного розуміння природної мови завдання, такі як надфразова єдність та усунення неоднозначності слів. Зазначається, що аналіз почуттів потребує аналізу лише слів, які використовуються для передачі почуття. Комплексний мережевий аналіз може дати кращі результати при роботі з довільним текстом. Системи, засновані на знаннях, успішно забезпечують кращі результати на унікальних завданнях. Пропонуються й інші рішення машинного навчання, але всі вони не без недоліків.

1.1.4 Автоматизований аналіз тексту з метою створення формальних представлень

У 2017 році Прашант Гупта та ін. [31] створили інструмент для генерації формальних запитів Intelligent Querying Framework (QS), що дозволяє користувачеві створювати власні запитання, що перетворюються у формальні SQL-запити. У системі є модуль, який перетворює речення англійською мовою на твердження, які використовуватиме SQL-запит, створюючи запити для отримання відповідей згідно потребам користувача. Як наслідок, це зменшує обсяг та полегшує проведення досліджень. Запропонована архітектура дозволяє неекспертам робити запити до бази даних і полегшує користування нею. QS дуже ефективний навіть у створенні складних запитів. Таким чином він надає простий спосіб отримати доступ до інформації з бази даних.

Автори роботи [36] розробили метод перекладу запитів і інструкцій природною мовою в комп’ютерний код. Щоб подолати виклик, дослідники комп’ютерних наук використовують технології семантичної павутини для розробки CodeOntology, спільної платформи, щоб зробити код з «першокласним громадянином Інтернету», де він може бути взаємопов’язаний з іншими ресурсами. Цей підхід використовує онтологію

коду для отримання різноманітних методів і зразків коду. Вони оцінюються та комбінуються для перетворення природномовної специфікації у вихідний код на мові програмування Java. Результати показують, що рішення має бути порівнянним за ефективністю з пропрієтарними, такими як Wolfram Alpha.

У 2020 році Ш. Пілер та ін. [39] розширили свою попередню роботу над інтерфейсами запитів на природній мові для онлайн-даних, додавши перехідні дієслова та прийменникові речення, а також підхід для розміщення запитів, які вимагають ланцюжкових складних дієслів. У реляційній базі даних ця властивість може визначати n-арні відношення, у яких властивість є транзитивною.

1.2 Онтологічна інженерія, як прогресивний метод до систематизації знань у предметній області

1.2.1 Поняття і роль онтології у інформатиці

Онтологія [22, 23] – це первинно філософський термін, що означає філософія буття, розділ, який займається тим, «що є, тобто природою реальності, і структурними аспектами існування як такого». Інтернет-словник методів соціальних наук SAGE (2006) описує онтологію як «концепцію, пов’язану з існуванням і взаємозв’язками між різними аспектами суспільства, такими як соціальні актанти, культурні норми та соціальні структури». Онтологія – це дослідження реальності та наших переконань щодо речей [24]. Онтологія є сутністю всесвіту та існування істини, але вона також визначає те, що можна про неї сказати. Брайман (2008) визначає соціальну онтологію як філософське значення, яке приписується соціальному світу. Припускають, що люди думають про те, чи існує істина, яка існує окремо від людських уявлень та інтерпретацій, і чи існує повсякденна реальність чи кілька контекстно-специфічних [10, 25].

У інформатиці під поняттям «онтологія» розуміється форма структурованого представлення знань певної предметної області. Онтологія –

це формалізована модель знань, яка описує концепції, відношення і атрибути в предметній області [46]. Онтології можуть бути побудовані з використанням різних мов та стандартів, таких як OWL (Web Ontology Language) і RDF (Resource Description Framework). Графові бази даних використовуються для збереження та оптимізованого оброблення онтологічних даних [47].

Зазвичай онтологія являє собою графову базу знань поняття у якій поєднані визначеними предикатами. Найбільш поширеною формою опису онтології є формат OWL (Web Ontology Language), що передбачає наявність наступних основних сутностей: класи, властивості і іменовані сутності (екземпляри). Класи мають ієрархічну деревоподібну структуру типу клас – підклас. Кожен клас може мати одного або більше батьківських класів. Наявність дочірніх класів є опійною, а їх кількість теоретично не обмежена. Кореневим для всіх класів є клас “Thing” («Річ»). Іменовані сутності (“Named Individual”) є фактично екземплярами класів і означають конкретні речі і поняття даної предметної області, на відміну від класів, що описують біль абстрактні, узагальнені поняття. Іменовані сутності можуть поєднуватися між собою властивостями, тобто: об’єкт А може мати властивість В зі значенням С, де А і С – іменовані сутності. Наприклад об’єкт «Роман “Діти капітана Гранта”» має властивість «Авторство» зі значенням «Жюль Верн». Наявність значення властивості є опійною, тому як іноді сама властивість вже є достатньою характеристикою, наприклад: об’єкт «ворона» має властивість «вміє літати». Властивості також мають характеристики Domain та Range. Їх значеннями є класи. Domain вказує сукупність класів, екземпляри (іменовані сутності) яких можуть мати дану властивість. Range визначає сукупність класів, екземпляри яких можуть бути значенням для даної властивості. Властивості можуть мати додаткові обмеження, наприклад, якщо властивість є функціональною, то один конкретний об’єкт може мати лише одне значення даної властивості. Класи у онтології окрім ієрархічної структури можуть поєднуватися у об’єднання і перетинання, що

виступають як окремі сутності у онтології. Також горизонтальні зв’язки між класами можуть бути наведені через Domain/Range певних властивостей. Можливо явно задати тотожність або відмінність певних сутностей у онтології. Даний короткий опис не вичерпує всі особливості структури онтології, а лише характеризує основне уявлення про такі структури даних і стандарту OWL. Більш детальний опис можна знайти за посиланням [43].

Відмінною особливістю саме онтології, що відрізняє її від просто однієї із форм представлення графової бази даних є те, що з онтологічної структури за допомогою дескриптивних логік можуть робитися висновки і бути отримані нові знання не представлені у ній явно. Це можна проілюструвати простим прикладом. Так, уявімо, у онтології є сутності «Кішка Мурка», «Кішка Ліза» і «Кіт Васька», а також властивість «Мати матір». Властивість є функціональною – кіт може мати тільки одну матір. Але для «Кота Віськи» ця властивість задана із двома значеннями: «Кішка Мурка» і «Кішка Ліза». Таким чином, можна зробити висновок, що сутності «Кішка Мурка» і «Кішка Ліза» тут є тотожними. Тобто це просто одна й та сама кішка, відома під різними іменами. Це досить примітивний, але зрозумілий приклад. У великих онтологіях, якщо вони правильно побудовані і структуровані, дескриптивні логіки здатні виявити досить змістовні висновки. Проте ніщо не заважає використовувати онтологію і просто у якості графової бази даних, але в такому випадку слід скоріш говорити про графову структуру даних онтологічного типу, побудовану за стандартом OWL і представлену у вигляді RDF/XML. Слід зазначити, що у наявній роботі онтології розглядаються скоріш у такому контексті – форма графової бази знань.

Мовою формальних запитів до онтології є SPARQL. Основою такого запиту є перелік умов записаний у формі RDF-трійок (суб’єкт – предикат – об’єкт), яким повинні відповідати обрані результати. Більш змістовний опис мови SPARQL її синтаксису і можливостей можна знати в офіційній документації за посиланням [44].

1.2.2 Онтологічна інженерія

1.2.2.1 Визначення і основні задачі онтологічної інженерії. Задачі створення онтології, коректної і зрозумілої її структуризації, що забезпечують найбільш ефективне її практичне використання формують окрему область досліджень – онтологічна інженерія. Поєднання онтологічних підходів і методів машинного навчання відкриває нову галузь, що називають у англомовній літературі “ontology learning”.

Онтологічна інженерія є сучасним підходом до систематизації знань в різних предметних областях, особливо в інформатиці, кібернетиці та інженерії знань. Цей підхід базується на створенні та використанні онтологій – формалізованих моделей знань, які допомагають організувати інформацію і розуміти взаємозв'язки між різними концепціями в предметній області. Онтологічна інженерія дозволяє автоматизовано створювати онтології та використовувати їх для покращення пошуку, аналізу та інтеграції знань [45].

Онтологічна інженерія має широкі застосування в інформатиці та кібернетиці. Онтології допомагають підвищити ефективність пошуку та інтеграції даних в різних інформаційних системах. Вони дозволяють створювати семантично багаті моделі, що полегшують автоматизовану обробку та розуміння даних [49]. Крім того, онтологічна інженерія допомагає створювати експертні системи та інші інтелектуальні додатки, які базуються на формалізованих знаннях [50].

Онтологічна інженерія продовжує активно розвиватися. З ростом обсягу доступної інформації та збільшенням складності предметних областей, важливість систематизації та розуміння знань стає ще більшою. Машинне навчання та інші технології сприяють автоматизованому створенню онтологій та покращенню інтеграції даних [51]. Онтологічна інженерія обіцяє стати ключовим інструментом у розвитку інтелектуальних систем і розумного аналізу даних.

1.2.2.2 Автоматизоване створення онтологій. Одним з важливих аспектів онтологічної інженерії є автоматизоване створення онтологій. Цей

процес включає в себе застосування методів і інструментів, які допомагають визначити концепції, відношення та атрибути в предметній області. Одним із підходів є ontology learning, який використовує машинне навчання та аналіз текстової інформації для видобуття знань та створення онтологій [48]. Такий підхід спрощує і прискорює процес створення онтологій, особливо в великих і складних предметних областях.

Онтологічна інженерія є прогресивним методом до систематизації знань у предметних областях, особливо в інформатиці, кібернетиці та інженерії знань. Використання онтологій та графових баз даних сприяє покращенню пошуку та аналізу даних, а також розвитку інтелектуальних систем. Автоматизоване створення онтологій, включаючи методи ontology learning, робить процес створення онтологій більш доступним та ефективним. Загалом, онтологічна інженерія відкриває нові можливості для організації та використання знань у різних предметних областях. Завдяки автоматизації та розвитку нових технологій, цей підхід стає ще більш актуальним і обіцяє значний внесок у розвиток інформаційних технологій та інтелектуальних систем.

Автоматизоване створення онтології та графових баз даних на основі аналізу природномовних текстів є актуальною темою в галузі інформатики, кібернетики та інженерії знань. Цей підхід використовується для створення формалізованих моделей знань та баз даних, які базуються на інформації, видобутій з текстових джерел. Розглянемо ключові аспекти автоматизованого створення онтології та графових баз даних з використанням природномовного аналізу текстів.

Створення онтології зазвичай вимагає великої кількості ручної роботи та експертного знання в предметній області. Проте завдяки розвитку технологій NLP та машинного навчання, створення онтологій може бути автоматизовано. Одним з підходів є ontology learning, який використовує аналіз текстів для видобування знань та автоматичного побудови онтології [48].

Цей процес включає в себе кроки, такі як:

Видобування текстової інформації: Аналіз текстових джерел для ідентифікації концепцій, відношень і атрибутів.

Очищення та обробка даних: Перетворення текстової інформації у структуровані дані.

Побудова онтології: Створення формалізованої моделі знань на основі видобутої інформації.

Для автоматизованого створення онтології використовуються інструменти та бібліотеки, такі як Stanford NLP, OpenNLP, та інші, які надають можливість аналізу природномовних текстів та видобування знань.

Після створення онтології, графові бази даних можуть бути використані для збереження та оптимізованого оброблення даних, представлених у вигляді графа. Графові бази даних дозволяють зберігати зв'язки між концепціями та об'єктами, що дозволяє легко виконувати запити та аналізувати дані [52].

1.2.2.3 Онтологія і графові бази даних. Застосовуючи графові бази даних разом із створеною онтологією, можна досягти декілька важливих цілей:

Пошук і аналіз інформації: Графові бази даних дозволяють виконувати складні запити, включаючи пошук шляхів між концепціями та аналіз графової структури для знаходження важливих зв'язків.

Інтеграція даних: Графові бази даних допомагають об'єднати дані з різних джерел, що має важливе значення для обробки даних в глобальних системах.

Візуалізація даних: Графові бази даних надають можливість візуалізувати графову структуру, що полегшує розуміння та аналіз даних.

Автоматизоване створення онтології та графових баз даних на основі аналізу природномовних текстів знаходить широке застосування в різних галузях. Наприклад, в медицині такий підхід може використовуватися для створення онтологій, які допомагають враховувати різні медичні терміни та

їх взаємозв'язки. В області біоінформатики автоматизована побудова онтологій допомагає інтегрувати дані про гени, білки та реакції в одну загальну модель. Також цей підхід застосовується у сфері електронної комерції для аналізу поведінки покупців та рекомендаційних систем. У фінансовому секторі графові бази даних використовуються для виявлення фінансових шахраїв та аналізу ризиків.

Автоматизоване створення онтології та графових баз даних на основі аналізу природномовних текстів є важливим напрямком досліджень в інформатиці, кібернетиці та інженерії знань. Використання технологій NLP та графових баз даних дозволяє автоматизувати процес створення та обробки знань, що має значення для багатьох галузей. Результати досліджень у цьому напрямку вже застосовуються у різних сферах, і цей підхід має потенціал для подальшого розвитку та вдосконалення.

1.3 Методи синтезу осмислених природномовних текстів

1.3.1 Задачі і застосування синтезу природномовних текстів

Синтез природномовних текстів (Natural Language Generation, NLG) є важливою галуззю штучного інтелекту та обробки природної мови (NLP), яка вивчає створення текстуального вмісту, який є легко зрозумілим та зв'язаним для людей. Ця технологія має різноманітні застосування від автоматичного створення новинних статей та звітів до освітніх систем та чат-ботів. Розглянемо розвиток методів синтезу осмислених природномовних текстів, а також звернемо увагу на сучасні досягнення, зокрема сильні мовні моделі, які значно поліпшили якість NLG.

Синтез природномовних текстів має довгу історію розвитку, починаючи з ранніх систем, які створювали текст на основі генерації шаблонів та правил. Однак такі системи були обмежені і надавали не завжди задовільні результати. Спроби використання генерації текстів на основі знань і онтологій також виявилися складними та трудомісткими завданнями. Проте

із зростанням обчислювальних потужностей та розвитком NLP прийшли нові можливості для покращення синтезу текстів.

1.3.2 Методи синтезу природномовних текстів

Сучасні методи синтезу NLG можна розділити на декілька категорій:

Системи засновані на наборі правил. Ці системи використовують набір правил та шаблонів для створення тексту. Вони можуть бути ефективними для створення обмежених типів тексту, але вимагають багато ручної роботи на етапі розробки і налаштування [53].

Генерація на основі штучних нейронних мереж (NNG): З введенням машинного навчання та глибоких нейронних мереж, NLG став доступнішим та ефективнішим завдяки моделям, таким як ChatGPT. Ці моделі використовуються для генерації тексту на основі великого обсягу текстових даних [54].

Семантичний підхід: Синтез тексту може базуватися на семантичних моделях, де текст генерується на основі розуміння семантики вхідних даних. Це дозволяє створювати більш осмислені та контекстуально відповідні тексти [55].

1.3.3 Застосування великих мовних моделей для задач синтезу природномовного тексту

Однією з найбільших досягнень в галузі синтезу текстів є розвиток сильних мовних моделей, таких як GPT (Generative Pre-trained Transformer). Моделі, які навчаються на великому обсязі текстових даних, виявилися дуже успішними у створенні осмислених текстів. Ці моделі можуть генерувати текст, який надзвичайно схожий на текст, написаний людиною, і вони широко використовуються в NLG завданнях [56], в тому числі будучи інтегрованими в інші програмні системи у якості модулю або сервісу. Однак такі моделі мають виклики, пов'язані з генерацією біасованого чи необґрунтованого контенту, які потребують додаткових досліджень та вдосконалень.

1.3.4 Сфери застосування синтезу природномовного тексту

Синтез природномовних текстів знайшов застосування в багатьох галузях, включаючи:

Генерація новинних статей: Системи NLG використовуються для автоматичного створення новин та статей на основі структурованих даних.

Освітні системи: NLG може використовуватися для створення освітніх матеріалів та навчальних програм.

Чат-боти та особисті асистенти: NLG використовується для генерації відповідей та діалогів в чат-ботах та особистих асистентах, що полегшує спілкування з користувачами.

Автоматичні звіти та аналітика: NLG може бути використаний для автоматичної генерації звітів та аналітичних матеріалів на основі даних.

Помічники для осіб з обмеженими можливостями: Синтез тексту може бути корисним для створення текстових варіантів для осіб з обмеженими можливостями, які мають проблеми з мовленням.

1.3.5 Прогрес у розвитку моделей синтезу природномовного тексту

1.3.5.1 Моделі типу «послідовність до послідовності». Нейронні моделі типу «послідовність до послідовності» забезпечили новий життєздатний підхід для абстрактного резюмування тексту (це означає, що вони не обмежуються простим вибором і переставлянням уривків з оригінального тексту). Однак у цих моделей є два недоліки: вони схильні неточно відтворювати фактичні деталі та мають тенденцію повторюватися. У роботі [58] запропоновано нову архітектуру, яка доповнює стандартну модель «послідовність до послідовності» двома ортогональними способами. По-перше, використано гібридну мережу генератора вказівників, яка може копіювати слова з вихідного тексту за допомогою вказівки, що сприяє точному відтворенню інформації, зберігаючи при цьому можливість створювати нові слова через генератор. По-друге, використано охоплення, щоб відстежувати те, що було підсумовано, що запобігає повторенню. Розроблена модель була застосована для зведення CNN / Daily Mail.

1.3.5.2 Моделі трансдукції домінантної послідовності. Моделі трансдукції домінантної послідовності базуються на складних рекурентних або згорткових нейронних мережах у конфігурації кодера та декодера. Найефективніші такі моделі також з’єднують кодер і декодер за допомогою механізму контролю уваги. У роботі [59] запропоновано нову, просту мережеву архітектуру, яка базується виключно на механізмі уваги, повністю позбавляючись від повторень і згорток. Експерименти з двома завданнями машинного перекладу показують, що ці моделі не тільки є кращими за якістю, але є більш розпаралелюваними та потребують значно менше часу для навчання. Модель із 165 мільйонами параметрів забезпечує 27,5 BLEU при перекладі з англійської на німецьку, покращуючи існуючий найкращий результат ансамблю більш ніж на 1 BLEU. Що стосується перекладу з англійської на французьку, було перевершено попередню найсучаснішу модель на 0,7 BLEU, досягаючи оцінки BLEU 41,1.

1.3.5.3 Моделі типу "Transformer". Модель "Transformer" названа так через свою архітектуру, яка повністю замінила рекурентні нейронні мережі (RNN) у задачах обробки послідовностей даних, таких як обробка природної мови (NLP). Ця архітектура була представлена у статті "Attention is All You Need" [59] у 2017 році і швидко стала однією з найвпливовіших у галузі NLP та машинного навчання загалом. Назва "Transformer" пов'язана з тим, що ключовим елементом цієї архітектури є механізм уваги (attention mechanism). Цей механізм дозволяє моделі дивитися на різні частини вхідних даних з різним ступенем уваги та зважувати їхній вплив при обробці інформації. Це дозволяє моделі ефективно працювати з послідовностями різної довжини та легко узагальнювати інформацію з тексту. Transformer-архітектура стала основою для багатьох сучасних моделей в NLP, включаючи GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers) та інших. Ці моделі досягли значних успіхів у завданнях обробки тексту та навчанні з підкріпленням, і вони вважаються важливими досягненнями у галузі штучного інтелекту.

1.3.6 Актуальні проблеми в області синтезу природномовного тексту

Останні нейронні моделі показали значний прогрес у проблемі створення коротких описових текстів на основі невеликої кількості записів бази даних. Експерименти показують, що ці моделі створюють плавний і зрозумілий текст, але не можуть переконливо наблизити його до створених людиною документів. Більше того, навіть шаблонні базові лінії перевищують продуктивність цих нейронних моделей за деякими показниками, хоча розширення на основі копіювання та реконструкції призводять до помітних покращень [57].

Попри названі виклики і досягнення, синтез природномовних текстів залишається активним напрямком досліджень та розвитку, і разом зі зростанням потужностей мовних моделей, можливості та застосування NLG продовжують розширюватися.

Незважаючи на досягнення в галузі синтезу природномовних текстів, існують деякі виклики та питання, які залишаються відкритими. Зокрема, це включає в себе питання контролю над змістом, відповідальності та етики використання таких систем. Додатковою проблемою є біаси, які можуть з'являтися в сгенерованому контенті.

Синтез природномовних текстів є важливою галуззю в штучному інтелекті та обробці природної мови. З історії використання правил та шаблонів до сучасних сильних мовних моделей, NLG продовжує розвиватися та знаходити нові застосування в різних сферах, включаючи новини, освіту, чат-ботів та інші. Однак існують виклики, пов'язані з контролем, біасами та етикою використання NLG, які потребують уваги та досліджень.

1.4 Практичне застосування аналізу і синтезу природномовних текстів при створенні довідкових і діалогових програмних систем

Синтез та аналіз природномовних текстів (NLP) є важливою складовою в галузі штучного інтелекту, яка має значний вплив на різні аспекти сучасного життя. Системи, які здатні розуміти та створювати текст, знаходять застосування у різних галузях, включаючи довідкові та діалогові програмні системи. Розглянемо практичне застосування аналізу та синтезу природномовних текстів у створенні довідкових та діалогових програмних систем.

1.4.1 Роль довідкових і діалогових програмних систем у сферах людської діяльності

Історія використання природномовного аналізу та синтезу в програмних системах сягає своїм корінням далеко назад. Починаючи з ранніх систем обробки тексту до перших спроб створення діалогових систем, історія розвитку NLP була пов'язана з багатьма викликами та досягненнями. Наприклад, системи розпізнавання мови та синтезу мови були розроблені для взаємодії з комп'ютерами та контролю над ними. Однак ці системи були обмеженими в обсязі та якості результатів.

Сучасні довідкові системи використовують аналіз природномовних текстів для розуміння запитів користувачів та надання відповідей. Однією з ключових складових таких систем є відповідність запиту користувача до вмісту системи. Для цього використовуються методи семантичного аналізу, розпізнавання іменованих сутностей, аналізу тональності та інші [60]. Наприклад, сучасні довідкові чат-боти можуть ефективно взаємодіяти з користувачами та надавати інформацію на основі аналізу природномовних текстів.

Діалогові системи, такі як голосові асистенти та чат-боти, використовують синтез природномовних текстів для генерації відповідей на запити користувачів. Сучасні моделі генерації тексту, включаючи сильні

мовні моделі, дозволяють створювати більш природні та змістовні відповіді [56]. Наприклад, ChatGPT може генерувати текст, який легко співзвучить з текстом, написаним людиною, і надає користувачам інтуїтивно зрозумілу взаємодію.

Системи на основі NLP знаходять застосування в освіті, де вони можуть надавати студентам підтримку та відповіді на запитання. Наприклад, платформи для електронного навчання використовують діалогові системи для надання інструкцій та пояснень, що допомагають студентам зрозуміти складні матеріали [61].

Незважаючи на значні досягнення в галузі NLP, існують виклики та перспективи, які потребують уваги та досліджень. Один з головних викликів полягає в управлінні біасами в аналізі та синтезі природномовних текстів, що може призвести до необ'єктивних результатів та негативного впливу на користувачів. Також, важливим аспектом є забезпечення етичного використання NLP систем та діалогових асистентів, а також захист приватності даних користувачів [62].

Ще однією перспективою є постійний розвиток та удосконалення сильних мовних моделей, таких як GPT-3 та ChatGPT. Це включає в себе покращення якості генерованого тексту, розширення можливостей аналізу семантики, а також зменшення вразливостей до атак та зловживань [54].

Приклади застосування включають системи діалогових асистентів, які можуть допомагати користувачам у вирішенні різних завдань, від пошуку інформації до здійснення покупок он-лайн. Наприклад, чат-боти в інтернет-магазинах можуть автоматично відповідати на запити клієнтів та надавати рекомендації щодо продуктів.

1.4.2 Актуальність задач обробки природної мови при створенні діалогових систем

Аналіз та синтез природномовних текстів грають важливу роль в сучасних програмних системах, зокрема в довідкових та діалогових

системах. Ці технології дозволяють покращити взаємодію користувачів з комп'ютерами та надати швидку та ефективну підтримку.

Діалогові системи розроблялися протягом кількох десятиліть, з моменту появи комп'ютерів з інтерфейсом користувача. Концепція використання природної мови для взаємодії людини і машини завжди була дуже бажаною. Вона пропонує зручність і легкість у порівнянні з необхідністю вивчати певну формальну мову та слідувати заздалегідь визначеним інструкціям. Поява текстових інтерфейсів, а потім інтерфейсів на основаних на вікнах меню, стало значним проривом, зробивши комп’ютери доступними та широко використовуваними інструментами для різних користувачів. Однак, незважаючи на їх практичність і зручність, цим інтерфейсам все ще бракує необхідної гнучкості. Вони часто характеризуються жорсткими наперед визначеними структурами і можуть стати складними та заплутаними. Отже, користувачі повинні вкладати значний час і зусилля в ознайомлення з усіма функціями таких інтерфейсів. Було б ідеально, якби користувачі могли просто виражати свої бажані дії або запитувану інформацію природною мовою, або за допомогою письмового і усного мовлення, тим самим усуваючи потребу в широкому дослідженні інтерфейсу. Завдяки впровадженню можливостей розуміння та обробки природної мови в діалогові системи користувачі отримають переваги від більш інтуїтивно зрозумілої та зручної взаємодії. Цей прогрес підвищує ефективність і зручність використання цих систем, забезпечуючи більш зручну роботу користувача та скорочуючи криву навчання, зазвичай пов’язану зі складними інтерфейсами.

1.4.3 Сучасні віртуальні помічники, що працюють із природною мовою

Наразі існує ряд віртуальних помічників, які використовують обробку природної мови як у письмовій, так і в усній формі. Яскраві приклади включають такі системи штучного інтелекту, як Apple Siri, Google Assistant, Amazon Alexa та Microsoft Cortana та інші. Однак розробка ChatGPT (яка

базується на базових моделях GPT OpenAI GPT-3, GPT-3.5 і GPT-4 [63] і була налаштована для розмовних програм із використанням як контрольованих методів навчання, так і з підкріпленням) [64 – 66] ознаменувала значний прорив у сфері штучного інтелекту, зокрема в сферах обробки природної мови (NLP) і розуміння (NLU). «GPT» — generative pre-trained transformer — тип великої мовної моделі (LLM). ChatGPT є суто текстовою системою, і їй бракує здатності розпізнавати та створювати усне мовлення або взаємодіяти з фізичними об’єктами матеріального світу. Тим не менш, її потенціал як потужної системи природної мови величезний, пропонуючи широкий спектр можливостей для надання, генерації та структурування інформації. За допомогою ChatGPT можна реалізувати різні функції та прийоми програми.

1.4.4 Великі мовні моделі як компонент діалогової системи

Можливість інтеграції ChatGPT з програмними системами. ChatGPT не тільки служить цінним автономним віртуальним помічником і компаньйоном, але також має великий потенціал для інтеграції в інші програмні системи, використовуючи свої можливості для виконання конкретних цілей. Ця перспектива відкрила нові шляхи для досліджень, зокрема для вивчення того, як можна використовувати ChatGPT для досягнення певних цілей, необхідних для діяльності системи. Тепер дослідники мають можливість заглибитися в дослідження методів ефективного використання та оптимізації можливостей ChatGPT у певних доменах. Це включає в себе адаптацію та налаштування ChatGPT для виконання завдань і вирішення проблем відповідно до унікальних вимог різних програмних систем. Ефективно «приборкавши» ChatGPT, дослідники можуть використати його сильні сторони та узгодити його з конкретними цілями різних програм, що призведе до подальшого прогресу в області обробки природної мови та розширить межі того, чого можна досягти за допомогою інтелектуальних інформаційних систем.

Особливості роботи із API ChatGPT. ChatGPT не має звичайного API, який складається з фіксованої кількості URL-адрес і відповідних команд із

заздалегідь визначеними діями. Однак він має API, який приймає команди природноою мовою [69]. Тим не менш, є певні особливості, пов'язані з цим підходом. По-перше, незважаючи на те, що ChatGPT володіє багатьма знаннями, рівень його розуміння та володіння кожною мовою може відрізнятися. Англійська є основною мовою для ChatGPT, і команди та інструкції мають бути написані англійською, навіть якщо мова йде про інші мови. Іншим важливим аспектом є те, що інструкції, які надаються ChatGPT, мають бути добре структурованими, зрозумілими та точними. Через обмеження на кількість токенів, які може обробити ChatGPT, інструкції мають бути короткими, але інформативними. Експериментальні дані з ChatGPT [70 – 73] показали, що одним з ефективних підходів для доставки лаконічних, але вичерпних команд і інструкцій є їх форматування у форматі JSON. Структуруючи інструкції у форматі JSON, дослідники та розробники можуть переконатися, що інформація впорядкована, легко інтерпретована та максимально ефективна для ChatGPT. Цей підхід дозволяє чітко повідомляти системі бажані дії та очікування, оптимізуючи взаємодію між користувачами та ChatGPT. Крім того, важливо знайти баланс між стислістю та ясністю в інструкціях, які надаються ChatGPT. Хоча інструкції мають бути стислими, щоб відповідати обмеженням маркерів, вони також мають містити достатньо інформації, щоб отримати точні інструкції ChatGPT. Досягнення цього балансу гарантує, що відповіді системи відповідають намірам і очікуванням користувача.

Незважаючи на відсутність традиційного API, ChatGPT пропонує API, який приймає команди природною мовою. Дотримання англійської як основної мови, структурування інструкцій у зрозумілий і точний спосіб і використання форматування JSON для лаконічних, але вичерпних команд є ключовими факторами для ефективного використання можливостей ChatGPT. Ці стратегії покращують взаємодію між користувачами та системою, сприяючи більш точним і значущим відповідям у порівнянні з технікою ланцюжка думок [72, 74 – 76].

Приклад застосування структурованих інструкцій-підказок для ChatGPT. У публічному репозиторії GitHub «Mr. Ranedeer: Your personalized AI Tutor!» [70] можна знайти приклад набору інструкцій, спрямований на перетворення ChatGPT на віртуального репетитора. Ці інструкції відформатовано у вигляді вкладених словників із стислими ключовими термінами, що представляють основні поняття передбачуваної мети. Значеннями, пов'язаними з цими ключами, можуть бути словники, які надають додаткові відомості, або природномовні фрази (англійською), що пропонують вичерпні та чіткі пояснення. Щоб скористатися цією функцією віртуального викладача, можна просто скопіювати та вставити надану інструкцію в інтерфейс ChatGPT. Завдяки цьому ChatGPT можна адаптувати як віртуального репетитора з різних предметних галузей, охоплених його базою знань. Концепція використання структурованих підказок для вказівок щодо поведінки ChatGPT у бажаний спосіб є спокусливою та багатообіцяючою.

Підключення додаткових модулів до ChatGPT. Крім того, ChatGPT має можливість включати плагіни, що зберігаються у зовнішніх ресурсах [77, 78]. Посилання на ці ресурси також можна включити в інструкцію, розширивши тим самим спектр функціональних можливостей. Це відкриває нові шляхи дослідження, які називають швидкою інженерією та метанавчанням. Останні дослідження [74, 75, 79, 80] підкреслюють важливість і актуальність вивчення цих напрямків. Основна мета оперативного проектування полягає в тому, щоб скерувати ChatGPT до відповідних реакцій, особливо в завданнях, які вимагають логічних виводів. Можна створювати інструкції, щоб прояснити тонкощі завдання та розбити його на послідовні кроки, направляючи ШІ до бажаного результату. Оперативна інженерія схожа на форму мистецтва, яка передбачає ретельний вибір конкретних слів, фразових структур та їх порядок, щоб викликати бажану поведінку ШІ. Було розроблено різні стратегії, включаючи використання імперативів для визначення ролі ШІ, запланованих

послідовностей, структурованих форматів даних (таких як JSON, XML, YAML), ланцюгів самокритики та інших. Визначення найефективнішої стратегії для оперативного проектування залишається відкритим питанням, але багатообіцяючі підходи були описані в [73], [81]. Поєднання результатів цих досліджень з іншими відповідними дослідженнями може дати цінну інформацію та сприяти розвитку галузі.

Методи розширення знань та можливостей ChatGPT. Оперативне проектування відкриває можливості для цілеспрямованого та конкретного навчання ChatGPT, дозволяючи йому отримати глибше розуміння областей, про які він може не мати достатніх знань. Хоча в ChatGPT існують такі механізми, як навчання моделей і тонке налаштування, вони можуть бути дорогими та вимагати великих, ретельно підібраних наборів даних. У деяких випадках застосування цього підходу може бути неможливим або непрактичним. Натомість цінну інформацію можна передати в текстовій формі або через структури даних у поєднанні з підказками JSON, які вказують алгоритмам ChatGPT, як обробляти надані дані. Це дозволяє розширити знання та можливості ChatGPT, що робить його придатним для діалогових систем або навіть систем керування. Хоча використання жорсткої структури, як описано в [82], може бути функціональним підходом, є місце для подальшого розвитку та дослідження. Системи, які взаємодіють із ChatGPT, можуть використовувати різноманітні інструкції чи шаблони, адаптовані для різних цілей. Самі підказки можна зробити більш гнучкими, додавши додаткові поля та надавши різні пояснення (значення) для кожного поля. Така система повинна містити інструкції про те, коли і як використовувати шаблони з ChatGPT і які конкретні значення слід використовувати в різних випадках. Ці інструкції щодо створення та використання структурованих підказок у ChatGPT можуть бути організовані в онтології, що приводить нас до системи, керованої онтологією.

Перспективи поєднання онтологокерованого підходу і великих мовних моделей. Використання онтологій, або мета-онтологій, як сховища

правил поведінки системи обговорюється в [81], хоча і без прямого посилання на чат ChatGPT або подібні програми. У цьому підході онтологія служить модулем прийняття рішень, керуючи системою тим, як обробляти конкретні типи даних і представляти їх в інтерфейсі користувача. Використовуючи онтології для керування поведінкою ChatGPT і використовуючи структуровані підказки, ми можемо розробити потужну систему, керовану онтологіями. Ця система покращує здатність ChatGPT адаптуватися та навчатися в певних сферах, використовуючи гнучкість підказок і вигоду від знань, що зберігаються в онтології. Поєднання оперативного проектування, використання онтології та систем на основі ChatGPT має великий потенціал для просування інтелектуальних інформаційних систем і технологій знань у різних областях. Фундаментальні концепції інформаційних систем з онтологічно керованою архітектурою широко обговорюються в [82, 83]. У «Тлумачному онтографічному словнику для інженерії знань» [67] архітектура, керована онтологіями, визначається як архітектура системи, яка обертається навколо двох основних компонентів: «Активної» комп'ютерної онтології та «Вирішувача задач». Ці компоненти спільно керують процесом обробки інформації, приділяючи особливу увагу вирішенню практичних проблем користувачів і підтримці цільової діяльності. Крім того, онтологічно керована інформаційна система [67] характеризується як комплексна система, що складається з кількох ключових елементів. До них відноситься база знань, яка тісно пов'язана з онтологіями (зазвичай представлена як кінцева колекція систематично інтегрованих баз знань у певних предметних областях), механізм логічного висновку, підсистема обробки додатків та інтерфейси (UI, API) для взаємодії з користувачем та/або інтеграція зовнішнього середовища. Разом ці компоненти сприяють ефективному використанню онтологічних знань в інформаційній системі.

1.4.5 Задачі перетворення природномовних висловів на формальні запити до бази знань

1.4.5.1 Роль текстових аналізаторів у роботі діалогових систем. Як було вже зазначено, проектування та розвиток природномовної діалогової системи — це комплексне завдання, яке включає побудову бази даних і модулів для взаємодії з нею, семантичний аналізатор тексту природної мови, процедури формування та надання відповідей у контексті діалогу.

Одним з найважливіших завдань тут є створення формальних запитів і формування природної мови відповіді, що основному і формує природномовний інтерфейс бази знань, яким фактично і є діалогова або довідкова система. Також постають задачі забезпечення оптимальної структури онтології та методів її автоматичного створення, які розглянуть, наприклад у роботі [84], а також проблема перебування в контексті діалогу, яка розглядається у роботі [85].

1.4.5.2 Приклади реалізації конвертацій природномовних запитів у формальні. Природномовні діалогові системи, так звані чат-боти, мають довгу історію та низку підходів. Нижче ми розглянемо деякі цікаві приклади діалогових систем, розроблених в останні роки (до епохи «моделей-трансформерів» і GPT), зокрема ті, що в тій чи іншій мірі використовують у своїй структурі онтологію.

Шаблонні структури речень. Хороший приклад діалогової системи природної мови описано у роботах [86, 87]. Як і більшість інших, автори розглядають там англійську мову та її структурні особливості. Засоби аналізу вхідних фраз користувачів передбачають, що речення записані англійською мовою мають досить регулярну структуру, яку можна виразити через досить обмежений набір шаблонів. Постійна частина такого шаблону відповідає його семантичному типу (так званому «наміру»), а змінні частини показують місця у фразі, з яких мають бути витягнуті поняття. Ці місця вказуються відповідно до певних очікуваних «намірів», що їх конкретизують витягнуті поняття. Наприклад, є шаблон: "Show me {@M} by {@D} for {@V}.".

Фігурні дужки тут позначають місця, де очікуються значущі поняття. Маркери в заповнювачах показують наступне: @M відповідає основній запитаній концепції; @D — це вибір категорії для таких понять, як @M; @V є параметром фільтрації. Наприклад, є фраза "Show me admits by major diagnostic category for 2017", що повністю відповідає наведеному вище шаблону. Поняття, яке користувач просить показати, є визнанням (у цьому випадку це «кількість госпіталізацій»), Категорія відбору і сортування є основною діагностичною категорією (основними діагностичними категоріями), а параметром фільтрації є «2017», у вигляді якого можна вгадати концепцію року. Структура та постійна частина запиту визначає його «намір». Для кожного «наміру» існує певний пакет запитів до баз даних та інструкції щодо візуалізації та представлення їх результатів у користувацький інтерфейс. Бази даних, що містять базову інформацію, в цьому випадку є переважно реляційними. Однак, названа система також містить онтологію, яка служить для структурування та категоризації типів і вимірювання даних, що зберігаються в основній базі даних. «Наміри» та поняття, що надходять від вихідної фрази користувача порівнюється з онтологією для визначення найбільш близької до запитуваної категорії з наявних у базах даних. Тобто онтологія в даному випадку грає скоріш допоміжну роль. Онтологія створюється автоматично на основі реляційної моделі даних. Автори відзначають здатність системи залишатися в контексті діалогу. Проте, фактично, запропонований у роботі підхід до цієї проблеми переважно орієнтований на заміну займенників. Якщо змінні шаблону аналізу відображаються як займенники або відсутні, тоді програма використовує відповідні дані з останнього з попередніх запитів. В якщо в попередніх запитах немає інформації, підставляються значення за замовчуванням. Ці стандартні значення формуються на основі найпоширеніших запитів, зібраних під час використання системи. Наразі система не містить автоматизованого навчання, хоча автори декларували можливість його розвиток у майбутньому.

Основні особливості системи з [86, 87] можна коротко описати так: працює тільки з англійською мовою та адаптована до її особливостей; аналіз вихідних фраз здійснюється за шаблонами; здатність до навчання відсутня; основні дані зберігаються в реляційній базі даних; онтологія існує, але грає допоміжну роль, і створюється автоматично на основі реляційної бази даних; має набір зазначених "намірів" та пов'язані з ними схеми подання інформації (у вигляді таблиць, діаграм і графіків); не створює природних мовних відповідей; у частковій мірі реалізує методи перебування в контексті діалогу.

Фреймворк PAROT. Діалогові системи, які використовують онтологію як основну базу знань, зазвичай є просто природними мовні інтерфейси графової бази даних. У якості мови для формальних запитів часто використовується SPARQL. Основним завданням, яке виникає при їх розробці, є перетворення природного мовного запиту користувача у формальний. Нижче наведено кілька прикладів таких перетворювачів, які були створені в останні роки. Один із прикладів автоматизованого перетворення природномовних запитів у SPARQL є PAROT [88]. Він використовує підхід, який генерує найімовірніші RDF-трійки на вимогу користувача. RDF-трійка потім перевіряється спеціальним модулем, що містить аналізатор для обробки запитів користувачів. Тоді триплети RDF, отримані таким чином трансформуються в онтологічні триплети за допомогою спеціального тезауруса. Згенеровані онтологічні триплети використовується для створення запиту SPARQL, за допомогою якого отримується відповідь із онтології. Тестування фреймворку PAROT авторами [88] показало, що для простих запитань він показує точність приблизно 81 – 82 %, для комплексних – близько 43 – 56 %, а для конкретного тематичного набору даних (географія) точність зростає до 88%.

Фреймворк FREyA. Іншим прикладом реалізації методів перетворення природної мови в SPARQL є FREyA [89]. Вона доступна на GitHub [90]. FREyA пропонує інтерактивний природномовний інтерфейс для онтологічних запитів. Він використовує синтаксичний аналіз у поєднанні з

пошуком на основі онтології для інтерпретації питань і, при необхідності залучає користувача. Вибір користувача використовується для навчання системи, що підвищує точність функціонування системи. FREyA наразі реалізована лише для англійської мови. У [90] наведено кілька прикладів, які ілюструють, як запитання природною мовою можна перетворити на SPARQL за допомогою FREyA. Слід зазначити, що конфігурацію FREyA можна налаштувати під певну структуру онтології.

Система LODQA. Також варто загадати систему LODQA (Linked Open Data Question Answering) представлену в роботі [91]. Вона приймає запит природною мовою у якості вхідних даних та повертає запити SPARQL із відповідними відповідями в результаті. Система складається з кількох модулів. Перший модуль обробляє природномовний запит. Він відповідає за розбір і створення графового представлення запиту, яке називається псевдографічним шаблоном. Псевдографічний шаблон містить вузли та зв'язки. Вузли зазвичай відповідають основним іменним групам, а зв'язки залежностям між ними. Крім того, псевдографічний шаблон вказує, який вузол з онтологічного графу є фокусом запиту, тобто те, що користувач збирається отримати у відповідь. Псевдографічний шаблон — це шаблон графа пошуку цільового графа із RDF-підграфів, який відповідати йому. Однак його називають псевдографічним шаблоном, оскільки він ще не базується на цільових даних. Щойно перший модуль згенерував псевдографічний шаблон із заданого природномовного запиту, активується наступний модуль, який відповідає за пошук URI та вузлів значення в псевдографічному шаблоні. URI та значення мають бути присутніми в цільовому наборі даних. Кожен вузол псевдографічного шаблону пов'язується з URI набору даних. Природномовна концепція може бути нормалізована (зведена до початкової граматичної форми) в більш ніж в один спосіб через можливу неоднозначність. Таким чином, можна отримати більше ніж один шаблон з одного псевдографічного шаблону. Третій модуль для створеного псевдографічного шаблону виконує пошук у цільовому

наборі даних для відповідних частин, беручи до уваги можливі зміни, які можуть бути відбуваються в наборі даних. Для врахування структурних відмінностей між зв’язками псевдографічного шаблону та фактичної структури цільового набору даних, цей модуль намагається створити SPARQL запити для всіх можливих структурних варіацій. Потім запити SPARQL надсилаються до цільової кінцевої точки, де отримуються відповіді, які потім надсилаються користувачеві. Аргументи запиту можуть бути а примітивного типу, наприклад S, N або NP, або складного типу, наприклад S/NP або NP/N. Слеш означає, що аргумент має відображатися праворуч, а зворотна коса риска означає, що аргумент має відображається зліва. У системі використовується таке позначення частин мови, наприклад, NN – іменник, DT – означення (прикметник), VB – дієслово. Щоб полегшити ідентифікацію RDF- триплетів, слова у реченні лемматизуються та надаються відповідні граматичні характеристики. Розглянута система LODQA орієнтована на роботу тільки з англійською мовою. Детальні особливості його функціонування в [91] не наведено, обмежуючись загальним описом та аналізом прикладів роботи.

1.4.5.3 Інші приклади онтологокерованих діалогових систем. Хоча розвиток діалогових систем, а також машинна обробка і «розуміння» тексту природною мовою, здебільшого здійснюються для англійської мови, вони не є такими обмежується нею. Наприклад, в [92] представлена діалогова система для німецької мови. Ця стаття здається цікавою, оскільки розглянута у ній система також включає онтологію. У цьому випадку онтологія виступає як діалог-менеджер (OntoDM), який підтримує стан розмови. Онтологія також використовується тут як а база знань. Ці ролі поєднуються. Знання предметної області використовуються для відстеження об'єктів, тобто вузлів онтологічного графу (класи), які є продуктами та послугами, представленими в базі знань. Таким чином була введена можливість запам'ятовування історії розмови. Також значна частина статті [92] присвячена проблемам роботи із лінгвістичними особливостями німецької мови. На момент публікації [92]

дослідницька робота все ще тривала і оцінки формальних критеріїв для системи ще не було отримано. Робота [93] є прикладом розробки а діалогової системи для корейської мови, яка принципово відрізняється від європейських мов.

1.4.6 Графова СУБД Neo4J і мова запитів Cypher

Окрім СУБД, що працюють у зв'язці OWL/SPARQL, такі як Jena Fuseki, що де-факто наразі є стандартом, існують і альтернативні підходи до графових баз даних, які також можуть бути використані для зберігання і роботи з онтологією. Однією з найперспективніших таких систем керування графовими базами даних (СУБД) є Neo4j [94], яка забезпечує досить високу продуктивність і масштабованість, а також підходить для роботи з великими обсягами даних. Наразі це також одна з найпопулярніших графових СУБД. Мовою формальних запитів прийнятим у Neo4j є Cypher. Вона має широкий спектр можливостей, досить гнучкий і відкрита для додаткової функціональності через плагіни, наприклад, для реалізації типових алгоритмів на графах. Однак на даний момент, на відміну від SPARQL, існує не так багато розробок для перетворення запитів природною мовою у формальні запити на Cypher. Серед небагатьох прикладів можна назвати роботи [95, 96]. Запропонована в [96] система досить примітивна. Запити повинні мати заздалегідь визначену структуру. Це підхід близький до представленого в [86]: набір шаблонів речень на природній мові, де деякі фрагменти замінюються спеціальними позначеннями, як місця, з яких потрібно витягти поняття для підстановки у шаблон запиту. Кожне таке шаблонне речення відповідає певному запиту на Cypher. Описаний підхід має свої переваги та недоліки. Основною перевагою є його простота. А головним недоліком є те, що справжня діалогова система потребує великої кількості таких речень-шаблонів, які містять усі можливі варіанти запитань. Більш того, такий підхід може бути виправданим для мов із регулярною структурою речень, таких як англійська, де потрібна значно менша кількість шаблонів фраз. Флективні мови, як-от українська, мають складну структуру речення із

досить вільним порядок слів. Цей факт значно збільшує кількість необхідних шаблонів.

Таким чином, однією з основних задач роботи постає розробка природномовної діалогової системи на основі на онтології, яка створюється автоматично шляхом семантичного аналізу тексту, враховуючи особливості флективних мов, зокрема української, з використанням як SPARQL запитів до онтології, так і дослідження можливостей Neo4j та мови запитів Cypher для роботи з базою знань.

1.5 Висновки за розділом 1 і постановка задачі дослідження

Методи аналізу і синтезу природномовного тексту пройшли довгий шлях розвитку, було створено велику кількість моделей і підходів до даної проблеми. Існувало безліч спроб більш і менш вдалих, проте всі вони мали наукову цінність, принаймні даючи знати, які способи працюють краще, а які призводять до посередніх результатів, або супроводжуються певними труднощами. Останніми роками у вказаних областях було досягнуто значного прогресу, основою якого стало створення комплексних нейронномережевих моделей-трансформерів, вершиною яких стали сильні мовні моделі, такі як ChatGPT. Такі моделі здатні аналізувати і генерувати великі обсяги текстів високої якості різними мовами із багатьох предметних областей.

Разом з тим, більш прості, старіші підходи, в тому числі системи засновані на прописаних правилах також продовжують розвиватися і повністю не втрачають своєї актуальності. Це обумовлено рядом факторів. Так, сильні мовні моделі, такі як ChatGPT, на перший погляд працюють дуже ефективно і здатні виконувати широкий спектр задач. Але і вони не позбавлені певних недоліків, принаймні, у їх стані розробки на поточний час. Серед таких недоліків можна виділити наступне. Сильні мовні моделі дійсно мають широку базу знань, проте вона не є всеосяжною. Існує багато

спеціалізованих предметних областей не представлених у них у наявному вигляді або представлених досить бідно. З цим зазвичай починаєш зіткатися при роботі з тим же ChatGPT, коли питання/завдання починають стосуватися вузьких наукових і технічних областей, де загальні знання виявляються недостатніми або застарілими (оновлення бази знань таких моделей довгий і затратний процес, що не може відбуватися так часто, як хотілося би). Звичайно, у таких моделей існують механізми донавчання і тонкого налаштування, але і вони стикаються з певними труднощами. Донавчання потребує великого за обсягом набору розмічених даних, створення і підготовка якого є непростою і кропіткою роботою. Також воно потребує значних обчислювальних ресурсів і займає досить багато часу. Робота з такими моделями, особливо, якщо ми бажаємо мати прийнятний час відповіді також потребує великих обчислювальних ресурсів. Таким чином, особливо в умовах обмеженості ресурсів і наявності задач, що можуть бути вирішені в інший спосіб великі мовні моделі стають занадто «великою гарматою» з якої безглуздо «стріляти по горобцях». Окрім того, наразі робота не з усіма природними мовами у таких моделях представлена на однаково високому рівні. Найбільш ефективно вони працюють з англійською мовою, корпуси даних задіяні для якої були найбільшими. Коли мова заходить про роботу з менш поширеними і менш популярними мовами, результати виявляються дещо скромнішими. Таким чином, у ряді випадків видається простішим і доцільнішими створення простішої моделі, в тому числі і побудованої на правилах.

Більш того, існують широкі перспективи поєднання підходів, що може дозволити в певних випадках досягти ще більшої ефективності їх використання задіючи сильні сторони різних методів. Так постає задача створення формальних моделей (як інформаційних, так і функціональних) і розробка методологічних основ для використання онтологічно керованої системи структурованих підказок у взаємодії з ChatGPT. Ця система дозволяє надавати інформацію та робити розширення бази знань шляхом отримання

нової інформації з існуючих одиниць знань на основі певного набору контекстів. Логічна дедукція передбачає отримання нової інформації на основі встановлених фактів, правил і логічних принципів. Це дозволяє системі робити логічні висновки та встановлювати зв’язки між різними фрагментами інформації. Використовуючи дедуктивне мислення, система може розширити своє розуміння та створити додаткові одиниці знань, які не були надані явно. Цей процес отримання нових інформаційних одиниць із раніше відомих відіграє вирішальну роль у збільшенні знань системи та покращенні її загальної функціональності. Це дозволяє системі робити інтелектуальні висновки, виявляти приховані зв’язки та надавати користувачеві більш повну та цінну інформацію.

Розробляючи формальні моделі, це дослідження забезпечує структуровану основу для організації та представлення знань у систематичний спосіб. Такі моделі, що охоплюють як інформаційні, так і функціональні аспекти, закладають основу для ефективної інтеграції онтологічного підходу з ChatGPT. Об’єднана система забезпечує складну діалогову взаємодію, яка включає висновки та використовує контекстну інформацію для надання значущих відповідей.

Таким чином, на основі огляду сучасного стану розробки проблем аналізу і синтезу природномовних текстів постають наступні задачі дослідження, що вирішуються у даній роботі:

1. Розробити методи аналізу природномовного тексту прийнятні для мов флективного типу, зокрема, української, у двох основних аспектах призначення:

- автоматичне створення графових баз знань онтологічного типу на базі природномовного тексту або набору таких текстів;

- аналіз коротких фраз і запитів користувачів для визначення намірів і іменованих сутностей, що їх конкретизують, висловлених у них, необхідних для побудови формальних запитів до бази знань.

2. Створення гнучких шаблонів формальних запитів на мовах SPARQL та Cypher і програмних засобів для роботи з ними, необхідних для отримання релевантної інформації з бази знань.

3. Розробка методів генерації осмислених природномовних відповідей і їх представлення користувачеві із залучанням як підходів, заснованих на правилах, так і з залученням великих мовних моделей.

4. Практичне створення природномовних діалогових і довідкових систем, які втілюють розроблені підходи до аналізу і синтезу тексту із залученням баз знань онтологічного типу.

5. Проведення апробації і тестування розроблених діалогових систем, визначення формальних критеріїв якості їх роботи.

РОЗДІЛ 2

РОЗРОБКА МЕТОДІВ АВТОМАТИЗОВАНОГО СЕМАНТИЧНОГО АНАЛІЗУ ПРИРОДНОМОВНОГО ТЕКСТУ В АСПЕКТІ ОНТОЛОГІЧНОЇ ІНЖЕНЕРІЇ

2.1 Вступ

Суттєву роль у онтологічній інженерії відіграють процеси автоматичного семантичного аналізу природномовного тексту. Можна виділити дві основних сфери застосування такого аналізу. Перша стосується автоматизації процесів побудови описів онтологічних графів, друга фактично є створенням природномовних інтерфейсів для подібних баз знань.

Важливість автоматизації побудови онтологій продиктована в першу чергу значними витратами часу і сил на ручне створення онтологій. Джерелом для створення онтологічних баз може бути текст на природній мові. З такого тексту можна виділити певну інформацію і структурувати її у вигляді графу, що ілюструє описані у тексті сутності і взаємовідносини між ними. Розвиток техніки наразі дозволяє у необхідній мірі автомати зівани процес знаходження таких відношень між сутностями описаними словами, словосполученнями і реченнями тексту.

Другий названий аспект є важливою складовою при розробці довідкових і діалогових систем. Такі системи є значною мірою природномовними інтерфейсами баз даних (реляційних, графових тощо). Застосування природної мови для машинно-людинної взаємодії замість формальних мов запитів (SQL, SPARQL, Cypher і т.д.) багато в чому здатне спростити діяльність користувача таких систем, зазвичай далекого від програмування, формальних мов і, окрім того, не знайомого з фактичною структурою використаної у системі бази даних. Тут і виникає задача переходу від природномовних фраз (питань та інших реплік користувача) до

запитів спрямованих безпосередньо до бази даних, які формулюються на формальних мовах. У даній роботі ми не торкаємося SQL- запитів і реляційних баз даних, хоча цей напрямок досліджень також актуальний і активно розвивається. Основна наша увага була прикута до онтологій, що представляють собою форми графів, а тому і взаємодія з ними обслуговується графовими СУБД. Основною мовою формальних запитів при цьому є SPARQL. Проте графові СУБД можуть використовувати і інші робочі мови. Однією з таких мов, що також розглядалася у нашій роботі є Cypher, що його застосовує Neo4J.

2.2 Розробка феноменологічної теоретичної моделі системи семантичних категорій

Однією з найважливіших задач комп'ютерного аналізу природномовних текстів є встановлення сенсу, ідеї, що подається автором. Іншою, не менш важливою, проблемою є автоматизована побудова баз даних і баз знань, спираючись на інформацію отриману з природномовних текстів. В обох випадках мова йде насамперед про семантичний аналіз, сутність якого полягає у встановленні семантичних (смислових) типів висловлювань і прив'язаних до них понять, що конкретизують висловлювання певного типу.
Центральним питанням у рамках проблеми, що розглядається є класифікація семантичних типів висловлювань. Роботи в даному напрямку ведуться вже протягом багатьох років. Зокрема, проблематиці семантичних категорій у онтологічній інженерії присвячена монографія [108]. Проробляли різні підходи, як то створення системи семантичних відмінків, класифікація елементарних висловлювань, семантична типізація зв'язків між словами, побудова універсальної семантичної метамови. Попре це, єдиної феноменологічної теорії семантичних типів досі не створено. Натомість існують незалежно розроблені класифікаційні системи, що нараховують від десятків до більш ніж сотню типів семантичних зв'язків або семантичних

відмінків. Такі класифікації можуть частково бути подібними між собою, частково різнитися, але тенденції збігання до єдиного консенсусу не спостерігається.

Однією зі спроб створення ієрархічної системи семантичних категорій є теорія В. Н. Сагатовського. В рамках даної концепції робиться змога вивести систему семантичних категорій, виходячи з базового аксіоматичного набору. Таким набором аксіоматичних типів за В. Н. Сагатовським є: «Елемент», «Множина», «Буття», «Небуття», «Змінення». Далі за допомогою логічної комбінації цих базових понять і їх наступного поєднання таких комбінацій виводиться понад сотню семантичних категорій. Але недоліком системи В. Н. Сагатовського є те, що це є скоріш філософська концепція, а не чітка логічна конструкція, а мотивування породження нових категорії на базі існуючих часто виглядає доволі суб’єктивно і навіть сумнівно. Тим не менш, теорія В. Н. Сагатовського є однією з небагатьох спроб побудувати чітку і стрункy систему семантичних (смислових) категорії, що може бути універсальною і здатною до масштабування, а не побудованою суто для конкретної предметної області та мови.

Спираючись на підходи, запропоновані у роботі [108], а також на системи семантичних класифікації, запропоновані такими авторами, як Ю. Д. Апресян, Н. М. Леонтьєва та ін., нами було запропоновано наступний принцип побудови типів семантичних категорій шляхом логічної комбінації. У запропонованому нами варіанті є 8 базових аксіоматичних категорій: «Буття», «Час», «Простір», «Монада», «Постійність (константність)», «Множина», «Міра», «Заперечення». Кожна наступна категорія отримується шляхом комбінування двох інших. Комбінувати категорії можна довільно: базові з базовими, похідні з похідними, базові з похідними. Одночасне комбінування більш ніж двох категорії виглядає надлишковим, тому як подібне поєднання можна розкласти на більш прості, бодай іноді і умоглядні та технічні, комбінації. Так, наприклад, потрійне поєднання можна розкласти

на три подвійних, які, в свою чергу, можна комбінувати як між собою так і з вихідними категоріями.

На рисунку 2.1 наведено приклад схеми виведення семантичних категорій на базі кореневих аксіоматичних категорій, а на рисунку 2.2 – на основі похідних відношень.

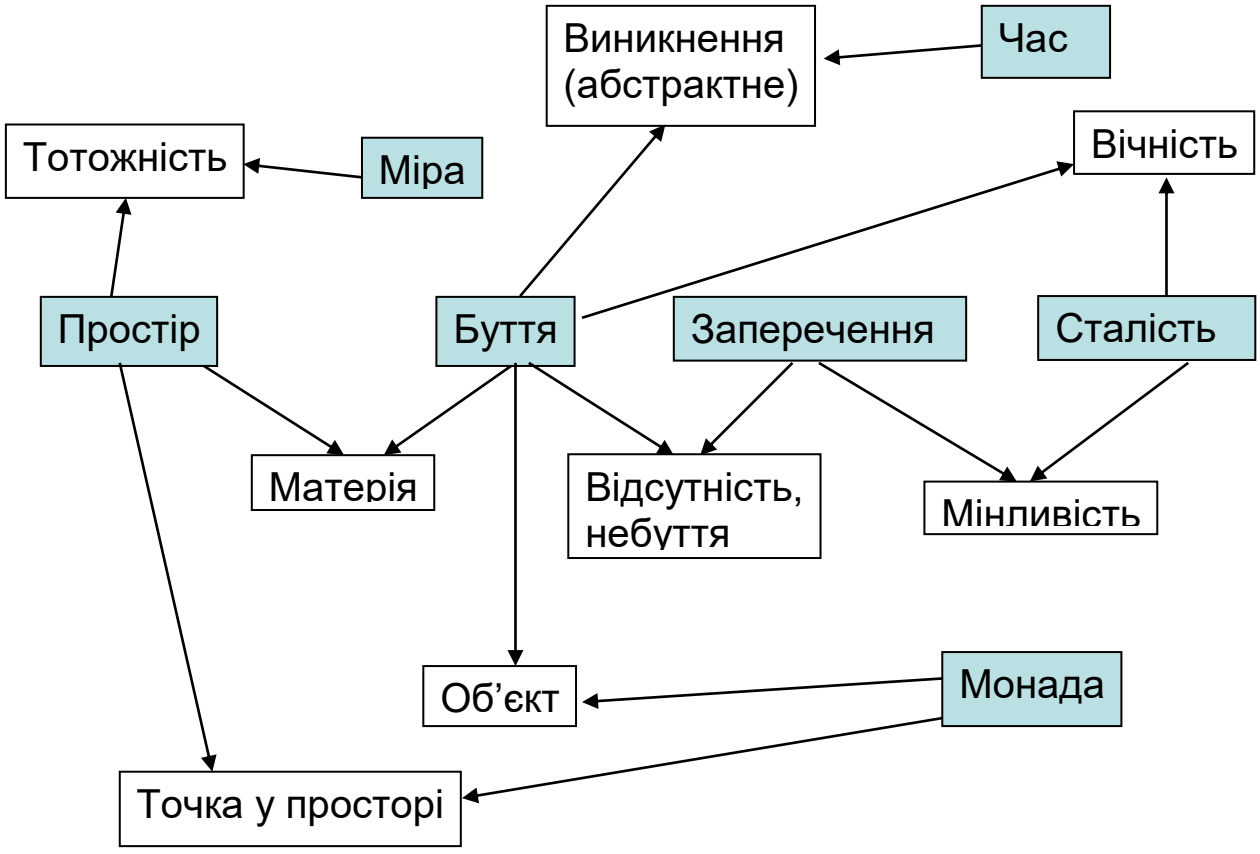


Рисунок 2.1 – Приклад схеми виведення похідних семантичних категорії на основі базових (аксіоматичних) категорій

Наведемо приклади виведення семантичних категорії за запропонованим способом. Так безпосереднє комбінування категорії вищого рівня може дати наступні типи, наприклад: «Буття»+ «Простір» → «Матерія»; «Буття»+ «Заперечення» → «Відсутність, небуття»; «Буття» + «Монада» → «Об’єкт»; «Постійність» + «Заперечення» → «Мінливість»; «Постійність» + «Міра» → «Тотожність»; «Постійність» + «Буття» →

«Вічність»; «Монада» + «Простір» → «Точка у просторі»; «Час» + «Буття» → «Виникнення (абстрактне)».

Наведемо також декілька прикладів виведення семантичних категорій нижчого рівня: «Виникнення» + «Заперечення» → «Зникнення»; «Зникнення» + «Виникнення» → «Заміщення»; «Заперечення» + «Тотожність» → «Відмінність»; «Відмінність» + «Міра» → «Відношення»; «Множина» + «Відношення» → «Структура»; «Мінливість» + «Простір» → «Неоднорідність»; «Матерія» + «Мінливість» → «Дія (абстрактна)»; «Дія» + «Об'єкт» → «Вплив»; «Вплив» + «Об'єкт» → «Взаємодія»; «Міра» + «Взаємодія» → «Властивість».

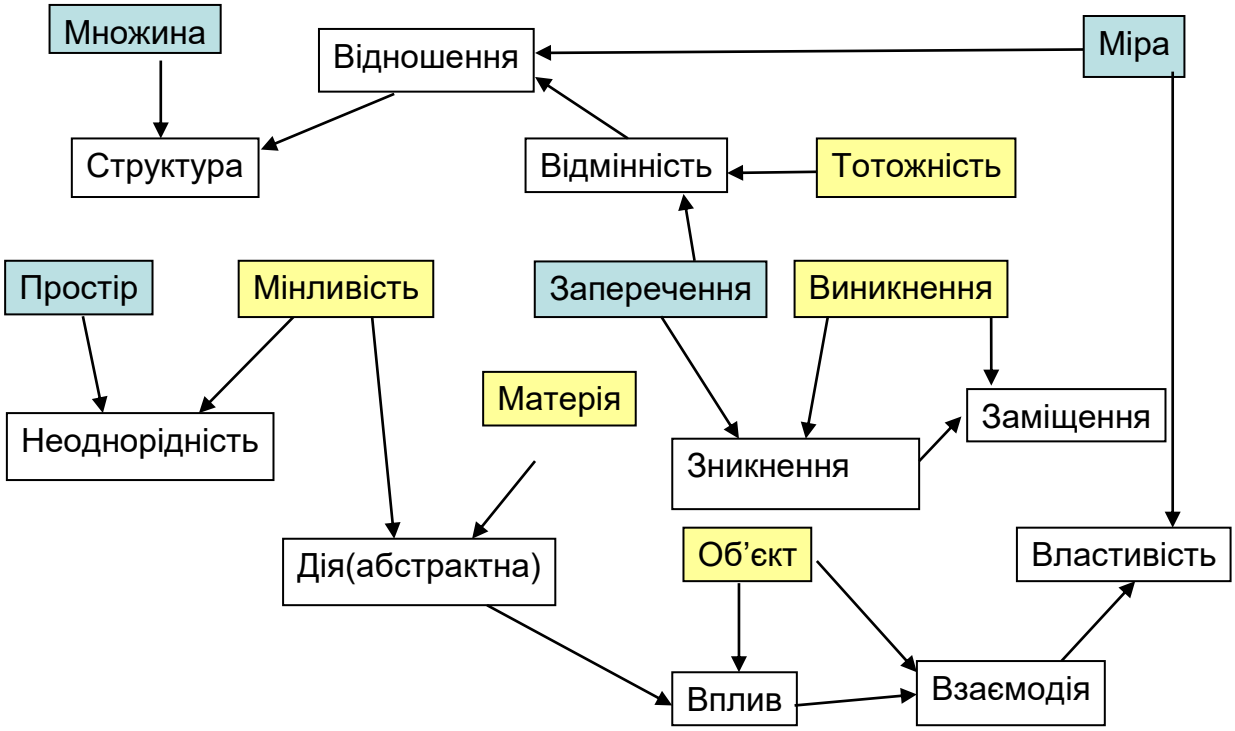


Рисунок 2.2 – Приклад схеми виведення семантичних категорії нижчого рівня на основі базових (аксіоматичних) та похідних категорій

У такий спосіб нами було виведено 148 семантичних категорій, що включають у себе більшість семантичних типів, запропонованих Ю. Д. Апресяном, Н. М. Леонтєвою та іншими авторами. Ця цифра не є

границею. Очевидно, що таким чином можна вивести будь-які семантичні категорії реального та умоглядного світу.

Побудова системної класифікації семантичних відношень створює ґрунт для розробки баз знань онтологічного типу. В свою чергу, це дає базову систему, на яку можна орієнтуватися при розробці методів аналізу природномовних текстів. Розробник може обмежити, використовуваний для конкретної задачі і предметної області, набір семантичних категорій, пошук яких планується здійснювати у тексті, або використовувати при створенні онтології, а також при побудові запитів до онтології на основі природномовного тексту. Окрім того, така система надає можливості для побудови додаткових висновків, ґрунтуючись на графо вій базі даних онтологічного типу, в основі типізації зв'язків між вершинами якої, закладено запропоновану систему виведення семантичних категорії.

2.3 Методи автоматизованої побудови онтології на основі природномовного тексту

2.3.1 Проста онтологія, що будується на розмічених текстах

Структура бази даних повинна відповідати наступним основним вимогам: бути такою, що може бути легко і надійно сформованою автоматично з використанням допоміжного програмного засобу з вихідного текстового матеріалу; бути придатною для отримання якомога релевантних відповідей.

Структура онтології

Розглянемо докладніше запропоновану структуру.

Основні класи онтології:

Action – поняття, що позначають дії, дієслова;

Adjective – прикметники, що з якихось причин при розборі тексту не увійшли до іменних груп;

Adverb – прислівники;

Number – числівники (в тому числі і записані цифрами);

Object – поняття, які позначають предмети, об’єкти, явища (іменники і іменні групи);

Marker – питальні слова і підрядні сполучники («де», «ким», «що», «кого», тощо);

ReadyAnswerLink – посилання на контексти;

Count – позначення числа кратності входження поняття у контекст (при входженні більше ніж один раз);

Quantity – фактична кратність входження поняття у конкретний контекст (при входженні більше ніж один раз);

ConditionIntersection – комбінації (перетини) ключових понять, що характеризують контекст.

Основні властивості (ObjectProperty) онтології:

MainProperty – прив’язує комбінації (перетини) ключових понять – нащадки класу ConditionIntersection (входять у domain) до посилань на контексти – нащадки класу ReadyAnswerLink (входять у range).

BindQuantityToInteraction – прив’язує контексти (входять у domain) до кратностей входження у них певних понять – нащадки класу Quantity (входять у range).

IntersectionCount – прив’язує фактичні кратності входження понять у контексти – нащадки класу Quantity (входять у domain) до позначення числа кратності входження – нащадки класу Count (входять у range).

Підкласи класу Object, можуть мати своїх нащадків, що є іменними групами. Ієрархія нащадків класу Object – є наступною: найвище йдуть окремі іменники, що є головними у іменних групах і визначають найбільш загальний варіант поняття, далі по ієрархії вниз йдуть іменні групи по мірі нарощування слів у них, що визначають більш конкретні поняття.

Один і той же підклас Count, може бути пов'язаний з різними підкласами Quantity, що дозволяє уникнути дублювання – так, наприклад, три рази можуть зустрічатися різні поняття у різних контекстах. В свою

чергу, підклас класу Quantity може бути прив’язаний до декількох посилань на тексти відповіді – одне й теж поняття може входити, наприклад, два рази, у різні контексти.

Взаємозв'язок між класами забезпечується за допомогою властивостей, які успадковуються від основних (кореневих) властивостей. Ці успадковані властивості забезпечують з'єднання конкретних нащадків основних класів, які увійшли до їх розділів domain і range. Тип зв'язку визначається певною кореневою властивістю, яку успадковує конкретна з’єднуюча властивість.

Належність сутностей (слів) до класів Action, Adjective, Adverb, Number, Object, Marker, а також прив’язки їх до певних підкласів класу Quantity можуть бути застосовані при ранжуванні отриманих відповідей.

Спрощена структурна схема основних класів запропонованої онтології наведена на рисунку 2.3

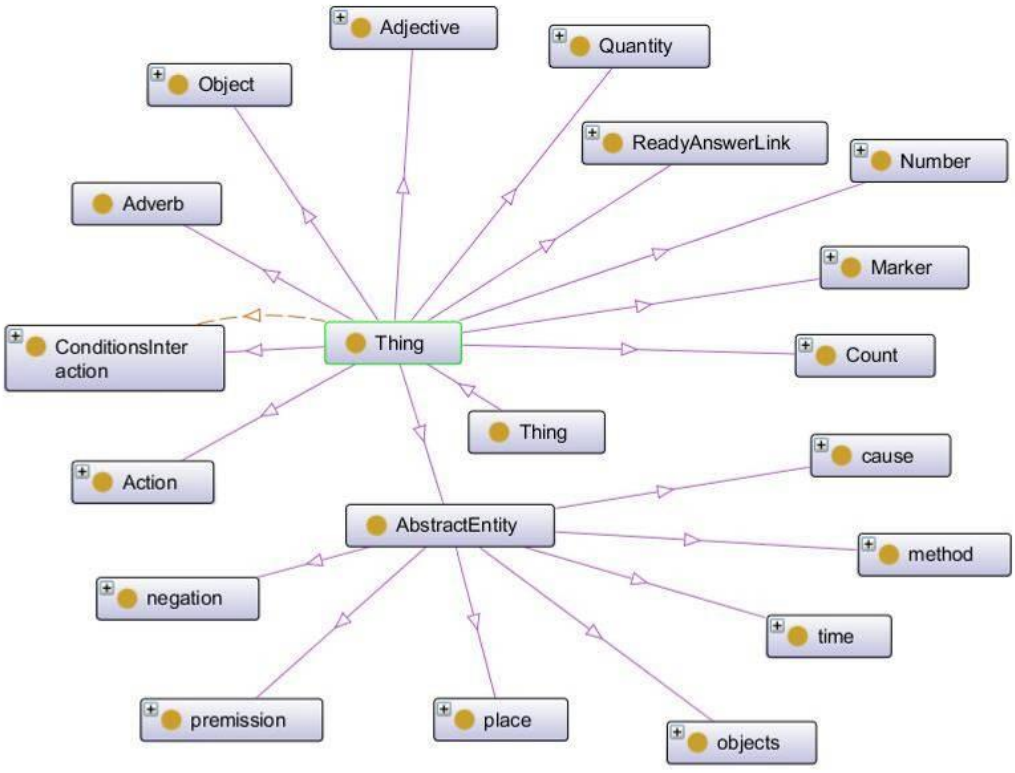


Рисунок 2.3 – Структурна схема основних класів простої онтології, створюваної на базі текстів

Формат зберігання онтології. Повні контексти можуть бути відсутніми у прямому вигляді в OWL-файлі онтології; замість них у файлі можуть міститися лише посилання на ці контексти. Такий підхід обумовлений прагматичними міркуваннями, оскільки включення довгих текстових матеріалів безпосередньо може призвести до значного збільшення розміру OWL-файлу, негативно позначаючись на продуктивності обробки SPARQL-запитів. Додатково, контексти можуть включати символи, що не відповідають стандартам OWL та/або XML-формату. Важливо відзначити, що такий підхід дозволяє розширити базу знань не лише текстовим, а й мультимедійним матеріалом, таким як зображення, аудіо та відео, а також інтегрує комплексні веб-ресурси та застосування.

Особливості вхідних даних для створення онтології розглянутого типу. При автоматизованому створенні онтології подібного типу передбачається, що вхідний текст має теги або регулярну структуру, що полегшує отримання гарної якості кінцевої онтології. Розмічений тегами текст – це фактично набір текстів, кожен із яких має принаймні заголовок та основні частини. Такий набір текстів може бути отриманий, наприклад, шляхом сканування веб-сторінок інформаційного ресурсу. На кожній сторінці можна знайти назву статті, її текст і, можливо, деяку іншу інформацію в певних місцях: дату, автора, місце розташування, тематику, тип статті тощо. Додатковий семантичний аналіз блоку основного тексту та заголовка (якщо він є) може бути використаний для оцінки типу та тематики.

Переваги та недоліки онтології запропонованої структури.

Переваги. Створення онтології з запропонованою структурою можна легко автоматизувати для різних текстових матеріалів. Коли вона використовується як база знань для довідкової системи, то ефективно забезпечує можливість отримання відповідей, порівнюючи концепції із вхідної фрази користувача із наборами понять, пов'язаних з контекстами. Така структура також легко дозволяє отримати серію текстових відповідей, пов'язаних загальним дискурсом, що додає користувачеві додаткову цікаву

інформацію. Крім того, ця структура спрощує процес ранжування відповідей за відповідністю до питання користувача. Важливою перевагою такої структури є її простота і уніфікованість у формулюванні SPARQL-запитів.

Недоліки. Однак цей підхід до структури онтології має і численні недоліки. Серед них можна виділити, на сам перед, низьку семантичність. Так, уніфікований шаблон SPARQL-запиту, за однаковою схемою формується незалежно від типу питання користувача. Відповіді є однорідними і не структурованими, як і їх прив'язка до певних класів, через уніфіковані властивості. Таким чином, знижується розуміння питання користувача, а відповідь часто буває «засміченою» побічними контекстами, або, навпаки, потребується багатократна редукція понять з фрази користувача і повторення запитів, використовуючи всякий раз інші варіанти редукованих наборів понять. Тому така структура не є оптимальною для випадків, де типи запитуваних відповідей можуть бути чітко класифіковані через свою різнорідність.

Автоматизація побудови простої онтології на базі тексту

Особливості вхідних даних. Заповнення будь-якої бази даних, в тому числі і графової, вручну являє собою довгий, нудний процес, під час якого можливі випадкові помилки викликані людським фактором. Особливо це стає актуальним при великих обсягах даних, які не збираються поступово, а повинні бути введені якомога скоріше, бажано – всі відразу. Особливо складно стає, у випадку, коли вихідні дані не представлені у структурованому вигляді, а наприклад являють собою просто текст чи набір текстів, сутності з яких, що вносяться до бази, ще належить визначити.

Тому актуальною стає задача автоматизації (принаймні часткової) створення графових баз даних онтологічного типу з використанням природномовних текстів.

Строго кажучи, автоматизація у даному випадку є частковою, тому як побудова OWL онтології відбувається з використанням заздалегідь вручну розміченого тексту. В одному з варіантів реалізації розмічений текст являє

собою формально XML-файл з досить простою структурою. Текст поділяється на логічні блоки, кожен такий блок представлено контейнерним XML-тегом, наприклад:

```
<data>
     <paragraph>
          <title>заголовок фрагменту</title>
          <text>контекст</text>
     </paragraph>
     …
     <paragraph>…</paragraph>
</data>
```

Підготовлений розмічений файл передається на вхід спеціально створеній програмі, що на виході генерує два файли – відповідну OWL онтологію і список ключових слів до неї.

Програмна реалізація побудови онтології. Схема роботи програми для побудови OWL-онтології із розміченого тексту наведена на рисунку 2.4 у вигляді UML-діаграми дій.

Програма генерації онтології, спеціально розроблена нами для даної задачі, працює наступним чином. Розбору піддається фраза подана у тегу <title> для кожного з <paragraph>. Текст фрази розбивається на окремі слова, з яких формується список, програмних об’єктів. Проводиться частиномовний аналіз і виділяються іменні групи. Об’єкти, що представляють слова та іменні групи. Ці об’єкти також містять морфологічні характеристики слів згідно їх ролі у даному реченні. Згідно з приналежності до частини мови, сутності розкладаються у списки об’єктів. Ці програмні об’єкти мають вбудовані методи для їх серіалізації в OWL класи у форматі RDF/XML.

Іменні групи стають підкласами редукованих її варіантів (що описують менш конкретне поняття) і так до головного слова іменної групи, що стає безпосереднім нащадком класу Object. Новий об’єкт поміщається до відповідного списку, призначеного для серіалізації в OWL класи, тільки у разі відсутності аналогічного.

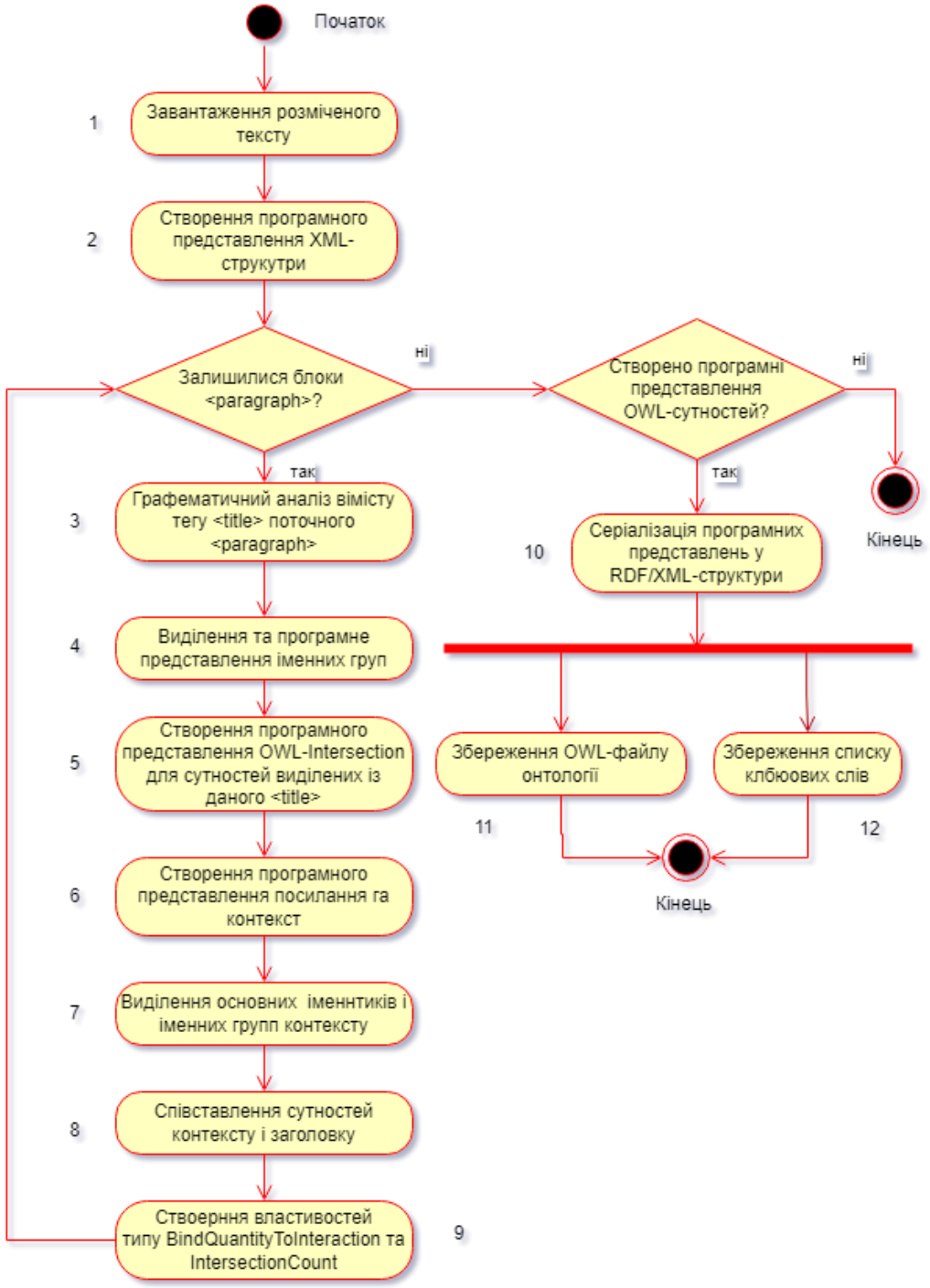


Рисунок 2.4 – UML-діаграма дій створення OWL-онтології на базі розміченого тексту

Для кожного тексту з тегу <title> створюється об’єкт типу перетинання (Intersection), що є нащадком класу верхнього рівня ConditionIntersection. Ім’я такого класу перетинання складається з початкових форм слів, що входять до

нього і приписки «_Intersection». До класу перетинання включається колекція об’єктів всіх значущих слів з тексту, розміщеного в <title>.

Створюється посилання на повний текст (нащадок класу верхнього рівня ReadyAnswerLink) і записується в окремий список.

Також на цьому етапі цьому створюються програмні об’єкти, що підлягають подальшій серіалізації в OWL властивості. В поле domain таких об’єктів записується створений клас типу Intersection, що містить колекцію, класів відповідних поняттям, які входять у текст даного тегу <title>. В поле range записується сформоване посилання на повний текст. Далі проводиться аналіз повного тексту відповіді (фактично, він проводиться один раз для кожного <paragraph>). З нього, на відміну від заголовку, виділяються лише іменники і іменні групи. З них відбираються тільки ті, які входять до даного тексту більш ніж один раз. З цих в свою чергу відокремлюються тільки ті, що були до того виділені у відповідному даному тексту заголовку. За наявності списку подібних понять, для кожного з них створюються об’єкт типу Count, що відповідає кратності входження даного поняття у повний текст. Цей об’єкт поміщається у відповідний список, що підлягає подальшій серіалізії, але лише в тому разі, якщо об’єктів, що відповідають подібній кратності у даному списку немає. Також створюються об’єкт типу Quantity – прив’язка кратності до конкретного поняття. Він також включається до відповідного списку подальшої серіалізації на умовах унікального входження (фактично, у програмі використовуються не списки, а множини об’єктів, а об’єкти відповідного типу мають прописані функції хешування, що є необхідною умовою для включення їх у контейнерні класи типу множин). Також створюються об’єкти, що серіалізуються в OWL властивості, нащадки властивостей верхнього рівня BindQuantityToInteraction та IntersectionCount. Перше приєднує відповідний об’єкт типу Quantity до посилання на текст відповіді, до якого приєднано через об’єкт-нащадок MainProperty об’єкт типу ConditionIntersection, який містить колекцію понять з заголовку відповіді. Друге приєднує до даного об’єкту типу Quantity відповідний йому об’єкт

типу Count. Далі відбувається безпосереднє формування структури OWL-файлу. Спочатку створюються заголовки і оговорені класи і властивості верхнього рівня. Далі відбувається серіалізація в відповідний тип OWL формату об’єктів зі списків, призначених для серіалізації. Методи серіалізації прописані у самих класах. Кожен OWL фрагмент, отриманий при OWL серіалізації програмного об’єкта додається до створюваного OWL-файлу. На сам кінець, до файлу приписуються кінцеві теги та файл зберігається під зазначеним іменем (за замовчуванням ontology.owl). Всі поняття з заголовка включаються до списку ключових слів для даної онтології на умовах унікальності, щоб уникнути дублювання. Ключові слова записуються у вигляді окремого файлу. Наведений опис є дещо скороченим і узагальненим, він не розкриває багатьох нюансів реалізації і подається для створення загального уявлення про сутність процесу.

2.3.2 Семантично структурована онтологія, що будується на текстах з регулярною структурою

У ряді випадків текст (або набір текстів) має регулярну наперед відому структуру. Тобто у певних його місцях гарантовано, чи з великою імовірністю містіть інформацію певного типу, наведену у визначеному вигляді. Таким чином можна створити програмну інструкцію, за якою машина зможе збудувати базу онтологічного типу з набору подібних текстів.

Конкретними прикладами такого вхідного матеріалу з нашої практики є, наприклад, збірка листів або набір PDF-файлів статей EBSCO з питань медицини та медичної реабілітації.

Приклад структури онтології листів

Онтологія містить наступні основні класи:

Date – дати листів;

LetterType – типи листів за характером змісту. Має класи-нащадки, що відповідають наступним типам:

Apology –вибачення;

Attachment –вкладення у посилки;

Congratulation – поздоровлення;

Invitation – запрошення;

Letter – звичайний, традиційний листопад;

Narration – розповідь;

Telegram – телеграма;

WithPoems – містить вірші

Name – імена;

Patronymic – по батькові;

Surname – прізвища;

Person – автори листів;

Place – місця (зазвичай, міста), з яких відправлялися листи;

Year – коки, за які є відправлені листи.

TextLink – посилання на повні тексти листів.

Властивості, що входять до онтології наступні:

Authorship – авторство, нащадки цієї властивості пов’язують авторів з написаними ними листами. У блок domain кожного нащадка цієї властивості входить один автор, у блок range входить список посилань на всі його листи.

FullName – нащадки цієї властивості поєднують імена, по батькові і прізвища (чи інші складові повного імені, наявні для конкретного випадку) з відповідним класом, що представляє автора (нащадок класу Person). Нащадок Person входить у блок domain, складові імені – у блок range. Такий підхід потрібний через те, що в своєму запиті, користувач не завжди вказує повне ім’я автора листа, а може вказати тільки ім’я і прізвище, або тільки прізвище, чи навіть ім’я і по батькові.

LettersTypes – прив’язка листів до їх типів. Ця властивість мА набір нащадків, обмежений відповідними класами, нащадками LetterType. Кожна така нащадкова властивість містить у своєму блоці domain відповідний клас-нащадок LetterType. У блоці range розташовано список посилань на тексти відповідних листів.

Locations – нащадки цієї властивості поєднують посилання на листи і місці їх відправлення. Місце відправлення (нащадок класу Place) стоїть у блоці domain, а у блоці range розміщено список листів, що були відправлені з даного місця.

SendingDate – нащадки цієї властивості поєднують посилання на листи з датами цих листів. Посилання на лист стоїть у domain, а дата відправлення (написання) у range.

YearOfDate – нащадки цієї властивості слугують для прив'язки років (підкласи Year) до дат (підкласи Date). Кожен рік входить у domain окремої властивості, а список відповідних йому дат – у range.

Схема основних класів запропонованої онтології листів наведена на рисунку 2.5.

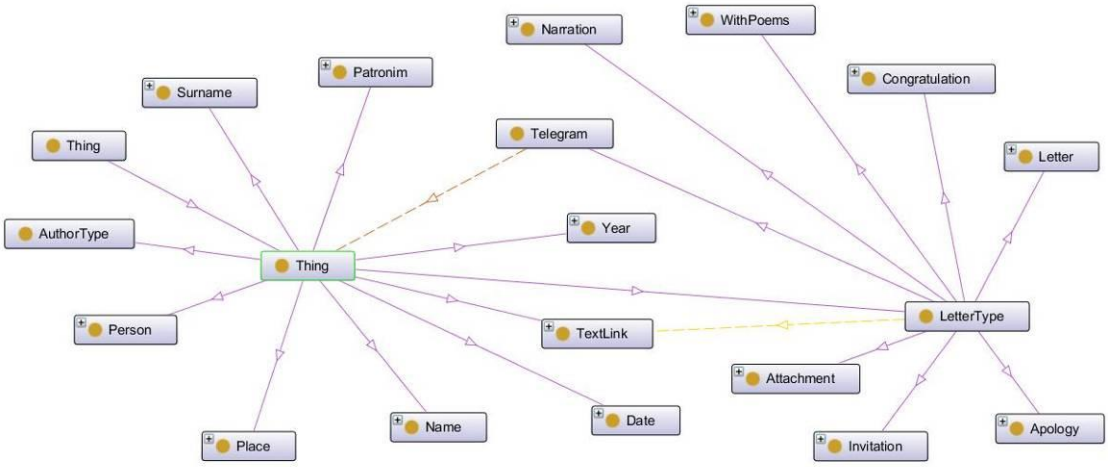


Рисунок 2.5 – Структурна схема базових класів онтології листів

Аналіз запропонованого прикладу семантично структурованої онтології на базі текстів з регулярною структурою

Як бачимо, така онтологія є більш семантично структурованою ніж описана у попередньому розділі 2.3.1. Для роботи вона потребує набору SPARQL-запитів. Кожен такий запит направлений на отримання певного роду понять, виходячи з заданих вихідних даних. Визначення потрібного шаблону запиту і виділення сутностей для підстановки в нього потребує семантичного аналізу фрази користувача. Очікується, що контекстна

семантично структурована онтологія сприятиме зменшенню «засмічення» результатів слабо релевантними даними, що спостерігалося при роботі із онтологіями структури, описаної у розділі 2.3.1. Тому, можливо, необхідність обчислення метрик релевантності і ранжування результатів при їх функціонуванні у якості бази знань довідкової системи значно знизиться або взагалі відпаде. Також при роботі із ними виключається, або, принаймні, значно скорочується процес багатократних повторних запитів з сутностями редукованої фрази. Все це здатне значно пришвидшити роботу такої діалогової чи довідкової системи.

Таким чином, не дивлячись на те, що онтологія описаного у даному підрозділі типу потребує більшої кількості шаблонів запитів і є більш вибагливою до семантичного аналізу тексту вона дозволяє отримувати більш надійні результати різних типів, ґрунтуючись на відношеннях між сутностями, які входять у її структуру.

Автоматизація створення онтології на базі текстів із регулярною структурою

Первинна обробка. Попередньо вхідний текст, що повинен бути представлений у вигляді текстового файлу. Проводиться попереднє очищення файлу від «текстового сміття», я то: номерів сторінок, розірваних переносів, поміток посилань, заголовків підрозділів (наприклад, «листи за певний рік»), колонтитулів. Названі процеси також в деякій мірі автоматизуються з використанням засобів текстового редактора, або доволі простого скрипта-замінника певних патернів у тексті.

Автоматизоване розмічення тексту

Принцип автоматизованої розмітки тексту. Процес створення розміченого тексту в даному випадку також є автоматизованим. Програмна реалізація процесу розмічення тексту сильно зав’язана на особливостях структури аналізованих текстів, тому застосування такого підходу доцільно у разі, коли загальний обсяг текстів достатньо великий. Наприклад деякі розділи відокремлені один від одного рядком з трьома зірочками через

пробіл («* * *»). Це буде вважатися маркером для розбиття на теги з <paragraph>. Вони, в свою чергу, можуть містити інші вкладені теги, при чому наявність деяких із них може бути опійною або передбачається можливість відсутності тексту у деяких тегах. Наприклад, тег <title> відповідає заголовку, але деякі із текстів у аналізованому наборі можуть і не мати заголовків.

Приклад автоматизації розмічення тексту. При розмітці збірки листів можуть бути наявними такі теги у контейнері <paragraph>:

<link> – посилання на текст листа для онтології;

<date> – дата написання (відправки) листа;

<place> – місце написання листа;

<author> – автор листа.

Місце у документі і характерні оточуючі слова є маркерами при автоматизованій розмітці. Так, наприклад, документи починаються рядком з повним іменем автора, що йому передує слово «Від». Таким чином, відкинувши це «від» і поставив те що далі слідує у початкову форму, отримується інформація для тегу <author>. Далі, наприклад, ми знаємо, що дати у документах представлені у вигляді: число (цифрами), назва місяця (словами), рік (цифрами) і скорочення «р.» вкінці без розділових знаків між частинами. Проте, іноді, у ряді документів їх вхідного набору, може бути відсутнім число, чи навіть місяць. У деяких випадках дається інтервал чисел через дефіс. Маркером для визначення географічної локації слугують такі слова, як «місто», «город», «село», «селище», «м.», «г.», «с.» у коротких рядках до або після основного тексту документа. Слово, що йде після них є назвою населеного пункту у складі адреси. Здача автоматизованого визначення ознак географічної локалізації може бут спрощена за рахунок створення списків міст, селищ та інших географічних назв. Їх наявність у характерних ділянках документу, також визначає зміст тегу <place>. Процес автоматичного розмічення текстів навіть великого набору документів займає декілька секунд, аналогічне ручне розмічення могло би зайняти багато годин.

Формування OWL-онтології на основі розміченого тексту. Розмічений текст передається на вхід програмі для формування онтології, яка створює на його базі OWL файл. Базові принципи створення OWL файлу схожі с попередньо описаними у пункті 2.3.1.5: створюються списки (точніше множини) програмних об’єктів, які потім серіалізуються у відповідні OWL компоненти – класи і властивості.

Схема створення даних програмних об’єктів у цьому випадку є дещо іншою і навіть простішою. Наведемо ряд прикладів, характерних для онтології листів:

1) Тег <author>. Інформація з тегів <author> розбивається на частини, з неї за характерними ознаками виділяються прізвища і по батькові. Інші частини (розділені пробілами) трактуються як імена. З них створюються відповідні об’єкти. Повний зміст тегу <author> стає об’єктом Person.

2) Тег <link>. До вмісту тегів <link> додається рядок “_TextLink”, а на їх базі створюються об’єкти, що містять посилання на основні тексти.

3) Дати і роки. З дат формується об’єкти типу Date, до назви яких спереду додається “Date_” – назви класів в OWL в деяких інтерпретаціях не повинні починатися з цифри. До поля label відповідного об’єкта дата записується як є. Для створення об’єктів типу Year (роки) зі строки дати, розбитої по пробілах береться останній компонент, що є числом. До назви об’єкта року спереду додається “Year_” у label рік записується цифрами у строковому форматі. Вміст тегу <place> береться напряму.

4) Типізація контекстів. Один з найочевидніших і найпростіших способів визначення типу повного контексту за змістом відбувається, базуючись на входження до нього наборів характерних слів, або за текстуальними структурами. Так, наприклад, вірші у складі текстів ідентифікуються як послідовність коротких рядків з близькою (та не обов’язково рівною) кількістю знаків. Для того, щоб бути ідентифіковані, як розповідні, контексти повинні мати досить великий обсяг і містити високу концентрацію дієслів в особових формах третьої або першої особи

теперішнього або минулого часу. Можливо також залучення методів машинного навчання і використання моделей-трансформарів, призначених для класифікації текстів – визначення настроїв, «тональності», позитивного/негативного/нейтрального відношення, схвальності або критики та інших критеріїв класифікації, передбачених у подібних моделях.

5) Поєднання створених OWL-сутностей. Так як всі характерні ознаки фрагменту тексту (документу) поєднані у одному контейнері, наприклад <paragraph>, неважко створити відповідні об'єкти властивостей, що поєднують об'єкти OWL класів між собою. Далі списки об'єктів OWL класів і властивостей серіалізуються і зберігаються у вигляді файлу.

Заповнення бази повних текстів також відбувається автоматизовано. Це може бути орієнтованої на документи бази даних (така як MongoDB), або реляційна база даних (PostgreSQL, MySQL, Oracle) або просто набір текстових файлів із визначеними іменами і розташуванням. В будь-якому варіанті – кожен документ у колекції повинен мати заголовок, що відповідає посиланню на нього у OWL-онтології.

Приклад автоматизованої побудови OWL-онтології з медичної реабілітації на базі файлів статей EBSCO

Вхідні дані. Іншим прикладом автоматизованого підходу до побудови онтології на базі набору текстів із заданою регулярною структурою стало створення бази знань з питань медичної реабілітації на базі набору наукових статей відповідного напрямку, по своїй структури близьких до медичних протоколів.

База знань створювалася автоматично на базі набору даних статей EBSCO, присвячених медичній реабілітації, що складається з 1013 PDF-файлів загальним обсягом 192 МБ. Усі статті англійською мовою. Основною концепцією автоматизованого створення бази знань є узгоджена та попередньо визначена структура файлів, яка слугує своєрідною інструкцією для програмної системи.

Реалізація створення OWL-онтології. Для реалізації створення бази знань у формі OWL онтології у форматі RDF/XML були розроблені спеціальні скрипти на мові Python. Процес складається з двох етапів.

1) Автоматизоване створення JSON представлення вхідних файлів статей. На першому етапі програма витягує та аналізує текст із файлів PDF, автоматично ідентифікуючи та структуруючи розділи та теми в кожному файлі як JSON структури визначеного вигляду. Таким чином, створюється набір файлів JSON, причому кожен файл відповідає вихідному PDF файлу і представляє його структурований вміст. Структура JSON шаблону представлення змісту файлу наведена у ***Додатку А***.

Фрази, словосполучення і поняття, що є ключами наведеного словнику, відшукуються у заголовках різних рівнів у текстах. Рівень заготовку для пошуку визначаться рівнем вкладення. У текстах заголовки різних рівнів відрізняються форматом тексту і характерними символами, як то: "›", "•", "–". Приймаються до увага також і маркери закінчення, такі як пропущений рядок, комбінація крапки або крапки з комою і переходу рядка, тощо.

Слід зазначити, що у аналізованих файлах ключові значення не завжди зустрічаються саме у такому вигляді, як подані у шаблонній структурі. Тому було створено словник синонімів, в якому вказані базові значення ключів словнику із зазначенням можливої їх варіативності. Наприклад:

```
"causes & risk factors": [
     "causes pathogenesis & risk factors",
     "causes pathophysiology & risk factors",
     "causes and risk factors",
     "causes pathophysiology and risk factors"
]
```

Для кожного розібраного PDF-файлу створюється відповідний структурований JSON файл вказаного вигляду.

2) Формування OWL-онтології. На другому етапі з використанням отриманого набору структур JSON створюється OWL-онтологія. Ієрархічна структура ключів словника JSON формує основу майбутньої системи класів OWL, тоді як відповідні контекстні значення стають іменованими сутностями у своїх відповідних класах. Кожне ім’я файлу статті

перетворюється на іменовану сутність у класі «Articles». Властивість OWL «Зв’язати зі статтею» встановлює зв’язки між контекстами та відповідними статтями, у яких вони з’являються. Іменовані сутності, визначені в контекстах, також перетворюються на іменовані сутності в класі «Word» і пов’язуються з відповідними контекстами за допомогою OWL властивості «Зв’язати з контекстом». Ця структура дозволяє вибирати певні контексти в онтології за допомогою SPARQL запитів.

Базова схема класів онтології, отриманої з набору медичних статей описаним методом наведено на рисунку 2.6. Через великий розмір і складність на даній ілюстрації цю онтологію наведено лише частково.

Через значний розмір отриманої бази знань для практичного використання її було розділено на 10 розділів, до яких можна робити запити паралельно.

Загальні принципи роботи з онтологією зазначеного вигляду. Робота з онтологією може здійснюватися наступним чином. Наприклад, лікар описує симптоми, що спостерігаються. Також вказує явно який тип інформації він хоче отримати, як то: передбачувані діагнози, методи діагностики цих діагнозів, методи лікування, тощо. У введеному опису визначаються іменовані сутності, за ними відбираються контексти з певного розділу, наприклад, описи симптомів. Далі, за властивістю "relate to article" витягується відповідна стаття або набір статей і пов'язані з ними (статтями) пункти, що шукаються (Individuals за класами типу представленої інформації). Ці тексти з Individuals (записані, наприклад, у їх “comment” чи “label”) повертаються користувачеві, або йдуть на подальший аналіз за допомогою великої мовної моделі.

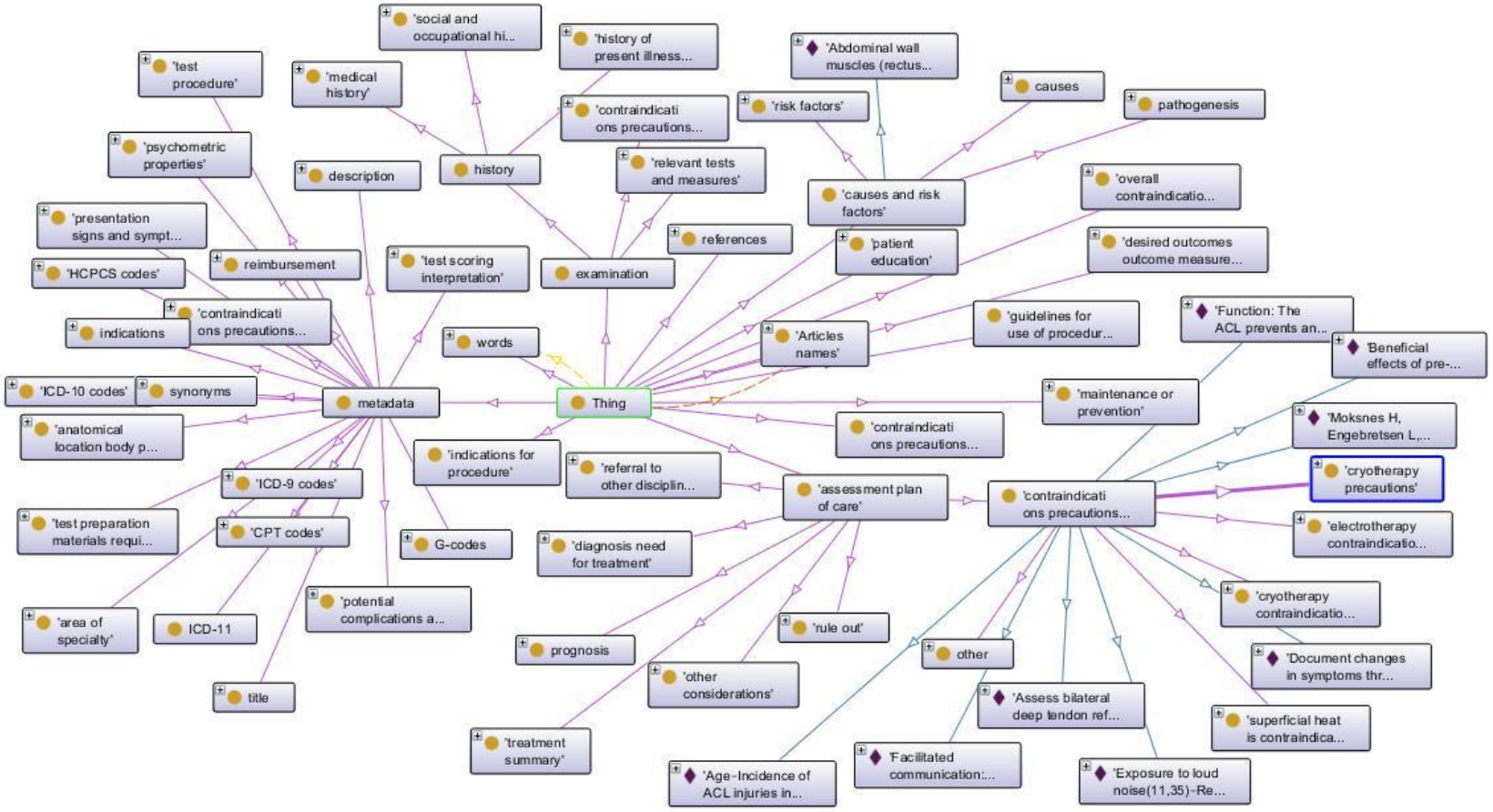


Рисунок 2.6 – Схема (скорочена) онтології за матеріалами статей по медичній реабілітації

2.3.3 Повністю автоматизована побудова онтології на основі синтактико-семантичного аналізу природномовного тексту на мові флективного типу

Ідея повністю автоматизованої побудови онтології на базі тексту. В основу запропонованого підходу покладено правила синтаксисно-семантичного аналізу. Відомо, що в мовах флективного типу основну роль у зв'язуванні слів у реченні відіграє поєднання певних флексій (змінних закінчень слів). Розвинена мовна система різноманітних комбінацій флексій для різних частин мови дозволяє висловити значну семантичну інформацію. Таким чином, аналіз наявних комбінацій форм слів дозволяє отримати хоч і не всю, але значну частину семантичної інформації. У розглянутому підході нами запропоновано близько 90 семантичних типів, кожен з яких фактично може мати кілька (до більше сотні) підтипів, що залежить від таких додаткових характеристик, як рід, час, число або певний прийменник для кожного зі слів із розглянутої пари. Ці додаткові підтипи на даний момент не використовуються при створенні онтології, але вони можуть бути корисними в майбутньому для ще більш глибокої та точної структуризації інформації. Крім того, вони надходять безпосередньо з так званих «кореляторів», які фактично є частиною аналізатора, що надає програмі інструкції щодо варіантів, яким чином може бути виражений той чи інший семантичний тип.

Слід зазначити, що на даний час існують різні системи семантичних типів. Набір, який тут розглядається, може бути отриманий з набору аксіоматичного базису методом, запропонованим у розділі 2.2.

Програмна реалізація створення онтології на базі тексту

Синтактико-семантичний аналіз тексту флективною мовою. Перша і, мабуть, основна і найбільш навантажлива частина полягає в синтаксисно-семантичному аналізі вхідного тексту. Відповідний програмний модуль включає створений вручну набір так званих «кореляторів» і «детермінантів». «Детермінанти» являють собою комбінації можливих відмінків і прийменників між ними (якщо такі є) і відповідних кожному варіанту

семантичних підтипів. Також вказано, чи є сполучення зворотним і яке зі слів пари вважати головним. Назви семантичних підтипів подаються у символьному вигляді, наприклад K1001, K4801, K6201 тощо. Ось приклад рядка з файлу з «визначниками» для української мови:

```
ім під ям L    I    K6201K8644K8646
```

тут ми бачимо флексії для першого та другого слів, якими є «ім» та «ям». Передбачається, що між ними стоїть прийменник «під». Символ «L» позначає, що головне слово ліве, символ «I» говорять про те, що посилання зворотне. Можливі семантичні підтипи для даної комбінації позначаються як K6201, K8644, K8646.

«Корелятори» являють собою відповідність кожного із семантичних підтипів можливим варіантам сполучень частин мови, в тому числі їх порядку в парі. Також у файлі наводяться назви семантичних типів («макротипів»). Для кожного з цих «макротипів» може бути кілька (до більше сотні) відповідних підтипів. Ось приклад рядка з файлу з «кореляторами» для української мови:

```
K3506 відємність_дії S4S1;S4S6;S4S13;S4S5;S4S3;S4S10;S4S11;S4S12;S4S18;S4S22;S4S25;S4S28
```

Тут ми маємо «K3506», що є назвою семантичного підтипу. Далі йде “відємність_дії” (англ. “separability of an action”), яка є вербальною назвою відповідного семантичного типу. Потім йде послідовність можливих пар частин мови, поданих як пари відповідних символів. Наприклад, S1 — «іменник», S4 — «дієслово». Пари розділяються крапкою з комою.

Також у програмі є словник, до якого входять основи слів, леми та флексії. У словнику подано відповідність між основами, лемами та наборами флексій. Словники зберігаються в спеціальному компактному форматі та можуть бути створені автоматично за допомогою відкритих мовних даних.

Основна мета програми полягає в тому, щоб за допомогою цих даних і вихідного тексту знаходити і типізувати слова в ньому, визначати в них основи і флексії, розпізнавати зв'язки між словами і з'ясовувати їх семантичні типи.

Іншим результатом такого аналізу є отримання груп зв'язаних слів у реченнях. Формально група є повнозв'язним графом. Практично такі групи можуть відповідати простому реченню, частині складного речення або дієприкметниковому звороту.

Проміжні файли результатів роботи семантичного аналізатора. Результатом роботи цього модуля є два XML-файли allterms.xml і parce.xml. Вони використовуються для подальшого створення OWL-онтології.

1) Опис файлу allterms.xml. Файл allterms.xml — представляє список термінів — іменників і іменних груп, які містяться в аналізованому тексті, і їх основні характеристики. Він складається з двох основних частин: <exportterms> і <sentences>. Перший включає терміни. Ось приклад представлення терміна:

```
<term>
     <ttype>Noun_noun</ttype>
     <tname>тіло людини</tname>
     <wcount>2</wcount>
     <osn>тіл</osn>
     <osn>люд</osn>
     <sentpos>1/1</sentpos>
     <sentpos>1/2</sentpos>
     <reldown>2</reldown>
     <reldown>4</reldown>
</term>
```

Тег <ttype> представляє послідовність частин мови, яка утворює термін. Тег <tname> – це текст терміна, як його наведено у тексті. Тег <wcount> визначає кількість слів у терміні. Теги <osn> подано для кожного слова, вони представляють основи слів. Теги <sentpos> задають позиції слів терміну в тексті у флрматі *номер речення – (від 0)* / *позиція слова в реченні – (від 1)*. Теги <reldown> і <relup> необов'язкові. Вони показують зв'язок розглянутого терміна з іншими термінами з файлу. Тег <reldown> вказує на термін звуження контексту – кожне слово якого можна знайти в цьому терміні. Тег <relup> вказує на термін розширення контексту – містить усі

слова з цього терміну та деякі інші. Теги <reldown> і <relup> допомагають побудувати ієрархію термінів у створюваній онтології. Частина <sentences> містить лише тексти всіх речень із розглянутого тексту в тегу <sent> кожна.

2) Опис файлу parce.xml. Файл parce.xml представляє синтаксисно-семантичну схему кожного речення тексту. Структури речень подано в контейнерних тегах <sentence>. Цей контейнер містить такі теги: кілька тегів <item>, що представляють слова та їхні характеристики; <sent_pos> – номер речення в тексті (від 1); <sent> – текст речення. Ось приклад тегу <item>:

```
<item>
        <word>Книга</word>
        <osnova>кни</osnova>
        <lemma>книга</lemma>
        <kflex>a</kflex>
        <flex>га</flex>
        <number>1</number>
        <pos>1</pos>
        <group_n>1</group_n>
        <speech>S1</speech>
        <relate>0</relate>
        <rel_type>K0</rel_type>
</item>
```

Тег <word> містить текст слова, як наведено в розглянутому тексті. Тег <osnova> представляє основу слова. Тег <lemma> надає лему – початкову форму слова. Теги <kflex> і <flex> є флексіями. <kflex> – це безпосередньо граматичне закінчення, а <flex> – це саме змінна частина слова, включаючи випадки слів, що змінює основу. Тег <number> – це номер слова в реченні. Тег <group_n> показує приналежність слова до пов'язаної групи з речення. Тег <speech> містить позначку відповідної частини мови. Тег <relate> вказує номер слова, від якого йде семантичний зв'язок до розглянутого слова. Якщо слово не має жодних вхідних посилань, як у прикладі вище, вміст цього тегу

встановлюється на нуль. Тег <rel_type> – це тип (підтип) семантичного зв’язку. Значення K0 означає відсутність зв'язку або його невідомий тип.

Створення OWL-онтології на базі результатів аналізу тексту. Отримані два файли використовуються для створення OWL-онтології. Методика створення OWL-онтології у своїй основі близька до описаних вище у попередніх розділах: створюється програмне представлення класів, властивостей, іменованих сутностей – як кореневих (наперед заданих), так і отриманих на основі результатів розбору тексту. Далі із них формується зв’язана структура, яка потім серіалізується у форматі RDF/XML.

Маючи два згаданих файли (allterms.xml і parce.xml) та файл з «детермінантами», які містять вербальні імена типів семантичних зв’язків, можна сформувати OWL-онтологію. Усі сутності OWL спочатку створюються як ООП об’єкти відповідних класів Python. Кореневі, наперед задані класи та властивості створюються першими і є обов’язковими. Потім за допомогою файлу allterms.xml створюється ієрархічна структура термінів (іменників та іменних груп). Потім за допомогою parce.xml створюються класи інших типів, а також властивості, що відповідають зв’язкам між словами. При цьому створюються властивості типу WordsLink, які представляють семантичні типи. Оскільки у файлі parce.xml подано лише певний підтип, для визначення семантичного «макротипу» використовуються «визначники» з відповідного файлу.

Оскільки слова належать до пов’язаних груп, а групи – до речень, ця інформація використовується для створення відповідних нащадків властивостей Groups і SentenceGroups. Поле label таких властивостей-нащадків Groups і SentenceGroups містить текст відповідної групи або речення. Ці контексти можуть бути корисними для надання більш інформативних відповідей при запитах. Також їх використовували для порівняння із результатом у експериментах зворотного синтезу тексту на базі онтології.

Типізація груп зв'язаних слів здійснюється за наявністю певних слів у групі: підрядний союз, дієприкметник, дієприслівник. Тестування онтології показало її придатність для отримання різних типів інформації відповідно запиту, а також контексти, у яких містяться відповідні сутності пов’язані певними семантичними відношеннями. Таким чином, розроблена програмна система може бути корисним інструментом для автоматизації створення баз знань довідкових систем та агрегації даних в Інтернеті.

Структура OWL-онтології, побудованої автоматично на основі природномовного тексту. Розглянемо докладно структуру створюваного вищеописаним чином OWL-опису онтології. Схема основних класів такої онтології подана на рисунку 2.7.

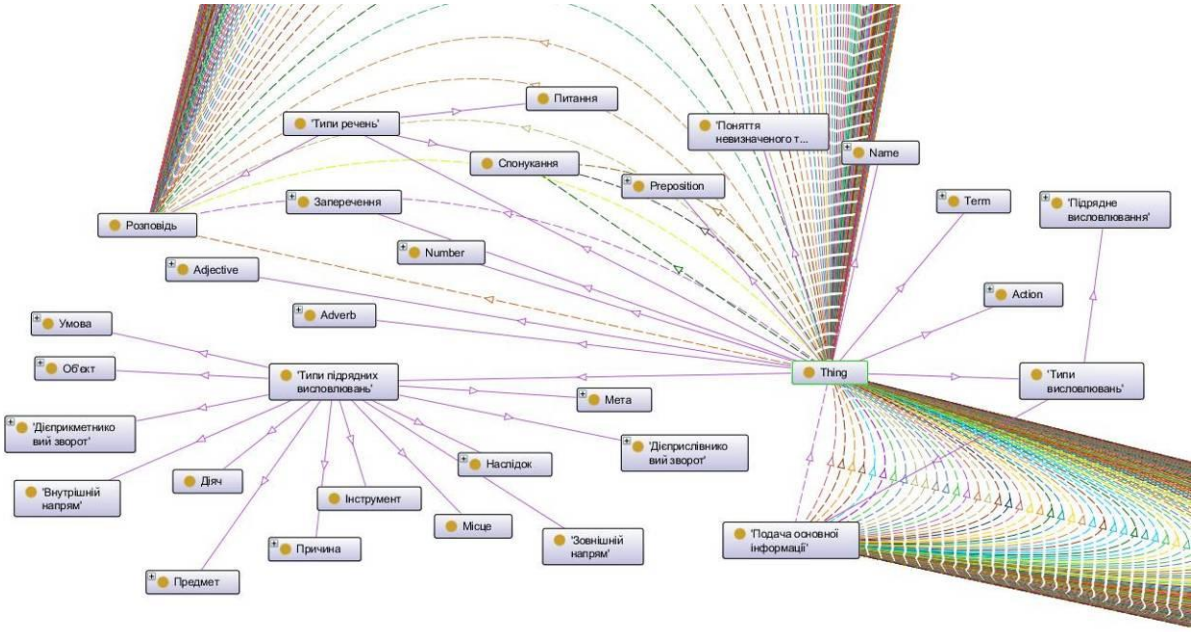


Рисунок 2.7 – Схема основних класів онтології, що формується автоматично із тексту

1) Базові класи

Онтологія має наступні базові класи:

- Action – дії, висловлені дієсловами;
- Adjective – прикметники та дієприкметники;
- Adverb – прислівники та дієприслівники;
- Name – власні назви;

- Number – числівники (в тому числі і невизначені – «кілька») та цифрові символи;

- Preposition – прийменники;

- Term – іменники та іменні групи. Має ієрархічну структуру від загальніших понять (з одного слова) до більш конкретизованим із двох і більше слів;

- Negation – заперечні частки;

- UndefinedEntities – зарезервований для понять з тексту, тип яких не вдалося віднести до жодного з вищезгаданих класів.

- PhraseType – типи зв'язаних груп. Має два дочірні класи: MainNarration (головне речення) та SubordinatePhrase (підрядне речення). Дані класи не мають спадкоємців, а застосовуються як “Domain” для властивостей, відповідальних за характеристику групи.

- SubordinatePhraseType – типізація підрядних речень. На даний момент має підкласи: movement_in (рух всередину), actor (діяч), Participial (дієприкметниковий зворот), AdverbialPhrase (дієприслівниковий зворот), movement_out – рух назовні, goal (ціль), place (місце), consequence (слідство), object (об'єкт), subject (предмет), cause (причина), condition (умова), інструмент (інструмент). Дані класи не мають спадкоємців, а використовуються як "Range" для властивостей, що відповідають за характеристику груп, чий "Domain" має значення SubordinatePhrase.

- SentTypes – типізація речень. Має підкласи: Narration (розповідь), Interrogative (питання) та Imperative (спонукання). Дані класи не мають спадкоємців, а застосовуються як “Range” для властивостей, відповідальних за характеристику речень.

2) Основні властивості

Властивості в онтології поділені на три великі (кореневі) групи:

- WordsLink – зв'язок між окремими поняттями;

- Groups – пов'язані групи слів;

- SentenceGroups – речення.

Властивість «WordsLink» має нащадків, які відповідають семантичним типам. Нащадки “WordsLink” мають ієрархічну структуру. Схема структури нащадків “WordsLink” представлена у вигляді дерева на рис. 2.8.

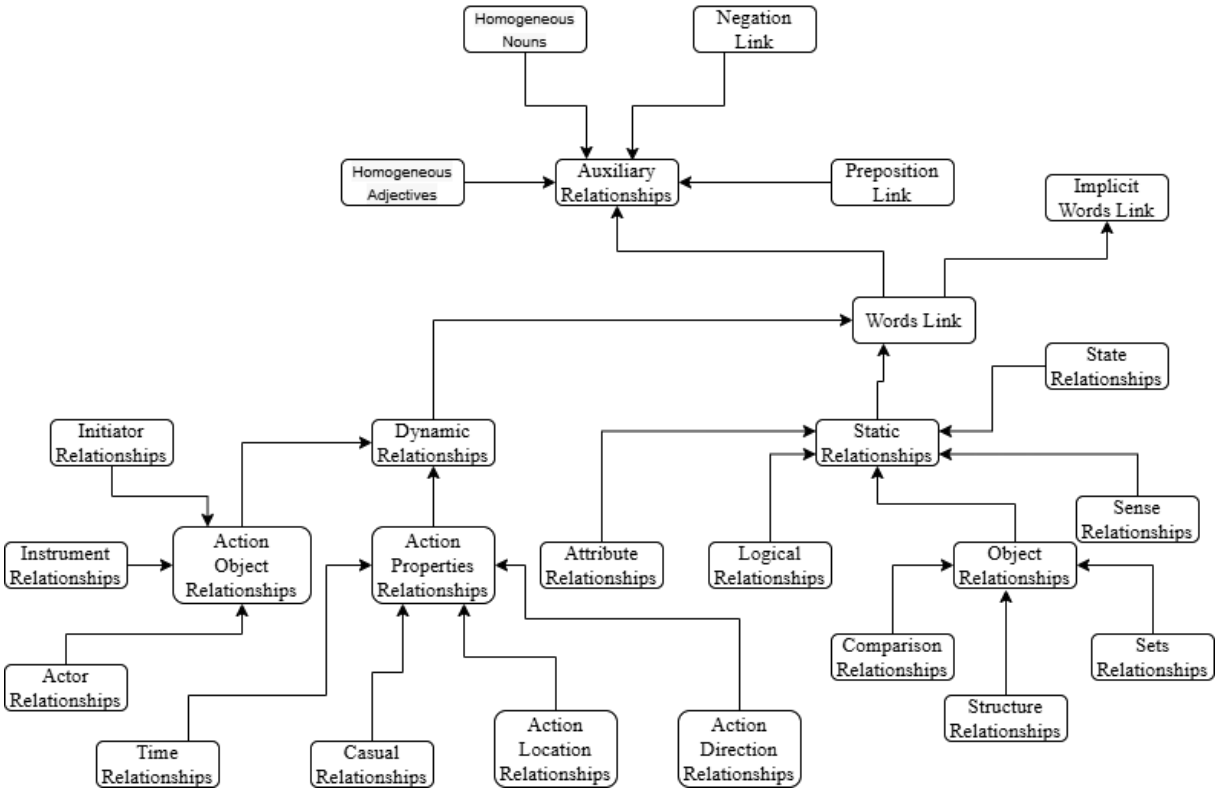


Рисунок 2.8 – Схема ієрархії нащадків властивості «WordsLink»

Наведена тут структура охоплює лише вищий рівень типізації ієрархії семантичних зв’язків, яка залишається незмінною для всіх побудованих онтологій. Наявність сутностей нижчого рівня залежить від їх присутності в розглянутому тексті. Вони також можуть мати ієрархічну структуру. На даний момент розроблена система оперує приблизно 80 – 90 можливими кінцевими семантичними категоріями, і це, очевидно, не межа. Кінцеві нащадки властивості «WordsLink» відповідають певним типам зв’язків між поняттями. Кожен з них зустрічається в онтології лише один раз, навіть якщо його можна зустріти кілька разів у розглянутому тексті. «DOMAIN» такої властивості відноситься до основного поняття зв’язаної пари, а «RANGE» — до залежного. Крім того, ці властивості також є спадкоємцями груп, де спостерігається пара понять, пов’язана таким чином.

Властивість Groups характеризує групи. Властивості спадкоємці Groups відповідають конкретним групам. Дочірніми властивостями конкретних груп є вищезгадані властивості, що показують типізовані зв'язки між конкретними поняттями, що входять до цієї групи.

Властивості SentenceGroups відповідають реченням. У якості параметру label вони містять повний текст речення (контекст). Їх спадкоємці – це властивості типу Groups, що відповідають групам даного речення.

Для роботи з онтологією описаного виду використовується СУБД Neo4J, куди завантажується файл OWL. При цьому і класи, і властивості стають відповідного типу вершинами такого графа (Class або Relationship, відповідно). Зв'язки між вершинами мають таку типізацію: SCO – subclass of; SPO – sub property of; DOMAIN; RANGE. Для запитів використовується мова Cypher.

2.4 Автоматизований семантичний аналіз окремих природномовних фраз з метою побудови формальних запитів до онтології

Як згадувалося на початку розділу, важливою складовою природномовних інтерфейсів баз даних і баз знань (в тому числі діалогових і експертних систем) є перехід від природномовних реплік користувача до формальних запитів, що виконуються СУБД. В нашій роботі було розроблено цілий ряд підходів до вирішення цієї задачі, кожен з яких має свої переваги і недоліки, а оптимальність застосування того чи іншого методу залежить від особливостей використаної у системі онтології. Далі будуть розглянуті як випадки автоматично створюваних онтологічних структур, що були описані у попередньому підрозділі, а також онтології створені вручну.

Всі розроблені у нашій роботі підходи об’єднує одна концепція, що включає наступні фази: обрання оптимальної онтології для запиту, наявність

одного чи декількох шаблонів формальних запитів, що відповідають різним семантичним категоріям, якщо у системі присутні більш одного шаблона формальних запитів, то, відповідно, існує і модуль для визначення найбільш релевантного типу шаблону або набору шаблонів.

2.4.1 Визначення найбільш відповідної онтології

У довідковій системі можуть бути наявними декілька онтологій. Таким чином, постає задача вибору найбільш оптимального для кожного випадку варіанта. У ряді випадків окрему онтологію можна вибрати лише за ознакою наявності у фразі користувача певних слів, або словосполучень. Наприклад, онтологія листів обирається при наявності у фразі користувача слів похідних слів від «лист», «відправляти», «надсилати». В інших випадках відбувається зіставлення понять з фрази користувача, обраних для формування SPARQL запиту з ключовими словами, пов’язаними з кожною з наявних онтологій. Оскільки набір сутностей, які виділяються з фрази користувача може піддаватися редукції, то він буде відрізнятися для кожного запиту зі створеного пакету. Для кожного запиту (тобто відповідного набору понять для нього) обирається окремо найбільш оптимальна онтологія. таким чином пакет запитів може виявитись адресованим до декількох онтологій. В такому випадку можливе паралельне виконання запитів.

Ступінь відповідності онтології набору сутностей, представлених до запиту обчислюється наступним чином (2.1):

$$\eta = \frac{N_{match}}{N_{query}} \tag{2.1}$$

де: N_{match} – кількість слів обраних для запиту, що співпали з поняттями зі списку ключових слів онтології; N_{query} – загальна кількість слів обраних для запиту

Оптимальною вважається та онтологія, для якої отримана таким чином метрика η буде максимальною.

Незважаючи на примітивність описаного підходу, він виявився доволі дієвим у системах, що містили до десятка онтології, навіть близьких за тематикою.

2.4.2 Створення запитів до онтології для отримання простих контекстуальних відповідей за уніфікованим шаблоном

2.4.2.1 Структура шаблону. У разі використання у системі автоматично створеної з набору розмічених текстів простої контекстуальної онтології, у якій контекстам поставлені у відповідність набори сутностей всі формальні запити формується по єдиному універсальному шаблону. Структура шаблонів запиту надається в окремому XML-файлі і може бути зміненою без втручання у код програми. Шаблон запитів до онтології вказаного типу наведено у ***Додатку Б***.

Дамо пояснення щодо структури шаблону.

Характеристика тегів:

<language> – визначає природну мову, для якої призначений цей шаблон.

<type> – тип шаблону за типом запиту.

<variables> – розділ, який містить список опису змінних, які входять до шаблона запиту.

<name> – ім’я змінної у шаблоні запиту.

<var_type> – тип даних, який може передати у дану змінну.

<destination> – роль змінної при формуванні запиту.

<allow_list> – вказівка, чи може бути у якості даної змінної передано список. При значенні true, у разі передачі списку у запиті буде сформовано ряд ідентичних зазначеному рядків, кожен з яких містить наступне з переданого списку значення даної змінної. В противному випадку, елементи списку будуть конкатоновані у єдиний рядок.

<query_base> – перший (базовий) рядок формального запиту.

<conditions> – список умов, що ставляться у розділ WHERE запиту.

<condition> – рядок, що визначає умову у розділі WHERE запиту. Змінні записуються у квадратних дужках. Текст, що не у квадратних дужках іде до запиту, що формується, у вигляді безпосередньо як подано у шаблоні.

<ordering> – (відсутній у наведеному прикладі) – умови впорядкування результатів запиту у розділі ORDER BY. Тег є необов'язковим.

2.4.2.2 Формування запиту за шаблоном. Формальний запит на мові SPARQL створюється за шаблоном, приклад якого наведено у попередньому розділі. Якщо у рядках шаблону містяться позначені у квадратних дужках змінні, зазначені у відповідному розділі шаблона то відбувається їх підстановка. Форма проведення підстановки залежить від призначення конкретної змінної, що вказано у тегу <destination> в її описі:

- якщо у <destination> вказано «input», то на місце змінної підставляється її значення, що було попередньо отримане з фрази користувача. Якщо то є список значень, рядок дублюється з підставленням у нього кожен раз нового значення зі списку;

- якщо у <destination> вказано «inner», то це внутрішня проміжна змінна, необхідна для зазначення потрібних зв'язків у запиту. На місце такої змінної при формуванні запиту ставиться «?*Ім'я_Змінної_Як_У_Шаблоні*» (імена змінних у SPARQL починаються зі знака «?»);

- значення «result» у <destination> означає, що ця змінна призначена для виводу результату. Вона замінюється аналогічно до внутрішніх проміжних змінних. Значення <destination> типу «result» відіграє скоріш семантичну роль при створенні шаблону запиту – що саме є запитуваним результатом.

Хоча вищеописаний шаблон запиту призначений для SPARQL, подібний підхід може бути використаним і для інших мов формальних запитів. Так нами також були створено подібний шаблон для мови запитів Cypher, що використовується СУБД Neo4j.

2.4.2.3 Процеси виконання і обробки результатів запиту. Слід також коротко зупинитися на особливостях обробки результатів подібного запиту, що включають також елементи природномовного аналізу.

Виконання запиту. Запити з надісланого пакету виконуються послідовно до відповідної, вказаної для кожного запиту онтології. Також виконуються допоміжні запити. Для сутностей, що були представлені до запиту виконується перевірка на входження їх у текст відповіді два і більше разів. Для цього слугують запити з використанням Quantity і Count. Якщо така інформація в онтології наявна, то до відповідей також додається число кратності відповідних понять, що збільшить її вігу при ранжуванні.

При отриманні порожньої відповіді на SPARQL запит, виконується новий запит з застосуванням редукованого списку сутностей. Такі запити виконуються для кожного наступного рівня редукції до отримання відповіді або до вичерпання списку рівнів редукції. На практиці, навіть у разі отримання відповіді, виконуються додаткові запити за наступними рівнями редукції списку вхідних сутностей, як змога отримати ще й список додаткових пов’язаних контекстів.

Спосіб ранжування отриманих відповідей за релевантністю. Ранжування відповідей здійснюється на підставі метрики, що враховує збіг і розбіжність основних слів з фрази користувача зі словами з заголовка відповіді.

Нами пропонується метрика, що обчислюється з наступних міркувань. Збіг іменників і іменних груп з фрази користувача з такими зі списку ключових слів або з заголовка відповіді є позитивним фактором, бо фігурують ті ж сутності. Наявність в ключових словах або заголовку відповіді додаткових, які не містяться у фразі користувача, іменників або іменних груп є фактор, скоріше, негативний. Це розширення додатковими сутностями, які можуть вносити інший зміст або бути не цікаві користувачеві.

Збіг дієслів з фрази користувача з такими зі списку ключових слів або з заголовка відповіді – також позитивний фактор, хоча він має меншу вагу, ніж збіг іменників і іменних груп. Наявність в ключових словах або заголовку відповіді додаткових, які не містяться у фразі користувача дієслів не слід

вважати негативним фактором. Навпаки, користувачеві, з практичної точки зору, може бути швидше цікаво, що можна робити з даними сутностями або які дії вони здійснюють, ніж їх визначення.

Наявність збігів в інших частинах мови, якщо такі потрапили в розбір з заголовка можна вважати позитивним фактором, хоча і не настільки вагомим. Відсутність збігів по іншим частинам мови передбачається або не враховувати або враховувати як слабкий негативний фактор.

Виходячи з викладених положень, запропоновано наступну формулу для оцінки релевантності готової відповіді по відношенню до фрази користувача:

$$f_{ngm} = \frac{N_{add.n.g.}}{N_{i.n.g} + 1} \tag{2.2}$$

$$f_{mng} = \frac{N_{c.n.g.}}{N_{i.w.} + 1} \tag{2.3}$$

$$f_{mv} = \frac{N_{c.v}}{N_{i.w.} + 1} \tag{2.4}$$

$$f_{vm} = \frac{N_{add.v}}{N_{i.w.} + 1} \tag{2.5}$$

$$f_{om} = \frac{N_{add.o.}}{N_{i.w.} + 1} \tag{2.6}$$

$$M = a_1 \cdot f_{nmg} + a_2 \cdot f_{mng} + a_3 \cdot f_{mv} + a_4 \cdot f_{vm} + a_5 \cdot f_{om} \tag{2.7}$$

де: f_{ngm} - фактор відносної кількості надлишкових іменників і іменних груп; $N_{add.n.g.}$ - кількість додаткових, відсутніх в фразі користувача іменників і іменних груп; $N_{i.n.g}$ - загальна кількість іменників і іменних груп у фразі користувача; f_{mng} - фактор відносної кількості збігів іменників і іменних груп; $N_{c.n.g.}$ - кількість співпадаючих іменників і іменних груп; $N_{i.w.}$ - загальна кількість слів у фразі користувача; f_{mv} - фактор відносної кількості співпадаючих дієслів; $N_{c.v}$ - кількість співпадаючих дієслів; f_{vm} - фактор відносної кількості надлишкових дієслів; $N_{add.v}$ - кількість додаткових,

відсутніх в фразі користувача дієслів; $f_{от}$ - фактор відносної кількості співпадаючих інших частин мови; $N_{add.o.}$ - кількість співпадаючих інших частин мови; M – метрика релевантності відповіді; a_1 ... a_5 – чисельні коефіцієнти. Їх значення можна охарактеризувати їх значення в такий спосіб: значення a_1 негативне, так як наявність додаткових сутностей знижує релевантність і досить велика. Значення a_2 позитивне і найбільше – збіг іменників і іменних груп є найважливішим фактором. Значення a_3 позитивне, але менше, ніж a_2. Значення a_4 позитивне, але не велике. Значення a_5 негативне і мале.

У першому наближенні нами були прийняті наступні значення цих коефіцієнтів: a_1 = -1; a_2 = 2; a_3 = 1; a_4 = 0,5; a_5 = -0,5.

Найбільш релевантним буде вважатися відповідь, для якої значення вищевказаної метрики буде максимальним.

Для результатів запиту, які направлені до онтологій інших типів зазвичай ранжування відповідей не потребується, тому як структура такої онтології і запитів до неї є більш чітко семантично визначеною. Проте у ряді випадків, особливо це стосується онтологій створених вручну, що містять багатослівні назви вершин, подібні підходи можуть бути застосовані для адаптації вихідних понять при підстановці їх до запиту. Більш докладно цей підхід описано у наступних підрозділах.

2.4.3 Визначення шаблону формального запиту шляхом семантичного аналізу тексту вихідної фрази з використанням дерева прийняття рішень

У разі наявності у системі набору шаблонів формальних запитів, що відповідають різним семантичним характеристикам запитуваної інформації постає задача обрання такого шаблону, або їх набору. Одним з підходів, що було запропоновано у даній роботі для вирішення вказаної задачі було застосування аналізу послідовності слів у вихідній фразі із використанням спеціально створеного дерева прийняття рішень. Такий підхід на практиці

було задіяно при роботі з автоматично створеною онтологією листів, а також для більшості створених вручну онтологій з різних предметних галузей на українській та англійській мовах, зокрема для онтології по «Білій книзі з фізичної та реабілітаційної медицини в Європі».

Нижче наведено загальний принцип проведення такого аналізу. Спочатку фраза користувача очищується від зайвих символів та слів, що не несуть визначального семантичного навантаження. Також можливе додаткове очищення фрази від понять, які не містяться в онтології та/або в умовах дерева прийняття рішень. Це значно пришвидшує процес аналізу і дозволяє уникнути завідома непродуктивних перевірок.

Очищена фраза перетворюється на список слів. Словоформи при цьому зберігаються, так як вони можуть мати визначальне семантичне значення. Далі ведеться послідовний аналіз слів і груп слів, які входять в отриманий список. Аналіз відбувається послідовно у декілька кроків. Інструкція щодо схеми і критеріїв такого аналізу подається в окремому XML-файлі у вигляді своєрідного дерева прийняття рішення. Слова із списку, отриманого з фрази користувача зіставляються з наборами слів з умов відповідної позиції даного дерева. Співставленню можуть підлягати як окремі слова, так і їх послідовні групи. Умовами перевірки може бути не тільки текстовий збіг слів, а й відповідність слова з фрази користувача зазначеному критерію, наприклад – будь-який іменник, послідовність іменників, не іменник і не прикметник, число, тощо. Залежно від обраного критерію, що підходить для даного сова або групи слів відбувається перехід на вказану позицію у дереві прийняття рішення рівнем нижче. Якщо не один з зазначених у позиції критеріїв не виявився задовільним, результатом стає – «тип шаблону невизначено». Запит в такому разі створено не буде. Якщо на даній позиції вже залишився тільки один варіант шаблону, все одно можливі додаткові перевірки на більш низьких рінях дерева для визначення відповідності не тільки необхідним але й достатнім умовам.

Одночасно відбувається і обрання сутностей для запиту. Для цього по мірі проходження дерева прийняття рішення записується шлях проходження відповідних позицій у даному дереві і слів або їх груп, що задовольнили певному критерію у рамках цієї позиції. Коди процес аналізу доходить до кінцевої позиції, то у ній вказано, сутності з яких позицій у які змінні запиту потрібно підставити.

Загальна схема запропонованого алгоритму формування запиту від природномовного тексту до формального запиту проілюстрована діаграмою активності UML на рисунку 2.9.

Розміщення шаблонів запитів і схеми дерева вибору шаблонів в окремі файли дозволяє розробнику адаптувати систему до певної онтології, не торкаючись логіки програмного коду. Програмний код міститься у файлі Python, до якого прив’язаний лінгвістичний аналізатор для роботи із певною природною мовою.

Інформація на вході — це фраза користувача природною мовою, а на виході — формальний запит на SPARQL або Cypher. При реалізації модуль включено до оболонки програмного агента (веб-служби), тому він може легко стати частиною діалогової або довідкової природномовної системи.

2.4.4 Спосіб визначення семантичного типу висловлювання через набір перевірок вихідного тексту

Ідея визначення семантичного типу виразу через набір перевірок. Важливим недоліком способу визначення шаблону запиту з використанням дерева та аналізу послідовності слів, запропоновано нами у роботі [97] і описаного у попередньому розділі, є те, що він вимагає значних зусиль і часу для побудови самого такого дерева. Він добре себе зарекомендував у системах, що мають відносно невелику кількість шаблонів та спеціалізованих на вузькій предметній області. Також він є більш прийнятним для природних мов з досить чітким порядком слів у реченні (наприклад, англійської мови), де він успішно застосовувався на практиці.

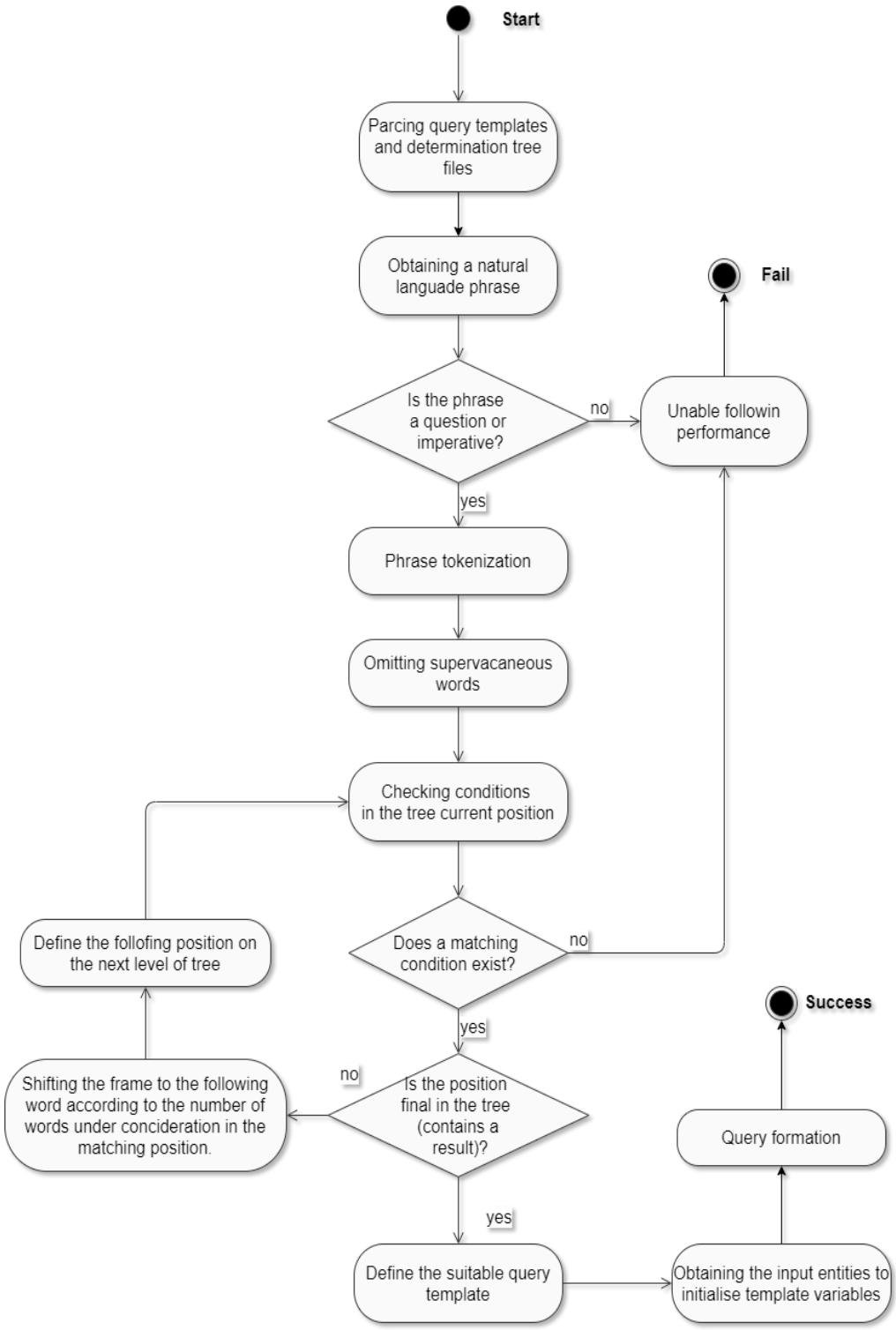


Рисунок 2.9 – UML-діаграма активності процесу перетворення фрази на природній мові у формальний запит

В той же час для флективних мов (наприклад, української) порядок слів є менш суттєвим фактором. Важливішою видиться сама наявність певних слів та їх словоформ. Тому часто достатнім виявляється просто перевірити

вхідну фразу користувача на відповідність кількох критеріям. На підставі результатів таких перевірок можна визначити як найбільш відповідний шаблон запиту (або групу шаблонів), так і виділити вхідні поняття для підстановки. Означений спосіб є чи не єдиним зручним у реалізації варіантом для онтології, створених шляхом цілковито автоматичного син тактико-семантичного аналізу тексту, описаного вище.

Приклад простого набору перевірок для визначення семантичного типу висловлювання. У спрощеній версії реалізації такого способу, яку було розроблено і опробувань у рамках дослідження, є 4 основні перевірки:

1 – питальне слово – 6 списків + відсутність такого слова. Результат: номер списку від 1 до 6, або 0 – якщо немає питального слова

2 – наявність слова зі списків (переважно дієслова) – 6 списків + немає слів з жодного зі списків. Результат: номер списку від 1 до 6, або 0 – якщо немає слів зі списків

3 – наявність іменника в називному відмінку (підмету), виключаючи слова з критерію (2), якщо такі були. Результат: 1 – таке слово є (+ саме слово), 0 – такого слова немає. Можливе виділенню декількох понять.

4 – наявність дієслова (присудка), виключаючи слова з критерію (2), якщо такі були. Результат: 1 – слово є (+ саме слово), 0 – такого слова немає. Можливе виділення декількох дієслів.

На базі зазначеного набору перевірок обирається базовий шаблон запиту, або їх набір.

Потім також проводиться додаткова перевірка на наявність пов'язаних зі словом (словами) з пункту 3 прикметників (вони повинні стояти поруч і бути узгодженими щодо роду та числа). Далі перевіряється наявність додаткових умов – іменники у непрямих відмінках та пов'язані з ними прикметники. Також перевіряється наявність предикатів невідповідності. Це робиться для перевірки необхідності додавання відповідних шаблонів модифікаторів до базового шаблону.

XML-шаблони запитів на мові Cypher. Шаблони запитів зберігаються у вигляді XML-файлу спеціальної структури. Тут ми наводимо приклади шаблонів для мови запитів Cypher, яка використовується у графовій СУБД Neo4j. Її застосування є більш прийнятним через великий розмір онтології, які створюються шляхом цілковито автоматичного синтактико-семантичного аналізу тексту. Це обумовлено високою продуктивністю СУБД Neo4j при роботі з великими обсягами даних.

Приклад простого основного шаблону запиту на мові Cypher. У ***Додатку В (1)*** наведено приклад одного з таких (найпростіших) шаблонів.

Розділи XML-шаблону <match>, <where> і <return> відповідають певним розділам формального запиту на мові Cypher [94]. Деякі фрагменти вмісту (тексту) цих розділів є змінним шаблону. Змінні описуються у розділі <variables>. Для кожної змінної визначається її ім’я – <name> і призначення – <destination>. Призначення може мати значення input – значення що підставляється до шаблону або output – такі змінні не заміщаються при формуванні запиту певними вихідними значеннями, але це виказання на імена і кількість параметрів, значення яких отримуються в результаті виконання запиту. Ідентифікатор шаблону <id> слугує для його співставлення з результатом аналізу фрази користувача, а також відповідним шаблоном формування відповіді. Тег <verbose_name> міститься лише для розпізнання шаблонів запитів людиною при розробці і підтримці програмної системи. Надалі приклади шаблонів запиту будуть наводитися у спрощеному вигляді, без XML-тегів. Тег <type> вказує на тип шаблону, наразі є два типи: base – основний шаблон і additional – додатковий шаблон-модифікатор.

Приклад додаткового шаблону запиту на мові Cypher. Слід також зупинитися на структурі додаткових шаблонів. У ***Додатку В (2)*** у якості прикладу наведено один з них.

Цей тип шаблонів також має блоки <match>, <where> і <return>. Вміст цих розділів додається до відповідних блоків основного шаблону. Кожен з блоків може бути пустим (або відсутнім). Відмінними рисами додаткових

шаблонів є теги <block_union> та <next_item_union>. Тег <block_union> містить тип приєднання усього сформованого блоку <where> до базового шаблону. Тег <next_item_union> вказує на тип приєднання повторюваних елементів блоку <where> у разі, якщо відповідна вхідна змінна представлена списком (масивом). Так, наприклад, для наведеного вище шаблону змінній INPUT_VALUE_ADJ може відповідати цілий ряд зв’язаних з підметом прикметників. Значення параметрів <block_union> та <next_item_union> можуть бути “and” (І) чи “or” (АБО). Також для додаткових шаблонів притаманний тертій тип (<destination>) змінних – intermediate. Такі змінні не приймають участі ні у передачі даних до запиту, ні у їх безпосередньому отриманні. Їх головною особливістю є те, що при повторенні блоку при формуванні запиту ці змінні не дублюються повністю, а додаються з наступним порядковим номером, тобто, наприклад: ADJ_PLUS_1, ADJ_PLUS_2, ADJ_PLUS_3, …, тощо.

Спосіб автоматичного формування запитів за шаблонами. Зупинимося докладніше на структурі формальних запитів і способі їх формування. Структура онтології дозволяє проводити цілеспрямований пошук як контекстів, так і окремих понять. Вона дозволяє враховувати як наявність даних понять у контексті, так і їх зв’язаність по критерію певного семантичного типу. У запропонованій схемі є базовий шаблон запиту, спрямований отримання інформації певного типу у зазначеній формі і додаткові шаблони-модифікатори, які опціонально добудовують рядки запиту у відповідних блоках основного запиту, вносячи додаткові умови. Розглянемо приклади деяких шаблонів запитів.

Шаблон запиту контексту за словом, що входить до нього. Почнемо з, напевно, найпростішого, що його було наведено вище. Даний шаблон спрямований на отримання контексту (речення), в яке входить запитуване одиночне поняття (слово). Але не просто входить, а утворює якийсь зв'язок з іншими словами. Це гарантує, що поняття «органічно» входить у контекст.

Запити в Cypher поділяються на три основні блоки: MATCH, WHERE та RETURN. Блок MATCH задає патерн зв'язаності вершин в орієнтованому графі. У блоці WHERE накладаються умови на властивості (характеристики) вказаних у блоці MATCH вершин та/або зв'язків. Блок RETURN показує, що слід вивести як результат і під яким ім'ям (алеасом). У даному випадку є певний клас, позначений змінною inp. У блоці WHERE ми наклали умову, що властивість label вершини inp має дорівнювати запитаному поняттю (тут і далі в шаблонах запитів INPUT_VALUE – це текст вхідного поняття). У блоці MATCH зазначено, що inp – вершина (круглі дужки), яка має тип Class. Вона пов'язана з іншою вершиною n, що має тип Relationship (то є властивість із OWL). Тип зв'язку при цьому не визначений (квадратні дужки порожні), напрямок зв'язку також не вказано. Тобто він може бути прив'язаний як DOMAIN так і до RANGE. Напрямки вказувати у разі немає сенсу, оскільки відомо, що такі зв'язки будуються від властивості до класу. Також ми вказали, що ця властивість також має пов'язувати деякий клас x. Далі ми вказали, що властивість, що об'єднує ці класи, повинна відноситись до якогось речення rel_sent. Умова sent_super.name = "SentenceGroups" гарантує, що rel_sent буде саме реченням. Як результат ми просимо вивести rel_sent.label, який містить контекст речення під алеасом CONTEXT.

Шаблон запиту характеристики об’єкта. Розглянемо дещо складніший приклад. Ми хочемо запросити відомі в онтології характеристики (визначення) об'єкта INPUT_VALUE. Тобто, яким може бути (буває) INPUT_VALUE? Запит буде мати наступний вигляд:

```
MATCH (inp:Class)-[]-(n:Relationship),
    (n:Relationship)-[]-(x:Class),
    (n)-[:SPO]->(prop_type_1),
    (n)-[:SPO]->(rel_group),
    (rel_group)-[:SPO]->(rel_sent),
    (rel_sent)-[:SPO]-(sent_super)
WHERE
    inp.name = "INPUT_VALUE" and
```

```
    (prop_type_1.label = "свойство объекта" or
    prop_type_1.label = "свойство действия" or
    prop_type_1.label = "отделимость действия" or
    prop_type_1.label = "уровень воздействия")
     and
    sent_super.name = "SentenceGroups"
RETURN DISTINCT x.label as result,  rel_sent.label as cotnext;
```

Порівняно з попереднім прикладом, у блоці MATCH додано один рядок: (n)-[:SPO]->(prop_type_1). Це дає інформацію про те, що властивість n має бути дочірньою по відношенню prop_type_1 (тип зв'язку). Тут ми явно вказуємо напрямок зв'язку. У блоці WHERE ми конкретизуємо prop_type_1 через можливі значення параметра label. Для того, щоб зробити шаблон більш універсальним, так як ми апріорі не знаємо, чи INPUT_VALUE іменник, чи дієслово, наводиться кілька варіантів значення prop_type_1.label через логічне АБО. При введенні у онтології додаткової ієрархії семантичних відношення, цю конструкцію можна спростити:

```
MATCH (inp:Class)-[]-(n:Relationship),
    (n:Relationship)-[]-(x:Class),
    (n)-[:SPO]->(prop_type_1),
    (n)-[:SPO]->(rel_group),
    (rel_group)-[:SPO]->(rel_sent),
    (rel_sent)-[:SPO]-(sent_super),
    (prop_type_1)-[:SPO]->(prop_type_category)
WHERE
    inp.name = "INPUT_VALUE" and
    prop_type_category.label = "типы свойств"
     and
    sent_super.name = "SentenceGroups"
RETURN DISTINCT x.label as result,  rel_sent.label as cotnext;
```

У якості результату ми виводимо label для вершин x. Це буде характеристики (властивості) об'єкта inp. Також ми запитуємо і текст речення, щоб дізнатися, в якому контексті згадано цю властивість.

Аналогічно можна запросити дії об'єкта. Для цього треба лише вказати інше значення prop_type_1.label у блоці WHERE, зокрема: `prop_type_1.label = "объект-действие"`.

Шаблон запиту з поверненням уточненого семантичного типу. У разі використання умови з декількома можливими варіантами типів зв'язку (prop_type_1.label), можна в результат також включити отримане значення типу, що допоможе при синтезі відповіді. Наведемо приклад, де запитується локалізація об'єкта без уточнення типу локалізації («Де розташований INPUT_VALUE?»).

```
MATCH (inp:Class)-[]-(n:Relationship),
    (n:Relationship)-[]-(x:Class),
    (n)-[:SPO]->(prop_type_1),
    (n)-[:SPO]->(rel_group),
    (rel_group)-[:SPO]->(rel_sent),
    (rel_sent)-[:SPO]-(sent_super)
    (prop_type_1)-[:SPO]->(prop_type_category)
WHERE
     inp.label = " INPUT_VALUE " and
    prop_type_category.label = "типы локализации" and
    sent_super.name = "SentenceGroups"
RETURN DISTINCT x.label as result, rel_sent.label as context,
           prop_type_1.label as predicate;
```

Основна відмінність тут – наявність у блоці RETURN виразу prop_type_1.label as predicate. Це дозволяє додатково повернути конкретний семантичний тип отриманого результату. На нього слід орієнтуватись при генерації тексту відповіді.

У ряді випадків крім предикатів у запитах можна використовувати списки варіантів понять (характерних дієслів, іменників, прикметників). Головна відмінність тут, що умови накладаються на вершину онтолгафу пов'язану з x.label. Тобто запитуваний об'єкт повинен бути пов'язаний з деяким поняттям x зв'язком певного типу, наприклад «об'єкт-дія», причому ця «дія» має бути за текстом параметру label однією з перерахованих у

списку. У перспективі планується ввести в онтологію апріорну (не залежну від аналізованого тексту) класифікацію дій, ознак, понять, що спростить запит і позбавить його наявності подібних списків – поняття х просто має бути дочірнім по відношенню до, наприклад, дієслів певної категорії.

Робота із додатковими шаблонами-модифікаторами. Окремо слід згадати про шаблони-модифікатори – фрагменти, що добудовуються до основних шаблонів запитів. Наприклад, вхідним параметром є не окреме слово, а іменна група. Тобто є зв'язані іменники та прикметники. Так, для прив'язки до вхідного поняття прикметника слід до відповідних блоків додати рядки:

у розділі MATCH:

```
(inp:Class)-[]-(adj_plus:Relationship),
(adj_plus:Relationship)-[]-(inp_adj_1:Class),
(adj_plus)-[:SPO]->(rel_group)
```

у розділі WHERE:

```
 and
inp_adj_1.label = "INPUT_VALUE_ADJ"
```

Для таких додаткових прикметників можуть бути також додані подібні блоки, але зі змінними inp_adj_2, inp_adj_3 тощо. Можна додати до запиту умову наявності іменнику у непрямому відмінку за допомогою наступних блоків:

у розділі MATCH:

```
(inp_noun_1:Class)-[]-(noun_plus:Relationship),
(noun_plus)-[:SPO]->(rel_group)
```

у розділі WHERE:

```
 and
inp_noun_1.label = "INPUT_VALUE_NOUN"
```

Тут додається умова лише входження цього додаткового іменника до тієї ж групи, у яку входить основне запитуване поняття. До цього додаткового іменника також можна накласти умови наявності пов'язаних з ним прикметників:

у розділі MATCH:

```
(inp_noun_1:Class)-[]-(adj_plus_add:Relationship),
(adj_plus_add:Relationship)-[]-(inp_adj_add:Class),
(adj_plus_add)-[:SPO]->(rel_group)
```

у розділі WHERE:

```
 and
inp_adj_add.label = "INPUT_VALUE_ADJ_ADD"
```

Також в окремих випадках до запиту потрібно прив'язати предикат невідповідності. Для цього слід накласти умови входження у групу заперечної частки чи іншого предиката невідповідності. Для чого до запиту добудовується конструкція:

у розділі MATCH:

```
(neg:Class)-[]-(neg_rel:Relationship),
(neg_rel)-[:SPO]->(rel_group)
```

у розділі WHERE:

```
 and
(neg.label = "не" or
 neg.label = "ні" or
 neg.label = "заборонено" or
 neg.label = "нема" or
 neg.label = "непотрібно" or
 neg.label = "неможна" or
 neg.label = "неможливо")
```

2.4.5 Спосіб приведення сутностей, виділених з вихідної природномовної фрази до значень найближчих понять наявних у конкретній онтології

Ідея приведення сутностей до найближчих наявних у онтології. Часто трапляється так, особливо це стосується онтології створюваних вручну, що сутності у вершинах онтологічного графа висловлені у вигляді довгих багатослівних формулювань. Таким чином, існує досить низька вірогідність того, що сутності виокремленні з вихідної фрази користувача

будуть точно співпадати з наявними у вершинах онтологічного графу. Таким чином, постає задача пошуку найбільш близького до отриманого поняття присутнього у онтології.

Для полегшення обробки попередньо автоматично, за допомогою спеціального скрипта, формується файл, який містить всі наявні у онтології сутності. Для підвищення продуктивності подальшої роботи сутності розбиті на відповідні категорії (назви класів, предикатів і екземплярів). До кожної такої назви поставлено у відповідність список слів, що входять до неї у лематизованій формі. В свою чергу кожне з цих слів попередньо характеризується щодо приналежності до певної частини мови. Окрім того, даний файл доповнюється також поняттями присутніми в умовах дерева прийняття рішень чи списках перевірки за набором тестів (залежно від обраного методу аналізу вхідних висловлювань).

Процес приведення сутностей до найближчих наявних у онтології. Виділена з вихідної фрази сутність (вона може складатися з одного чи декількох слів) розбивається на окремі слова, кожне з яких лематизується. З отриманого списку спочатку видаляються слова, що взагалі відсутні як у онтології, так і в умовах перевірок. Це пришвидшує процес подальшої обробки. Для решти слів, що залишилися у списку, визначається їх приналежність до певної частини мови. Далі йде процес співставлення слів з отриманого списку зі словами, поставленими у відповідність поняттям з онтології. Цей процес чимось близький до описаного вище встановлення метрики релевантності відповідей. Так, співпадіння слів з вихідного списку з таким зі списку слів для відповідного поняття з онтології дає позитивний ваговий коефіцієнт, чисельне значення якого залежить від частини мови, до якої відноситься порівнюване слово. Натомість, як розширення, так і звуження контексту збоку вихідного списку, або списку слів для відповідного поняття з онтології дає, відповідно, негативний коефіцієнт. Розширення та звуження контексту можуть мати різні чисельні значення вагових коефіцієнтів. Самі конкретні значення вагових коефіцієнтів

встановлюються експериментально. Отримані значення сумуються та відносяться до загальної кількості слів. Найбільш близьким вважається поняття, для якого обчислено максимальне значення зазначеного показника.

Значення критерію відповідності на практиці розраховувалося за формулою 2.8:

$$M = 2 \cdot \eta_{match} - 0.5 \cdot \eta_{extra} - 0.5 \cdot \eta_{nf} \tag{2.8}$$

де η_{match} – фактор збігу слів; η_{extra} – фактор наявності додаткових слів; η_{nf} – фактор відсутності сдів.

Фактори η_{match}, η_{extra} та η_{nf} із формули (2.8) розраховуються за запропонованими формулами (2.9) – (2.11):

$$\eta_{match} = \frac{a_{11} \cdot N_{N-match} + a_{12} \cdot N_{A-match} + a_{13} \cdot N_{V-match} + a_{14} \cdot N_{O-match}}{W_{words-1} + W_{entity-1}} \tag{2.9}$$

$$\eta_{extra} = \frac{a_{21} \cdot N_{N-extra} + a_{22} \cdot N_{A-extra} + a_{23} \cdot N_{V-extra} + a_{24} \cdot N_{O-extra}}{W_{words-2}} \tag{2.10}$$

$$\eta_{nf} = \frac{a_{31} \cdot N_{N-nf} + a_{32} \cdot N_{A-nf} + a_{33} \cdot N_{V-nf} + a_{34} \cdot N_{O-nf}}{W_{entity-2}} \tag{2.11}$$

де a_{nm} – емпіричні вагові коефіцієнти; $N_{N\text{-}match}$ – кількість збігів для іменників; $N_{A\text{-}match}$ – кількість збігів для прикметників та дієприкметників; $N_{V\text{-}match}$ – кількість збігів для дієслів; $N_{O\text{-}match}$ – кількість збігів для слів, що належать до інших частин мови; $N_{N\text{-}extra}$ – кількість додаткових іменників; $N_{A\text{-}extra}$ – кількість додаткових прикметників та дієприкметників; $N_{V\text{-}extra}$ – кількість додаткових дієслів; $N_{O\text{-}extra}$ – кількість додаткових слів, що належать до інших частин мови; $N_{N\text{-}nf}$ – кількість відсутніх іменників; $N_{A\text{-}nf}$ – кількість відсутніх прикметників та дієприкметників; $N_{V\text{-}nf}$ – кількість відсутніх дієслів; $N_{O\text{-}nf}$ – кількість відсутніх слів, що належать до інших частин мови; $W_{words\text{-}n}$ – зважена сума слів у вхідній сутності; $W_{entity\text{-}n}$ – зважена сума слів у порівняльній сутності.

Зважені суми слів W із формул (2.9) – (2.11) обраховуються за принципом, що кожному слову присвоюється відповідне числове значення, залежно від приналежності до певної частини мови. Ці значення

підсумовуються. Ці числові значення відповідають ваговим коефіцієнтам a_{nm} у відповідній формулі.

Експериментально були встановлені наступні величини емпіричних коефіцієнтів a_{nm}, що наведені у таблиці 2.1:

Таблиця 2.1 – Експериментально встановлені значення коефіцієнтів a_{nm}

	Номер коефіцієнту			
Формула	1	1	3	4
2.9 (1)	0,43	0,26	0,18	0,13
2.10 (2)	0,44	0,22	0,22	0,12
2.11 (3)	0,5	0,25	0,13	0,12

Саме такі значення вагових коефіцієнтів сумування були задіяні при розробці діалогової системи за матеріалами «Білої книги з ФРМ».

2.4.6 Використання великих мовних моделей для задач аналізу тексту – виявлення намірів та іменованих сутностей

2.4.6.1 Особливості інтеграції великих мовних моделей у програмні системи. Поява великих мовних моделей відкрила можливість використання їх для виконання задач семантичного аналізу природномовних текстів. Прикладом використання такої великої мовної моделі, що розглядається у даній роботі є ChatGPT. Хоча GPT відомий в першу чергу як природномовна діалогова система широкого профілю, він також може стати інструментом, що виконує певні задачі з обробки текстової інформації: визначення намірів, іменованих сутностей, побудова логічних висновків, пошук інформації і навіть задачі корегування стилю і машинного перекладу. І це лише частка можливостей моделі GPT. В тому числі API ChatGPT (або іншої великої мовної моделі) дозволяє інтегрувати його у іншу програмну систему, як складову, що виконує певні задачі.

Модель ChatGPT надає можливість взаємодії через API, що передбачає передачу команд природною мовою. Слід зазначити, що вважається кращим варіантом використовувати англійську мову для таких команд та інструкцій, навіть при роботі із іншими мовами. Це обумовлено тим, що англійська є основною мовою для ChatGPT.

З огляду на обмеження кількості токенів, які може обробити ChatGPT, інструкції повинні бути короткими, але точними і лаконічними. Досліди показали, що форматування інструкцій у форматі JSON є ефективним підходом для надання зрозумілих та вичерпних команд моделі. Цей підхід сприяє структурованості і легкій інтерпретації інформації, що подається на вхід. Використання цих стратегій покращує якість взаємодії між користувачами та системою, що призводить до більш точних і значущих результатів порівняно з іншими методами, такими як, наприклад, техніка ланцюжка думок.

2.4.6.2 Використання великої мовної моделі як компоненту онтологокерованої діалогової системи. Підхід, що розглядається у даному підрозділі, відрізняється від методології, використаної в наших попередніх дослідженнях [84, 85, 97]. Так, там онтологія служила основним сховищем інформації в системі, а не контейнером для правил та інструкцій для організовані роботи системи. Тим не менш, розглянутий тут підхід включає і певні елементи цих попередніх розробок, наприклад, використовуються такі методи, як вилучення іменованих сутностей, аналіз пов’язаного контексту та автоматичне створення формальних SPARQL запитів [84] із наданих користувачем природномовних фраз.

Представлену методологію можна розділити на два ключові компоненти, кожен з яких має окрему мету в розробці системи OntoChatGPT.

По-перше, ми зосереджуємося на техніці метанавчання на основі підказок і створенні структурованих підказок для ChatGPT. Цей підхід передбачає використання підказок для інструкцій процесу метанавчання ChatGPT, дозволяючи йому генерувати контекстуальніші та точніші

відповіді. Ми заглиблюємося в методологію, що лежить в основі розробки та впровадження цих підказок, підкреслюючи їх значення для покращення розмовних можливостей ChatGPT.

Друга частина методології зосереджена навколо розробки автоматичної діалогової системи, керованої онтологією, яка об’єднує ChatGPT і структуровані підказки. Основна ідея цієї системи полягає в тому, щоб об’єднати конкретні предметні області та пов’язані з ними контексти, які можуть містити інформацію, пов’язану з доменом, яку не повністю охоплено в базі знань ChatGPT. Ці контексти зберігаються в базі даних, такій як MongoDB або реляційній базі даних, і пов’язані з наборами іменованих сутностей із власною онтологічною структурою. Крім того, можна використовувати аналіз настроїв для класифікації контекстів. Прив’язка іменованих сутностей до відповідних контекстів включає семантичні компоненти, які конкретизують роль сутності в контексті.

Ці додаткові функції спрямовані на підвищення релевантності та чіткості вибраного контексту для подальшої обробки. Для автоматизації цих процесів ми використовуємо наші раніше розроблені інструменти [82, 98, 99, 100] і включаємо попередньо навчені трансформаторні моделі на основі BERT, такі як [101].

2.4.6.3 Формування та використання структурованих інструкцій-підказок. ChatGPT виявляється цінним ресурсом для семантичного аналізу та виділення іменованих сутностей із наданих користувачем фраз. Спеціально для цього створюються структуровані інструкції-підказки. Крім того, ChatGPT використовується для аналізу намірів, висловлених у фразах користувача. Визначені наміри і відповідні їм іменовані сутності, а також вибраний із бази знань список контекстів потім надаються як вхідні дані для ChatGPT. Ці вхідні дані супроводжуються відповідними структурованими підказками, які пояснюють інформацію, яку потрібно отримати, і бажаний формат представлення.

Однією з особливостей системи є гнучкість її структурованих підказок для ChatGPT. Замість жорстких підказок вони динамічно генеруються на основі конкретної ситуації за допомогою інструкцій, наданих у формі мета-онтології. Ця мета-онтологія описує поля, які слід включити в структуру JSON (або XML), і відповідні фрази підказок, які потрібно вставити. Кожна інструкція або структурована підказка для ChatGPT має власний набір полів і попередньо визначених значень, які можна у неї включити. Крім того, підказка містить структуру шаблону для відповіді, яка забезпечує послідовність і спрощує подальшу обробку. Створення фраз-підказок використовує перевірені методи з [74, 102] для отримання ефективних і зв'язних інструкцій.

Підказка може містити різні можливі наміри, такі як «кількість», «спосіб дії», «об'єкт», «суб'єкт», «дія», «місце», «напрямок», «місце дії», «умови», «інструмент», «співпраця», «відношення», «причина», «послідовність», «походження» та ін. Ці наміри представляють семантичні категорії та забезпечують основу для розуміння запиту користувача. Крім того, структурована підказка містить поля для інформації, яку потрібно надати, мови, що використовується, та інші технічні деталі, пов'язані із введенням і виведенням.

Визначення іменованих сутностей виконується за допомогою ChatGPT за допомогою іншої інструкції. Результат повинен надати лемматизовані слова, згруповані за сутностями, вказати тип кожної групи (іменна чи дієслівна) і вказати головне слово в кожній групі.

Результати такого аналізу далі можна використовувати у відповідній діалоговій інформаційній системи для запровадження обробки обраних контекстів на предмет синтезу відповіді відповідно до визначених у такий спосіб намірів висловлених користувачем у переданім системі тексті.

2.5 Висновки за розділом 2

Запропоновано систему семантичних відношень, основною відмінністю якої є виведення значного розмаїття семантичних категорій з аксіоматичного набору понять верхнього рівня. Отримання нових семантичних категорій при цьому відбувається шляхом комбінації будь-яких двох категорій, кожна з яких може бути первинною, або похідною будь-якого рівня. Кількість похідних категорій не обмежена. Метод дозволяє створити обґрунтовану онтологію семантичних категорій будь якої глибини і повноти відповідно предметної області і призначення створюваної онтології. Це в свою чергу дає змогу систематизувати наявні у онтології предикати і тим самим більш уніфікувати їх структуру і спростити систему запитів для пошуку інформації.

Розроблено методи синтактико-семантичного аналізу природномовних текстів, орієнтовані на флективні мови. Дві основні задачі, що вирішуються за допомогою запропонованих методів – це автоматична побудова онтологій на основі природномовного тексту і аналіз природномовних висловлювань для визначення їх семантичних типів і сутностей, що їх конкретизують. Друге розглядається в аспекті побудови релевантних формальних запитів для отримання інформації з онтології.

Розроблено три основних підходи до автоматизованого створення онтології на базі природномовного тексту, або набору текстів. Кожен з них має свої переваги і недоліки, а також оптимальні області застосування.

Автоматично створені онтології першого типу створюються на наборах текстів, розмічених по принципу «заголовок – контекст». Такий набір текстів може бути отриманий, наприклад, шляхом сканування веб-сторінок інформаційного ресурсу. На кожній сторінці можна знайти назву статті, її текст і, можливо, деяку іншу інформацію в певних місцях (ілюстративний матеріал, посилання, аудіо- та відео матеріали). Такі онтології можуть бути корисними для діалогових і довідкових систем. Для них досить легко створити обмежену кількість уніфікованих формальних шаблонів запитів.

Довідкова система, в основу якої покладено подібну онтологію, стає досить надійною, швидкою у створенні, не вибагливою до структури природної мови.

Другий тип онтології може бути автоматично створено на основі набору текстів з регулярною структурою. Структура онтології має бути наперед заданою, також повинні бути встановлені і правила, за якими розбираються структуровані документи. Перевагою методу є можливість створення досить глибоко семантично структурованої онтології, при цьому її структура буде зрозумілою і наочної для людини при перегляді. При цьому структура онтології є довільною, як і набір правил розбору. Метод має обмежену область придатності і його застосування виправдане при наявності дійсно великого обсягу однотипних текстів, коли раціональним виглядає розробляти спеціальний набір правил розбору і структуру онтології (фактично індивідуальну програмну інструкцію). Також індивідуальним буде і набір правил для роботи з такою онтологію. Тому ступінь уніфікації методу вкрай низька, при високій семантичній структурованості і швидкості обробки однотипних документів.

Третій з запропонованих методів забезпечує цілком автоматичне створення OWL-онтології з тексту природною мовою. Основна особливість методу полягає в тому, що він не потребує ні попередньої розмітки тексту, ні наявності набору текстів за регулярною (повторюваною) наперед визначеною структурою. Суть методики полягає в заснованих на правилах методах синтаксисно-семантичного аналізу і на тому факті, що велика кількість семантичної інформації у флективних мовах може бути отримана шляхом аналізу поєднань різних частин мови у певних словоформах, а також прийменників. Онтології створені в такий спосіб є глибоко семантично структурованими і водночас досить регулярно організованими. Але їх недоліками є великий об’єм, що значно перевищує обсяг первинного тексту і має вкрай не наочний для людини вигляд. Також слід зазначити, що даний підхід дуже вибагливий до якості первинного синтаксичного розбору тексту.

Тим не менш, він видається найбільш перспективним інструментом, який може суттєво та ефективно автоматизувати створення онтологічних графів, використовуючи звичайні тексти без попереднього ручного маркування чи наперед визначеної структури.

Розроблено методи аналізу природномовних фраз для визначення основних семантичних відношень висловлених у них, а також понять, що конкретизують ці відношення.

В основі першого з зазначених методів лежить дерево прийняття рішень та аналіз фрази користувача як послідовності слів. Умовами переходу до відповідної вершини на наступному рівні дерева є відповідність слова або групи слів таким зі списків у кожній з вершин. У кінцевій вершині зазначені найбільш вірогідні семантичні категорії, а також позиції, з яких слід виокремити поняття, що конкретизують відповідні семантичні відношення. Недоліком даного методу є трудоємність ручної побудови відповідного дерева. Тому він більш придатний для мов з більш регулярною структурою речення (англійська, норвезька), а також відносно обмеженим набором семантичних типів, що визначаються в такий спосіб.

Другий метод є більш адаптованим до флективних мов, у яких структура речення є досить довільною, а первинну роль відіграють словоформи, та їх комбінації. В основі способу покладено набір перевірок, що їх проходить вихідна фраза. Кожна перевірка має певний набір результатів. Комбінація отриманих результатів перевірок визначає найбільш вірогідний семантичний тип висловлення, або їх набір. Оскільки кожна з перевірок направлена на наявність у реченні певних сутностей, вона здатні і виокремити їх (за їх наявності). Застосування такого методу аналізу видиться перспективним при роботі з цілком автоматично створеними онтологіями, тобі як попередній спосіб більш використований при роботі з онтологіями створеними вручну.

Основна задача вищезазначених методів аналізу полягає у створенні на базі природномовної фрази формального запиту до онтології на мовах

SPARQL або Cypher. Для цього кожному з семантичних типів, що визначаються, поставлено у відповідність певний шаблон формального запиту. До зазначених позицій шаблону підставляються поняття виокремленні з вихідної фрази, на певних етапах проходження аналізу (позиція на шляху обходу дерева, або перевірка певного типу).

РОЗДІЛ 3
РОЗРОБКА МЕТОДІВ СИНТЕЗУ ОСМИСЛЕНОГО ПРИРОДНОМОВНОГО ТЕКСТУ НА ОСНОВІ РЕЗУЛЬТАТІВ ВИКОНАННЯ ФОРМАЛЬНИХ ЗАПИТІВ ДО ОНТОЛОГІЇ

3.1 Вступ

Як зазначалося у попередньому розділі, відповідь на формальний запит до онтології може являти собою готовий контекст. Такий підхід часто цілком виправдний. Проте, нерідко результатом формального запиту є просто набір (список, таблиця) сутностей, також при використанні онтологій певних типів можливе повернення зв’язаних структур – сутності та семантичні відношення між ними. В цьому разі з метою збільшення зрозумілості і дружності людино-машинної взаємодії постає задача синтезу на основі цих результатів природномовної відповіді. Така відповідь повинна бути осмисленою, релевантною до вихідного запиту користувача і граматично коректною. Саме задачі синтезу такої відповіді було присвячено наші дослідження освітлені у даному розділі.

Основною метою дослідження була розробка методу генерації змістовної та граматично правильної, відповідної потребам користувача діалогової або довідкової системи. Основний акцент було зроблено на природних мовах флективного типу, зокрема українській. Окрім результатів запиту матеріалом для синтезу відповіді слугували також і результати семантичного аналізу вихідної фрази користувача. Розглянуті шляхи підвищення релевантності відповіді, а також методики синтез тексту з набору понять, отриманих шляхом виконання формального запиту. Проаналізовано генерацію природномовних фраз для різних ситуацій: нормально сформована відповідь, відсутність результатів формального запиту, перевищення обмеження кількості результатів запиту, проблемні ситуації, пов’язані з неправильною ідентифікацією іменників на етапі аналізу.

Також виконано експерименти зі зворотного синтезу природномовних речень українською мовою на базі їх онтологічного переставлення. Для вирішення даної задачі були задіяні великі мовні моделі, зокрема, можливості ChatGPT.

3.2 Метод синтезу природномовних відповідей, на базі результатів формального запиту до онтологічної бази знань

3.2.1 Зв'язок процесів аналізу і синтезу природномовних текстів в аспекті діалогових систем

У внутрішній структурі інтерактивних діалогових і довідкових систем виділяють три основні функціональні блоки: аналізатор вхідних повідомлень, база знань і система генерації відповідей. Всі три блоки однаково важливі і взаємодіють один з одним. Цей розділ присвячено третьому блоку, який відповідає за генерацію (синтез) відповідей. Але слід зазначити, що характер і особливості його діяльності безпосередньо залежать від принципу функціонування та структури інших. Тому в міру необхідності будемо звертатися до них для конкретизації описаних процедур. Коротко опишемо роботу модулів аналізу та бази знань. Надійшовши з інтерфейсу користувача, природномовна фраза піддається різним видам аналізу: графематичному (виведення слів, знаків пунктуації, речень, абзаців, складання розірваних слів, видалення непередбачених символів), морфологічному (визначення частин мови та їх словоформи), синтаксичному (виділення іменних груп, побудова графу синтаксичних зв’язків) і семантичному (конкретизація загальних думок, висловлених фразою, наприклад визначення чи це є запитання чи твердження, і якщо, скажемо, це питання, то що саме є об’єктом запиту: назви сутностей, спосіб дії, кількість предметів, час, коли відбулася чи буде відбуватися подія, тощо). Результати цього аналізу створюють необхідний матеріал для побудови формальних запитів до бази знань. Існує багато способів представлення бази знань і організації її структури. Мова та

структура формальних запитів залежать головним чином від цього фактора.

У даній роботі насамперед звертається увага на онтологічні бази знань і побудову формальних запитів на мові SPARQL. Проблеми побудови SPARQL-запитів на основі аналізу природного тексту ми розглядали у попередньому розділі 2. Тут ми лише згадуємо цей факт. Часто на основі фрази користувача можна побудувати не один формальний запит, а їх пакет. Це може бути пов'язано з кількома причинами: фраза користувача досить складна, містить твердження та/або питання різних категорій; необхідно з'ясувати, до яких класів вищої категорії відносяться класи та сутності, отримані як відповіді на формальний запит. Певні обмеження на те, нащадком якого класу або класів можуть бути відповіді, можуть бути вже реалізовані в самому запиті, але це не завжди можливо та/або доцільно. Побудова запитів з використанням редукованих сутностей може здійснюватися у двох випадках: якщо запитувані сутності в онтології відсутні саме в такій специфічній формі; коли очікується, що користувач також має отримати додаткову пов’язану інформацію у якості відповіді. Ступінь скорочення запитуваних сутностей і спосіб його реалізації залежить від цілей і технічних характеристик проектованої системи.

Результатом формального запиту є деяка сутність або їх набір. Цей результат і отримані в ході аналізу дані, насамперед семантичні, стають вихідними параметрами для побудови відповіді. Таким чином, переходячи до опису синтезу відповідей, можна зазначити, що вихідними даними для синтезу фрази відповіді є набір сутностей, отриманих у результаті формального запиту до бази знань та результат семантичного аналізу, а саме, до яких типів висловлювань відноситься вихідна фраза та на які наміри та інтереси користувача вона вказує.

3.2.2 Принципи синтезу природномовних відповідей залежно від організації діалогової системи.

Подальшу стратегію можна розділити на два принципово різних шляхи, які залежать від структури онтології та того, що саме має бути видано у

якості відповіді. Перший випадок досить простий, коли результатом запиту є готова відповідь (контекст) або блок відповідей. Другий випадок — отримання певного набору сутностей, можливо, із зазначенням, до яких класів вищої ієрархії або властивостей в базі знань вони належать. Це вимагає більш складного підходу до синтезу граматично правильної та зрозумілої фрази. Ми розглянемо обидва варіанти.

3.2.2.1 Формування відповіді діалогової системи за умов отримання контекстних результатів

Обґрунтування умов повернення контекстних результатів. Перший згаданий принцип, незважаючи на уявну тривіальність, має деякі особливості, описані нижче. Крім того, його застосування не є чимось поганим і недосконалим, навпаки, в деяких випадках, коли відповідь є досить великим текстом, містить формули, табличні дані, графіки, ілюстрації тощо, наявність готової відповіді видається кращим рішенням. Як і в природному спілкуванні, нам іноді простіше показати у якості відповіді певну статтю, надати таблицю або графік, написати математичну формулу, а не намагатися пояснити співрозмовнику необхідну інформацію, як кажуть, «на пальцях».

Особливості обробки контекстних відповідей для представлення їх користувачу. Розглянемо особливості цього підходу. Перша з них полягає в тому, що на основі виконання формального запиту можна отримати ряд готових відповідей (або посилань на них). У цьому випадку система повинна вибрати, яку з отриманих відповідей надати користувачеві або надати як основну. Спосіб вирішення цієї проблеми полягає в ранжуванні відповідей. Для забезпечення можливості зручного ранжування готові відповіді повинні містити перелік ключових слів або короткий, але місткий заголовок, що характеризує дану відповідь, а точніше типове запитання, на яке передбачається отримати цю відповідь. При наявності заголовка передбачається провести його графематичний і морфологічний аналіз з метою виявлення основних значущих слів і їх приналежності до певної частини мови.

Методика автоматизоване ранжування контекстних відповідей за ступенем релевантності. Ранжування здійснюється на основі метрики, яка розраховується з використанням збігу та невідповідності головних слів із фрази користувача зі словами зі списку ключових слів відповіді та виражає ступінь її релевантності. Ми пропонуємо метрику, розраховану з наступних міркувань. Збіг іменників та іменникових словосполучень із фрази користувача з такими зі списку ключових слів або із заголовка відповіді є позитивним фактором, оскільки саме в них фігурують основні сутності. Наявність у ключових словах відповіді додаткових іменників, які не містяться у фразі користувача, є скоріше негативним фактором. Це розширення контексту за допомогою додаткових сутностей, які можуть мати інший сенс або бути нецікавими для користувача. Збіг дієслів із фрази користувача з дієсловами зі списку ключових слів відповіді також є позитивним фактором, але він має меншу вагу, ніж збіг іменників. Наявність у ключових словах або в заголовку відповіді додаткових дієслів, які не містяться у фразі користувача, не слід вважати негативним фактором. Навпаки, з практичної точки зору, користувача може більше цікавити те, що можна робити з цими сутностями або які дії вони виконують, а не просто отримання їх визначення. Наявність збігів в інших частинах мови, якщо вони були введені в ключові слова або були розібрані з заголовка, можна вважати позитивним фактором, хоча і ще менш суттєвим. Відсутність збігу в інших частинах мови передбачається або не брати до уваги, або враховувати як слабкий негативний фактор. Виходячи з наведених вище тверджень, можна запропонувати наступну формулу для визначення релевантності відповіді фразі користувача:

$$f_{ngm} = \frac{N_{add.n.g.}}{N_{i.n.g} + 1} \tag{3.1}$$

$$f_{mng} = \frac{N_{c.n.g.}}{N_{i.w.} + 1} \tag{3.2}$$

$$f_{mv} = \frac{N_{c.v}}{N_{i.w.} + 1} \quad (3.3)$$

$$f_{vm} = \frac{N_{add.v}}{N_{i.w.} + 1} \quad (3.4)$$

$$f_{om} = \frac{N_{add.o.}}{N_{i.w.} + 1} \quad (3.5)$$

$$M = a_1 \cdot f_{nmg} + a_2 \cdot f_{mng} + a_3 \cdot f_{mv} + a_4 \cdot f_{vm} + a_5 \cdot f_{om} \quad (3.6)$$

де: f_{ngm} - фактор відносної кількості додаткових іменників; $N_{add.n.g.}$ - кількість додаткових іменників, яких немає у фразі користувача; $N_{i.n.g}$ - загальне число іменників у фразі користувача; f_{mng} - фактор відносної кількості збігів в іменниках; $N_{c.n.g.}$ - число узгоджених іменників; $N_{i.w.}$ - загальна кількість слів у фразі користувача; f_{mv} - фактор відносної кількості збігів дієслів; $N_{c.v}$ - кількість відповідних дієслів; f_{vm} - фактор відносної кількості додаткових дієслів; $N_{add.v}$ - кількість додаткових дієслів, які відсутні у фразі користувача; f_{om} - фактор відносної кількості збігів в інших частинах мови; $N_{add.o.}$ - кількість відповідних інших частин мови; M – метрика релевантності відповіді; $a_1 \ldots a_5$ – коефіцієнти. Їх значення можна пояснити таким чином: значення a_1 негативне, оскільки наявність додаткових сутностей знижує релевантність. Значення a_2 є додатним і найбільшим, тому що збіги в іменниках є найважливішим фактором. Значення a_3 додатне, але воно менше a_2. Значення a_4 додатне, але невелике. Значення a_5 є негативним і малим.

Точні числові значення коефіцієнтів з формули (3.6) визначаються експериментально на основі аналізу роботи системи. У першому наближенні ми прийняли такі значення цих коефіцієнтів:

$a_1 = -1$;

$a_2 = 2$;

$a_3 = 1$;

$a_4 = 0{,}5$;

$a_5 = -0{,}5$.

Найрелевантнішою буде відповідь із максимальним значенням вищезазначеної метрики. Якщо є кілька відповідей з однаковими показниками, слід використовувати додаткові критерії. Наприклад, якщо заголовок відповіді містить менше слів, то, ймовірно, він описує більш загальну концепцію. Оскільки питання, відповідно до метрики, не однаково визначено щодо розширення додатковими сутностями, відповідь, яка описує більш загальну концепцію, ймовірно, буде більш доречною.

Ранжування за цією метрикою також може бути застосовано, якщо необхідно надати користувачеві додаткові відповіді, що стосуються його питання, крім основного.

3.2.2.2 Синтез відповіді на основі набору сутностей, отриманих в результаті виконання запиту

Сутність методу генерації природномовної відповіді з використанням шаблонів. Другий принцип заснований на синтезі оригінальної відповіді на основі набору сутностей, отриманих в результаті формального запиту до бази знань. У цьому випадку наперед підготовленої відповіді (контексту) немає, і її потрібно сформулювати. Для побудови відповіді використовуються гнучкі шаблони, які містять підготовлені фрагменти тексту (можливо з варіаціями) і позиції для вставних сутностей, отримані шляхом виконання формального запиту до бази знань. Також шаблон може містити позиції для заміни слів і фраз з оригінальної фрази користувача. Шаблон також надає варіанти побудови відповідей для виняткових ситуацій, наприклад, випадок відсутності сутностей, відповідних запиту. Вибір шаблону для відповіді базується на результатах семантичного аналізу вхідної фрази користувача, а саме, до якого типу речень відноситься ця фраза, що саме хоче знати користувач. Форма того чи іншого шаблону залежить від граматичних особливостей відповідної мови. У практичній реалізації шаблони відповідей зручно зберігати у вигляді xml-файлів. Такий

файл фактично є інструкцією, за якою програма будує свою відповідь. Орієнтовна мінімальна кількість таких шаблонів, необхідних для чат-бота довідкової системи, становить від 10 і більше штук.

Приклад шаблону для випадку запиту переліку сутностей

1) Вигляд шаблону. Приклад такого шаблону для української мови наведено у ***Додатку Г (1)***. У цьому випадку це шаблон для ситуації, коли фраза користувача є питанням, яке передбачає список деяких сутностей як відповідь. Це дуже типовий випадок.

2) Опис загальної структури шаблону. Характеристики відповідного вхідного виразу визначені в тегу `<input_expression>`. Цей шаблон можна застосувати у випадку, якщо тип фрази визначено як запитання (`<type>question</type>`), у якому запитується список об'єктів (`<subtype>objects</subtype>`) з умовою наявності таких слів-маркерів: «який», «яка», «яке», «які», що помічено в тегу `<marker_words>`.

Шаблон побудови відповіді описано в розділі `<output_structure>`. У тегу `<order>` описаний звичайний порядок формування частин відповіді. Кожна частина описана в тегу `<part>`. Атрибут `id` вказує на порядок частин. Типовий параметр вказує на тип і спосіб формування замінної частини. Він може мати наступні значення:

`"place_circumstance"` – додаткова обставина з вихідної фрази зі значенням місця;

`"genitive_circumstance"` – додаткова обставина з вихідного словосполучення зі значенням іменника в родовому відмінку. Наявність такого компонента є досить рідкісним випадком, оскільки іменники в родовому відмінку часто входять в іменні групи;

`"accusative_circumstance"` – додаткова обставина з вихідного словосполучення зі значенням іменника в знахідному відмінку;

"instrumental_circumstance" – додаткова обставина з вихідного словосполучення зі значенням іменника в орудному відмінку;

"dative_circumstance" – додаткова обставина з вихідного словосполучення зі значенням іменника в давальному відмінку;

"verb_circumstance" – додаткова обставина з вихідної фрази, виражена дієсловом;

"match_predicate" –предикат відповідності або невідповідності;

"main_instance" – основний концепт з оригінальної фрази;

"introductory" – місце для вставки шаблонного тексту;

"query_results" – безпосередні результати запиту до онтології.

Також можливі інші варіанти елементів шаблону, які залежать від мови та можуть додаватися в міру розвитку системи. Послідовність елементів шаблону побудована з урахуванням особливостей української мови для створення максимально нейтральної, стилістично витриманої фрази. Якщо будь-яка з перелічених частин відсутня в оригінальній фразі, її потрібно пропустити у генерованій відповіді.

Інші атрибути тегу <part> вказують на форму поняття, що підставляється, а також атрибут "preposition" визначає відповідний прийменник. Значення "as_input" цього атрибута говорить про те, що ця характеристика даної частини мови така ж, як і у вхідній фразі. Значення "conform" вказує на необхідність узгодження цієї частини з поточною ситуацією (наприклад, якщо об'єктів кілька, то поставити слово у форму множини, якщо тільки об’єкт тільки один, то у форму однини).

У частинах типу "introductory" можлива наявність декількох варіантів для підстановки. Вони знаходяться в тегах <options>. Значення атрибута random="true" вказує на можливість випадкового вибору програмою однієї з перерахованих опцій. Також у опції для форм однини та множини можуть бути явно вказані в тегах <single> і <plural>. Частина "query_results" може бути обмежена за допомогою атрибута "limit".

Також можна вказати значення атрибута класу, яке обмежує вивід лише тими сутностями, які походять до цього класу в ієрархії онтології. Можливо перерахувати кілька назв класів розділених комами без пробілів.

У розділі `<exceptions>` описана нестандартна ситуація синтезу відповіді.

У розділі `<no_results>` описаний порядок побудови відповіді для випадку відсутності результатів формального запиту.

Атрибут `"action"` визначає тип дії програми в ситуації, що виникає. Він може мати значення `"replace"`, що означає, що будується цілком нова відповідь, або `"insert"`, що говорить про вставку додаткових частин до тексту поточної відповіді. Можливе розширення іншими значеннями. У випадках значення `"insert"` у тегах `<part>` присутній атрибут `"insert"`, який вказує на те, *перед* якою наявною частиною з відповідним значенням атрибута `id` слід підставити дану частину, що розглядається. Для вставки в кінець використовується значення `"end"` (за замовчуванням).

Незважаючи на те, що розроблена система в першу чергу орієнтована на українську мову, запропонована методика може бути перенесена на інші мови. Це призведе до зміни структури шаблонів, але принципи та методологія залишаться близькими.

Приклад шаблону для випадку запиту часу події (для англійської мови)

У ***Додатку Г (2)*** наведено приклад шаблону для англійської мови. Цей шаблон використовується для відповіді на запитання про час події. Визначальними словом є «when» («коли»).

1) Особливості шаблону відповіді для англійської мови. При збереженні загальної структури, англійський шаблон має свої особливості. Перша з них полягає в тому, що англійська мова, на відміну від української чи російської, загалом не є мовою флективного типу. В англійській мові, наприклад, майже немає відмінків іменників, а дієслова не змінюються за родами. Тому ці атрибути видаляються з тегів англійського шаблону. Також

в англійській мові є свій специфічний порядок слів, який відрізняється від української та російської і є менш гнучким.

Незважаючи на те, що англійська мова не має відмінків іменників, семантично можна визначити частини речення, що описують різні ролі предметів. Вони займають своє місце в структурі речення, а роль уточнюють прийменники. Таким чином, ми також маємо різні типи додаткових обставин, схожих на ті, які зазначені у попередньому українському шаблоні.

2) Структура шаблону відповіді. Розглянемо структуру шаблону. Характеристики фрази користувача, до якої прийнятний цей шаблон, наведені в розділі <input_expression>. Він складається з наступних частин:

<type> - в даному випадку «question» («питання»);

<subtype> - тип питання, в даному випадку «питання часу»;

<marker_words> - слова-маркери для часового питання — “when” and “in what time” («коли» і «в який час»).

Структура форми відповіді описана в розділі <output_structure>. Розділ <order> містить інструкцію щодо порядку частин відповіді. Перша частина — "main_instance" — слово або іменник, до якого спрямоване питальне слово. Потім слідує "match_predicate", яке робить речення відповіді позитивним або негативним. Наступною частиною є "verb_circumstance", яка фактично є предикатом питального речення, вираженого дієсловом. Його число і час мають відповідати таким у вхідній фразі. Потім йде "accusative_circumstance". Ця частина виражає предмет, з яким або на якому виконується дія, наприклад: “play piano” («грає на піаніно»), “read a book” («читає книгу»), “make dinner” («готує вечерю»). Після нього слідує «instrumental_circumstance», що уточнює, за допомогою чого або чим виконується дія. Наприклад: “write a letter by hand” («пише листа від руки»), “paint with a brush” («малює пензлем»), “study by microscope” («вивчає під мікроскопом»). Ця сутність часто має прийменник

“by” або “with”. Вибір правильного прийменника залежить від контексту і правил погодження прийменників у англійській мові.

Наступна частина – "dative_circumstance". Вона вказує на предмет або особу, до якої спрямована дія. Наприклад: “give books to students” («роздати книжки студентам»), “make a party for children” («влаштувати свято для дітей»).

Остання частина, яку слід взяти з вхідної фрази, це "place_circumstance", яка описує, де або в яких умовах відбувається дія, наприклад “on the street” («на вулиці»), “in our shop” («у нашому магазині»), “in the university” («в університеті»).

Потім у шаблоні йде вступна частина. У цьому випадку це прийменник, який слід поставити перед виразами часу. У нього є кілька параметрів, але значення атрибута "random" тут "false". Це означає, що параметри не повинні вибиратися випадково, а відповідати наступному виразу часу.

Далі є частина "query_results", де перераховані результати формального запиту. Ці результати повинні бути підкласами «часу» в ієрархії онтології. На це вказує на значення "time" атрибута "class".

Розділ <no_results> у <exceptions> є інструкцією для побудови відповіді для випадку, коли формальних результатів запиту не отримано. Тут можне стояти кілька різних варіантів фраз з цього приводу. Особливістю шаблону «запитання про час» є спосіб отримання запропонованих тем. Вони повинні бути взяті з онтології при виконання формального запиту. Цей запит має запитувати класи, для яких задано властивість часу. Цей список результатів обмежений, у розглянутому шаблоні обмеження становить 5 позицій. Якщо результатів забагато, залучається розділ <overlimit>.

Приклади формування відповідей за шаблонами для різних семантичних типів наведені у ***Додатку Д***.

Загальний принцип формування фрази для відповіді за іншими типами шаблонів аналогічний. Відмінність полягає в послідовності частин утворюваної фрази, словоформах замінених слів, стандартних фрагментах

тексту. У випадку, якщо під час семантичного аналізу вихідного тексту виділено кілька типів фраз, то побудова відповіді відбувається послідовно за кількома шаблонами. Результатом, у цьому випадку, є набір фраз, кожна з яких є відповіддю або коментарем до відповідного типу фрази користувача. Недоліком підходу з шаблонною підстановкою порівняно з використанням готових відповідей є те, що він застосовний, перш за все, для видачі досить простих і коротких відповідей. Як зазначалося вище, підготовлена відповідь може містити як довгий детальний текст, що містить, за необхідності, графічні матеріали, таблиці, математичні формули тощо. Тому для підвищення ефективності системи можливе поєднання обох підходів: деякі з відповідей надходять у готовому вигляді (контексти), а деякі формуються за гнучкими шаблонами. Таким чином, система може містити ряд готових відповідей, які надають довгу і складно структуровану детальну інформацію. Якщо результат запиту вказує на таку відповідь, вона і надається користувачеві. Прості та лаконічні відповіді не готуються заздалегідь, а формуються вищеописаним способом. Отже, якщо результат формального запиту не вказує на конкретну готову відповідь або їх групу, а повертає набір понять, то включається описаний механізм генерації відповіді за гнучкими шаблонами.

Окрім складно структурованих, детальних відповідей, система також може містити підготовлені відповіді для типових стандартних ситуацій, коли відповідь не потребує звернення до бази знань і побудови формального запиту (що вимагає значних ресурсів). До таких випадків відносяться, наприклад, фрази вітання, прощання, вибачення, вираження подяки, образи тощо. Якщо фраза користувача ідентифікується як один із цих випадків, то швидка відповідь (випадкова із заздалегідь заготовленого списку) видається для зазначеної для неї ситуації. При цьому не будуються формальні запити і не здійснюється доступ до бази знань, також детальний аналіз фрази користувача може не проводитися. Це істотно розвантажує систему і прискорює реагування на стандартні ситуації.

3.3 Особливості формування природномовних відповідей на базі онтології, створеної автоматично на базі синтактико-семантичного аналізу тексту

3.3.1 Особливості синтезу відповідей на запит до семантично структурованої бази знань

Яз зазначалося у попередньому розділі онтологію тексту на природній мові флективного типу можна створити автоматично на базі результатів синтактико-семантичного розбору. Результатом такого розбору є окремі поняття зв’язані між собою семантичними предикатами відповідно до того, як вони були пов’язані у реченнях тексту. Формування природномовних відповідей з результатів виконання формального запиту до такої онтології має свої особливості, які слід розглянути окремо.

Інтерфейс користувача діалогової системи, у якому відображаються лише результати виконання запиту, навіть і красиво оформлені, може виглядати не дуже дружнім, а часом і не зовсім зрозумілим. Тому важливою проблемою є синтез відповідей природною мовою. Загалом, при розробці способів формуванні відповіді діалогової системи доводиться балансувати між наданням готових контекстів і безпосереднім синтезом тексту. Так, наприклад, для подачі деяких таблиць і графічних та інших медіа матеріалів, що ілюструють відповідь, кращим варіантом буде використання готових контекстів, що містять посилання на відповідні файли. Контекстні відповіді можуть бути кращим варіантом при необхідності надання докладної розгорнутої інформації. Синтезовані відповіді надають більшої зручності сприйняття для більш конкретних питань, формальною відповіддю на які просто список сутностей з онтології.

3.3.2 Приклади шаблонів-інструкцій для синтезу відповідей для типових випадків

Нижче наведемо приклади шаблонів-інструкцій для синтезу відповідей для деяких типових випадків. Ці шаблони також надають користувачеві

контексти (речення), що ілюструють і підтверджують висловлювання. Шаблони нижче подаються у людино зрозумілій формі (свого роду метамові). В програмній реалізації вони являють собою програмні сутності (класи з методами) на мові Python, що приєднуються до системи у окремому файлі-модулі. Також робилася спроба додавати шаблони формування відповідей у вигляді XML-описів, що однак призвело до більшої складності реалізації і меншої продуктивності програмних модулів.

3.3.2.1 Запит про властивості об'єкту. Наведемо шаблон відповіді на питання про можливі властивості (характеристики) сутності – «Яким буває N?».

```
Повторювати для всіх result:
    якщо INPUT_VALUE іменник:
        INPUT_VALUE + може бути + result (узгодити рід)
        + context
    якщо INPUT_VALUE дієслово:
        INPUT_VALUE + можна + result
        + context
```

Для визначення характеристик слів (частина мови, рід і т.п.), а також для постановки слів у відповідну форму, використовується програмна бібліотека морфологічного розбору PyMorphy2 [103]. У наведеному вище простому прикладі перевіряється частина мови для INPUT_VALUE. Воно може бути іменником або дієсловом. У разі іменника, значення result, що може бути прикметником або дієприкметником потрібно узгодити за граматичним родом з INPUT_VALUE.

3.3.2.2 Запит про локалізацію об'єкту. Розглянемо складніший приклад. В даному випадку предметом запиту є локалізація об'єкта за набором можливих предикатів локалізації. При цьому конкретний предикат локалізації не визначений у вихідних параметрах запиту, а повертається у

його результатах. Як згадувалося вище, конкретизований семантичний предикат може бути використано при синтезі тексту відповіді.

Повторювати для всіх *result*:

INPUT_VALUE + знаходиться +

якщо *predicate* = "локалізація у множині":

+ серед + *result* (*множина, родовий відмінок*)

+ *context*

якщо *predicate* = "локалізація наближена":

+ біля + *result* (*родовий відмінок*)

+ *context*

якщо *predicate* = "локалізація об’єктна":

+ на + *result* (*місцевий відмінок*)

+ *context*

якщо *predicate* = "входження об’єктне":

+ у + *result* (*місцевий відмінок*)

+ *context*

якщо *predicate* = "локалізація між об’єктами":

+ між + *result* (*множина, орудний відмінок*)

+ *context*

якщо *predicate* = "локалізація за об’єктом":

+ за + *result* (*орудний відмінок*)

+ *context*

якщо *predicate* = "локалізація перед об’єктом":

+ перед + *result* (*орудний відмінок*)

+ *context*

якщо *predicate* = "локалізація під об’єктом":

+ під + *result* (*орудний відмінок*)

+ *context*

якщо *predicate* = "локалізація зверху":

```
        + над + result (орудний відмінок)
        + context
    якщо predicate = "локалізація всередині":
        + всередині + result (родовий відмінок)
        + context
```

У наведеному прикладі ми бачимо, що конкретний тип семантичного предикату (у даному разі локалізації) визначає відповідний прийменник та відмінок для значення змінної result.

3.4 Використання великих мовних моделей у комбінації із онтологічним підходом для синтезу природномовних текстів

3.4.1 Застосування великих мовних моделей для генерації відповідей на базі переданих природномовних контекстів

Хоча великі мовні моделі є генеративними, тобто здатні синтезувати текст на основі заданих вхідних положень, використовуючи результати свого попереднього навчання, як згадувалося раніше у розділі 2, вони також можуть виконувати поставлені чіткі завдання. Таким завданням може бути генерація відповіді на основі переданих із бази знань контекстів і визначеного раніше набору намірів користувача на основі якого потрібно надати відповідь базуючись на переданих контекстах.

Названі концепції були втілені у системі, розробленій у рамках поточної роботи. Так, на основі виділених іменованих сутностей та їхніх семантичних ролей (намірів) система вибирає відповідні контексти з онтології контекстів. Далі до API ChatGPT надсилається запит, який містить попередньо вибрані контексти та списки намірів, а також відповідні сутності (суб’єкти та об’єкти), актуалізовані в підказках. Форма цієї підказки може значно відрізнятися та залежить від конкретного наміру чи списку намірів. Правила її створення, включаючи обов’язкові поля та підказкові фрази,

визначені в метаонтології. Метаонтологія також містить правила для остаточного представлення результатів, залежно від типів залучених полів.

Створення мета-онтології включає мета-навчання, яке охоплює розробку та тонке налаштування відповідних підказок. Цей ітеративний процес включає інженера знань і “playground” ChatGPT. Спочатку інженер знань формулює структуровану підказку у форматі JSON (XML, YAML тощо), яка складається з ключів і фраз підказки, розроблених із певною метою, спираючись на минулий досвід швидкої розробки та загальні знання. Це підказка потім надається ChatGPT, і відповідь аналізується, щоб визначити, чи вона адекватно задовольняє намічену мету. Якщо відповідь ChatGPT неправильна або має недоліки, до початкового запиту вносяться зміни. Ці зміни можуть передбачати додавання додаткових полів, видалення зайвих або неефективних аспектів підказки та редагування фраз підказки для підвищення чіткості завдання. Покращена підказка потім передається назад у ChatGPT для подальших ітерацій. Цей ітераційний процес триває до тих пір, поки отримана відповідь не збігається з бажаним і очікуваним результатом. В такому разі можна казати про створену і налаштовану інструкцію-підказку, яка дозволяє великій мовній моделі генерувати найбільш якісний і релевантний встановленим вимогам текст відповіді.

3.4.2 Зворотній синтез природномовних речень на основі онтологічного представлення

3.4.2.1 Задача зворотного синтезу тир одномовного речення на основі онтологічного представлення. Як зазначалося у розділі 2, автоматично створена на основі природномовного тексту онтологія містить сутності та семантичні зв’язки що їх поєднують. Ці зв’язки, конкретизовані сутностями пов’язані із групами висловів, а ті в свою чергу із реченнями вхідного тексту. Таким чином, маючи подібне семантичне представлення можна сформулювати назад природномовне речення. Для цієї задачі може бути застосовано метод гнучких шаблонів, як це описано у підрозділі 3.3. Проте із розвитком підходів, заснованих на використанні нейронних мереж

глибокого навчання, апофеозом яких стали великі мовні моделі-трансформери, такі як ChatGPT, постає завдання розгляду саме їх у якості інструменту для синтезу природномовних речень на основі семантичних структур. Така спроба була зроблена у рамках нашої роботи.

У якості тестової онтології була взята база знань створена на основі тексту «Склад обчислювальної системи». Так як база знань зберігалася під управлінням графової СУБД Neo4J, мовою запитів до неї є Cypher.

3.4.2.2 Формальний запит до онтології для повернення семантичного представлення речення. Нижче наведено текст тестового запиту до вказаної онтології для отримання тексту певного речення (для порівняння результату) за його ID, а також відповідного набору семантичних категорій та пов'язаних ними пар понять (основного і залежного) для даного речення. Результати такого запиту слугували вхідними даними для задачі зворотного синтезу природномовного речення:

```
MATCH (inp:Relationship)-[:SPO]->(inp_type:Relationship),
      (inp:Relationship)<-[:SPO]-(linked_group:Relationship),
      (linked_group:Relationship)-[:SPO]-
>(linked_group_type:Relationship),
      (linked_group:Relationship)<-[:SPO]-
(certain_words_link:Relationship),
      (certain_words_link:Relationship)-[:SPO]-
>(sem_type:Relationship),
      (sem_type:Relationship)-[:SPO]-
>(w_link_type:Relationship),
      (certain_words_link:Relationship)-[:DOMAIN]-
>(main_entyty:Class),
      (certain_words_link:Relationship)-[:RANGE]-
>(dependent_entyty:Class)
WHERE
    inp_type.name = "SentenceGroups" and
    linked_group_type.name = "Groups" and
    w_link_type.name = "WordsLink" and
    ID(inp) = вказати ID речення
RETURN DISTINCT ID(inp) as id, inp.label as text, main_entyty.la
bel as main_entyty,dependent_entyty.label as dependent_entyty, s
em_type.label as sem_type;
```

3.4.2.3 Вхідні дані для зворотного синтезу природномовної фрази за допомогою великої мовної моделі. Приклад результату виконання такого запиту наведено у таблиці 3.1.

Таблиця 3.1 – Приклад результатів виконання запиту семантичної структури речення

Текст речення	Головна сутність	Залежна сутність	Семантичний тип
Принтер призначений для створення твердих копій документів.	принтер	призначений	іменний присудок
	принтер	створення	призначення
	створення	для	прив’язка прийменника
	створення	копія	об’єкт
	копія	документ	приналежність
	копія	твердий	властивість об’єкту

Поняття (слова) із речення представлені парами сутностей пов’язаних семантичними предикатами. Можна звернути увагу, що у вищенаведеному прикладі слово «призначений» не представлене як сутність. Натомість воно стало основою для семантичного предикату приналежності, що поєднує сутності «принтер» і «створення». Це приклад особливостей результатів семантичного аналізу вихідного тексту при автоматизованому створенні онтології. Також у онтології додатково зберігається предикат «прив’язка прийменника». То є додатковим уточненням, що вказує на те, що дане слово у реченні використано саме з цим прийменником.

Представлена семантична структура виглядає достатньою для побудови вихідного природномовного речення відповідного змісту.

Для запуску задачі синтезу великій мовній моделі (ChatGPT) потрібно надати відповідну інструкцію-підказку. Як вже згадувалося вище, для таких інструкцій краще використовувати англійську мову. Сама інструкція структурована у форматі JSON. У ***Додатку Е*** наведено відповідний текст інструкції-підказки.

У розділі наведеної інструкції “`Intriduction`” задаються початкові установки для великої мовної моделі щодо її подальшого поводження і дається базове роз’яснення вхідних даних.

У розділі "Action to perform" формулюється безпосередньо завдання для виконання.

Розділ "Restrictions" дає додаткові вказівки щодо формованого тексту на виході і намагається усунути неоднозначність тлумачення інструкції.

Розділ "Additional data to provide" в даному випадку слугує для вказування моделі провести власну оцінку якості виконання завдання.

У розділі "The essence of the syntactic-semantic relationship names and meaning explanation" представлено словник, що надає пояснення до тлумачення конкретних типів семантичних зв'язків і як їх використовувати при побудові результуючого речення. Оскільки в онтології існує досить велика кількість семантичних категорій, а довжина вхідних повідомлень для ChatGPT обмежена за кількістю символів, то на практиці об'єм такого словника можна обмежити лише наявними у даному реченні семантичними категоріями.

Безпосередньо пари сутностей і відповідні семантичні зв'язки, що їх поєднують перераховуються у списку словників "Input data".

На виході у якості результату отримуємо сформоване природномовне речення і власну оцінку великої мовної моделі (ChatGPT) вірогідності того, що речення було сформовано вірно і відповідає оригіналу (про вигляд якого модель не знає).

Експеримент і оцінка ефективності роботи запропонованого методу представлена у розділі 5 цієї роботи.

3.5 Висновки за розділом 3

Розглянуто методи генерації відповідей чат-бота за результатами формального запиту до бази знань. Проаналізовано прийоми видачі готової відповіді та синтезу відповіді на основі понять, отриманих із результатів

формального запиту та фрази користувача за шаблоном, вибір якого базується на семантичному аналізі вихідної фрази.

Запропоновано метод ранжування готових відповідей для випадків, коли формальний запит не дає однозначного результату або потрібна додаткова інформація. Метод заснований на обчисленні метрики близькості вихідної фрази користувача до набору ключових слів або заголовка запропонованої відповіді.

Запропоновано приклад гнучкої структури шаблону для генерації відповідей на основі набору понять, отриманих шляхом виконання формального запиту до бази знань і слів, включених до фрази користувача. Наведено приклади шаблонів відповідей на запитання, що пропонують у якості відповіді перелік понять (для української мови) і час події (для англійської мови). Шаблон визначає місця підстановки окремих слів та їх груп з вихідного речення та їх словоформи, додаткові текстові вставки з можливістю їх варіювання, місця підстановки результатів запиту та обмеження, що накладаються на них. Шаблон також передбачає виняткові ситуації, наприклад, відсутність результатів запиту або коли їх занадто багато. Він також пропонує можливість підставлення текстових вставок в певних позиціях у разі особливих ситуацій.

Вказано, що результати формального запиту у вигляді готових відповідей можна комбінувати з генерацією відповідей на основі результатів у вигляді набору понять. Так, пропонується використовувати готові відповіді для випадків, коли необхідно надати розгорнуті відповіді (контексти), відповіді, що містять нетекстові матеріали (зображення, таблиці, аудіо- та відеозаписи), прості відповіді для типових ситуації (привітання, подяка, вибачення, образа, прощання тощо). Відповіді доцільно генерувати з набору понять за відсутності готової відповіді в онтології, в тому числі, якщо це недоцільно, наприклад, якщо запитувані дані часто змінюються. За допомогою запропонованого шаблону аналізується ряд прикладів фраз

природною мовою як відповідей на запитання користувача. Отримані фрази зрозумілі і, загалом, граматично правильні.

На підставі аналізу результатів синтезу речень за запропонованим шаблоном зроблено висновок, що в іменні групи під час розбору вихідної фрази слід включати всі іменники безпосередньо пов’язані між собою в будь-якому відмінку, за винятком відношення «підмет – іменний присудок». Це дозволяє правильно групувати їх при формуванні відповіді.

Запропоновано методи генерації природномовних фраз на основі даних, отриманих із автоматично створеної онтології на основі тексту, включаючи як використання шаблонів, так і великих мовних моделей. У якості вхідних даних виступають поняття із онтології та семантичні предикати, що їх поєднують. Метод, заснований на застосуванні великих мовних моделей використовує спеціально-створену інструкцію-підказку, що містить чітке формулювання завдання синтезу та роз’яснення суті переданих семантичних предикатів в аспекті генерації природномовних фраз.

РОЗДІЛ 4
ПРАКТИЧНА РЕАЛІЗАЦІЯ АНАЛІЗУ І СИНТЕЗУ ПРИРОДНОМОВНИХ ТЕКСТІВ ПРИ РОЗРОБЦІ ДІАЛОГОВИХ І ДОВІДКОВИХ СИСТЕМ

4.1 Вступ

4.1.1 Природномовний інтерфейс

Інтерфейс користувача є важливою складовою в діалоговій системі. Так Досить поширеною є ситуація, коли користувач має отримати якусь інформацію від системи (наприклад, з її бази даних) або система має отримати якусь інформацію чи інструкції від користувача. Відомо, що пересічний користувач не зовсім знайомий ні з програмуванням, ні з формальними мовами запитів. Таким чином, є два основних варіанти: перший – розробка інтерфейсу з попередньо підготовленими графічними елементами управління, такими як кнопки, меню і так далі, які прив'язані до визначених процедур програми; другий – зробити природномовний інтерфейс, який виключає мінімум традиційних елементів управління, але основні інструкції користувач повідомляє системі природною мовою (українською, англійською, норвезькою тощо). Перший варіант на сьогоднішній день залишається найпопулярнішим, оскільки він досить простий і зрозумілий у своїй розробці, а функціонування такого інтерфейсу достатньо передбачуване. Але якщо управління системою ускладнюється, то зростає і складність інтерфейсу. Якщо це інтерфейс бази даних, він стає або дуже обмеженим у можливих параметрах або потребує довгих і складних меню. Якщо система призначена для отримання інформації від людини, її користувач стає досить обмеженим у варіантах відповіді. Таким чином, для обох зазначених типів діалогових систем природномовний інтерфейс буде більш зручним і бажаним.

4.1.2 Задачі при створенні природномовних інтерфейсів

Побудова природномовних інтерфейсів передбачає вирішення ряду складних завдань, серед яких: синтаксичний і семантичний аналіз фраз на природній мові; перетворення семантичної інформації та отриманих сутностей у формальні запити та/або програмні інструкції; представлення результатів виконаних дій у доступній і зрозумілій формі; утримання в рамках поточного контексту діалогу.

У цьому розділі даної роботи ми намагаємося представити та проаналізувати наш досвід побудови природномовних діалогових програмних систем різного типу та призначення. Також деяку увагу приділено проблемі збереження предмета (контексту) розмови і її особливостям і важливості для різних типів діалогових систем.

4.2 Огляд типів діалогових систем за їх функціонуванням і призначенням

4.2.1 Призначення діалогових систем

Діалогова програмна система по своїй суті є формою програмного інтерфейсу, призначеного для взаємодії з користувачем, імітуючи розмову. Тип такого інтерфейсу, який розглядається в даній роботі, реалізує спілкування між користувачем і програмою на природній мові. Користувач може, залежно від призначення та реалізації конкретної системи, виконувати одну або кілька наступних дій: поставити запитання, попросити (наказати) програмі виконати певну дію, відповісти на запитання, яке дає система, висловити розповідну фразу, яка не є явним запитанням чи наказом, а передбачає відповідь програми (коментар, порада чи дія відповідно до описаної ситуації). Система може виконувати наступні дії: дати відповідь на запитання, поставити запитання, виконати певні дії (якщо вони можливі та передбачені цією системою), дати коментар чи розповідь.

У певній програмній системі перераховані види дій можуть бути реалізовані лише частково і вибірково, залежно від її призначення. Природномовна взаємодія між програмою та користувачем на даному рівні розвитку техніки може здійснюватися як у письмовій, так і в усній формах. У цій роботі розглядається лише письмовий тип взаємодії, оскільки усну взаємодію фактично можна звести до письмової за допомогою програмних рішень, які реалізують розпізнавання та синтез мовлення. Наразі досягнуто значних успіхів у розвитку таких трансформаційних процесів, і їх подальший розвиток триває [88, 91]. Однак ці методи є окремою галуззю знань і тут не розглядаються.

Версії діалогових систем, що працюють з письмовою мовою, при бажанні можна адаптувати для роботи з усним мовленням, додавши до них модулі розпізнавання мовлення в текст і синтезу мовлення, тобто генерування відповідного звуку на основі тексту.

4.2.2 Варіанти діалогових систем

Залежно від функціональності, реалізованої в конкретній діалоговій системі, як з боку користувача, так і з боку програми можливі різні варіанти. Кожен з них має свою область застосування та особливості завдань, які вирішуються при його розробці. Розглянемо кілька типових варіантів.

Довідкова система «питання/відповідь». Користувач може задавати питання, а також вводити наказові словосполучення (запити) зі значенням запиту інформації, але не більше. Система лише відповідає на запитання користувача, але не ставить запитань, не надає жодної інформації без запиту користувача, не запускає жодних процесів, не пов’язаних з пошуком та наданням інформації. Цю версію можна назвати довідковою системою «запитання/відповідь», природномовним інтерфейсом до бази даних або їх комбінацією.

Інтерактивна довідкова система. Користувач може задавати запитання та вводити наказові фрази, які мають мету запиту інформації.

Система не тільки дає відповіді на запитання користувача, а й задає питання, але обмежені. Суть обмеження полягає в наступному: питання, які задає програма, мають скоріше технічну мету, наприклад, уточнення семантики питання користувача, запит предметної області. Можливі також запитання суто технічного характеру, наприклад «Чи бажаєте ви завершити діалог?», «Чи задоволені ви отриманою відповіддю?». і т. д. Це, по суті, теж довідкова система, але доповнена можливістю доопрацювання для підвищення релевантності відповідей.

Діалогова система для анкетування. Система, яка задає запитання користувачеві. Відповіді на інтерфейс користувача або взагалі не маються на увазі, або є чисто формальними та супроводжуються запитанням. Також можна отримати результуючу відповідь системи після серії запитань. Така відповідь системи базуватиметься на колекції відповідей, наданих користувачем. Користувач, у свою чергу, може лише відповідати на поставлені системою запитання, які в рамках розглянутої парадигми відбуватимуться природною мовою. Такий тип реалізації діалогової системи може бути прийнятним для систем автоматичного анкетування, тестування чи діагностики.

Система активного діалогу. Користувач може як задавати питання системі, так і відповідати на питання, що надходять з неї. Типи дій, які може виконувати система, схожі. Також додатковими типами дій у цьому варіанті є введення та виведення оповідальних фраз без питального чи наказового значення, які, у свою чергу, можуть коментуватися, ігноруватися, а також супроводжуватися запитанням, проханням чи розповідною фразою. Це тип системи «віртуального компаньйона», типовий розмовний чат-бот.

4.2.2.5 Природномовний інтерфейс керування. Користувач дає вказівки системі у формі імперативних операторів, спрямованих на початок або припинення певної дії. Також за бажанням користувач може задавати питання, але тільки про стан виконання запущеного процесу або про завершення дії. За вказівками користувача, залежно від свого призначення,

система виконує певну дію або запускає процес. Це може бути як чисто програмний процес на комп'ютері, так і дії, що виконуються над навколишніми об'єктами («ввімкнути/вимкнути пристрій», «змінити гучність звуку», «запустити/зупинити технологічний процес» тощо). Дія виконується, якщо це технічно можливо і передбачено цією системою. Також така система може опціонально давати відповіді природною мовою, пов’язані з фактом початку/закінчення дії, статусом процесу, результатами виконання, можливістю виконати запитану дію тощо. Це, по суті, інтерфейс системи управління. Цей тип не розглядається в рамках поточної роботи, оскільки він, по суті, є різновидом типів 1 і 2, доповненим запуском процесу та керуванням ним. Запуск програмою процесів на комп’ютері та керування ними відноситься до системного програмування, а керування об’єктами матеріального світу – до автоматизації та робототехніки, що виходить за межі нашого дослідження.

Розглянемо більш докладно означені типи діалогових систем.

4.2.3 Довідкова система

4.2.3.1 Загальна схема діалогової системи типу «питання/відповідь». Загальна схема діалогової системи типу «питання/відповідь» виглядає наступним чином. Повідомлення користувача аналізуються для вилучення значущих сутностей. Також можна провести семантичний аналіз фрази користувача. Його результатом є певний семантичний тип вхідної фрази, який потім може бути використаний для наступних цілей: визначення схеми виділення суттєвих понять, вибір парадигми побудови формального запиту, вибір онтології, на яку буде направлено сформований формальний запит, визначення форми відповіді, яка буде представлена користувачеві. Потім на основі отриманих даних будується формальний запит до бази даних або пакет запитів. Отриманий формальний запит або набір запитів передається на виконання сервісу, який взаємодіє з СУБД. Ця служба також може виконувати деякі додаткові запити

до бази даних. Це можна зробити для наступних цілей: запити з сутностями, які зазнали скорочення, якщо результати запиту не отримані (таким чином запитуються більш загальні поняття або тісно пов’язані поняття, що збільшує ймовірність отримання відповіді, хоча, можливо, менш релевантної); запит додаткової технічної інформації, яка може бути використана, наприклад, для ранжування відповідей (такою інформацією, наприклад, може бути повторення запитуваних понять у тексті відповіді, якщо відповідь є певним фрагментом тексту); отримання додаткової інформації, що стосується запитуваної інформації, яка також може зацікавити користувача (в цьому випадку, наприклад, до відповіді може бути доданий розділ «дивись також по темі» або «можливо вам буде цікаво», який містить у згорнутому вигляді відповідні інформаційні елементи, отримані таким чином).

4.2.3.2 Обробка і відповідей. Подальші дії будуть пов’язані з характером відповідей, отриманих із цієї бази даних. Найпримітивніший варіант – просто повернути отриманий набір результатів користувачеві як відповідь. Але це не завжди прийнятно. Таким чином, результатом запиту може бути просто посилання або набір посилань на тексти відповідей, отримані з іншої (документної) бази даних. Також відповідь може містити медіа контент – зображення, відео та звукові вставки, а також зовнішні посилання до іншого джерела. У цьому випадку необхідно провести додаткову обробку тексту відповіді з метою заміни у відповідних позиціях зазначених у ньому відповідних матеріалів або посилань на них. Також не виключений випадок отримання кількох відповідей в результаті запиту. У цьому випадку подальші дії залежать від очікуваної семантики. Таким чином, у ряді ситуацій питання користувача пропонує перелік певних сутностей як відповідь, наприклад "Які журнали існують для проблеми N.?" або «Від яких письменників N. отримував листи в 1968 р.»? У цій ситуації зі списку отриманих результатів формується перелік, який підставляється у відповідь. Аналогічна ситуація, коли передбачається формування відповіді з кількох частин. Наприклад, запит повертає в результаті певний набір сутностей

(заголовок і основний текст; опис події, дата і місце тощо). Зовсім інша ситуація, коли в результаті виходить готовий текст відповіді (визначення поняття, детальне пояснення, інструкція тощо). У цьому випадку необхідне ранжування отриманих відповідей за релевантністю. Також ранжуванню підлягають додаткові відповіді, отримані шляхом виконання додаткових запитів, якщо такі є. Процес ранжування може відбуватися навіть на етапі формування та виконання запитів. Таким чином, запит, зроблений з використанням більш скорочених сутностей, матиме нижчий рейтинг. Також можливо, що передбачається декілька типів запитів. Вони ранжуються відповідно до зменшення ймовірної релевантності. Наступний виконується, якщо для попереднього немає результатів, або просто для отримання додаткової пов’язаної інформації. Наприклад, питання виглядає так: "Що таке N.?" Для нього передбачено ряд видів формальних запитів: запит на визначення поняття N.; запит на роз'яснення поняття N.; запит класу, до якого належить поняття N; запит для суб'єктів, пов'язаних з поняттям N. будь-яким чином. Очевидно, що потенційна релевантність у цих семантичних типах варіантів, пов'язаних з цим питанням, відповідно, зменшується.

4.2.3.3 Відображення результатів у інтерфейсі користувача. Після отримання відповідей на запити та їх ранжування (за необхідності) наступним кроком є надання цієї інформації користувачеві. Одна з можливих операцій вже була названа – заміна медіа контенту, якщо такий є. Якщо таким чином отримано повний готовий текст відповіді, то він просто передається в інтерфейс користувача, туди ж переноситься додаткова інформація, але для відображення в згорнутому вигляді, який при бажанні можна розгорнути. Однак бувають ситуації, коли отриманий результат є не повним текстом, а лише набором сутностей. Надати таку відповідь безпосередньо користувачам можливо, але це буде виглядати неохайно і недостатньо доброзичливо. Тому більш прийнятним варіантом в цій ситуації є генерація фрази для відповіді. Для цього використовуються такі дані:

семантичний тип запиту користувача, а точніше тип формального запиту на його основі; слова з оригінальної фрази користувача; попередньо визначені елементи фрази (вибрані на основі семантичного типу); самі результати формального запиту.

4.2.3.4 Робота із базою знань

Робота з кількома онтологічними графами. Розглянемо більш детально такі складові вищезазначеного процесу, як семантичний аналіз вихідної фрази та формування формального запиту, оскільки ці етапи відіграють у ньому ключову роль і визначають релевантність відповіді. База даних системи може являти собою одну онтологію (один граф) або набір з кількох. Другий тип підходить, коли є кілька різних предметів або тем, яким присвячена система. Кілька менших графів у цій ситуації є більш прийнятними, оскільки операції з ними можуть виконуватися швидше, а ймовірність надання інформації від сторонніх суб'єктів значно менша. Але якщо база даних має декілька графів, виникає проблема вибору найбільш підходящого для виконання певного запиту. У наших системах вона вирішується досить простим способом. Кожна з онтологій має набір ключових слів, пов'язаних з нею, які представляють поняття та терміни, включені в неї. Ці списки досить довгі – від десятків до сотень слів, залежно від розміру графу. Ці списки ключових слів необхідно порівняти зі списком сутностей, вибраних із вхідного запитання. Кожен із цих термінів необхідно перевірити, чи присутній він у списку ключових слів. Необхідно вибрати ту онтологію, список ключових слів якої містить більшість вхідних понять. Якщо це число стає рівним для кількох онтологій, слід вибрати ту, список ключових слів якої містить менше інших понять, тобто є коротшим. Це зменшує ймовірність появи пошукового шуму. Також може бути попередній етап, на якому вибирається певна група онтологій. Для цього використовуються слова безпосередньо з вхідної фрази. Наявність чи відсутність там деяких слів залишає лише кілька онтологій для наступного вибору описаним вище методом.

Варіант діалогової системи з уніфікованим шаблоном запиту. Окрім вибору найбільш релевантного графу, дуже важливою частиною семантичного аналізу є визначення семантичного типу питання. Однак на практиці це дійсно потрібно не у всіх парадигмах діалогових систем. Ми створили та протестували кілька діалогових систем, які фактично використовують лише один універсальний шаблон запиту для кожного випадку. Опишемо цей спосіб, який має як переваги, так і недоліки. По-перше, це структура онтологічного графа, яка коротко викладається наступним чином: граф містить поняття, розбиті на слова, класифіковані на кілька груп відповідно до частин мови; на нижчих рівнях ієрархії (підкласах) є групи імен, які розширюють батьківське поняття, може бути будь-яка кількість рівнів ієрархії, на кожному наступному понятті міститься ще одне додаткове слово; в онтології є посилання на готові відповіді, які є нащадками відповідної сутності «Готова відповідь»; є група властивостей, що успадковує «Основну властивість», яка прив’язує набори (перетини) присутніх в онтології концептів до відповідного посилання на готову відповідь, де DOMAIN у наборі концептів і RANGE є посиланням на відповідь текст. Також у графі зберігається деяка додаткова технічна інформація, наприклад дані про повторення певних понять у відповідних текстах, яка використовується як один із факторів ранжування відповідей. З фаз користувача програма виділяє всі значущі поняття, крім допоміжних слів. Усі сутності в лематизованій формі використовуються для підстановки в шаблон запиту. Потім виділений набір слів піддається редукції. В результаті кожен наступний запит спрямований на отримання ширшого поняття, ніж запит із початковим набором слів. На цьому етапі виникає проблема порядку редукції, тобто видалення термінів, які призводять до більшої чи меншої втрати значення та можливого неправильного тлумачення. Також тут виникає інша проблема: якщо початкова фраза і витягнутий таким чином набір слів досить довгий, то буде велика кількість варіантів скорочення. Обробка всіх можливих варіантів може зайняти досить багато

часу і ресурсів, що небажано для діалогових систем, що працюють в режимі реального часу. У наших проектах ми, спираючись на практичний досвід і можливу семантичну цінність різних типів слів, запропонували наступну схему скорочення. Виконується кілька рівнів скорочення, кожен з яких вважається, можливо, менш відповідним: 1 – видалення лише дієслів; 2 – видалення лише числівників та номерів; 3 – вилучення лише питальних і узгоджувальних слів; 4 – вилучення дієслів і питальних слів; 4 – вилучення дієслів, чисел, питальних, узгоджувальних і сполучних слів; 5 – вилучення прикметників; 6 – вилучення прикметників, питальних і зв'язувальних слів; 7 – вилучення іменників та прикметників; 8 – залишаються лише дієслова та числа; 9 – залишаються лише дієслова. Потім програма виконує серію випадкових скорочень за допомогою простого скорочення фрази без прив'язки до частин мови. Цей етап може бути досить тривалим, тому як скорочення фраз користувача зазвичай не дуже жадібний до ресурсів процес, на відміну від виконання запитів. Тому на практиці часто потрібно виконувати не всі запити, сформовані за допомогою зазначених скорочень. Якщо потрібна лише одна відповідь, процес може бути зупинений після отримання першої не порожньої відповіді на запит. Якщо необхідно отримати додаткову інформацію, буде достатньо для виконання тієї кількості запитів, яка дасть бажану кількість нееквівалентних відповідей. Цей нескладний підхід був реалізований і протестований нами для кількох діалогових довідкових систем і показав досить прийнятні результати: він витягував відповіді, пов'язані принаймні з кількома поняттями, а також витягував більш-менш пов'язану інформацію (як після одного запиту, так і після їх серії). Крім того, отримані таким чином відповіді (за винятком довгих наборів відповідей, повернутих виконанням одного запиту) виглядають заздалегідь організованими. Після наступного ранжування найбільш релевантні відповіді знаходяться на початку.

Переваги і недоліки уніфікованого шаблону запиту. Перевагами такого підходу є: простота – глибокий семантичний аналіз і багато шаблонів

запитів не потрібні; універсальність – потрібна досить проста структура онтології, побудова якої з текстів природною мовою може бути досить легко автоматизована; низька залежність від структури мови при високій здатності отримати відповіді (більш-менш релевантні) у більшості випадків, пов’язаних із предмет онтології. Однак метод також має деякі вагомі недоліки, серед яких такі: він націлений на певну просту структуру онтології, тому він не в змозі мати справу зі складними семантичними графами розширеної структури, створеними окремими особами та групами на основі їх власного бачення предмета і не знайомими з описаним вище підходом; він позбавлений семантичної спрямованості, і це призводить до деяких недоліків – іноді для отримання хоч якоїсь інформації потрібен досить довгий набір запитів, що займає багато часу; немає гарантії релевантності семантичних відтінків, особливо при використанні скорочених наборів понять.

Особливості роботи із семантично структурованими онтологічними графами. Таким чином, для роботи з більш складними семантичними графами був розроблений більш прогресивний підхід. Тут ми даємо його короткий опис. Передбачається, що терміни у вершинах графа не розподілені на окремі слова та групи, а можуть бути розгорнутими концепціями, що містять до десятків слів. Існує багато типів зв'язків між вузлами графа, які відповідають семантичним типам. Типи семантичних зв’язків можуть бути загальними або специфічними для предмета, якщо може спростити для певного випадку онтологію та структуру запитів. Такий граф може зберігатися у форматах RDF/XML або Neo4j. Обмеження використання лише стандартних предикатів і типів OWL призводить до ускладнення як онтологій, так і запитів. Розглянемо приклад: просте твердження «антифрикційна втулка виготовлена з олов’яної бронзи» потрібно виразити формально – як частину онтологічного графа. Якщо ми хочемо використовувати лише стандартну методологію OWL, один із способів буде таким. Існує два класи «антифрикційна втулка» та «олов’яна бронза», вони можуть бути нащадками вищих класів ієрархії, наприклад «втулка» та

«бронза», відповідно, які також можуть бути нащадками більш високих класів «деталі машин» та «матеріал». Головною метою, яка тут розглядається, є семантичне зв’язування, яке не є ієрархічним. Тому в OWL нам потрібна властивість, це може бути щось на зразок «material» або «is made of» (щоб уникнути невідповідності з назвою відповідного класу). Ця властивість сама по собі є абстрактною або зв’язує лише вищі класи ієрархії, такі як «обладнання» та «матеріал» у цьому випадку. Для більш певного прив'язування сутностей він повинен мати нащадків, кожен з яких прив'язує певний продукт до свого матеріалу або матеріалів (якщо він виготовлений з кількох). Тут ця властивість може бути чимось на зразок «матеріалу антифрикційної втулки». RANGE цієї властивості буде «антифрикційна втулка», а її DOMAIN «олов’яна бронза», які є посиланнями на відповідні класи. Використовуючи RDF/XML, конструкція може бути простішою. Ієрархія класів залишається незмінно, але в цьому разі нам потрібен лише один довільний предикат «матеріал» або «зроблено з». Отже, маємо готову RDF-трійку: об’єкт – «антифрикційна втулка», предикат – «матеріал», суб’єкт – «олов’яна бронза». Така сама конструкція з’являється з використанням Neo4j. В даному випадку ми маємо дві вершини «антифрикційна втулка» і «олов'яна бронза» і спрямований зв'язок між ними типу «матеріал». Напрямок зв'язку йде від вершини «антифрикційна втулка» до «олов'яної бронзи». А тепер порівняємо формальні SPARQL (або Cypher) запити для розглянутих випадків. Припустимо, метою запиту є списку повернення деталей машин, виготовлених із олов’яної бронзи. Для описаної вище структури OWL запит SPARQL буде таким:

```
SELECT DISTINCT ?result WHERE {
    ?propName rsdfs:subPropertyOf :IsMadeOf.
    ?propName rdfs:range :TinBronze.
    ?propName rdfs:domain ?className.
    ?className rdfs:label ?result.
    ?className rsdfs:subClassOf ?intermediateClass.
```

```
    ?intermediateClass rsdfs:subClassOf :Machinery.
}
```

У наведеному вище прикладі запиту нам потрібна змінна для властивості, RANGE якої ми знаємо, але маємо запитати про її DOMAIN. І формально ця властивість має бути нащадком властивості “IsMadeOf”, бо нам тут не потрібні інші зв’язки. Крім того, клас, який ми тут запитуємо в результаті, має бути нащадком чогось, що є нащадком класу «Machinery». Для RDF/XML із довільними предикатами запит SPARQL із такою ж метою буде мати наступний вигляд:

```
SELECT DISTINCT ?result WHERE {
    ?className rdfs:IsMadeOf :TinBronze.
    ?className rdfs:label ?result.
    ?className rsdfs:BelongTo :Machinery.
}
```

Тут ми маємо довільні предикати «IsMadeOf» зі значенням семантичного посилання на матеріал та «BelongTo», який описує належність до чогось незалежно від ієрархії. Це, здавалося б, не надто суттєве спрощення може мати вирішальну роль для складніших запитів, де це може дозволити уникнути заплутаних ланцюжків посилань. Ось ще один приклад, де ми просто хочемо запитати роки, коли були написані листи людиною на ім’я Сергій Олексійович Завгородній. Ви можете побачити, наскільки складним стає запит, використовуючи онтологію тільки зі стандартними предикатами OWL:

```
    SELECT DISTINCT ?result WHERE {
                ?p rdfs:range : Sergiy.
                ?p rdfs:range : Oleksiyovych.
                ?p rdfs:range : Zavgorodniy.
                ?p rsdfs:subPropertyOf :FullName.
                ?p rdfs:domain ?y.
                ?y rdfs:subClassOf :Person.
                ?x rdfs:domain ?y.
                ?x rdfs:range ?t.
                ?x rsdfs:subPropertyOf :Authorship.
                ?t rdfs:subClassOf :TextLink.
```

```
                ?d rdfs:domain ?t.
                ?d rdfs:range ?r.
                ?d rsdfs:subPropertyOf :SendingDate.
                ?r rdfs:subClassOf :Date.
                ?yd rdfs:domain ?year.
                ?yd rdfs:range ?r.
                ?year rdfs:subClassOf :Year.
                ?year rdfs:label ?result.
        } ORDER BY ?year
```

Тут ланцюжок із кількох властивостей використовується для побудови зв'язку від частин повного імені людини до року, який фактично є частиною дати, яка прив'язана до тексту листа, який прив'язаний до особи за допомогою властивості «авторство», і кожна «особа» пов'язана частинами свого імені. Це реальний приклад запиту SPARQL, який може бути автоматично створений за шаблоном в одній із наших діалогових довідкових систем. Якби онтологія була побудована на дещо іншій парадигмі з використанням прямих довільних предикатів і довгих розгорнутих імен класів, запит з тією ж метою мав би такий вигляд:

```
SELECT DISTINCT ?result WHERE {
    ?text rdfs:subClassOf :TextLink.
    ?text rdfs:Authorship
Sergiy_Oleksiyovych_Zavgorodniy.
    ?text rdfs:SendingDate ?date.
    ?date rdfs:YearOfDate ?year.
    ?year rdfs:label ?result.
} ORDER BY ?year
```

Якщо в якості СУБД використовується Neo4j, запити на мові Cypher для розглянутих вище двох прикладів будуть такими:

```
MATCH
    (n:MachineryPart)-[:MadeOf]-(m:Material)
WHERE
    m.name = "Tin bronze"
RETURN n.label as result;
```

```
MATCH
    (n:Person)-[:Authorship]-(t:TextLink)-
[:SendingDate]-(d:Date)
WHERE
    n.name = "Sergiy_Oleksiyovych_Zavgorodniy"
RETURN d.year as result;
```

Тож ми бачимо, що в Cyper запити з тією ж метою можуть бути більш простими та зрозумілими. Слід зазначити, що складніші запити також можливі в Cyper, а окрім того виконання графових алгоритмів, що зробило його використання кращим для ряду випадків, припускаючи великі графи зі складною структурою, які також потребують запитів зі складною та розгалуженою семантикою.

4.2.3.5 Метод приведення вхідних сутностей до найближчих наявних в онтологічному графі. Як було зазначено, використання прямих довільних предикатів зі значеннями семантичних зв’язків і розгорнутих сутностей для вершин робить онтологію більш семантично структурованою та спрощує запити, одночасно роблячи їх більш семантично націленими. Але в умовах розробки діалогових систем виникає проблема співвідношення між поняттями, витягнутими з фрази користувача, і тими, які потрібно підставити у формальний запит. І на відміну від версії, яка містить єдиний шаблону запиту тут з'являється цілий ряд семантичних типів і шаблонів для них. Тож ще однією важливою проблемою постає вибір відповідного шаблону, що потребує семантичного аналізу. Але спочатку давайте розглянемо проблему приведення понять, витягнутих із фрази користувача, до таких, які зберігаються в онтології. У випадку онтології, де терміни атомізовані до окремих слів, у більшості випадків буде достатньо лише вищезгаданої редукції. І також редукція термінів у такому випадку було досить простим процесом – достатньо вибрати і вилучити менш значущі слова. Проте цей підхід має свої недоліки, зазначені вище. Якщо в онтології є довгі

багатослівні терміни, слова у яких ще й також розташовані в певному порядку, існує досить низька ймовірність того, що вони будуть точно співпадати з таким з фрази, яку написав користувач. Окрім того, у даному випадку терміни потрібно витягувати лише з певних місць. І скорочення тут не дуже допомагає. По-перше, такий підхід не враховує порядок слів (а перерахування варіантів перетасування є ресурсоємним процесом), по-друге, цей метод безпорадний для термінів з онтології, які мають додаткові слова (розширення контексту – взагалі проблема неоднозначна). Тому в цьому випадку необхідно розширити або скоротити контекст отриманих понять до найбільш підходящого терміна, що зберігається в онтології. Іноді деякі слова, отримані з фрази користувача, потрібно опустити, але інші потрібно додати, щоб підібрати формулювання до того виду, що зберігається в онтології. Для вирішення цієї проблеми ми пропонуємо наступний підхід. Попередньо готується файл, де зберігаються всі терміни з онтології в тому вигляді, як вони там представлені. До кожного з них прив'язаний список слів із поняття. Слова зберігаються в лематизованому вигляді, і кожному з них ставиться у відповідність його частина мови (це виключає її визначення під час виконання програми, що прискорює робочий процес). Набір слів, що відповідає виділеному терміну, порівнюється з цими словами. Враховуються як додаткові, так і пропущені слова. Для оцінки ступеня подібності набору вхідних слів і одного з файлу використовується метрика, подібна до такої з роботи [104]. Метрика дає різні оцінки відповідним, додатковим і пропущеним словам залежно від їхньої частини мови. Щоб пришвидшити процес, не аналізуються деякі поняття, які мають менше слів, ніж найбільш придатне на поточний момент процесу визначення. Таким чином у сформований запит підставляється найбільш підходящий термін, і його присутність в онтології гарантована.

4.2.3.6 Підстановка вхідних понять у запит до бази знань. Іншим важливим завданням є виділення необхідних для підстановки у запит термінів. У нашому підході цей процес тісно поєднується з визначенням

семантичного типу фрази. Для визначення семантичного типу запропоновано деревоподібний метод, описаний у розділі 2. Дерево для визначення семантичного типу розроблено таким чином, що забезпечує визначення позиції, звідки беруться поняття, для підстановки у запит. Ці поняття виходять після визначення семантичного типу та представлені набором слів, який трансформується в поняття онтології описаним вище способом.

Для довідкової діалогової системи, де користувач просто задає питання, проблема збереження контексту не є досить важливою. Крім того, така система приречена на гнучке перемикання теми діалогу відповідно до вхідних запитів користувача. Однак іноді, щоб система була зручною для користувачів, слід враховувати деякі особливості. Одним із них є заміна займенників. У деяких випадках це легко зробити, проаналізувавши такі характеристики займенника, як його рід і число. Але це не завжди працює, тому що можуть виникати неоднозначності. Неоднозначності можуть бути різними залежно від мови. Наприклад, у мовах, де неживі істоти не розрізняються за граматичним родом, як, наприклад, в англійській мові, використання таких займенників, як «він» або «вона», демонстративно вказує на живу істоту (часто особу), згадану раніше, але займенник «воно», а також вказівні займенники все ж можуть призводити до неоднозначності. У багатьох інших європейських мовах, в тому числі і в українській, існує родове розрізнення іменників для неживих істот. Тож здогадатися, на що вказує займенник, було б легше навіть для неживих істот. Але це не повністю вирішує проблему, тому що у попередньому контексті може бути згадано кілька сутностей з однаковим граматичним родом і числом. Одним із способів в цьому випадку є використання евристики, яка намагається вгадати вказану сутність за ймовірністю наявності характеристик або здійснення дій (наприклад, «автомобілі не мають капелюхів», «кішки зазвичай не можуть бути зеленими», «дерева не можуть бути учнями» тощо). Такий підхід спрацює в більшості випадків, за винятком дуже специфічних або абсурдних фраз, які можуть заплутати навіть людей. Але згаданий підхід потребує

багато ресурсів для його повної реалізації. Має бути ще одна додаткова база даних, ймовірно, онтологічного типу, де перераховані принаймні деякі можливі комбінації сутностей. Перерахування неможливих не потрібне з двох основних причин: якщо комбінація не вказана як можлива, вона розглядається як неможлива або принаймні малоймовірна; може бути нескінченна кількість неможливих ситуацій. Простіший спосіб – це запитати користувача про те, що він має на увазі, якщо виникає плутанина в заміні займенника. Отже, ми переходимо до другого типу діалогових систем, які припускають задавати користувачеві деякі запитання.

Описаний тип діалогових систем задає користувачеві запитання лише за наявності технічної потреби. Одна з них згадана вище: заміна займенників. Можуть бути й інші випадки: система «вважає», що питання вичерпано, тому задає питання, чи є у користувача інше питання, або слід вважати розмови закінчено; користувач ініціює завершення діалогу, тому системі «цікаво», чи отримав користувач правильні відповіді (ці дані можуть бути зібрані з відповідними історіями діалогів і далі використані для вдосконалення програми та бази даних); інформування користувача про те, що його питання не по темі, що в тій чи іншій мірі збереженням контексту.

Проблема збереження контексту не надто актуальна для систем з вузькою предметною онтологією. Там усі запитання, далекі від теми, призведуть до порожньої відповіді, що призводить до попередження користувача в діалозі про те, що питання не стосується предмету, на який орієнтована система. Якщо система багатопредметна і, ймовірно, має кілька онтологій. В цьому випадку для довідкової системи буде кращим варіантом гнучке перемикання від однієї онтології до іншої, ніж спроби утримати користувача біля однієї теми. Останнє може навіть дратувати людину. Уявіть собі ситуацію, коли користувач просто хоче запитати про інше, що його також цікавить, але система відмовляється відповідати, а замість цього пропонує не змінювати тему розмови.

4.2.4 Діалогова система для анкетування

Тип діалогових розроблений скоріше ставити запитання, а не відповідати на них систем є досить специфічним. Такі системи також дуже корисні в деяких ситуаціях, серед яких: інтерактивне заповнення анкети, яке таким чином може бути гнучким; системи тестування, які, наприклад, перевіряють розуміння студентом вивченого матеріалу; інтерактивні діагностичні системи. У простому типі такої системи є лише фіксований набір запитань, які потрібно поставити користувачеві. Єдиною відмінністю його від звичайного комп'ютерного тесту з автоматичною перевіркою є відповідь користувача природною мовою. Тут не питання, а відповіді користувача проходять семантичний аналіз і виділення значущих сутностей. Тому фрази користувача сприймаються скоріше як розповідні, а не питальні.

Залежно від семантичного типу та виділених понять будується формальний запит або їх набір. Його результати зберігаються і можуть бути використані надалі.

Може бути більш складний тип системи, де наступне питання системи залежить від попередніх. Це нелінійна діалогова анкетна система. Варіантів реалізації може бути декілька. Одна з них – система тестування з адаптацією питань. Вона визначає, на які питання користувач дає які відповіді. Таким чином, залежно від мети, наступні питання будуть, наприклад, скоріше з тем, з якими знайомий користувач (більш правильні відповіді були надані раніше), або, навпаки, щодо предметів, у яких користувач не обізнаний. Інша версія – це адаптація запитань у об’єктивних посиленнях: якщо користувач дає правильні відповіді на легші питання, наступні будуть складнішими, і навпаки.

Інша послідовність нелінійного діалогу в системах такого типу може бути наступною: в основі діалогового керування є дерево рішень. Умови для точок біфуркації визначаються відповідями користувача. Таким чином, наступне питання буде відрізнятися в залежності від результату, отриманого програмою з використанням матеріалу останньої фрази користувача.

Наприкінці діалогу система може дати відповідь, яка по суті є результатом тесту або діагностики. Якщо система призначена для тестування, таким остаточним результатом може бути, наприклад, оцінка правильних і неправильних відповідей з інформацією про те, чи тест пройдено та на яку оцінку. Якщо діалогова система є джерелом для умов біфуркації дерева рішень, ця результуюча відповідь визначається кінцевою вершиною дерева, яка пов’язана з діагнозом. Якщо призначенням системи є просто анкета, розроблена для заповнення бази даних на основі відповідей користувачів, результуюча відповідь може не існувати взагалі або бути просто формальною, наприклад «Дякуємо за інформацію. Нам було приємно з вами поговорити» або «ОК. Дані успішно збережені».

4.2.5 Діалогова система з активною ініціативою

Загальна характеристика і особливості діалогових систем із активною ініціативою. Найскладнішим є четвертий тип діалогових систем. Він передбачає активну участь програми в процесі діалогу, тобто він повинен задавати питання і відповідати на такі, що надходять, повідомляти користувачеві якісь факти, в тому числі про такі, що на згадувалися раніше у даній розмові, надавати деякі коментарі, пов'язані не з питальними, а розповідними фразами користувача, а іноді використовувати їх для постановки своїх запитань.

В основі такої системи могла б лежати комбінація типів 2 і 3, але однієї лише комбінації недостатньо. Система повинна гнучко, автоматично перемикатися з одного типу на інший і робити це в потрібний момент. Можуть бути дві загальні схеми розгортання діалогу в залежності від того, яка зі сторін починає тему розмови першою змістовною фразою (не формальним привітанням).

Випадок ініціювання діалогу користувачем. Розглянемо випадок, коли користувач вводить таку репліку. Це може бути: питання, спонукання або розповідь. Розглянемо кожен із цих трьох варіантів окремо. Якщо початкова фраза користувача є питанням, передбачається, що система

повинна спробувати на нього відповісти. Це можна зробити, як у базових типах діалогових систем (типи 1 і 2). Якусь відповідь можна отримати в будь-якому випадку: інформативна відповідь, попередження про незнання релевантної відповіді, остаточно отримана відповідь після уточнюючого запитання (тип 2). Якщо система обмежуватиме себе випадками питальної фрази користувача лише поверненням відповіді, вона може виродитися до простої довідкової системи. Тому ініціатива програми не повинна пригнічуватися користувачем. Перше, що може зробити система в цьому випадку (як і в інших розглянутих далі), це визначити загальну тему першої фрази. Це можна зробити за допомогою обраної онтології або подібним чином. Якщо для наступних користувачів буде визначено, що фраза належить до іншої теми, це призведе до запитання про зміну теми та її причину. Це вже часткове перехоплення ініціативи системою. Тому як така дія змушує користувача відповісти на системне запитання, в тому числі беззаперечно про причини зміни теми. Але найефективніший спосіб перехоплення ініціативи зі сторони системи – це задати питання користувачеві безпосередньо після щойно даної відповіді, не чекаючи, поки користувач виконає наступну дію діалогу. Це може бути не питання, а розповідь, однак питання є кращим варіантом, тому що не зовсім схожа на запитання розповідь після відповіді може сприйматися як така собі специфічна частина відповіді. Однак такі фрази, якщо вони навіть є розповідними, можуть мати певний початок, наприклад «Чи знаєте ви, що існують також такі речі, як…», що здатні відокремити таку фразу від безпосередньо відповіді. Такі дії можуть йти не після кожної відповіді, але з'являтися час від часу. Щоб підтримувати тему розмови, ці питання, ініційовані програмою, можуть бути викликані випадковим чином, але мають бути пов’язані з основною темою розмови. Якщо для такої фрази потрібно вибрати тему розповіді, її можна вибрати випадковим чином із пов’язаної інформації, отриманої разом з основною відповіддю, як описано вище. Якщо початкова фраза користувача є розповідним текстом, ситуація полегшується

для програми. Відповіддю системи в цьому випадку може бути інший пов’язаний текст або запитання. Імперативні фрази користувача трактуються так само, як питання.

Випадок ініціювання діалогу програмною системою. Інший випадок, коли програма ініціює першу осмислену фразу діалогу. Це може бути як питання, так і розповідь. Якщо це питання, передбачається, що користувач відповість на нього. Як використовувати відповідь користувача та чи вона взагалі має значення для подальшого діалогу, залежить від призначення системи та поточного запитання. Якщо, наприклад, відповідь стосувалася імені користувача, це ім’я, ймовірно, можна витягти з відповіді. У наступних фразах програма може використовувати це ім'я. Після того, як користувач дає відповідь, система досить вільно вибирає тип наступної фрази. Якщо система починає діалог з розповіді, вона може очікувати від користувача всіх варіантів типу фрази. Подальша поведінка визначається фразою користувача майже так само, як це було б на початку.

Проблема збереження предмета діалогу

Сутність проблеми. Проблема збереження предмета діалогу для такого типу системи є найбільш актуальною порівняно з іншими. Її можна розділити на дві частини: довгострокове ведення теми та короткострокове. Можливим рішенням задачі довгострокового утримання теми діалогу може бути поділ онтології на тематичні частини, а фрази, ініційовані системою, повинні знаходитися у рамках відповідної частин, яка задіяна в даний момент діалогу. І в принципі не має вагомої практичної причини змушувати користувача зберігати лише одну тему протягом усього діалогу. Кращим способом буде вибрати найбільш підходящу онтологію для поточної теми фрази користувача. Але може бути якась причина надавати перевагу останній залученій онтології у випадку, якщо є дві або більше антологій, визначених як однаково або дуже схожі. У будь-якому випадку довгострокове збереження теми потрібне скоріше для уникнення дивних і кумедних ситуацій, коли система кілька разів «стрибає» з однієї теми на іншу.

Короткострокове збереження теми може бути більш важливим. Воно визначає, якої подальшої поведінки очікувати від користувача, і як ставитися до його наступної фрази. Дві основні частини цієї проблеми: заміна займенників і семантичне очікування. Першу з них уже було згадано у даній роботі вище. Вона може бути вирішена керуючись наступними принципами: пошук іменників і іменних груп в одній-двох попередніх фразах; якщо є кілька кандидатів на заміну, запитати у користувача правильний варіант; якщо в останніх двох фразах немає кандидата на заміну, просто запитайте користувача, що він має на увазі, використовуючи займенник. Для фраз, що генеруються збоку програми було б кращим способом уникати використання займенників без попередньо використаного відповідного іменника в тому самому діалоговому акті. Це лише робить фрази програми більш зрозумілими для людини.

Шляхи вирішення задачі збереження предмета діалогу. Вирішення проблеми семантичного очікування полягає в запам’ятовуванні типу останньої фрази та відповідних програмних інструкціях, які визначають відповідну реакцію на наступну дію користувача. Розглянемо типові випадки:

1) Програма задає запитання користувачеві. У цьому випадку очікування буде відповіддю на це питання. Однак користувач може набрати що завгодно: його фраза може бути розповідною, наказовою чи питальною. Зазвичай відповідь на запитання має бути розповіддю, але в реальних випадках це відбувається не завжди. Але також може виникнути плутанина, коли фраза користувача у формі запитання чи наказу насправді є відповіддю. Наприклад, система запитує користувача «Який твій улюблений фільм?», а користувач відповідає «Розкажи мені казку». Граматично ця фаза є спонуканням. Тож це можна розглядати так, ніби користувач проігнорував запитання та попросив систему розповісти йому історію. Але це також може бути назвою фільму. Можуть бути й інші подібні приклади – назви книг і фільмів іноді можуть складатися у формі запитання чи звернення. Тут

виникає питання: чи для всіх типів питань відповідь може бути не в розповідній формі? Ні, можливо, не для всіх. Наводимо перелік семантичних типів запитань, які можна очікувати відповіді, що виглядає як спонукання: вказівка, адреса, вхідний напрямок, вихідний напрямок, розділення, об’єднання, заміна, інструмент, об’єкт дії, мета, значення, тема, об'єкт характеристики. У наведеному вище прикладі семантичний тип питання – «об’єкт характеристики». Запитання пояснює, що цей об’єкт має належати до категорії «фільм» і мати характеристику «улюблене» для користувача. Очікувана відповідь – назва фільму, що іноді може бути подана у формі наказового способу. Питальні відповіді можуть бути правильними для таких семантичних типів: вхідний напрямок, вихідний напрямок, поділ, об’єднання, об’єкт дії, мета, причина, значення, оцінка, тема, об’єкт характеристики. Якщо фраза користувача, що йде після питання програми цих типів, є спонуканням або питанням, хорошим рішенням для системи буде задати уточнююче запитання про те, чи справді ця фраза є об’єктом очікуваної відповіді. Якщо відповідь користувача на уточнююче запитання «так», це буде оброблено як відповідь, інакше, якщо відповідь «ні», фраза буде оброблена звичайним чином згідно її типу та змісту. В останньому випадку поставлене питання буде вважатися залишеним без відповіді.

2) Фраза з боку програми є розповіддю. Будь-який тип фрази, що надходить від користувача розглядатиметься відповідно до його типу.

3) Фраза з боку програми є спонуканням. Тут реакція на наступну фразу користувача залежить від наступних факторів. Якщо імператив передбачає дію, як-от повідомлення певної інформації системі, це фактично дорівнює питанню. Якщо він пропонує користувачеві виконати якусь дію в реальному світі, це, ймовірно, порада, яка може бути дана як реакція на попередні фрази. У цьому випадку не повинно бути ніякого впливу на сприйняття наступної фрази користувача.

Даний тип діалогової системи також може бути укомплектований деревом прийняття рішень, як описано вище для третього типу (анкета).

Таким чином, відповіді користувачів на запитання системи стають критеріями біфуркації, які визначають наступні фрази, які система дає користувачеві.

Описаний тип діалогової системи може бути корисним для широкого спектру призначень, в тому числі, таких для систем вищезгаданих типів. Крім того, можливість як задавати запитання, так і відповідати на них може зробити роботу з діалоговою системою більш зручною для користувача та менш формалізованою. Крім того, програма дозволяє отримувати інформацію від користувача в ненав'язливий спосіб.

4.3 Мультиагентна схема для реалізації інтелектуальної діалогової системи

4.3.1 Модель інтелектуального агента

Інтелектуальний агент, що розглядається повинен мати наступні модулі: модуль завантаження і перевірки оновлення даних, база даних, що містить завантажені і оброблені дані, база даних, що містить технічну інформацію, необхідну для роботи застосування, аналізатор даних, що встановлює взаємозв’язки між наборами даних і формує зв’язану табличну структуру, аналізатор даних, що опрацьовує зібрані дані, намагаючись знайти приховані закономірності, зробити прогнози, формує таблиці похідних знань, модуль взаємодії з користувачем, аналізатор інформації, що надходить від користувачів і впливає на діяльність модулю завантаження даних і аналізатору, намагаючись вгадати потреби користувачів і задовольнити їх.

В основі модулю, що виконує збір і перевірку оновлення даних лежить програма, що завантажує файли «сирих» даних за допомогою відправлення HTTP запитів на адреси, що зберігаються у відповідній таблиці, що містить технічну інформацію для роботи застосування. Відбір наборів даних для завантаження відбувається на основі інформації, що поступає в результаті взаємодії з користувачами. Так ключові моменти, що цікавили користувачів

зберігаються у певному розділі бази даних технічної інформації застосування. Модуль збору даних, ґрунтуючись на зібраній інформації щодо потреб користувачів виконує скрапінг сторінок порталу, аналізуючи заголовки наборів даних та описи до них. Розділи сторінок порталу мають регулярну одноманітну структуру, HTML-теги мічені значущими атрибутами. Це робить автоматичний скрапінг у даному випадку здійсненою і не досить складною задачею. Єдиною адресою, що повинна бути заданою вручну є адреса сторінки, що містить посилання на набори даних. Здатною до самоналаштування є регулярність оновлення інформації у внутрішній базі знань застосування. Для встановлення первісного значення періодичності оновлення орієнтиром слугує дата останнього оновлення (є на сторінці). Період оновлення розраховується як різниця поточної дати і дати останнього оновлення, поділена на фактор, що враховує вірогідність більш швидкого оновлення. Обчислення цього фактору є предметом довгострокової оптимізації роботи агенту. Дати оновлення зберігаються у базі технічної інформації і на підставі цього списку згодом періодичність може бути зменшена або збільшена.

Завантажені сирі «дані» підлягають первісній обробці. Згідно з форматом завантаженого файлу запускається відповідний парсер що перетворює їх на уніфіковані табличні структури. Виконується інтелектуальний пошук зв’язків між стовбцями таблиць, отриманих з різних наборів даних. Підставами для встановлення зовнішніх зв’язків між отриманими таблицями є схожість заголовків таблиць та наявність однакових послідовностей значень. Утворена мережа є основою заданої до зростання бази знань інтелектуального агента.

Періодично система наявних даних опрацьовується аналізатором. Основою для генерації цілей аналізу слугують запити користувачів. Задачі вирішувані аналізатором зводяться до класифікації, лінійної і нелінійної регресії, побудови дерев прийняття рішень. Для безпосереднього здійснення цієї діяльності можуть бути використані як власні ресурси, так і API

зовнішніх відкритих сервісів. Результати кожного сеансу аналізу зберігаються в окремих таблицях бази даних і можуть слугувати вихідними даними для аналізу другого порядку (аналіз над результатами ряду аналізів, проведених у різні періоди).

Модуль взаємодії з користувачем складається з чат-бота і візуалізатора. Можлива також система автоматичного періодичного формування звітів. Для більшої універсальності і незалежності агент має бути обладнано REST API. Саме через API повинне здійснюватися спілкування агента з оболонкою чат-бота і візуалізатора. Наявність REST API робить агента незалежним від платформ, що реалізують зовнішні інтерфейси, а також робить результати його діяльності доступними для опрацювання безпосередньо програмою (в тому числі і іншими агентами). Дані, що поступають від користувачів опрацьовуються. Вони відправляються до аналізатору, що виділяє з них основні ідеї. На їх основі формулюється відповідь на підставі наявної бази знань. За допомогою системи мовного синтезу на формулюються відповіді на природній мові. Окрім того, якщо це буде визнано потрібним формуються зручні візуальні представлення даних: таблиці, діаграми, графіки, схеми. Окрім того, дані передані користувачем передаються для опрацювання модулям, що здійснюють постановку задач для подальшого пошуку нових даних і цілей аналізу наявних.

4.3.2 Приклад мультиагентної схеми для діалогової системи

Мультиагентна система дозволяє розподіляти завдання та функціонал між незалежними агентами, які не контролюються безпосередньо користувачем [105]. Поведінка агента намагається реалізувати модель BDI. Така архітектура має ряд переваг [106]. Мультиагентна парадигма передбачає, що виконання завдань може бути розподіленим, і припускає можливість зниження навантаження на апаратне забезпечення. Кожен агент може працювати як мікросервіс. Але на відміну від мікросервісів, агенти можуть приймати самоконтрольовані рішення відповідно до ситуації, яка

відображається в їхніх «переконаннях» і «бажаннях». Один агент може бути замінений новим або модернізований без необхідності перебудови інших. Якщо якийсь агент виходить з ладу, це не призводить до збою всієї системи, а виниклу проблемну ситуацію можна легше локалізувати та ліквідувати. Однією із реалізацій агентської системи є інтелектуальний чат-бот [107]. У цьому випадку функціональні можливості бази знань, взаємодія з користувачами та управління системою розподіляються між агентами.

Для реалізації взаємодії між компонентами чат-бота розроблено агентну систему. Агентами системи є: Ontology Agent, Intermediary Agent, Register Agent, Control Agent, Client Agent. Функціонування останнього є тимчасовим, тобто його працюючі екземпляри і існують одночасно в кількості, що дорівнює кількості користувачів, які зараз взаємодіють із системою. На рисунку 1 нижче наведена схема розробленої агентної системи.

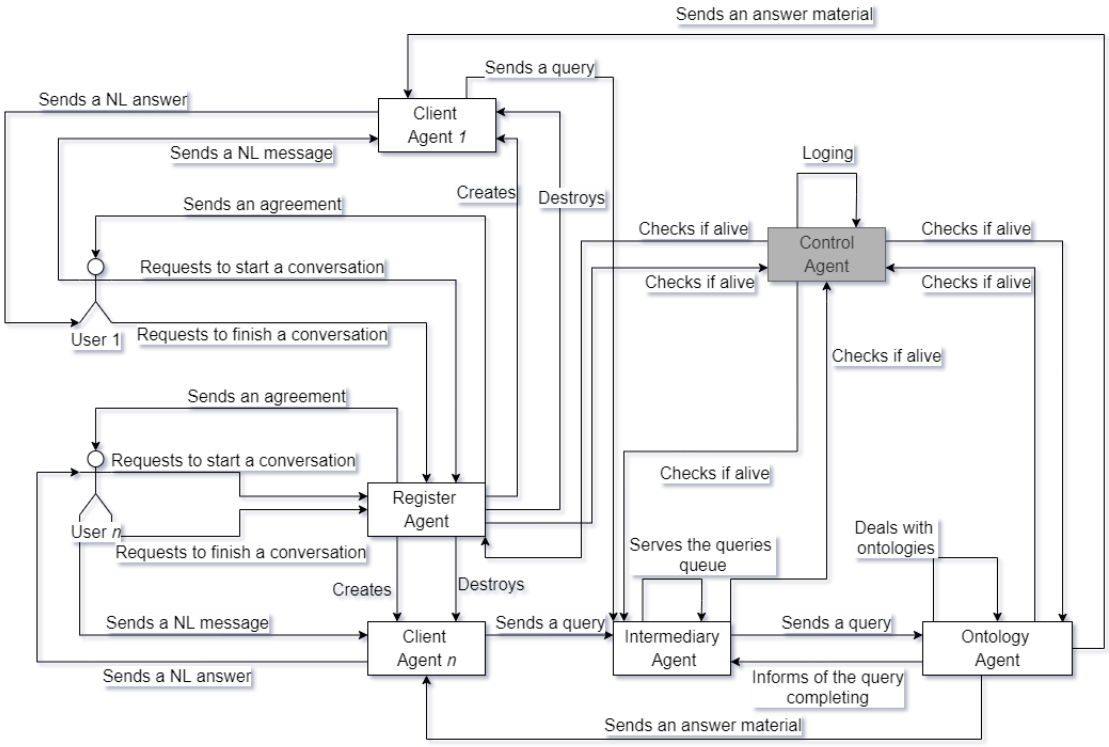


Рисунок 4.1. Схема розробленої агентної системи

Кожен агент системи є незалежним HTTP-сервером, який взаємодіє за протоколом FIPA [106]. Схема передбачає, що користувач має HTTP-клієнт, який може взаємодіяти з Register Agent і Client Agent API за допомогою

протоколу FIPA. Щоб розпочати розмову, користувач надсилає повідомлення Register Agent з проханням почати її. Тоді Register Agent запитує Control Agent, чи система працює належним чином. Якщо повідомлення правильне та в системі немає аварійних ситуацій, Register Agent надсилає користувачеві повідомлення про згоду та адресу створеного ним екземпляру Client Agent. Подальша взаємодія між користувачем і системою буде здійснюватися через цього клієнтського агента, за винятком запиту на завершення розмови, який буде адресований безпосередньо реєстраційному агенту. Наявність клієнтських агентів забезпечує незалежність і асинхронність розмов різних користувачів паралельно. Клієнтський агент може містити розмову та особисту інформацію користувача. Він виконує операції NLP і NLU з повідомленнями користувача і формує набори запитів SPARQL. Ці процедури досить обтяжливі, і їх виконання займає досить тривалий час. Тому видається розумним рішення зробити цих агентів кількома і працювати незалежно. Реєстраційний агент може мати можливість запускати клієнтського агента на найменш завантаженому сервері, або також може бути корисним впровадження клієнтського агента на машині користувача.

Сформовані запити, пов’язані з розмовою та ідентифікаторами повідомлень користувача, надсилаються агенту-посереднику, який обслуговує чергу запитів. Кожне повідомлення від клієнта надсилається ним до черги запитів. В іншому потоці повідомлення одне за одним витягуються з черги та надсилаються до агента онтології.

Основна роль агента онтології полягає у виконанні запитів SPARQL до онтологій. Також попередньо формується вибір найбільш підходящої онтології. Щоб прискорити виконання запитів, цей процес можна розпаралелити. Після отримання набору результатів з онтології виконується наступний набір спеціальних запитів SPARQL, щоб визначити, що це за сутності в цілому. Після завершення одного пакета запитів Ontology Agent інформує агента-посередника. Отримані результати надсилаються безпосередньо до Client Agent.

Після отримання повідомлення від агента онтології клієнтський агент формує повідомлення природною мовою та надсилає його користувачеві. Якщо користувач хоче завершити розмову, він змушує свого клієнта надіслати агенту реєстрації спеціальне повідомлення з наміром припинити розмову. Якщо повідомлення вірне і немає перешкод для дії, Register Agent знищує екземпляр (процес) Client Agent і інформує користувача про завершення розмови.

Control Agent працює протягом усього часу роботи системи, і його основна мета — перевірити, чи належним чином працюють агенти, і спробувати автоматично відновити їх, якщо це необхідно. Якщо деякий агент не отримує повідомлення ping від Control Agent кілька разів поспіль, той, якщо потрібно, намагається його відновити.

4.4 Практична реалізація діалогових систем, працюючих з онтологією

4.4.1 Приклади реалізації діалогової довідкової природномовної системи із контекстною базою знань

За розглянутими принципом було створено декілька діалогових систем. Вони розроблені для української мови, також мають подібну структуру використаної онтології.

UML-діаграма прецедентів для діалогової системи даного типу наведена на рисунку 4.2.

Архітектура системи. Для реалізації діалогової системи (далі – «система») запропонована мультиагентна архітектура. Це обумовлено тим, що багатоагентна система дозволяє розподіляти завдання та функції між незалежними агентами, які не контролюються безпосередньо користувачем. Поведінка агента намагається реалізувати модель BDI. Така архітектура має певні переваги. Багатоагентна парадигма передбачає, що виконання завдань може бути розподіленим, і вона передбачає можливість зниження апаратного

навантаження. Кожен агент може працювати як мікросервіс, запитуваний іншими агентами, коли це необхідно. Але на відміну від звичайних мікросервісів, агенти можуть приймати самостійні рішення відповідно до ситуації, яка відображається в їхніх «переконаннях» та «бажаннях». Один агент може бути замінений новим або модернізований без необхідності оновлення чи заміни інших. Якщо у якомусь агенті виникне помилка, це не призведе до збою всієї системи, а виниклу ситуацію можна буде простіше знайти та ліквідувати. У випадку діалогової системи такі процеси, як функціональність бази знань, взаємодія з користувачами та управління системою, розподіляються між агентами.

Схема архітектури і міжагентної (міжсервісної) взаємодії показана на рисунку 4.3.

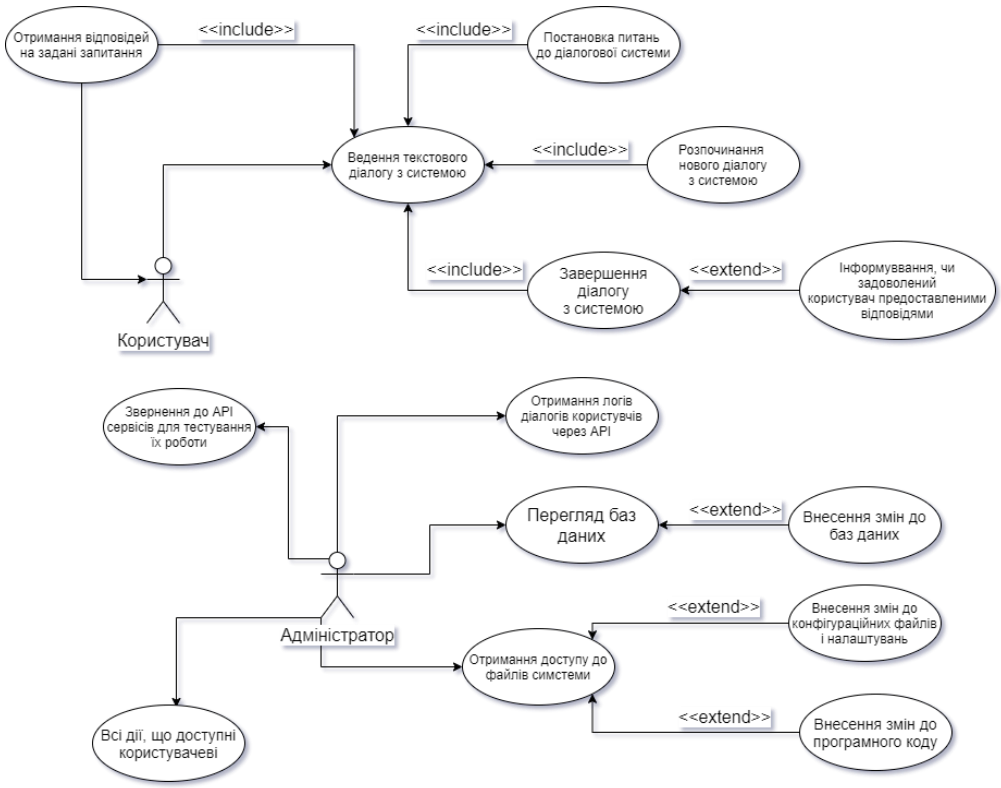


Рисунок 4.2 – UML-діаграма прецедентів діалогових систем типу, що розглядається

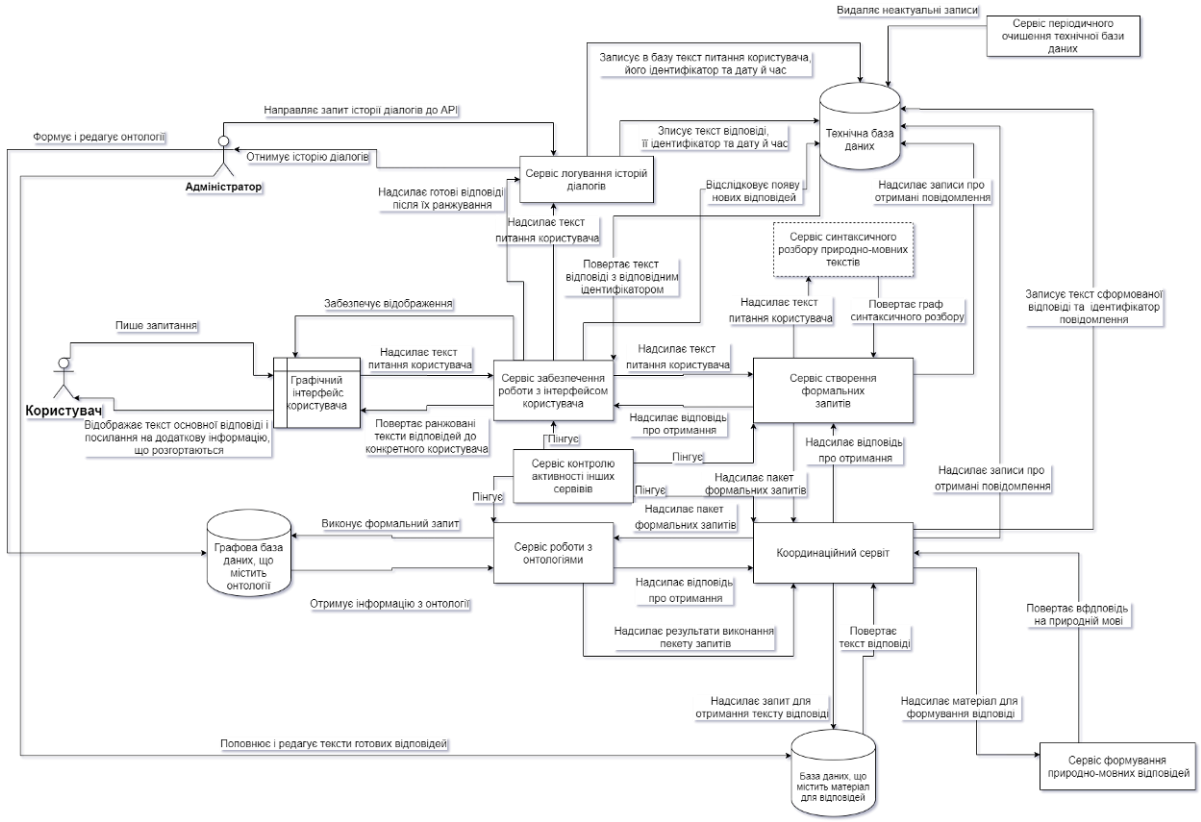


Рисунок 4.3 – Схема архітектури діалогової системи типу, що розглядається

Система передбачає наявність наступних компонентів:

- графічний інтерфейс користувача;

- сервіс забезпечення роботи між інтерфейсом користувача і іншими компонентами системи;

- сервіс (агент) створення формальних запитів;

- сервіс (агент) роботи з онтологіями;

- координаційний сервіс;

- сервіс логування історії діалогів;

- сервіс періодичного очищення технічної бази даних;

- сервіс синтаксичного розбору природномовних текстів (опційний);

- сервіс формування природномовних відповідей чи, принаймні підготовки інформації, отриманої з бази даних, до формату, придатного до відображення;

- графова база даних, що містить онтології;

- база даних, що містить матеріали для відповідей;

- технічна база даних;

- сервіс, що контролює активність інших сервісів.

Функціонування системи. Кожен агент системи являє собою окремий сервіс, що взаємодіє з іншими через HTTP/HTTPS протокол. Для передачі даних між агентами системи використовуються POST-запити. Агенти матють можливість відповідати за всіма адресами свого API також і на GET-запити, але лише сповідуючи, що агент працює за допомогою HTTP-відповіді. Структури даних, що передаються між агентами системи відповідають, чи, принаймні, максимально близькі до протоколу FIPA.

Сервіс забезпечення роботи з інтерфейсом користувача, агент створення формальних запитів, агент роботи з онтологіями, координаційний сервіс та сервіс логування історії діалогів обов'язково мають бути асинхронними, тобто здатними одночасно і незалежно обробляти кілька запитів, що походять від різних користувачів. Також асинхронно мають працювати системи керування базами даних.

Графічний інтерфейс користувача. Користувач застосовує графічний інтерфейс для початку і завершення діалогу з системою та передачі їй текстів своїх повідомлень. Текст повідомлень користувача надходить до сервісу, що контролює діяльність і відображає графічний інтерфейс користувача. Він також виконує первинну обробку даного тексту. Після первинної обробки текст фрази користувача, якщо та не є пустою або шаблонною і написана українською мовою, відправляється до сервісу формування формальних запитів. В іншому випадку, відправляється відповідь без задіяння інших компонентів системи.

Сервіс формування формальних запитів. Після отримання повідомлення, що містить фразу користувача, сервіс формування формальних запитів надсилає відповідь відправнику про успішне отримання і приступає до її обробки.

Сервіс формування формальних запитів може відмовити в обробці фрази користувача відправнику у наступних випадках:

- неправильний формат повідомлення;

- неправильний або відсутній пароль;

- запит надійшов від невідомого відправника;

- запит не є адресованим даному сервісу;

- мова фрази, що позначена у запиті не є українська;

Обробка тексту фрази користувача сервісом формування формальних питань полягає в наступному:

- синтаксичний розбір фрази користувача і побудова синтаксичного графу (для цього можуть бути використані місцеві додатки та програмні бібліотеки або API відповідного сервісу);

- семантичний аналіз з використанням даних синтаксичного графу, редукції фрази для узагальнення окремих понять і запитуваної інформації загалом (необхідно для збільшення ймовірності отримання відповіді, а також для отримання додаткової пов’язаної інформації);

- побудова пакету формальних запитів за обраними схемами-шаблонами;

- вибір для кожного запиту найбільш відповідної онтології з тих, що містяться у системі.

Сформований пакет формальних запитів відправляється до координаційного сервісу.

Координаційний сервіс. Координаційний сервіс відповідає про успішне отримання пакету формальних запитів. Він може відмовити в подальшій обробці пакету формальних запитів відправнику у наступних випадках:

- неправильний формат повідомлення;

- неправильний або відсутній пароль;

- запит надійшов від невідомого відправника;

- запит не є адресованим даному сервісу;

- вказана мова формальних запитів не підтримується сервісами даної системи, які працюють з онтологіями

Координаційний сервіс після відповіді відправнику про успішне отримання пакету формальних запитів направляє цей пакет до відповідного сервісу роботи з онтологіями (ім’я обраного сервісу залежить від обраної раніше онтології та зазначеної у повідомленні мови формальних запитів).

Сервіс роботи з онтологіями. Сервіс роботи з онтологіями після отримання пакету формальних запитів направляє відповідь відправнику про успішне отримання пакету формальних запитів. Сервіс роботи з онтологіями може відмовити в подальшій обробці пакету формальних запитів відправнику у наступних випадках:

- неправильний формат повідомлення;
- неправильний або відсутній пароль;
- запит надійшов від невідомого відправника;
- запит не є адресованим даному сервісу;
- вказана мова формальних запитів не підтримується даним сервісом роботи з онтологіями

Сервіс по роботі з онтологіями після відповіді відправнику про успішне отримання пакету формальних запитів виконує ці запити до відповідних вказаних онтологій і формує відповідний пакет відповідей. відповідями можуть бути, залежно від типу запиту і обраної онтології, посилання на готові відповіді чи набори понять. Після завершення виконання пакету формальних запитів сервіс по роботі з онтологіями надсилає результати виконання цих запитів координаційному сервісу.

Координаційний сервіс відповідає про успішне отримання результатів виконання формальних запитів. Координаційний сервіс може відмовити в подальшій обробці отриманих результатів формальних запитів відправнику у наступних випадках:

- неправильний формат повідомлення;
- неправильний або відсутній пароль;
- запит надійшов від невідомого відправника;
- запит не є адресованим даному сервісу;

- токен зазначений у запиті не міститься серед списку токенів запитів попередньо успішно відправлених даним сервісом до сервісу формування формальних запитів та сервісу по роботі з онтологіями (перевіряється у технічній базі даних).

Обробка і представлення результатів. Після відправки повідомлення про успішне отримання відповідей на формальні запити координаційний сервіс приступає до обробки результатів формальних запитів. У разі якщо відповіддю є посилання на готову відповідь у документно-орієнтованій базі даних до цієї бази відправляється відповідний запит для отримання тексту відповіді. Якщо відповіддю є набор понять, то відправляється повідомлення сервісу формування природномовних відповідей, яке містить результати запиту, та текст вихідної фрази користувача. Отримавши відповідь від документно-орієнтованої бази даних чи сервісу формування природномовних відповідей у вигляді природномовної фрази призначеної для користувача (це може бути також пакет відповідей) координаційний сервіс робить відповідний запис цих результатів до технічної бази даних (реляційна або документо-орієнована база даних).

Сервіс, що забезпечує роботу з клієнтським інтерфейсом користувача реагує на запис у технічній базі даних про отримання пакету відповідей, що відносяться до діалогового акту користувача (маркується відповідним унікальним токеном). Результат зчитується з технічної бази даних. Після зчитування з технічної бази даних пакету відповідей, призначених для користувача, сервіс що забезпечує роботу з клієнтським інтерфейсом користувача переходить до процесу ранжування відповідей за релевантністю вихідній фразі користувача, у разі, якщо пакет містить більше однієї відповіді. По закінченні ранжування (якщо таке потребується) сервіс, що забезпечує роботу з клієнтським інтерфейсом користувача надсилає обрану найбільш релевантну за його визначенням відповідь для відображення на сторінці інтерфейсу діалогу з певним користувачем. Інші відповіді надсилаються у вигляді коротких заголовків з посиланнями, за якими вони

можуть бути отримані з технічної бази даних за допомогою спеціальної компоненти даного сервісу, що забезпечує роботу з клієнтським інтерфейсом користувача. Основна та додаткові відповіді також направляються агенту логування історії діалогів, що записує їх до бази даних разом із ідентифікатором даного діалогу з даним користувачем та датою і часом отримання цієї відповіді.

Допоміжні технічні засоби системи. Для реалізації процесу логування діалогів у системі передбачено відповідний сервіс, що записує до технічної бази даних вхідні фрази користувачів і дані на них системою відповіді. Цей сервіс має API, до якого через надсилання POST-запиту може бути надіслано історію діалогів системи з користувачами за певний період.

Для очищення технічної бази даних від застарілої тимчасової інформації передбачено сервіс, який періодично видаляє з неї дані, що втратили актуальність.

Також розроблювана система має агент, що контролює активність інших агентів через надсилання їм тестових GET-запитів, на які ті мають відповідати про факт активності. Це дозволяє своєчасно спостерігати збої у роботі агентів, та запобігає їх «засинанню» при тривалій відсутності активності.

Дані діалогові системи використовують онтології побудовані автоматично на попередньо розмічених по принципу заголовок/контекст природномовних текстах українською мовою. Докладніше про структуру онтологій та методи їх побудови йдеться у розділі 2 даної роботи. Також у тому розділі наведено основні принципи формулювання запитів до онтології на основі природномовних реплік користувача, створення і ранжування відповідей.

Всі фрази користувача і дані на них відповіді системи записуються до реляційної бази даних з вказуванням дати і часу кожної репліки, а також унікальним ідентифікатором даного діалогу з даним користувачем.

Окрім того, в іншу таблицю записується протокол всіх повідомлень відправлених і отриманих агентами системи. В цій таблиці також вказується час відправлення/отримання і те, до якого діалогу вони належать.

Дані цих протоколів можна переглянути через графічний інтерфейс адміністратора, при введенні потрібного паролю, або отримати за допомогою API. У другому випадку можливо отримання як повної історії діалогів, так і за зазначений період часу. Запити до API надсилаються за допомогою POST-запитів у форматі JSON. Нижче наведено приклад, що ілюструє структуру такого запиту:

```
{
        "password": "rksor6fj82g2gdj31gj",
        "period": {
                "beginning": "June 11 2020 11:24PM",
                "end": "June 12 2020 1:49PM"
        }
}
```

Поле "password" містить пароль доступу до API. Поле "period" є необов’язковим, воно потрібно у випадку надання протоколів діалогів за певний період часу. Якщо цікавлять лише діалоги, починаючи з якогось часу, чи до якогось часу, слід вказати тільки поле "beginning" або "end", відповідно.

4.4.2 Діалогова система за матеріалами «Білої книги з фізичної та реабілітаційної медицини в Європі»

Загальна характеристика системи. В основі даної діалогової системи лежить створена вручну онтологія, що базується на змісті «Білої книги з фізичної та реабілітаційної медицини в Європі».Онтологія містить вершини, типізовані як класи і екземпляри та іменовані зв’язки між ними. Назви зв’язків семантично забарвлені та адаптовані до предметної області. Вершини мають назви і містять контексти. Контексти також є типізованими. Опис онтологічного графу створювався вручну за допомогою спеціально

розробленої системи проектування онтологічних графів “Graph Editor”. Він зберігає файли описів онтології у специфічному XML-форматі. Для роботи у діалоговій системі опис онтології було переведено до стандарту RDF/XML автоматично з використанням спеціально розробленої програми конвертера.

У системі використано метод семантичного аналізу вихідних фраз користувача за допомогою дерева прийняття рішень. Отримані з фрази користувача сутності не підставляються напряму до шаблону формального запиту. Перед цим проводиться їх приведення до найбільш близької сутності, наявній у означеній онтології (дивись пункт 4.2.3.5 вище). Це робить метод більш надійним з огляду на вірогідність отримання відповіді на запит. Це багато в чому обумовлено особливістю даної онтології, у яких назви сутностей представлені складними довгими фразами.

Поміщення шаблонів запитів та схеми дерева семантичного аналізу в окремі XML-файли дозволяє розробнику застосувати розроблений програмний продукт до певної онтології, не торкаючись логіки програмного коду. Код програми знаходиться у файлі python, до якого прив’язаний лінгвістичний аналізатор для роботи з певною природною мовою.

Взаємодія системи із сервісом обміну повідомленнями «Telegram». Діалогова підсистема працює як мережевий застосунок, розташований на сервері. Вона може мати і має свій власний інтерфейс користувача. Тим не менш, потенційні користувачі системи зацікавлені у використанні діалогової системи за допомогою звичних їм популярних засобів обміну повідомленнями. Одним з таких засобів є «Telegram». Цей сервіс надає можливість створення так званих ботів – віртуальних користувачів, що можуть бути підключені до зовнішнього сервісу, що аналізує отримані повідомлення і може надсилати свої. Засобами «Telegram» можна лише створити такого віртуального користувача і отримати код і спосіб зовнішнього програмного доступу до нього. При цьому програмні засоби обробки і формування повідомлень потрібно створювати самостійно.

Діалогова система має свій власний уніфікований API, що не налаштований на роботу з конкретним зовнішнім сервісом обміну повідомленнями, зокрема «Telegram». Таким чином, стає задача створення програмного шлюзу, що забезпечує взаємодію API діалогової системи і сервісу «Telegram».

Було розроблено програмний модуль на мові Python, який забезпечує взаємодію розташованої на сервері програмної діалогової підсистеми і програмного інтерфейсу сервісу обміну повідомленнями “Telegram”. Для спрощення опрацювання API “Telegram” було використано програмну бібліотеку “telebot”.

При запуску програма через API діалогової системи створює окрему специфічну діалогову сесію з нею. Специфіка цієї сесії в тому, що вона не має явно встановленого терміну закінчення.

Програма прослуховує повідомлення, які надходять до зазначеного Telegram-боту. Тексти цих повідомлень надсилаються за допомогою HTTP POST-запитів до діалогової системи, де стають у чергу на обробку. Програма надсилає запити діалоговій системі про статус готовності відповіді. Коли надходить повідомлення, що відповідь створено, надсилається запит для отримання цієї відповіді. Програма отримає відповіді діалогової системи і обробляє їх у відповідності до вимог форматування повідомлень, що надсилаються через API “Telegram”.

У повідомленнях надісланих у сервіс “Telegram” можуть створюватися кнопки для отримання додаткової інформації за її наявності. Для отримання цієї інформації при натисканні на таку кнопку створена спеціальна функція зворотного виклику. Якщо повідомлення, що надсилається містить посилання на зображення, то зображення оформлюються в окремі повідомлення, що містять саме зображення і його заголовок. Довгі тексти відповідей розбиваються на кілька повідомлень, не більше 4096 символів (вимоги до повідомлень у “Telegram”).

Програма є незалежним застосунком, що запускається однократно і далі працює неявно для користувачів (daemon).

Діалогова система повинна мати зручний інтерфейс користувача. Оскільки розроблювана система ТІСП є серверним застосунком, одним з найочевидніших рішень буде створення графічного інтерфейсу користувача у вигляді веб-сторінки. Іншим поширеним варіантом є підключення системи до роботи з сервісом обміну повідомленнями, наприклад, “Telegram”. В свою чергу, діалогова система повинна мати відповідний програмний інтерфейс, що забезпечує взаємодію між графічним інтерфейсом користувача і функціональною програмною частиною, розташованою на сервері.

Графічний web-інтерфейс користувача системи

Графічний інтерфейс у вигляді веб-сторінки було реалізовано у двох видах.

Інтефейс – web-сторінка. У першому на сторінці відображається історія діалогу і форма для відправлення користувачем запитань. Перед початком діалогу користувач бачить форму з кнопкою і пропозицію розпочати діалог, а також коротку інформацію про дану конкретну діалогову систему. Натискання цієї кнопки ініціює нову сесію діалогу. При цьому відображається сторінка з формою для відправки повідомлень, а також кнопка для завершення діалогу. При завершені користувачем діалогу відображається форма з запитанням, чи отримав користувач відповіді на свої питання і двома кнопками: «Так» і «Ні». Сторінка перезавантажується після кожного нового повідомлення. У блоках відповідей системи можуть також знаходитися кнопки для розгортання додаткових відповідей за їх наявності. При натисканні на таку кнопку відповідний текстовий інформаційний блок з’являється наприкінці ланцюжку діалогу. При відображенні нової відповіді системи сторінка після завантаження прокручується на позицію останньої відповіді. Якщо користувач більше ніж близько 20 хвилин не виявляє активності сесія діалогу завершується автоматично.

Цей тип інтерфейсу побудовано за допомогою шаблонів Django. Він вимагає перевантаження сторінки після для кожного діалогового акту зі сторони системи. Але після будь-якого перезавантаження сторінки історія поточного діалогу відображається заново з самого початку. Це зроблено за рахунок зберігання у базі даних системи історії діалогу. При перезавантаженні сторінки система запрошує історію діалогу, пов’язаного з конкретною сесією з даним користувачем і відновлює блоки повідомлень.

Інтерфейс – спливаюче вікно. У другій версії інтерфейс діалогу має вигляд вікна, розміщеного у куті сторінки і здатного згортатися і розгортатися. Користувач також має явну можливість розпочати і завершити діалог. Однак окрім візуальної цей варіант має і функціональну відмінність. А саме, після кожного діалогового акту сторінка не перезвавнтажується. Йде очікування відповіді, а потім її відображення в кінці ланцюжка діалогу. Така поведінка організована з використанням ajax-запитів до API діалогової системи. Так, запитання користувача відправляється на обробку за допомогою POST-запиту, у відповідь на який одержується ідентифікатор повідомлення. Далі слідує періодичне опитування API діалогової системи на предмет наявності відповіді. При отриманні задовільного результату відправляється запит на отримання матеріалу для формування відповіді. Отримавши його, за допомогою Java Script нове повідомлення динамічно створюється і розміщується у ланцюжку діалогу. Після перезавантаження сторінки, якщо діалог не було явно завершено, історія спів розмови також відтворюються. Тільки в цьому разі з цією метою відправляється відповідний POST-запит, у відповідь на який записи динамічно відтворюються у хронологічному прядку. Такий запит відправляється після завантаження сторінки, якщо існує наявна сесія. В іншому, функціонально цей варіант є ідентичним першому.

Задіяння сервісу “Telegram” у якості інтерфейсу користувача. Інтерфейс користувача у сервісі повідомлень “Telegram” обумовлюється насамперед реалізацію самого цього сервісу. Якщо даний користувач

розпочинає діалог (чат) із системою (ботом) в перший раз, то він має кнопку «розпочати діалог», що ініціює прив’язку програмного шлюзу до цього користувача і конкретного розпочатого їм чату. Користувач надсилає свої запитання через відповідне текстове поле, що надається сервісом. Користувач може надіслати ряд запитань, навіть не дочекавшись відповіді. Всі вони будуть оброблені, а відповіді на них будуть дані по мірі отримання їх у програмній системі. Якщо у відповіді наявні елементи додаткової інформації, вони будуть представлені у вигляді кнопок. Натискання на таку кнопку перехоплюється функцією зворотного виклику і у даний чат додається текст відповідного нового повідомлення. Графічні матеріали, за їх наявності у відповіді, подаються наприкінці відповідного текстового повідомлення-відповіді у вигляді окремих спеціальних повідомлень типу «зображення з підписом».

Якщо користувач у якості інтерфейсу застосовує сервіс обміну повідомленнями “Telegram”, то, якщо він звертається до системи вперше він повинен розпочати чат натисканням відповідної кнопки. Далі він вільний надсилати свої запитання до системи через відповідне текстове поле.

За аналогічним принципом нами також реалізовано ряд тестових діалогових систем, що працюють з англійською мовою.

4.4.3 Діалогова система з питань медичної реабілітації на базі набору статей EBSCO

Загальна характеристика системи. Для забезпечення медичної реабілітаційної підтримки розроблено інформаційну природномовну діалогову систему. Ця система в першу чергу призначена для використання медичними працівниками та студентами реабілітаційної медицини, слугуючи цінним інформаційним ресурсом. Вона використовує графову базу знань на основі онтології та не включає моделі нейронних мереж. Замість цього в ній реалізовано виявлення намірів та сутностей у вхідних текстах користувача на основі наявності слів-маркерів із певних списків. Цей підхід довів свою високу ефективність, пропонуючи як швидкість, так і ефективність ресурсів у

порівнянні з великими мовними моделями. База знань створюється автоматично на основі набору статей у вигляді PDF файлів, що описано у розділі 2 цієї роботи.

Система має природномовний інтерфейс користувача та надає API для взаємодії з іншими додатками через POST-запити заданої структури. Наразі користувальницький інтерфейс доступний у формі Telegram-бота.

Ключові алгоритми, що відповідають за роботу системи, реалізовані як серверний додаток на мові Python. Для взаємодії з онтологічною базою знань використовується СУБД Apache Jena Fuseki.

Структура системи. Програмна частина складається з таких модулів:

preprocess_input.py: цей модуль проводить первинний аналіз повідомлення користувача та визначає семантичні категорії (наміри), виражені в ньому.

form_queries.py: відповідає за створення пакетів SPARQL запитів.

process_queries.py: керує виконанням запитів на сервері Jena Fuseki та обробляє їх результати.

processor.py: організовує робочий процес модулів preprocess_input.py, form_queries.py і process_queries.py, починаючи від отримання вихідного тексту до генерації відповіді.

Додатково в систему входять такі модулі:

webhook.py: полегшує взаємодію зі службою обміну повідомленнями Telegram.

api.py: реалізує API для цієї програми.

Схема архітектури системи представлена на рисунку 4.4.

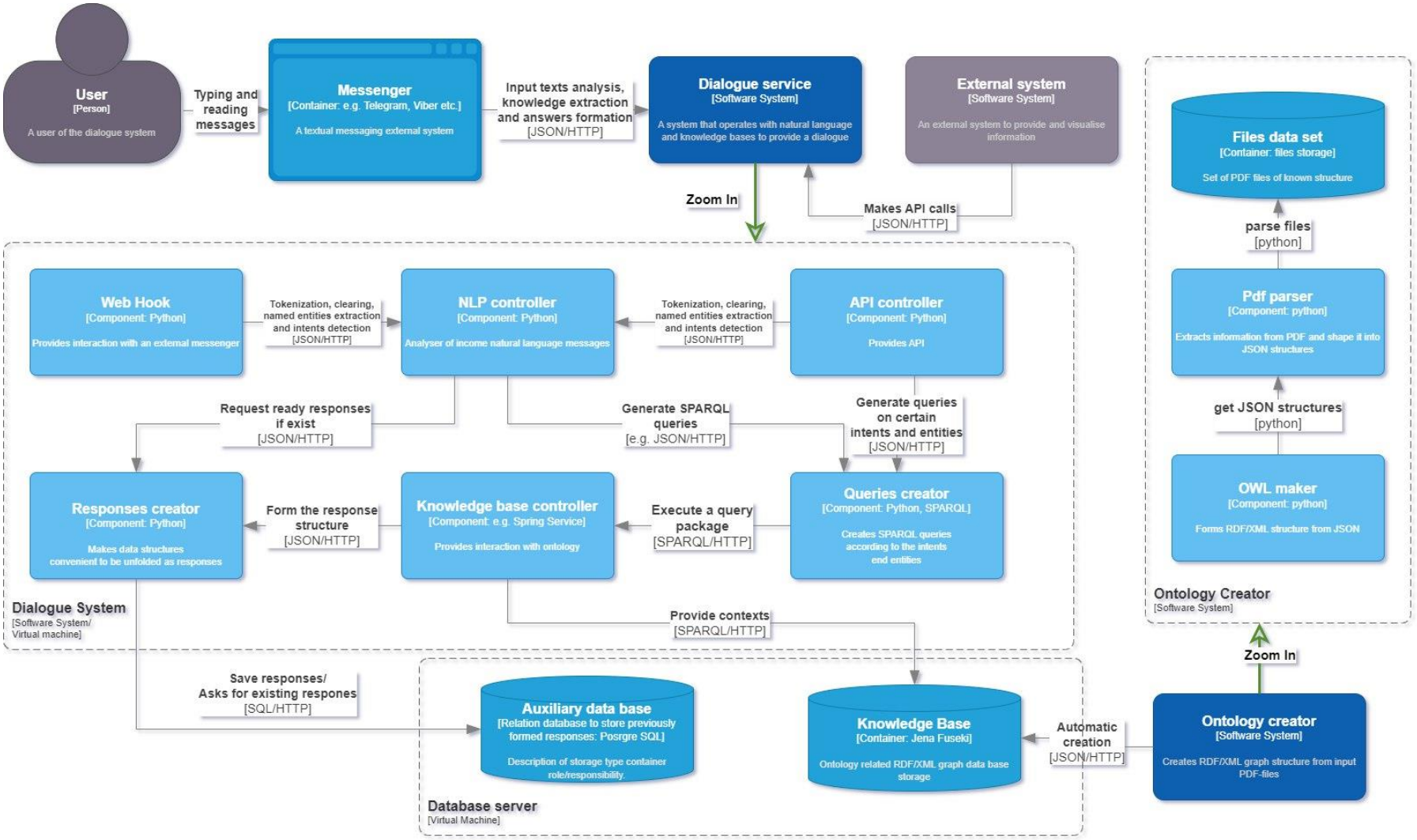


Рисунок 4.4 – Схема архітектури діалогової системи з питань медичної реабілітації на базі набору статей EBSCO

Функціонування системи

Первинна обробка повідомлень. Система отримує текстове повідомлення від користувача, яке може надходити або з інтерфейсу користувача, або через API. Спочатку текст очищається, щоб видалити будь-які незнайомі символи. За допомогою інструментів, наданих бібліотекою NLTK, текст розбивається на окремі слова, які потім лематизується, а неінформативні стоп-слова виключаються. Крім того, текст очищається від слів, яких немає в списку понять у базі знань. Таким чином, оброблений текст містить список значущих слів, приведених до їх основної форми.

Визначення семантичних категорій. Згодом система прагне визначити конкретні семантичні категорії або їх комбінацію, виражені в повідомленні користувача. Для цього кожна семантична категорія відповідає певному шаблону запиту SPARQL у системі. Зараз система охоплює 28 таких категорій, але допускає можливість розширення цієї кількості. Визначення семантичної категорії ґрунтується на наявності конкретних слів-маркерів у результуючому списку слів. Зіставлення цих категорій на слова-маркери встановлюється за допомогою словника, визначеного у файлі marker_words.json. У цьому словнику кожне відображення категорії представлено у вигляді списку або набору списків слів-маркерів. Щоб класифікувати введені користувачем дані, система перевіряє список введених слів за всіма доступними категоріями. Під час цього процесу програма відстежує, які слова-маркери зі списку відповідають кожній конкретній категорії. Якщо списки слів-маркерів збігаються, система вибирає категорію, пов’язану з найдовшим списком, за умови, що всі його слова присутні в аналізованому введенні. Слова, які належать до однієї синтаксико-семантичної групи (наприклад, речення або частина речення) і не позначені як маркери для жодної категорії, ідентифікуються як параметри запиту. Зрештою, семантична категорія визначає відповідний шаблон SPARQL запиту для подальшої обробки.

Формування і виконання SPARQL запитів. Модуль 'form_queries.py' відіграє ключову роль у формуванні SPARQL запитів у системі. Кожна спеціальна семантична категорія, визначена в системі, відповідає окремому шаблону SPARQL запиту. Ці шаблони зберігаються як словник JSON у файлі query_templates.json. У цьому словнику ключі представляють назви відповідних спеціальних семантичних категорій, тоді як значення є словниками, що містять такі ключі:

"verbose": цей ключ містить фрагмент фрази, яка передує основній відповіді.

"parts": містить список компонентів, які використовуються для побудови SPARQL запиту (основний розділ).

"outputs": цей ключ містить інструкції щодо організації структури відповіді на основі даних, отриманих під час виконання запиту.

Значення «parts» – це список, у якому кожен елемент відповідає сегменту запиту. Шаблон «parts» запиту – це словник, який містить такі ключі:

"type": вказує на тип частини. Доступні типи: «константа» (частина вставляється в запит без змін), «вхід зі списку» (частина замінюється на кожен із наданих параметрів/слів під час формування запиту) і «єдиний вхід» (один параметр замінюється один раз, а для наступних параметрів, якщо такі є, формується новий запит). "n": позначає порядок, у якому підставляється частина шаблону під час створення запиту, запобігаючи перемішуванню частин.

"body": Містить список рядків, які формують згенерований запит. Ці рядки можуть містити заповнювачі для параметрів, позначених як >input<, а також додаткові значення, позначені як >order<. Хоча розділ «outputs» разом із «verbose» безпосередньо не використовується модулем «form_queries.py», вони передаються разом із запитом і є життєво важливими для формування наступної відповіді.

Взаємодія з СУБД Jena Fuseki здійснюється модулем 'process_queries.py', який використовує інструменти з програмної бібліотеки SPARQLWrapper. Виконання SPARQL є навантажливим процесом, і для його прискорення запити до різних частин онтології виконуються паралельно в кількох потоках.

Обробка і представлення результатів запитів. Змістовні відповіді можна отримати з одного або кількох розділів онтології. Кожна відповідь є представленням JSON таблиці з полями, що відповідають запиту. Відповідь, отримана від Jena Fuseki на запити, має структуру погано придатну для прямого представлення користувачам в інтерфейсі діалогової системи або для відповідей API. Окрім метаданих, така відповідь містить таблицю результатів у форматі JSON, яка складається зі списку словників. Кожен елемент списку відповідає рядку в таблиці, представленій у вигляді словника, де ключі — це імена полів (стовпців), а значення містять відповідні дані. Для кращої взаємодії з користувачем бажано використовувати вкладену ієрархічну деревоподібну структуру. Цей модуль обробляє перетворення в таке представлення на основі інструкцій, наданих у розділах «виводів» шаблонів запитів для кожного типу запиту. Визнаючи, що виконання запитів SPARQL потребує ресурсів і часу, система містить механізм для зберігання попередньо обчислених відповідей. Для цього використовується невелика реляційна база даних, якою керує СУБД PostgreSQL.

Інтерфейси системи. Для реалізації користувальницького інтерфейсу створено Telegram-бот (@MedicalRehabBot). Взаємодія з цим ботом здійснюється модулем webhook.py, який служить точкою входу в систему. Бібліотека пакетів telebot використовується для взаємодії з API служби Telegram.

Окрім діалогового інтерфейсу користувача, система пропонує програмний інтерфейс (API), реалізований у модулі «api.py». Хоча цей модуль працює окремо від webhook.py, він має спільні основні модулі з системою.

4.4.4 Концепція діалогової системи OntoChatGPT, що включає комбінацію онтологічного підходу і великої мовної моделі

Структура системи. Розроблено прототип запропонованої системи що включає комбінацію онтологічного підходу і великої мовної моделі. Щоб забезпечити візуальне представлення загальної схеми системи, ми наводимо діаграму моделі C4 контекст/контейнер, як зображено на рисунку 4.5. Ця діаграма пропонує вичерпний огляд архітектури системи, демонструючи взаємодію між різними компонентами та їхніми взаємозв'язками. Вона служить наочним посібником для розуміння основної структури та функціональності системи OntoChatGPT.

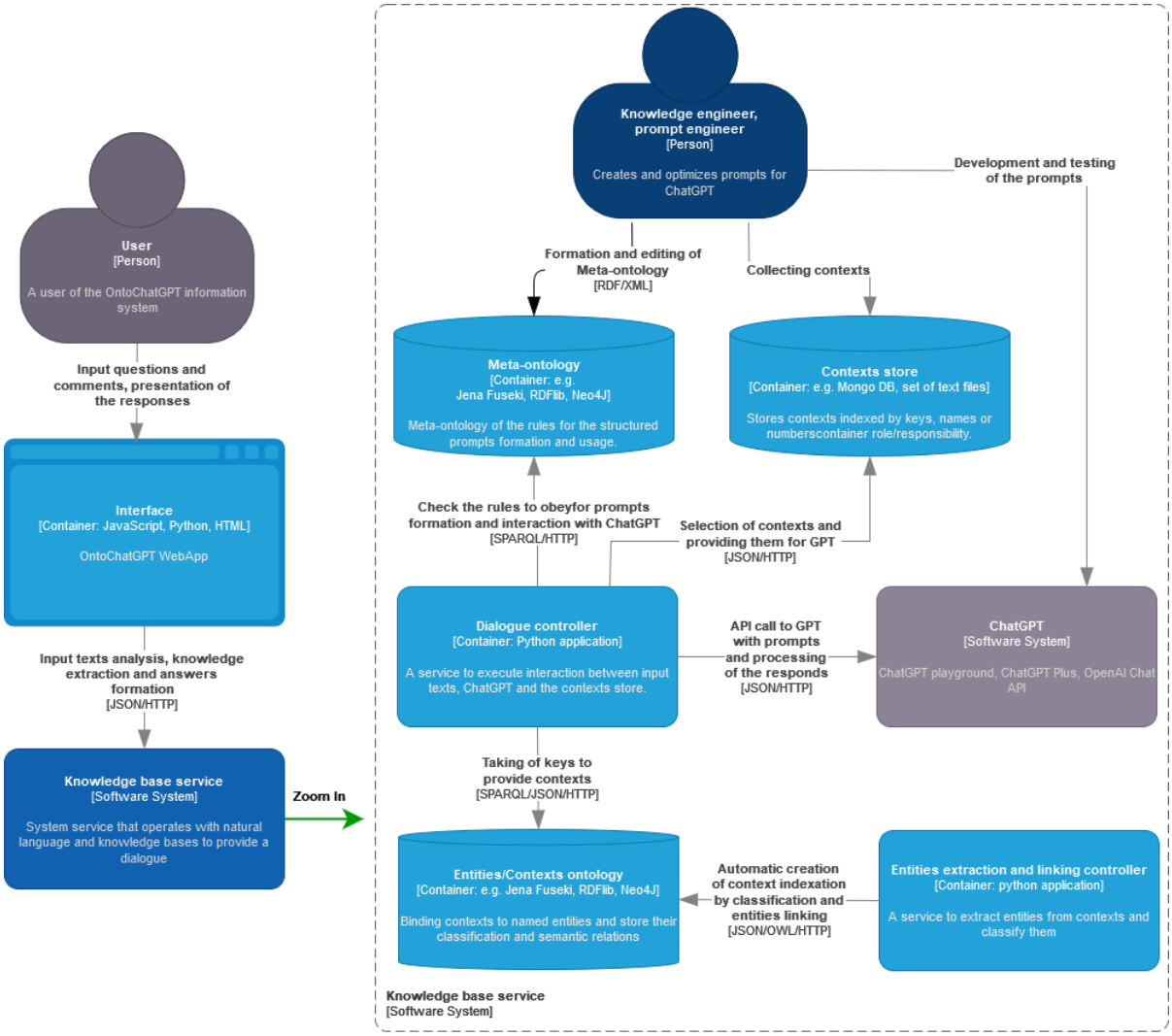


Рисунок 4.5 - Контекстна/контейнерна діаграма C4 моделі інформаційної системи OntoChatGPT.

Інструкції-підказки для аналізу вхідних повідомлень. Інструкції-підказки в системі формулюються на основі наданих правил і представляються у вигляді структур JSON. У схемі JSON із ***Додатку Є*** наведено приклад такої інструкції, яка зосереджується на визначенні намірів та їх зв'язку з відповідними сутностями, якщо такі присутні.

Це приклад короткої та простої підказки, яка охоплює кілька полів, що дозволяє вичерпно пояснити основні функції. Одним з важливих аспектів є інструкція `"information to provide"` («інформація, яку необхідно надати»), яка проголошує основні цілі завдання. Вхідне повідомлення для аналізу вказується в полі `"text"`. Для підвищення продуктивності доцільно вказати мову вхідного тексту, наприклад українську. Однак можна використовувати будь-яку іншу відповідну природну мову залежно від вмісту, який обробляється ChatGPT. Щоб дати вказівки ChatGPT і обмежити його можливості визначення намірів наперед визначеним списком, бажані наміри слід чітко вказати в полі `"possible intents"`. Кілька полів логічного типу надають технічну інформацію про результат і вигляд у якому його потрібно отримати. У цьому випадку вони включають `"several intents"`, `"intents probability"` і `"show intent subject"`. Перше поле дозволяє ідентифікувати кілька намірів у заданому тексті, друге поле дозволяє ChatGPT оцінити ймовірність присутності кожного з намірів у тексті, а останнє поле наказує ChatGPT визначити сутності, які конкретизують наміри. Щоб запобігти визначенню надмірної кількості намірів, їх кількість можна обмежити і відсортувати за ймовірністю.

Також потрібно визначати шаблон для структури вихідних даних, вказуючи формат, у якому інформація має повертатися в результаті. Таким чином, було включено поле `"output representation template"` («шаблон представлення вихідних даних»). Остаточні відповіді, надані ChatGPT на основі визначених намірів і вибраних контекстів, можуть відрізнятися залежно від самих намірів. Однак вони мають загальну структуру, яка є списком словників із полями `"intent"` (назва наміру) і

"results" (текст або список текстів чи інших структур даних) або просто «немає», якщо є немає відповідної інформації для надання.

Функціонування системи (приклад). Розглянемо приклад. На вході є фраза (українською): «На що повинна спиратися ФРМ?». Система розпізнає такі наміри:

```
[
      {
            "intent": "subject",
            "type": "interrogation",
            "probability": 0.8,
            "subject": "ФРМ",
            "object": null
      },
      {
            "intent": "cause",
            "type": "narration",
            "probability": 0.6,
            "subject": "ФРМ",
            "object": "спиратися"
      },
      {
            "intent": "way of doing",
            "type": "interrogation",
            "probability": 0.4,
            "subject": null,
            "object": "спиратися"
      }
]
```

Для вибору контекстів було знайдено такі іменовані сутності:

```
[
      {
            "words": ["ФРМ"],
            "type": "noun",
            "main word": "ФРМ"
      },
      {
            "words": ["повинна", "спиратися"],
            "type": "verb",
            "main word": "спиратися"
      }
]
```

Таким чином, ми маємо наступні наміри, які були визначені: «суб’єкт» (питання), «причина» (розповідь) і «спосіб дії» (питання). Усі ці наміри мають відносно високу ймовірність і їх слід враховувати для отримання остаточного набору відповідей.

Однак лише другий намір (причина/розповідь) призвів до того, що ChatGPT надав повністю правильну відповідь. Дивно, але це не був намір з найвищою ймовірністю. Інші наміри також призвели до вичерпних і лаконічних відповідей, але вони не мали прямого відношення до початкового запитання. Перший намір надавав інформацію про фізичну та реабілітаційну медицину і «Білу книгу з фізичної та реабілітаційної медицини в Європі», а третій намір зосереджувався на цілях Міжнародної класифікації функціонування, інвалідності та здоров'я (ICF). Хоча ця додаткова інформація може бути цікава для користувача, вона не має прямого відношення до вихідного запитання.

4.5 Допоміжне застосування для переведення описів онтологічних графів з формату XML "Graph Editor" у стандартний формат RDF/XML і у зворотному напрямку

Розроблені в Інституті кібернетики застосунки Graph Editor і КІТ «Поліедр» є зручними і потужними засобами для створення і візуалізації описів графових баз даних, в тому числі значної складності. Але формат зберігання даних графів в даному випадку є специфічним і тому не прийнятним для розбору за допомогою сторонніх програм і програмних бібліотек, що працюють з графовими базами даних. Поширеним і стандартизованим способом опису і зберігання графових баз даних онтологічного типу є мова OWL. Існує цілий ряд форматів для зберігання OWL-онтологій, серед них нами було обрано RDF/XML. Цей формат підтримується більшістю програм, СУБД і програмних бібліотек для роботи з графовими базами даних. Такі файли можуть бути, наприклад: розібрані, візуалізовані і відредаговані за допомогою застосунку «Protege»; використані програмною бібліотекою мови Python «RDFlib» у якості вхідних даних, до яких можуть бути застосовані запити на мові SPARLQ; конвертовані у

формат графової СУБД Neo4J. Так чи інакше, це є відомий стандартний формат.

Графові бази даних у наш час набувають поширеності, особливо для представлення даних з природною графовою структуро. Існує доволі багато випадків, коли графова база даних є значно кращим варіантом у порівнянні, наприклад, з реляційною чи іншим типом. База даних, в даному випадку графова, створюється щоб бути компонентом програмної системи. Таким чином, вона повинна мати формат, що легко сприймається машиною, в тому числі з використанням сторонніх програмних інструментів. Для цього підходить формат RDF/XML. В той же час для ручного створення і редагуванні опису графової бази даних потребується зручний, потужний і доступний редактор з графічним інтерфейсом. Нажаль, редактори, що працюють зі стандартними форматами далеко не завжди здатні задовольнити потреби розробника. Але таким можна вважати КІТ «Поліедр». Таким чином, стає актуальною задача створення конвертору, що переводить файл з формату Graph Editor у стандартний формат RDF/XML, еквівалентний за представленою в ньому інформацією.

Також важливою є задача і зворотної конвертації RDF/XML-файлів у формат Graph Editor. Це обумовлено тим, що у форматі RDF/XML представлено чимало сторонніх графових баз даних. Для їх зручного редагування і перегляду у КІТ «Поліедр» треба перевести у відповідний формат. Також деякі застосунки працюють саме з форматом Graph Editor.

Для реалізації застосунку для конвертації описів графових баз даних у формат RDF/XML з формату Graph Editor і у зворотному напрямку була створена програмна система, що являє собою web-сервіс. Для написання програмного коду використовувалася переважно мова Python – серверна частина, та Java Script – клієнтська частина.

Принципова схема роботи програмної системи наочно представлена на рисунку 4.6 у вигляді UML-діаграми.

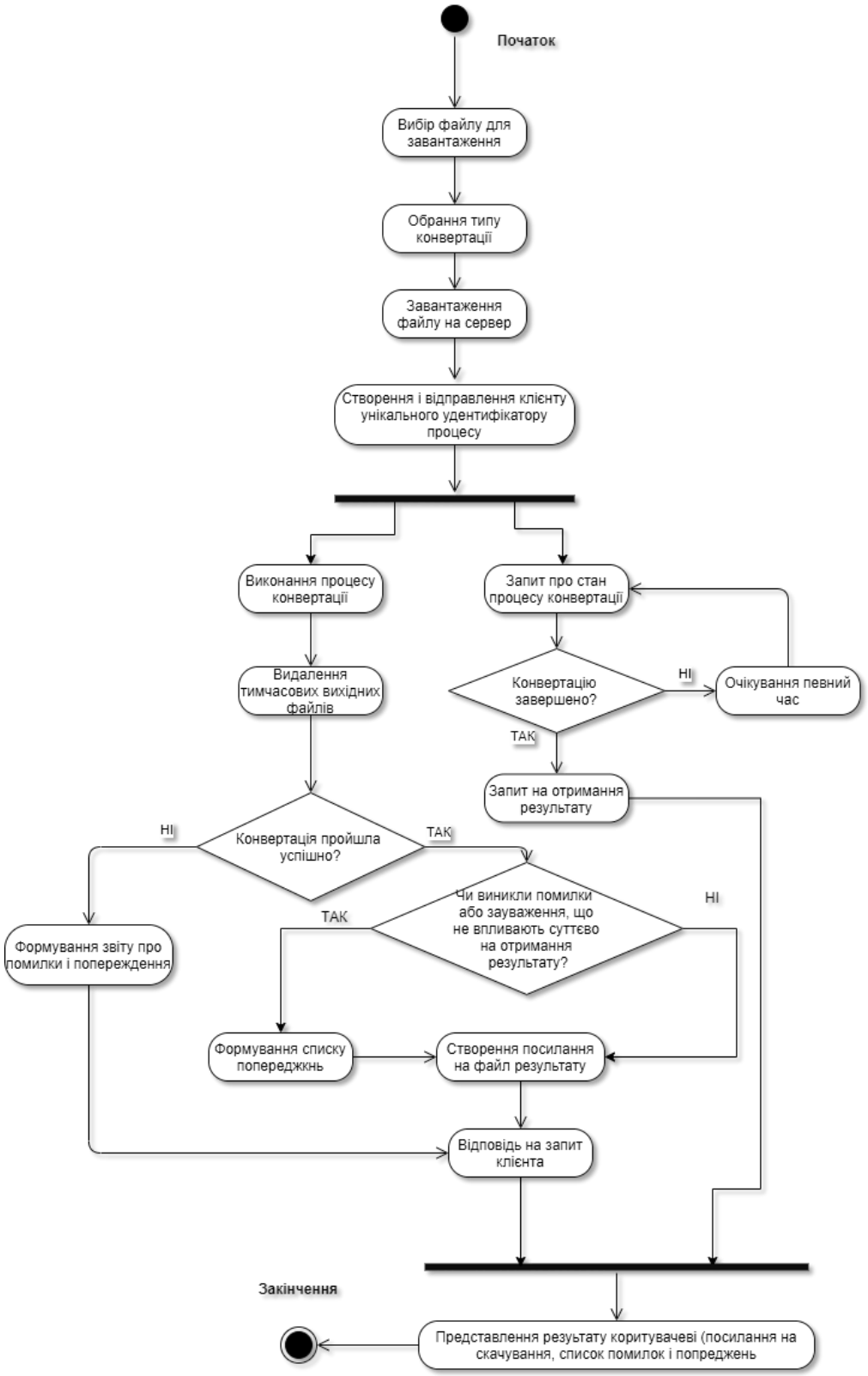


Рисунок 4.6 – UML-діаграма дій, що описує загальну процедуру конвертації у системі

Коротко роботу системи можна охарактеризувати наступним чином. Користувач обирає файл для завантаженні з метою конвертації. Після цього

стають активними кнопки для двох варіантів режиму конвертації. Натискання такої кнопки відправляє на сервер файл і інформацію про тип конвертації, яку потрібно виконати. Також на сервер відправляється інформація, як інтерпретувати неіменовані зв’язки у файлі формату Graph Editor (актуально тільки для нього). Остання інформація береться зі стану відміченості відповідного поля. Так, вони можуть бути інтерпретовані як «є підкласом», або як зв'язок «за замовчуванням».

Отриманий файл записується на сервері у певну директорію. Клієнт отримає унікальний ідентифікатор процесу (послідовність символів), за яким він може направляти запити про стан конвертації на даний момент. Цей ідентифікатор, а також технічна інформація (ім’я файлу, час початку процесу, поточний стан конвертації – завершено або ні, список помилок і попереджень) записується у допоміжну базу даних словникового типу під управлінням Redis.

У паралельному потоці стартує відповідний процес конвертації. Запуск паралельного процесу дозволяє відправити клієнту швидку відповідь, не очікуючи завершення конвертації. Це дозволяє уникнути технічної затримки відповіді серверу. Далі клієнт відправляє періодичні ajax-запити про стан процесу конвертації (чи його завершено?).

В ході даного процесу інформація про помилки і попередження, у разі їх виникнення записується у базу даних словникового типу під управлінням Redis. Після завершення конвертації, успішного чи ні, створюється відповідний запис у Redis-словнику. Файл з результатами, якщо його успішно отримано, записується у відповідну директорію на сервері. Вихідний тимчасовий файл видаляється з серверу.

Якщо після чергового запиту від клієнта виявляється, що конвертацію завершено, клієнту відправляється повідомлення, що містить посилання на файл результату на сервері, якщо такий було успішно створено, і список помилок і попереджень, за наявності таких. Отримавши таку відповідь від сервера, клієнт відображає у інтерфейсі користувача посилання для

завантаження файлу, а також список помилок і попереджень, за наявності таких. Також відправляється строк дійсності посилання (фактично термін зберігання файлу результатів на сервері).

На сервері йде безперервний періодичний паралельний процес «збирання сміття». Цей процес видаляє старі файли результатів, відповідні записи у Redis, а також об’єкти завершених процесів конвертації з пам’яті комп’ютера.

Безпосередньо процеси конвертації суттєво відрізняються залежно від напрямку. Процес конвертації файлу з формату Graph Editor у RDF/XML можна коротко охарактеризувати наступними основними етапами:

- створення з записів вихідного файлу списків структур словникового типу, кожна з яких характеризує відповідну вершину, зв'язок, або групу зв’язків;

- створення програмного об’єктного представлення OWL-сутностей (класів, властивостей, іменованих сутностей) з вершин вихідного графу;

- формування опису типізованих структур даних, що містяться у вершинах графу, прийнятної для висловлення засобами RDF/XML;

- формування у створених програмних об’єктних представленнях OWL-сутностей зв’язків на основі інформації про зв’язки у вихідному файлі;

- формування технічних властивостей (ObjectProperty) для зберігання розширених даних зв’язків (при їх наявності);

- серіалізація створених програмних об’єктних представлень OWL-сутностей у форматі RDF/XML.

Процес зворотної конвертації протікає наступним чином:

- створюється програмне об’єктне представлення вершин графу (Node) на основі наявних OWL-сутностей;

- створення програмного об’єктного представлення зв’язків (Edge) на основі інформації, що міститься у відповідних OWL-сутностях;

- формуються структури вкладених структур даних у вершинах графу за їх наявності на основі відповідної інформації в OWL-сутностях;

- серіалізація програмного об’єктного представлення вершин, зв’язків і груп графу у форматі XML, прийнятному щодо роботи у Graph Editor.

4.6 Висновки за розділом 4

Проаналізовано діалогові систем різних типів. Визначено особливості і типові задачі, що виникають при їх створенні. Зокрема, їх можна умовно поділити на наступні типи: довідкові системи, у яких основна ініціатива йде від користувача, який задає питання; системи автоматизованого опитування, у яких ініціатива у діалозі йде цілковито збоку програми, а наступні репліки системи можуть визначатися попередньою історією діалогу (відповідями користувача); діалогова система віртуального співрозмовника, у якій ініціативу у діалозі поділяють програма і користувач; природномовні керуючі та контрольні інтерфейси.

Вказано, що для кожного з названих типів діалогових систем перед розробником стоїть задача створення чи застосування розроблених методів автоматизованого семантичного аналізу коротких природномовних текстів, якими є фрази користувача. Під розумінням системою семантики таких фраз мається на увазі насамперед три основні задачі: визначення основної тематики фрази для обрання відповідної онтології для подальшої роботи; визначення основних намірів, що їх висловлює користувач своєю фразою, та відповідних мовних сутностей, що ці наміри конкретизують; побудова формального запиту чи їх пакету для отримання з онтології інформації, відповідної до намірів, висловлених користувачем. Розроблені нами методології такого аналізу наведені у розділі 2.

Проаналізовано проблему утримання контексту діалогу у автоматичних діалогових системах та її актуальність для систем різного типу. Так, ця проблема майже не стоїть у явному вигляді для систем довідкового типу. Користувач вільний задавати будь-які питання, що не виходять за рамки зазначеної предметної області. Якщо система налаштована на декілька предметних областей, то скоріше стоїть задача визначення такої з контексту

поточної фрази і швидке переключення на неї. У системах автоматичного опитування контекст подальших реплік системи визначається цілковито програмно і може бути або жорстко заданим, або визначатися на основі дерева діалогу – умовами переходу до наступної вершини такого дерева є відповіді користувача. Система віртуального співрозмовника потребує особливої уваги до збереження контексту діалогу, тому у ній повинна бути реалізована довгострокова і короткострокова пам'ять історії поточного діалогу. Під довгостроковою пам’яттю тут розуміється знаходження у рамках визначеної на початку розмови переметної області і відповідної онтології. Тому при виникненні декількох варіантів онтології за результатами аналізу фрази, що включають і поточну перевагу слід віддавати їй. При цілковитому переключенні на іншу предметну область, відповідь має містити попередження про цей факт користувача. Розглянута тут короткострокова п’янять діалогової системи поширюється на одну-дві попередні фрази. У її рамках розглядаються дві наступні основні проблеми: заміна займенників і наявність очікування від користувача фрази певного типу. Тобто, наприклад, якщо програма задає питання, то очікуванням від користувача є відповідь на питання не залежно від структури фрази, що напише користувач. Більш того, інтерпретація відповіді користувача повинна бути у рамках тематики попередньої фрази програми. Тобто, наприклад, якщо система запитувала назву фільму, то у відповіді користувача система має намогтися знайти назву фільму і діяти далі відповідним, згідно дерева ходу діалогу чином.

Розроблено мультиагентний підхід до розробки природномовних діалогових систем, що містять в своїй основі онтологію. Складовими частинами такої системи є програмні агенти – сервіси, що можуть взаємодіяти між собою і здатні приймати самостійні рішення щодо характеру взаємодії. Програмні агенти мають вузькі задачі, що виконують: аналіз вихідної фрази, побудова формальних запитів, взаємодія з онтологією і виконання запитів, формування і ранжування відповідей, взаємодія з

інтерфейсом користувача і забезпечення цього інтерфейсу, збір і аналіз вихідних даних для наповнення онтології, координація роботи інших агентів.

Описано приклади реалізації розроблених підходів природномовної людино-машинної взаємодії при створенні конкретних діалогових систем, що працюють з українською мовою.

РОЗДІЛ 5

ТЕСТУВАННЯ І ОЦІНКА ЯКОСТІ РОЗРОБЛЕНИХ АВТОМАТИЗОВАНИХ ДІАЛОГОВИХ СИСТЕМ

5.1 Діалогова система без залучення великих мовних моделей

Проводилося тестування дієздатності розроблених підходів до створення автоматизованих діалогових систем, що містять семантично структуровану онтологію створену автоматично на базі природномовного тексту. Усі використані онтології були представлені у форматі OWL (RDF/XML), а мова SPARQL використовується для запитів до них. Графова СУБД Neo4j і запити Cypher використовуються в іншій розробленій нами діалоговій системі за словником по «Білій книзі з ФРМ».

Для тестування роботи системи було проведено серію експериментів. Як вхідні дані використано природномовні (українські) питальні та наказові словосполучення. Для оцінки отримання негативних і позитивних результатів 30 фраз мали безпосереднє відношення до розглянутої в онтології теми і відповідь на них мала міститися в онтології, інші 30 були зроблені спеціально для отримання негативного результату. Ці фрази для отримання негативного результату не були абсолютно беззмістовними чи граматично неправильними (оскільки для таких не існує певного позитивного результату). Фактично їх можна поділити на три групи: у першій групі фрази які хоч і присвячені темі онтології, але передбачуваної відповіді на них у онтології немає (20 фраз), у другій – із суміжної предметної області, яка так чи інакше перетинається із темою онтології (10 фраз), а в третій – фрази з іншої предметної області, але такі що можуть містити окремі слова (синоніми понять) із області онтології (10 фраз). Загалом у тестуванні було використано набір із 60 тестових фраз, усі фрази були граматично правильними українськими реченнями. Відповідь системи вважалася позитивною, якщо

відповідь була інформативною та відповідною меті питання. Дійсно негативним результатом є відсутність запитуваної інформації (але не неправильної чи не релевантної), але лише у випадку, коли ми цього очікували. Як хибно негативні ми вважали всі результати, де очікувалася певна відповідь, але була отримана неправильна відповідь, неповна відповідь або відсутність відповіді. Хибно позитивний результат може містити будь-яку інформацію (повну чи неповну, можливо навіть не зовсім правильну) у випадку, коли очікувалася відсутність відповіді. Отриманий масив результатів був використаний для розрахунку критеріїв точності, і відповідності. Також вимірювалися часові інтервали процесу аналізу фрази та формування запиту. Цей час включає лише ці процеси, не час виконання запиту, надсилання повідомлень між агентом і користувачем, візуалізацію сторінки тощо.

Отримані в ході випробувань результати розмиву для кожного виду наведені в таблиці 5.1.

Таблиця 5.1 – Результати експерименту

Тип результату	Кількість результатів
Позитивні (T_p)	24
Негативні (T_n)	29
Хибно негативні (F_n)	6
Хибно позитивні (F_p)	1

Для перевірки якості результатів розробленої системи використано наступні критерії: accuracy (A), precision (P), recall (R) і критерій F1, які є загальноприйнятими для такого роду систем. Ці показники залежать від наступних факторів: кількості позитивних результатів (T_p), негативних результатів (T_n), хибно-позитивних результатів (F_p) і хибно-негативних результатів (F_n) відповідно:

$$A = \frac{T_p + T_n}{T_p + F_p + F_n + T_n} \tag{5.1}$$

$$P = \frac{T_p}{T_p + F_p} \tag{5.2}$$

$$R = \frac{T_p}{T_p + F_n} \tag{5.3}$$

$$F_1 = \frac{2 \cdot P \cdot R}{P + R} \tag{5.4}$$

Значення формальних критеріїв оцінки наведені в таблиці 5.2. Через брак сталих назв цих критеріїв українською мовою вони наведені англійською мовою.

Таблиця 5.2 – Значення критеріїв точності роботи системи

Критерій	Значення
Accuracy (A)	0.883
Precision (P)	0.960
Recall (R)	0.800
F_1	0.873

Слід зауважити, що одна відповідь, яка розглядається тут як справжній хибно-позитивний результат, насправді не є повністю правильною відповіддю, але справжнім негативним результатом для нього має бути «немає відповіді». З хибно-негативних результатів фактично в чотирьох випадках відповіді не було, а в двох відповідь була лише неправильною.

Середній часовий інтервал процесу аналізу та формування запиту становив 6±3 мс. Будь-якого статистично значущого уповільнення від довжини фрази та відповіді не спостерігалося.

5.2 Діалогова система що використовує великі мовні моделі як один із компонентів – OntoChatGPT

Запропоновану методику комбінування онтолого-керованого підходу і великих мовних моделей було апробовано на основі діалогової системи за матеріалами першого розділу української версії «Білої книги з фізичної та

реабілітаційної медицини в Європі» як предметної області, яка послужила основою для побудови контекстної онтології. Відповіді, отримані системою, були класифіковані за такими категоріями:

• Дійсно позитивні: відповідь надано ChatGPT і вона була правильною.

• Дійсно негативні: ChatGPT не надав відповіді, що вказує на те, що він або визнав брак знань, або вказав на недостатність інформації у переданих контекстах. Ця категорія також включає випадки, коли ChatGPT повертав «Немає» як відповідь, коли відповідна інформація справді була відсутня в контекстах.

• Хибно позитивниі: система спробувала надати відповідь, але вона була неправильною.

• Хибно-негативниі: відповідь не була надана ChatGPT, навіть якщо правильна відповідь була присутня в контекстах.

Важливо відзначити, що етап тестування виключав питання та фрази з непов’язаних предметних областей, які мали імена сутностей, відсутні у онтології контексту. Для таких випадків дійсно негативний результат є гарантованим, оскільки відповідні контексти не будуть вибрані та подальша їх обробка не може бути здійснена. Тому всі запити, включені в тестування, були сформульовані таким чином, щоб пройти всі етапи запропонованого підходу. Крім того, слід підкреслити, що запропонований підхід передбачає можливість множинних відповідей на одне запитання, насамперед через наявність можливості визначення кількох намірів у вхідному природномовному запиті. У процесі оцінювання враховувалися всі надані відповіді, незалежно від того, надані вони у відповідь на однакові чи різні запитання.

Ці міркування забезпечують комплексну оцінку продуктивності системи та її здатності обробляти різні запити, враховуючи визначені наміри та витягаючи релевантну інформацію з вибраних контекстів. Результати тестування наведені в таблиці 5.3.

Таблиця 5.3 – Результати експерименту

Тип результату	Кількість результатів
Позитивні (T_p)	17
Негативні (T_n)	7
Хибно негативні (F_n)	1
Хибно позитивні (F_p)	9

Під час тестування запропонованого методу отримано наступні значення стандартних оціночних показників наведені у таблиці 5.4.

Таблиця 5.4 – Значення критеріїв точності роботи системи

Критерій	Значення
Accuracy (A)	0.7059
Precision (P)	0.6534
Recall (R)	0.9444
F_1	0.7724

Ці показники забезпечують кількісну оцінку продуктивності системи з точки зору її точності, запам’ятовування та загальної ефективності. Показник Accuracy являє собою частку правильних відповідей, наданих системою, порівняно із загальною кількістю запитів. Показник Precision вимірює здатність системи надавати точні відповіді серед відповідей, які вона генерує. Показник Recall вказує на здатність системи отримувати всі релевантні відповіді з доступних контекстів. Оцінка F1 поєднує в собі Precision і Recall, щоб забезпечити збалансований показник загальної продуктивності.

На додаток до стандартних показників ми також врахували додаткові критерії, а саме Precision* і Recall* - формули (5.5, 5.6). Ці показники відрізняються від стандартних показників Precision і Recall тим, що вони сприймають як дійсно позитивні, так і дійсно негативні результати як

позитивні результати, не розрізняючи їх. Ці додаткові критерії забезпечують ширшу оцінку ефективності системи в отриманні правдивих відповідей і ідентифікації релевантної інформації з контекстів. Розглядаючи як позитивні, так і негативні результати, ми отримуємо більш повне розуміння продуктивності системи з точки зору точності та запам’ятовування.

$$Precission^* = \frac{TP + TN}{TP + TN + FP} = 0.7273 \tag{5.5}$$

$$Recall^* = \frac{TP + TN}{TP + TN + FN} = 0.96 \tag{5.6}$$

Таким чином: $F1^* = 0.8276$.

Отримані значення метрик демонструють потенційну придатність запропонованого методу, хоча ще є можливості для вдосконалення. Одним із головних недоліків, виявлених у поточній реалізації, є високий рівень хибно позитивних відповідей, що призводить до відносно низького значення точності. Таку поведінку можна пояснити схильністю ChatGPT намагатися надати відповідь, навіть якщо інформації в певних контекстах недостатньо. Крім того, визначені можливі наміри не завжди можуть ідеально узгоджуватися з наданим повідомленням, хоча таким намірам часто призначається відносно низька ймовірність. Однак варто зазначити, що багато з цих хибно позитивних відповідей супроводжувалися дійсно позитивними відповідями. Іншими словами, хоча в деяких випадках була надана неправильна відповідь, разом із нею була дана і правильна відповідь. Такі супроводжувані хибно позитивні відповіді можна розглядати як додаткову інформацію, яка може бути дотичною до основної відповіді. Хоча наявність хибних спрацьовувань впливає на значення точності, той факт, що вони часто співіснують із справжніми спрацьовуваннями, свідчить про те, що система здатна надавати додаткову статистику чи пов’язану інформацію. Це спостереження підкреслює потенційну цінність розгляду супроводжуваних хибно позитивних відповідей у практичному контексті. Вирішення проблеми

помилкових спрацьовувань і уточнення узгодження між можливими намірами та вмістом повідомлення є областями для подальшого вдосконалення для підвищення точності системи.

5.3 Експеримент зі зворотного синтезу природномовних речень на основі їх онтологічного представлення із залученням великої мовної моделі

5.3.1 Сутність експерименту

Для зворотного синтезу природномовних речень українською мовою на базі онтологічного представлення було використано методику, описану у підрозділі 3.4.2 цієї роботи.

Сутність експерименту полягала у наступному. З тестової онтології, створеної на базі тексту «Склад обчислювальної системи», за допомогою запиту на мові Cypher (див. пункт 3.4.2.1) витягувалися окремі речення та відповідні їм пари сутностей із семантичними категоріями, що їх поєднують у рамках даного речення. Далі за допомогою спеціальної інструкції-підказки (див. пункт. 3.4.2.2) великій мовній моделі (ChatGPT) передавалося завдання по створенню граматично правильного речення українською мовою на основі набору пар сутностей за умов вказаних семантичних зв’язків між ними. Інструкція підказка дає роз’яснення моделі, що саме являють наявні семантичні категорії, зокрема, в аспекті створення речення. Саме оригінальне речення не подавалося на вхід системі.

У якості результату поверталося сформоване речення і власна оцінка моделі вірогідності того, що речення відтворено правильно.

Для випробування було використано 10 речень із вказаного тексту.

5.3.2 Метод оцінки якості відтворення природномовного речення із онтологічного переставлення із застосуванням косинусної подібності

Для порівняння сформованого на основі онтологічного представлення реченням із оригінальним було застосовано показник косинусної подібності.

Косинусна подібність – це міра подібності між двома векторами передгільбертова простору, яка використовується для вимірювання косинуса кута між ними. Так, якщо є два вектори ознак (A і B), то косинусна подібність, cos(θ), може бути представлена із використанням скалярного добутку і норми (5.7):

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (5.7)$$

У випадку інформаційного пошуку, косинусна подібність двох документів змінюється від 0 до 1, оскільки частота терміна (ваги tf-idf) не може бути від'ємною, а кут між двома векторами частоти терміна не може бути більше ніж 90 °. Косинусна подібність ефективна у якості оціночної міри, особливо для розріджених векторів, тому як необхідно враховувати тільки ненульові виміри.

«М'яка» косинусна міра враховує подібність між парами ознак. Традиційна косинусна подібність розглядає ознаки векторної моделі як незалежні або повністю відокремлені, тоді як «м'яка» косинусна міра враховує подібність ознак у векторній моделі. Це дозволяє узагальнити ідею косинусної міри, а також ідею подібності об'єктів у векторному просторі.

В області обробки природної мови подібність між об'єктами доволі інтуїтивна. Такі ознаки як слова, N-грами або синтаксичні N-грами можуть бути доволі подібні, хоча формально вважатися різними ознаками векторної моделі. У випадку N-грам або синтаксичних N-грам може бути застосована відстань Левенштейна (окрім того, відстань Левенштейна може також бути застосована до слів).

Для розрахунку «м'якої» косинусної міри вводиться матриця s подібності між ознаками. Вона розраховується, використовуючи відстань Левенштейна або інші міри подібності, наприклад, різноманітні міри подібності Wordnet. Потім здійснюється помноження із використанням даної матриці.

Так, якщо є два N-мірних вектори a і b, м'яка косинусна міра обчислюється наступним чином (5.8):

$$soft\_\cos ine_1(a,b) = \frac{\sum_{i,j}^{N} s_{ij} \cdot a_i \cdot b_j}{\sqrt{\sum_{i,j}^{N} s_{ij} \cdot a_i \cdot a_j} \cdot \sqrt{\sum_{i,j}^{N} s_{ij} \cdot b_i \cdot b_j}} \quad (5.8)$$

де s_{ij} – подібність (ознака$_i$, ознака$_j$).

За відсутності подібності між ознаками ($s_{ii} = 1$, $s_{ij}=0$ для $i \neq j$), рівняння (5.8) еквівалентне загальноприйнятій формулі косинусної подібності.

Оскільки, строго кажучи, проводити математичні обчислення над строками неможливо, а обчислення такої міри як косинусна подібність вимагає наявності векторів, то природномовні тексти для обробки і аналізу підлягають векторизації. У контексті обробки природної мови (NLP) речення представляються як вектори у багатомірному просторі, використовуючи такі методи, як embeddings слів. Для отримання векторного переставлення речень у нашому випадку використовувалася програмна Python бібліотека spaCy (були задіяні мовні моделі uk_core_news_lg – для української мови та xx_ent_wiki_sm – мультиязична), а також метод TF-IDF. Реалізовані у spaCy методи були задіяні і для обчислення значень косинусної подібності.

5.3.3 Результати експерименту і їх обговорення

Значения кількісних оцінок, що характеризують наближеність сформованого за допомогою великої мовної моделі речення до оригіналу наведені у таблиці 5.5. У таблиці порівнюються значення отриманої косинусної подібності за умов різних способів векторного представлення речень, що аналізуються (оригінального і сформованого). Також наведена власна оцінка вірогідності вірного відтворення від ChatGPT, що не може

вважатися у повній мірі об’єктивною мірою, а скоріше слугує орієнтиром і оцінкою ступеню самокритики моделі GPT.

Із наведених результатів видно, що кількісне значення оцінки косинусної подібності сильно залежить від способу векторного представлення аналізованих текстів.

Таблиця 5.5 – Кількісні оцінки якості зворотного синтезу речень із онтологічного представлення

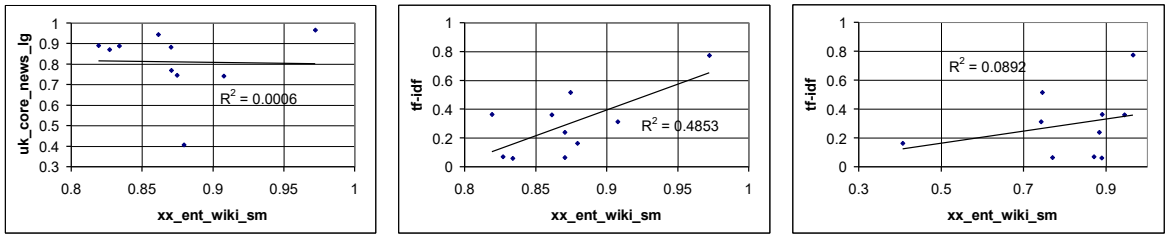
Оцінка вірогідності вірного відтворення фрази ChatGPT		Косинутна подібність					
		Модель xx_ent_wiki_sm		Модель uk_core_news_lg		Метод tf-idf	
Середнє значення ± довірчій інтервал	Інтервал варіювання	Середнє значення ± довірчій інтервал	Інтервал варіювання	Середнє значення ± довірчій інтервал	Інтервал варіювання	Середнє значення ± довірчій інтервал	Інтервал варіювання
0,845 ±0,037	0,75 – 0,90	0,8716 ±0,0335	0,8193 – 0,9722	0,8108 ±0,1224	0,4067 – 0,9653	0,2927 ±0,1718	0,0607 – 0,7745

Відразу видно, що мовні моделі векторизації xx_ent_wiki_sm та uk_core_news_lg приводять до досить високих значень косинусної подібності (0,8716 і 0,8108, відповідно). В той же час більш простий метод векторизації, заснований на tf-idf дає суттєво менші середні значення і великий інтервал варіювання. Проаналізуємо таке поводження.

Модель xx_ent_wiki_sm (мультиязична) дає малий інтервал варіювання і досить високе середнє значення косинусної подібності. Зменшення середнього показника при використанні моделі uk_core_news_lg (для української мови) обумовлено більшим варіюванням у нижню сторону. При цьому максимальні отримання значення для цих двох моделей є досить близькими. Простіше кажучи, застосування моделі uk_core_news_lg у ряді випадків призводить до отримання значно нижчої оцінки косинусної подібності.

Співставлення показників косинусної подібності, отриманих за моделями векторизації xx_ent_wiki_sm та uk_core_news_lg, наведене на

рисунку 5.1 (а), показує відсутність хоч якоїсь значущої кореляції між отриманими значеннями. Значення R^2 складає лише 0,0006. Ці моделі дещо по різному сприймають природномовний текст.



а) б) в)

Рисунок 5.1 – Співставлення значень косинусної подібності оригінальних і сформованих речень при векторизації тексту за різними моделями:

а) xx_ent_wiki_sm / uk_core_news_lg;

б) xx_ent_wiki_sm / tf-idf;

в) uk_core_news_lg / tf-idf

Аналіз безпосередньо згенерованих речень показав, що при векторизації за моделлю uk_core_news_lg знижені оцінки косинусної подібності виникають у випадках генерації складного розгалуженого речення, в той час коли оригінальне речення є значно простішим, хоча і близьким за змістом. Наприклад оригінальне речення мало вигляд: «Жорсткий диск (вінчестер) також входить в системний блок.». З його онтологічного представлення великою мовною моделлю було відтворено наступне речення: «Диск, відомий як вінчестер, є жорстким, входить у блок, який є системним.». Очевидно, що друге (згенероване) речення виглядає дещо неприроднім, наповненим надлишковими сутностями і зворотами. Хоча за переданим змістом воно і близьке до оригіналу. Модель xx_ent_wiki_sm виглядає менш чутливою до таких проявів і дає оцінку косинусної подібності 0.8795 (близьку до середнього значення по вибірці). В той же час модель uk_core_news_lg дає показник 0.4067, що є значно нижчим. Подібну

тенденцію можна прослідкувати неодноразово, хоча і з меншою різницею – модель uk_core_news_lg більш чутлива до утворення формально перекручених надлишкових фраз, в той же час xx_ent_wiki_sm більше схильна до аналізу змісту, а не форми.

Але, як ми бачимо з графіку, чіткої кореляції між моделями немає. Тобто наявні і випадки, коли uk_core_news_lg дає вищу оцінку косинусної подібності, а xx_ent_wiki_sm, відповідно, нижчу. Аналіз конкретних випадків дає підстави вважати, що у таких випадках при збереженні близької до оригіналу лексичної і синтаксичної структури було дещо перекручено зміст. Прикладом такого може слугувати наступний випадок. Оригінальна фраза: «Головним пристроєм комп’ютера є центральний процесор.»; згенерована фраза: «Пристрій, який належить до комп'ютера, є об'єктом, де центральний процесор є головним пристроєм.». Тут також ми спостерігаємо спотворення стилістики фрази на виході. Але також має місце і перекручування змісту. Тобто, у сгенерованому реченні виходить, що у комп’ютера є якійсь пристрій, у якому головним є центральний процесор, що не зовсім відповідає оригінальному реченню.

В той же час у показників косинусної подібності а моделлю xx_ent_wiki_sm спостерігається досить помітна кореляція із показниками, отриманими для методу tf-idf. Найвищі показники за цими методами були отримані тоді, коли речення практично повністю співпадали: «Вміст цієї пам’яті зберігається лише при увімкненому живленні.» і «Вміст пам'яті зберігається лише при увімкненому живленні». Щодо найнижчих, показників, то tf-idf виявляється також більш чутливим до деформованих речень, тому показники будуть чисельно нижчими. Проте кореляція за змістом зберігається. В той же час модель uk_core_news_lg слабо корелює із методом tf-idf.

Таким чином, якщо нам мало важлива форма представлення, а важливішим є замість, то можна скористуватися векторизацією за моделлю xx_ent_wiki_sm. В той же час, модель uk_core_news_lg є чутливою, як до

змісту, так і до форми переставлення. Таким чином її можна застосовувати для отримання більш жорсткого і чутливого співставлення за косинусної подібністю. Метод tf-idf є дуже чутливим до форми переставлення, але і не досить добре розуміється на подібності змісту. Його краще застосовувати, коли важлива лише формальна подібність текстів.

В цілому, співставлення показників отриманих за різними методами і візуальний огляд результатів експерименту дозволяють підсумувати, що запропонований підхід генерації природномовних речень українською мовою на основі їх онтологічного представлення із застосуванням великої мовної моделі здатен передати загальний зміст і сенс оригінальної фрази. Проте часто, хоча і не у всіх випадках, згенерована фраза може виглядати дещо неприродно, мати надлишкові сутності і звороти. У ряді випадків можливі похибки пов’язані із поганим розумінням великою мовною моделлю термінології та професійного сленгу. Прикладом такої ситуаціє є наступний випадок: оригінальна фраза – «Принтер призначений для створення твердих копій документів.»; згенерована фраза – «Принтер, призначений для створення копій, які є документами та мають твердий стан.». Тобто, ми бачимо що модель не має переставлення про поняття «тверда копія» і намагається сконструювати словосполучення на основі наявної у ній парадигми звичайного сполучення слів.

Такий результати свідчить про те, що хоча великі мовні моделі і можуть бути застосовані для генерації текстів на основі онтологічного представлення і передавати загальний зміст, за формою (а іноді і за нюансами сенсу) такі фрази часто виглядають далекими від ідеалу. Це свідчить про актуальність систем генерації тексту на основі онтологічних представлень (в тому числі результатах запитів до онтології) побудованих на правилах і гучних шаблонах, як це було показано у розділі 3. Такі підходи за наявністю добре опрацьованої та докладної системи правил і шаблоні можуть генерувати значно якісніші природномовні фрази на базі семантичних представлень, ніж це роблять великі мовні моделі. При цьому великі мовні

моделі можуть бути успішно застосовані у задачах семантичного аналізу, що підвищує релевантність визначених намірів, а тобто і матеріалу, що отримується із бази знань для синтезу відповіді. Також великі мовні моделі бобре справляються із генерацією текстових відповідей ґрунтуючись на наборі контекстів, та переліку відповідних намірів.

5.4 Висновки за розділом 5

Для перевірки якості результатів розроблених діалогових систем використано наступні критерії: accuracy (A), precision (P), recall (R) і критерій F1, які є загальноприйнятими для такого роду систем.

Тестування системи, що не використовує великі мовні моделі, показало загалом непогані результати. Особливо слід звернути увагу на високий показник precision, що свідчить про малу схильність такої системи до хибно позитивних результатів. Тобто, діалогові системи подібного типу якщо чогось не знають, то чесно в цьому «признаються», не намагаючись видати у якості відповіді хоч щось. З іншого боку, знижений показник recall свідчить про те, що система все ж схильна до видачі хибно негативних результатів. Тобто ми знаємо, що відповідь у базі знань є, але система її не знаходить.

У разі залучення великих мовних моделей на нейронних мережах у комбінації із онтологічним підходом результат виявляється дещо інакшим. Це демонструється прикладом тестування системи OntoChatGTP. Тут ми навпаки бачимо велике значення показнику recall. Тобто хибно негативних відповідей така система практично не дає. Якщо інформація з даного питання є в базі знань і була передана великій мовній моделі у вигляді фрагментів контекстів, то користувач скоріш за все отримає відповідь. Але в свою чергу, така система демонструє знижений показник precision. Це означає, якщо відповіді насправді немає в базі, модель все одно намагається відповісти бодай що і видати хоч якусь так чи інакше пов’язану інформацію. Однак варто зазначити, що багато з цих хибно позитивних відповідей

супроводжувалися насправді і дійсно позитивними відповідями. Іншими словами, хоча в деяких випадках була надана неправильна відповідь, разом із нею була дана також і правильна. Ці супроводжувані хибно позитивні відповіді можна розглядати як додаткову інформацію, яка може бути дотичною до основної відповіді. Хоча наявність хибних спрацьовувань впливає на значення точності, той факт, що вони часто співіснують із дійсно позитивними, свідчить про те, що система здатна надавати додаткову статистику чи пов’язану інформацію. Це спостереження підкреслює потенційну цінність розгляду супроводжуваних хибно позитивних відповідей у практичному контексті.

Проведено експеримент із зворотної генерації природномовних речень на основі їх онтологічного представлення з використанням великої мовної моделі. В ході експерименту були отримані досить високі показники косинусної подібності між оригінальною та згенерованою фразою, проте детальний аналіз результатів показує, що підхід генерації на основі онтологічного представлення може передавати загальний зміст оригіналу але при деякому викривленні саме форми фрази. Так, генеровані фрази часто виглядають неприродно та містять надлишковість, особливо у випадках неправильного розуміння термінології та сленгу. Це підтверджує актуальність систем генерації тексту на основі онтологій з правилами та шаблонами, що можуть надавати якісніші результати у порівнянні з великими мовними моделями. Великі мовні моделі, в свою чергу, підходять для семантичного аналізу та генерації текстових відповідей на основі контексту.

ВИСНОВКИ

1. Розроблено новий метод підходу до створення системи семантичних відношень. Цей підхід дозволяє отримати значне розмаїття семантичних категорій шляхом комбінування існуючих понять верхнього рівня. Метод дозволяє створити обґрунтовану онтологію семантичних категорій будь-якої глибини та повноти, що відповідає предметній області і призначенню.

2. Розроблені методи семантичного аналізу природномовного тексту, включаючи мови флективного типу, зокрема українську. Це сприяло створенню формальних запитів за допомогою гнучких шаблонів. Особливість цього підходу полягає в автоматичному створенні формальних запитів з блоків шаблонів, включаючи основні і допоміжні, які налаштовуються відповідно до семантичних категорій, визначених у тексті, та сутностей, що їх конкретизують.

3. У роботі було розвинуто методи автоматичної генерації онтологій, як на основі текстів із визначеною регулярною структурою, так і на основі нерозмічених текстів з використанням глибокого синтактико-семантичного аналізу. Були розроблені практичні методи і програмні реалізації для побудови онтологій на основі наборів текстів з визначеною регулярною структурою. Запропоновані способи є ефективними як для мов флективного типу, зокрема, української, так і для англійської мови.

4. В рамках дослідження було розроблено комплексний підхід до створення природномовних діалогових систем, який використовує великі мовні моделі, мета-онтологію та онтологічну базу знань. Цей підхід базується на використанні структурованих інструкцій-підказок, створених з використанням мета-онтології, які передаються великій мовній моделі для аналізу вхідного природномовного тексту та отримання релевантних відповідей з контекстної інформації, витягнутої з онтологічної бази знань за допомогою формальних запитів.

5. Розроблені методи і системи демонструють практичну цінність і можуть бути використані в різних галузях, включаючи літературознавство, педагогіку, фінанси, екологію та інші. Вони стали основою для створення діалогових довідкових природномовних систем, включаючи ті, що стосуються фінансів і інвестицій (норвезька мова), проблем змінення клімату та екології (англійська мова), листування, економіки сталого розвитку (українська мова), «Білої книги з ФРМ» (українська мова) та реабілітаційної медицини (англійська мова).

6. Була створена комбінована система OntoChatGPT, яка використовує великі мовні моделі, включаючи API ChatGPT, та онтологічну базу знань для мета-навчання та покращення таких моделей. Особливість цієї системи полягає в наявності мета-онтології, яка створює структуровані інструкції-підказки для великої мовної моделі. Застосування цього підходу значно розширює можливості сильної мовної моделі, надаючи їй доступ до вибраних фрагментів спеціалізованої інформації з вузької предметної галузі, що підвищує змістовність і релевантність відповідей, наданих моделлю. Присутність структурованих інструкцій-підказок дозволяє з одного боку чітко визначити виражені наміри у вхідному тексті і відповідні поняття, які їх конкретизують, а з іншого боку, провести аналіз інформації, представленої в контекстній онтології, і синтезувати природномовну відповідь відповідно до заданої форми і структури.

7. Розроблені програмні діалогові системи відзначаються високою точністю в заданих предметних областях і ефективно доповнюють великі мовні моделі. Це підтверджується результатами тестування, в яких визначаються формальні критерії якості роботи, включаючи параметри Recall, Precision і F1.

8. Проведено експеримент із зворотної генерації природномовних речень на основі їх семантичного представлення у онтологічній базі знань, для чого було задіяно велику мовну модель (ChatGPT) та створені відповідні інструкції-підказки для неї. Хоча за результатами експерименту були

отримані досить високі показники косинусної подібності між оригінальними та згенерованими у вказаний спосіб реченнями та загальний зміст оригіналу в цілому зберігається, спостерігається помітне викривлення форми і стилю фрази. Так, генеровані фрази часто виглядають неприродно та містять надлишковість. Такий підхід підтверджує актуальність систем генерації тексту на основі онтологій з правилами та шаблонами, що можуть при вирішенні саме такої задачі надавати якісніші результати у порівнянні з великими мовними моделями.

ПЕРЕЛІК ПОСИЛЕНЬ

1. *Maulud H. M., Zeebaree S. R. M., Jacksi K., Sadeeq M. A. M., Hussein K.* A State of Art for Semantic Analysis of Natural Language Processing. Quabahan academic journal. 2021, P. 21 – 28.

2. *Abdulazeez A. M., Zeebaree S. R., Sadeeq M. A.* Design and Implementation of Electronic Student Affairs System. Academic Journal of Nawroz University. Vol. 7, 2018, P. 66 – 73.

3. *Zebari R. R., Zeebaree S. R., Jacksi K.* Impact Analysis of HTTP and SYN Flood DDoS Attacks on Apache 2 and IIS 10.0 Web Servers. 2018 International Conference on Advanced Science and Engineering (ICOASE). 2018, P. 156 – 161.

4. *Jelodar H., Wang Y., Rabbani M., Xiao G., Zhao R.* A collaborative framework based for semantic patients-behavior analysis and highlight topics discovery of alcoholic beverages in online healthcare forums. Journal of medical systems. Vol. 44, 2020, P. 1 – 8.

5. *Jacksi K., Dimililer N., Zeebaree S.* State of the art exploration systems for linked data: a review. Int. J. Adv. Comput. Sci. Appl. IJACSA. Vol. 7, 2016, P. 155 – 164.

6. *Wang X., Dong X., Chen S.* Text Duplicated-checking Algorithm Implementation Based on Natural Language Semantic Analysis. 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC). 2020, P. 732 – 735.

7. *Jacksi K., Zeebaree S. R., Dimililer N.* LOD Explorer: Presenting the Web of Data. Int. J. Adv. Comput. Sci. Appl. IJACSA. Vol. 9, 2018.

8. *Rayz J. T., Raskin V.* Logic of natural language: Through the eyes of ontological semantics. 2016 IEEE 15th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC). 2016, P. 511 – 515.

9. *Selamat S. A. A.-Z. A.* Electronic Learning Management System Based on Semantic Web Technology: A Review. Int. J. Adv. Electron. Comput. Sci. Vol. 4, 2017, P. 1 – 6.

10. *Okba K., Hamza S., Hind B., Amira A., Samir B.* Semantic natural language translation based on ontologies combination. 8th International Conference on Information Technology (ICIT). 2017, P. 315 – 321.

11. *Ibrahim R., Zeebaree S., Jacksi K.* Survey on Semantic Similarity Based on Document Clustering. Adv. sci. technol. eng. syst. j. Vol. 4, 2019, P. 115 – 122.

12. *Hood K., Kuiper P. K.* Kuiper Improving student surveys with natural language processing. Second IEEE International Conference on Robotic Computing (IRC). 2018, P. 383 – 386.

13. *Hadiya N., Nanavati N.* Indic SentiReview: Natural Language Processing based Sentiment Analysis on major Indian Language. 3rd International Conference on Computing Methodologies and Communication (ICCMC). 2019, P. 322 – 327.

14. *Jacksi K., Dimililer N., Zeebaree S. R.* A survey of exploratory search systems based on LOD resources. 2015.

15. *Jacksi K., Zeebaree S., Dimililer N.* Design and Implementation of LOD Explorer: A LOD Exploration and Visualization Model. Journal of Applied Science and Technology Trends. Vol. 1, 2020, P. 31 – 39.

16. *Young T., Hazarika D., Poria S., Cambria E.* Recent trends in deep learning based natural language processing. Computational intelligen Ce magazine. Vol. 13, 2018, P. 55 – 75.

17. *Liao X., Zhu Z.* Classification of Natural Language Semantic Relations under Deep Learning. IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA). 2020, P. 1025 – 1027.

18. *Maksutov A. A., Zamyatovskiy V. I., Vyunnikov V. N., Kutuzov A. V.* Knowledge Base Collecting Using Natural Language Processing Algorithms. IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). 2020, P. 405 – 407.

19. *Rokade A., Patil B., Rajani S., Revandkar S., Shedge R.* Automated grading system using natural language processing. Second International Conference on Inventive Communication and Computational Technologies (ICICCT). 2018, P. 1123 – 1127.

20. *Sadeeq M. J., Zeebaree S. R.* Semantic Search Engine Optimisation (SSEO) for Dynamic Websites: A Review. International Journal of Science and Business. Vol. 5, 2021, P. 148 – 158.

21. *Otter D. W., Medina J. R., Kalita J. K.* A survey of the usages of deep learning for natural language processing. IEEE Transactions on Neural Networks and Learning Systems, 2020.

22. *Al-Saadi H.* Demystifying Ontology and Epistemology in research methods. Research gate. Vol. 1, 2014, 1 – 10.

23. *Zeebaree A., Adel A., Jacksi K., Selamat A.* Designing an ontology of E-learning system for duhok polytechnic university using protégé OWL tool. J Adv Res Dyn Control Syst. Vol 11, P. 24 – 37.

24. *Ageed Z. S., Ibrahim R. K., Sadeeq M. A.* Unified Ontology Implementation of Cloud Computing for Distributed Systems. Current Journal of Applied Science and Technology. 2020, P. 82 – 97.

25. *AL-Zebari A., Zeebaree S., Jacksi K., Selamat A.* ELMS–DPU ontology visualization with Protégé VOWL and Web VOWL. Journal of Advanced Research in Dynamic and Control Systems. 2019, Vol. 11, P. 478 – 85.

26. *Kanakaraj M., Guddeti R. M. R.* Performance analysis of Ensemble methods on Twitter sentiment analysis using NLP techniques. Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015). 2015, P. 169 – 170.

27. *Ismail S. S., Aref M., Moawad I. F.* A model for generating Arabic text from semantic representation. 2015 11th International Computer Engineering Conference (ICENCO). 2015, P. 117 – 122.

28. *Hassan T., Hassan S., Yar M. A., Younas W.* Semantic analysis of natural language software requirement. 2016 Sixth International Conference on Innovative Computing Technology (INTECH). 2016, P. 459 – 463.

29. *Khan M. T., Durrani M., Ali A., Inayat I., Khalid S., Khan K. H.* Sentiment analysis and the complex natural language. Complex Adaptive Systems Modeling. Vol. 4, 2016, P. 1 – 19.

30. *Wang S.-Z., Zhang Q.-C., Zhang L.* Natural language semantic corpus construction based on cloud service platform. 2017 International Conference on Machine Learning and Cybernetics (ICMLC). 2017, P. 670 – 674.

31. *Gupta P., Goswami A., Koul S., Sartape K.* IQS-intelligent querying system using natural language processing. 2017 international conference of electronics, communication and aerospace technology (ICECA). 2017, P. 410 – 413.

32. *Sengloiluean K., Arch-int N., Arch-int S., Thongkrau T.* A semantic approach for question answering using DBpedia and WordNet. 2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE). 2017, P. 1-6.

33. *Sleimi A., Sannier N., Sabetzadeh M., Briand L., Dann J.* Automated extraction of semantic legal metadata using natural language processing. 2018 IEEE 26th International Requirements Engineering Conference (RE). 2018, P. 124 – 135.

34. *Şenel L. K., Utlu İ., Yücesoy V., Koç A., Çukur T.* Generating semantic similarity atlas for natural languages. 2018 IEEE Spoken Language Technology Workshop (SLT). 2018, P. 795 – 799.

35. *Zait F., Zarour N.* Addressing lexical and semantic ambiguity in natural language requirements. 2018 Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT). 2018, P. 1 – 7.

36. *Atzeni M., Atzori M.* Translating natural language to code: an unsupervised ontology-based approach. 2018 IEEE first international conference on artificial intelligence and knowledge engineering (AIKE). 2018, P. 1 – 8.

37. *Sarker J., Billah M., Al Mamun M.* Textual Question Answering for Semantic Parsing in Natural Language Processing. 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT). 2019, P. 1 – 5.

38. *Gunasekara L., Vidanage K.* UniOntBot: Semantic Natural Language Generation based API approach for Chatbot Communication. 2019 National Information Technology Conference (NITC). 2019, P. 1 – 8.

39. *Peelar S., Frost R.* A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore. 2020 IEEE 14th International Conference on Semantic Computing (ICSC). 2020, P. 257 – 262.

40. *Li W.* Analysis of Semantic Comprehension Algorithms of Natural Language Based on Robot's Questions and Answers. 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA). 2020, P. 1021 – 1024.

41. *Li M., Wang Y., Zhao Y., Li Z.* Transgender Community Sentiment Analysis from Social Media Data: A Natural Language Processing Approach. arXiv preprint arXiv:2010.13062, 2020.

42. *Sadhuram M. V., Soni A.* Natural Language Processing based New Approach to Design Factoid Question Answering System. 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA). 2020. P. 276 – 281.

43. OWL Web Ontology Language [Електронний ресурс] // The OWL Working Group. 2009. Режим доступу до ресурсу: https://www.w3.org/TR/owl-features/.

44. SPARQL Query Language for RDF [Електронний ресурс] // The SPARQL Working Group. 2008. Режим доступу до ресурсу: https://www.w3.org/TR/rdf-sparql-query/.

45. *Gruber T. R.* Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies. Vol. 43, No 5-6, 1995, P. 907 – 928.

46. *Studer R., Benjamins R., Fensel D.* Knowledge engineering: Principles and methods. Data & Knowledge Engineering. Vol. 25, No. 1-2, 1998, P. 161 – 197.

47. *Noy N. F., McGuinness D. L.* Ontology development 101: A guide to creating your first ontology. Stanford knowledge systems laboratory technical report KSL-01-05. 2001.

48. *Maedche A., Staab S.* Discovering conceptual relations from text. Data & Knowledge Engineering. Vol. 36, No 3, 2001, P. 265 – 292.

49. *Gómez-Pérez A., Fernández-López M., Corcho O.* Ontological engineering. Springer. 2004.

50. *Brachman R. J., Levesque H. J., Reiter R.* Knowledge representation and reasoning. Elsevier. 2004.

51. *Pinto H. S., Martins J. P.* Ontology learning: A survey of methods and tools. Knowledge Engineering Review. Vol. 20, No 2, 2004, P. 81 – 136.

52. *Bizer C., Seaborne A., Cyganiak R.* D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. 3rd International Semantic Web Conference. 2004.

53. *Reiter E., Dale R.* Building natural language generation systems. Cambridge University Press. 2000.

54. *Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I.* Language models are unsupervised multitask learners. OpenAI. 2019.

55. *Mellish C., Knott A., Oberlander J., O'Donnell M.* Generating natural language under pragmatic constraints. Journal of Artificial Intelligence Research. Vol 9, 1998, P. 103 – 133.

56. *Brown T. B., Mann B., Ryder N., Subbiah M., Kaplan J., Dhariwal P. Agarwal S.* Language models are few-shot learners. arXiv preprint arXiv:2005.14165. 2020.

57. *Wiseman S., Shieber S. M., Rush A. M.* Challenges in data-to-document generation. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Vol. 1, 2017, P. 590 – 600.

58. *See A., Liu P. J., Manning C. D.* Get to the point: Summarization with pointer-generator networks. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL). Vol. 1, 2017, P. 1073 – 1083.

59. *Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Polosukhin I.* Attention is all you need. Advances in Neural Information Processing Systems (NeurIPS). 2017, P. 30 – 38.

60. *Jurafsky D., Martin J. H.* Speech and Language Processing (3rd ed.). Pearson. 2020.

61. *Mitrovic A.* Fifteen years of constraint-based tutors: what we have achieved and where we are going. User Modeling and User-Adapted Interaction, Vol. 22, No. 1 – 2, 2012, P. 39 – 72.

62. *Bender E. M., Gebru T., McMillan-Major A., Shmitchell M.* On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? arXiv preprint arXiv:2101.02268. 2021.

63. Models - OpenAI API [Електронний ресурс] // OpenAI Platform, 2023, Режим доступу до ресурсу: https://platform.openai.com/docs/models/overview. (accessed Jun. 01, 2023).

64. Introducing ChatGPT [Електронний ресурс] // OpenAI, 2022, Режим доступу до ресурсу: https://openai.com/blog/chatgpt. (accessed Jun. 01, 2023).

65. GPT-4 Developer Livestream [Електронний ресурс] // YouTube, 2023, Режим доступу до ресурсу: https://www.youtube.com/watch?v=outcGtbnMuQ. (accessed Jun. 01, 2023).

66. *Palagin O. V., Petrenko M. G.* The explanatory ontograph dictionary for knowledge engineering [Електронний ресурс]. Iterservice, 2017, Режим доступу до ресурсу: http://www.aduis.com.ua/books/tlumachny-ontohrafichny-slovnyk-z-inzhenerii-znan.pdf.

67. *Palagin O. V., Malakhov K. S., Velychko V. Yu., Semykopna T. V.* Hybrid e-rehabilitation services: SMART-system for remote support of rehabilitation activities and services. Int J Telerehab, no. Special Issue: Research Status Report. Ukraine, 2022. doi: 10.5195/ijt.2022.6480.

68. OpenAI API Reference [Електронний ресурс] // OpenAI Platform. 2023. Режим доступу до ресурсу: https://platform.openai.com/docs/api-reference. (accessed Jun. 01, 2023).

69. JushBJJ/Mr.-Ranedeer-AI-Tutor: A GPT-4 AI Tutor Prompt for customizable personalized learning experiences [Електронний ресурс] // GitHub. 2023. Режим доступу до ресурсу: https://github.com/JushBJJ/Mr.-Ranedeer-AI-Tutor. (accessed Jun. 01, 2023).

70. Structured JSON Prompts are even better in GPT-4., [Електронний ресурс] // {Structured} Prompt. Режим доступу до ресурсу: https://structuredprompt.com/structured-json-prompts-are-even-better-in-chatgpt-4/ (accessed Jun. 01, 2023).

71. GPT 4 is Smarter than You Think: Introducing SmartGPT [Електронний ресурс] // YouTube. 2023. Режим доступу до ресурсу: https://www.youtube.com/watch?v=wVzuvf9D9BU.

72. *Jush B.* Mr. Ranedeer [Електронний ресурс]. JushBJJ’s Substack. 2023. Режим доступу до ресурсу: https://jushbjj.substack.com/p/mr-ranedeer. (accessed Jun. 01, 2023).

73. *Hebenstreit K., Praas R., Kiesewetter L. P., Samwald M.* An automatically discovered chain-of-thought prompt generalizes to novel models and datasets. arXiv, May 04, 2023. doi: 10.48550/arXiv.2305.02897.

74. *Kojima T., Gu S. S., Reid M., Matsuo Y., Iwasawa Y.* Large Language Models are Zero-Shot Reasoners. arXiv, Jan. 29, 2023. doi: 10.48550/arXiv.2205.11916.

75. *Wei J.* Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv, Jan. 10, 2023. doi: 10.48550/arXiv.2201.11903.

76. ChatGPT plugins [Електронний ресурс]. OpenAI blog. 2023. Режим доступу до ресурсу: https://openai.com/blog/chatgpt-plugins. (accessed Jun. 01, 2023).

77. OpenAI Plugins API [Електронний ресурс]. OpenAI Platform. 2023. Режим доступу до ресурсу: https://platform.openai.com/docs/plugins/introduction. (accessed Jun. 01, 2023).

78. *Moghaddam S. R., Honey C. J.* Boosting Theory-of-Mind Performance in Large Language Models via Prompting. arXiv, Apr. 26, 2023. doi: 10.48550/arXiv.2304.11490.

79. *Hendrycks D.* Measuring Massive Multitask Language Understanding. arXiv, Jan. 12, 2021. doi: 10.48550/arXiv.2009.03300.

80. *Zhou Y.* Large Language Models Are Human-Level Prompt Engineers. arXiv, Mar. 10, 2023. doi: 10.48550/arXiv.2211.01910.

81. *Quamar A., Özcan F., Miller D., Moore R. J., Niehus R., Kreulen J.* Conversational BI: an ontology-driven conversation system for business intelligence applications. Proc. VLDB Endow. Vol. 13, no. 12, 2020, P. 3369 – 3381. doi: 10.14778/3415478.3415557.

82. *Palagin A. V.* Architecture of ontology-controlled computer systems. Cybern Syst Anal. Vol. 42, No. 2, 2006, P. 254–264. doi: 10.1007/s10559-006-0061-z.

83. *Palagin O. V., Petrenko M. G., Velychko V. Yu., Malakhov K. S.* Development of formal models, algorithms, procedures, engineering and functioning of the software system 'Instrumental complex for ontological engineering purpose. CEUR Workshop Proceedings, Kyiv, Ukraine: CEUR-WS, May 2014, P. 221 – 232. Available: http://ceur-ws.org/Vol-1843/221-232.pdf

84. *Litvin A., Velychko V., Kaverinsky V.* A new approach to automatic ontology creation from the untagged text on the natural language of inflective type. Proceedings of the International conference on software engineering "Soft Engine 2022", NAU, Kyiv Ukraine, 2022, P. 37 – 45.

85. *Litvin A., Velychko V., Kaverinsky V.* Development of natural language dialogue software systems, Information Theories and Applications 28. 2021, P. 233 – 270. doi: 10.54521/ijita28-03-p03

86. *S. J. Young, Gasic M., Thomson B., Williams J. D.* Pomdp-based statistical spoken dialog systems. A review. Proceedings of the IEEE. Vol. 101, No. 5, 2013, P. 1160 – 1179.

87. *Quamar A., Lei C., Miller D., Ozcan F., Kreulen J., Moore R.J., Efthymiou V.* An ontology-based conversation system for knowledge bases. Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020. P. 361 – 376. doi: 10.1145/3318464.3386139

88. *Ochieng P.* PAROT: Translating natural language to SPARQL. Expert Systems with Applications. Vol. 5, 2020, P. 1 – 16. doi: 10.1016/j.eswax.2020.100024

89. *Damljanovic D., Agatonovic M., Cunningham H.* FREyA: an interactive way of querying linked data using natural language. The Semantic Web: ESWC 2011 Workshops. 2011, P. 125– 138.

90. FREyA documentation [Електронний ресурс] // GitHub – Режим доступу до ресурсу: https://github.com/nmvijay/freya

91. *Shaik S., Kanakam P., Hussain S.M., Suryanarayana D.* Transforming natural language query to SPARQL for semantic information retrieval. International Journal of Engineering Trends and Technology. Vol. 7, 2016, P. 347 – 350. doi: 10.14445/22315381/IJETT-V41P263

92. *Duygu A.* Ontology-Based Dialogue Management System for Banking and Finance Dialogue Systems, in: 1st Financial Narrative Processing Workshop. Japan, Miyazaki. 2018. doi: 10.48550/arXiv.1804.04838.

93. *Jung H., Kim W.* Automated conversion from natural language query to SPARQL query, Journal of Intelligent Information Systems. Vol. 55, 2020, P. 501-520.

94. *Goel A.* Neo4J Cookbook. Birmingham: Pact Publishing Ltd. 2015.

95. *Sun C.* A Natural Language Interface for Querying Graph Databases, Master's thesis, USA: Massachusetts Institute of Technology, Cambridge, USA, 2018.

96. Convert English sentences to Cypher queries documentation [Електронний ресурс] // GitHub – Режим доступу до ресурсу: https://github.com/gsssrao/english2cypher

97. *Kaverynskyi V. V., Litvin A. A., Velychko V. Yu.* Tree-based semantic analysis method for natural language phrase to formal query conversion. Radio Electronics, Computer Science, Control. No. 2, 2021, P. 105 – 113. doi: 10.15588/1607-3274-2021-2-11

98. *Bossche M. V., Ross P., MacLarty I., Nuffelen B. V., Pelov N.* Ontology driven software engineering for real life applications. Proceedings of the 3rd Intl. Workshop on Semantic Web Enabled Software Engineering. 2007. Available at: https://www.semanticscholar.org/paper/Ontology-Driven-SoftwareEngineering-for-Real-Life-BosscheRoss/aabbe8ecd227bd931b44da8cea2aa8d2d1f76519

99. WebProtégé [Електронний ресурс] // Biomedical Informatics Research Group – Режим доступу до ресурсу: https://webprotege.stanford.edu/.

100. *Curé O., Blin G.* RDF database systems: triples storage and SPARQL query processing, First edition. Amsterdam; Boston: Morgan Kaufmann, 2015.

101. LanguageTool API NLP UK [Електронний ресурс] // Corpus of modern Ukrainian language. 2023. Режим доступу до ресурсу: https://github.com/brownuk/nlp_uk.

102. *Panda S., Kaur N.* Revolutionizing language processing in libraries with SheetGPT: an integration of Google Sheet and ChatGPT plugin. Library Hi Tech News. 2023, https://doi.org/10.1108/LHTN-03-2023- 0051

103. *Korobov M.* Morphological Analyzer and Generator for Russian and Ukrainian Languages. Analysis of Images, Social Networks and Texts. 2015, P. 320–332.

104. *Kaverynskyi V., Litvin A., Velychko V.* Method of information obtaining from ontology on the basis of a natural language phrase analysis. Workshop Proceedings, CEUR-WS, Kyiv, Ukraine. 2020, P. 323 – 330. https://doi.org/10.15407/pp2020.02-03.322

105. *Nvana H.* Software. Agents: An Overview. Knowledge Engineering Review. Cambridge University Press. Vol. 21, Issue 3, 1996, P. 205 – 244.

106. *Poslad S.* Specifying Protocols for Multi-agent System Interaction. ACM Transactions on Autonomous and Adaptive Systems. Vol. 4, Issue 4, 2007. P. 15 – 39.

107. *Galitsky B.* Developing Enterprise Chatbots. Learning Linguistic Structures. Springer, San Jose, 2019

108. *Палагин А. В., Крывый С. Л., Петренко Н. Г.* Онтологические методы и средства обработки предметных знаний. – Луганск: изд-во ВНУ им. В. Даля, 2012, 324 с.

109. *Palagin O., Malakhov K., Kaverinsky V. V., Litvin A. A.* OntoChatGPT Information System: Ontology-Driven Structured Prompts for ChatGPT Meta-Learnin. International Journal of Computing, V. 22, No. 2, 2023, P. 170 – 183.

110. *Palagin O., Malakhov K., KaverinskyV. V., Litvin A. A.* Ontology-driven development of dialogue systems. South African Computer Journal, V. 35, No. 1, 2023, P. 37 – 62.

111. *Velychko V. Yu., Kaverinsky V. V., Litvin A. A.* A New Approach to Automatic Ontology Generation from the Natural Language Texts with Complex Inflection Structures in the Dialogue Systems Development. CEUR Workshop Proceedings, 2023, V. 3501, P. 172–185.

ДОДАТОК А

Структура JSON шаблону представлення змісту файлу статті із набору EBSCO

```
{"metadata": {
  "title": "",
"synonyms": [],
    "anatomical location body part affected": [],
    "area of specialty": [],
    "description": [],
    "indications": [],
    "ICD-9 codes": {},
    "ICD-10 codes": {},
    "ICD-11": {},
    "G-codes": {},
    "CPT codes": {},
    "HCPCS codes": {},
    "reimbursement": [],
    "presentation signs and symptoms": [],
    "contraindications precautions to test": [],
    "psychometric properties": [],
    "potential complications adverse effects": [],
    "test preparation materials required": [],
    "test procedure": [],
    "test scoring interpretation": []
  },
  "causes & risk factors": {},
  "indications for procedure": [],
  "overall contraindications precautions": [],
  "contraindications precautions to procedure": {},
  "guidelines for use of procedure": {},
  "examination": {
    "contraindications precautions to examination": [],
    "history": {
      "history of present illness": {
        "mechanism of injury or etiology of illness": "",
        "course of treatment": {
          "medical management": "",
          "surgical management": [],
          "medications": "",
          "diagnostic tests completed": [],
          "alternative therapies": "",
          "previous therapy": ""
        },
        "aggravating easing factors": "",
        "body chart": "",
        "nature of symptoms": "",
        "rating of symptoms": "",
        "pattern of symptoms": "",
        "sleep disturbance": "",
```

```
        "other symptoms": "",
        "respiratory status": ""
      },
      "medical history": {
        "past medical history": {
          "previous history of same similar diagnosis": "",
          "comorbid diagnoses": "",
          "medications previously prescribed": "",
          "other symptoms": ""
        }
      },
      "social and occupational history": {
        "patient's goals": "",
        "vocation avocation and associated repetitive behaviors,
if any": "",
        "functional limitations assistance with ADLs adaptive
equipment": "",
        "living environment": ""
      }
    },
"relevant tests and measures": {
      "general": "",
            "anthropometric characteristics": "",
            "assistive and adaptive devices": "",
            "balance": "",
            "cardiorespiratory function and endurance": [],
            "circulation": "",
            "functional mobility": "",
            "gait locomotion": "",
            "muscle strength": "",
            "observation inspection palpation including skin
assessment": [],
            "posture": "",
            "range of motion": "",
            "self-care activities of daily living": "",
      "reflex testing": "",
      "sensory testing": "",
      "special tests specific to diagnosis": [],
      "speech and language examination": "",
      "neurological examination": "",
      "oral structure and oral motor function": "",
      "perception": "",
      "sensory testing": "",
      "special tests specific to diagnosis": "",
      "arousal attention cognition": "",
      "aac assessment": "",
      "swallow examination": "",
      "tracheostomy examination": ""
    }
  },
  "assessment plan of care": {
    "contraindications precautions": {
      "cryotherapy contraindications": [],
```

```
      "cryotherapy precautions": [],
      "contraindications precautions": [],
      "superficial heat is contraindicated with": [],
      "electrotherapy  contraindications  precautions  include":
[],
      "other": []
    },
    "diagnosis need for treatment": "",
    "rule out": "",
    "prognosis": [],
    "referral to other disciplines": "",
    "other considerations": [],
    "treatment summary": []
  },
  "desired outcomes outcome measures": {},
  "maintenance or prevention": [],
  "patient education": [],
  "references": []
}
```

ДОДАТОК Б

Шаблон запитів до онтології для отримання простих контекстуальних відповідей

```
<template>
     <language>ukrainian</language>
     <type>ready_answer</type>
     <marker_words />
     <variables>
          <variable>
               <name>inputEntity</name>
               <var_type>str</var_type>
               <destination>input</destination>
               <allow_list>true</allow_list>
          </variable>
          <variable>
               <name>result</name>
               <var_type>str</var_type>
               <destination>result</destination>
               <allow_list>true</allow_list>
          </variable>
          <variable>
               <name>intersectionElementID</name>
               <var_type>auto</var_type>
               <destination>inner</destination>
               <allow_list>false</allow_list>
          </variable>
          <variable>
               <name>intersectionName</name>
               <var_type>auto</var_type>
               <destination>inner</destination>
               <allow_list>false</allow_list>
          </variable>
          <variable>
               <name>propName</name>
```

```
                    <var_type>auto</var_type>
                    <destination>inner</destination>
                    <allow_list>false</allow_list>
               </variable>
          </variables>
          <query_base>SELECT DISTINCT [result] </query_base>
          <conditions>
               <condition>[intersectionElementID]
rdf:rest*/rdf:first [inputEntity].</condition>
               <condition>[intersectionName]   owl:intersectionOf
[intersectionElementID].</condition>
               <condition>[propName]                  rdfs:domain
[intersectionName].</condition>
               <condition>[propName]                   rdfs:range
[result].</condition>
               <condition>?result                 rdfs:subClassOf
:ReadyAnswerLink.</condition>
          </conditions>
     </template>
```

ДОДАТОК В

Приклади XML-шаблонів запитів до бази знань на мові Cypher

1. Приклад основного шаблону запиту на мові Cypher для запиту загальної інформації про об’єкт

```
<template>
     <verbose_name>Загальна інформація</verbose_name>
     <id>1</id>
     <type>base</type>
     <variables>
          <variable>
               <name>INPUT_VALUE_1</name>
               <destination>input</destination>
          </variable>
          <variable>
               <name>CONTEXT</name>
               <destination>output</destination>
          </variable>
     </variables>
     <match>
          (inp:Class)-[]-(n:Relationship),
          (n:Relationship)-[]-(x:Class),
          (n)-[:SPO]->(rel_group),
          (rel_group)-[:SPO]->(rel_sent),
          (rel_sent)-[:SPO]-(sent_super)
     </match>
     <where>
          inp.label = "INPUT_VALUE" and
          sent_super.name = "SentenceGroups"
     </where>
     <return>
          DISTINCT rel_sent.label as CONTEXT;
     </return>
</template>
```

2. Приклад додаткового шаблону запиту на мові Cypher

```
<template>
     <verbose_name>Пов'язаний із підметом
прикметник</verbose_name>
     <id>1</id>
     <type>additional</type>
     <variables>
          <variable>
               <name>INPUT_VALUE_ADJ</name>
               <destination>input</destination>
          </variable>
          <variable>
               <name>ADJ_PLUS</name>
               <destination>intermediate</destination>
          </variable>
          <variable>
               <name>INP_ADJ</name>
               <destination>intermediate</destination>
          </variable>
     </variables>
     <block_union>and</block_union>
     <next_item_union>or</next_item_union>
     <match>
          (inp:Class)-[]-(ADJ_PLUS:Relationship),
          (ADJ_PLUS:Relationship)-[]-(INP_ADJ:Class),
          (ADJ_PLUS)-[:SPO]->(rel_group)
     </match>
     <where>
          INP_ADJ.label = "INPUT_VALUE_ADJ"
     </where>
     <return></return>
</template>
```

ДОДАТОК Г

Приклади XML-шаблону для формування відповіді на основі результатів запиту до бази знань

1. Шаблон для синтезу відповіді українською мовою для випадку запиту набору сутностей

```
<template>
      <input_expression>
            <type>question</type>
            <subtype>objects</subtype>
            <marker_words>
                  <word>який</word>
                  <word>яка</word>
                  <word>яке</word>
                  <word>які</word>
            </marker_words>
      </input_expression>
      <output_structure>
            <order>
                  <part id=1 type="place_circumstance" case="loct"
preposition="as_input" number="as_input" />
                  <part id=2 type="instrumental_circumstance" case="inst"
preposition="as_input" number="as_input" />
                  <part id=3 type="accusative_circumstance" case="accus"
preposition="as_input" number="as_input" />
            <part id=6 type="dative_circumstance" case="dative"
preposition="as_input" number="as_input" />
                  <part id=4 type="match_predicate" />
                  <part id=5 type="verb_circumstance" number="as_input"
tense="as_input" gender="as_input" />
                  <part id=7 type="introductory" >
                        <options random="true">
                              <option id=1>
                                    <single>наступний</single>
                                    <plural>наступні</plural>
                              </option>
                              <option id=2>
                                    <single>такий</single>
                                    <plural>такі</plural>
                              </option>
```

```
                        </options>
                  </part>
                  <part id=8 type="main_instance" number="conform"
case="accus" />
                  <part id=9 type="genitive_circumstance" case="gent"
preposition="as_input" number="as_input" />
                  <part id=10 type="query_results" class="object" case="nomn"
limit=5 />
            </order>
            <exceptions>
                  <no_results action="replace">
                        <part id=1 type="introductory" >
                              <options random="true">
                                    <option id=1>Нажаль, мені
невідомо,</option>
                                    <option id=2>Вибачте, але я не
знаю,</option>
                                    <option id=3>У мене на даний момент немає
інформації, про те,</option>
                              </options>
                        </part>
                        <part id=2 type="marker_word" >
                        <part id=3 type="main_instance" number="conform"
case="accus" />
                        <part id=4 type="match_predicate" />
                       <part id=5 type="verb_circumstance" number="as_input"
tense="as_input" gender="as_input" />
                        <part id=6 type="introductory" >
                              <options random="true">
                                    <option id=1>у вказаних вами
обставинах.</option>
                                    <option id=2>при таких умовах.</option>
                                    <option id=3>, у всякому випадку, в даному
контексті.</option>
                              </options>
                        </part>
                        <part id=7 type="introductory" >
                              <options random="true">
                                    <option id=1>Тим не менш, я можу
розповісти про наступне:</option>
                                    <option id=2>Одначе, у мене є інформація
про такі речі:</option>
```

```
                                    <option id=3>Можливо, вам буде цікаво
поговорити на наступні теми:</option>
                              </options>
                        </part>
                        <part id=8 type="request_possible" case="nomn"
limit=5>
                  </no_results>
                  <overlimit action="insert">
                        <part id=1 type="introductory" insert="8" >
                              <options random="false">
                                    <option id=1>, например,</option>
                              </options>
                        </part>
                        <part id=2 type="introductory" insert="end" >
                              <options random="true">
                                    <option id=1>Це дуже широке питання.
Можливо, Вас цікавить щось більш конкретне?</option>
                                    <option id=2>По даному питанню є чимало
інформації. Що сама Вас цікавить?</option>
                              </options>
                        </part>
                  </overlimit>
            </exceptions>
      </output_structure>
</template>
```

2. Приклад шаблону синтезу відповіді для випадку запиту часу події (для англійської мови)

```
<?xml version="1.0" encoding="utf-8"?>
<template>
      <input_expression>
            <type>question</type>
            <subtype>time</subtype>
            <marker_words>
                  <word>when</word>
                  <word>in what time</word>
            </marker_words>
      </input_expression>
      <output_structure>
            <order>
```

```
                  <part id=1 type="main_instance" number="conform"
article="definite" />
                  <part id=2 type="match_predicate" />
                  <part id=3 type="verb_circumstance" number="as_input"
tense="as_input" />
                  <part id=4 type="accusative_circumstance"
preposition="as_input" number="as_input" article="as_input" />
                  <part id=5 type="instrumental_circumstance"
preposition="by,with" number="as_input" article="definite" />
                  <part id=6 type="dative_circumstance" preposition="
as_input" number="as_input" article="definite" />
                  <part id=7 type="place_circumstance" preposition="as_input"
number="as_input" article="as_input" />

                  <part id=8 type="introductory" >
                        <options random="false">
                              <option id=1>
                                    <single>at</single>
                                    <plural>at:</plural>
                              </option>
                              <option id=2>
                                    <single>in</single>
                                    <plural>in:</plural>
                              </option>
                              <option id=3>in the</option>
                              <option id=4>at the</option>
                        </options>
                  </part>
                  <part id=10 type="query_results" class="time" limit=10 />
            </order>
            <exceptions>
                  <no_results action="replace">
                        <part id=1 type="introductory" >
                              <options random="true">
                                    <option id=1>Unfortunately, we don't
know</option>
                                    <option id=2>Sorry, but I do not
know</option>
                                    <option id=3>I don't have any
information</option>
                              </options>
                        </part>
                        <part id=2 type="marker_word" >
```

```
                        <part id=3 type="main_instance" number="conform"
article="definite" />
                        <part id=4 type="match_predicate" />
                       <part id=5 type="verb_circumstance" number="as_input"
tense="as_input" />
                        <part id=6 type="introductory" >
                              <options random="true">
                                    <option id=1>at least in this
case.</option>
                                    <option id=2>with all these
conditions.</option>
                                    <option id=3>, at least, in the
context.</option>
                              </options>
                        </part>
                        <part id=7 type="introductory" >
                              <options random="true">
                                    <option id=1>But I can say when:</option>
                                    <option id=2>However, I know
when:</option>
                                    <option id=3>May be you are also
interested when:</option>
                              </options>
                        </part>
                        <part id=8 type="request_possible" limit=5>

                  </no_results>
                  <overlimit action="insert">
                        <part id=1 type="introductory" insert="1" >
                              <options random="true">
                                    <option id=1>For example,</option>
                                    <option id=2>For instance,</option>
                                    <option id=3>Some of</option>
                              </options>
                        </part>
                        <part id=2 type="introductory" insert="end" >
                              <options random="true">
                                    <option id=1>There are too many times this
fact ocures. May be you need something more certain?</option>
                                    <option id=2>The event you asked happen
many times. What do you want to know exactly? </option>
                              </options>
                        </part>
```

```
                    </overlimit>
                </exceptions>
            </output_structure>
</template>
```

ДОДАТОК Д

Приклади формування відповідей українською мовою
за шаблонами для різних семантичних типів

Приклад 1

Фраза користувача: «Які мови програмування можна вивчити на курсах в IT-Академії?». Ця фраза відповідає описаному вище шаблону: це запит до списку об’єктів, слово-маркер «які». Припустимо, що в результаті запиту до онтології вийшов такий список понять: «Java», «PHP», «Python», «JavaScript», «C#», «C++».

Опишемо процес створення відповіді.

Результати існують, і їх кількість не перевищує встановленого ліміту, тому розділ `<exceptions>` не розглядається.

Перша частина є додатковою обставиною місця з вихідної фрази. В даному випадку їх два: «на курсах» і «в IT-Академії». Відповідно до вказівки шаблону ставимо їх у місцевому відмінку в тому ж числі і з тим же прийменником і в тій же послідовності, що і в вихідному словосполученні. Тому початок фрази буде таким: «На курсах в IT-Академії…».

Наступна частина — додаткова обставина з вихідного словосполучення зі значенням іменника в орудному відмінку. В оригінальній фразі такого немає.

Наступна частина — додаткова обставина з вихідного словосполучення зі значенням іменника в знахідному відмінку. Такого також немає в оригінальній фразі.

Наступна частина — додаткова обставина з вихідної фрази зі значенням іменника в давальному відмінку. В оригінальній фразі такого немає.

Переходимо до наступного пункту. Далі підставляється вираз предикату відповідності. У цій фразі таким є слово «можна». Підставимо його. Фраза набуває вигляду: «На курсах в IT-Академії можна…».

Потім відбувається підстановка додаткової обставини з вихідного словосполучення, вираженого дієсловом. У цьому випадку є дієслово «вивчити». Після підстановки отримуємо: «На курсах в IT-Академії можна вивчити…».

Далі слідує вступна фраза. Вона має два варіанти. Припустимо, було обрано перший з них. Результати запиту містять більше одного поняття, тому ми підставляємо цей параметр у множині. Це слово «наступні». Таким чином фраза стає: «На курсах в IT-Академії можна вивчити наступні…»

Наступна частина — це основна поняття з оригінальної фрази, на яке вказує слово-маркер. В даному випадку це іменна група «мови програмування». Підставимо його в утворювану фразу. Вона набуде вигляду: «На курсах в IT-Академії можна вивчити наступні мови програмування:…»

Наступна частина є додатковою обставиною з вихідного речення зі значенням іменника в родовому відмінку. В оригінальній фразі такого немає, тому пропускаємо її.

Далі ми перераховуємо в називному відмінку поняття з результатів запиту. В результаті отримуємо: «На курсах в IT-Академії можна вивчити наступні мови програмування: Java, PHP, Python, JavaScript, C#, C++». Ця фраза є граматично правильною відповіддю на поставлене питання.

Приклад 2

Та сама фраза, але припустимо, що запит до онтології не дав результатів. У цьому випадку відповідь формується відповідно до розділу `<no_results>` з розділу `<exceptions>`.

Перша частина — це вибір випадкової фрази зі списку. Уявимо, що вибрано другий варіант: «Вибачте, але я не знаю». Далі вставляється слово-маркер з вихідної фрази. Виходить: «Вибачте, але я не знаю, які…». Потім підставляємо основне поняття з оригінальної фрази, на яке вказує слово-маркер, в даному випадку це «мови програмування». В результаті маємо: «Вибачте, але я не знаю, які мови програмування …». Потім підставляємо вираз предиката відповідності. У цій фразі таким є слово «можна». Тож

формована фраза тепер виглядає так: «Вибачте, але я не знаю, які мови програмування можуть…». Потім відбувається підставлення додаткової обставини з вихідного словосполучення, вираженого дієсловом. У цьому випадку є дієслово «вивчати». Після підстановки отримуємо: «Вибачте, але я не знаю, які мови програмування можна вивчати…». Потім вставляється одна зі стандартних фраз зі списку. Уявимо, що це перша фраза. Отримуємо: «Вибачте, але я не знаю, які мови програмування можна вивчати у вказаних Вами обставинах…». Далі підставляємо фразу з наступного списку, наприклад, третю. Після цього відповідь стає такою: «Вибачте, але я не знаю, які мови програмування можна вивчати у вказаних Вами обставинах. Вам може бути цікаво поговорити на такі теми: …»).

Далі підставляються 5 випадкових понять з онтології, наприклад: «математичне моделювання», «нейронні мережі», «метод зворотного розповсюдження помилки», «дерева прийняття рішень», «багатопараметрична оптимізація». У результаті відповідь може набути наступного вигляду: «Вибачте, але я не знаю, які мови програмування можна вивчати за вказаних вами обставин. Вам може бути цікаво обговорити наступні теми: математичне моделювання, нейронні мережі, метод зворотного розповсюдження помилки, дерева прийняття рішень, багатопараметрична оптимізація.».

Приклад 3

Фраза користувача: «Які модулі є в стандартній бібліотеці мови Python?». Ця фраза відповідає описаному вище шаблону: це запит для списку об’єктів; слово-маркер «які». Припустимо, що результат запиту онтології повернув досить довгий список. Опишемо процес побудови відповіді в цьому випадку.

Результати отримані, але їх кількість перевищує встановлений ліміт. Отже, повинен бути задіяний розділ `<exceptions>`; незважаючи на це, основа фрази буде формуватися звичайним способом.

Перша частина — додаткова обставина місця з вихідної фрази. В даному випадку це іменник «в стандартній бібліотеці мови Python». Стоїть у місцевому відмінку. Відповідно до вказівки шаблону ставимо його в місцевий відмінок в тому ж числі і з тим же прийменником, що і в вихідному словосполученні. Таким чином, фраза набуде вигляду: «В стандартній бібліотеці мови Python…».

Далі, за шаблоном, слідує додаткова обставина з вихідного речення зі значенням іменника в орудному відмінку. В оригінальній фразі такого немає.

Наступна частина є додатковою обставиною з вихідного речення зі значенням іменника в знахідному відмінку. Такого також немає в оригінальній фразі.

Наступна частина — це додаткова обставина з вихідного речення зі значенням іменника в давальному відмінку. В оригінальній фразі такого немає. Тож переходимо до наступного пункту.

Далі підставляється вираз предиката відповідності. У вихідному реченні, що розглядається присутнє слово «є». Підставте його. Фраза стає: «В стандартній бібліотеці мови Python є…».

Потім відбувається підстановка додаткової обставини з вихідного речення, вираженого дієсловом. У цьому випадку воно збігається з предикатом відповідності, тому ми його опускаємо.

Далі йде вступ. Він має два варіанти. Припустимо, обрано другий. Результати запиту мають більше одного поняття. Тож підставляємо варіант для множини. Це слово «такі». Фраза набуде вигляду: «В стандартній бібліотеці мови Python є такі…».

Наступна частина — це основне поняття оригінальної фрази, на яку вказує слово-маркер. В даному випадку це слово «модулі». Підставимо його у формовану фразу і поставимо після нього двокрапку: «В стандартній бібліотеці мови Python є такі модулі:…».

Наступна частина — додаткова обставина з вихідного речення зі значенням іменника в родовому відмінку. В оригінальній фразі такого немає. Перейти до наступного пункту.

Далі перераховуються поняття з результатів запиту в називному відмінку, обмежуючись випадковими п’ятьма поняттями. Наприклад: «модуль sys», «модуль contextlid», «модуль datetime», «модуль os», «модуль threading». Однак цю фразу згідно з шаблоном потрібно доповнити згідно розділу `<overlimit>`, тому що в даному випадку буде показано лише кілька результатів. Отже, спочатку в сформовану фразу потрібно підставити слово «наприклад». Позиція вставки – перед частиною 8 (перед словом «такі»). В результаті отримуємо: «В стандартній бібліотеці мови Python є, наприклад, такі модулі: модуль sys, модуль contextlid, модуль datetime, модуль os, модуль threading».

У кінці сформованої відповіді додається одна з випадкових фраз зі списку. Наприклад, буде обрано другий варіант: «По даному питанню є дуже багато інформації. Що саме Вам цікаво?». Таким чином, повна відповідь може виглядати так: «В стандартній бібліотеці мови Python є, наприклад, такі модулі: модуль sys, модуль contextlid, модуль datetime, модуль os, modul threading. По цьому питанню є дуже багато інформації. Що саме Вам цікаво?».

Приклад 4

Фраза користувача: «Які програмні бібліотеки в Python допоможуть мені використовувати алгоритми машинного навчання для розпізнавання зображень методом головних компонентів і лінійного дискримінантного аналізу?». Ця фраза спеціально створена, щоб містити всі частини генерованої фрази, присутні в шаблоні. Ці частини перераховані нижче в порядку, в якому вони зустрічаються у шаблоні:

place_circumstance – «на мові Python»

accusative_circumstance – «алгоритми машинного навчання»

dative_circumstance – «мені» (змінюється на «Вам»)

match_predicate – «допоможуть»

verb_circumstance – «використовувати»

main_instance – «програмні бібліотеки»

genitive_circumstance – «для розпізнавання зображень»

instrumental_circumstance – «метод головних компонентів і лінійний дискримінантний аналіз»

Ці фрагменти є матеріалом для підготовки фрази-відповіді. У першій частині згідно за правилами заміни «мені» змінюється на «вам». Таким чином, початок фрази для відповіді буде виглядати наступним чином (частини умовно розділені знаком «|»): «На мові Python | алгоритми машинного навчання | Вам | допоможуть | використовувати | наступні | програмні бібліотеки | для розпізнавання зображень | методом головних компонентів і лінійного дискримінантного аналізу:…»

Потім, відповідно до шаблону, результати запиту підставляються через кому, наприклад: «На мові Python | алгоритми машинного навчання | Вам | допоможуть | використовувати | наступні | програмні бібліотеки | для розпізнавання зображень | методом головних компонентів і лінійного дискримінантного аналізу: | scikit-learn, Keras, numpy».

Особливу увагу слід звернути на додаткову обставину з вихідного словосполучення зі значенням іменника в родовому відмінку (genitive_circumstance). У разі нормальної роботи аналізатора вона не повинна фігурувати тут окремим фрагментом, а входити до складу іменної групи. Тут вона введено штучно. Отже, у кращому випадку має бути іменна група «алгоритми машинного навчання для розпізнавання зображень». Але припустимо, чомусь таку довгу і складну іменну групу не було визначено в ході аналізу, а замість неї знайдено два фрагменти: «алгоритми машинного навчання» та «для розпізнавання зображень». Питання про розташування такого фрагмента в шаблоні не настільки однозначне. Це може бути місце після головного слова. У наведеному вище прикладі цей варіант є цілком граматично і стилістично правильним і не змінює значення фрази. Але так

може бути не завжди, тому що слово іменна група в родовому відмінку можуть бути пов’язані з будь-яким з іменників (розірвана іменна група), в даному випадку, швидше за все, пов’язане з даною іменною групою є словосполучення «алгоритми машинного навчання», а «програмні бібліотеки». Більш правильний варіант місця підстановки таких фрагментів – після слова, від якого до нього йде зв'язок. Це можна реалізувати програмно. У цьому випадку місце такого фрагмента в шаблоні можна або опустити, або воно буде умовним. В тому випадку, коли присвійна форма виражена родовим відмінком, наявність такого елемента в розборі словосполучення цілком допустимо (див. приклад 5). Цю групу іменників можна додатково розширити за рахунок включення компонента instrumental_circumstance. У цьому випадку він матиме вигляд: «алгоритми машинного навчання для розпізнавання зображень методом головних компонентів і лінійного дискримінантного аналізому». Отже, можна зробити висновок, що під час визначення іменникових словосполучень на етапі аналізу до них слід включати всі безпосередньо пов’язані іменники незалежно від відмінка. Таким чином ми можемо отримати більш коректну фразу: «У мові Python | алгоритми машинного навчання для розпізнавання зображень методом головних компонентів і лінійного дискримінантного аналізому | Вам | допоможуть | використовувати | наступні | програмні бібліотеки | : scikit-learn, Keras, numpy».

Приклад 5

Фраза користувача: «Яка задача у POST-запиту?». Відповідь на це питання, наприклад, така: «передача даних серверу або клієнту». Визначимо фрагменти фрази, що стосуються тієї чи іншої частини шаблону:

place_circumstance – відсутня;

instrumental_circumstance – відсутня;н

accusative_circumstance – відсутня;

match_predicate – відсутній (позитивний, не виражений спеціальним словом);

verb_circumstance – відсутня;

main_instance – «задача»;

genitive_circumstance – «у POST-запитанні»;

Крім основної сутності («задача») у цьому питанні присутній лише блок у родовому відмінку. Ця позиція заміни була включена до шаблону, як показано вище, як запасний елемент для випадку неповністю зібраної в ході аналізу іменної групи. Але в цьому прикладі це навіть виглядає досить доречно. Відповідь можна сформувати так: «Задача | у POST-запиту: | передача даних серверу або клієнту». При обов'язковій підстановці на позиції 7 виходить: «Така | задача | у POST-запиту: | передача даних серверу або клієнту». В принципі, фраза виглядає досить чітко.

Приклад 6

Фраза користувача: «Які існують фреймворки для бэк-енд розробки?». Тут також є випадок додаткової обставини, вираженої іменною групою у родовому відмінку. Визначимо фрагменти фрази, що стосуються тієї чи іншої частини шаблону:

place_circumstance – відсутня;

instrumental_circumstance – відсутня;

accusative_circumstance – відсутня;

dative_circumstance – відсутня;

match_predicate – «існують»;

verb_circumstance – те ж саме, що і у match_predicate «існують »;

main_instance – «фреймворки»;

genitive_circumstance – «для бэк-енд разработки».

Припустимо, що результати відповіді на запит представлені таким списком понять: Django, Flask, ASP.NET, YII, NodeJS. Відповідь, згідно з шаблоном, буде сформована таким чином: «Існують | наступні | фреймворки | для бэк-енд розробки: | Django, Flask, ASP.NET, YII, NodeJS». У цьому випадку «фреймворки для бэк-енд разработки» також можна розглядати як

єдину групу, але існує ймовірність її розриву при аналізі через наявність у ній прийменника «для».

Приклад 7

Фраза користувача: «Які книги по машинному навчанню у Вашому магазині підходять досвідченому спеціалісту?». Визначимо фрагменти фрази, що стосуються окремих частин шаблону:

place_circumstance – «у Вашому магазині» (буде замінено на «у нашому магазині»);

accusative_circumstance – відсутня;

dative_circumstance – «по машинному навчанню», «досвідченому спеціалісту»;

match_predicate – «підходять»;

verb_circumstance – те саме, що і match_predicate «підходять»;

main_instance – «книги»;

genitive_circumstance – відсутня;

instrumental_circumstance – відсутня.

Припустимо, що з онтології отримано наступний список книг: С. Хайкін «Нейронні мережі: повний курс», К. Аггарвал «Нейронні мережі та глибоке навчання: навчальний курс», С. Рашка, В. Мірджалі «Python і машина». навчання», Девід Дж. Маккей «Теорія інформації, логічний висновок і алгоритми навчання», Карл Едвард Расмуссен «Процеси Гауса для машинного навчання».

Відповідно до шаблону відповідь буде сформована так: «В нашому магазині | по машинному навчанню | досвідному спеціалісту | підходять | наступні | книги: | С. Хайкін «Нейронні мережі: повний курс», Ч. Аггарвал «Нейронні мережі та глибоке навчання: навчальний курс», С. Рашка, В. Мирджалії «Python і машинне навчання», Девід Дж. Маккей «Теорія інформації, логічний висновок і алгоритми навчання», Карл Едвард Расмуссен «Процеси Гауса для машинного навчання».

Згенерована фраза здається зрозумілою. Однак правильніше було б побудувати фразу: «У нашому магазині | досвітченому спеціалісту | підійдуть | наступні | книги по машинному навчанню: | С. Хайкін «Нейронні мережі: повний курс», Ч. Аггарвал «Нейронні мережі та глибоке навчання: навчальний курс», С. Рашка, В. Мирджаліі «Python і машинне навчання», Девід Дж. Маккей «Теорія інформації, логічний висновок і алгоритми навчання», Карл Едвард Расмуссен «Процеси Гауса для машинного навчання». Поняття «книги по машинному навчанню» також можна розглядати як іменну групу, хоча залежний іменник входить до неї в давальному відмінку, а не в родовому. З цього випливає важливий для реалізації процесу розбору висновок: іменні групи повинні включати всі зв’язані між собою іменники відношення незалежно від їх відмінку, за винятком відношення «підмет – іменний присудок», коли обидва іменники стоять у називному відмінку через тире.

У цій роботі в основному описується використання шаблону для синтезу відповідей українською та російською мовами, але варто навести хоча б один приклад для описаного вище англійського шаблону.

Припустімо, фраза користувача наступна: «When does the nearest advanced machine learning course start at your computer school?» («Коли у вашій комп’ютерній школі починається найближчий курс поглибленого машинного навчання?»). Це питання про час зі словом-маркером «when» («коли»). Отже, поданий вище шаблон є прийнятним.

Розглянемо побудову відповіді. Перша частина речення відповіді – це "main_instance". У цьому випадку це іменна група «the nearest advanced machine learning course» («найближчий курс машинного навчання»). Таким чином, початок відповіді має бути таким: «The nearest advanced machine learning course…» («Найближчий курс поглибленого машинного навчання…»). Наступна частина — «match_predicate». Він позитивний і виражається допоміжним дієсловом «does». Але для вираження нейтрального твердження його слід опустити. Далі йде «verb_circumstance». Це дієслово

«starts» («починається»). Підставляючи його, отримуємо: «The nearest advanced machine learning course starts…» («Найближчий курс поглибленого машинного навчання починається…»). У вхідній фразі відсутні такі частини, як "accusative_circumstance", “instrumental_circumstance”, and "dative_circumstance". Отже, ми їх опускаємо. Наступною частиною є "place_circumstance". Вона у нас тут така: «at our computer school» («у нашій комп'ютерній школі»). Після підстановки фраза стає такою: «The nearest advanced machine learning course starts at our computer school» («Найближчий курс поглибленого машинного навчання починається в нашій комп’ютерній школі…»). Далі йде «`introductory`». Його вибір залежить від типу вираження часу. Припустімо, що результат запиту лише один і це дата, наприклад, «1 серпня 2020». Для такого випадку варіант із розділу «`introductory`» повинен бути підставлений варіант «at the». Тож фраза відповіді стає: «The nearest advanced machine learning course starts at our computer school at the…». Остання частина шаблону – це "query_results". Припустімо, результатом запиту є «August 1, 2020» («1 серпня 2020»). Таким чином сформувалася відповідь: «The nearest advanced machine learning course starts at our computer school on August 1, 2020.» («Найближчий курс поглибленого машинного навчання починається в нашій комп’ютерній школі 1 серпня 2020 року»). Ця фраза граматично правильна і може бути відповіддю на запитання користувача.

ДОДАТОК Е

Інструкція-підказка для великої мовної моделі для завдання синтезу природномовного речення українською мовою на основі онтологічного представлення

```
{
     “Intriduction”: “You are an expert in knowledge engineering and ontologies as well as in meaningful text generation in inflect languages. You will be provided with data obtained from some ontology through a query. The ontology was made automatically basing on the results of semantic analysis of a natural language text. The results are pairs of lemmatized words (“main entity” and “dependent entity”) accompanied with a name of syntactic-semantic relationship that linked them in the certain sentence.”,
     “Action to perform”: “Assuming that all the data you will be provided belong to one sentence you are to make a try to restore the original sentence using such a prompt. Language of the ontology, input and output data is Ukrainian.”,
     “Restrictions”: “Do not put the semantic relationships as a phrase as it given in the sentence you generate, it will be definitely wrong. It is just a prompt for syntactic linking. Remember that the provided words are lemmatized, so you are to put them in a correct form according to other entities of the sentence and the given syntactic-semantic relationships of the prompt.”,
     “Additional data to provide”: “Also provide an estimated value of probability that the generated sentence corresponds the intent of the prompt given.”,
     “The essence of the syntactic-semantic relationship names and meaning explanation”:
     {
          “властивість об’єкту: “the dependent entity express a property or some characteristic, or quality of the main entity. When the response sentence generation you should use the dependent entity as an adjective with the main entity which is noun”,
          “властивість дії”: “the dependent entity express a property or some characteristic, or quality of the main entity which is an action. When the response sentence generation you should use the dependent entity as an adverb with the main entity which is verb”,
          “зміна якості”: “the dependent entity express that the main entity may be subjected to some quality changes, which may follow from the other context”,
          “пизначання”: “the dependent entity express the destination of the main entity”,
          “об’єкт”: “the object (noun) affected throw the action expressed by the main entity”,
```

```
          “об’єкт / дія”: “the main entity performs an action
expressed by the dependent entity”,
          “прив’язка прийменника”: “merely shows that the main
entity here in the context of the provided sentence is to be
used with the preposition which is the dependent entity. This
means that you should use this preposition with the main entity
when the response sentence generation”,
          “приналежність”: “the dependent entity or somewhat
relates to the main entity. When generation this usually should
be expresses using genitive case”,
          “тотожність”: “the different name of the entity or an
equivalent entity”,
          “входження об’єктне”: “the main entity is a part or
member of the dependent entity”,
          “стан”: “a state or a constant characteristic of the
main entity if it is noun or an entity linked to in if it is a
verb”
     },
     “Input data”: [
          {
               “main entity”: “some word 1”,
               “dependent entity”: “some word 2”,
               “semantic relationship”: “semantic category 1”
          },
          ……
          ……
          {
               “main entity”: “some word n”,
               “dependent entity”: “some word n+1”,
               “semantic relationship”: “semantic category n”
          }
     ]
}
```

ДОДАТОК Є

Інструкція-підказка для великої мовної моделі для визначення висловлених у повідомлення намірів та їх зв’язку з відповідними сутностями

```
{
  "information to provide": [
    "define intents",
    "find subjects",
    "find objects"
  ],
  "text": "<A text to be analyzed>",
  "language": "Ukrainian",
  "input information field": "text",
  "possible intents": [
    "quantity",
    "place",
    "way of doing",
    "object",
    "subject",
    "action",
    "location",
    "direction",
    "scene of action",
    "conditions",
    "instrument",
    "collaborator",
    "relation",
    "cause",
    "sequence",
    "origin"
  ],
  "several intents": true,
  "intents probability": true,
  "show intent subject": true,
  "max intents number": 4,
  "intents arrange": "by probability",
  "output format": "JSON",
  "output representation template": {
    "result": [
      {
        "intent": "intent name - string",
        "type": "narration, interrogation or imperative",
        "probability": "float value",
        "subject": "subject of the intent as a name group - string",
        "object": "object of the intent as a name or verb group - string"
      }
    ]
  }
}
```