НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«_____»_____2023р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмний застосунок ідентифікації морських об'єктів на основі онтології і прецедентів»

Виконав:
студент IV курсу, групи ТІ-91
Антонюк Артем Олегович _____
(підпис)

Керівник:
доцент каф. ІПЗЕ, д.т.н Мусієнко А.П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
доцент каф. ЦТЕ, к. ф.-м. н., доцент Донець А. Г. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ (підпис)
«____»_____202_р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Антонюку Артему Олеговичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи
«Програмний застосунок ідентифікації морських об'єктів на основі онтології і прецедентів»
керівник роботи Мусієнко Андрій Петрович д.т.н, професор кафедри ІПЗЕ_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від "__"_____2023 року №____
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи: програма надає результати ідентифікації морських об'єктів, включаючи назву, тип, атрибути та метадані.
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): огляд літератури, розробка онтології, аналіз прецедентів, впровадження за допомогою jcolibri, оцінка та результати.____
_____
_____
5. Перелік ілюстративного матеріалу: Основне вікно TopBraid Composer, Діаграма прецедентів, Data flow diagram, Основні класи онтології, Зв'язки між класами та об'єктами, Архітектура JCOLIBRI, Меню запиту, Меню налаштування запиту, Приклад результату запиту, Кнопки для перегляду інших схожих випадків, Кнопка для закриття застосунку

6. Дата видачі завдання «__» _____ 202_ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	30.09.2022	виконано
2	Дослідження предметної області	10.04.2023 – 16.04.2023	виконано
3	Постановка вимог до проєктування системи	17.04.2023 – 20.05.2023	виконано
4	Дослідження існуючих рішень	21.04.2023 – 25.04.2023	виконано
5	Розробка програмного продукту	26.04.2023 – 11.05.2023	виконано
6	Тестування	12.05.2023 – 14.05.2023	виконано
7	Захист програмного продукту	17.05.2023	виконано
8	Оформлення дипломної роботи	22.05.2023 – 28.05.2023	виконано
9	Передзахист	5.06.2023 – 11.06.2023	виконано
10	Захист	19.06.2023 – 25.06.2023	виконано

Студент _____ Антонюк Артем
(підпис) (ім'я, прізвище)

Керівник роботи _____ Андрій Петрович
(підпис) (ім'я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 51 сторінки, 11 рисунків, 2 додатки та 12 посилань.

Метою роботи є розробка програмної системи для ідентифікації морських об’єктів за допомогою онтології та прецедентів, використовуючи структуру JCOLIBRI. Результати цієї роботи можуть бути використані різними організаціями, такими як агентства морської безпеки, судноплавні компанії та науково-дослідні установи, щоб покращити свою роботу та покращити розуміння морського середовища.

Для досягнення поставленої мети виконано наступні завдання:

- описано онтологію та прецеденти;
- розроблено алгоритм ідентифікації об’єктів на основі JCOLIBRI;
- розроблено зручний інтерфейс;
- проаналізовано отримані дані та визначено ефективність роботи алгоритму ідентифікації об’єктів.

Практичне значення одержаних результатів полягає в отриманні інструменту, що дозволить на основі отриманої інформації ідентифікувати морські об’єкти. Результати цієї роботи можуть бути використані різними організаціями, такими як агентства морської безпеки, судноплавні компанії та науково-дослідні установи, щоб покращити свою роботу та покращити розуміння морського середовища.

Ключові слова: морські об’єкти, онтологія, прецеденти, ідентифікація, фреймворк JCOLIBRI, класифікація, штучний інтелект, інтелектуальний аналіз даних, база знань, системи на основі правил, експертні системи, заходи подібності, підтримка прийняття рішень.

ABSTRACT

Structure and scope of the thesis. The work contains 51 pages, 11 figures, 2 appendices and 12 references.

The purpose of the work is the development of a software system for the identification of marine objects using ontology and precedents, using the JCOLIBRI framework. The results of this work can be used by various organizations such as maritime safety agencies, shipping companies and research institutions to improve their work and to improve their understanding of the marine environment.

To achieve the goal, the following tasks were performed:

- the ontology and precedents are described;
- an object identification algorithm based on JCOLIBRI was developed;
- a convenient interface has been developed;
- the obtained data were analyzed and the efficiency of the object identification algorithm was determined.

The practical value of the obtained results consists in obtaining a tool that will allow to identify marine objects on the basis of the received information. The results of this work can be used by various organizations such as maritime safety agencies, shipping companies and research institutions to improve their work and to improve their understanding of the marine environment.

Keywords: marine objects, ontology, precedents, identification, JCOLIBRI framework, classification, artificial intelligence, data mining, knowledge base, rule-based systems, expert systems, similarity measures, decision support.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Онтологія – формальний опис знань про поняття, взаємозв'язки між ними та властивості об'єктів у певній предметній області. Онтології використовуються для організації і структуризації інформації, що дозволяє програмам аналізувати та розуміти дані більш ефективно.

Прецедент – ситуація, яка вже сталася і може служити моделлю або прикладом для вирішення подібних проблем. Прецеденти можуть використовуватися для побудови моделей і рекомендаційних систем, що допомагають в ідентифікації морських об'єктів.

Ідентифікація – процес визначення або розпізнавання об'єкта на підставі його характеристик і властивостей.

JCOLIBRI – це фреймворк, який допомагає в розробці систем, заснованих на кейс-підході (Case-Based Reasoning), який використовує прецеденти для прийняття рішень.

Морські об'єкти – об'єкти або структури, які існують або знаходяться в морському середовищі, такі як кораблі, риба, коралові рифи, айсберги, буйки тощо.

Метадані – дані, які надають додаткову інформацію про морські об'єкти, такі як географічні координати, джерело даних, тип судна, розмір і швидкість, стан моря та температура води, тощо.

Кейс-підхід (Case-Based Reasoning) – метод штучного інтелекту, в якому рішення приймається на основі аналізу аналогічних ситуацій з минулого (прецедентів). Кейс-підхід можна використовувати для ідентифікації морських об'єктів, порівнюючи їх з вже відомими прецедентами.

Розпізнавання об'єктів – процес визначення або класифікації об'єктів на основі їх характеристик або особливостей. Розпізнавання об'єктів може означати

визначення типу морських об'єктів (наприклад, риба, судно, маяк) на підставі їх атрибутів.

Семантичні мережі – модель знань, в якій поняття та взаємозв'язки між ними представлені у вигляді вузлів та ребер графа. Можна використовувати семантичні мережі для визначення та моделювання взаємозв'язків між різними морськими об'єктами у онтології.

ВСТУП

Морські об’єкти, такі як кораблі, підводні човни та інші морські транспортні засоби, відіграють вирішальну роль у різних сферах, включаючи транспорт, військові операції та наукові дослідження. Точна ідентифікація цих об’єктів має важливе значення для підтримки морської безпеки та забезпечення ефективних операцій у морському середовищі. Однак завдання ідентифікації морських об’єктів може бути складним, оскільки вимагає знання різних параметрів, таких як тип судна, розмір і швидкість, а також факторів навколишнього середовища, таких як погодні умови, стан моря та температура води.

Одним із можливих рішень цієї проблеми є розробка автоматизованої системи ідентифікації морських об’єктів на основі онтології та прецедентів. Ця система може використовувати алгоритми машинного навчання для аналізу даних і визначення шаблонів, які можна використовувати для класифікації морських об’єктів на основі їхніх характеристик. Крім того, використовуючи онтологію для визначення понять предметної області та зв’язків, система може гарантувати послідовність і точність представлення знань, сприяючи ефективній та надійній ідентифікації морських об’єктів.

Крім того є важливим використанням вже здобутих знань про об’єкти для ідентифікації, тож застосування онтології є дуже важливою частиною роботи.

Для досягнення цієї мети фреймворк JCOLIBRI можна використовувати для розробки системи програмного забезпечення, яка включає в себе машинне навчання та підходи на основі онтології для ідентифікації морських об’єктів. Ця система може враховувати різні типи джерел даних, такі як сигнали AIS (системи автоматичної ідентифікації), дані радарів і гідролокаторів, а також супутникові зображення, щоб надати повну та точну картину морського середовища. Система також може включати міркування на основі прецедентів, коли система навчається на минулому досвіді та відповідно адаптує свою поведінку, підвищуючи свою

точність і ефективність з часом. А додаток Protégé можна використовувати для побудови онтології, а саме визначення класів об’єктів, їх характеристики, атрибути та зв’язки.

Метою цієї дипломної роботи є розробка програмної системи для ідентифікації морських об’єктів за допомогою онтології та прецедентів, використовуючи структуру JCOLIBRI. Результати цієї роботи можуть бути використані різними організаціями, такими як агентства морської безпеки, судноплавні компанії та науково-дослідні установи, щоб покращити свою роботу та покращити розуміння морського середовища.

1 МЕТА, ПОСТАНОВКА ЗАДАЧІ ТА ОСНОВНІ ЗАВДАННЯ

Метою даної роботи є розробка методу ідентифікації морських об'єктів на основі онтології та прецедентів з метою поліпшення автоматизованого процесу розпізнавання та класифікації таких об'єктів. Онтологія в цьому контексті використовується для формалізації знань про морські об'єкти, їх властивості та взаємозв'язки, що дозволяє створити однозначну модель для їх розпізнавання.

Для розробки методу ідентифікації буде використаний фреймворк JCOLIBRI, який забезпечує потужні інструменти для розробки експертних систем та систем рекомендацій. JCOLIBRI дозволяє використовувати методи машинного навчання, індуктивного та аналогового мислення, що сприяє розвитку комплексних інтелектуальних моделей для ідентифікації морських об'єктів.

Прецеденти в даній роботі використовуються для попереднього накопичення досвіду ідентифікації морських об'єктів, що дозволить системі використовувати попередні випадки для прийняття рішень в нових ситуаціях. Прецеденти включатимуть інформацію про попередні випадки успішної ідентифікації, що дозволить

У процесі розробки програмного забезпечення для ідентифікації морських об'єктів будуть використані різноманітні інструменти та фреймворки, зокрема JCOLIBRI, що забезпечать ефективну обробку та аналіз даних, а також побудову та управління онтологією морських об'єктів.

Використання JCOLIBRI дозволить зручно виконувати обробку великого обсягу даних про морські об'єкти, включаючи збір, зберігання та індексацію. Фреймворк надає інтерфейс для ефективного розподіленого обчислення та паралельної обробки даних, що сприяє підвищенню швидкості та продуктивності ідентифікації.

У роботі будуть вирішені такі завдання:

- розробка онтології морських об'єктів, яка відображатиме їх характеристики, зв'язки та взаємодії;
- визначення прецедентів ідентифікації, що базуються на попередніх випадках розпізнавання морських об'єктів;
- розробка алгоритмів ідентифікації на основі комбінації онтологічного підходу та прецедентів;
- розробка алгоритмів ідентифікації на основі комбінації онтологічного підходу та прецедентів.

Нижче описані вимоги до складових частин програмного продукту.

Програмний продукт має відповідати наступним вимогам:

- програмний продукт повинен забезпечувати швидку та ефективну обробку даних морських об'єктів для ідентифікації;
- програмний продукт повинен забезпечувати високу точність ідентифікації морських об'єктів;
- програмний продукт повинен мати зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

Клієнтська частина має відповідати наступним вимогам.

Висновки до розділу 1

У цьому розділі бакалаврської дипломної роботи була сформульована мета. Крім того, був проведений аналіз та описано основні вимоги до компонентів програмного продукту. Також був наведений короткий огляд початкових інструментів, які планується використовувати під час розробки та необхідні задачі, які мають бути вирішені протягом написання роботи.

2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ, МЕТОДІВ, МОДЕЛЕЙ, ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

Аналіз існуючих систем, методів, моделей і опис предметної області є ключовими компонентами в дослідженнях ідентифікації морських об'єктів на основі онтології та прецедентів. Вивчаючи попередню роботу, можна отримати цінну інформацію про поточні найсучасніші підходи, визначити проблеми та відкрити успішні методології. Цей аналіз допомагає удосконалити власні методології, усунути прогалини в поточних підходах і зробити внесок у вдосконалення методів ідентифікації морських об’єктів. Поєднання онтології та прецедентів дозволяє ефективно організувати та міркувати про морські об’єкти, використовуючи минулий досвід та експертні знання для більш точної та надійної ідентифікації. Зрештою, це дослідження має значні наслідки для моніторингу навколишнього середовища, морської безпеки та зусиль щодо збереження.

2.1 Аналіз існуючих систем

На ринку застосунків представлено багато альтернативних рішень і систем. Це включає в себе різноманітні комерційні та відкриті програмні продукти, фреймворки та бібліотеки, які пропонують різні підходи для розробки онтології та до ідентифікації об'єктів.

2.1.1 Owlready2

Owlready2 — це бібліотека Python для роботи з онтологічно-орієнтованим програмуванням і семантичними веб-додатками. Він надає API високого рівня, який дозволяє розробникам створювати, маніпулювати та запитувати онтології за допомогою мови веб-онтології (OWL). Бібліотека побудована на основі бібліотеки

RDFLib, яка забезпечує потужну структуру для роботи з даними RDF (Resource Description Framework).

Однією з ключових особливостей Owlready2 є його підтримка міркувань за допомогою онтологій. Він включає в себе інтегровану аргументацію на основі аргументу HermiT, яка дозволяє автоматично класифікувати та робити висновки щодо онтологій. Ця можливість аргументації дозволяє розробникам використовувати весь потенціал онтологій шляхом отримання неявних знань із явних знань, викладених в онтології.

Owlready2 підтримує різні профілі OWL, включаючи OWL 2 DL, OWL 2 EL і OWL 2 RL, що дозволяє розробникам вибирати відповідний рівень виразності та можливостей аргументації для своїх програм. Бібліотека також забезпечує підтримку імпорту та експорту онтологій у різних форматах, таких як OWL/XML, RDF/XML і Turtle, що робить її сумісною з іншими інструментами та фреймворками в екосистемі семантичної мережі.

На додаток до основних функцій онтології, Owlready2 пропонує зручні функції для роботи з онтологічними анотаціями, представленнями онтології та запитами на основі онтології. Він надає Pythonic та об'єктно-орієнтований інтерфейс для створення та маніпулювання сутностями в онтології, такими як класи, особи, властивості та аксіоми. Це полегшує розробникам інтеграцію онтологічних міркувань і семантичних веб-можливостей у свої програми Python.

Owlready2 набув популярності серед дослідників, розробників і спеціалістів із обробки даних, які працюють у сфері семантичної мережі та онтологічної інженерії. Його інтуїтивно зрозумілий і потужний API у поєднанні з обширною документацією та активною підтримкою спільноти роблять його цінним інструментом для створення інтелектуальних програм, які використовують онтології та семантичні технології. Незалежно від того, чи розробляєте ви системи, засновані на знаннях, семантичні пошукові системи або рішення для інтеграції

даних, Owlready2 надає гнучку та ефективну платформу для роботи з онтологіями на Python [1].

2.1.2 TopBraid Composer

TopBraid Composer — потужний і універсальний інструмент моделювання та розробки семантичного веб-сайту. Він призначений для полегшення створення, керування та дослідження графів знань і онтологій. Розроблений TopQuadrant, TopBraid Composer пропонує широкий набір функцій, які дозволяють користувачам створювати семантичні моделі та керувати ними, інтегрувати дані з багатьох джерел і розробляти семантичні програми. Інтерфейс прогарми наведено нижче (рис 2.1). Це основне вікно прграми.

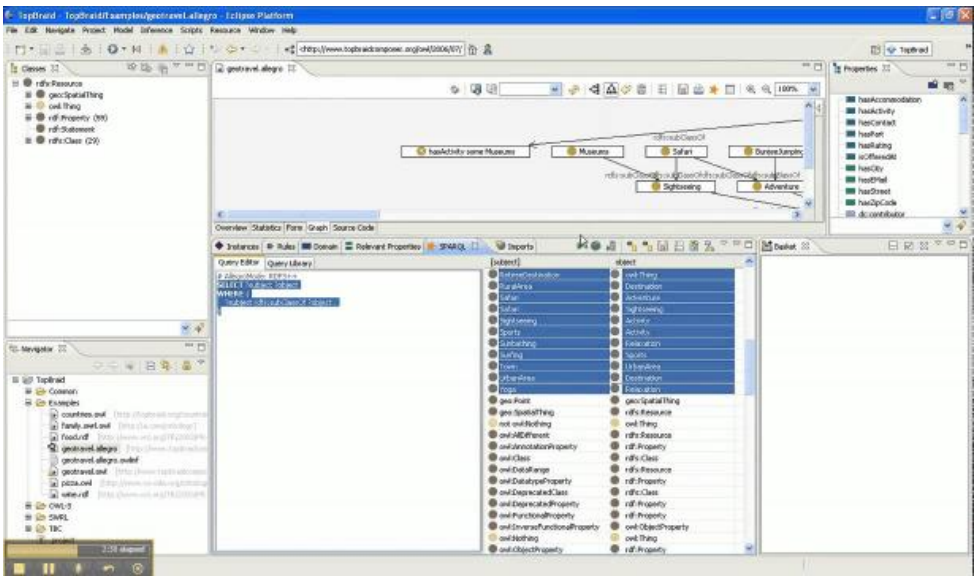

Рисунок 2.1 — Основне вікно TopBraid Composer

По суті, TopBraid Composer забезпечує зручний інтерфейс для створення та редагування графіків RDF (Resource Description Framework) і онтологій OWL (Web Ontology Language). Він підтримує різні методи моделювання, включаючи ієрархії класів і властивостей, правила висновку та обмеження. Користувачі можуть

візуально проектувати свої онтології за допомогою редактора на основі графів або безпосередньо редагувати базовий код RDF і OWL [2].

Однією з ключових переваг TopBraid Composer є підтримка інтеграції даних. Це дозволяє користувачам інтегрувати дані з різних джерел, включаючи бази даних, електронні таблиці, XML-файли та веб-сервіси, у свої семантичні моделі. Інструмент пропонує багатий набір можливостей трансформації та відображення, що дозволяє користувачам визначати складні робочі процеси інтеграції даних. Це робить його безцінним ресурсом для створення та підтримки графіків знань, які поєднують структуровані та неструктуровані дані.

На додаток до своїх можливостей моделювання та інтеграції, TopBraid Composer забезпечує надійну підтримку для розробки онтологій і керування ними. Він пропонує розширені функції для керування версіями онтологій, співпраці та документування, що полегшує командну роботу над великомасштабними проектами. Інструмент також підтримує автоматичне обґрунтування та перевірку, допомагаючи користувачам виявляти та вирішувати невідповідності або логічні помилки в їхніх онтологіях.

Крім того, TopBraid Composer містить низку вбудованих інструментів для запитів, дослідження та візуалізації графіків знань. Користувачі можуть виконувати запити SPARQL для отримання даних зі своїх семантичних моделей і використовувати можливості візуалізації інструменту, щоб отримати уявлення про структуру та зв’язки в межах графіка. Це полегшує дослідження та аналіз складних наборів даних і виявлення прихованих закономірностей або зв’язків.

Загалом, TopBraid Composer — це комплексний і багатофункціональний інструмент для розробки семантичної мережі та керування графами знань. Його інтуїтивно зрозумілий інтерфейс, потужні можливості інтеграції та підтримка розробки онтологій роблять його цінним активом для організацій і дослідників, які працюють із семантичними технологіями. Незалежно від того, чи йдеться про

створення онтологій, інтеграцію даних чи дослідження графів знань, TopBraid Composer забезпечує надійне та гнучке середовище для підтримки цих завдань.

2.1.3 MyCBR

MyCBR, що означає «my Case-Based Reasoning», — це потужний і універсальний інструмент, який використовується в галузі штучного інтелекту та інженерії знань. Це фреймворк програмного забезпечення з відкритим вихідним кодом, який дозволяє розробляти та розгортати системи міркування на основі випадків (CBR). CBR – це підхід до вирішення проблем, який використовує минулий досвід або випадки для вирішення нових проблем [3].

Основною концепцією MyCBR є представлення та пошук випадків. Кейси складаються з опису проблеми, рішення та їх зв’язку. Фреймворк дозволяє користувачам визначати власні структури випадків і адаптувати їх до свого конкретного домену чи програми. Ця гнучкість робить MyCBR придатним для широкого спектру завдань із вирішення проблем у різних галузях.

Однією з ключових особливостей MyCBR є його здатність навчатися та адаптуватися з часом. Він використовує методи машинного навчання, щоб покращити свою продуктивність шляхом автоматичного оновлення та вдосконалення бази випадків. У міру додавання нових і перегляду наявних випадків MyCBR може реорганізовувати та оптимізувати свої знання, щоб покращити свої можливості вирішення проблем.

Розробка систем CBR з використанням MyCBR включає кілька етапів. По-перше, проблемна область аналізується для виявлення відповідних функцій і зв’язків. Далі створюється база випадків шляхом збору та впорядкування минулого досвіду. База випадків індексується та зберігається для ефективного пошуку. Нарешті, систему можна навчити та перевірити за допомогою різних алгоритмів навчання для підвищення її точності та продуктивності.

MyCBR було успішно застосовано в багатьох областях, таких як медична діагностика, підтримка клієнтів і аналіз несправностей. Його здатність використовувати минулий досвід і адаптуватися до нових ситуацій робить його особливо корисним у динамічних і складних середовищах. Використовуючи силу аргументації на основі конкретних випадків, MyCBR дає можливість організаціям приймати більш обґрунтовані рішення та ефективно вирішувати проблеми.

Підсумовуючи, MyCBR — це фреймворк програмного забезпечення з відкритим вихідним кодом, який полегшує розробку та розгортання систем аргументації на основі випадків. Це дозволяє представляти, витягувати та адаптувати випадки, дозволяючи користувачам використовувати попередній досвід для вирішення нових проблем. Завдяки можливостям машинного навчання MyCBR постійно покращує свою продуктивність, оновлюючи та удосконалюючи свою базу випадків. Він був успішно застосований у різних областях, що робить його цінним інструментом у сфері штучного інтелекту та інженерії знань.

2.1.4 jColibri.NET

jColibri.NET — це потужна та універсальна бібліотека на основі Java, розроблена для розробки інтелектуальних систем, які використовують методи аргументації на основі випадків (CBR). Це еволюція оригінальної бібліотеки jColibri, що пропонує розширені функції та можливості для програм CBR. Одна помітна відмінність полягає в тому, що jColibri.NET спеціально розроблено для роботи в рамках .NET, що робить його сумісним із широким спектром платформ і технологій [4].

Однією з значних переваг jColibri.NET є його покращена продуктивність і масштабованість. Він пропонує ефективні механізми індексації та пошуку, що дозволяє швидко й точно шукати справи у великих базах справ. Бібліотека також надає передові алгоритми для оцінки подібності та адаптації регістру, що дозволяє

розробникам створювати надійні та адаптивні системи CBR. Крім того, jColibri.NET пропонує повну інтеграцію з різними джерелами даних, включаючи бази даних і файли XML, що полегшує включення та використання різноманітних наборів даних.

Однак, як і будь-яка бібліотека програмного забезпечення, jColibri.NET також має свої обмеження та недоліки. Одним із помітних обмежень є відносно крута крива навчання, пов'язана з використанням jColibri.NET. Хоча бібліотека пропонує обширну документацію та приклади, освоїти її тонкощі та зрозуміти основні концепції CBR може бути складним для новачків. Крім того, як і будь-який фреймворк, jColibri.NET може мати певні обмеження, коли справа доходить до налаштування та розширення його функцій за межі наданих модулів. Розробники можуть зіткнутися з ситуаціями, коли їм потрібно змінити бібліотеку або обійти її обмеження, щоб відповідати конкретним вимогам.

Незважаючи на ці недоліки, jColibri.NET залишається цінним інструментом для розробки додатків CBR у рамках .NET. Його покращена продуктивність, масштабованість і можливості інтеграції роблять його підходящим вибором для розробників, які прагнуть використовувати потужність аргументації на основі прецедентів у своїх проектах. Завдяки ретельному розгляду та відданості jColibri.NET може надати розробникам можливість створювати інтелектуальні системи, які ефективно вирішують складні проблеми, використовуючи минулий досвід і приклади.

2.2 Опис предметної області

Морські об'єкти відносяться до різних об'єктів, які знаходяться в морському середовищі, таких як підводні човни, кораблі, буї, морські ссавці або будь-які інші об'єкти, присутні в океані. Процес ідентифікації включає визначення назви, типу, атрибутів і метаданих, пов'язаних із цими морськими об'єктами.

Онтологія відіграє вирішальну роль у цьому дослідженні, оскільки вона забезпечує структуроване представлення предметних знань, пов’язаних з морськими об’єктами. Він визначає поняття, властивості та взаємозв’язки між різними об’єктами, що забезпечує систематичний і стандартизований підхід до ідентифікації та класифікації морських об’єктів.

Крім того, дана тема включає концепцію прецедентів, яка передбачає використання минулого досвіду та знань для покращення процесу ідентифікації. Аналізуючи попередні випадки та навчаючись на них, платформа може приймати обґрунтовані рішення та підвищувати точність ідентифікації морських об’єктів.

Фреймворк jcolibri служить практичним інструментом для реалізації процесу ідентифікації. Він складається з алгоритмів, моделей і методів, які використовують онтологію та підходи на основі прецедентів для досягнення ефективних результатів ідентифікації. Ця структура потенційно використовує методи машинного навчання, інтелектуального аналізу даних і розпізнавання образів для аналізу та інтерпретації атрибутів і метаданих, пов’язаних з морськими об’єктами.

Висновки до розділу 2

У другому розділі було розглянуто провідні аналоги програм, проведено аналіз та їх основний функціонал. Було описано предметну область теми дипломної роботи. Розглянуто проблематику та основні вимоги пов’язані з предметною областю.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі буде наведено та описано використані технології під час розробки даного програмного продукту, а також їх ключові переваги, через які вибір на користь використання було віддано їм.

3.1 Мова програмування Java

Java — це універсальна та широко використовувана мова програмування, яка набула величезної популярності з моменту свого створення в середині 1990-х років. Розроблена компанією Sun Microsystems (тепер належить корпорації Oracle), Java була розроблена з метою бути незалежною від платформи, безпечною та надійною. З тих пір він став одним із найпопулярніших варіантів для створення широкого спектру додатків, включаючи веб-, корпоративні, мобільні та вбудовані системи.

Однією з ключових особливостей, яка відрізняє Java від інших, є її принцип "напиши один раз, запусти будь-де". Це означає, що програми на Java можна скомпільувати в байт-код, який потім можна виконати на будь-якій платформі за допомогою віртуальної машини Java (JVM). Ця крос-платформна сумісність робить Java дуже портативною та дозволяє розробникам націлюватися на декілька операційних систем, не переписуючи всю кодову базу.

Парадигма об'єктно-орієнтованого програмування (ООП) Java є ще одним важливим аспектом її привабливості. Він забезпечує структурований підхід до розробки програмного забезпечення, дозволяючи розробникам моделювати об’єкти реального світу, зв’язки та поведінку у своїх програмах. Мова включає такі функції, як класи, об’єкти, успадкування та поліморфізм, які сприяють повторному використанню коду, модульності та розширюваності.

Крім того, Java має велику стандартну бібліотеку, яка пропонує багатий набір вбудованих класів і API. Ця бібліотека надає розробникам готові до використання компоненти для типових завдань, таких як операції введення/виведення, мережеве підключення до бази даних і розробка інтерфейсу користувача. Доступність цих ресурсів скорочує час і зусилля на розробку, дозволяючи програмістам зосередитися на вирішенні конкретних бізнес-проблем, а не винаходити колесо.

Надійність і функції безпеки Java роблять її особливо хорошою для критично важливих програм. Він включає автоматичне керування пам'яттю за допомогою збирання сміття, що допомагає запобігти витоку пам'яті та переповненню буфера. Крім того, надійні механізми перевірки типів і обробки винятків Java сприяють стабільності та стійкості мови.

Ще однією помітною характеристикою Java є її процвітаюча екосистема. Мова користується широкою підтримкою спільноти з величезним набором відкритих бібліотек, фреймворків та інструментів, доступних для розробників. Ці ресурси забезпечують швидку розробку, повторне використання коду та інтеграцію з іншими технологіями, сприяючи живому та спільному середовищу розробки.

За останні роки Java також зробила значний крок вперед, щоб не відставати від нових тенденцій у індустрії програмного забезпечення. Він охопив такі сучасні концепції, як функціональне програмування, завдяки введенню лямбда-виразів і потоків у Java 8. Крім того, Java адаптувалася до зростання контейнеризації та хмарних обчислень, забезпечуючи підтримку таких технологій, як Docker і Kubernetes.

Загалом, універсальність Java, незалежність від платформи, сильні принципи ООП, обширна стандартна бібліотека, надійність, безпека та динамічна екосистема зробили її популярним вибором для широкого кола програм. Його здатність масштабуватись від невеликих настільних утиліт до великомасштабних корпоративних систем у поєднанні з його акцентом на продуктивності розробника

та зручності обслуговування коду зміцнює його позицію як основної мови для багатьох розробників програмного забезпечення та організацій у всьому світі [5].

3.2 Середовище розробки програмного забезпечення Eclipse

Eclipse — це інтегроване середовище розробки (IDE) з відкритим кодом, яке широко використовується розробниками для різних мов програмування, включаючи Java. Він забезпечує надійне та багатофункціональне середовище, яке полегшує розробку програмного забезпечення, налагодження та розгортання. Eclipse набув значної популярності серед розробників завдяки своїй гнучкості, розширюваності та широкому набору плагінів та інструментів [6].

Однією з ключових особливостей Eclipse є його модульна архітектура, яка дозволяє розробникам налаштовувати IDE відповідно до своїх конкретних вимог. Eclipse складається з основної платформи, відомої як Eclipse Platform, яка забезпечує базову функціональність для створення IDE. Потім розробники можуть розширити цю платформу, додавши різні плагіни та інструменти, такі як плагін Java Development Tools (JDT), який дозволяє розробляти Java в Eclipse.

Eclipse пропонує повний набір функцій для розробки програмного забезпечення, що робить його потужним інструментом для розробників. Він містить такі функції, як редагування коду з підсвічуванням синтаксису, доповнення коду та інструменти рефакторингу, які допомагають підвищити продуктивність і якість коду. IDE також підтримує системи контролю версій, такі як Git, що дозволяє розробникам ефективно керувати вихідним кодом.

Іншою помітною особливістю Eclipse є його вбудовані можливості налагодження. Розробники можуть встановлювати контрольні точки, виконувати покроковий код, перевіряти змінні та аналізувати потік програми під час виконання, що полегшує виявлення та виправлення помилок. Eclipse також надає

конструктор графічного інтерфейсу користувача, відомий як WindowBuilder, який спрощує створення графічного інтерфейсу користувача для програм.

Крім того, Eclipse підтримує різні фреймворки та технології, що робить його придатним для різних типів проектів. Наприклад, платформа веб-інструментів Eclipse (WTP) дозволяє розробляти веб-додатки за допомогою Java EE, HTML, CSS і JavaScript. Eclipse Modeling Framework (EMF) дозволяє розробникам створювати моделі домену та генерувати з них код, оптимізуючи процес розробки.

Спільнота Eclipse є великою та активною, з великою кількістю розробників, які роблять внесок у її екосистему. Ця активна спільнота забезпечує постійне вдосконалення, часті оновлення та доступність широкого спектру плагінів та інструментів. Він також надає розширену документацію, навчальні посібники та форуми, що полегшує розробникам розпочинати роботу та звертатися за допомогою за потреби.

Підсумовуючи, Eclipse — це потужна та універсальна IDE, яка пропонує широкий спектр функцій та інструментів для розробки програмного забезпечення. Його модульна архітектура, розгалужена екосистема плагінів і активна спільнота роблять його популярним вибором серед розробників. Незалежно від того, розробляєте ви додатки Java, веб-додатки чи працюєте з іншими мовами програмування, Eclipse забезпечує надійне та ефективне середовище для оптимізації процесу розробки та підвищення продуктивності.

3.3 Java фреймворк JCOLIBRI

JCOLIBRI — це потужний фреймворк, призначений для підтримки розробки інтелектуальних систем, які включають аргументацію на основі випадків (CBR) для вирішення проблем. CBR — це парадигма вирішення проблем, яка використовує минулий досвід або випадки для вирішення нових проблем шляхом пошуку подібних випадків і адаптації їх рішень до поточної ситуації. JCOLIBRI спеціально

фокусується на застосуванні методів CBR для ідентифікації морських об’єктів, використовуючи принципи онтології та прецедентів.

За своєю суттю JCOLIBRI забезпечує гнучку та розширювану архітектуру для створення систем CBR. Він пропонує широкий спектр функціональних можливостей, включаючи представлення випадків, оцінку подібності, пошук, адаптацію та оцінку. Ці компоненти працюють разом, щоб забезпечити ідентифікацію морських об’єктів на основі їхніх атрибутів, метаданих і контекстної інформації [7].

Однією з ключових особливостей JCOLIBRI є його онтологічний підхід. Онтологія — це формальне представлення знань, яке визначає поняття, зв’язки та властивості в межах домену. Використовуючи онтологію, JCOLIBRI дає змогу структуровано та узгоджено моделювати та організовувати знання про морські об’єкти. Це полегшує ефективний пошук і міркування щодо відповідних випадків для завдань ідентифікації.

Крім того, JCOLIBRI включає концепцію прецедентів у процес ідентифікації. Прецеденти стосуються раніше вирішених випадків, які служать прикладами для вирішення подібних майбутніх проблем. JCOLIBRI дозволяє зберігати та отримувати прецеденти, дозволяючи системі вчитися на минулому досвіді та з часом покращувати свою продуктивність. Компонент адаптації JCOLIBRI гарантує, що рішення, отримане з прецеденту, буде відповідним чином модифіковано для вирішення конкретних характеристик поточної проблеми.

Програма, створена на основі фреймворку JCOLIBRI, надає результати ідентифікації морських об’єктів, які включають назву, тип, атрибути та метадані ідентифікованих об’єктів. Ці результати отримані шляхом використання онтології та прецедентів, що зберігаються в системі. Фреймворк сприяє ефективній і точній ідентифікації, використовуючи знання, отримані в онтології, і застосовуючи методи CBR для пошуку відповідних прецедентів.

Загалом JCOLIBRI пропонує надійну та гнучку структуру для ідентифікації морських об’єктів на основі онтології та прецедентів. Його інтеграція методів CBR, моделювання онтології та управління прецедентами забезпечує ефективне представлення знань, пошук і адаптацію. Фреймворк дає змогу розробникам створювати інтелектуальні системи, які можуть ідентифікувати морські об’єкти, використовуючи попередній досвід і структуровані знання предметної області. Використовуючи JCOLIBRI, дослідники та практики можуть підвищити точність та ефективність ідентифікації морських об’єктів, сприяючи таким різноманітним сферам, як морська біологія, моніторинг навколишнього середовища та безпека на морі.

3.4 Java фреймворк Swing

Swing — потужний і популярний фреймворк Java для створення графічних інтерфейсів користувача (GUI). Він є частиною Java Foundation Classes (JFC) і надає набір компонентів, інструментів і утиліт, які дозволяють розробникам створювати інтерактивні та візуально привабливі настільні програми. Swing було представлено як заміну старішого Abstract Window Toolkit (AWT) і пропонує значні покращення з точки зору функціональності, гнучкості та продуктивності.

Однією з ключових переваг Swing є його незалежність від платформи. Компоненти Swing повністю написані на Java і не покладаються на рідні віджети, які надає операційна система. Це означає, що програми на основі Swing можуть працювати на будь-якій платформі, яка підтримує Java, включаючи Windows, macOS і Linux, не вимагаючи жодних змін. Така портативність є головною перевагою для розробників, які хочуть, щоб їхні програми охопили широку аудиторію.

Swing надає багатий набір компонентів GUI, таких як кнопки, мітки, текстові поля, таблиці та меню, які можна легко налаштувати та впорядкувати для

створення складних та інтуїтивно зрозумілих інтерфейсів користувача. Ці компоненти легко конфігуруються та пропонують широкий спектр можливостей для керування їхнім виглядом і поведінкою. Крім того, Swing підтримує використання менеджерів макета, які дозволяють розробникам динамічно налаштовувати розташування та розміри компонентів, гарантуючи, що графічний інтерфейс добре адаптується до різних роздільних здатностей екрана та розмірів вікон.

Іншою помітною особливістю Swing є підтримка програмування, керованого подіями. Компоненти Swing генерують події, такі як натискання кнопок або рухи миші, які можуть бути захоплені та оброблені слухачами подій. Це дозволяє розробникам створювати адаптивні та інтерактивні програми, визначаючи конкретні дії чи поведінку, які мають відбуватися у відповідь на взаємодії користувача. Обробка подій у Swing відповідає моделі подій JavaBeans, яка сприяє слабкому зв'язку та полегшує повторне використання коду.

Swing також надає повний набір інструментів і утиліт для створення надійних і багатофункціональних програм. Серед них підтримка інтернаціоналізації та доступності, а також розширені функції, як-от функція перетягування, перевірка даних і можливості друку. Розширена документація Swing і велика онлайн-спільнота дозволяють розробникам легко знаходити ресурси, навчальні посібники та приклади, які допоможуть їм вивчити та ефективно використовувати структуру.

Незважаючи на численні переваги, останніми роками Swing зазнав певної критики через появу альтернативних фреймворків GUI, таких як JavaFX і веб-технологій, таких як HTML5 і JavaScript. Ці новіші фреймворки пропонують розширені можливості, кращу продуктивність і покращену інтеграцію з сучасними веб-технологіями. Як наслідок, популярність Swing впала, і багато розробників перейшли на ці нові альтернативи [8].

Підсумовуючи, Swing є потужним і гнучким фреймворком Java для створення програм графічного інтерфейсу користувача. Його незалежність від

платформи, великий набір компонентів і утиліт, а також підтримка керованого подіями програмування роблять його популярним вибором для розробників. Хоча він зіткнувся з конкуренцією з боку нових фреймворків, Swing продовжує залишатися життєздатним варіантом для створення настільних програм, особливо для тих, які націлені на кілька платформ або потребують розширеного налаштування та контролю над інтерфейсом користувача.

3.5 Protege

Protege — це широко використовуваний і високо оцінений інструмент розробки онтологій, який відіграє вирішальну роль у сфері представлення знань і семантичних технологій. Розроблений у Стенфордському університеті, Protege забезпечує зручний та інтуїтивно зрозумілий інтерфейс для створення, редагування та підтримки онтологій. Він призначений для підтримки побудови онтологій для різних областей і є особливо корисним у контексті ідентифікації морських об'єктів на основі онтології та прецедентів, як ви згадали у своїй дисертації.

Однією з ключових особливостей Protege є його гнучкість у підтримці різних мов онтології, включаючи мову веб-онтології (OWL). OWL — це стандартизована мова для представлення знань і онтологій в Інтернеті, і Protege пропонує комплексну підтримку для створення онтологій OWL. Це дозволяє дослідникам і експертам у галузі визначати морські об'єкти, їхні властивості, взаємозв'язки та класифікації структурованим і стандартизованим способом.

Protege надає багатий набір інструментів і функцій для розробки онтологій. Він пропонує графічний інтерфейс користувача, який дозволяє користувачам візуально створювати та редагувати класи, властивості та екземпляри. Цей візуальний підхід спрощує процес побудови онтології та дозволяє користувачам легко визначати структуру та семантику морських об'єктів. Protege також підтримує розширені функції моделювання, такі як визначення ієрархії класів,

визначення обмежень властивостей і створення правил і аксіом для охоплення складних зв’язків і обмежень у межах домену.

Окрім редагування онтології, Protege пропонує потужні можливості аргументації. Він підтримує різні механізми міркування, які можуть виводити нові знання на основі визначеної онтології та її аксіом. Ці можливості міркування дозволяють дослідникам перевіряти послідовність онтології, перевіряти наявність логічних помилок і отримувати неявну інформацію про морські об’єкти. Це автоматизоване міркування значно підвищує точність і повноту результатів ідентифікації морських об’єктів, отриманих завдяки застосуванню фреймворку jcolibri.

Protege також підтримує співпрацю та взаємодію. Це дозволяє декільком користувачам одночасно працювати над однією онтологією, полегшуючи спільну розробку онтології. Крім того, Protege забезпечує бездоганну інтеграцію з іншими інструментами та фреймворками за допомогою стандартних протоколів і форматів. Ця сумісність гарантує, що система ідентифікації морських об’єктів, розроблена з використанням фреймворку jcolibri, може використовувати онтологію, створену в Protege, та інтегруватися з іншими відповідними програмними компонентами.

Protege має велику та активну спільноту користувачів, яка сприяє його постійному вдосконаленню та надає цінні ресурси для підтримки та обміну знаннями. Спільнота Protege пропонує онлайн-форуми, списки розсилки та навчальні посібники, що дозволяє дослідникам і практикам обмінюватися ідеями, шукати допомоги та вивчати найкращі методи розробки онтології.

Таким чином, Protege є потужним інструментом розробки онтологій, який полегшує створення та підтримку онтологій для ідентифікації морських об’єктів. Його зручний інтерфейс, підтримка OWL, розширені можливості моделювання, функції аргументації, функції співпраці та взаємодію роблять його ідеальним вибором для вашої дипломної роботи та інтеграції з фреймворком jcolibri. Використовуючи Protege, можна ефективно визначити онтологію, отримати знання

про морські об’єкти та підвищити точність і ефективність системи ідентифікації морських об’єктів. До апаратних частин моніторингу виробниками створені додатки для перегляду статистики, проте зазвичай мають недолік – вони прив’язані до конкретного апаратного модулю збору статистики.

3.6 CBR

Case-based reasoning (CBR) — це методологія вирішення проблем, яка спирається на минулий досвід для вирішення нових проблем. Це форма штучного інтелекту, яка зосереджена на використанні знань, отриманих у попередніх випадках, для прийняття обґрунтованих рішень щодо нових ситуацій. CBR передбачає пошук, повторне використання та адаптацію раніше вирішених випадків для вирішення подібних проблем у майбутньому [10].

Основна ідея CBR полягає в тому, що подібні проблеми мають однакові рішення. Коли стикаються з новою проблемою, системи CBR шукають минулі випадки, схожі на поточну проблему на основі їхніх атрибутів, характеристик або інших відповідних особливостей. Ці справи, також відомі як прецеденти, містять знання та досвід, які можна застосувати до поточної ситуації.

Процес CBR зазвичай складається з кількох етапів. По-перше, система отримує відповідні випадки з бібліотеки випадків або бази даних на основі опису проблеми або запиту. Відновлені випадки потім оцінюються та ранжуються на основі їх подібності до поточної проблеми. Для адаптації відбираються найбільш схожі випадки.

Етап адаптації передбачає модифікацію отриманих випадків відповідно до поточної проблеми. Це може включати коригування значень атрибутів, застосування перетворень або об’єднання кількох випадків для створення рішення. Потім адаптоване рішення застосовується до поточної проблеми та оцінюється

його ефективність. Результат рішення використовується для оновлення бібліотеки випадків, гарантуючи, що нові знання фіксуються для майбутнього використання.

CBR має ряд переваг, які роблять його корисним підходом у різних областях, включаючи ідентифікацію морських об'єктів. По-перше, він використовує наявні знання та досвід, що дозволяє ефективно вирішувати проблеми. Замість того, щоб починати з нуля, системи CBR можуть швидко отримувати та адаптувати відповідні справи, заощаджуючи час і зусилля.

По-друге, CBR адаптивний і гнучкий. У міру додавання нових випадків до бібліотеки випадків база знань системи розширюється, дозволяючи їй обробляти ширший діапазон типів і варіантів проблем. Ця адаптивність є особливо цінною для ідентифікації морських об'єктів, де характеристики та атрибути об'єктів можуть сильно відрізнятися.

Крім того, CBR є прозорою та зрозумілою методологією. Процес міркування можна простежити до конкретних випадків, що полегшує розуміння та перевірку рішень, прийнятих системою. Ця прозорість є важливою в таких сферах, як ідентифікація морських об'єктів, де точність і надійність результатів є вирішальними.

Підсумовуючи, CBR є цінним підходом для ідентифікації морських об'єктів на основі онтології та прецедентів. Використовуючи минулий досвід і адаптуючи рішення до поточних проблем, системи CBR можуть ефективно ідентифікувати та класифікувати морські об'єкти. Однак слід ретельно розглянути пошук випадків, оцінку подібності та підтримувати оновлену та всебічну бібліотеку випадків, щоб максимально підвищити ефективність CBR у цій галузі.

3.7 OWL

OWL, що означає мову веб-онтології, є стандартизованою мовою, яка використовується для представлення та міркування про знання в машинному

форматі. Він розроблений спеціально для створення онтологій, які є формальними описами понять та їхніх зв’язків у певній області [11].

OWL базується на RDF (Resource Description Framework) Консорціуму Всесвітньої павутини та надає багатий набір конструкцій для визначення класів, властивостей та індивідів разом із їхніми характеристиками та зв’язками.

Однією з ключових особливостей OWL є його здатність представляти складні відносини між поняттями. OWL підтримує різні типи зв’язків, включаючи ієрархічні зв’язки, такі як зв’язки підкласів і суперкласів, а також складніші зв’язки, такі як обмеження властивостей і обмеження кількості елементів. Ця гнучкість дозволяє OWL фіксувати нюанси зв’язків, які існують у домені, забезпечуючи більш точні та змістовні представлення.

Іншим важливим аспектом OWL є підтримка логічних міркувань. Онтології OWL можна обробляти за допомогою автоматизованих механізмів міркування, які можуть виводити нові знання на основі визначених зв’язків і обмежень. Ця можливість аргументації дозволяє перевіряти узгодженість, класифікувати екземпляри в класи і навіть відкривати неявні знання, які можуть бути явно не викладені в онтології.

OWL також забезпечує підтримку анотування елементів онтології метаданими. Метадані можуть містити додаткову інформацію про елементи, таку як авторство, дата створення та версія. Ці метадані необхідні для управління та підтримки онтологій протягом тривалого часу, особливо в середовищах спільної роботи, де багато зацікавлених сторін можуть сприяти розвитку онтології.

У контексті ідентифікації морських об'єктів OWL може відігравати вирішальну роль. Створюючи онтологію за допомогою OWL, можна охопити різні типи морських об’єктів, їхні атрибути та зв’язки. Наприклад, онтологія може визначати класи для різних типів морських організмів, таких як риби, ссавці та рослини, разом із їхніми специфічними атрибутами, такими як розмір, колір та

середовище існування. Крім того, OWL може представляти відносини між цими об’єктами, такі як відносини хижак-жертва або симбіотичні асоціації.

Використання OWL у поєднанні з фреймворком jcolibri може покращити процес ідентифікації морських об’єктів. Фреймворк jcolibri використовує експресивну силу OWL для міркування про зібрані дані та надання значущих результатів. Використовуючи можливості міркування OWL, структура може отримати додаткову інформацію про ідентифіковані морські об’єкти на основі визначеної онтології. Це може включати висновок про відсутні атрибути, категоризацію об’єктів у більш конкретні класи або навіть ідентифікацію зв’язків, які не були явно надані у вхідних даних.

Підсумовуючи, OWL є потужною мовою для створення онтологій і міркувань про знання. Його здатність представляти складні зв’язки, підтримувати логічне мислення та надавати анотації метаданих робить його цінним інструментом для різних областей, включаючи ідентифікацію морських об’єктів. Використовуючи OWL у структурі jcolibri, дослідники можуть розширити свою здатність точно й ефективно ідентифікувати морські об’єкти, використовуючи багатство онтології та можливості міркування OWL.

3.8 GitHub

GitHub — це широко використовувана веб-платформа для контролю версій і співпраці над проектами розробки програмного забезпечення. Він надає розробникам централізоване місце для зберігання, керування та спільної роботи над своєю кодовою базою. GitHub використовує систему контролю версій Git, яка дозволяє розробникам відстежувати зміни у своєму коді та легко керувати різними версіями свого проекту [12].

Однією з ключових особливостей GitHub є його здатність сприяти співпраці між розробниками. Це дозволяє кільком розробникам працювати над одним

проектом одночасно, причому кожен розробник має свою власну гілку для роботи. Це дозволяє легко інтегрувати зміни та запобігає конфліктам між різними модифікаціями коду. GitHub також надає інструменти для перегляду коду, відстеження проблем і керування проектами, що робить його комплексною платформою для робочих процесів розробки програмного забезпечення.

Іншим важливим аспектом GitHub є його широкі соціальні функції. Розробники можуть стежити за іншими користувачами та проектами, позначати сховища зірочками та брати участь в обговореннях через проблеми та запити на вилучення. Це сприяє сильному почуттю спільноти та дозволяє розробникам вчитися та робити внесок у проекти з відкритим кодом. Платформа також пропонує такі функції, як GitHub Pages, що дозволяє користувачам розміщувати статичні веб-сайти безпосередньо зі своїх сховищ.

GitHub не обмежується окремими розробниками чи невеликими командами; багато великих організацій і підприємств також використовують GitHub для потреб розробки програмного забезпечення. Він надає функції для керування контролем доступу, дозволяючи організаціям визначати дозволи та ролі для членів своєї команди. Крім того, GitHub пропонує корпоративні рішення, які можна розгортати на приватних серверах, забезпечуючи додатковий захист і контроль для організацій, яким це потрібно.

Доступність величезної кількості проектів з відкритим кодом на GitHub робить його цінним ресурсом для розробників. Розробники можуть досліджувати існуючі проекти, вчитися на високоякісному коді та робити внесок у проекти з відкритим вихідним кодом, які їх цікавлять. GitHub також пропонує такі функції, як розгалуження та запити на отримання, які спрощують розробників вносити зміни назад у вихідний проект.

Підсумовуючи, GitHub є потужною та широко використовуваною платформою для контролю версій і співпраці в розробці програмного забезпечення. Його функції для керування кодом, співпраці та соціальної взаємодії роблять його

безцінним інструментом для окремих розробників, невеликих команд і великих організацій. Величезна екосистема проектів з відкритим кодом на GitHub ще більше підвищує його цінність як ресурсу, на якому розробники можуть вчитися та робити свій внесок.

Висновки до розділу 3

У третьому розділі дипломної роботи було проведено дослідження різних інструментів, які використовувалися під час розробки програмного продукту. Дослідження охоплювало комплексний аналіз різних ресурсів, платформ і фреймворків, які відіграли вирішальну роль у досягненні бажаних цілей системи ідентифікації морських об’єктів. Одним із основних використовуваних інструментів був фреймворк jcolibri, бібліотека на основі Java з відкритим вихідним кодом, яка полегшувала впровадження методів аргументації на основі випадків (CBR) для процесу ідентифікації. Використання jcolibri дозволило інтегрувати функції CBR, дозволяючи системі отримувати, адаптувати та застосовувати попередні випадки, що зберігаються в базі знань, для вирішення нових проблем ідентифікації морських об’єктів.

Крім того, опис стосувався використання інших відповідних технологій і платформ, таких як Java, Eclipse, Protege і OWL. Java, будучи універсальною мовою програмування, забезпечила основу для розробки програмного продукту, пропонуючи надійність, сумісність між платформами та багату екосистему бібліотек і фреймворків. Eclipse, популярне інтегроване середовище розробки (IDE), слугувало основною платформою для програмування та налагодження, підвищуючи продуктивність і сприяючи ефективній співпраці серед команди розробників.

Крім того, Protege, інструмент розробки та редагування онтології, використовувався для створення та керування онтологією морських об’єктів,

дозволяючи системі визначати та представляти концепції, зв’язки та атрибути, пов’язані з різними типами морських об’єктів. Формат OWL (мова веб-онтології) використовувався разом із Protege, щоб виразити онтологію машиночитаним і семантично насиченим способом. Використовуючи ці інструменти в поєднанні з jcolibri та Swing, інструментарієм Java GUI, програмний продукт зміг надати точні результати ідентифікації морських об’єктів, охоплюючи детальну інформацію, таку як ім’я, тип, атрибути та метадані, тим самим сприяючи комплексному аналізу та ухваленню рішень. створення в морських сферах.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі буде описано реалізацію системи ідентифікації морських об’єктів на основі онтології та прецедентів. Додаток використовує структуру jcolibri для полегшення процесу ідентифікації, надаючи вичерпні результати, включаючи назву, тип, атрибути та метадані морських об’єктів.

Система використовує передові алгоритми, спеціально розроблені для ідентифікації морських об'єктів. Ці алгоритми використовують потужність машинного навчання, розпізнавання образів і методи семантичного мислення. Вони ретельно розроблені та налаштовані для точного розпізнавання та класифікації різних морських об’єктів на основі їхніх характеристик та контекстної інформації.

Реалізація системи ідентифікації морських об’єктів також включає такі важливі аспекти, як структура проекту, вибір моделі проекту та методи тестування. Структура проекту ретельно розроблена для забезпечення модульності, масштабованості та зручності обслуговування програмного забезпечення.

Продуманий аналіз структури проекту, вибору моделі проекту та методів тестування забезпечує розробку якісного та надійного програмного продукту.

4.1 Use Case Diagram

Діаграма варіантів використання служить основоположним інструментом у розробці програмного забезпечення та системному аналізі. Це відіграє ключову роль у захопленні та передачі основних функцій і взаємодій системи. Візуально представляючи учасників, випадки використання та їхні взаємозв’язки, діаграма сприяє спільному розумінню між зацікавленими сторонами та допомагає визначити вимоги до системи. Це також допомагає виявити потенційні прогалини або області для вдосконалення процесу ідентифікації, сприяючи більш комплексній

та ефективній системі. Крім того, діаграма варіантів використання заохочує співпрацю та обговорення між зацікавленими сторонами, уможливлюючи більш ітеративний і вдосконалений підхід до розробки та оцінки програми ідентифікації морських об’єктів. На рисунку 4.1 представлена діаграма прецедентів системи.

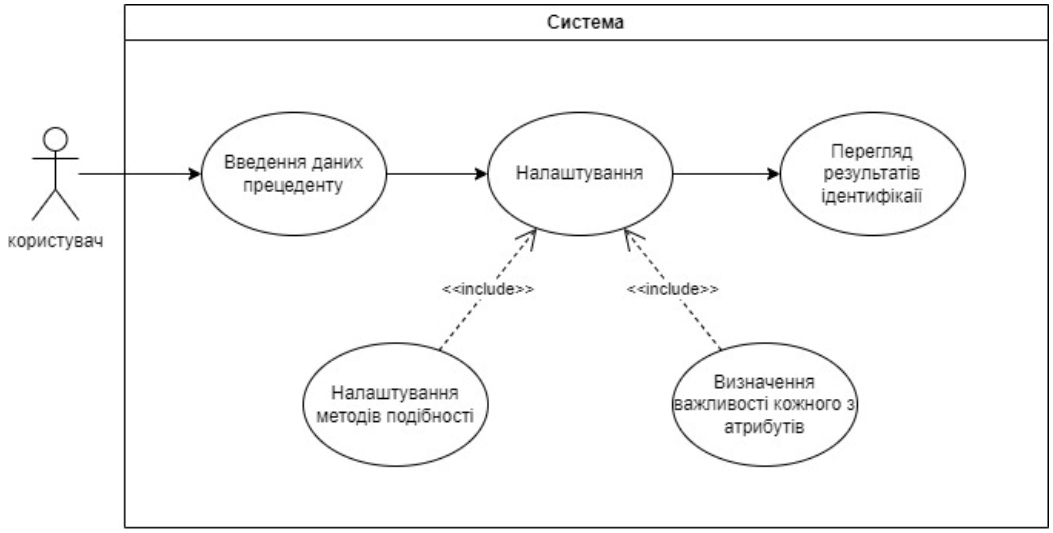

Рисунок 4.1 – Діаграма прецедентів

Діаграма прецедентів описує можливі варіанти взаємодії користувача із програмним продуктом. Актором є користувач, який представляє особу, яка використовує програму ідентифікації морських об’єктів.

Випадки використання:

- введення даних прецеденту, такі як спосіб отримання даних, географічні координати, та інше;
- налаштування параметрів ідентифікації, що включає в себе налаштування та вибір методів подібності по кожному параметру, а також визначення важливості кожного з атрибутів;
- перегляд результатів ідентифікації, а саме перегляд списку найбільш схожих прецедентів, що наявні в базі, а також перегляд значення схожості, що може мати значення від 0 (не схоже зовсім) до 1 (повне співпадіння).

4.2 Data Flow Diagram

Діаграма потоку даних представляє потік даних та інформації в процесі ідентифікації морських об’єктів, який використовує структуру jcolibri. Він ілюструє, як різні компоненти взаємодіють і обмінюються даними для точної ідентифікації морських об’єктів. Нижче (рис 4.2) приведена діаграма потоку даних.

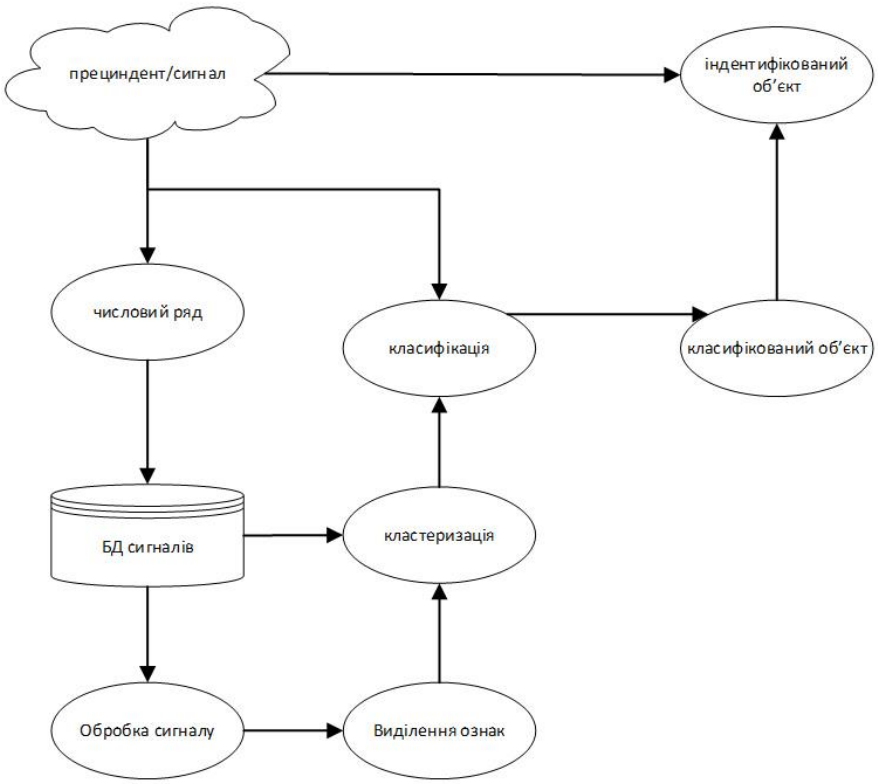

Рисунок 4.2 — Data flow diagram

Наведена діаграма потоку даних зображує потік даних і процеси, пов’язані з ідентифікацією морських об’єктів за допомогою онтології та прецедентів. Діаграма потоку даних ілюструє як попередні етапи, через які дані проходять перед тим, як досягти даної роботи і зможуть використовуватись як вхідні дані, так і етапи, які реалізовані та описані в даній роботі, зосереджуючись на кластеризації, класифікації та ідентифікації об’єктів.

Початковий крок передбачає отримання прецедентів або сигналів, які можуть бути різними формами даних, що стосуються морських об’єктів. Потім ці сигнали обробляються та перетворюються на числові послідовності, що дозволяє проводити подальший аналіз. Крім того, сигнали також класифікуються на основі їхніх характеристик і атрибутів.

Числові послідовності та класифіковані сигнали зберігаються в базі даних, призначеній для зберігання сигналів. Ця база даних служить сховищем для сигналів, полегшуючи їх пошук і подальшу обробку. Обробка сигналів включає різні етапи, такі як обробка сигналів, виділення ознак і кластеризація.

Обробка сигналів передбачає маніпуляції та перетворення сигналів для покращення їх якості або вилучення певної інформації. Виділення ознак виконується для ідентифікації та виділення відповідних атрибутів або функцій із сигналів. Витягнуті ознаки потім використовуються в процесі кластеризації, де подібні сигнали групуються разом на основі їх подібності або шаблонів і можуть використовуватися для даної роботи.

Процес кластеризації покладається на витягнуті функції та базу даних сигналів для кластеризації подібних сигналів у окремі групи. Цей крок допомагає визначити шаблони та схожість між сигналами, що дозволяє краще класифікувати морські об’єкти. Результати процесу кластеризації потім використовуються на наступному етапі класифікації.

Класифікація передбачає присвоєння міток або категорій сигналам на основі їхніх характеристик і подібності. Результати кластеризації в поєднанні з класифікованими сигналами з попереднього етапу сприяють більш точному та вдосконаленому процесу класифікації. Остаточним результатом етапу класифікації є класифіковані морські об’єкти.

Нарешті, класифіковані морські об’єкти додатково обробляються для ідентифікації конкретних об’єктів, які вони представляють. На цьому етапі

ідентифікації враховуються назва, тип, атрибути та метадані, пов’язані з морськими об’єктами, що призводить до ідентифікації самих об’єктів.

Загалом, ця діаграма потоку даних забезпечує візуальне представлення всього процесу, наголошуючи на найважливіших етапах кластеризації, класифікації та ідентифікації в контексті аналізу морських об’єктів із використанням онтології та прецедентів.

4.3 Онтологія

Щоб створити онтологію в Protege, було зроблено наступні кроки. По-перше, було проведено ретельний аналіз сфери морських об’єктів, щоб визначити ключові поняття, зв’язки та атрибути.

Після завершення аналізу ідентифіковані поняття та їхні зв’язки були створенні класи для представлення різних типів морських об’єктів, а також визначення їхніх властивостей і атрибутів. Результат показано нижче (рис 4.3).

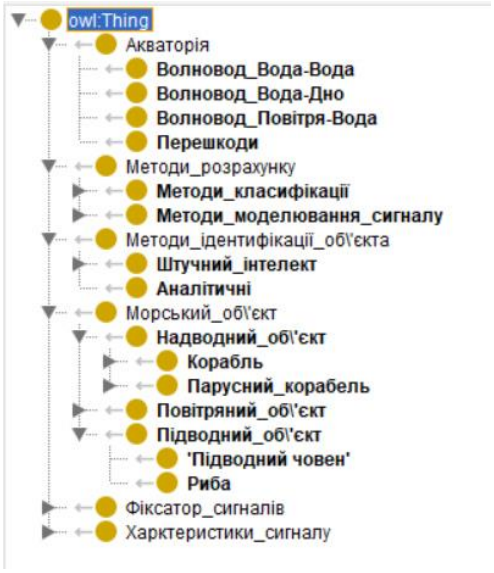

Рисунок 4.3 — Основні класи онтології

Відносини між цими класами були встановлені за допомогою властивостей об’єктів, таких як зв’язки «є» та «частина».

Нижче показана онтологія в схематичному зображені (рис 4.4), а також приклади зв’язків між об’єктами.

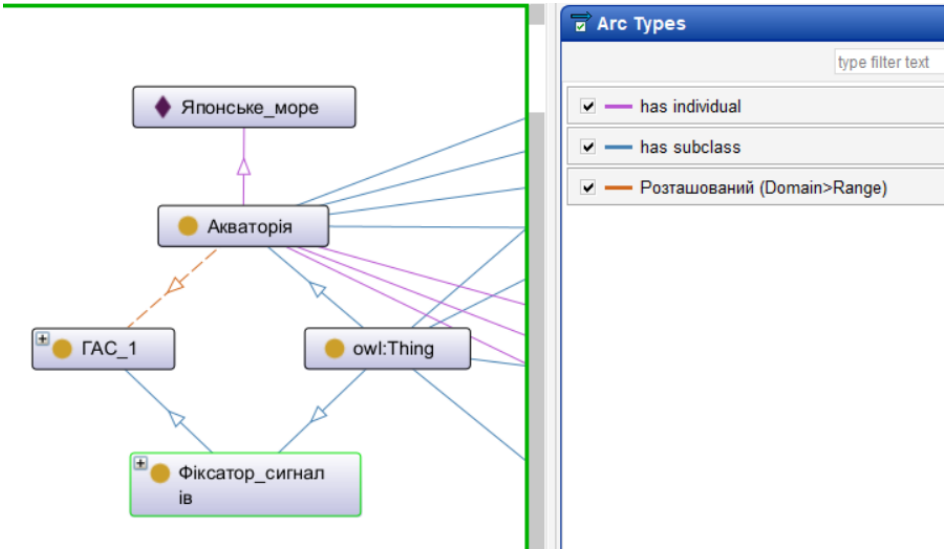

Рисунок 4.4 — Зв’язки між класами та об’єктами

Також були створені конкретні об’єкти, визначено їхній клас та відношення до інших об’єктів.

4.4 Архітектура CBR

CBR – система міркування на основі випадків, що використовує накопичений досвід для вирішення нових проблем або прийняття рішень. Її основна ідея полягає в тому, що для розв'язання поточного завдання використовуються раніше вирішені подібні задачі. Вона складається з кількох ключових компонентів. Спочатку створюється база даних випадків, яка зберігає інформацію про раніше вирішені задачі. Кожен випадок містить опис проблеми, рішення, що було вжите, та контекст, у якому це рішення було прийняте. Далі є механізм виявлення схожості, який порівнює поточне завдання з випадками в базі даних. Використовуються методи, такі як відстань Хеммінга, косинусна схожість або метод найближчих сусідів. Після виявлення схожих випадків відбувається передостаннім етапом є

адаптація. Знання зі схожих випадків переносяться на поточну ситуацію шляхом зміни параметрів чи додавання нових умов. Останнім етапом є оцінка та валідація результату, де оцінюється якість запропонованого рішення та його придатність для поточної ситуації. Задовільний результат приймається, і рішення може бути виконане, в іншому випадку можуть використовуватися інші методи міркування. В основі CBR системи в даній роботі лежить фреймворк JCOLIBRI.

На рисунку 4.5 зображено архітектура фреймворку JCOLIBRI.

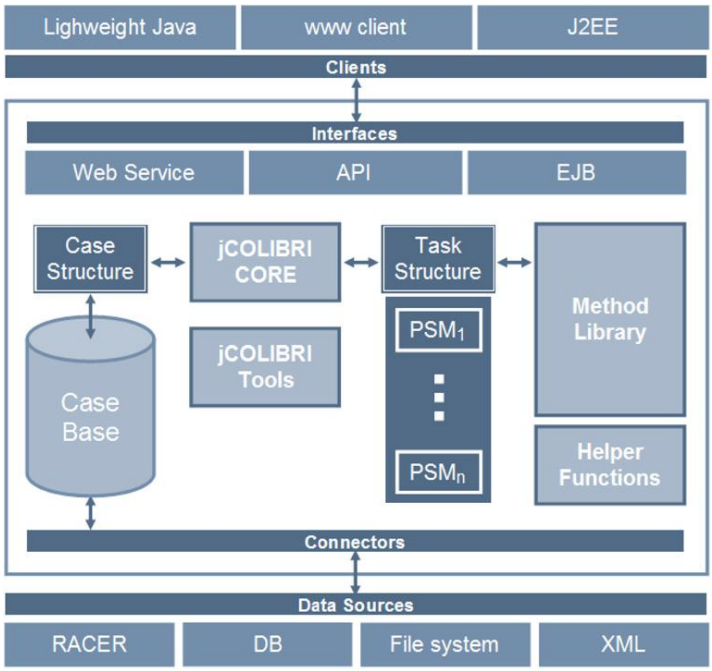

Рисунок 4.5 — Архітектура JCOLIBRI

Нижче описані модулі цієї системи [10].

Case base, data sources and connectors – системи на основі прецедентів (CBR) повинні мати ефективний доступ до збережених прецедентів, що стає все більш актуальною проблемою зі збільшенням розміру бази прецедентів. jCOLIBRI розбиває проблему управління базою прецедентів на два окремі, але пов'язані аспекти: механізм постійного збереження за допомогою конекторів та організацію в оперативній пам'яті. Надаються різні конектори та структури даних для організації в оперативній пам'яті.

Case structure and similarity measures – залежно від системи, прецеденти можуть бути представлені як прості випадки з атрибутами та значеннями, текстові випадки або складні ієрархічні (об'єктно-орієнтовані) структури, де атрибути пов'язані між собою. В залежності від структури прецедентів можуть застосовуватися різні функції схожості для порівняння атрибутів прецедентів.

Tasks structure and problem-solving methods (PSMs) – У програмному застосунку на основі CBR в jCOLIBRI визначено, що він складається з послідовності завдань, які потребують розв'язання за допомогою методу розкладання або декомпозиції. Декомпозиційні методи розбивають кожне завдання на набір підзавдань. Наприклад, до завдань можуть відноситися попередня обробка прецедентів, отримання запиту, пошук, використання, коригування, збереження, обчислення схожості та інші. Для кожного завдання CBR-дизайнер налаштовує метод, який його вирішує. Наш підхід до специфікації компетенції PSM (післяумови) та вимог (передумови) використовує онтології і має два основні переваги. По-перше, він дозволяє формальні специфікації, які надають точне значення та підтримку розуміння. По-друге, цей підхід забезпечує значні переваги щодо повторного використання, оскільки онтології завдань і методів можуть бути спільно використані різними системами.

Method library and helper functions – у бібліотеці методів включено рівень реалізації (тобто рівень виконання) методів розв'язання проблеми на основі знань. У разі відсутності потрібного методу розв'язання у бібліотеці, дизайнер системи CBR повинен створити Java-код. Допоміжні функції надають підтримку для розробки нових методів та функцій схожості.

Core and tools – роль ядра фреймворку є важливою в процесі проектування, оскільки воно дозволяє інтерактивно тестувати систему, яка має частково налаштований функціонал. Це означає, що під час розробки додатку на базі фреймворку, коли окремі компоненти або завдання ще не повністю налаштовані, розробники можуть взаємодіяти з системою та проводити тестування для перевірки

його функціональності та правильності роботи. Це дозволяє виявляти та виправляти помилки, вдосконалювати функціональні можливості та забезпечувати ефективний процес розробки додатку на основі фреймворку

Висновки до розділу 4

У четвертому розділі було розглянуто питання архітектури системи, вирішено завдання з вибору архітектури для всієї системи, описано взаємодію окремих компонентів системи, створено та описано діаграму прецедентів, описано онтологію, та основні етапи обробки даних.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для того, щоб протестувати програмний застосунок потрібно запустити програму, після чого відкриється вікно для запиту (рис 5.1).

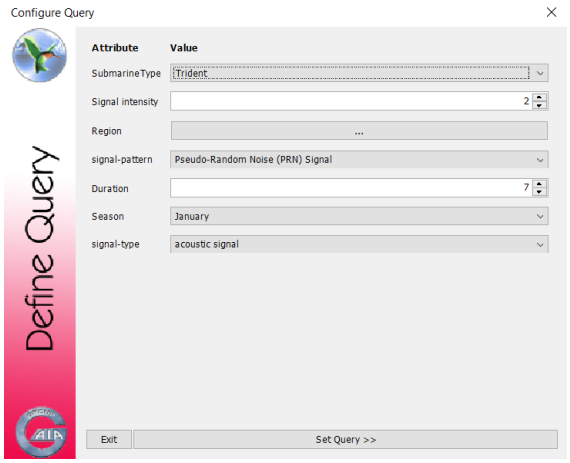

Рисунок 5.1 — Меню запиту

Після цього потрібно ввести данні у поля, та натиснути кнопку “Set Query”, після чого відкриється вікно налаштування запиту (рис 5.2).

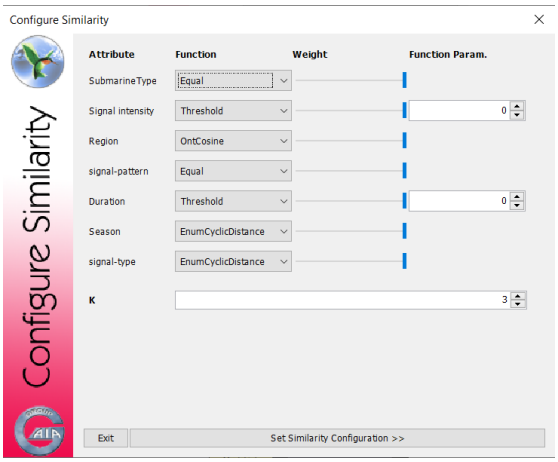

Рисунок 5.2 — Меню налаштування запиту

Далі необхідно вибрати метод схожості, за яким система буде шукати подібні випадки. Наприклад можна вибрати метод «Equal» що буде просто порівнювати

значення, та видавати «True» або «False» в залежності від того, чи однакове значення у введеному прецеденті, та шуканому. Також можна налаштувати ваги кожного атрибуту для врахування важливості кожної характеристики. Чим більш важливим є атрибут, тим правіше треба перетягти повзунок, і навпаки, чим менш важливим є артибут, тим правіше треба перенести повзунок. Деякі методи схожості мають додатковий параметр, саме його і треба вводити в додаткове поле «function param».

Після того як всі налаштування будуть введені, для наступного пункту треба натиснути на кнопку «Set Similarity Configuration», після чого відкриється вікно з результатами пошуку (рис. 5.3).

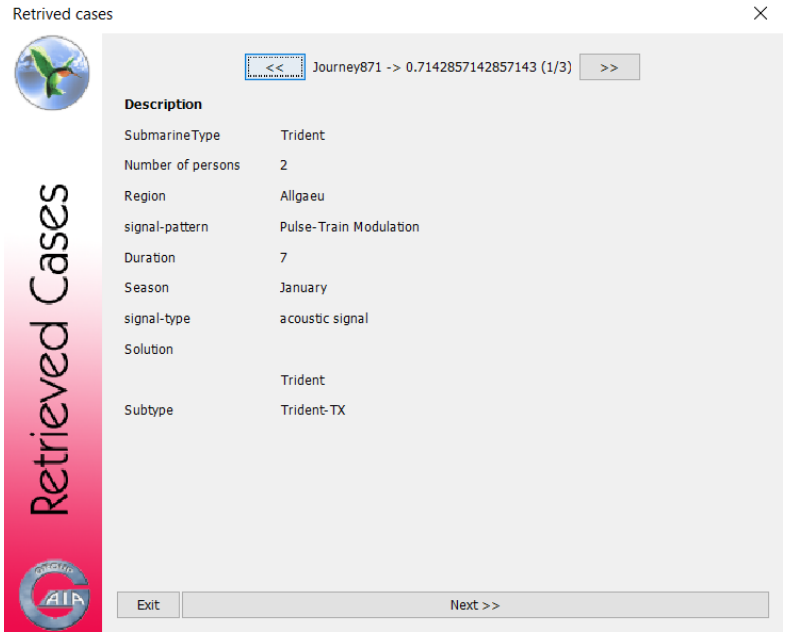

Рисунок 5.3 — Приклад результату запиту

Основний результат знаходиться навпроти поля «Subtype». Додаткова інформація по об’єкту знаходиться вище.

Система також може знаходити одразу декілька схожих випадків. Зверху є число, що показує схожість об’єкта. Максимальне значення е одиниця, що означає що об’єкт повністю збігається, а мінімальне значення це нуль, що каже про те, що об’єкт немає нічого спільного з даним прецедентом.

Для того щоб побачити інші схожі випадки зверху є 2 кнопки (рис. 5.4), за допомогою яких можна натискаючи переглянути інші схожі випадки які знайшла CBR система.

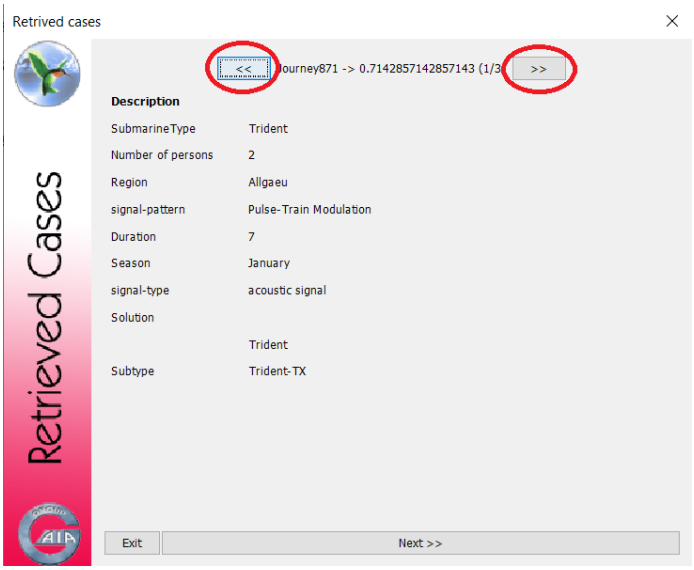

Рисунок 5.4 — Кнопки для перегляду інших схожих випадків

Для того щоб закрити застосунок слід натиснути кнопку «Exit» (рис. 5.5).

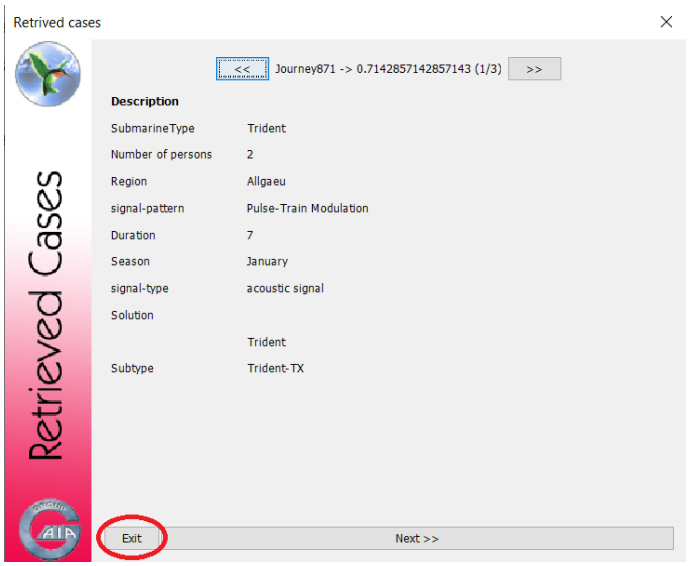

Рисунок 5.5 — Кнопка для закриття застосунку

Таким чином можна виконувати запити на знаходження випадків по заданому прецеденту та заданими параметрами пошуку конфігурації.

Висновки до розділу 5

У п’ятому розділі було розглянуто процес взаємодії користувача та системи, основні елементи графічного інтерфейсу, та важливі їх принципи. Взаємодія користувача та системи включає в себе різні етапи, такі як введення даних, обробка інформації, відображення результатів та зворотній зв'язок з користувачем. Основні елементи графічного інтерфейсу включають в себе вікна, кнопки, прапорці, поля введення, списки, меню та інші елементи, які допомагають користувачу взаємодіяти з системою.

ВИСНОВКИ

У ході виконання дипломної роботи увага була спрямована на усунення обмежень існуючих систем ідентифікації морських об'єктів та розробку нової та ефективної системи ідентифікації. Шляхом ретельного аналізу були виявлені сильні та слабкі сторони поточних систем, джерел даних та програмного забезпечення, що дозволило сформулювати конкретні вимоги до нової системи: ефективність, точність та простоту використання.

Запропонована система ідентифікації була розроблена з індивідуальною архітектурою, що використовує принципи онтології та прецедентів, забезпечуючи доступність, простоту використання та повну інтеграцію.

Крім того, був проведений докладний аналіз доступних інструментів, бібліотек та платформ, що дозволило вибрати відповідні компоненти для розробки системи. Ці компоненти включають функціональності, такі як інтеграція даних, алгоритми машинного навчання та міркування на основі онтології, що є важливими для точної та повної ідентифікації морських об'єктів.

Розроблений програмний додаток використовує алгоритми для аналізу даних, розпізнавання патернів та призначення категорій на основі визначеної онтології. Крім того, система використовує міркування на основі випадкових ситуацій для постійного навчання на основі минулого досвіду, що покращує точність ідентифікації

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lamy JB. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. Artificial Intelligence In Medicine 2017;80:11-28 (дата звернення: 25.05.2023)
2. Topbraid Composer: веб-сайт. URL: https://allegrograph.com/topbraid-composer/ (дата звернення: 24.05.2023)
3. myCBR: веб-сайт. URL: http://mycbr-project.org/ (дата звернення: 24.05.2023)
4. jColibri.NET: веб-сайт. URL: https://www.onworks.net/ru/software/windows/app-jcolibri-cbr-framework (дата звернення: 24.05.2023)
5. Java Documentation: веб-сайт. URL: https://docs.oracle.com/en/java/ (дата звернення: 24.05.2023)
6. Eclipse Documentation: веб-сайт. URL: https://www.eclipse.org/ (дата звернення: 24.05.2023)
7. JCOLIBRI Documentation: веб-сайт. URL: https://gaia.fdi.ucm.es/research/colibri/jcolibri/ (дата звернення: 25.05.2023)
8. SWING Documentation: веб-сайт. URL: https://devdocs.io/openjdk~8_gui/ (дата звернення: 24.05.2023)
9. Protege Documentation: веб-сайт. URL: https://protege.stanford.edu/ (дата звернення: 25.05.2023)
10. Building CBR systems with jCOLIBRI☆, 16 October 2007. – С. 70–71.
11. OWL: веб-сайт. URL: https://www.w3.org/TR/2004/REC-owl-semantics-20040210/ (дата звернення: 24.05.2023)
12. GitHub: веб-сайт. URL: https://docs.github.com/en/communities/documenting-your-project-with-wikis/about-wikis (дата звернення: 24.05.2023)

ДОДАТОК А

Програмний застосунок ідентифікації морських об'єктів на основі онтології і прецедентів

Програмний код

Аркушів 8

2023

```
package jcolibri.examples.Recommender;

import java.awt.Dimension;
import java.util.ArrayList;
import java.util.Collection;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

import jcolibri.casebase.LinealCaseBase;
import jcolibri.cbraplications.StandardCBRApplication;
import jcolibri.cbrcore.Attribute;
import jcolibri.cbrcore.CBRCase;
import jcolibri.cbrcore.CBRCaseBase;
import jcolibri.cbrcore.CBRQuery;
import jcolibri.cbrcore.Connector;
import jcolibri.connector.DataBaseConnector;
import jcolibri.examples.Recommender.gui.AutoAdaptationDialog;
import jcolibri.examples.Recommender.gui.QueryDialog;
import jcolibri.examples.Recommender.gui.ResultDialog;
import jcolibri.examples.Recommender.gui.RetainDialog;
import jcolibri.examples.Recommender.gui.RevisionDialog;
import jcolibri.examples.Recommender.gui.SimilarityDialog;
import jcolibri.exception.ExecutionException;
import jcolibri.method.retrieve.RetrievalResult;
import jcolibri.method.retrieve.NNretrieval.NNConfig;
import jcolibri.method.retrieve.NNretrieval.NNScoringMethod;
import jcolibri.method.retrieve.NNretrieval.similarity.global.Average;
import jcolibri.method.retrieve.selection.SelectCases;
import jcolibri.method.reuse.NumericDirectProportionMethod;
import jcolibri.util.FileIO;
import es.ucm.fdi.gaia.ontobridge.OntoBridge;
import es.ucm.fdi.gaia.ontobridge.OntologyDocument;


public class Recommender implements StandardCBRApplication {

  private static jcolibri.examples.Recommender.Recommender _instance = null;
  public  static jcolibri.examples.Recommender.Recommender getInstance()
  {
    if(_instance == null)
      _instance = new jcolibri.examples.Recommender.Recommender();
    return _instance;
  }

  private Recommender()
  {
  }

  /** Connector object */
  Connector _connector;
  /** CaseBase object */
  CBRCaseBase _caseBase;

  SimilarityDialog similarityDialog;
  ResultDialog resultDialog;
  AutoAdaptationDialog autoAdaptDialog;
```

```
RevisionDialog revisionDialog;
RetainDialog retainDialog;


public void configure() throws ExecutionException {
  try {
    //Emulate data base server
    jcolibri.test.database.HSQLDBserver.init();

    // Create a data base connector
    _connector = new DataBaseConnector();
    // Init the ddbb connector with the config file
    _connector.initFromXMLfile(jcolibri.util.FileIO
        .findFile("jcolibri/examples/Recommender/databaseconfig.xml"));
    // Create a Lineal case base for in-memory organization
    _caseBase = new LinealCaseBase();

    // Obtain a reference to OntoBridge
    OntoBridge ob = jcolibri.util.OntoBridgeSingleton.getOntoBridge();
    // Configure it to work with the Pellet reasoner
    ob.initWithPelletReasoner();
    // Setup the main ontology
    OntologyDocument mainOnto = new OntologyDocument("http://gaia.fdi.ucm.es/ontologies/-destinations.owl",
            FileIO.findFile("jcolibri/examples/Recommender/destinations.owl").toExternalForm());
    // There are not subontologies
    ArrayList<OntologyDocument> subOntologies = new ArrayList<OntologyDocument>();
    // Load the ontology
    ob.loadOntology(mainOnto, subOntologies, false);

    // Create the dialogs
    similarityDialog = new SimilarityDialog(main);
    resultDialog     = new ResultDialog(main);
    autoAdaptDialog  = new AutoAdaptationDialog(main);
    revisionDialog   = new RevisionDialog(main);
    retainDialog     = new RetainDialog(main);

  } catch (Exception e) {
    throw new ExecutionException(e);
  }
}

public CBRCaseBase preCycle() throws ExecutionException {
  // Load cases from connector into the case base
  _caseBase.init(_connector);
  // Print the cases
  java.util.Collection<CBRCase> cases = _caseBase.getCases();
  for(CBRCase c: cases)
    System.out.println(c);
  return _caseBase;
}

public void cycle(CBRQuery query) throws ExecutionException {
  // Obtain configuration for KNN
  similarityDialog.setVisible(true);
  NNConfig simConfig = similarityDialog.getSimilarityConfig();
  simConfig.setDescriptionSimFunction(new Average());

  // Execute NN
```

```
  Collection<RetrievalResult> eval = NNScoringMethod.evaluateSimilarity(_caseBase.getCases(), query, simConfig);

  // Select k cases
  Collection<CBRCase> selectedcases = SelectCases.selectTopK(eval, similarityDialog.getK());

  // Show result
  resultDialog.showCases(eval, selectedcases);
  resultDialog.setVisible(true);

  // Show adaptation dialog
  autoAdaptDialog.setVisible(true);

  // Adapt depending on user selection
  if(autoAdaptDialog.adapt_Duration_ampl())
  {
    // Compute a direct proportion between the "Duration" and "ampl" attributes.
    NumericDirectProportionMethod.directProportion(    new Attribute("Duration",Description.class),
                          new Attribute("ampl",Solution.class),
                          query, selectedcases);
  }

  if(autoAdaptDialog.adapt_NumberOfPersons_ampl())
  {
    // Compute a direct proportion between the "Duration" and "ampl" attributes.
    NumericDirectProportionMethod.directProportion(    new Attribute("NumberOfPersons",Description.class),
                          new Attribute("ampl",Solution.class),
                          query, selectedcases);
  }

  // Revise
  revisionDialog.showCases(selectedcases);
  revisionDialog.setVisible(true);

  // Retain
  retainDialog.showCases(selectedcases, _caseBase.getCases().size());
  retainDialog.setVisible(true);
  Collection<CBRCase> casesToRetain = retainDialog.getCasestoRetain();
  _caseBase.learnCases(casesToRetain);

}

public void postCycle() throws ExecutionException {
  _connector.close();
  jcolibri.test.database.HSQLDBserver.shutDown();
}

static JFrame main;
void showMainFrame()
{
  main = new JFrame(" Recommender");
  main.setResizable(false);
  main.setUndecorated(true);
  JLabel label = new JLabel(new ImageIcon(jcolibri.util.FileIO.findFile("/jcolibri/test/main/jcolibri2.jpg")));
  main.getContentPane().add(label);
  main.pack();
  Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
  main.setBounds((screenSize.width - main.getWidth()) / 2,
    (screenSize.height - main.getHeight()) / 2,
```

```
      main.getWidth(),
      main.getHeight());
    main.setVisible(true);
  }

  public static void main(String[] args) {

    jcolibri.examples.Recommender.Recommender recommender = getInstance();
    recommender.showMainFrame();
    try
    {
      recommender.configure();
      recommender.preCycle();

      QueryDialog qf = new QueryDialog(main);


      boolean cont = true;
      while(cont)
      {
        qf.setVisible(true);
        CBRQuery query = qf.getQuery();

        recommender.cycle(query);
        int ans = javax.swing.JOptionPane.showConfirmDialog(null, "CBR cycle finished, query again?", "Cycle finished",
javax.swing.JOptionPane.YES_NO_OPTION);
        cont = (ans == javax.swing.JOptionPane.YES_OPTION);
      }
      recommender.postCycle();
    }catch(Exception e)
    {
      org.apache.commons.logging.LogFactory.getLog(jcolibri.examples.Recommender.Recommender.class).error(e);
      javax.swing.JOptionPane.showMessageDialog(null, e.getMessage());
    }
    System.exit(0);
  }
}


package jcolibri.examples.Recommender;

import jcolibri.cbrcore.Attribute;

public class Solution implements jcolibri.cbrcore.CaseComponent {

  String id;
  Integer ampl;
  String submarineType;

  public String toString()
  {
    return "("+id+";"+ampl+";"+submarineType+")";
  }

  public Attribute getIdAttribute() {

    return new Attribute("id", this.getClass());
  }
```

```
  /**
   * @return Returns the subType.
   */
  public String getType() {
    return submarineType;
  }

  /**
   * @param subType The subType to set.
   */
  public void setType (String subType) {
    this.submarineType = subType;
  }

  /**
   * @return Returns the id.
   */
  public String getId() {
    return id;
  }

  /**
   * @param id The id to set.
   */
  public void setId(String id) {
    this.id = id;
  }

  /**
   * @return Returns the ampl.
   */
  public Integer getAmpl() {
    return ampl;
  }

  /**
   * @param ampl The ampl to set.
   */
  public void setAmpl(Integer ampl) {
    this.ampl = ampl;
  }

}

package jcolibri.examples.Recommender.gui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Collection;

import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
```

```
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SpringLayout;
import javax.swing.UIManager;

import jcolibri.cbrcore.CBRCase;
import jcolibri.cbrcore.CBRQuery;
import jcolibri.method.retrieve.RetrievalResult;
import jcolibri.util.FileIO;


public class AutoAdaptationDialog extends JDialog {

   private static final long serialVersionUID = 1L;

   JLabel image;

   JCheckBox np_ampl;
   JCheckBox duration_ampl;


   ArrayList<RetrievalResult> cases;
   int currentCase;
   Collection<CBRCase> selectedcases;
   CBRQuery query;

   public AutoAdaptationDialog(JFrame main) {
      super(main, true);
      configureFrame();
   }

   private void configureFrame() {
      try {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      } catch (Exception e1) {
      }

      this.setTitle("Adaptation");


      image = new JLabel();
      image.setIcon(new ImageIcon(FileIO.findFile("jcolibri/examples/Recommender/gui/step4.png")));
      this.getContentPane().setLayout(new BorderLayout());
      this.getContentPane().add(image, BorderLayout.WEST);


      /*************************************************************/
      JPanel panel = new JPanel();
      panel.setLayout(new SpringLayout());

      panel.add(np_ampl = new JCheckBox("Direct proportion between \"Number of Persons\" and \"ampl"));
      panel.add(duration_ampl = new JCheckBox("Direct proportion between \"Duration\" and \"ampl"));

      Utils.makeCompactGrid(panel,
            2, 1, //rows, cols
            20, 20,       //initX, initY
```

```
            30, 10);      //xPad, yPad

        JPanel panelAux = new JPanel();
        panelAux.setLayout(new BorderLayout());
        panelAux.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        panelAux.add(panel, BorderLayout.NORTH);

        JPanel buttons = new JPanel();
        buttons.setLayout(new BorderLayout());

        JButton ok = new JButton("Adapt Cases using Direct Proportions >>");
        ok.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              next();
           }
        });
        buttons.add(ok, BorderLayout.CENTER);
        JButton exit = new JButton("Exit");
        exit.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              try {
                 jcolibri.examples.Recommender.Recommender.getInstance().postCycle();
              } catch (Exception ex) {

org.apache.commons.logging.LogFactory.getLog(jcolibri.examples.Recommender.Recommender.class).error(ex);
              }
              System.exit(-1);
           }
        });
        buttons.add(exit, BorderLayout.WEST);

        panelAux.add(buttons, BorderLayout.SOUTH);
        this.getContentPane().add(panelAux, BorderLayout.CENTER);

        /*************************************************************/


        this.pack();
        this.setSize(600, this.getHeight());
        this.setResizable(false);
        Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width - this.getWidth()) / 2,
              (screenSize.height - this.getHeight()) / 2,
              getWidth(),
              getHeight());
    }

    void next() {
        this.setVisible(false);
    }


    public boolean adapt_NumberOfPersons_ampl() {
        return this.np_ampl.isSelected();
    }

    public boolean adapt_Duration_ampl() {
```

```
        return this.duration_ampl.isSelected();
    }


    /**
     * @param args
     */
    public static void main(String[] args) {
        AutoAdaptationDialog qf = new AutoAdaptationDialog(null);
        qf.setVisible(true);
        System.out.println("Bye");
    }


}
```