Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет ______________Комп'ютерних наук______________________
(повна назва)
Кафедра ____________Інформаційних_управляючих_систем__________
(повна назва)

# АТЕСТАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти _____________другий (магістерський)________________

_______Дослідження моделей і методів адаптації інформаційного та________
_____програмного забезпечення інформаційної системи до нових вимог_____
(тема)

Виконав:
студент __2_ курсу, групи __ІУСТм-19-1___
_________Асмоловському Б.І.______________
(прізвище, ініціали)

Спеціальність 122 – Комп'ютерні науки____
______________________________________
(код і повна назва спеціальності)
Тип програми ___освітньо-професійна____
(освітньо-професійна або освітньо-наукова)
Освітня програма _____Інформаційні_____
__управляючі системи та_технології______
(повна назва освітньої програми)

Керівник ______________________________
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ____________ ____________________
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет_____________________Комп'ютерних наук____________________
(повна назва)

Кафедра_________________Інформаційних_управляючих_систем_________
(повна назва)

Рівень вищої освіти _____________другий (магістерський)________________

Спеціальність _______________122 – Комп’ютерні науки_________________
(код і повна назва)

Тип програми _________________освітньо-професійна___________________
(освітньо-професійна або освітньо-наукова)

Освітня програма _____Інформаційні_управляючі системи та_технології_____
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____________
(підпис)
«____»_____________ 2020 р.

# ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові____________Асмоловському Богдану Ігоровичу_______________
(прізвище, ім’я, по батькові)

1. Тема роботи _ Дослідження моделей і методів адаптації інформаційного та програмного забезпечення інформаційної системи до нових вимог________________________________
_________________________________________________________________

затверджена наказом університету від _____ ______________20__ р. № ______

2. Термін подання студентом роботи до екзаменаційної комісії ____ __________ 20__ р.

3. Вихідні дані до роботи методи відображення онтологій; сучасні принципи мета______ модулювання_____________ _____ _______________________________________________ _____
_________________________________________________________________
_________________________________________________________________
_________________________________________________________________
_________________________________________________________________
_________________________________________________________________

4. Перелік питань, що потрібно опрацювати в роботі Аналіз проблеми повторного______ використання програмного коду; Розробка моделей повторного використання _________ програмного коду; Розробка методів повторного використання програмного коду;_______ Практична апробація отриманих наукових результатів_______________________________
_________________________________________________________________
_________________________________________________________________
_________________________________________________________________
_________________________________________________________________
_________________________________________________________________

**КАЛЕНДАРНИЙ ПЛАН**

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на дипломне | 10,09,2020 | |
| 2 | Аналіз завдання, літератури та аналогів з теми дипломної роботи | 11.09.2020-11.10.2020 | |
| | Постановка задачі | 12.10.2020 | |
| | Аналіз проблеми повторного використання програмного коду | 13.10.2020-18.10.2020 | |
| | Розробка моделей повторного використання програмного коду | 18.10.2020-24.10.2020 | |
| | Розробка методів повторного використання програмного коду | 25.10.2020-08.11.2020 | |
| | Практична апробація отриманих наукових результатів | 09.11.2020-22.11.2020 | |
| | Оформлення пояснювальної записки | 23.11.2020-03.12.2020 | |
| | Оформлення графічної частини та презентаційних матеріалів захисту | 13.12.2020 | |
| | Представлення на рецензування | 14.12.2020 | |
| | Представлення дипломного проекту в ДЕК | 17.12.2020 | |

Дата видачі завдання _____ _____________20__ р.

Студент ___________________________________
(підпис)

Керівник роботи __________________________ ____________________________
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка до магістерської атестаційної роботи містить: 101 с., 4 розділи, 31 рис., 4 табл., 48 джерел.

АДАПТАЦІЯ, ІНФОРМАЦІЙНА СИСТЕМА, ВИМОГИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОНТОЛОГІЯ, МЕТА МОДЕЛЮВАННЯ, ПОВТОРНЄ ВИКОРИСТАННЯ

У роботі виконано огляд методів відображення онтологій в задачах адаптації інформаційного та програмного забезпечення інформаційної системи до нових вимог. На підставі проведеного аналізу запропоновано метод відображення онтологій на основі методу рою частинок.

Об'єктом дослідження в рамках магістерської атестаційної роботи є процес адаптації інформаційного та програмного забезпечення інформаційної системи до нових вимог.

Метою даної роботи є дослідження моделей і методів адаптації інформаційного та програмного забезпечення інформаційної системи до нових вимог.

Для досягнення мети, необхідно досліджувати наступні питання:

– аналіз проблеми повторного використання програмного коду;
– огляд існуючих методів повторного використання програмного коду;
– розробка моделей повторного використання програмного коду;
– розробка методів повторного використання програмного коду;
– практична апробація отриманих наукових результатів.

# ABSTRACT

The explanatory note to the master's attestation work contains: 82 pages, 4 sections, 31 figures, 4 tables, 48 sources.

ADAPTATION, INFORMATION SYSTEM, REQUIREMENTS, SOFTWARE, ONTOLOGY, SIMULATION PURPOSE, RE-USE

The paper reviews the methods of ontology display in the problems of adaptation of information and software of the information system to new requirements. Based on the analysis, a method of ontology mapping based on the particle swarm method is proposed.

The object of research in the master's certification work is the process of adapting the information and software of the information system to the new requirements.

The purpose of this work is to study the models and methods of adaptation of information and software information system to new requirements.

To achieve this goal, it is necessary to explore the following issues:

– analysis of the problem of reuse of program code;
– review of existing methods of reusing program code;
– development of models of program code reuse;
– development of methods for reusing program code;
– practical approbation of the obtained scientific results.

## ЗМІСТ

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД –база даних;

БМД – база метаданих;

ІІС – інтелектуальна інформаційна система;

ІО –інформаційні об’єкти;

ІС – інформаційна система;

ЕС – експертні системи;

МРЧ – метод рою частинок;

ПЗ – програмне забезпечення;

ПрО – програмна оболонка;

СПЗ – спеціальне програмне забезпечення;

СУБД – система управління базами даних;

DSL – Domain Specific Language;

DSM – Domain Specific Modeling;

MDK – Meta Data Kernel;

OMV – Ontology Metadata Vocabulary;

OWL – Ontology Web Language;

PSM – Problem Solving Methods;

RLM – Role-limiting methods;

RUP – Rational Unified Process;

SOA – Service-Oriented Architecture (сервіс-орієнтована архітектура);

XP – eXtreme Programming

# ВСТУП

Сьогодні можна говорити, що ера, коли розробники інформаційних систем приходили в організацію і починали проекти інформатизації «з нуля», пройшла. Більшість організацій вже має деякі інформаційні системи (ІС), які з часом починають вимагати цілеспрямованої діяльності по їх реінжинірингу.

Результати досліджень стану інформатизації в різних організаціях дозволяють зробити висновок, що в даний момент більшість з них вже має деякі ІС. Ці ІС в різного ступеня автоматизують процеси, що протікають в організаціях.

Дослідження проектів інформатизації, і, в першу чергу, проектів розробки ІС так само показують, що створення нової інформаційної системи в більшості випадків передбачає зміну стану існуючих ІС. Типовими стали проекти: по розробці нових ІС і їх інтеграції з існуючими ІС; по розробці нових ІС з метою заміни існуючих ІС; по модернізації (нарощування функціональності, розвитку) існуючих ІС.

По суті, сьогодні можна говорити, що ера, коли розробники ІС приходили в організацію і починали проекти інформатизації «з нуля», пройшла. Настає час проектів за систематичною трансформації існуючих ІС або ера реінжинірингу ІС.

Наслідком ситуації, що склалася стає об'єктивна потреба в дослідженні, перегляд і переосмисленні існуючих підходів, методологій і технологій розробки ІС, що, в свою чергу, може зажадати їх модернізації, а можливо, і розробки нових рішень.

Ситуація ускладняються тим, що в даний момент різними дослідниками і практиками поняття реінжинірингу ІС трактується по різному. Багато в чому це обумовлено надзвичайно широким спектром завдань по реінжинірингу, з якими доводиться стикатися в реальних проектах.

Сьогодні в світі існує велика кількість підходів, методів і технологічних рішень, які безпосередньо або побічно співвідносяться з діяльністю з реінжинірингу ІС. Однак вони не інтегровані на рівні методологій (процесів розробки). Як результат, можна спостерігати наявність величезної кількості методологій, де основний акцент зроблений на розробку ІС «з нуля», і практична відсутність «струнких» методологій, метою створення яких було б комплексне, цілісне рішення задач реінжинірингу ІС.

Магістерська атестаційна робота виконується відповідно до вимог методичних вказівок по організації і виконанню магістерської атестаційної роботи [1, 2].

# 1 АНАЛІЗ ПРОБЛЕМИ ПОВТОРНОГО ВИКОРИСТАННЯ ПРОГРАМНОГО КОДУ

## 1.1 Основні проблеми розробки та інтеграції інформаційних систем

З різних причин, головним чином, – економічних, впровадження систем автоматизованого управління відбувається, як правило, поступово, еволюційно. Його починають з окремих функціональних підрозділів, найчастіше з бухгалтерії і/або відділу кадрів підприємства. Так що, типове інформаційне середовище автоматизації – це кілька програм, розроблених в різний час різними розробниками на різних платформах відповідно до того розумінням бізнес-процесів, яке існувало під час їх розробки.

Проблема полягає в тому, що ці програми починають функціонувати як окремі технологічні острівці, незалежні один від одного, з окремими, часто несумісними даними, відсутністю кваліфікованого обслуговуючого персоналу, технічної документації і служб супроводу, без яких неможливий розвиток і вдосконалення систем автоматизованого управління.

Таку автоматизацію управління підприємством іноді називають «клаптиковою». При клаптикової автоматизації практично неможливо побачити реальну картину діяльності підприємства. Отже, неможливо і скільки-небудь обґрунтовано планувати його діяльність і фінансові результати.

Результатом "клаптикового" інформаційного середовища є низька ефективність роботи його складових, збільшення витрат на підтримку, експлуатацію і розвиток, неможливість забезпечити необхідну інформаційно-облікову та аналітичну підтримку бізнес-процесів на належному рівні і, відповідно, втрати в ефективності бізнесу. Саме тому у керівництва компаній все частіше виникає задача інтеграції існуючих на підприємстві "клаптевих" програмних продуктів в єдину (комплексну, інтегровану) корпоративну ІС.

Однак, проблеми інтеграції не обмежуються тільки програмним забезпеченням, вони охоплюють всю інфраструктуру підприємства, яка повинна забезпечити можливість інтеграції не тільки програмним компонентам ІС, але її користувачам, інформації і бізнес-процесам без втрати гнучкості і масштабованості, необхідних для адаптації цієї системи до мінливих вимог бізнесу.

Інтегрована корпоративна ІС дозволяє:

– зберегти раніше зроблені інвестиції, часові та фінансові витрати на підтримку і розвиток інформаційного простору підприємства;

– використовувати для вирішення конкретних завдань найбільш ефективні системи окремих виробників;

– легко розширювати і розвивати окремі можливості існуючих інформаційних систем з уже накопиченими в них даними.

Спроби вирішити інтеграційну проблему виходять і від постачальників корпоративних програмних продуктів. Вони стверджують, що продукти класу ERP (ERP II, CSRP і BPM) "автоматично" знімають проблему інтеграції. В якості підтвердження своєї теорії вони наводять наступні аргументи:

– будь-яка ERP-система автоматизує більшість процесів: управління персоналом, нарахування заробітної плати, обробку замовлень, управління поставками і закупівлями тощо;

– всі ці програми вже "інтегровані", оскільки поставляються однією компанією-розробником;

– впровадження ERP-системи знімає необхідність вкладати значні кошти в інтеграцію додатків.

Однак, практика показала неспроможність їх теорій. Насправді, жодна з цих систем не в змозі вирішити всі задачі, які стоять перед підприємством. Отже, потрібно придбання додаткового модуля або розробка власного додатка, що реалізує необхідну функціональність, і, як результат, проведення інтеграції. Крім того, твердження, що ERP-система вже інтегрована, є досить умовним, оскільки при установці нової версії будь-якого модуля, що входить в ЕЯР-систему, потрібно оновлення і інших модулів. Тому постачальники повинні забезпечити можливість впровадження різних версій своїх додатків – що також вимагає інтеграції.

На даний час створено досить великий обсяг програмних продуктів, бібліотек, програмних модулів, систем управління базами даних (СУБД), систем програмування тощо, що істотно розрізняються по функціоналу. Задача сучасного проектування ІС зводиться до того, що на основі відомих рішень створюється програмне середовище, яке виконує деяку множину функцій. Це здійснюється через проектні рішення, наприклад, Agile-технології, що дозволяють створити систему з актуальним функціоналом в стислі терміни.

Розвиток теорії проектування ІС йде за двома напрямками. Перший шлях пов'язаний з нарощуванням функціонала інформаційного середовища, наприклад, в 1С, GreyGim.

Другий шлях розробки ІС націлений на виконання обмеженого класу функцій з використанням інструменту інтеграції. Для реалізації такої парадигми необхідний клас інтегруючих інструментів, одним з яких є стандартні програмні інтерфейси. В роботі [3] зроблений огляд стандартів та моделей даних, націлених на інтеграцію ІС підприємства на основі MES або ERP-систем. Однак цей приклад є окремим випадком і не може бути поширений на створення ІС іншого класу. Заявкою на інтеграцію стало створення стандартів, об'єднаних навколо моделі взаємодії відкритих систем OSI. На представницькому рівні моделі OSI реалізовані функції перетворення даних, перетворення між різними наборами символів, стиснення даних, шифрування.

Можливі наступні реалізації ідей інтеграції одно функціональних і мало функціональних ІС:

– вбудовування універсального модуля інтеграції в програмний продукт;

– створення середовища взаємодії одно функціональних модулів через відомі формати передачі даних.

Однак не всі ІС, які використовуються на підприємстві, мають можливості інтеграції. У такому випадку говорять про односторонню інтеграцію.

Для вирішення практичних задач інтеграції ІС важливо розуміти її цілі, а також оцінювати ресурси, які готове витратити підприємство на перебудову ІС. Саме тому дослідження критеріїв класифікації варіантів інтеграції і методів інтеграції даних так актуально і розглядається в безлічі робіт. Фактори, що ускладнюють інтеграцію, і варіанти мінімізації негативного впливу цих факторів описані в [4]. До числа таких факторів відносять концептуальну різницю в архітектурі, технологічну різницю, несумісність ліцензій. Пропонується розглядати проблему інтеграції з позиції представлення даних і виділяти синтаксичний і семантичний підходи.

За ступенем автоматизації виділяються три рівня інтеграції: ручна, автоматизована і автоматична. За методом створення зв'язку між інтегрованими елементами системи рекомендується виділяти рівні брокерів, даних, сервісів, інтерпретування метаінформації.

Кожен рівень класифікації має свої цілі інтеграції та інструменти і може бути рекомендований для вирішення відповідного класу задач.

Класифікація дає можливість визначити найбільш раціональний вид інструментів і набір методів створення інтегрованої системи з заданими характеристиками. Запропонована класифікація є спробою виділити ключові, з точки зору практичної реалізації, варіанти інтеграції ІС.

Залежність ступеня автоматизації і методу створення зв'язку при інтеграції представлена а таблиці 1.1.

Таблиця 1.1 – Залежність ступеня автоматизації і методу створення зв'язку

| Метод створення зв'язку | Ступінь автоматизації | | |
|---|---|---|---|
| | Ручна | Автоматизована | Автоматична |
| Рівень брокерів | Відсутня | Дозволена | Бажана |
| Рівень даних | Дозволена | Дозволена | Бажана |
| Рівень сервісів | Дозволена | Бажана | Дозволена |
| Рівень інтерпретування метаінформації | Дозволена | Дозволена | Бажана |

Інтеграція на рівні брокерів. Даний рівень інтеграції використовує набір інструментів API- і COM-технологій. Мета інтеграції – автоматична передача даних і запуск виконуваного коду на виконання. Для кінцевого користувача інтегровані компоненти представляють єдину систему. Інструментами на даному рівні є технології CORBA, COM +, DCOM, RPC..

Приклад реалізації конкретної архітектури описаний в роботі [5]. Групою авторів запропонована системна архітектура BizQuery, заснована на використанні багатофункціонального формату представлення даних XML і мови запитів XQuery. Іншим прикладом реалізації брокера повідомлень є використання web-служб для передачі даних в форматі XML.

Інтеграція на рівні даних. На даному рівні інтеграції додатка настроюються на роботу з єдиною або з декількома пов'язаними між собою базами даних (БД). Мета інтеграції - можливість кінцевих користувачів працювати з єдиними даними, обробляючи їх різними інструментами.

При використанні єдиної об'єктно-орієнтованої СУБД для задач інтеграції виникають проблеми дублювання даних або складності їх вилучення з ієрархічних структур. Для вирішення названих проблем пропонується використовувати паралельні структури, що дозволяє витягувати екземпляри об'єктів, уникаючи надмірності даних і різних аномалій [6].

Моделі інтеграції на рівні даних описані в роботі [7]. Для вирішення проблеми автори пропонують побудувати єдину систему інтеграції даних, основне завдання якої – забезпечення доступу користувачам до даних з усіх корпоративних ІС без структурування та накопичення в єдиній БД.

Інтеграція на рівні сервісів. При інтеграції на рівні сервісів найчастіше використовується сервіс-орієнтована архітектура (SOA) і шина даних. Сервіс являє собою одну або кілька прикладних функцій програми, що реалізують прикладну логіку процесу, що автоматизується. Основні переваги використання сервісів – можливість багаторазового застосування і слабка зв'язаність сервісів один з одним. Метою інтеграції є швидке відпрацювання корпоративної бізнеслогіки. Інтеграція заснована на фіксації інтерфейсів і форматів даних з двох сторін.

На даному рівні можлива найменш витратна інтеграція, де функції передачі даних буде виконувати людина, використовуючи уніфіковані інтерфейси відображення даних. Це так звана неавтоматизована інтеграція. Однак більш перспективний варіант – автоматизована інтеграція. Прикладом успішної реалізації технологічної платформи масової інтеграції слабо пов'язаних інформаційних ресурсів в єдину ІС є система ZooSPACE [8]. Комплекс ZooSPACE будується на довільній кількості слабо пов'язаних самодостатніх вузлів, які функціонують відповідно до єдиної політикою. Взаємодія вузлів між собою здійснюється за допомогою мережевих протоколів прикладного рівня на основі транспортного протоколу TCP/IP.

Інтеграція на рівні інтерпретування метаінформації. Мета інтеграції – ІС зберігає дані в сховище даних, що само описує дані, а користувач може модифікувати не тільки їх, але і метадані, модифікуючи ІС під постійно мінливі вимоги розв'язуваної задачі [9].

Прикладом успішної реалізації сховища даних, що само описує дані є система «COBRA ++» [10], яка дозволяє формувати інформаційну модель об'єктів предметної, швидко перенастроювати метаінформацію при інтеграції даних з нового джерела.

Таблиця 1.2 – Переваги та недоліки методів створення зв'язку в ІС

| Метод створення зв'язку | Переваги | Недоліки |
|---|---|---|
| Рівень брокерів | Універсальність – можливість створити додатковий програмний модуль, який буде звертатися в обидві системи | Складність, трудомісткість, а отже, висока вартість розробки, впровадження та володіння |
| Рівень даних | Низька вартість інтеграції | Руйнування цілісності даних. Якщо БД не має обмежень цілісності, різні додатки можуть приводити дані в суперечливі стани, в іншому випадку в паралельно працюючих з однією БД додатках будуть дубльовані частини коду. При змінах структури БД необхідно переписувати код всіх додатків, які з нею працюють |
| Рівень сервісів | Низька вартість інтеграції, швидке об'єднання систем, що сильно розрізняються без їх модифікації і дописування додаткових модулів | Присутня фіксація; а якщо структури або процеси змінюються, то утворюються проблеми і вузькоспеціалізовані, приватні рішення. Необхідність застосування стандартизованих компонентів |
| Рівень інтерпретування метаінформації | Гнучкість підходу і спрощення модифікації систем, зниження необхідного рівня кваліфікації користувача при внесенні змін до структури і функцій системи | Складність створення програмного забезпечення з підвищеним рівнем абстракції, в реалізації мета-моделі і в зіставленні інформаційних ресурсів в різних системах метаданих |

Аналізуючи дані наведені в таблиці 1.2 можна зробити висновок, що найбільш доцільним для використання є рівень інтерпретування метаінформації, тому з'являється необхідність більш докладно розглянити принципи мета модулювання.

У той же час умови розвитку сучасного підприємства такі, що вдосконалення корпоративної ІС йде еволюційно, шляхом поступового впровадження вузькоспеціалізованих програмних продуктів відповідно до поточних потреб і можливостей, і на підставі цих систем будується інтегрована система управління підприємством [11]. Такий підхід обумовлений безліччю причин, головним чином, економічними – розвиток ІС завжди відповідає поточним потребам підприємства.

## 1.2 Дослідження принципів мета модулювання інформаційних систем

Модель системи – це абстрактний опис на деякій формальній мові характеристик системи, важливих з точки зору мети моделювання, її поведінки. При створенні ІС не можна обмежуватися створенням тільки однієї моделі. Якщо система складна, то облік всіх її характеристик в одній моделі приведе до надзвичайної її складності [12].

Найкращий підхід при розробці нетривіальної системи – використовувати сукупність декількох моделей, які можуть бути практично незалежними одна від одної і дозволять зробити акценти на різних сторонах системи при вирішенні різних завдань підтримки її життєвого циклу.

У загальному випадку моделі можна розділити на наступні види:

- статичні – описують структурні властивості систем;
- динамічні – представляють поведінкові властивості систем;
- функціональні – описують функціональні властивості систем.

Статична модель описує складові частини системи, їх структуру, атрибути, взаємозв'язки між ними і операції, які вони можуть виконувати. Операції статичної моделі є подіями динамічної і функціями функціональної моделей.

Динамічна модель описує послідовність виконання операцій в процесі функціонування системи. Функціональна модель описує перетворення, що здійснюються системою. Вона розкриває зміст операцій статичної моделі і подій динамічної.

За ступенем абстракції моделі можна розділити на:

– концептуальні моделі – представляють високорівневий погляд на завдання в термінах предметної області;

– моделі специфікації – визначають «зовнішній вигляд» і зовнішню поведінку системи;

– моделі реалізації – відображають внутрішній устрій системи, конкретний спосіб реалізації поведінки системи.

Метамодель – це модель мови моделювання, яка застосовується для формалізації опису системи. Лінгвістична метамодель – це метамодель, яка описує предметно-незалежну мову моделювання. Онтологічна метамодель – це метамодель, яка описує предметно-залежну мову моделювання [13].

Чотирьохрівнева ієрархія моделей представляє класичний варіант метамоделювання при створенні ІС (рисунок 1.1). У цій ієрархії кожен попередній рівень визначає мову для опису нижчого рівня. Число рівнів при реалізації конкретних систем може змінюватися.

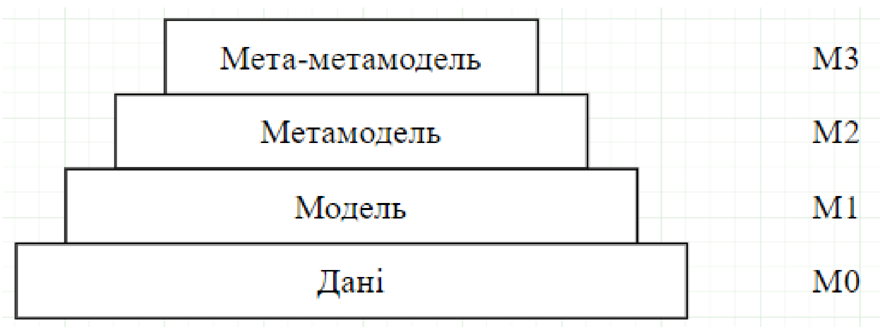


Рисунок 1.1 – Класична чотирирівнева ієрархія моделей ІС

При вирішенні задачі моделювання ІС на рівні M0 знаходяться дані, що описують стан предметної області, тобто модель стану. Рівень M1 є онтологічною метамоделлю. Для рівня M0 і містить модель предметної області. Рівень M2 визначає лінгвістичну метамодель для рівнів M1 і M0: на рівні M2 знаходиться модель мови моделювання, з яким працюють аналітики, розробники, CASE-засоби тощо. Самий верхній рівень (M3) визначає мову, на якому описуються метамоделі рівня M2 [13].

Залежно від кількості рівнів створюваних моделей і способу їх використання при розробці інформаційні системи і технології їх створення можна розділити на кілька класів. У традиційній інформаційній системі «всередині» системи знаходяться дані, що описують стан її предметної області. Ці дані відповідають деякій моделі предметної області, яка може бути описана на будь-якій мові (в тому числі і на природній). Модель розробляється аналітиками, після чого розробники реалізують її за допомогою обраних інструментальних засобів створення ІС, систем програмування. У разі зміни моделі доводиться переписувати вихідні коди програм системи. Найчастіше при реалізації ІС і в процесі її супроводу розробники «йдуть» від початкової моделі, яка не оновлюється при внесенні змін до структур даних і алгоритми їх обробки, що призводить до різних проблем [14].

У традиційних CASE-технологіях модель предметної області визначається формально і знаходиться «всередині» CASE-системи. Модель описується в термінах метамоделі, яка може бути визначена на будь-якій мові і реалізується за допомогою CASE-засобів. Зміна моделі веде до необхідності зміни коду, згенерованої CASE-системою.

Сучасні CASE-засоби надають інструменти для створення і редагування моделей, а також дозволяє згенерувати велику частину коду ІС. Отримана на виході система зазвичай реалізує всі необхідні структури даних, які визначаються моделлю, забезпечує доступ до даних в БД і надає стандартний інтерфейс користувача для роботи з ними. Програмні компоненти, реалізують специфічні для конкретної системи поведінкові і функціональні аспекти, дописують найчастіше вручну. У разі зміни моделі CASE-система дозволяє заново згенерувати код додатків ІС, при цьому код, доданий програмістами «вручну», зберігається. Перевагою підходу є те, що істотно економиться час на початкових етапах розробки. Крім того, підтримується відповідність між системою і моделлю.

Інформаційні системи, керовані метаданими, забезпечують більш потужні можливості для динамічної адаптації. В даному випадку також використовуються три рівні моделей, однак побудована модель предметної області знаходиться «всередині» інформаційної системи в процесі її експлуатації. Таким чином, програмне забезпечення інформаційної системи виступає в ролі інтерпретатора, а модель – в ролі «керуючої системи», яка задає правила функціонування ІС.

Недоліком такого підходу є зниження продуктивності системи під час її експлуатації. Крім того, якщо відсутня можливість підключення зовнішніх програмних компонентів, що розширюють функціональність системи, то страждає універсальність внаслідок неможливості реалізації специфічних для конкретної системи функцій, що відражають бізнес-логіку предметної області. До переваг слід віднести той факт, що при зміні моделі не потрібне повторне кодування – ІС просто починає працювати в відповідно до нової моделі.

Технологія DSM (Domain Specific Modeling) з генерацією коду забезпечує моделювання в термінах предметної області. В даному випадку для вирішення кожної задачі застосовується своя мова моделювання, в якій використовуються виключно поняття і відносини з предметної області ІС.

Тут використовується вже мета-метамодель, яка реалізується Мета-CASE-засобом. За допомогою цього засобу описується метамодель, яка визначає предметно-залежну мову моделювання. На основі цієї моделі генерується CASE-засіб, за допомогою якого описується модель предметної області і генерується ІС. Мета-CASE- і CASE-засоби можуть бути об'єднані в одну CASE-систему.

Використання предметно-залежної мови (Domain Specific Language, DSL) дозволяє істотно спростити процес створення моделей предметної області, в якому можуть брати активну участь експерти. Інші переваги і недоліки, пов'язані з генерацією коду, збігаються з відповідними характеристиками традиційної CASE-технології [15].

Технологія DSM з інтерпретацією метаданих (рисунок 1.2) забезпечує максимальні можливості адаптації. Даний варіант є комбінацією двох попередніх. Метамодель, модель і дані знаходяться «всередині» ІС. В цьому випадку CASE-засоби дозволяють створити моделі і інтерпретувати їх в ході експлуатації системи.

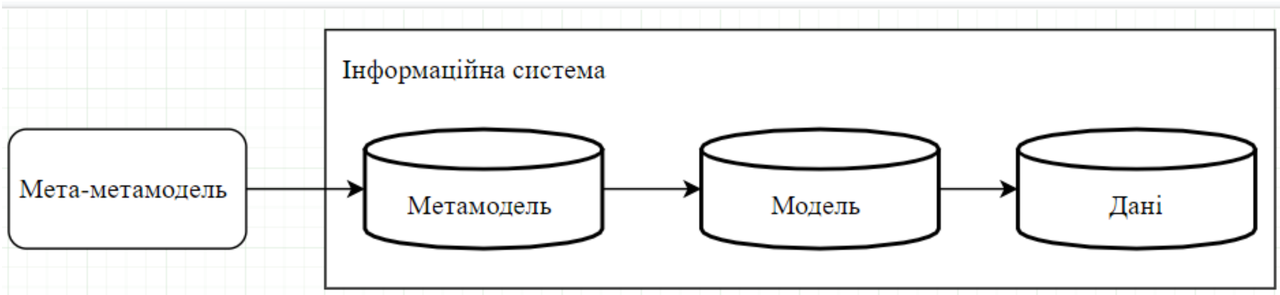

Рисунок 1.2 – Технологія DSM з інтерпретацією метаданих

Для того щоб цей підхід виявився придатним на практиці, необхідно щоб мета-метамодель була максимально виразною. Інтерпретація відразу двох рівнів метамоделей призводить до відчутною втрати продуктивності, однак при достатній виразності мета-метамоделі виходить надзвичайно гнучка система.

Максимальна гнучкість ІС може бути досягнута, якщо як при розробці системи, так і в ході її експлуатації застосовуються метадані, що описують особливості предметної області, для якої створюється система, умови її роботи і характеристики бізнес-процесів і користувачів.

CASE-технологія METAS (METAdata System) - це основа для створення динамічно настроюються інформаційних систем, керованих метаданими, підвищення їх адаптованості в процесі експлуатації за рахунок використання багаторівневих моделей. Ключовим моментом технології є використання взаємопов'язаних метаданих, що описують інформаційну систему і її оточення з різних точок зору і на різних рівнях деталізації.

Технологія METAS базується на використанні мови UML і предметно-орієнтованих мов для розробки моделей ІС, опису бізнес-правил, специфічних для конкретних предметних областей; на технології RUP (Rational Unified Process) і технології розробки XP (eXtreme Programming); на платформі .NET Framework і інструментальних засобах MDK Suite (Meta Data Kernel Suite).

Метадані представляють формалізований опис ІС, розміщене в базі метаданих (БМД), що використовується для налаштування програми на умови експлуатації в процесі його розробки, завантаження і виконання. Вони описують такі аспекти ІС: об'єкти ІС і їхню поведінку, бізнес-операції і бізнес-процеси, первинні документи і звіти, візуальний інтерфейс користувача ІС, модель захисту. Метадані представляють моделі, кожна з яких описує певну частину, аспект ІС (деякі моделі можуть описувати одні і ті ж частини ІС, але з різних точок зору) [16].

Метадані ІС розділені на шари, що представляють наступні основні моделі:

– фізична модель (Physical Model) – метадані, що описують уявлення об'єктів ІС в БД (наприклад, таблиць БД, в якій зберігаються дані про об'єкти, і зв'язків між ними). В процесі функціонування вони служать основою логічної моделі. Модель автоматично генерується по створеному на логічному рівні опису системи.

– логічна модель (Logical Model) – метадані, що описують сутність предметної області, для якої створюється ІС, їх поведінку (через операції), а також загальні операції ІС. Дана модель ґрунтується на нотациях мови UML і дозволяє працювати користувачам системи в термінах предметної області.

– презентаційна модель (Presentation Model) – метадані, що описують візуальний інтерфейс користувача при роботі з об'єктами ІС.

Моделі метаданих та компоненти METAS наведено на рисунку 1.3.

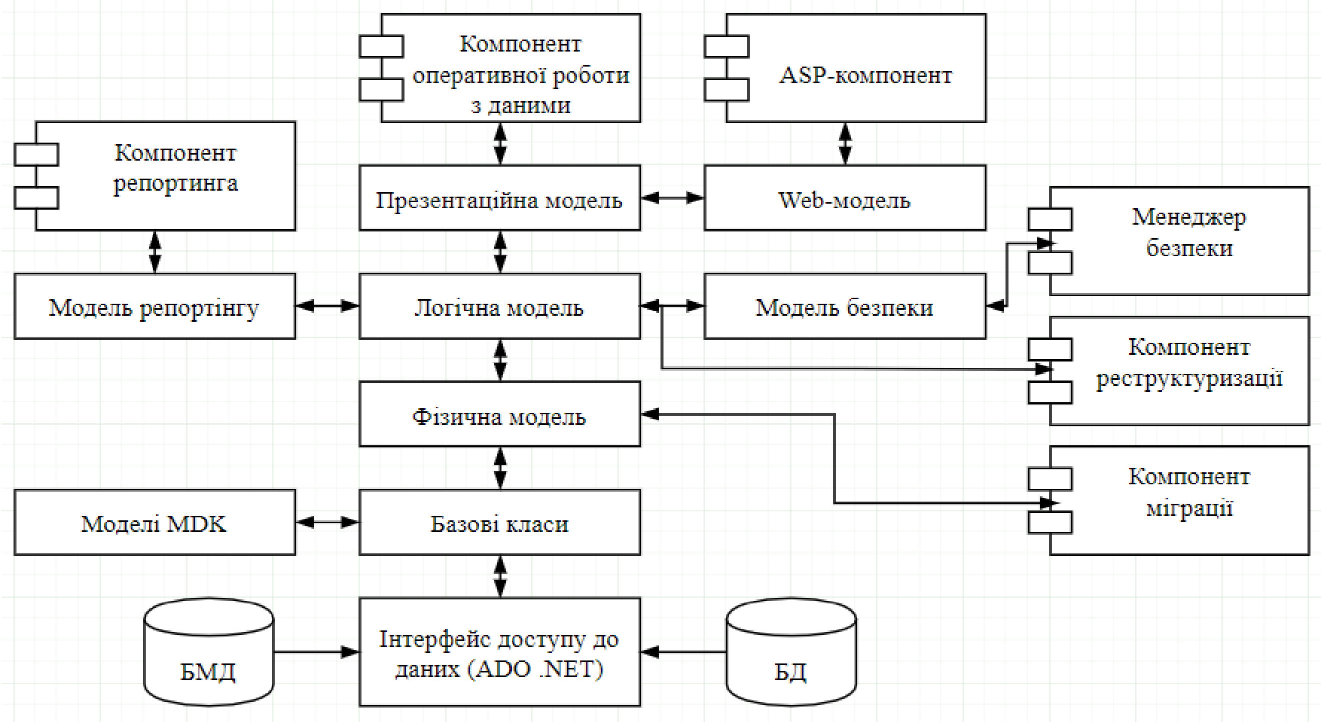


Рисунок 1.3 – Моделі метаданих та компоненти METAS

Набір характеристик, що відображаються в моделі, може бути динамічно розширено. набір метаданих може також розширюватися шляхом додавання нових моделей, що описують нові сторони і властивості ІС або існуючі, але з нових точок зору. Зокрема, в систему включено наступні моделі, які спираються на перераховані вище основні [17]:

– модель репортінгу (Reporting Model) – метадані, що описують запити, первинні документи і звіти, що формуються в ході виконання бізнес-операцій і бізнес-процесів, які використовуються для аналізу даних;

– модель бізнес-процесів (Business-process Model) – метадані, що описують бізнес-операції і бізнес-процеси, підтримувані ІС;

– Web-модель, яка забезпечує доступ до ресурсів ІС для віддалених користувачів через Web-інтерфейс.

Модель безпеки (Security Model) дозволяє контролювати повноваження користувачів, їх права на виконання операцій над об'єктами ІС або на доступ до моделей метаданих. підсистема захисту працює з власною БД.

Стандартна бізнес-логіка може бути розширена шляхом визначення нових типів і операцій, специфічних для конкретної ІС. Забезпечується адаптація ІС без перепрограмування її компонентів і без участі розробників. Архітектура ІС являє собою клієнт-серверний додаток (рисунок 1.4), розбитий на домени.

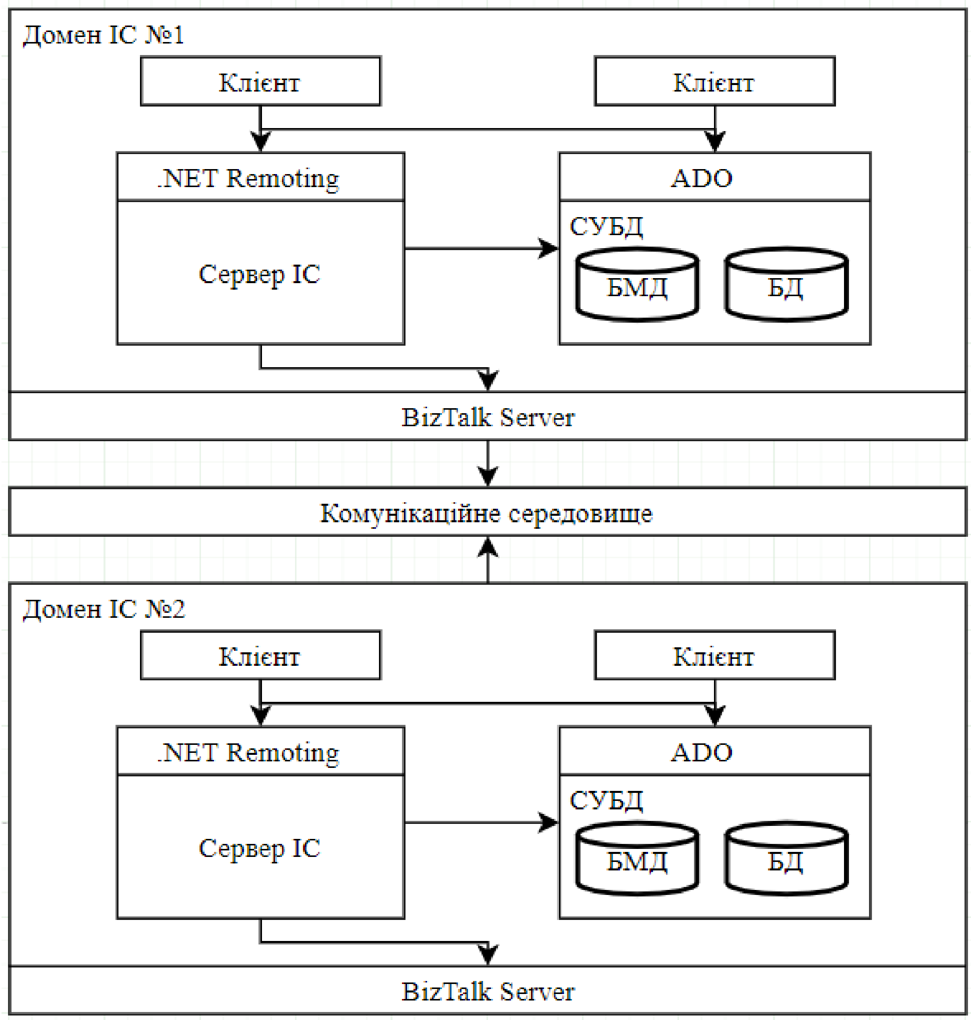


Рисунок 1.4 – Доменна архітектура ІС

Розбивка на домени призначена для реалізації розподілених ІС, що включають автономно функціонуючі підсистеми, що не мають оперативного зв'язку. Домен ІС – закінчений розподілений додаток, що представляє собою підсистему корпоративної ІС, встановлену в окремому закладі, що складається з одного сервера і кількох клієнтів. Зв'язок між доменами ІС встановлюється для реплицюваняя моделей і даних, обміну документами, звітами.

Основними перевагами представленої технології є:

– гнучкість і можливість динамічної адаптації системи до змін умов функціонування, потребам користувачів з мінімальними витратами;

– можливість інтеграції з зовнішніми системами;

– відсутність необхідності спеціальної підготовки користувачів, забезпечення можливості роботи в звичному для них середовищі і в термінах знайомої предметної області;

– невисокі вимоги до програмно-апаратній платформі при досить потужних можливостях збору, зберігання і обробки даних.

Але залишається проблема вибору метода для автоматичного використання вже існуючого програмного коду, що потребує аналізу цих методів.

## 1.3 Аналіз існуючих методів повторного використання коду

Якість розробки програмного забезпечення залежить від багатьох факторів, зокрема від того, наскільки добре були використані методики щодо збільшення відсотка повторно використовуваного коду.

Повторне використання коду – це методологія проектування комп'ютерних та інших систем, яка полягає в тому, що програмний модуль частково або повністю складається з частин, написаних раніше компонентів і/або частин іншої системи [18].

Практично структуру всього сучасного ПЗ можна представити у вигляді каркаса, в гніздах (спеціальних осередках) якого розміщуються різні елементи програми, назвемо їх наповнювачами. З цієї точки зору методи повторного використання можна розділити на ті що [19]:

– використовують наповнювачі як елементи повторного використання;

– використовують каркас системи як елемент повторного використання;

– використовують як елементи повторного використання як каркас, так і наповнювачі.

Найбільш відомим прикладом застосування методів першої групи є бібліотеки підпрограм. Механізм їх неодноразового використання відомий вже на протязі багатьох десятиліть і в найближчому майбутньому, очевидно, не здасть своїх позицій. Так, бібліотеки стандартних функцій використовуються у всіх високорівневих мовах програмування. Основний акцент в даному методі робиться на повторне використання коду.

Як приклад другої технології можна привести виділення в незмінний каркас роботу з об'єктами Windows. Тут маніпуляції, які супроводжують ініціалізацію, знищення тощо, виділені в незмінний каркас. Це дозволяє програмісту відразу приступати безпосередньо до роботи зі змінною частиною програми. Даний підхід застосовується в багатьох сучасних інструментах створення програмних додатків. Іншим прикладом багаторазово використовуваного каркаса може служити організація програм обчислювального експерименту. Тут у міру отримання нових результатів часто доводиться перебудовувати обчислювальну модель. Тому консервативну частину програми вигідно виділити в незмінний каркас, а мінливі зробити наповнюють елементами. Таким чином, каркас диктує певну архітектуру програми, тут увага приділяється повторному використанню дизайну, а не коду.

Третій метод  також досить поширений в даний час. Так, наприклад, класи в об'єктно-орієнтованому програмуванні об'єднують в собі важливі властивості як наповнювача, так і каркаса. З одного боку, до методів класу можна звернутися як до підпрограми для вирішення завдань, з іншого боку, механізм успадкування дозволяє використовувати клас як каркас для дочірніх класів. В принципі, абстрактні класи – не що інше, як каркас.

Ще однією ознакою, за якою можна класифікувати методи повторного використання, є критерії декомпозиції програми (розбивки програми на елементи повторного використання). За цією ознакою можна виділити наступні технології:

– функціональний підхід – один елемент відповідає за виконання однієї функції;

– підхід, орієнтований на дані – один елемент повинен інкапсулювати одну структуру даних;

– підхід, орієнтований на події – один елемент повинен розпізнавати і обробляти тільки одну подію;

– проблемно-орієнтований підхід – кожен елемент повинен відповідати одній і тільки одній речі (проблеми) в системі.

Критерієм для класифікації методів повторного використання також може служити програмний елемент, що приймається за одиницю повторного використання. За цією ознакою можна виділити:

– функціонально-орієнтований підхід;

– об'єктно-орієнтований підхід;

– суб'єктно-орієнтований підхід;

– компонентно-орієнтований підхід;

– шаблонно-орієнтований підхід.

Функціонально-орієнтований підхід заснований на повторному використанні різних функцій. Прикладом тут можуть служити підпрограми обчислення синуса або косинуса, включені в стандартні бібліотеки які використовуються багатьма програмістами. Розробники можуть також створювати свої функції, процедури і макроси, складати з них бібліотеки внутрішнього користування і застосовувати їх неодноразово в своїх проектах [20].

В об'єктно-орієнтованому підході одиниця повторного використання – об'єкт. Об'єкт являє собою структуру, яка об'єднує в собі деякі дані і функції для роботи з ними. Для об'єктів підтримуються такі механізми, як наслідування, інкапсуляція і поліморфізм. Об'єкт вважається за своєю природою пасивним елементом, тобто функції або методи об'єкта для виконання своїх специфічних дій повинні викликатися ззовні [21].

Творці суб'єктно-орієнтованого підходу стверджують, що дана технологія розширює концепцію об'єктів для моделі паралельних обчислень. Так звані суб'єкти інкапсулюють в собі потоки управління і мають інтерфейси (функції, методи) для взаємодії з іншими суб'єктами. Суб'єкт вважається активним елементом [22].

Ідея компонентно-орієнтованого підходу в корені відрізняється від двох попередніх. Програмна компонента – це повторно використовувані шматки коду і даних в двійковій формі, які можуть бути відносно легко вбудовані в інші програмні компоненти від інших виробників. Часто від розробника прихована внутрішня реалізація цих елементів. Як правило, виділення будь-якої частини програми в окрему компоненту відбувається не за функціональною ознакою, а з точки зору зручності програмування, поширення, інсталяції та ін. Сьогодні цей підхід знаходить все більше застосування, наприклад в технології ActiveX.

Шаблонно-орієнтований підхід заснований на використанні в якості багаторазово застосовуваних компонентів шаблонів [2]. Смисловими синонімами даного терміну є патерни, зразки, еталони. Шаблон в загальному випадку являє собою заготовку для створення будь-якого елементу системи. У процесі створення виробу на основі шаблону відбувається насичення шаблону конкретним змістом. Як правило, подібний зразок містить в собі набір істотних компонент, обмежень, правил, необхідний для створення закінченого об'єкта на його основі. В принципі шаблон в деякому роді нагадує каркас, але між ними існують суттєві відмінності: шаблони більш абстрактні, ніж каркаси, як архітектурні елементи шаблони дрібніше, ніж каркаси, шаблони менш спеціалізовані, ніж каркаси.

На даний момент технології, які використовують той чи інший підхід в чистому вигляді, практично відсутні. Так, разом з об'єктно-орієнтованим підходом може застосовуватися шаблонно-орієнтований. Більш того, прихильність тільки до однієї технології може знизити ефективність процесу розробки ПЗ.

Однак зростання складності програмних систем привів до того, що з'явилася необхідність в повторному використанні наскрізної функціональності. Тому виявилася потреба в новому підході повторного використання коду, який б охопив цей тип функціональності і вирішив актуальні проблеми. при розробці складних програмних комплексів з великою кількістю розробників відсоток повторно використовуваного коду знижується з ростом його обсягу. Це зв'язано з тим що кожен кваліфікований розробник повинен володіти знаннями про те, де є вже написаний раніше програмний код, який можливо підлягає повторному застосуванню. Таким чином, повторне використання коду залежить від особистісних якостей розробників.

Сучасні інструментальні системи розробки ПЗ, наприклад, такі як Microsoft Visual Studio (починаючи з версії 2005) дозволяють створювати так звані "сніпети" (snippets) – готові фрагменти коду для класів і їх членів, які можна повторно використовувати в програмах. Такий механізм надзвичайно корисний, так як він економить час розробки програмного забезпечення, дозволяючи визначати заздалегідь відформатований код з можливістю зручної кастомізації. Однак при цьому застосовуються тільки заздалегідь заготовлені шаблони, а не весь обсяг програмного коду розроблюваних систем. В сучасних умовах на ринку, при великій кількості компаній-розробників не існує засобів розробки і модулів розширення (plug-in) до них, що забезпечують здійснення інтелектуальної підтримки повторного використання програмного коду.

Виходячи з цього, з’явилася необхідність побудови інтелектуальних систем, заснованих на знаннях, які зможуть виконувати підбір вже існуючих компонентів або модулів інформаційних систем для реалізації нової ІС.

# 2 РОЗРОБКА МОДЕЛЕЙ ПОВТОРНОГО ВИКОРИСТАННЯ ПРОГРАМНОГО КОДУ

## 2.1 Дослідження інтелектуальних систем, заснованих на знаннях, з повторним використанням компонентів

В даний час виявлено великий інтерес до інтелектуальних систем, заснованих на знаннях у всіх областях сучасної науки і людської діяльності. Системи, засновані на знаннях (СОЗ) використовуються для вирішення різних типів завдань в найрізноманітніших проблемних областях, таких, як нафтова і газова промисловість, фінанси, енергетика, транспорт, фармацевтичне виробництво, космос, хімія, освіта, телекомунікації і зв'язок і ін. З іншого боку це вся діяльність обмежена недостатню розробленість даних систем.

СОЗ є системами програмного забезпечення, основними структурними елементами яких є база знань і механізм логічного виводу. Серед СОЗ можна виділити:

– інтелектуальні інформаційно-пошукові системи;
– експертні системи (ЕС).

В інформатиці ЕС розглядаються спільно з базами знань як моделі поведінки експертів у певній галузі знань з використанням процедур логічного висновку і прийняття рішень, а бази знань – як сукупність фактів і правил логічного висновку в обраній предметній області.

Під експертними системами розуміються складні програмні комплекси, що акумулюють знання фахівців в конкретних предметних областях і тиражують цей емпіричний досвід для консультацій менш кваліфікованих користувачів [23].

Базова структура експертної системи наведена на рисунку 2.1.

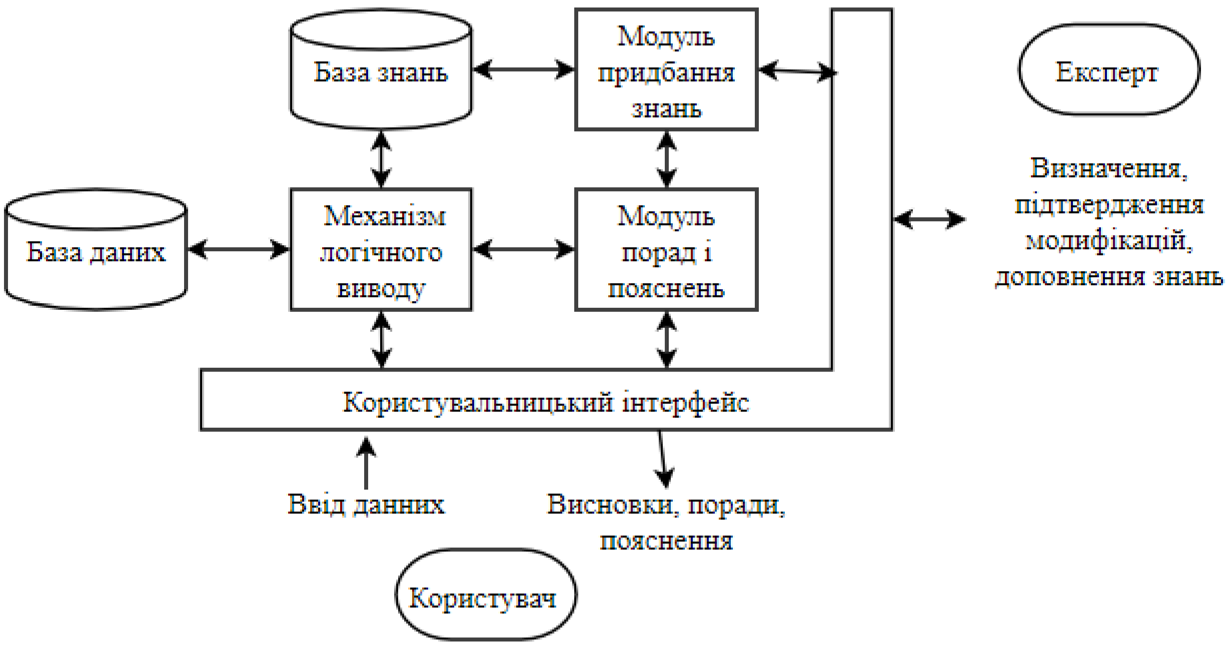


Рисунок 2.1 – Структура експертної системи

Структурні елементи, що складають ЕС, виконують наступні функції:

– база знань (БЗ), містить факти і правила, за якими в залежності від вхідної інформації приймається те чи інше рішення. Знання в базі знань представлені в конкретній формі, і організація бази знань дозволяє їх легко визначати, модифікувати і поповнювати. База знань реалізує функції представлення знань в конкретній предметній області і управління ними;

– механізм логічного виводу виконує логічний вивід на підставі знань, наявних в базі знань. Механізмом логічного виводу називаються загальні знання про процес знаходження рішення. Він виконує дві основні функції: зміна БЗ на основі аналізу БЗ і вихідної інформації та управління порядком обробки правил в БЗ.

– модуль придбання знань необхідний для отримання знань від експерта, підтримки бази знань і доповнення її при необхідності;

– модуль порад і пояснень формує висновок експертної системи і представляє різні коментарі, що додаються до висновку, а також пояснює мотиви ув'язнення;

– призначений для користувача інтерфейс – необхідний для правильної передачі відповідей користувачеві, інакше користуватися системою вкрай незручно.

Передбачалося, що використання СОЗ може бути застосовано до проектування. Однак ці надії не виправдалися за такими причинами:

– створення бази продукційних правил для нової предметної області виявилося досить дорогим;

– проблема «вузького горлечка» не може бути вирішена, так як існують труднощі опису предметної області в термінах понять низького рівня абстракції, необхідних для функціонування блоку логічного виводу.

Стало очевидним, що експертні знання, представлені на рівні продукційних правил не забезпечують досить абстрактну модель для створення СОЗ [24].

З метою абстрагування від конкретної задачі і переходу до мета-опису поведінки системи були запропоновані чотири основні підходи:

– поняття родової, загальною (generic) задачі, наприклад, використання предметно незалежного методу розв'язання задачі;

– поняття обмеженої ролі методу рішення задачі, яке дозволяє розглядати, наприклад, різні ролі складових методу рішення і, відповідно, визначати порцію знань для конкретної прикладної задачі;

– припущення про «відкритості світу»;

– бібліотека онтологічного опису знань.

Для подальшого дослідження, перш за все, необхідно визначити поняття «онтологія». Онтологія – це точна (виражена формальними засобами) специфікація концептуалізації [24]. Під концептуалізації розуміється процес переходу від подання ПрО до точної специфікації цього опису на деякій формальній мові, орієнтованій на комп'ютерне уявлення.

З цієї точки зору кожна БЗ, система, заснована на знаннях, або програмний агент явно або неявно базуються на деякій концептуалізації. Множина понять і відносин між ними відображаються в словнику. Таким чином, вважається, що основу онтології складають множина представлених у ній понять.

Онтологія є не абсолютною (єдиною) специфікацією концептуалізації ПрО, а залежить від цілей її створення, тобто задач, при рішенні яких планується її застосовувати. Незалежно від виду онтології вона повинна включати словник термінів і деякі специфікації їх значень, що дозволяє обмежувати можливі інтерпретації термінів і відображати взаємозв'язок понять даної ПрО.

Основною областю застосування онтології є інтеграція інформації. Онтології пов'язують два істотних аспекти [25]:

– визначають формальну семантику інформації, дозволяючи обробку інформації комп'ютером,

– визначають семантику реального світу, дозволяючи, ґрунтуючись на загальній термінології, пов'язувати інформацію, представлену у вигляді, необхідному для комп'ютерної обробки, з інформацією, представленою в зручній для сприйняття людиною формі.

Опис онтології в термінології словника зазвичай представляється у формі, запропонованій теорією логіки першого порядку, де слова зі словника виступають в якості унарних або бінарних імен предикатів, які називаються відповідно поняттями і відносинами. У найпростішому випадку онтологія описує ієрархію пов'язаних понять, в більш складних випадках, додаються відповідні аксіоми, що виражають зв'язки між поняттями і обмежують їх інтерпретацію.

Інтеграція онтології представлена на рисунку 2.2.

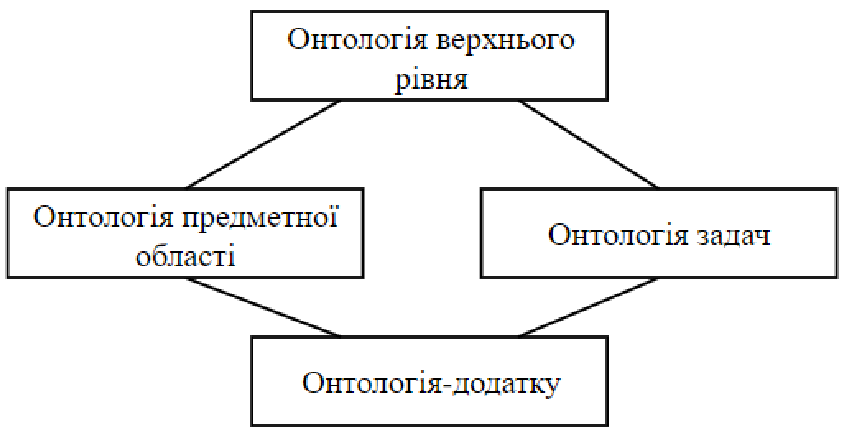


Рисунок 2.2 – Багаторівнева схема відносин між онтологіями

Представлені на рисунку онтології розглядаються наступним чином:

– онтології верхнього рівня описують загальні поняття такі, як простір, час, матерія, об'єкт, подія, дія тощо, що не залежать від конкретної задачі в ПрО;

– онтології предметної області складаються з об'єктів і зв'язків між ними, описаних в термінології конкретної предметної області;

– онтології задач орієнтовані на рішення конкретних проблем і включають всі поняття, необхідні для опису процесу логічного виводу, від самих абстрактних понять, що відносяться до схеми виводу, до більш спеціальних, характерних для окремих методів [26];

– онтології-додатки описують поняття, залежні, як від конкретної проблемної області, так і від задачі. Ці поняття часто характерні для обох онтологій, що з'єднуються і відповідають ролям, граною об'єктами предметної області при виконанні певної діяльності.

Вважається, що онтологія задачі є загальною, якщо вона незалежна від специфіки предметної області, або додатку, або методу виведення [27]. Наприклад, в проблемі планування дій онтологія завдання формалізується без вказівки будь-якої специфіки завдання планування, але в той же час витягує і об'єднує важливі концептуальні теоретичні відмінності, які існують в більшості парадигм планування. Поняття загальної задачі може бути поширене на метод вирішення, на онтологію предметної області і є ключовим підходом для побудови бібліотеки повторно використовуваних компонентів у процесі проектування систем, заснованих на знаннях.

Підхід, який використовує методи обмеження ролі (Role-limiting methods – RLM), можна охарактеризувати як оболонковий. Така оболонка виходить разом з реалізацією конкретного набору методів вирішення задач (Problem Solving Methods – PSMs), таким чином, її можна використовувати тільки для рішення такого типу завдань, для якого PSM підходить. Даний PSM також визначає ролі, які знання можуть грати в ході процесу вирішення завдань; він фіксує уявлення знань для ролей, так що експерт повинен підтвердити загальні поняття і відносини, які визначаються за допомогою цих ролей [28].

RLM, засновані за евристичної класифікації, наприклад, пропонують роль, яка точно визначається експертом. Використовуючи цю роль, експерт:

– повинен вказати, яке характерне для предметної області поняття відповідає цій ролі, наприклад, «дані про пацієнта»;

– повинен забезпечити екземпляри предметної області для цього поняття, наприклад, конкретні факти про пацієнтів.

Важливо підкреслити, що тип знань, який використовує RLM, визначено заздалегідь. Отже, отримання необхідних примірників, характерних для предметної області, може підтримуватися графічними інтерфейсами, які виконані спеціально для даного PSM.

Онтологія, що реалізована на мові OWL (Ontology Web Language), являє собою множину декларативних тверджень про сутності словника предметної області [29]. OWL передбачає концепцію «відкритого світу», відповідно до якої застосовність описів предметної області, поміщених в конкретному фізичному документі, не обмежується лише рамками цього документа – зміст онтології може бути використано і доповнено іншими документами, що додають нові факти про ті же сутності або описують іншу предметну область в термінах даної. «Відкритість світу» досягається шляхом додавання URI кожному елементу онтології, що дозволяє сприймати описану на OWL онтологію як частина загального об'єднаного знання.

Бібліотека онтологічного опису знань містить PSM, кожен з яких декомпозує завдання на підзадачі і / або примітивні логічні висновки і має в своєму розпорядженні структуру для контролювання виконання підзадач і примітивних висновків, а також – для отримання спільного рішення задачі. Підзадача – це задача, для вирішення якої може бути застосований інший відповідний метод. Кожна задача має опис інформації на вході і на виході.

Кожен PSM має множину критеріїв придатності, які використовуються для вибору методу в додатку. Ці критерії виражаються в термінах властивостей середовища, предметної області та наявних знань; в методології Common KADS (Knowledge Acquisition and Documentation Structuring) все це називається властивостями задачі.

Серед інструментальних засобів для побудови систем, заснованих на знаннях можна визначити:

– модельно-орієнтований підхід;

Вихідною мотивацією для моделі експертизи KADS послужила підтримка вербальної інформації від людини-експерта, зокрема, міркування вголос. Модель експертизи це реалізація незалежної моделі експертних знань про вирішення проблеми. Її можна розділити на чотири рівні за різних видом знань з обмеженою взаємодією між рівнями:

– рівень предметної області, який описує концепти і відносини між ними;

– рівень виводу, який описує елементарні виводи, які можна застосувати до

– знань про предметну область, звані джерелами знань;

– рівень задачі, зі структурою управління, званої структурою задачі, яка показує, як можна комбінувати джерела знань, щоб досягти мети;

– стратегічний рівень, містить знання про те, як створювати структури задачі і знаходити рішення в тупикових випадках.

Стратегічний рівень зазвичай явно не моделюється, зокрема тому, що судження про стратегії не здаються важливими при вирішенні рутинної задачі, яка є об'єктом при розробці системи. В СОЗ стратегічний рівень відображає здатність користувача вникнути в процес рішення задачі.

Так само як і технології Protégé, KADS зі часом розвивається. Методологія Common KADS створювалася з метою використання більш точної, формалізованої мови для опису моделей експертизи, а також – шляхом включення нових ідей щодо компонентів експертизи. Common KADS також ввела ідею бібліотеки повторного використання методів рішення проблем, побудованої на базі моделей рівня виводу і рівня задач KADS.

– підхід, заснований на декомпозиції задачі;

Модель знань Protégé заснована на моделі Open Knowledge Base Connectivity (OKBC), яка забезпечує одноманітну модель Knowledge Representation Systems (KRS). Знання зберігаються в спеціалізованому форматі. Це дозволяє читати і писати в PDF файли і використовувати формат реляційних баз даних.

Protégé-2000 дозволяє користувачам вносити зміни в онтологію і надає undo / redu функціональність. Більш того, надається можливість архівувати різні версії онтології і повернутися до попередніх версіями. Це один з небагатьох інструментів, які підтримують роботу з версіями онтології.

– підхід, заснований на Web. Проект IBROW3.

Основна мета проекту IBROW3 (An Intelligent Brokering Service for Knowledge Component Reuse on the World-Wide Web) полягала в розробці архітектури, яка полегшувала «інтелектуальний брокерський сервіс »для створення СОЗ, за допомогою багаторазового використання компонентів знань третьої сторони через WWW.

Мета-онтологія UPML (United Problem-Solving Method Development Language) була розроблена для підтримки визначення компонентів знань.

UPML вводить чотирьохкомпонентну архітектуру для систем знань за допомогою розширення моделі експертизи Common KADS. Ідентифікуються наступні компоненти:

– задача – визначає проблему, яка буде вирішуватися;

– метод рішення задачі – описує процес виводу висновків;

– модель предметної області – визначає знання про предметну область;

– адаптер – полегшує з'єднання трьох інших компонент, визначених за допомогою незалежних і, отже, багато разів використовуваних специфікацій.

За результатами дослідження можна зробити висновок, що найбільш доцільним для використання є підхід, заснований на Web, так як саме він, перш за все, націлений на повторне використання знань, а відповідно і програмного коду.

## 2.2 Дослідження репозиторіїв онтологій як засобу повторного використання знань

Повторне використання і відкритий доступ до знань є одним з провідних мотивацій для Semantic Web. В останні роки онтології застосовуються в багатьох прикладних інтелектуальних ІС, особливо в контексті Semantic Web. Слід зазначити, що, незважаючи на той значний інтерес, який проявляють дослідники до найрізноманітніших аспектів онтологічного аналізу – автоматизованої побудови онтологій, їх порівнянні, поповненню і аналізу, саме питання зберігання і створення метаописів онтологій залишаються практично не порушеними.

Академічні та промислові розробки, пов'язані з використанням онтологій, забезпечують нові технології в цій галузі. Сьогодні в Web знаходиться велика кількість онтологій з самих різних ПрО. Але через складність структури онтологій і їх великої кількості користувачеві важко не тільки вносити в них зміни і доповнення, але і взагалі знайти релевантну за тематикою і рівнем складності онтологію. Таким чином, накопичилося досить велика кількість онтологій, що зберігають інтелект багатьох експертів, що викликає потребу в зособах для їх спільного і повторного використання. Це робить актуальним задачу створення репозиторіїв онтологій, призначених для забезпечення зберігання та пошуку онтологій і онтологічних модулів. Найбільш доцільним для цього е семантичний пошук.

Семантичний пошук являє собою різновид автоматизованого інформаційного пошуку, в якому враховуються смислові аспекти запиту користувача, доступних інформаційних ресурсів (ІР), серед яких здійснюється пошук, і контексту запиту [31]. При семантичному пошуку його предметом може бути не конкретний ІР або його фрагмент, а інформаційний об'єкт певного класу.

Як правило, він передбачає семантичний аналіз природно-мовної складової об'єктів і запиту користувача. Для семантичного аналізу можуть застосовуватися контент-аналіз, метод семантичних відмінків, асоціативний аналіз, метод тематичної класифікації на основі моделі структурного представлення тексту, семантичний диференціал, латентно-семантичний аналіз тощо.

Принципова особливість семантичного пошуку полягає в тому, що тим чи іншим чином аналізуються не тільки формальні параметри розглянутих об'єктів, але і їх семантика. Ефективність пошуку можна значно підвищити за рахунок інтелектуального аналізу об'єктів, для якого застосовуються агентний і онтологічний підходи. При цьому онтології можуть застосовуватися як для опису семантики контенту окремого документа і його структури, так і для опису тих об'єктів, інформація про яких потрібна користувачеві.

При переході до семантичного пошуку виникає проблема розпізнавання різних ІО. Розпізнавання ІВ передбачає виявлення в будь-якому ІР відомостей про ту чи іншу ІВ, що цікавить користувача. Розпізнавання ІО можна розглядати як окремий випадок задачі розпізнавання образів.

При розпізнаванні образів необхідно провести класифікацію об'єктів із заданої множини за наявними описами об'єктів і класів. Стандартна постановка задачі розпізнавання образів полягає в тому, щоб для множини об'єктів M, описуваних набором ознак, і подається у вигляді об'єднання непересічних підмножин (класів)

(2.1)

таких, що

(2.2)

причому для K – частини об'єктів з M,

(2.3)

відомо, до якого саме класу вони відносяться, потрібно по набору значень ознак об'єктів з $L = M \setminus K$ визначити клас цих об'єктів.

Першим кроком в цьому напрямку були підходи по створенню колекцій онтологій і відповідних їм ресурсів. Початкові проекти зі збору бази існуючих онтологій пропонували створення бібліотечних систем, які надають різні функції для управління, адаптації та стандартизації груп онтологій. Ці системи є важливими засобами для угруповання і реорганізації онтологій, для їх подальшого повторного використання, інтеграції, технічного обслуговування, відображення і управління версіями. Прикладами бібліотечних систем є: WebOnto, Ontolingua, DAML Ontology Library System, SchemaWeb тощо.

Останнім часом з'явилося багато ініціатив в сфері створення семантично організованих інформаційних просторів, наприклад, робоча група Open Ontology Repositories спільноти Ontolog; Swoogle, WATSON. Однак всі ці підходи не дозволили забезпечити інтероперабельність онтологічних знань, достатню для підтримки завдань семантичного пошуку і розпізнавання. Основна проблема полягає в тому, що більшість колекцій онтологій пропонують пошук потрібної користувачеві онтології лише за ключовими словами і короткого опису, що в цілому не відображає семантику самих онтологій, і не встановлює загальноприйнятий стандарт для опису онтологій.

Репозиторії онтологій повинні полегшити виявлення багаторазово використовуваних онтологічних компонентів (цілих онтологій або їх частин), які відповідають вимогам користувача, і спільний загальний доступ до них.

Репозиторії онтологій подібні репозиторіям даних [32], які представляють собою набори цифрових даних, доступні для однієї або декількох сутностей для різних цілей і володіють рядом характеристик [33] :

– контент поміщається в репозиторій його творцем, власником або третьою стороною;

– архітектура сховища управляє як контентом, так і метаданими;

– репозиторій пропонує мінімальний набір базових сервісів, наприклад, отримання, пошук, управління доступом;

– репозиторій повинен бути стійким і надійним, добре підтримуваним і добре керованим.

Але репозиторії онтологій мають ряд специфічних якостей, пов'язаних з особливостями тих об'єктів, для зберігання яких вони призначені, – онтології і їх фрагменти можуть бути пов'язані один з одним.

Репозиторій онтологій (Ontology Repository – OR) – це структурований набір онтологій (схем і примірників), модулів і додаткових метазнань, який використовує словник онтологічних метаданих. Зв'язки і відносини між онтологіями і їх модулями є семантичною модель сховища онтологій [34]. Репозиторій онтологій складається з системи управління, по-перше, репозиторієм онтології і, по-друге, контентом, поміщеним в цей репозиторій. В даний час активно проводяться наукові дослідження в напрямку організації зв'язків між репозиторіями. Вирішення цієї проблеми дозволить здійснювати глобальний пошук та вивід репозиторіїв в глобальному масштабі на основі семантичного інформаційного простору.

Система управління репозиторієм онтології (Ontology Repository Management System – ORMS) являє собою програмне забезпечення для зберігання, організації, зміни і вилучення знань з репозиторію онтологій. ORMS підтримує семантику для функцій зберігання, організації, модифікування і вилучення знань з репозиторію онтологій. Крім того, в різних реалізаціях ORMS можуть підтримуватися і багато інших функцій, пов'язаних з обробкою онтологій. ORMS є аналогом СУБД для роботи зі специфічним контентом – онтологіями з урахуванням їх семантики.

Прикладами впровадження ORMS є децентралізована система Oyster [35], централізована Onthology і Generic Ontology Repository Framework (GORF).

Основною метою створення репозиторіїв онтологій є підтримка доступу до знань та їх повторного використання людиною і машиною. Репозиторії онтологій призначені для вирішення різних питань, пов'язаних з інженерією онтологій. Одна з ключових проблем інженерії онтологій – те, що більшість онтологій будують з нуля, а не повторно використовують вже існуючі, що призводить до зайвих інженерних зусиль і витрат. Причиною цього є те, що більшість існуючих онтологій будується з урахуванням конкретного сценарію програми, що робить їх подібними замовленому програмному забезпеченню. Такий підхід призводить до того, що онтології пристосовані для роботи з конкретними програмами, але не представляють собою артефакти знань в традиційному сенсі. При розробці таких онтологій інженери фокусуються на очікуваній поведінці додатку, а не на повторному використанні та сумісності з іншими онтологіями.

Іншою проблемою є те, що онтології, які намагаються представити знання досить широкою ПрО, виявляються занадто громіздкими для ефективного використання. Але і модульності онтології недостатньо для повторного використання онтологій, якщо розробники не можуть ефективно знаходити потрібні модулі. Тому виникає необхідність у відповідній інфраструктурі, яка підтримувала б інтелектуальне дослідження онтологій і їх вибір кінцевими користувачами.

З технічної точки зору різні практичні реалізації репозиторіїв онтологій значно відрізняються один від одного. Існує проблема інтероперабельності між ними, оскільки розробники застосовують різноманітні методи і технології для інтерпретації та використання метаданих. Крім того, в більшості існуючих репозиторіїв погано підтримуються такі функції, як модулярізація і версійність, так само як і відносини між репозиторіями онтологій і середовищем розробки онтологій для підтримки всього життєвого циклу онтології.

Незважаючи на це, відповідні компоненти або сервіси на концептуальному рівні є повторно використовуваними для різних технічних рішень. В представлена концептуальна структура для репозиторіїв онтологій (рисунок 2.3). Спираючись на різні реалізації репозиторіїв онтологій, можна визначити набір відповідних компонентів і сервісів, які повинні бути вбудовані в надійну структуру. Репозиторій повинен забезпечувати пошук онтологій, їх зручне для користувача представлення і персоналізацію.

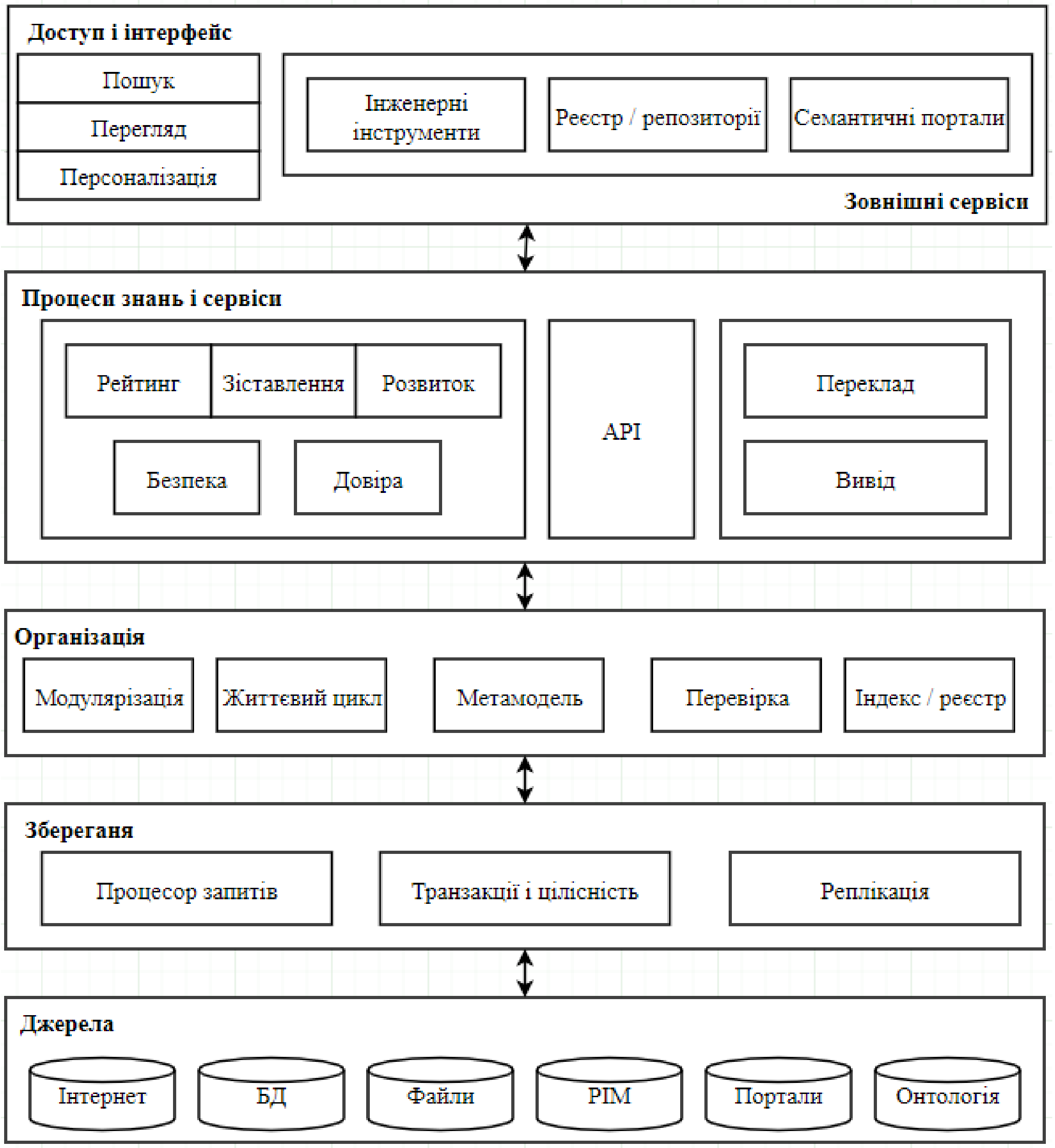


Рисунок 2.3 – Концептуальна структура сховища онтологій

В даний час повторне використання онтологій для академічних і промислових кіл ускладнює те, що більшість онтологій не супроводжується додатковою інформацією.

Це пов'язано з відсутністю єдиного загальноприйнятого стандарту подання відомостей про онтології – як про її формальних характеристиках, так і про семантику представлених в ній знань. Тому і виникає необхідність у виробленні вимог до стандарту метаданих для онтологій, аналіз яких забезпечить ефективний пошук і повторне використання створених раніше онтологічних знань.

## 2.3 Стандарти подання метаданих про онтології

Репозиторії онтологій потребують розробки та використання стандартів представлення метаданих про онтології. Вже розроблено стандарт метаданих OMV (Ontology Metadata Vocabulary) [36], який використовується в децентралізованому репозиторії онтологій Oyster. OMV являє собою модель метаданих для онтологій і пов'язаних з ними сутностей, що відображає ключові аспекти інформації про метадані онтології, наприклад, її походження і доступності. OMV реалізований у вигляді онтології на мові OWL і являє собою перший стандарт метаданих для онтології. Елементи метаданих в онтології OMV моделюються або за допомогою класів і екземплярів онтології, або за допомогою властивостей класів (вибір визначається складністю відповідного елемента метаданих). Якщо значення елементів метаданих можна легко зіставити зі звичайними типами даних (числами, літералами, списками значень), то елемент метаданих, як правило, представляється у вигляді DatatypeProperty, а більш складні елементи метаданих моделюються за допомогою додаткових класів, пов'язаних з ObjectProperties. В OMV для опису онтологій використовуються наступні терміни:

– метадані – дані про дані;

– метадані онтології – метадані, які надають інформацію про онтологію;

– онтологія метаданих – онтологія, що представляє метадані;

– елемент метаданих – елемент схеми метаданих (таблиця 2.1);

– OMV (Ontology Metadata Vocabulary) – скорочення для позначення пропонованої схеми метаданих онтології;

– категорії метаданих – в OMV розрізняють наступні три категорії метаданих для опису онтологічного контенту:

– обов'язкові елементи метаданих (відсутність будь-якого обов'язкового запису призводить до неповного опису онтології);

– додаткові метадані;

– розширені метадані, які не входять в ядро схеми метаданих.

Таблиця 2.1 – Шаблон для елемента метаданих в OMV

| | |
|---|---|
| Name | Ім'я елемента (entity) метаданих |
| Type | Тип онтологічного примітиву, використовуваного для подання елемента в OWL: Class, ObjectProperty або DatatypeProperty |
| Identifier | Унікальний ідентифікатор, який використовується для цього елемента |
| Occurrence Constraint | Один з наступних: required, optional або extensional |
| Category | Залежна від контенту категорія, до якої відноситься елемент |
| Definition | Коротке визначення мети, яка може бути детально описана в тегах коментарів |
| Domain | Предметна галузь елемента OMV (для властивостей OWL) |
| Range | Ранг елемента OMV entity (для властивостей OWL) |
| Cardinality | Потужність елемента OMV (MIN: MAX) |
| OMV version | Версія OMV, в якій представлений елемент |
| Comments | Детальний опис елемента |

В онтології OMV представлені такі класи, як Ontology, OntologyType, KnowledgeRepresentationParadigm, FormalityLevel, OntologyTask, Location тощо, які дозволяють описати різні аспекти можливого застосування онтологій, що містяться в репозиторії.

Для ефективного використання сховища онтологій не менш важливі, ніж засоби опису онтологій, що зберігаються в репозиторії, ті процеси і сервіси для обробки накопичених в репозиторії знань, які надаються користувачеві. До базових сервісів репозиторіїв онтологій можна віднести кошти рейтинг-кваліфікації, зіставлення і оцінки онтологій, забезпечення безпеки їх використання.

Засоби виставлення рейтингу дозволяють кожному користувачеві сховища (а не тільки експертам), застосовуючи різні метаоцінки, висловити свою думку про представлених у ньому онтологіях з точки зору їх корисності для інших користувачів. На підставі цих метаоцінок може бути обчислена мережа довіри до онтологічних елементів [37]. При цьому кожен елемент оцінюється тільки в цілому, а не його специфічні властивості окремо, що недостатньо для аналізу такого складного контенту, як онтології [38].

Засоби зіставлення онтологій, що зберігаються в репозиторії, дозволяють забезпечити взаємодії між різними моделями знань, представленими в них.

На відміну від рейтинг-кваліфікації (rating) онтологій, яка є досить суб'єктивною, оцінювання (evaluation) онтологій можна розглядати як оцінювання якості та адекватності онтології або її частин щодо конкретних намірів, мети або контексту. На сьогодні існує кілька методів оцінювання онтологій. Обрані стратегії оцінювання можуть бути реалізовані у відповідній компоненті сховища та застосовуватися при пошуку онтологій із заданими властивостями.

У зв'язку з правами інтелектуальної власності, комерційними ліцензіями, патентами або авторськими правами не всі артефакти знань можуть бути доступні громадськості. Тому в репозиторії повинні бути присутніми явний контроль доступу і функції управління правами. У той час як доступ до знань може бути обмежений, метазнання в OMV [39] залишаються доступними. В результаті цього артефакти знань, що комерційно використовуються можуть бути ідентифіковані в репозиторії, а доступ до них забезпечується спеціалізованими сервісами, такими як платіжні системи.

# 3 РОЗРОБКА МЕТОДІВ ПОВТОРНОГО ВИКОРИСТАННЯ ПРОГРАМНОГО КОДУ

## 3.1 Дослідження методів відображення онтологій

Створення промислових систем, заснованих на онтологіях, вимагає методів і інструментів, як для побудови онтологій, так і для цілого ряду задач, пов'язаних з їх супроводом. Для побудови онтологій з середини 90-х років минулого століття почали створюватися середовища розробки онтологій. У наступні роки паралельно з розвитком засобів розробки онтологій з'явилися засоби редагування і супроводу онтологій, засоби відображення, вирівнювання і об'єднання онтологій, а також засоби анотування онтологій. Таким чином, до теперішнього часу сформувалася ціла інженерія онтологій [40].

Одним з центральних понять інженерії онтологій є поняття «відображення онтологій» (ontology mapping), під яким розуміється діяльність по встановленню відповідності між декількома онтологіями або, іншими словами, знаходження семантичних зв'язків подібних елементів з різних онтологій. З найбільш загальної точки зору важливість задачі відображення онтологій обумовлена тим фактом, що потужність знань, укладених в онтологіях, проявляється в повній мірі тільки в тому випадку, коли вдається врахувати взаємозв'язку незалежних онтологій – встановлення факту подібності сутностей в різних онтологіях означає вилучення з цих онтологій додаткових знань.

У процедурі відображення онтологій можна виділити два етапи:

- локальне відображення сутностей – незалежне встановлення відповідностей між двома сутностями, що розглядаються онтологій;
- глобальне відображення сутностей – перегляд (перерахунок) локальних відображень з урахуванням відображень всіх інших елементів.

Близькою до проблеми відображення онтологій є проблема вирівнювання онтологій (ontology alignment), яка полягає в тому, щоб встановити різні види відповідності між двома онтологіями, а потім зберегти вихідні онтології разом з інформацією про знайдені відповідності з тим, щоб надалі використовувати інформацію про взаємозв'язки онтологій. Також варто відзначити, що на основі відображення онтологій вирішується задача інтеграції онтологій (ontology merging) – задача створення нової онтології або її фрагментів з двох і більше вихідних онтологій.

Компоненти, з яких складається онтологія, залежать від моделі онтології. Зазвичай онтологія описується за допомогою:

– концептів (понять, класів, сутностей, категорій);
– атрибутів концептів (слотів, властивостей, ролей);
– відносин між концептами (зв'язків, залежностей, функцій);
– додаткових обмежень (аксіом, фасет).

Елементи предметної області (елементи даного концепту) називаються екземплярами. Залежність між концептами, яка включає в себе необхідну умову і наслідок виконання цієї умови, називається правилом. Онтологія разом з множиною відповідних примірників складає базу знань.

Подоба деяких сутностей  визначається за допомогою функції подібності , яка має такі властивості:

– (об'єкти x, y ідентичні);
– (об'єкти x, y абсолютно різні і не мають схожих характеристик);
– (властивість зворотності функції подібності);
– (властивість симетричності функції подібності).

Нехай  – онтології, що розглядаються,  – деяка сутність онтології   . Подоба сутностей  , означає, що  , де  – гранична величина (рівень подібності, рівень відсікання).

Відображення онтології  на онтологію  означає спробу знайти для кожного з концептів онтології подібний до нього концепт в онтології . Іншими словами, відобразити онтологію  на онтологію означає відобразити кожну з сутностей , онтології  на відповідну сутність , онтології .

Будемо писати  та говорити, що сутність  є відображення сутності , якщо сутності , подібні, тобто якщо  ,. Тут  – функція відображення.

Якщо онтологія  є відображення онтології , то цей факт будемо записувати в вигляді .

Для вирішення задачі відображення онтологій необхідно розглянити критерії та мультикритерії подібності онтологій. Розглянуті критерії подібності онтологій побудовані на основі подібності сутностей відповідних семантичних мереж, дескриптивної логіки, обмежень і правил та інше [41].

Існують наступні групи критеріїв:

– критерії на основі подібності ідентифікаторів або URI сутностей;
– критерії на основі семантичної мережі онтології;
– критерії на основі дескриптивної логіки;
– критерії на основі обмежень;
– критерії на основі правил;
– критерії, що враховують специфіку словника програми;
– мультикритерії.

Першим із критеріїв класу критеріїв на основі подібності ідентифікаторів є критерій на основі ідентифікаторів сутностей (міток). Критерій формулюється так: якщо мітки двох сутностей подібні, то ці сутності подібні ().

Замість імен сутностей можуть порівнюватися імена їх синонімів (з використанням існуючих словників лексики або тезаурусов). За допомогою відповідного словника даний критерій подібності може бути використаний для порівняння сутностей, заданих на різних мовах.

Прикладом роботи є: нехай є два концепти з ідентифікаторами «комп'ютерна мишка» і «комп'ютерна миша» (рисунок 3.1).

```
<owl:Class rdf:ID=”id1”>
	<rdfs:label>комп’ютерна мишка<label/>
</owl:Class>
<owl:Class rdf:ID=”id2”>
	<rdfs:label>комп’ютерна миша<label/>
</owl:Class>
```

Рисунок 3.1 – Приклад 1

Порівняння цих ідентифікаторів будь-яким з методів порівняння міток покаже подібність зазначених концептів.

Наступний критерій на основі подібності ідентифікаторів будується на основі ідентифікаторів сутностей, унікальних для кожної з сутностей, наприклад, URI (Uniform Resource Identifier) і формулюється так: якщо дві сутності мають подібні URI, то ці сутності подібні ().

Прикладом роботи є: у онтології визначається концепт «Регіон», у цій же онтології є поняття «РегіонЦентральногоПобережжя» (рисунок 3.2).

```
<owl:Class rdf:ID="Регіон"/>
<owl:Class rdf:ID=»РегіонЦентральногоПобережжя»/>
```

Рисунок 3.2 – Приклад 2

Відповідно до критерію концепт «Регіон» і концепт "РегіонЦентральногоПобережжя" подібні.

Оскільки суті складаються у відносинах з іншими сутностями через свої атрибути, має місце наступний критерій подібності, на основі семантичної мережі онтології: якщо атрибути двох сутностей подібні, то ці сутності подібні ().

Прикладом є: примірники «КабернеСовіньон» і «ВіноградКабернеСовіньон» мають однаковими атрибутами і тому подібні (рисунок 3.3).

```
<Регион rdf:ID="РегионГорыСантаКруз">
        <locatedIn rdf:resource="#РегионКалифорния"/> </Регион>
<Винодельня rdf:ID="ВиноградникГораСантаКруз"/>
<КабернеСовиньон
        rdf:ID="КабернеСовиньонВинодельняГораСантаКруз">
                <расположенВ rdf:resource="#РегионГорыСантаКруз"/>
        rdf:resource="#ВиноградникГораСантаКруз"/> </КабернеСовиньон>
<ВиноградКабернеСовиньон
        rdf:ID="ВиноградКабернеСовиньонВинодельняГораСантаКруз">
                <расположенВ rdf:resource="#РегионГорыСантаКруз"/>
        rdf:resource="#ВиноградникГораСантаКруз"/> </ВиноградКабернеСовиньон>
```

Рисунок 3.3 – Приклад 3

Обмежити відносини між сутностями можна шляхом завдання області застосування і діапазону відповідних атрибутів сутностей. Ця обставина дозволяє сформувати наступний критерій на основі семантичної мережі: якщо область застосування і діапазон двох відносин подібні, то такі відносини подібні ().

Відносини «ЗробленоІзВінограду», «ОтриманоІзВінограду» мають однакові області застосування і діапазони і тому подібні (рисунок 3.4).

```
<owl:ObjectProperty rdf:ID="ЗробленоІзВінограду">
          rdfs:domain rdf:resource="#Вино"/>
          <rdfs:range rdf:resource="#Виноград"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ОтриманоІзВінограду">
           <rdfs:domain rdf:resource="#Вино"/>
           <rdfs:range rdf:resource="#Виноград"/>
</owl:ObjectProperty>
```

Рисунок 3.4 – Приклад 4

Критерії на основі дескриптивної логіки охоплюють онтології, які володіють складністю, достатньою для їх опису дескриптивними логіками. Наведемо, перш за все, приклад дескриптивної логіки: «розумні люди», «книги» і «розумна людина» – концепти; «читають» – відношення; «Сергій» – екземпляр. Примірник «Сергій» успадковує ставлення батьківського поняття. Той же приклад на мові OWL має вигляд наведений на рисунку 3.5.

```
<Owl: Class rdf: ID = "література" />
<Owl: Class rdf: ID = "книги">
          <Rdfs: subClassOf rdf: resource = "# література" /> </ Owl: Class>
<Owl: Class rdf: ID = "розумні люди">  <Owl: Restriction>
          <Owl: onProperty rdf: resource = "# читають" />
          <Owl: hasValue rdf: resource = "# книги" /> </ Owl: Restriction> </ Owl: Class>
<Owl: Thing rdf: ID = "Сергій"> <Rdfs: type rdf: resource = "# розумні люди" /> </ Owl: Thing>
```

Рисунок 3.5 – Приклад 5

Критерій  формується, виходячи з посилу, що подібні концепти з великою ймовірністю мають подібні батьківські поняття: якщо батьківські поняття двох концептів подібні, то самі концепти також подібні. Приклад роботи наведено на рисунку 3.6.

```
<Owl: Class rdf: ID = "Напій"> <Rdfs: subClassOf rdf: resource = "#ПродуктХарчування" /> </ Owl: Class>
<Owl: Class rdf: ID = "Їжа"> <Rdfs: subClassOf rdf: resource = "#ПродуктХарчування" /> </ Owl: Class>
```

Рисунок 3.6 – Приклад 6

Батьківські поняття концептів «Напій» і «Їжа» збігаються, тому зазначені концепти подібні.

Критерій  заснований на подобі дочірніх понять: якщо дочірні поняття порівнюваних концептів подібні,  то ці концепти також подібні. Приклад роботи наведено на рисунку 3.7.

```
<Owl: Class rdf: ID = "Їжа" />
<Owl: Class rdf: ID = "Страва" />
<Owl: Class rdf: ID = "Хліб">
        <Rdfs: subClassOf rdf: resource = "#Їжа" />
        <Rdfs: subClassOf rdf: resource = "#Страва" /> </ Owl: Class>
```

Рисунок 3.7 – Приклад 7

Концепти «Їжа» і «Страва» мають одне і теж дочірнє поняття «Хліб» і тому подібні.

Критерій  засновано на подобі концептів, що відносяться до того ж рівня ієрархії понять: якщо концепти мають подібні концепти того ж рівня ієрархії, то вони також подібні. Приклад роботи наведено на рисунку 3.8.

```
<Owl: Class rdf: ID = "Автомобіль" /> <Owl: Class rdf: ID = "Машина" />
<Owl: Class rdf: ID = "Порш" />
        <Rdfs: subClassOf rdf: resource = "# Автомобіль" /> </ Owl: Class>
<Owl: Class rdf: ID = "Жигулі" />
        <Rdfs: subClassOf rdf: resource = "# Автомобіль" />
        <Rdfs: subClassOf rdf: resource = "# Машина" /> </ Owl: Class>
<Owl: Class rdf: ID = "Мерседес" />
        <Rdfs: subClassOf rdf: resource = "# Машина" /> </ Owl: Class>
```

Рисунок 3.8 – Приклад 8

Концепти «Порш» і «Мерседес» подібні, оскільки кожен з цих концептів має подібний концепт «Жигулі» того ж рівня (рисунок 3.9).

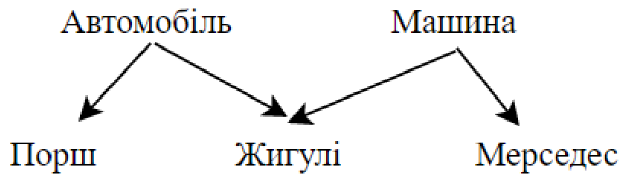


Рисунок 3.9 – Візуалізація прикладу 8

Критерії , будуються на основі подібності атрибутів дочірніх і батьківських сутностей: якщо подібні атрибути дочірніх сутностей, то атрибути батьківських сутностей також подібні (); якщо подібні атрибути батьківських сутностей, то атрибути дочірніх сутностей також подібні ().

Приклад роботи наведено на рисунку 3.10.

```
<Owl: Class rdf: ID = "ХарактеристикаВина" /> <Owl: Class rdf: ID = "КолірВина">
        <Rdfs: subClassOf rdf: resource = "# ХарактеристикаВина" /> </ Owl: Class>
<Owl: ObjectProperty rdf: ID = "МаєХарактеристикуВина">
        <Rdfs: domain rdf: resource = "# Вино" />
        <Rdfs: range rdf: resource = "#ХарактеристикаВина" /> </ Owl: ObjectProperty>
<Owl: ObjectProperty rdf: ID = "МаєКолір">
        <Rdfs: subPropertyOfrdf: resource = "#ВолодієХарактеристикоюВина" />
        <Rdfs: range rdf: resource = "# КолірВина" /> </ Owl: ObjectProperty>
<Owl: ObjectProperty rdf: ID = "МаєВідтінок">
        <Rdfs: subPropertyOf rdf: resource = "#ВолодієХарактеристикоюВина" />
        <Rdfs: range rdf: resource = "#КолірВина" /> </ Owl: ObjectProperty>
```

Рисунок 3.10 – Приклад 9

Атрибути «МаєКолір» і «МаєВідтінок» подібні, оскільки подібні їх батьківські атрибути.

Оскільки, як зазначалося вище, екземпляр являє собою елемент відповідного концепту, він успадковує всі атрибути цього концепту. Тому має місце наступний критерій подібності: якщо концепти включають в себе подібні екземпляри, то ці концепти подібні (). Приклад роботи наведено на рисунку 3.11.

```
<Owl: Class rdf: ID = "Автомобіль" /> <Owl: Class rdf: ID = "Машина" />
<Owl: Thing rdf: ID = "ПоршСергія"> <Rdfs: type rdf: resource = "# Автомобіль" />
        <Rdfs: type rdf: resource = "# машина" /></ Owl: Thing>
<Owl: Thing rdf: ID = "МерседесСергія"> <Rdfs: type rdf: resource = "# Автомобіль" />
        <Rdfs: type rdf: resource = "# Машина" /></ Owl: Thing>
```

Рисунок 3.11 – Приклад 10

Концепти «Автомобіль» і «Машина» включають в себе однакові екземпляри і тому подібні.

Критерій  є зворотним по відношенню до критерію  і записується в вигляді: якщо екземпляри належать подібним концептів, то ці екземпляри подібні.

Приклад роботи наведено на рисунку 3.12.

```
<Owl: Class rdf: ID = "Автомобіль" />
<Owl: Thing rdf: ID = "ПоршСергія">
<Rdfs: type rdf: resource = "# Автомобіль" /> </ Owl: Thing>
<Owl: Thing rdf: ID = "МерседесСергія">
<Rdfs: type rdf: resource = "# Автомобіль" /> </ Owl: Thing>
```

Рисунок 3.12 – Приклад 11

Примірники «ПоршСергія» і «МерседесСергія» подібні, оскільки належать одного концепту «Автомобіль».

Критерій  близький до критерію подібності  і формулюється так: якщо концепти мають схожу малу / велику частина примірників, ці поняття подібні. На відміну від критерію , подоба в даному критерії визначається подібністю структури дочірніх елементів, а не рівнем подібності самих елементів.

Приклад роботи наведено на рисунку 3.13.

```
<Owl: Class rdf: ID = "Автомобіль" /> <owl: Class rdf: ID = "Машина" />
<Owl: Thing rdf: ID = "ПоршСергія">
        <Rdfs: type rdf: resource = "# Автомобіль" />
        <Rdfs: type rdf: resource = "# Машина" /> </ owl: Thing>
<Owl: Thing rdf: ID = "МерседесСергія">
        <Rdfs: type rdf: resource = "# Автомобіль" />
        <Rdfs: type rdf: resource = "# Машина" /> </ owl: Thing>
```

Рисунок 3.13 – Приклад 12

Концепти «Автомобіль» і «Машина» включають в себе частину однакових примірників ( «ПоршСергія», «МерседесСергія») і тому подібні.

Критерій  має наступне формулювання: якщо два примірника пов'язані з деяким іншим екземпляром подібними відносинами,  то ці екземпляри подібні.

Приклад роботи наведено на рисунку 3.14.

```
<Owl: ObjectProperty rdf: ID = "РозташованийВ">
        <Rdf: type rdf: resource = "& owl; TransitiveProperty" />
        <Rdfs: domain rdf: resource = "& owl; Thing" />
        <Rdfs: range rdf: resource = "# Регіон" /> </ Owl: ObjectProperty>
<Регіон rdf: ID = "РегіонГориСантаКруз"> <РасположенВ rdf: resource = "# РегіонСША" /> </ Регіон>
<Регіон rdf: ID = "РегіонКаліфорнія"> <РасположенВ rdf: resource = "# РегіонСША" /> </ Регіон>
```

Рисунок 3.14 – Приклад 13

Примірники «РегіонГориСантаКруз» і «РегіонКаліфорнія» пов'язані з екземпляром «РегіонСША» ставленням «РозташованийВ» і тому подібні.

Критерій  є зворотним по відношенню до критерію  і формулюється наступним чином: якщо певний стосунок пов'язує екземпляри з одним і тим же примірником, то порівнювані відносини можуть бути подібні (рисунок 3.15).

```
<Owl: ObjectProperty rdf: ID = "РозташованийВ">
        <Rdf: type rdf: resource = "& owl; TransitiveProperty" />
        <Rdfs: domain rdf: resource = "& owl; Thing" />
        <Rdfs: range rdf: resource = "# Регіон" /> </ Owl: ObjectProperty>
<Owl: ObjectProperty rdf: ID = "ЗнаходитьсяВ">
        <Rdf: type rdf: resource = "& owl; TransitiveProperty" />
        <Rdfs: domain rdf: resource = "& owl; Thing" />
        <Rdfs: range rdf: resource = "# Регіон" /> </ Owl: ObjectProperty>
<Регіон rdf: ID = "РегіонГориСантаКруз"> <РозташованийВ rdf: resource = "# РегіонСША" /> </ Регіон>
<Регіон rdf: ID = "РегіонКаліфорнія"> <ЗнаходитьсяВ rdf: resource = "# РегіонСША" /> </ Регіон>
```

Рисунок 3.15 – Приклад 14

Тут екземпляри «РегіонГориСантаКруз» і «РегіонКаліфорнія» пов'язані з одним і тим же примірником «РегіонСША» відносинами «РозташованийВ» і «ЗнаходитьсяВ» відповідно. Тому зазначені відносини подібні.

Критерій  засновано на використанні відносин виду «SameClassAs» і «SameIndividualAs»: якщо дві сутності пов'язані між собою відношенням «sameClassAs» або відношенням «SameIndividualAs», то ці сутності подібні.

Приклад роботи наведено на рисунку 3.16.

```
<Вино rdf: ID = "Улюблена Вино Михайла">
        <Owl: sameAs rdf: resource = "# StGenevieve Техаське Біле" /></ Вино>
```

Рисунок 3.13 – Приклад 12

Критерій засновано на використанні відносин виду «EquivalentClass», «EquivalentProperty», «SameAs»: якщо дві сутності пов'язані між собою відносинами «EquivalentClass», «EquivalentProperty», «SameAs», то ці сутності подібні. Приклад роботи наведено на рисунку 3.17.

```
<owl:Class rdf:ID="БілеВіно">
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#Віно"/> <owl:Restriction>
                        <owl:onProperty rdf:resource="#МаєКолір"/>
                        <owl:hasValue rdf:resource="#Біле"/>
                </owl:Restriction> </owl:intersectionOf> </owl:Class>
<owl:Class rdf:ID="Шампанське">
         <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Віно"/> <owl:Restriction>
                        <owl:onProperty rdf:resource="#МаєКолір"/>
                        <owl:hasValue rdf:resource="#Біле"/>
                </owl:Restriction> </owl:intersectionOf> </owl:Class>
```

Рисунок 3.17 – Приклад 16

У цьому прикладі «БілеВіно» і «Шампанське» мають подібні обмеженнями ( «МаєКолір» «Біле»).

Подібними називаються правила, які мають подібні умови, а також подібні слідства виконання правила. Критерій на основі правил має вигляд: якщо дві сутності пов'язані між собою подібними правилами, то ці сутності подібні ().

Приклад роботи даного критерію представлено в таблиці 3.1.

Таблиця 3.2 – Приклад роботи критерію

| Правило A | Правило B | Слідство |
|---|---|---|
| Якщо – брат ) і ( – батько ), то ( – батько ) | Якщо – сестра ) і ( – батько ), то ( – батько ) | З A і B випливає, що відносини «брат» і «сестра» подібні |

Цей же приклад на мові OWL має вигляд (рисунок 3.17).

```
<ruleml:imp> <ruleml:_rlab ruleml:href="#Приклад1"/> <ruleml:_body>
 <swrlx:individualPropertyAtom swrlx:property="МаєБрата">
        <ruleml:var>x1</ruleml:var>
        <ruleml:var>x2</ruleml:var>  </swrlx:individualPropertyAtom>
 <swrlx:individualPropertyAtom swrlx:property="МаєОтця">
        <ruleml:var>x2</ruleml:var>
        <ruleml:var>x3</ruleml:var>  </swrlx:individualPropertyAtom> </ruleml:_body>
<ruleml:_head>  <swrlx:individualPropertyAtom swrlx:property=" МаєОтця ">
        <ruleml:var>x1</ruleml:var>
        <ruleml:var>x3</ruleml:var>  </swrlx:individualPropertyAtom> </ruleml:_head> </ruleml:imp>
<ruleml:imp> <ruleml:_rlab ruleml:href="#Приклад2"/> <ruleml:_body>
 <swrlx:individualPropertyAtom swrlx:property="МаєСестру">
        <ruleml:var>x1</ruleml:var>
        <ruleml:var>x2</ruleml:var>  </swrlx:individualPropertyAtom>
 <swrlx:individualPropertyAtom swrlx:property=" МаєОтця ">
        <ruleml:var>x2</ruleml:var>
        <ruleml:var>x3</ruleml:var>  </swrlx:individualPropertyAtom> </ruleml:_body> <ruleml:_head>
 <swrlx:individualPropertyAtom swrlx:property=" МаєОтця ">
        <ruleml:var>x1</ruleml:var>
        <ruleml:var>x3</ruleml:var>  </swrlx:individualPropertyAtom> </ruleml:_head> </ruleml:imp>
```

Рисунок 3.18 – Приклад 17

Відносини «МаєБрата» і «МаєСестру» подібні.

Розглянуті вище критерії подібності онтологій використовують властивості загального визначення онтологій. Крім цього, можливі онтології, які використовують особливий словник. Якщо цей словник строго визначений і загальнодоступний, то він також може бути використаний для формування критеріїв подібності онтологій.

Прикладом є SWAP-системи [42], в яких кожному файлу присвоюється унікальний хеш-код. Для таких систем мають місце критерії , : якщо хеш-коди двох елементів однакові, то і елементи подібні (); файли однакового MIME-type подібні, як мінімум, по формату ().

Зведення всіх 19 розглянутих критеріїв подібності онтологій представлено в таблиці 3.2.

Таблиця 3.2 – Зведення критеріїв подібності онтологій

| Формулювання критерію | Критерій | Що? |
|---|---|---|
| Поняття подібні, якщо | | мітки |

| | | |
|---|---|---|
| подібні <Що?> | | URI |
| | | властивості |
| | | батьківські поняття |
| | | поняття того ж рівня |
| | | дочірні поняття |
| | | екземпляри |
| | | вибірка примірників |
| | | відношення «sameAs» |
| | | обмеження |
| | | правила |
| Атрибути подібні, якщо подібні <Що?> | | мітки |
| | | URI |
| | | область і вектор |
| | | батьківські атрибути |
| | | дочірні атрибути |
| | | пов'язані екземпляри |
| | | відношення «sameAs» |
| | | обмеження |
| | | правила |

Кінець таблиці 3.2

| Формулювання критерію | Критерій | Що? |
|---|---|---|
| Примірники подібні, якщо подібні <Що?> | | мітки |
| | | URI |
| | | батьківське поняття |
| | | властивості і екземпляри |
| | | відношення «sameAs» |
| | | обмеження |
| | | правила |
| | | хеш-коди |
| | | MIPE-type |

Велика кількість мультикритеріїв, побудованих на основі розглянутих критеріїв подібності онтологій, запропоновано в роботі [43]. Найчастіше в якості мультикритерію подібності використовується адитивна згортка критеріїв - – тобто їх зважена сума

(3.1)

де – сутності онтологій відповідно; – ваговий коефіцієнт критерію .

Широко відома модифікація адитивної згортки (3.1) заснована на використанні сигмоидальной функції від критеріїв подібності -

(3.2)

тут – вільний параметр функції. Використання сигмоидальной функції в критерії подібності (3.2) дозволяє підвищити ваги критеріїв, що мають великі значення, і практично знехтувати критеріями з малими значеннями.

Значний інтерес представляє варіант побудови мультикритеріїв подібності на основі навченою нейронної мережі. Так в роботі [44] мультикритерій синтезується за допомогою тришарового персептрону нейронної мережі, в якому в прихованому шарі використовуються нейрони з функцією активації типу гіперболічний тангенс, а в вихідному шарі – типу сигмоїда.

Як зазначалося вище, сутності подібні, якщо , де – рівень відсікання. Існують наступні методи вибору величини :

– метод константи. У цьому методі для критерію подібності як рівня відсікання приймається деяка константа [1:19]. Значення цієї константи визначається на основі арифметичного середнього для величини критерію подібності або на основі експертних оцінок;

– метод допустимої помилки. Тут в якості рівня відсікання приймається різниця між максимальним значенням критерію подібності і деякою константою с, що визначає похибка подібності

[1:19]; (3.3)

– метод допустимої похибки. Метод аналогічний попередньому методу і відрізняється від нього лише тим, що в якості константи q с використовується величина, яка дорівнює деякої фіксованої частини від максимального значення критерію подібності:

[1:19]. (3.4)

Оскільки багато з розглянутих критеріїв подібності двох сутностей ґрунтуються на подобі інших пар сутностей, відображення онтологій є ітераційним процесом. При цьому на першій ітерації повинен використовуватися критерій подібності, який не ґрунтується на інших умовах (наприклад, критерій на основі подібності міток ).

У зв'язку з ітераційним характером процедури відображення онтологій виникає питання про вибір необхідної кількості ітерацій. Можливі наступні варіанти такого вибору:

- кількість ітерацій є заздалегідь фіксованою;
- ітерації продовжуються до тих пора, поки не припиняться зміни в відображенні;
- ітерації продовжуються до тих пора, поки зміни у відображенні не стануть нижче деякого заданого порогу відсікання;
- ітерації продовжуються до тих пора, поки не будуть перевищені допустимі ресурси часу і обчислювальної потужності.

Через ітераційний характер процедури відображення онтологій виникає ще одне питання: чи слід виробляти будь-яку обробку результатів даної ітерації перед їх перенесенням в наступну ітерацію? Тут можна виділити два підходи. По перше, на наступній ітерації можна розглядати тільки найкращі відображення (відображення з найбільшим значенням відповідного критерію подібності).

По-друге, на даній ітерації отримані відображення можна ранкувати – кращому відображенню призначити вагу 1, відображенню з наступним за значенням критерію подібності – вага , наступному відображенню – вагу . Останній підхід дозволяє з великою ймовірністю зберегти правильні відображення і, в той же час, в разі необхідності замінити ці відображення наступними за рівнем відповідності.

## 3.2 Розробка методу відображення онтологій

Одним з основних напрямків розвитку сучасних інтелектуальних інформаційних систем (ІІС) є розробка нових підходів уявлення і інтеграції знань. ІІС містять різнорідні бази знань, які мають власні локальні інформаційні моделі, підтримувані різними стандартами представлення даних і знань. Як наслідок, на етапі інтегрування різнорідних даних виникає безліч конфліктів: використовується різна термінологія при позначенні семантично близьких понять ПрО, існує неоднорідність специфікацій концептів тощо. Рішення проблеми інтеграції неоднорідного інформації можливо шляхом специфікації конкретної ПрО з подальшою перевіркою и відображенням розуміння різних ПрО.

В даний час для опису ПрО широко застосовується онтологічний підхід, оскільки онтологічні специфікації дозволяють задавати точну семантику прикладної області об'єктів н визначати їх контекст. Створення загальнодоступних онтологій ПрО дозволяє вирішити проблему неоднорідності онтологічних специфікацій для певних груп агентів. Однак в умовах відкритого інформаційного простору при вирішенні задач пред'являються різні вимоги до глибини і ступеня формалізація опису ПрО, використовуються неоднорідні онтологічні опису ПрО. Неоднорідність онтологічних специфікацій з'являється на рівнях модельної і понятійної семантики.

Аналіз літератури [45-48] дозволяє зробити висновок, що завдання об'єднання (узгодження, вирівнювання, відображення тощо) неоднорідних онтологічних моделей відносяться до класу NP-важких задач оптимізації та можуть бути вирішені із застосуванням еволюційних алгоритмів пошуку оптимальних рішень. Свою ефективність вже довели, наприклад, генетичні, мурашині, бджолині алгоритми і алгоритми, засновані на еволюційному моделюванні.

В роботі для вирішення задачі відображення онтологій запропоновано використовувати один з еволюційних методів оптимізації – метод рою частинок (МРЧ), який дозволяє автоматично визначати вагові коефіцієнти семантичної близькості концептів двох онтологій.

Нехай задані дві онтології  і , що формалізують семантику деякої ПрО. Причому, елементи онтологій схильні до зміни з плином часу. Зміни носять дискретний характер і вносяться відповідно до синтаксису вибраної мови представлення онтологій. Узагальнивши ряд визначень, які розглядають онтологію з точки зору структури і елементів, що становлять онтологію, її можна уявити як:

(3.5)

де O – онтологія ПрО; C – множина понять (класів), визначених для конкретної Пр0; P – множина властивостей понять; R – множина відносин, визначених між поняттями в C.

Відображення онтології  на онтологію  означає знаходження для кожного елемента множин , що складають онтологію , подібних елементів з множин онтології . Відображенням двох онтологій є множина

(3.6)

де:  – елементи множин С, – множина відносин; k – ваговий коефіцієнт, що

відображає ступінь впевненості щодо коректності конкретного відображення.

Процес відображення онтологій може бути заданий функцією F, на вхід якої надходять дві онтології  і , а на виході отримується  – матриця заходів близькості S, елементами якої () є величини міри близькості між i-м елементом онтології  и j-м елементом онтології . Семантичної близькістю елементів називається смислова схожість цих елементів. Семантична близькість може визначатися між різними компонентами триплетів. При цьому в якості базової близькості можна розглядати близькість між елементами онтологій (класами, предикатами, термінами).

Процедура відображення полягає в знаходженні семантичних зв'язків між концептами різних онтології. Отже, центральним завданням відображення онтологій є обчислення заходи семантичної близькості – кількісної оцінки семантичної схожості онтологічних сутностей. Міра семантичної близькості показує високі значення для пар сутностей, які знаходяться в семантичних відносинах (синонімія, гіпонімія, асоціативність, когіпонімія), і нульові значення для всіх інших пар. Як правило, міра близькості між сутностями різних онтологій визначається на декількох рівнях: лексична близькість, близькість атрибутів і відносин, близькість примірників понять і інші. Розрахунок близькості між сутностями в різних онтологіях є ітераційним процесом, оскільки багато заходів близькості двох сутностей ґрунтуються на близькості інших сутностей (понять, властивостей, примірників). Очевидно, чим повніше враховуються характеристики двох сутностей, тим якісніше є міра близькості. Таким чином, комплексні заходи близькості, що поєднують кілька підходів, є найбільш перспективними. Комплексна міра близькості розраховується як середнє арифметичне значення заходів близькості з урахуванням вагового коефіцієнта:

(3.7)

де – міра близькості за певним критерієм; - ваговий коефіцієнт, який визначає важливість заходів семантичної близькості (сума ваг дорівнює 1, ); n – число міри близькості.

Вагові коефіцієнти можуть визначатися або експертами, або автоматично за допомогою навченою нейронної мережі або еволюційного алгоритму. У разі автоматичного визначення вагових коефіцієнтів проблема відображення онтологій може бути вирішена шляхом знаходження оптимальних вагових коефіцієнтів заходів семантичної близькості концептів. Таким чином, завдання відображення онтологій може бути вирішена як завдання оптимізації:

(3.8)

де – множина відображень; – множина всіх можливих вагових коефіцієнтів, які використовувалися для обчислення комплексного відображення; – цільова функція, що дозволяє оцінити якість множини вагових коефіцієнтів ; – критерії повноти і точності відображення m, відповідно.

Точність (precisioп) і повнота (recall) є метриками, які використовуються при оцінці відображення онтологій Повнота обчислюється як:

(3.9)

Точність обчислюється як:

, (3.10)

де – знайдені релевантні відображення; – загальне число релевантних відображень.

Сформулюємо задачу багатокритеріальної оптимізації відображення онтологій:

(3.11)

де n – число відображень сутностей онтологій і (обчислених мір близькості); – ваговий коефіцієнт, який визначає важливість і-й заходи семантичної близькості.

Таким чином, цільова функція – це максимум параметрів точності і повноти.

Задача відображення онтологій вирішується як процес встановлення відображення двох онтологій на основі обчислення комплексної заходи семантичної близькості сутностей онтологій. Для її вирішення запропонований алгоритм, який включає в себе п'ять послідовно виконуваних етапів (рисунок 3.19).

На першому етапі обчислюються п'ять заходів семантичної близькості на лексичному, семантичному рівнях, рівні класів, рівні триплетів, таксономическом рівні. На лексичному рівні виконується оцінка семантичної близькості (редакційного відстані). На смисловому рівні використовується міра семантичної близькості, яка використовує схожість між словами в імені сутності (заснований на семантичній мережі WordNet). Для оцінки косинусной заходи семантичної близькості концепти онтології перетворюються в вектори (модель word2vec) і обчислюється її значення. Семантична близькість між триплетами обумовлена близькістю між їх компонентами. Вважається, що якщо відповідні компоненти двох триплетів є близькими за змістом, то ці триплети також є семантично близькими. Міра семантичної близькості на рівні таксономії обчислюється тільки для класу. Це прийом дозволяє виміряти різницю в локальній структурі об'єкта.

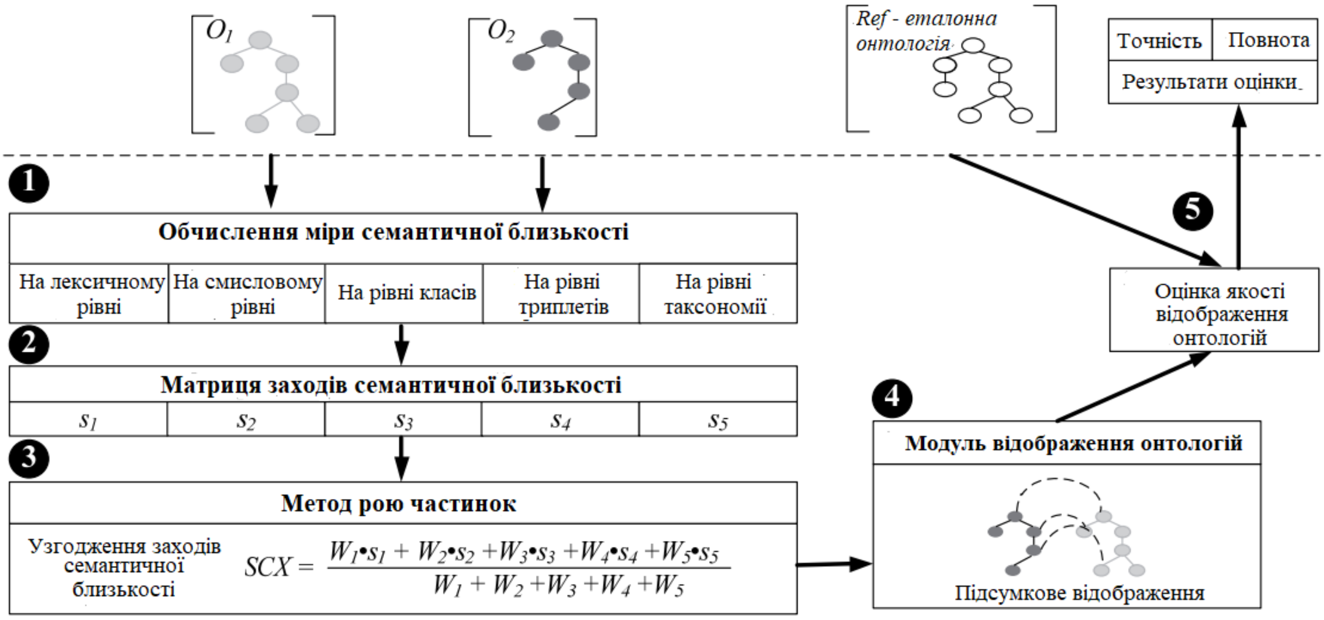


Рисунок 3.19 – Етапи відображення онтології з використання багатокритеріальної оптимізації методом рою частинок

Далі ініціалізується виконання алгоритму оптимізації МРЧ. З використанням формули (3.7) обчислюється комплексна міра семантичної близькості і визначається матриця комплексних заходів семантичної близькості. Для отриманої матриці застосовується алгоритм багатокритеріальної оптимізації МРЧ (3.11), обчислюються оптимальні вагові коефіцієнти заходів семантичної близькості і виконується підсумкове відображення онтологій.

Згідно МРЧ в процесі оптимізації підтримуються популяції можливих рішень, які називаються частками, які переміщаються в просторі рішень. Переміщення підкоряються принципу найкращого знайденого в цьому просторі положення, яке постійно змінюється при знаходженні частинками більш вигідних положень. Схема алгоритму оптимізації МРЧ приведена на малюнку 3.20.

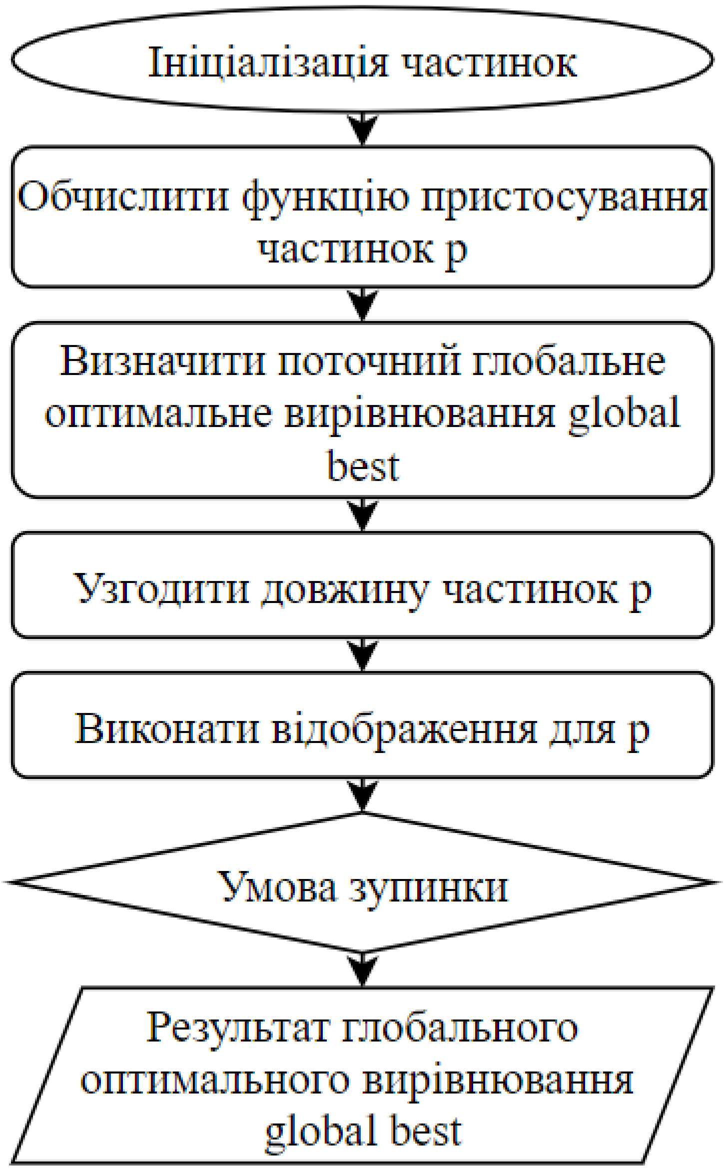


Рисунок 3.20 – Алгоритм оптимізації методом рою частинок

У нашому випадку в ролі частинок виступають вагові коефіцієнти семантичної близькості концептів. Ініціалізація процесу оптимізації МРЧ виконується за допомогою популяції випадкових частинок, після чого алгоритм виконує пошук оптимальних рішень шляхом безперервного поновлення популяцій. Відображення онтологій дискретним МРЧ визначається деяким числом N частинок (популяцією). У канонічному МРЧ на кожній ітерації для кожної частки застосовується вектор швидкості, який визначає її нове положення в просторі параметрів. Така еволюція відбувається за рахунок керованих випадкових повторних ітерацій в кожній частинці.

Ініціалізація. Популяція називається роєм, і вона складається з m відповідних рішень або частинок. Кожна частка має n клітин або позицій, що містять n вагових коефіцієнтів, що відповідають різним заходам подібності.

Наприклад, моделі кодування частинок з сімома осередками перетворюються в сім вагових коефіцієнтів (нормалізоване подання осередку) для семи заходів подібності (i-я частинка з сімома позиціями j, які можуть бути обчислені за формулою

(3.12)

Спочатку випадковим чином вибирається значення для кожного осередку в діапазоні від 0 до 1. Після того, як обрані початкові рої, розраховуються відповідні їм значення придатності. Первісна швидкість кожного осередку частки приймається рівною нулю. Вхідними даними запропонованої методики є: розмір рою; порогове значення .

Обчислення цільової функції. Згідно (3.11) використовуються критерії повноти (3.9) і точності (3.10) пошуку.

Створення рою наступного покоління. Операція здійснюється шляхом оцінки стану і швидкості частки. Кожна клітинка або позиція являє собою вагу (нормалізоване значення осередки) щодо заходів подібності. Осередки всередині частинки містять значення від 0 до 1, а швидкостям частинок задані нульові значення. Використовуючи результати попередньої стадії, положення кожної частинки і її швидкість оновлюються. Кожна частка відстежує кращу позицію, яку вона досягла. Найкраще становище серед усіх частинок називається глобальним кращим. Зміна швидкості частинки відбувається за наступним висловом:

, (3.13)

, (3.14)

де – тимчасова мітка, j-й кластер і частки; і – випадкові значення з діапазону від 0 до 1.

Далі обчислюється значення цільової функції кожної частки за формулою (3.11) і відбір частинок, які дають найбільші значення цільової функції.

## 3.3 Визначення експериментальної оцінки оптимізації відображення онтології

Запропонований алгоритм був реалізований в середовищі Matlab. В експерименті були використані дві онтології, структура яких наведена на рисунку 3.21.

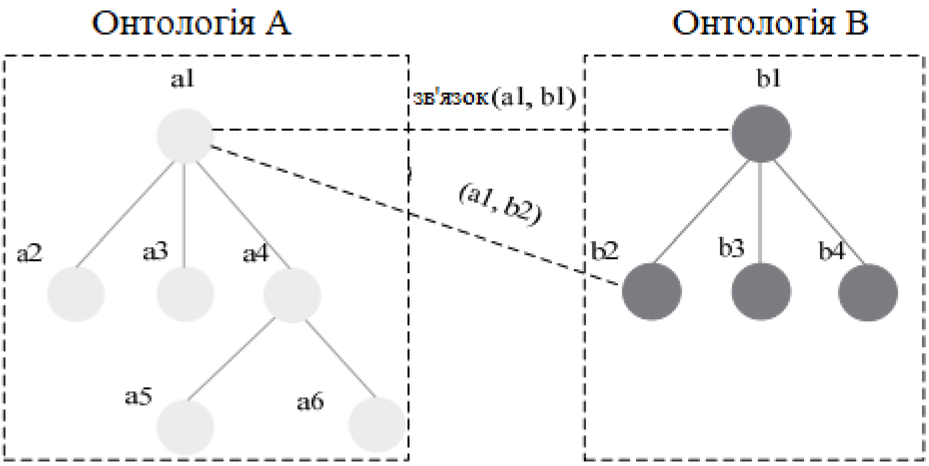


Рисунок 3.21 – Приклад відображення онтологій

Онтологія A має шість сутностей, а онтологія B – чотири сутності. Кожен елемент онтології A має зв'язок з усіма іншими елементами онтології B. Оскільки існують чотири зв'язки для кожного елемента онтології A, отже, в цілому маємо двадцять чотири парних зв'язки. На рисунку 3.21 умовно показані тільки зв'язку [a1, b1] і [a1, b2]. Значення подібності між a1 и b1 визначає ваговий коефіцієнт для зв'язку [a1, b1]. Результати порівняння пропонованого підходу з однокритеріальною оптимізацією (оптимізація була виконана окремо для критеріїв точності і повноти) наведені в таблиці 3.3.

Таблиця 3.3 – Результати одно і багатокритеріальної оптимізації

| Підхід | Точність | Повнота пошуку |
|---|---|---|
| Багатокритеріальна оптимізація | 0.81428 | 1.00 |
| Однокритеріальна оптимізація (точність) | 0.7142 | 0.86 |
| Однокритеріальна оптимізація (повнота) | 0.3333 | 1.00 |

За критерієм точності ефективність запропонованого підходу виявилася найвищою. За критерієм повноти пошуку пропонований метод також дозволив отримати високі результати.

# 4 ПРАКТИЧНА АПРОБАЦІЯ ОТРИМАНИХ НАУКОВИХ РЕЗУЛЬТАТІВ

Отримані в роботі результати можуть бути використані при проектуванні різних класів інтелектуальних систем, орієнтованих на спільне використання знань і забезпечення семантичної сумісністі різних моделей подання та обробки даних і знань.

Базовими онтологічними задачами є:

– класифікація понять;
– побудова таксономій з вхідними в них відносинами;
– інтеграція онтологічних структур;
– відображення і модифікація онтологій.

Як приклад розглянемо побудову онтології, що інтегрує проблемний простор (ПрП) знань про прецеденти в системах підтримки прийняття рішень (СППР).

В якості вхідної інформації дано.

ПрП – це модель аспектів експертної діяльності та компонент ПрО, з якими пов'язані (опосередковано або безпосередньо) знання, необхідні для вирішення різних завдань в цій ПрО.

Онтологія ПрО:

– онтологія об'єктів ПрО, формальна модель якої описується , де X – кінцева множина концептів (понять-об'єктів) заданої ПрО. R – кінцева множина семантично значущих відносин між поняттями-об'єктами ПрО. F – кінцева множина функцій інтерпретації, заданих на поняттях-об'єктах та / або відносинах;
– онтологія процесів ПДС.

У ПрП задано типовий набір задач, який описується трійкою , де – узагальнена задача проблемного простору, що складається з p задач, які, в свою чергу, складаються з w фрагментів кожна. Кожен фрагмент представлений процедурою, реалізованої на множині v, операцій кожна.

Крім того, задача  визначається множинами вхідних даних , вимог (умов, обмежень) , контексту задачі  і вихідними даними (або метою вирішення задачі) . – множина методів вирішення задач, Q – розв'язувач задач.

У ПрП задано множину методів , за допомогою яких вирішується задача , де q – множина методів для n-го типового набору задач; p – множина задач в n-му типовому наборі.

Обмеженнями  для опису онтології задач є наступні умови:

– якщо q <p, то для деяких (або всіх) задач  існує не більше ніж один метод їх вирішення;

– якщо q = p, то для кожної задачі  існує тільки один метод її рішення ;

– якщо q> p, то для деяких (або всіх) задач  існує більше одного способу їх вирішення.

Процес побудови онтології, що інтегрує є ітераційним і складається з трьох етапів.

На першому етапі будується онтологія для типового набору задач формується підмножина задач , фрагментів задач W і операцій V (рисунок 4.1).

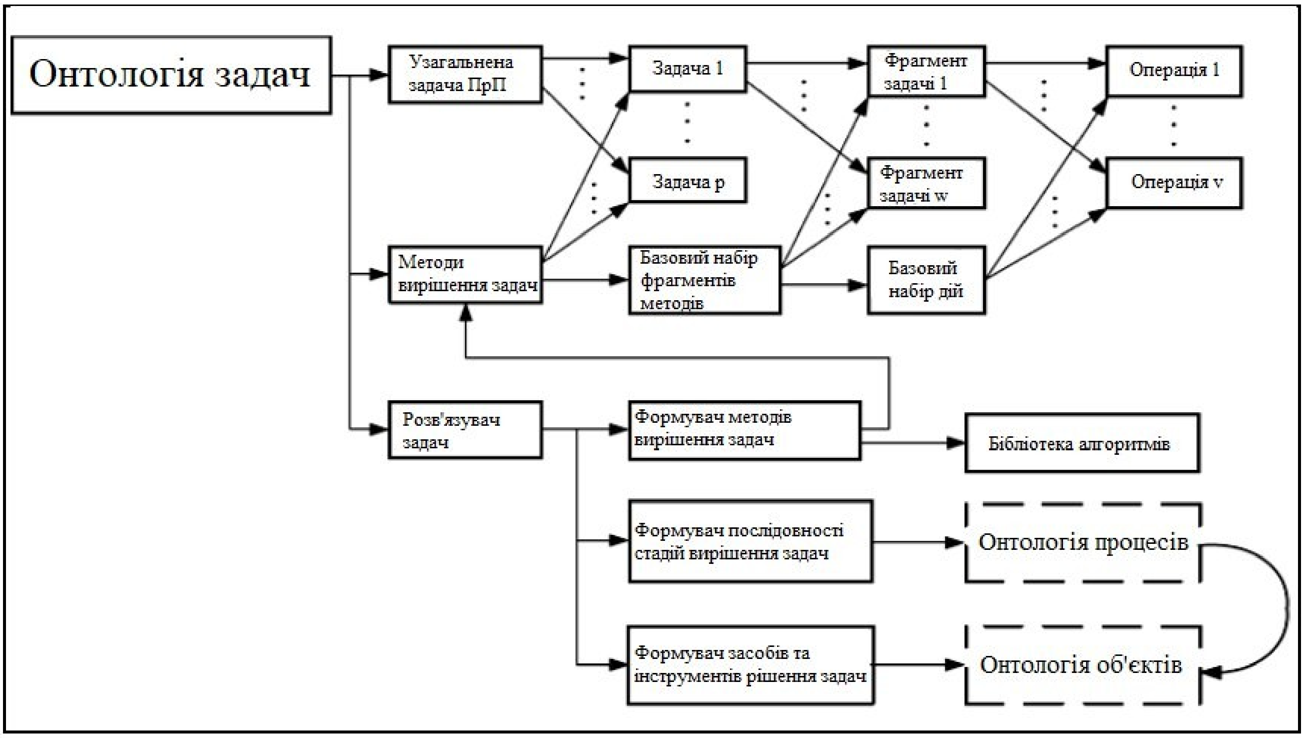


Рисунок 4.1 – Схема онтології задач

На наступному етапі описується онтологія розв'язувача задач (рисунок 4.2) для заданого типового набору.

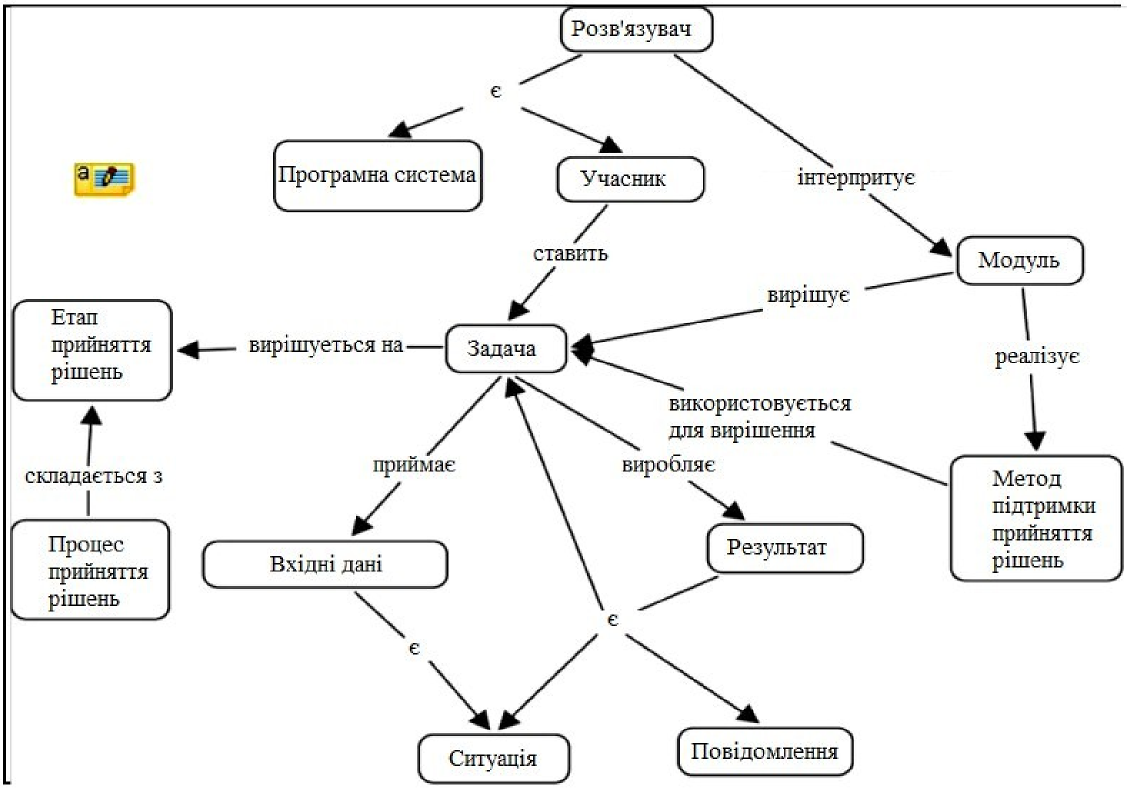


Рисунок 4.2 – Онтологія розв'язувача задач

В результаті на останньому етапі на основі запропонованого підходу була побудована інтегруюча онтологія проблемного простору знань про прецеденти, представлена на рисунку 4.3.

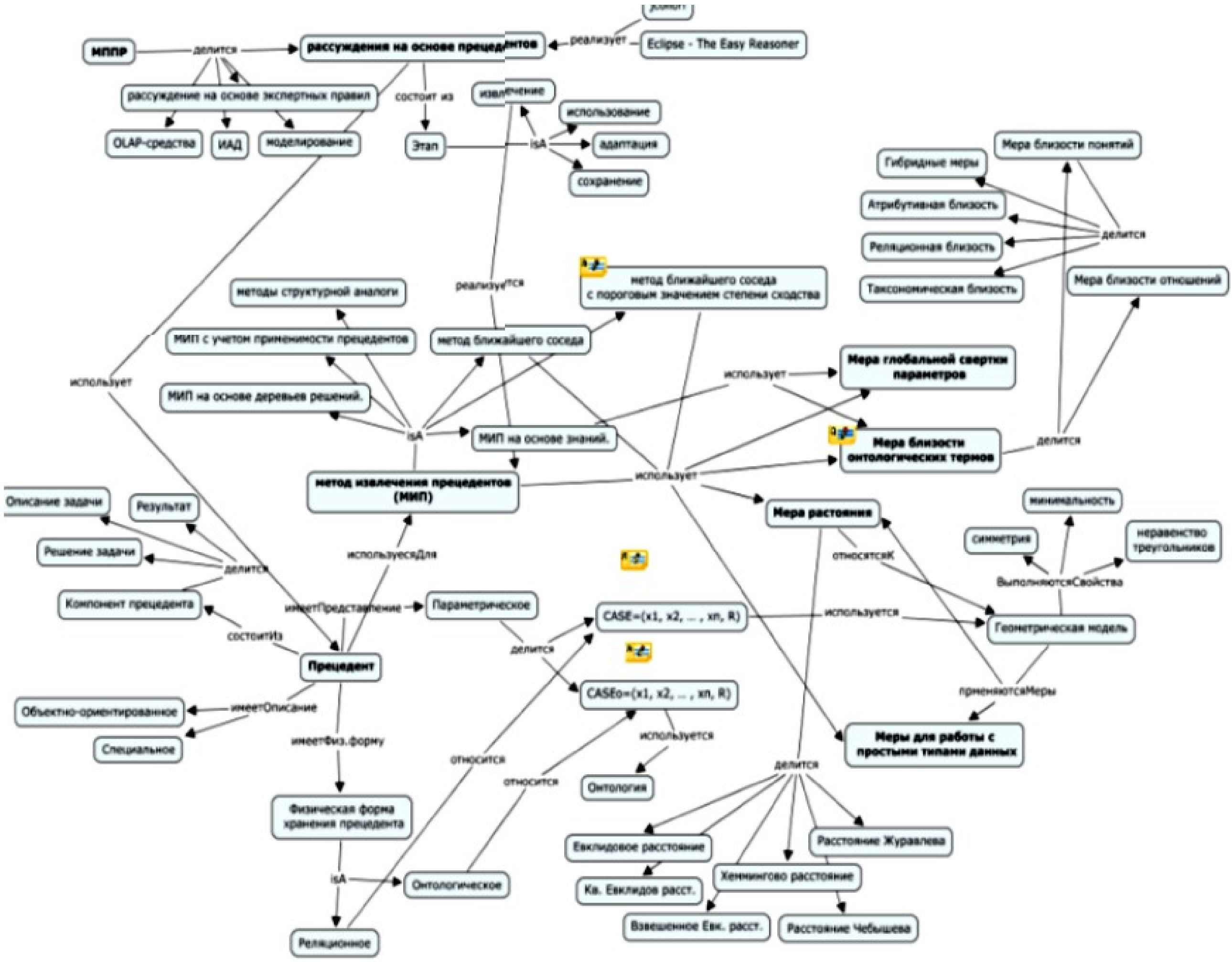

Рисунок 4.3 – Інтегруюча онтологія

В онтології формувачі методів вирішення завдань для множини методів виділяється базовий набір фрагментів методів і дій, представлених в таксономії методів підтримки прийняття рішень. Формуються послідовності процесів, задіяних в алгоритмах відповідних методів вирішення задач, і об'єктів, що беруть участь у вирішенні задач, з можливими значеннями, обмеженнями і умовами. Вказуються в явному вигляді, які конкретні об'єкти з онтології об'єктів беруть участь у відповідних процесах (алгоритмах) рішення задач.

Відзначимо, що представлена онтологія може бути розширена в залежності від числа вирішуваних задач.

# ВИСНОВКИ

У даній атестаційній роботі магістра виконано наступні роботи у межах дослідження обраної теми.

Виконано аналіз проблеми повторного використання програмного коду. Зокрема приділено увагу основним проблемам розробки та інтеграції інформаційних систем, дослідженню принципів мета модулювання інформаційних систем та аналізу існуючих методів повторного використання коду.

Проведено розробку моделей повторного використання програмного коду, в процесі якої були досліджені інтелектуальні системи, засновані на знаннях, з повторним використанням компонентів. За результатами дослідження було зроблено рішення, що найбільш доцільними для використання є онтології знань про ПрО.

Досліджені репозиторії онтологій як засіб повторного використання знань та стандарти подання метаданих про онтології.

Проведено розробку методів повторного використання програмного коду, в процесі розробки було проведено дослідження методів відображення онтологій та були встановлені основні критерії та мультикритерії подібності онтологій. Розроблено метод відображення онтологій заснований на методі рою частинок. Визначена експериментальна оцінка оптимізації відображення онтології.

Проведено апробацію отриманих наукових результатів. В рамках якої наведено модель процесу інтеграції системи знань розподілених інформаційних систем з неоднорідними онтологічними специфікаціями, що дозволяє аналізувати семантичні зв'язки, закономірності і залежності між ними. Визначена і описана область застосування результатів теоретичного дослідження при проектуванні різних класів інтелектуальних систем, орієнтованих на спільне використання знань.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки щодо розробки та оформлення магістерської атестаційної роботи за спеціальністю 122 Комп’ютерні науки (освітня програма «Інформаційні управляючі системи та технології» освітньо-кваліфікаційного рівня «магістр» / Упоряд.: Петров К.Е., Левикін В.М., Чалий С.Ф., Євланов М.В., Саєнко В.І., Міхнов Д.К., Міхнова А.В., Чала О.В. – Харків: ХНУРЕ, 2019. – 28 с.

2. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлювання. . – Чинний від 22.06.2015. – Київ: ДП «УкрНДНЦ», 2016. – 31 с.

3. Решетников И.С., Тупысев А.М., Владимирова М.В., Гревцев В.А. Стандарты интеграции многоуровневых информационных систем. Автоматизация в промышленности. 2019. № 9. С. 23-27.

4. Тимакин О.А., Радзивон В. Описание интеграционных решений информационной системы и особенности ее использования. URL: http:// joumalpTO.m/artides/opisanie-integratsionnykh-resheniy-mformatsi onnoy-sistemy-i-osobennosti-ee-ispolzovaniya/ (дата звернення: 27.05.2020).

5. Антипин К.В., Фомичев А.В., Гринев М.Н., Кузнецов С.Д., Новак Л.Г., Плешачков П.О., Рекуц М.П., Ширяев Д.Р. Оперативная интеграция данных на основе XML: системная архитектура BizQuery. URL: http://citforum.ru/internet/xml/bizquery/ (дата звернення: 02.06.2020).

6. Волков А.А., Шведенко В.Н. Модель формирования параллельных структур в объектно-ориентированных СУБД. Программные продукты и системы. 2016. № 3. С. 14-17.

7. Алаудинов А.Г. Построение единой системы интеграции данных в крупных корпорациях. URL: http://cyberleninka.ru/article/ n/postroenie-edinoy-sistemy-integratsii-dannyh-v-krupnyh-korpora tsiyah (дата звернення: 04.06.2020).

8. Жижимов О.Л., Федотов А.М., Шокин Ю.И. Технологическая платформа массовой интеграции гетерогенных данных. Вестн. НГУ. Информационные технологии. 2013. Т. 11. № 1. С. 24-41.

9. Ратманова И.Д., Павлов М.Н. Подход к организации средств интеграции данных в корпоративных информационно-аналитических системах. Информационные технологии. 2016. № 6. С. 2-11.

10. Набатов Р.А., Шведенко В.Н. Технология быстрой разработки баз данных и приложений пользователя в системе «COBRA++». Программные продукты и системы. 2018. № 2 (82). С. 39-41.

11. Кусов А.А. Проблемы интеграции корпоративных информационных систем. URL: http://uecs.ru/marketing/item/411-2011-04-25-10-08-37 (дата звернення: 10.06.2020).

12. Chung L., Subramanian N. Adaptable system/Software architectures. Journal of Systems Architecture. Vol. 50, Issue 7. 2014. Pp. 365-366.

13. Subramanian N., Chung L. Software Architecture Adaptability: An NFR Approach. URL: http://utdallas.edu/chung/ftp/IWPSE.pdf (дата звернення: 10.06.2020).

14. Cook St. Domain-Specific Modeling and Model Driven Architecture. MDA Journal. January 2017. Pp. 2-10.

15. Almeida J.P., Pires L.F. Costs and Benefits of Multiple Levels of Models in MDA Development. Methodologies and Transformations “Computer Science at Kent”. September 2018. Canterbury, UK. Pp. 12-21.

16. Favre J.-M., Nguyen T. Towards a Megamodel to Model Software Evolution Through Transformations. A satellite event of ICGT 2004. October 2019, Rome, Italy. Pp. 56-70.

17. Atkinson C., Kühne Th. The Essence of Multilevel Metamodeling. The Unified Modeling Language. Modeling Languages, Concepts, and Tools: 4th International Conference.. Toronto, Canada, October 2017. Springer. Pp. 19-33

18. Jones C. Assessment and Control of Software Risks. Engltwood Cliffs, N. J.:Prentice-Hall, 1994.

19. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб: Питер, 2001.

20. Горбунов-Посадов М. Облик многократно используемого компонента. Открытые системы. 2018. №3.

21. Элиенс А. Принципы объектно-ориентированной разработки программ. 2-е издание. М.: Вильямс, 2002.

22. Lee E.F. Embedded software. Advances in Computers. Vol. 56. Academic Press, London, 2012.

23. Экспертные системы. URL: http://itteach.ru/predstavlenie-znaniy/ekspertnie-sistemi (дата звернення: 10.06.2020).

24. Gruber T.R. A translation approach to portable ontologies. Knowledgeition. 2018. № 5(2). P. 199-220.

25. Guarino N., Guaretta P. Ontology's and Knowledge Bases. Towards a logical Calcification. Towards Very Large Knowledge Bases. Amsterdam: press, 2015.

26. Гаврилова Т. Онтология для изучения инженерии знаний. Труды Международной научно – практической конференции KDS-2019, 2019.

27. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4rd Edition. Prentice Hall, 2020.

28. Chandrasekaran B. Ontology of Tasks and Methods. B.Chandrasekaran, J.R.Josephson, V.R. Benjamins. IEEE Intelligent Systems, 14(1), 1998, P. 20-26.

29. Mizoguchi R., Vanwelkenhuysen J, Ikeda M. Task ontology for reuse of problem solving knowledge. Proceedings of the Second International Conference on Building and Sharing of Very Large-Scale Knowledge Bases. 2015. P. 45-59.

30. Hepp M. Ontologies: State of the Art, Business Potential, and Grand Challenges, Chapter 1. Ontology Management: Semantic WEB, Semantic WEB Services, and Business Application. 2007, Springer. P. 3-22.

31. Appelquist, D. A Standards-based, Open and Privacy-aware Social Web URL: http://www.w3.org/2005/Incubator/socialweb/XGR-socialweb-20101206/ (дата звернення: 10.06.2020).

32. Heery, R. Digital repositories review URL: http://www.jisc.ac.uk/media/documents/programmes/digitalrepositories/digitalrepositoriesreview2005.pdf (дата звернення: 10.06.2020).

33. Hartmann, J. Ontology Repositories URL: http://oa.upm.es/6430/2/0ntologyRepositories.pdf (дата звернення: 10.06.2020).

34. Hartmann, J. Ontology Repositories. Handbook on Ontologies, Edt. by S.Staab, R.Studer, Springer, 2019. P.551-572.

35. Palma, R. Oyster - sharing and re-using ontologies in a peer-to-peer community / R. Palma, P. Haase // International Semantic Web Conference, 2005. - P. 1059-1062.

36. Palma, R. OMV Ontology Metadata Vocabulary for the SemanticWeb URL: http://kent.dl.sourceforge.net/project/omv2/OMV%20Documentation/OMV-eportv2.4.1 .pdf (дата звернення: 10.06.2020).

37. Guha, R. Propagation of trust and distrust. Proc. of the Thirteenth International World Wide Web Conference. ACM Press, 2004. P. 403-412.

38. Noy, N.F. User ratings of ontologies: Who will rate the raters? Proc. of the AAAI 2015 Spring Symposium on Knowledge Collection from Volunteer Contributors. Stanford, CA, 2015.

39. Hartmann, J. OMV - Ontology metadata vocabulary / Hartmann J., Sure Y., Haase P., Palma R., Suarez-Figueroa M. C. Ontology Patterns for the Semantic Web. URL: http://link.springer.com/chapter/10#page-1 (дата звернення: 10.06.2020).

40. Проскудина Г.Ю. Обзор инструментов инженерии онтологий. URL: http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2004/part4/op (дата звернення: 10.06.2020).

41. Скворцов Н.А. Проектирование персонализованных информационных ресурсов над WEB-сайтами со слабоструктурированными данными. URL: http://synthesis.ipi.ac.ru/sigmod/seminar/s20000127 (дата звернення: 10.06.2020).

42. M. R. Quillan. Word concepts: A theory and simulation of some basic capabilities. Behavioral Science. 2007. 437 p.

43. Maedche A. A mapping framework for distributed ontology’s. Proceedings of the EKAW 2017. 2017. pp. 2-8.

44. Doan A. H. Learning to map between ontology’s on the semantic web. Proceedings to the Eleventh International World Wide Web Conference, Honolulu, Hawaii, USA, May 2002, pp.3-9.

45. Gruber, T.R. The role of common ontology in achieving sharable, reusable knowledge bases. Principles of Knowledge Representation and Reasoning. Proceedings of the Second International Conference. 2016. P.601-602.

46. Lambe, P. Organizing knowledge: taxonomies, knowledge and organisational effectiveness. Elsevier. 2014. 300 p.

47. Guarino, N. Formal ontology, conceptual analysis and knowledge representation. Int. J. of Human Computer Studies. 2015. Vol. 43(5/6). P.625-640.

48. Dou, D. Ontology Translation on the Semantic Web. Journal on Data Semantics. 2019. №2. P. 35-57.