Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Центр післядипломної освіти_____
(повна назва)

Кафедра _____Програмної інженерії_ _____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

Дослідження та розробка моделі знаходження рішень в задачах з невизначеним кінцевим станом
(тема)

Виконав: студент 6 курсу, групи ІПЗмзд-17-1___
спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)
Освітньо-наукової програми
Інженерія програмного забезпечення_____
(повна назва освітньої програми)

_____Вакарь Л.Г._____
(прізвище, ініціали)
Керівник _____проф. Четвериков Г.Г._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук_____

Кафедра Програмної інженерії_____

Рівень вищої освіти другий (магістерський)_____

Спеціальність _121-Інженерія програмного забезпечення_____
(код і повна назва)
освітньо-наукова програма _Інженерія програмного забезпечення_____
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри_____
(підпис)
«_____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові _____Вакарь Леоніду Германовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Дослідження та розробка моделі знаходження рішень в задачах з невизначеним кінцевим станом_____
затверджена наказом по університету від "___" _____20 ___ р № _____
заповнюється вручну після отримання наказу
2. Термін подання студентом роботи до екзаменаційної комісії
25 червня 2019 р.
3. Вихідні дані до роботи *алгоритми продукційних систем, двигуни висновків, експертні системи. Використовувати JMH бенчмарк, середовище об'єктно-орієнтованого проектування для мови програмування Java*_____
4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, архітектура двигунів висновків, провести аналіз алгоритмів продукційних систем, тестування продуктивності двигунів висновків, тестування використання пам'яті*_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *Архітектура експертної системи, приклад правила бази знань, схема роботи rete-алгоритму, схема перетворення правил, виконання правила з бази знань, спрямований та не спрямований граф і їх матриці, пошуку в глибину графа, алгоритм пошуку в глибину, алгоритм Кана, алгоритм топології методом глибокого пошуку, приклад роботи мережі обробки даних, алгоритм роботи провайдера бібліотеки, псевдокод алгоритму побудови графа, алгоритм дискримінації правил, алгоритм пошуку інструкцій, алгоритм обробки ситуації, схема роботи двигуна висновків, модель компонентів двигуна висновків, UML модель двигуна висновків, діаграма використання пам'яті, діаграма математичних моделей двигунів висновків, демонстраційні матеріали*

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	проф. Четвериков Г.Г.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Аналіз предметної галузі	26 квітня 2019р.	
2.	Огляд експертних систем	3 травня 2019р.	
3.	Алгоритми продукційних систем	10 травня 2019р.	
4.	Підготовка пояснювальної записки	27 травня 2019р.	
5.	Спецчастина		
6.	Підготовка презентації та доповіді		
7.	Попередній захист		
8.	Нормоконтроль, рецензування		
9.	Занесення диплома в електронний архів		
10.	Допуск до захисту у зав. кафедри		
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання _____ _____2019 р.

Студент _____
(підпис)

Керівник роботи _____проф. Четвериков Г.Г._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 96 с., 21 рис., 2 табл., 17 ф., 5 додатків, 22 джерела.

ДВИГУН ВИСНОВКІВ, ЕКСПЕРТНІ СИСТЕМИ, ОРІЄНТОВАНИЙ АЦИКЛІЧНИЙ ГРАФ, ПРОГРАМУВАННЯ ПОТОКУ ДАНИХ, СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Об’єктом дослідження є експертні системи. Експертні системи це різновид систем штучного інтелекту. Експертні системи здатні розв’язувати задачі з не заданим кінцевим станом.

Метою роботи є розробка моделі поліпшеного двигуна висновків для експертних систем. Нова модель двигуна висновків повинна використовувати менше оперативної пам’яті і не поступатися у швидкості існуючим аналогам.

В ході дослідження були проаналізовані алгоритми продукційних систем. Були проведені практичні дослідження існуючих двигунів висновків. На основі отриманих даних була створена модель двигуна висновків з новим алгоритмом продукційної системи.

Порівняльні дослідження довели менше використання оперативної пам’яті та швидшу роботу розробленої моделі порівняно з аналогами.

Модель може бути використана для створення менш ресурсомістких та швидших двигунів висновків. Розроблена модель має велике значення. Вона здатна розширити коло застосування експертних систем.

DATAFLOW PROGRAMMING, DIRECTED ACYCLIC GRAPH, EXPERT SYSTEMS, INFERENCE ENGINE, SYSTEMS OF ARTIFICIAL INTELLIGENCE, THEORY OF PROBLEM SOLVING.

The object of research is the expert system. Expert systems are a variety of artificial intelligence systems. They are capable to solve problems with an unspecified target state.

The aim of the work is to develop an improved inference engine for expert systems. The new inference engine model should use less RAM and have speed of existing analogs.

During the study, we analyzed algorithms of production system and made empirical study of existing inference engines. Based on the obtained data we create a model of the inference engine with a new production algorithm.

Comparative studies have shown less RAM usage and speed up against existing inference engine analogues.

The model can be used to developing less resource-intensive and faster inference engines. New model have big practice value. It is able to expand the scope of expert systems.

ЗМІСТ

Перелік умовних скорочень 6
Вступ 7
1 Аналіз стану досліджень 9
1.1 Системи підтримки прийняття рішень 9
1.2 Експертні системи та їх архітектура 12
1.3 Продукційні системи 16
1.4 Двигун висновків експертної системи 21
1.5 Визначення мети дослідження 25
2 Дослідження проблеми 27
2.1 Напрямок дослідження 27
2.2 Граф та його властивості 29
2.3 Огляд парадигм програмування 34
3 Розробка моделі двигуна висновків 38
3.1 План алгоритму роботи 38
3.2 Структура та робота правил двигуна висновків 40
3.3 Завантаження правил та побудова графа 41
3.4 Обробка та виконання правил 42
3.5 Оцінка швидкості роботи алгоритму 45
4 Імплементація та тестування 46
4.1 Вибір мови програмування для імплементації моделі 46
4.2 Розробка UML моделі 48
4.3 Розробка підходів до тестування 50
4.4 Результати тестування 52
5 Огляд розробленої моделі 57
5.1 Огляд переваг та недоліків розробленої моделі 57
5.2 Огляд шляхів подальшого розвитку та вдосконалення моделі 59
5.3 Потенційні галузі застосування розробленої моделі 61
Висновки 62
Перелік джерел посилання 63
Додаток А Програмний код 65
Додаток Б Слайди презентації 74
Додаток В Тези конференції «Реформування та розвиток гуманітарних та природничих наук» 81
Додаток Г Тези конференції «SCIENCE, RESEARCH, DEVELOPMENT #17» 88
Додаток Д Електронні матеріали (CD)

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

АСД	– абстрактне синтаксичне дерево
МДВ	– модель двигуна висновків
ООМП	– об’єктно-орієнтована мова програмування
ПС	– продукційна система
САГ	– спрямований ациклічний граф
СППР	– системи підтримки прийняття рішень

ВСТУП

Умовно всі задачі можна розділити на два типи. Задачі з заданим кінцевим станом і задачі з не заданим кінцевим станом. Для задач з заданим кінцевим станом характерний пошук шляху досягнення кінцевого стану. Задачі з не заданим кінцевим станом повинні прийти до кінцевого стану в ході процесу їх розв’язання. У таких завданнях результат не детермінований, певним і відомим є тільки початкова ситуація і набір методів її розв’язання. Розв'язанням таких задач займається різновид систем штучного інтелекту, а саме експертні системи. Експертні системи широко використовуються в різних галузях таких як охорону здоров'я, фінансова сфера, бізнес-планування тощо. Вони намагаються імітувати поведінку експерта при розв’язанні складних задач що потребують спеціальних галузевих знань.

В основі експертних систем знаходяться дані та логіка їх обробки. Для створення експертних систем використовується концепція “Знання + Висновки = Система”. База знань містить бізнес-правила які визначають, як програма обробляє дані. Бізнес-правила представлені у форматі “якщо-то”. Вони можуть бути легко змінені, без втручання в код додатку [1]. Бізнес-правила обробляє механізм висновків, або як його ще називають двигун висновків. Таким чином система розділена на два компоненти: база знань і двигун висновків. На даний момент на ринку існує велика кількість двигун висновків: Drools, Oracle Policy Automation, IBM Operational Decision Manager on Cloud, Blaze Advisor тощо. Однак всі ці рішення або використовують велику кількість апаратних ресурсів і повинні бути розгорнуті на окремому сервері, або становлять собою комерційні хмарні сервіси. Це пов’язано з використанням в їх основі ефективного але дуже ресурсномісткого rete алгоритму. Стороннім ефектом використання великих об’ємів пам’яті є потенційне зниження швидкодії програмної системи.

Вищезгадані програмні продукти не можуть бути використані як компонент невеликої програмної системи яка працює в апаратному середовищі зі скромними

ресурсами. Наприклад у мобільних додатках чи побутових розумних пристроях. У зв’язку з цим є актуальною розробка моделі двигуна висновків яка б використовувала невеликий об’єм апаратних ресурсів і при цьому не поступалася, а бажано перевищувала, за швидкодією сучасні двигуни висновків.

Для досягнення поставленої мети були розглянуті причини використання великої кількості апаратних ресурсів та сповільнення роботи існуючих двигунів висновків. Була проаналізована структура роботи rete алгоритму. Його концептуальні переваги та недоліки. Також було проведене порівняльне дослідження систем які використовують різні алгоритми продукційних систем і виявлені переваги у представленні правил у вигляді класів мови програмування.

В ході проведеного дослідження та розробки моделі здатної знаходити рішення для задач з не заданим кінцевим станом була висунута нова концепція продукційної системи яка може бути легко реалізована на об’єктно-орієнтованій мові програмування. Тестові імплементації моделі довели значно менший обсяг використаної пам’яті та значно більшу продуктивність в порівнянні з низкою існуючих продуктів. На основі розробленої моделі двигуна висновків можна створювати нові програмні продукти здатні працювати в системах з маленькими об’ємом операційної пам’яті. Також висока швидкість роботи моделі дозволяє використовувати її для побудови високонавантажених систем.

Окремі частини дослідження знайшли своє відбиття в тезах двох конференцій. На конференції, “Реформування та розвиток гуманітарних та природничих наук” яка проходила в місті Харків з 24 по 25 травня 2019 року, було зроблено доповідь під назвою “Концепція побудови продукційної системи для двигунів висновків експертних систем”. В ній була розглянута концепція нової продукційної системи. На міжнародній конференції “SCIENCE, RESEARCH, DEVELOPMENT №17” яка проходила в місті Белград (Сербія) з 30 по 31 травня 2019 року, були опубліковані тези під назвою “Порівняльне дослідження двигунів висновків експертних систем”. В тезах було описано проведене порівняльне дослідження швидкодії двох двигунів висновків Drools та EasyRules.

1 АНАЛІЗ СТАНУ ДОСЛІДЖЕНЬ

1.1 Системи підтримки прийняття рішень

Дослідники та інженери інформаційних систем розробляли та досліджували системи підтримки прийняття рішень протягом майже 40 років, а це означає, що концепція інтерактивної комп'ютерної системи, яка допомагає компаніям приймати кращі бізнес-рішення, існує з часів використання комп'ютерів. Системи підтримки прийняття рішень (СППР) створюються, щоб допомогти людям приймати рішення шляхом надання доступу до інструментів аналізу інформації. СППР це спосіб моделювання даних і прийняття на його основі якісних рішень. СППР являють собою клас комп'ютерних інформаційних систем, включаючи системи на основі знань які підтримують діяльність з прийняття рішень.

Хоча багато людей думають про підтримку прийняття рішень як про спеціалізовану частину бізнесу. Проте більшість компаній фактично інтегрували цю систему у свою щоденну операційну діяльність. Багато компаній постійно завантажують та аналізують дані про продажі, створюють бюджети та роблять прогнози. Вони оновлюють свою стратегію після аналізу поточних результатів. Завдання систем підтримки прийняття рішень полягає в тому, щоб зібрати дані, відфільтрувати та проаналізувати зібрані дані, а потім спробувати прийняти правильні рішення або побудувати стратегії з аналізу [2].

Існує декілька різних класифікацій СППР, запропонованих різними людьми. Наприклад, Гатеншвіллер говорить про пасивні, активні та кооперативи СППР. Він робить це розрізнення на підставі того, як користувач використовує систему. Пасивна система, допомагає у процесі прийняття рішень, але не висуває пропозицій або приклади рішень. Активна СППР висуває альтернативні пропозиції. Кооперативна система дозволяє особі переглядати та вдосконалювати пропозиції, що надаються системою, поки не буде прийняте остаточне рішення. Даніель Альтер розробив класифікацію ідея якої полягає в тому, що операції з підтримки прийняття рішень поширюються по одному виміру, починаючи від простих систем

зображення даних до систем логічної обробки даних. На одному кінці цієї класифікації знаходяться системи, які просто забезпечують доступ до елементів даних. Наприклад, моніторинг обладнання в реальному часі або системи моніторингу та пере упорядкування запасів. Наприклад, планування продукту та аналіз або прогнози продажів на основі маркетингової бази даних. На другому кінці – системи підтримки прийняття рішень, що використовують логічні моделі, які пропонують конкретні рішення для добре зрозумілого завдання, наприклад, розрахунок ставки поновлення страхування. Даниель Пауер поділяє СППР на п'ять класів: комунікаційно-керовані, керовані даними, документо-орієнтовані, керовані знаннями, модельно-орієнтовані. Комунікаційно-керовані СППР дозволяють більш ніж одній особі працювати над завданням, наприклад, програма веб-конференцій. СППР керований даними або СППР, орієнтовані на дані, дозволяють обробляти дані, які зберігаються в базі даних або сховищі даних. Інформація класифікується в хронологічному порядку, щотижневі продажі, щомісячні витрати тощо. Документо-орієнтовані СППР пов'язані з організацією інформації в безлічі електронних форматів. Прикладом може бути пошукова система, яка займається пошуком HTML-сторінок, PDF-файлів, файлів зображень, відео файлів тощо. СППР керовані знаннями, забезпечують експертне розв'язання проблем, що зберігаються як процедури та алгоритми правил, наприклад, при торгівлі акціями обмеження на ціну продажу розглядається як модель, керована знанням. Модельно-орієнтовані СППР дозволяють обробляти інформацію за допомогою кількісних моделей. Наприклад, організація ротації робочих місць, прогнозування майбутніх витрат, податкове планування, як правило, здійснюється з використанням модельної системи підтримки прийняття рішень [3].

Теоретично СППР може бути побудована в будь-якій галузь знань. Одним з прикладів є клінічна система підтримки прийняття рішень для медичної діагностики. Існує чотири етапи еволюції клінічної системи підтримки прийняття рішень: примітивна версія є автономною і не підтримує інтеграцію; друге покоління підтримує інтеграцію з іншими медичними системами; третє покоління – працює на основі медичних стандартів, а четвертий – на основі службової моделі [4].

СППР широко використовується в бізнесі та управлінні. Завдяки СППР вся інформація будь-якої організації може бути представлена у вигляді діаграм, графіків, тобто узагальнено, що допомагає керівництву приймати стратегічне рішення. Наприклад, одним з додатків СППР є управління та розвиток складних систем боротьби з тероризмом [5]. До інших прикладів можна віднести банківського позичальника, який перевіряє можливість кредитної позики, або інженерної фірми, що має ставки на декілька проектів, і хоче знати, чи вони можуть бути конкурентоспроможними виходячи з їх витрат.

Постійно зростає сфера застосування СППР, концепцій, принципів і методів у виробництві сільськогосподарської продукції. Наприклад, пакет DSSAT4, розроблений через фінансову підтримку USAID у 80-х і 90-х роках, дозволив швидко оцінити декілька систем сільськогосподарського виробництва в усьому світі, щоб полегшити прийняття рішень на рівні ферм та державного планування.

СППР також поширені в управлінні лісами, де довгий горизонт планування і просторовий вимір проблем планування вимагають конкретних вимог. Всі аспекти ведення лісового господарства, від транспортування колоди та планування урожаю до стабільності та захисту екосистем були впроваджені в сучасних СППР. У цьому контексті розглядаються єдині або численні цілі управління, пов'язані з наданням товарів і послуг, які виконуються або не виконуються і часто підлягають обмеженням ресурсів в проблемах прийняття рішень. Система підтримки лісового господарства використовує велике сховище знань про побудову та використання систем підтримки лісового господарства.

Ще один приклад стосується Канадської національної залізничної системи, яка регулярно перевіряє своє обладнання, використовуючи систему підтримки прийняття рішень. Проблемою, з якою стикається будь-яка залізниця, є зношені або дефектні рейки, які можуть призводити до сотень сходів потягів за рік. Завдяки СППР, Канадська національна залізнична система, вдалося зменшити частоту сходів, в той час як інші компанії зазнали зростання. Таким чином СППР не тільки здатні підвищувати ефективність роботи, але і рятувати життя.

1.2 Експертні системи та їх архітектура

Експертні системи в області штучного інтелекту представлені як комп'ютерні програми, які можуть виконувати різні завдання. Ефективно проводити процес пошуку та відновлення взаємопов'язаної інформації, що зберігається у великих масштабах. Формально представляти символічні знання на теренах, в яких діє система. Робити висновки на основі експертного знання. Пояснювати надані висновки, співпрацюючи з користувачем. Експертні системи являють собою ефективну допомогу, коли потрібна інтерпретація складної інформації, у вигляді експертної консультації. Експертні системи містять можливості роз'яснення міркувань, що випливають з розв'язання проблем і придбання нових знань, а також представляють прості процедури взаємодії з користувачами [6].

Перша експертна система була розроблена в 1965 році Едвардом Фейгенбаумом і Джошуа Ледербергом зі Стенфордського університету в Каліфорнії, США. Експертні системи тепер мають комерційне застосування в таких різноманітних областях, як медична діагностика, нафтопромисловість та фінансове інвестування тощо.

Для того, щоб виконати свою інтелектуальну функцію, експертна система спирається на дві складові: базу знань і механізм висновків. База знань – це організований набір фактів про галузь в якій працює система. Механізм висновків тлумачить і оцінює факти в базі знань, щоб дати відповідь. Типовими завданнями для експертних систем є класифікація, діагностика, моніторинг, проектування, та планування спеціалізованих робіт.

Факти для бази знань повинні бути отримані від кваліфікованих експертів через інтерв'ю або спостереження. Отримані знання зазвичай представлені у формі правил “якщо-то”. Якщо деяка умова є істинною, то можна зробити наступний висновок, або вжити певні дії. Бази знань великих за розміром експертних систем можуть включати тисячі правил або сотні тисяч правил. Фактор ймовірності часто додається до кожного правила і до кінцевої рекомендації, оскільки висновок не

завжди може бути сто відсоткове визначеним. Наприклад, система діагностики захворювань очей може на основі інформації, що надійшла до неї, вказувати на 90-відсоткову ймовірність того, що у людини є глаукома, і вона також може перераховувати висновки з меншою ймовірністю. Експертна система може показувати послідовність правил, через які вона прийшла до свого висновку. Відстеження цього потоку допомагає користувачеві оцінити довіру до його рекомендацій. Також це важливо для ефекту зворотного навчання коли знання одного експерта збережені в базі знань навчають менш кваліфікованого експерта шляхом логічного обґрунтування висновку.

Окрім правил типу “якщо-то”, людські експерти часто використовують евристичні правила, або “правила великого пальця”, разом із простими продукційними правилами. Наприклад, кредитний менеджер може знати, що заявник з поганою кредитною історією, але високим доходом з моменту придбання нової роботи, насправді може бути хорошим клієнтом. Експертні системи допомагають, але не замінюють людських фахівців. У багатьох галузях прийняття рішення покладається на людського експерта.

Експертні системи відіграли велику роль у багатьох галузях, включаючи фінансові послуги, телекомунікації, охорону здоров'я, обслуговування клієнтів, транспорт, відеоігри, виробництво, авіацію та письмове спілкування. Дві перші експертні системи були впроваджені в медичній галузі для постановки діагнозів: Dendral, який допоміг хімікам визначити органічні молекули, і MYCIN, який допомагав виявити бактерії, такі як бактеріємія і менінгіт, і рекомендувати антибіотики та дозування. Ще одним прикладом може бути комп'ютерна програма CaDet розроблена для того, щоб допомогти лікарям первинної медичної допомоги виявити рак на ранній стадії. Це дозволяє своєчасно почати лікування та не допустити розвиток хвороби. Вона аналізує епідеміологічні та клінічні ознаки окремих пацієнтів, а потім ідентифікує та представляє лікарям моделі, які можуть потребувати додаткової уваги. Комп'ютерна програма CaDet, добре пристосована до завдання допомоги лікарям покращити раннє виявлення, проте вона ще проходить клінічне тестування [7].

Експертна система базується на великому обсязі знань про конкретну проблемну область. Ці знання організовані як сукупність правил, що дозволяють системі робити висновки з даних “Знання + Висновки = Система”. Цей підхід до проектування систем, що базується на знаннях, був еволюційною зміною з революційними наслідками. Вона замінила традиційний підхід до розробки програмного забезпечення “Дані + Алгоритм = Програма”. Новий підхід був зосереджений навколо “бази знань” і “механізму висновків” [8]. На рисунку 1.1 змальована архітектура експертної системи.

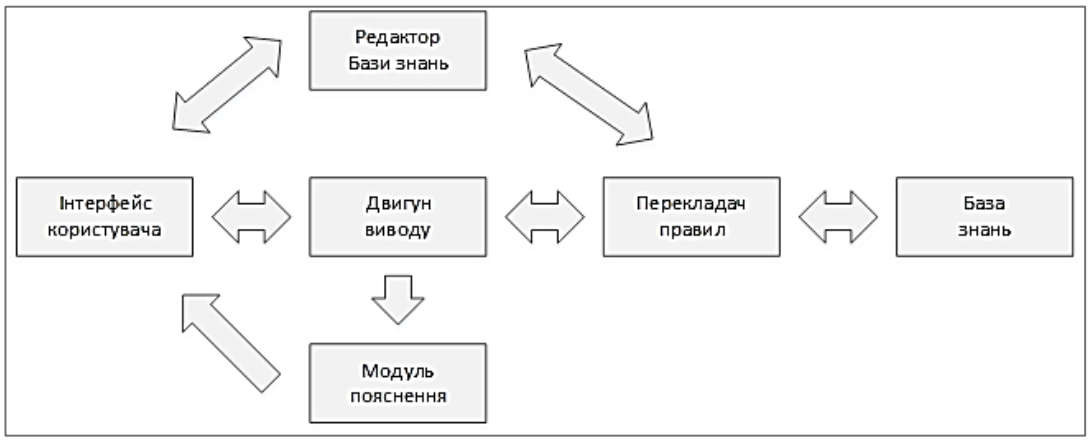

Рисунок 1.1 Архітектура експертної системи

Існують чотири важливі компоненти повноцінної експертної системи: база знань, двигун висновків, модуль придбання знань, пояснювальний інтерфейс. База знань містить факти та правила. Факти це короткострокова інформація, яка може швидко змінюватися, наприклад, під час консультації. Правила це більш довгострокова інформація про те, як генерувати нові факти або гіпотези з того, що зараз відомо. Основна відмінність від звичайної бази даних полягає в тому, що база знань є більш творчою. Факти в базі даних, як правило, пасивні. Вони там або є або нема. База знань, з іншого боку, активно намагається заповнити відсутню інформацію. Виробничі правила є найпоширенішим засобом інкапсуляції знань. Вони дозволяють легко змінювати поведінку додатку.

Ці правила не вбудовані в програмний код. Це дані для високорівневого інтерпретатора, а саме двигуна висновків. Якщо база знань розглядається як

програма, то механізм виведення є інтерпретатором. Вирази в мові представлення знань є його входом, а результати його роботи становлять інтерпретацію цього вводу щодо збережених знань. Інтерпретатор може використовувати логіку тому що він працює в певному логічному формалізмі, наприклад, обчислення предикатів першого порядку. Функція механізму виводу полягає в отриманні відповідних знань з бази знань, їх інтерпретації та пошуку рішення, яке відповідає проблемі користувача. Механізм висновку витягує правила зі своєї бази знань і застосовує їх до відомих фактів для виведення нових фактів. Приклад правила бази знань наведено на рисунку 1.2.

```
Rule 99
IF
      the home team lost their last home game,
AND
      the away team won their last home game
THEN
      the likelihood of a draw is multiplied by 1.075;
      the likelihood of an away win is multiplied by 0.96.
```

Рисунок 1.2 Приклад правила бази знань

Модуль збору та навчання це компонент функція якого полягає в тому, щоб дозволити експертній системі набувати все більше знань з різних джерел і зберігати їх у базі знань. Також цей модуль надає можливість редагувати знання.

Інтерфейс користувача обробляє запити від користувачів. Логіка інтерфейсу користувача направляє ці запити до відповідного програмного модуля. Наприклад, через цей модуль користувач, який не є експертом, може взаємодіяти з експертною системою та знаходити розв'язання проблеми. Також галузевий експерт який бажає створити або змінити правило, використовує модуль клієнтського інтерфейсу, щоб відредагувати базу знань. Сучасні експертні системи можуть навіть використовувати для цього елементи штучного інтелекту [9].

1.3 Продукційні системи

В даний час експертними системами найбільш широко застосовується тип систем збудованих на правилах. У системах, заснованих на правилах, знання представлені не декларативно у статичний спосіб, а в формі численних правил, які вказують, які висновки повинні бути зроблені або не зроблені в різних ситуаціях. Система, заснована на правилах, складається з правил “якщо-то”, фактів і інтерпретатора, який керує тим, яке правило має бути викликано в залежності від наявності фактів в робочій пам'яті.

Системи, засновані на правилах, відносяться до двох основних різновидів: системи з прямим логічним висновком і системи зі зворотним логічним висновком. Система з прямим логічним висновком починає свою роботу з відомих початкових фактів і продовжує роботу, використовуючи правила для виведення нових висновків або виконання певних дій. Система зі зворотним логічним висновком починає свою роботу з деякою гіпотези, або цілі, яку користувач намагається довести, і продовжує роботу, відшукуючи правила, які дозволять довести істинність гіпотези. Для розбиття великої задачі на дрібні фрагменти, які можна буде легше довести, створюються нові під цілі. Системи з прямим логічним висновком в основному є керованими даними, а системи зі зворотним логічним висновком – керованими цілями. В наш час системи часто реалізують обидва підходи.

Широке застосування систем, заснованих на правилах, обумовлено наступними причинами. Завдяки модульній організації спрощується уявлення знань і розширення експертної системи. Такі експертні системи дозволяють легко створювати засоби пояснення за допомогою правил, оскільки антецеденти правила точно вказують, що необхідно для активізації правила. Засіб пояснення дозволяє стежити за тим, запуск яких правил було здійснено, тому дає можливість відновити хід міркувань, які привели до певного висновку. Наявна аналогія з пізнавальним процесом людини. Згідно з результатами, отриманими Ньюеллом і Саймоном, правила, є природним способом моделювання процесу вирішення завдань

людиною. А при здійсненні спроби виявити знання, якими володіють експерти, простіше пояснити експертам структуру представлення знань, оскільки застосовується просте уявлення правил “якщо-то”. Правила відносяться до типу продукцій, ідея яких виникла в 1940-х роках [10]. У загальному випадку продукційну модель можна представити у вигляді формули (1).

$$i = \{S; P; A \rightarrow B; Q\} \quad (1)$$

де S – опис класу ситуації;

P – умова при котрій продукція активізується;

$A \rightarrow B$ – ядро продукції;

Q – результат виконання продукційного правила.

Продукційні системи були вперше використані в символічній логіці Постом, тому ім'я цього вченого увійшло в назву зазначених систем. Пост довів такий важливий і несподіваний результат, що будь-яка система математики або логіки може бути оформлена у вигляді системи продукційних правил певного типу. Цей результат показав величезні можливості застосування продукційних правил для представлення важливих класів знань, а це означає, що продукційні правила не зводяться до кількох обмежених типів. Основна ідея Посту полягала в тому, що будь-яка математична або логічна система являє собою набір правил, який вказує, як перетворити один рядок символів в інший послідовний набір символів. Це означає, що продукційне правило після отримання вхідного рядка (антецедента) здатне виробити новий рядок (консеквент). Така ідея є також дійсною щодо програм і експертним системам, в яких початковий рядок символів є вхідні дані, а вихідний рядок стає результатом певних перетворень, яким були піддані вхідні дані.

Наступний великий крок у розробці методів застосування продукційних правил був зроблений на підставі відкриття, зробленого Марковим, яке дозволило визначити структуру управління для продукційних систем. Алгоритм Маркова – це упорядкована група продукцій, які застосовуються в порядку пріоритету до вхідного рядка. Якщо правило з найвищим пріоритетом є непридатним, то

використовується наступне правило в порядку пріоритету. Алгоритм Маркова завершує свою роботу після виявлення одного з наступних умов: по-перше, остання продукція не може бути застосована до рядка або, по-друге, застосована продукція, яка закінчується крапкою.

Алгоритм Маркова передбачає застосування цілком певної стратегії управління, оскільки правила з високим пріоритетом розташовані по черзі на перших місцях. За умови, що може бути застосовано правило з найвищим пріоритетом, використовується це правило. В іншому випадку в алгоритмі Маркова робляться спроби використовувати правила з нижчим пріоритетом. Безумовно, алгоритм Маркова може бути застосовуваний в якості, основи експертної системи, але він є дуже неефективним способом створення систем з великою кількістю правил. Якщо потрібно створити експертну систему для вирішення реальних завдань, що містить сотні або тисячі правил, то проблема ефективності набуває найбільший пріоритет. Незалежно від того, наскільки прийнятними є всі інші характеристики системи, якщо користувачеві доведеться довго чекати відповіді, то він не буде працювати з такою системою. Тому потрібно було створити алгоритм, який має повну інформацію про всі правила і може застосувати будь-яке потрібне правило, не роблячи спроби послідовно перевіряти кожне правило.

Розв'язанням цієї проблеми є rete-алгоритм, розроблений Чарльзом Л. Форг в університеті Карнегі-Меллона в 1979 році в рамках дисертації по командному інтерпретатору експертної системи на здобуття ступеня доктора філософії. Термін rete-алгоритм походить від латинського слова rete, яке означає мережу. Rete-алгоритм функціонує як мережа, призначена для зберігання великого обсягу інформації та забезпечує значне скорочення часу відгуку і підвищення швидкодії при запуску правил в порівнянні з великими групами правил “якщо-то”, які повинні перевірятися один за іншим у звичайній системі, заснованій на правилах. Rete-алгоритм заснований на використанні динамічної структури даних, яка автоматично реорганізується в цілях оптимізації пошуку [10].

Rete-алгоритм являє собою дуже швидкий засіб зіставлення з шаблонами, висока швидкодія якого досягається завдяки зберіганню в оперативній пам'яті

інформації про правила, які перебувають в мережі. Цей алгоритм призначений для підвищення швидкодії систем з прямим логічним висновком, заснованих на правилах, завдяки обмеженню обсягу роботи, необхідної для повторного обчислення конфліктної безлічі після запуску одного з правил. Недоліком цього алгоритму є його високі потреби в пам'яті, але в наші дні, коли мікросхеми пам'яті стали такими дешевими, цей недолік не має великого значення. У rete-алгоритмі втілені два описаних нижче емпіричних спостереження, на підставі яких була запропонована структура даних, що лежить в його основі. Тимчасова надмірність. Зміни, що виникають в результаті запуску одного з правил, зазвичай зачіпають лише кілька фактів, а кожне з цих змін впливає тільки на кілька правил. Структурна подібність, один і той же шаблон часто виявляється в лівій частині більше ніж одного правила. Схему rete-алгоритму можна знайти на рисунку 1.3.

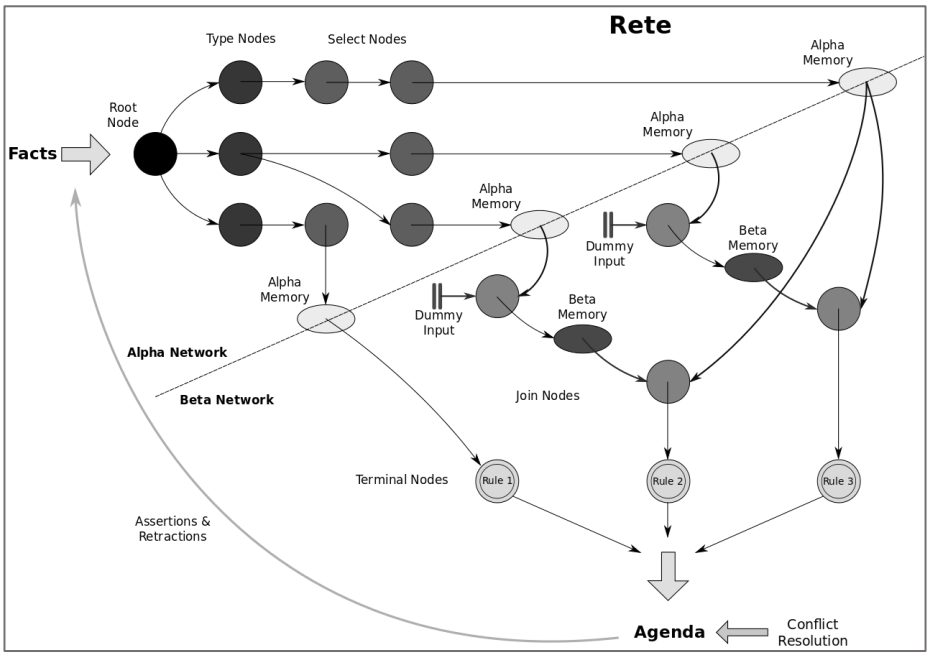

Рисунок 1.3 Схема роботи rete-алгоритму

Ліва, альфа частина графа, утворює сортувальну мережу, відповідальну за вибір записів щодо відповідності їх атрибутів встановлених констант. Вузол може перевіряти кілька атрибутів запису. Якщо запис відповідає умові, вона передається далі по графу. У більшості систем перші від кореня вузли перевіряють ідентифікатор або тип запису, тому всі записи одного типу направляються по одній

гілці сортувальної мережі. Кожна гілка містить пам'ять, в якій накопичуються записи. Записи, які не відповідають хоча б одній умові, не включаються до відповідних альфа-списків. Гілки альфа-вузлів можуть розділятися для зменшення надмірності умов. Можливою модифікацією є додавання пам'яті до кожного вузла, що дозволяє легше реалізовувати модель програмно [11].

Права, бета частина графа, виконує складання записів. Вона використовується тільки при необхідності. Кожен вузол має 2 входи і направляє результат в бета-пам'ять. Бета-вузли працюють з мітками, що позначають записи в альфа-пам'яті. Мітка є одиницею зберігання і пересилки між вузлами і пам'яттю. Бета-вузол в якості результату може створювати нові мітки для списку записів або створювати списки міток. Цей підхід не вимагає копіювання самих записів, вузол просто створює нову мітку, додає її до голови списку і зберігає в буфері виходу. При проходженні запису по бета-мережі до неї додаються нові атрибути. Записи в кінцевих вузлах бета-мережі повністю відповідають умові продукції і передаються в виходи rete-мережі [11].

Якщо в системі задані сотні або тисячі правил, то підхід до організації роботи, в якому комп'ютер послідовно перевіряє вірогідність того, чи повинен бути виконаний запуск кожного правила, стає дуже неефективним. Завдяки розробці rete-алгоритму з'явилася практична можливість створення інструментальних засобів експертних систем навіть на тих повільних комп'ютерах, які застосовувалися в 1970-х роках. В наші дні rete-алгоритм продовжує залишатися важливим засобом підвищення швидкодії в тих випадках, коли експертна система містить багато правил. Rete-алгоритм чутливий не до кількості правил, а до їх предикатів.

У rete-алгоритмі в кожному циклі контролюються тільки зміни в узгодженнях, тому в кожному циклі "розпізнавання-дія" не доводиться погоджувати факти з кожним правилом. Завдяки цьому істотно підвищується швидкість узгодження фактів з антецедентами, оскільки статичні дані, які не змінюються від циклу до циклу, можуть бути проігноровані. Але rete-алгоритм має ряд обмежень. Звичайні моделі rete не можуть охоплювати складну бізнес-логіку. Rete-алгоритм жертвує пам'яттю для реалізації швидкості обробки [6].

1.4 Двигун висновків експертної системи

Правила складені експертами в якійсь галузі, не є безпосередньо виконуваними. Спочатку вони повинні бути перетворені з їхньої придатної для читання людиною форми у форму, яка може бути інтерпретована двигуном висновків. Перетворення правил з однієї форми в іншу виконується перекладачем правил. На рисунку 1.4 змальована схема перетворення правил.

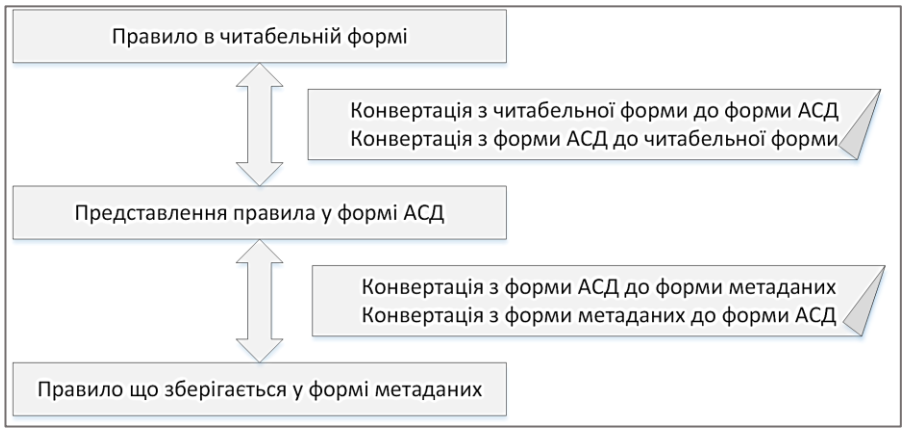

Рисунок 1.4 Схема перетворення правил

Переклад правил у їх первісному вигляді в машиночитну форму вимагає аналізу для отримання структури даних, що називається абстрактним синтаксичним деревом (АСД). Абстрактне синтаксичне дерево правил – це структура даних, яка керує виконанням механізму виводу, коли настає час застосовувати правило. АСД є абстрактним типом даних, призначеним для спрощеної та ефективної інтерпретації. Цей абстрактний тип даних дуже виразний і дозволяє будувати дуже складні та потужні правила. Існує третя форма, в якій правила можуть бути виражені. Правило АСД перетворюється в еквівалентну форму, придатну для зберігання в базі знань. Спосіб відбиття цієї інформації в базі знань залежить від технології зберігання. Реляційні бази даних забезпечують особливу зручність та ефективність зберігання метаданих. Метадані, що відповідають АСД, у цьому випадку будуть знаходитися в наборі таблиць. Конкретна схема, обрана для збереження метаданих, повинна дозволити АСД

швидко реконструюватись за допомогою запитів до бази даних. На рисунку 1.3 змальована узагальнена діаграма операцій перетворення, які перекладач правил має виконати, коли додає правила до бази знань, а також витягує правила з бази знань для виконання та отримання експертного висновку[12].

Після створення АСД перетворюються в метадані правила і зберігаються в базі знань. Метадані правил – це просто компактне представлення АСД. При перекладі правил з однієї форми в іншу, структура вихідного правила ніколи не втрачається. Завжди можна відтворити правильне для читання людини правило, саме з його представлення бази знань або з його подання АСД.

Двигун висновків відповідає за виконання правил бази знань. Він отримує правила з бази знань, перетворює їх в АСД, а потім надає їх перекладачеві правил для виконання. Інтерпретатор двигуна висновків проходить через АСД, виконуючи дії, вказані в правилі на цьому шляху. Цей процес зображений на рисунку 1.5.

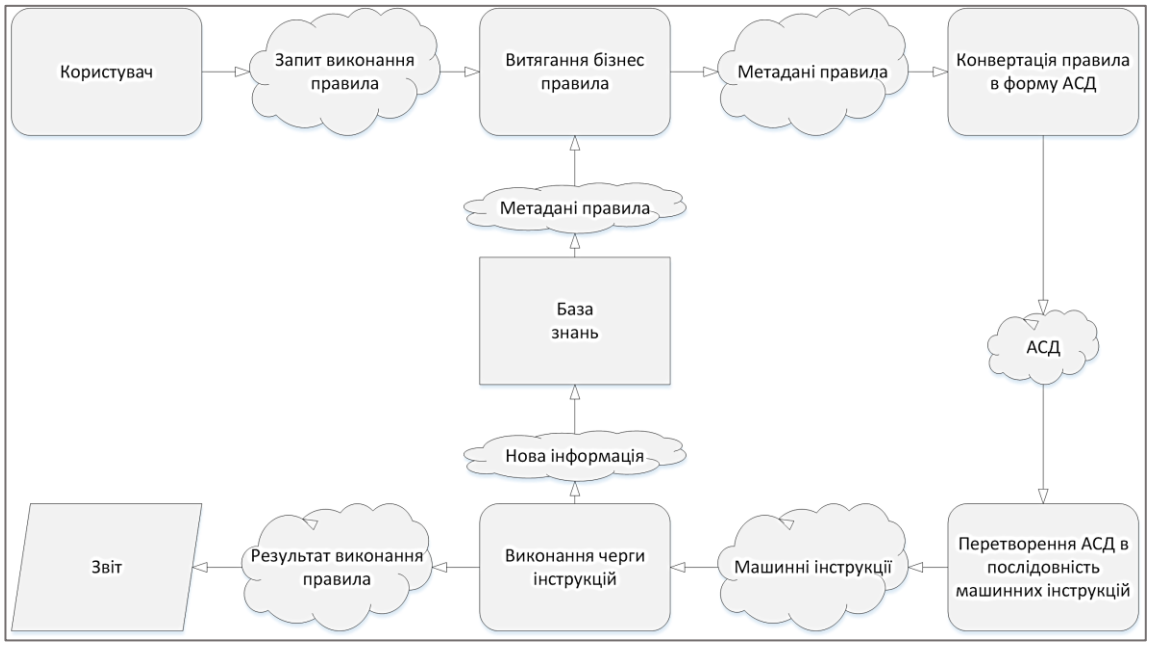

Рисунок 1.5 Виконання правила з бази знань

Таким чином правила що зберігаються в базі знань спочатку повинні бути інтерпретовані двигуном висновків у зрозумілу комп’ютеру мову і тільки потім вони можуть бути виконані. Інтерпретація правил дорогий с точки зору використання комп’ютерних ресурсів процес.

Існує багато імплементацій двигунів висновку експертних систем. Частина з них є платними наприклад Oracle Policy Automation та IBM Operational Decision Manager частина безкоштовними або мати безкоштовні обмежені версії наприклад, Drools , Jess, JRules тощо.

Drools - це система управління бізнес-правилами з прямим і зворотним механізмом обґрунтування, більш відомий як двигунів висновків. Він використовує розширену реалізацію алгоритму rete. Drools підтримує стандарт Java Engine (Java Specification Request 94) для свого двигуна висновків. Він забезпечує інструменти для побудови, обслуговування та забезпечення дотримання бізнес-логіки в додатках, службах або корпоративних сервісах.

Oracle Policy Automation - це набір програмних продуктів для моделювання та розгортання бізнес-правил у корпоративних додатках. Корпорація Oracle придбала Oracle Policy Automation в грудні 2008 року. Oracle Policy Automation була розроблена компанією RuleBurst для перетворення законодавчих та політичних документів у бізнес-правила, зокрема для розрахунку прав на виплати та розміру виплат. Хоча Oracle Policy Automation спочатку була розроблена і продана державному сектору, вона може бути використана в інших галузях.

IBM Operational Decision Manager on Cloud – це хмарна служба, яка допомагає збирати дані, автоматизувати і адмініструвати прийняття бізнес-рішень на основі правил. Вона дозволяє бізнес-користувачам швидко ініціювати проекти управління бізнес-правилами з меншими витратами, зменшивши необхідність допомоги IT-фахівців і придбання апаратного та програмного забезпечення. Використовуючи IBM Cloud, компанія може швидко виконувати масштабування та адаптуватися до мінливих бізнес-вимог без шкоди для безпеки та конфіденційності і без зростання ризиків пов’язаних з не хмарними продуктами.

Jess є двигуном правил для платформи Java, розробленої Ернестом Фрідманом-Хілл з національних лабораторій Sandia. Це різновид мови програмування CLIPS. Вперше він був написаний наприкінці 1995 року. Мова передбачає програмування на основі правил для автоматизації експертної системи та часто називається оболонкою експертної системи. В останні роки також

розвивалися системи інтелектуальних агентів, які залежать від подібних можливостей. Замість процедурної парадигми, коли єдина програма має цикл, який активується лише один раз, декларативна парадигма, яку використовує Jess, постійно застосовує набір правил до набору фактів. Правила можуть змінювати набір фактів, або вони можуть виконувати будь-який Java-код. Двигун правил Jess використовує алгоритм rete. Пізніше він отримав підтримку алгоритму rete-2, а за відсутності повної відкритої специфікації rete-2 цю інформацію не можна перевірити. Проте швидкість обробки правил зросла.

Система управління правилами Blaze Advisor дозволяє максимально контролювати операційні рішення з великим обсягом. Blaze Advisor надає підприємствам в різних галузях масштабоване рішення, яке забезпечує безпрецедентну гнучкість і дієвість для розумніших, прозорих і зрозумілих бізнес-рішень. Розгортання хмари одним натисканням допомагає підприємствам швидко реалізовувати рішення. Це комплексне рішення для управління корпоративними правилами, що включає моделювання рішень, розробку рішень, розробку, тестування, розгортання та обслуговування.

JRules, Drools і ряд інших двигунів висновків використовують те, що вони називають розширеною версією rete, що підвищує продуктивність. За винятком OPSJ PST, тільки Blaze Advisor використовує rete-2, що ліцензію Fair Isaac від PST. Враховуючи, що кількість правил і об'єктів у системі завжди зростає, можна почати з найкращої максимальної продуктивності для першої реалізації. Алгоритм rete-NT щонайменше в 500 разів швидше, ніж оригінальний rete, і в 10 разів швидше, ніж rete-2. Основна мета двигунів висновків - дозволити швидко та легко змінювати бізнес-правила у складних додатках у відповідь на потреби бізнесу. Кілька виробників (IBM, Oracle, Fair Isaac, Pegasystems, Visual Rules тощо) домоглися максимального спрощення цього процесу, навіть дозволяючи бізнес-користувачам вводити та змінювати правила безпосередньо. Тепер проблема полягає в продуктивності. Перехресне зіставлення мільйонів об'єктів і тисячі правил може зіткнутися з трильйонами порівнянь лише для одного виконання [13].

1.5 Визначення мети дослідження

Експертні системи з’явилися, ще у другій половині 20 століття і с тих пір їх архітектура залишалася майже не змінною. Одним із суттєвих її недоліків є підхід до зберігання правил. В сучасних системах правила зберігаються у вигляді метаданих, які при кожному виконанні правила конвертуються в дерево абстрактного синтаксису правила з якого будується послідовність машинного виконання. Такий підхід не є ефективним ані з точки зору використання комп’ютерної пам’яті, та часу процесора ані з міркувань швидкодії. Ця проблема прямо пропорційна кількості виконуваних правил у базі знань. Для пришвидшення виконання запитів до експертних систем зараз використовуються сучасні аналоги rete-алгоритму. Сучасні експертні системи використовують двигуни висновків створені на основі імплементацій rete-алгоритму. Rete-алгоритм становить собою дуже швидкий засіб зіставлення фактів з предикатами правил. Висока швидкодія досягається шляхом зберігання в оперативній пам'яті інформації про правила представлені у вигляді мережі [10, с. 83]. Вони намагаються розв'язувати питання швидкості використовуючи мережу з предикатів правил. Час за який виконується послідовність правил для клієнтського запита до систем з rete алгоритмом можна представити у вигляді формули (2) за умови, що послідовність виконуваних правил є відомою заздалегідь, тобто пре розрахована.

$$t = \sum_{i=1}^{n} (Alpha_i + Exec_i) \qquad (2)$$

де $Alpha_i$ – час побудови альфа мережі;

$Exec_i$ – час виконання.

Правила для rete-алгоритму повинні бути написані з використанням спеціально розробленого синтаксису. Rete-алгоритм дозволяє виконувати двигуну

висновків тільки ті правила які можуть бути виконані виходячи з наданих вхідних фактів ефективно відкидаючи непотрібні правила.

Попри свою ефективність, моделі засновані на rete-алгоритмі використовують багато оперативної пам’яті. Rete-алгоритм жертвує пам’яттю заради швидкості виконання правил. Метою роботи є розробка моделі двигуна висновків яка б використовувала невеликий об’єм апаратних ресурсів і при цьому не поступалася, а бажано перевищувала за швидкодією сучасні двигуни висновків.

Одним із варіантів збільшити швидкість виконання запитів є зберігання правил у базі знань у вигляді безпосередньо машинних інструкцій для яких вже відомий порядок виконання правил виходячи з стандартних вхідних даних. Тобто змінити формулу наступним чином (3).

$$t = \sum_{i=1}^{n} Exec_i \tag{3}$$

де $Exec_i$ – час виконання.

Такий підхід може значно пришвидшити виконання запитів. Хоча rete-алгоритм і наближений до такої моделі він витрачає додатковий час на співставлення вхідної інформації з предикатами правил.

2 ДОСЛІДЖЕННЯ ПРОБЛЕМИ

2.1 Напрямок дослідження

Відштовхуючись від об’єкта досліджень, а саме експертної системи, а також від мети роботи, а саме розробка моделі двигуна висновків яка б використовувала невеликий об’єм апаратних ресурсів і при цьому не поступалася, а бажано перевищувала за швидкодією сучасні двигуни висновків було визначено два основні напрямки дослідження. Перший напрямок намагатися розробити модель двигуна висновків експертної системи на основі вже наявних алгоритмів продукційних систем. Другий напрямок спробувати розробити новий алгоритм для продукційної системи який може частково використати окремі частини вже розроблених алгоритмів або підходів до порядку виконання правил.

Якщо намагатися реалізувати перший підхід, то існують три альтернативи: продукційна модель Поста в якій до кожного об’єкта вхідної інформації застосовуються всі правила наявні в базі знань, алгоритм Маркова який дозволяє скоротити кількість правил, що застосовуються до кожного вхідного інформаційного об’єкта в системі шляхом введення елементів граматики в процес виконання правил, та rete-алгоритм який вміє ефективно обирати для виконання тільки необхідні для прийняття експертного рішення правила. Перші два підходи не є ефективними з точки зору швидкості роботи алгоритмів. Алгоритм Маркова може бути основою продукційної системи. Але створення ефективної системи необхідно буквально в ручному режимі керувати послідовністю виконання правил. Не дозволяє використовувати правила написані на мовах програмування із-за складності їх синтаксису [14]. Тому доцільним є другий напрямком дослідження.

Фактично в ході проведення дослідження довелося розробити новий тип продукційної системи пристосований до використання правил написаних на ООМП. Відштовхуючись від попередньо проведеного дослідження [14], розроблювана продукційна система повинна поєднувати в собі можливість використовувати в правилах складну бізнес-логіку та ефективно обирати тільки

необхідні для експертного рішення правила. Концепція ПС повинна поєднати дві базові ідеї. Перша це представлення правил у вигляді класів ООМП. Друга – використання ідей реалізованих в rete-алгоритмі для зменшення кількості правил які можуть бути використані під час процесу отримання експертного рішення. Ефективність роботи rete-алгоритму забезпечує представлення наборів предикатів правил у вигляді спрямованого ациклічного графа (САГ) на кінці якого знаходяться елементи виводу нових фактів [15]. Така концепція алгоритму дозволяє швидко знаходити правила, які можна застосувати до набору вхідних фактів проходячи до них крізь мережу графу. З теоретичної точки зору час роботи rete-алгоритму не залежить від кількості правил які знаходяться в базі знань. Більш критичним показником є загальна кількість унікальних предикатів правил. Також на час роботи алгоритму впливає кількість параметрів які інкасуються у фактах які служать вхідною інформацією для правил. Хоча синтаксис ООМП і не дозволяє побудувати мережу з предикатів правил, можна побудувати мережу із самих правил. Якщо мережа правил буде представлена у формі САГ виникне можливість використання властивостей графу для знаходження необхідних для виконання правил. Наприклад, для цього може бути використаний алгоритм пошуку в глибину [16, с. 26]. Для реалізації такого підходу концепція ПС повинна обробляти особливу структуру правила написаного на ООМП. Правило повинно складатися з масиву вхідних фактів, набору предикатів та функції. Функція виводить новий факт із масиву вхідних фактів, якщо ті відповідають набору предикатів. Факт який виводить одне правило може бути використаний іншим правилом. Це дозволяє побудувати мережу правил пов'язаних між собою зв’язками вхідних та вихідних фактів. Факт який продукує правило не може бути використаний самим правилом. Така заборона дозволяє, для заданого набору правил, побудувати САГ.

Таким чином експертні системи створені на основі запропонованої концепції зможуть швидше обробляти запити користувачів в складних, бізнес орієнтованих додатках. Також вони не будуть містити мережу предикатів правил, що повинно зменшити кількість використовуваної оперативної пам’яті. Це дозволить робити складніші та швидші системи, які менш примхливі до апаратних ресурсів.

2.2 Граф та його властивості

У математиці, а точніше в теорії графів, граф це структура що складається з набору об'єктів, в яких деякі пари об'єктів у певному сенсі пов'язані. Об'єкти відповідають математичним абстракціям, що називаються вершинами. Кожна зі споріднених пар вершин називається ребром. Як правило, граф зображується у вигляді діаграми з набору точок або кіл для вершин, з'єднаних лініями або кривими для ребер. З математичної точки зору граф це впорядкована пара вершин (4). Графи використовуються для розв'язання реальних проблем. Наприклад, вони використовуються для представлення мереж.

$$G = (V, E) \tag{4}$$

де V – це набір вершин;

E – набір ребер.

Графи можна класифікувати за низкою ознак. Граф, який має непусту множину вершин, з'єднаних не більше ніж одним ребром, називається простим. У мультиграфах або як їх ще називають гіперграфах дозволяється мати декілька ребер між двома вершинами. Псевдографи дозволяють ребрам з'єднати вершину з собою. Графи поділяються на орієнтовані та неорієнтовані. Орієнтований граф складається з непустого набору вузлів та безлічі спрямованих ребер. Кожне ребро задається упорядкованою парою вершин. Граф має обмежену кількість ребер (5).

$$E < V^2 \tag{5}$$

де V – це набір вершин;

E – набір ребер.

Орієнтований граф простий, якщо він не має петель, тобто не має можливості досягти початкову вершину з самої себе, і не має кратних ребер. Орієнтований граф

може містити ребра, що пов’язують вершини в обидва напрямки. Якщо кількість ребер графу близька до V*logV граф називається щільним, якщо E=V*logV граф називається розрідженим, де *V* це кількість вершин, а *E* кількість ребер.

Існує два стандартних способи представлення графів: як сукупність списків суміжностей, або як матрицю суміжностей вершин. Представлення списку суміжності використовується для представлення розріджених графів. Представлення матриці суміжності може бути кращим, коли граф щільний. Вершини у списку суміжності зберігаються у довільному порядку. Потенційний недолік представлення графа списком суміжності полягає в тому, що немає швидкого способу визначити, чи є ребро між двома заданими вершинами. Цей недолік усувається матричним відбиттям суміжності. Це бінарна матриця значеннями якої є нуль або одиниця. Якщо між двома вершинами є ребро, то ставиться значення один. Якщо між вершинами немає зв’язку то ставиться значення нуль [17]. На рисунку 2.1 представлені матриці спрямованого та не спрямованого графів, незважаючи на схожість вони мають відмінності.

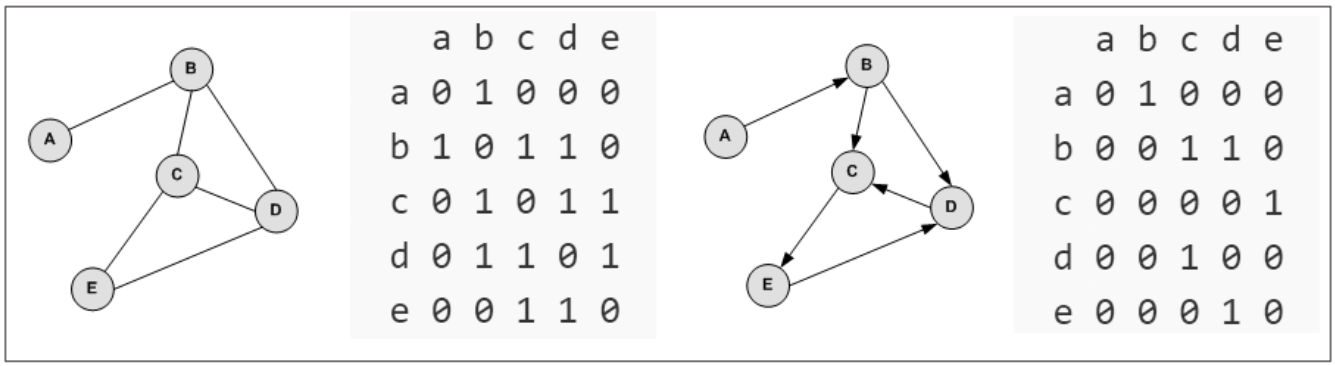

Рисунок 2.1 Спрямований та не спрямований граф і їх матриці

При побудові матриці суміжності вершин треба бути дуже обережними, бо матриці суміжності спрямованих та не спрямованих графів відрізняються. В матриці спрямованого графа зображуються тільки вхідні для вершини ребра.

Граф формується набором вершин і ребер, де вершини є безструктурними об'єктами, які попарно з'єднані ребрами. У випадку спрямованого графа, кожне ребро має орієнтацію, від однієї вершини до іншої вершини. Шлях у спрямованому

графі може бути описаний послідовністю ребер, де кінцева вершина кожного краю в послідовності є такою ж, як і початкова вершина наступного краю послідовності. Граф має цикл, якщо початкова вершина його першого краю дорівнює кінцевій вершині останнього краю. Направлений ациклічний граф є спрямованим графом, який не має жодного циклу в середині себе циклів [16].

Вершину спрямованого графа називають досяжною з іншої вершини, коли існує траєкторія, яка починається з першої та закінчується на другій. Як окремий випадок, кожна вершина вважається досяжною від себе. Якщо вершина може досягати себе через нетривіальний шлях (шлях з одним або декількома ребрами), то цей шлях є циклом, тому інший спосіб визначення спрямованих ациклічних графів полягає в тому, що вони є графами, в яких не можна досягти вершини через нетривіальний шлях, тобто з самої себе [18].

Однією з властивостей САГ є можливість пошуку у глибину. Пошук в глибину – один з методів обходу графа. Стратегія пошуку в глибину, як і випливає з назви, полягає в тому, щоб йти “вглиб” графа, наскільки це можливо. На рисунку 2.2 проілюстровано порядок пошуку в глибину графа.

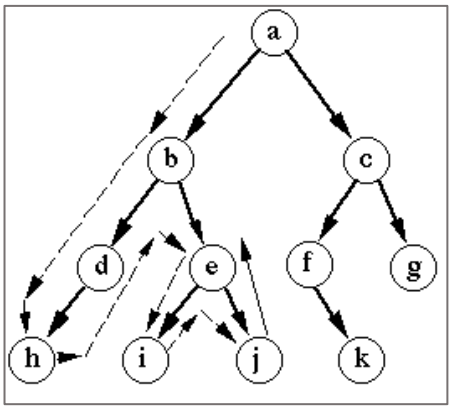

Рисунок 2.2 Пошуку в глибину графа

Алгоритм пошуку описується рекурсивно. Він перебирає всі вихідні з даної вершини ребра. Якщо ребро веде в вершину, яка не була розглянута раніше, то запускаємо алгоритм від цієї нерозглянутої вершини, а після цього повертається і продовжує перебирати ребра. Повернення відбувається в тому випадку, якщо в

даній вершині не залишилося ребер, які ведуть до нерозглянутих вершину. Якщо після завершення алгоритму не всі вершини були розглянуті, то необхідно запустити алгоритм від однієї з нерозглянутих вершин. Рекурсивний псевдокод алгоритму представлений на рисунку 2.3.

```
DFS(G, u)
u.visited = true
for each v ∈ G.Adj[u]
      if v.visited == false

DFS(G,v)
init() {
for each u ∈ G
      u.visited = false
for each u ∈ G
      DFS(G, u)
}
```

Рисунок 2.3 Псевдокод алгоритму пошуку в глибину

Алгоритм пошук в глибину дуже швидкий. Складність алгоритму становить порівняно досить не висока і напряму залежить від суми вузлів та ребер між ними.

Кожен спрямований ациклічний граф має топологічне впорядкування, впорядкованість вершин таким чином, що початкова кінцева точка кожного ребра виникає раніше у порядку, ніж кінцева точка краю. Існування такого впорядкування може бути використано для характеристики спрямованого ациклічного графа: орієнтований граф є ациклічним тоді та тільки тоді, коли він має топологічне впорядкування. Для побудови топології спрямованого ациклічного графа існують декілька алгоритмів. Згідно алгоритму Кана представленому на рисунку 2.4 спочатку будується список вершин із кількістю вхідних ребер для кожної вершини. Потім обираються всі вершини з кількістю суміжних вершин нуль та додаються до списку результатів L. Після цього всі вершини зі ступенем 0 додаються в чергу. Далі циклом дістають вершину з черги та потім: збільшують кількості відвідуваних вузлів на 1, зменшують ступінь на 1 для всіх сусідніх вузлів, якщо ступень сусідніх вузлів дорівнює нулю, то він додається в чергу. Цикл повторюється доки черга не стане пустою. Якщо кількість відвідуваних вузлів не дорівнює кількості вузлів у

графі, то топологічне сортування неможливе для даного графа. Якщо граф є спрямованим та ациклічним, рішення буде міститися в списку L. В іншому випадку граф повинен мати принаймні один цикл. Якщо граф має принаймні один цикл побудова топологію графа неможлива [19].

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edge
while S is non-empty do
    remove a node n from S
    add n to tail of L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
if graph has edges then
    return error   (if graph has at least one cycle)
else
    return L   (a topologically sorted order)
```

Рисунок 2.4 Псевдокод алгоритму Кана

Існує також альтернативний алгоритм для топологічного сортування заснований на пошуку в глибині. Він зображений на рисунку 2.5.

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then return
    if n has a temporary mark then stop   (not a DAG)
    mark n with a temporary mark
    for each node m with an edge from n to m do
        visit(m)
    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

Рисунок 2.5 Псевдокод алгоритму топології методом глибокого пошуку

Алгоритм побудови топології через пошук в глибину проходить через кожен вузол графа, у довільному порядку, ініціюючи пошук у глибині, який закінчується,

коли він потрапляє на будь-який вузол, який вже був відвіданий з початку топологічного сортування, або вузол не має вихідних країв. Кожен вузол n отримує попередній список L виходу тільки після розгляду всіх інших вузлів, які залежать від n (всі нащадки n у графу). Зокрема, коли алгоритм додає вузол n, він гарантує, що всі вузли, які залежать від n, вже знаходяться у списку виходів L: вони були додані до L або шляхом рекурсивного виклику для відвідування, який закінчився до виклику n або за допомогою виклику для відвідування, який почався ще до виклику для відвідування n. Оскільки кожний край і вузол відвідується один раз, алгоритм виконується в лінійний час. Цей алгоритм, що базується на глибокому першому пошуку, має невисоку обчислювальну складність (6).

$$O(V + E), \quad E < V^2 \tag{6}$$

де V – кількість вершин;

E – кількість ребер.

Практичні дослідження не виявили суттєвих переваг у швидкості роботи між розглянутими алгоритмами побудови топології. Топологічне впорядкування графа не є унікальним. Він має унікальне топологічне впорядкування тоді та тільки тоді, коли воно має спрямований шлях, що містить всі вершини, в цьому випадку впорядкування є таким же, як порядок, в якому вершини пов'язані між собою ребрами. Будь-які два графи, що представляють один та той же частковий порядок, мають один і той же набір топологічних порядків [20].

2.3 Огляд парадигм програмування

Програмування потоку даних – це парадигма програмування, модель виконання якої може бути представлена спрямованим графом, що представляє потік

даних між вузлами, подібно до діаграми потоку даних. Враховуючи це порівняння, кожен вузол є виконуваним блоком, який має входи даних, виконує перетворення над ним і потім пересилає його до наступного блоку. Додаток потоку даних складається із блоків обробки, що мають один або більше вхідних блоків і один або більше кінцевих блоків, пов'язаних спрямованим графом. Програмування потоку даних вже понад 40 років вивчається в області програмної інженерії. Його запропонував Теза Берта Сазерленда [21].

Всі моделі в програмуванні потоку даних представлені трьома елементами: вузол, маркер даних та арка. Вузол це елемент який отримує, обробляє дані, та продукує нові. Арки зв’язують між собою вузли зв’язком вхідних та вихідних об’єктів. Маркери даних це дані які переміщуються від вузла до вузла. Найбільш ранні пропозиції щодо потоку даних уявляли собою маркери даних як пасивні елементи, які залишалися на дугах, поки вони не були прочитані, а не фактично контролювали виконання.

Проте, це швидко стало нормальним при проектуванні систем потоку даних. Існувало два способи зробити це в теоретичній реалізації моделі чистого потоку даних. На рисунку 2.6 змальована схема роботи мережі вузлів обробки потоку даних для виконання математичних розрахунків.

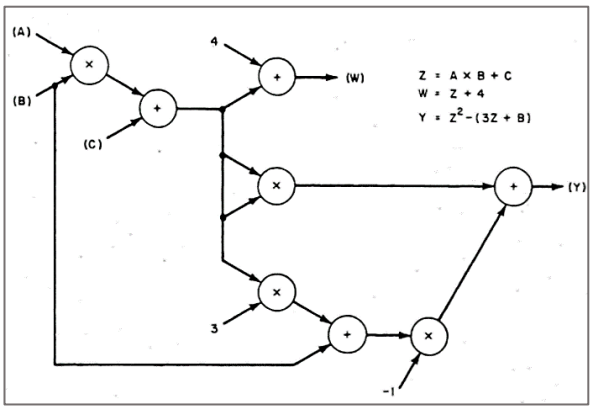

Рисунок 2.6 Приклад роботи мережі обробки даних

Перший підхід відомий як підхід керований даними. По суті, вузол неактивний, коли дані надходять до нього. Підхід, керований даними, є двофазним

процесом. На першій фазі вузол активується коли він має повний набір даних для роботи. На другій фазі він оброблює вхідні дані та розміщує вихідні маркери на своїх вихідних дугах. Другий підхід називається підходом вимоги. У цьому підході вузол активується тільки тоді, коли він отримує запит на дані зі своєї дуги виходу. На цьому етапі вимагаються дані з усіх відповідних вхідних дуг. Після того, як він отримав свої дані, він виконує і розміщує маркери даних на своїх вихідних дугах. Таким чином, підхід, що керується вимогою, є чотири фазним процесом. Спочатку середовище вузла запитує дані, потім вузол активується і запитує дані з його середовища, середовище відповідає даними, вузол їх опрацьовує та розміщує вихідні данні на своїх вихідних дугах. Виконання програми починається тоді, коли середовище вимагає певного виходу з графа. Кожен з цих підходів має певні переваги. Підхід, керований даними, має ту перевагу, що він не має додаткових витрат на розповсюдження запитів даних по графу потоку даних. З іншого боку, підхід, керований попитом, має ту перевагу, що деякі типи вузлів можуть бути не використані у разі відсутності попиту на їх данні. Це пояснюється тим, що потрібні лише необхідні дані [22].

Програмування потоку даних – це область, яка залишається відкритою для подальших досліджень. Програмування потоку даних може бути виконано візуальними мовами програмування, проте немає жодних рамок, що не могли б забезпечити інтеграцію цих функцій на сучасних мовах програмування. Програмування потоку даних може бути реалізований без середовища візуального програмування, хоча графічне представлення того, як вузли обробляють основні потоки даних надають користувачеві краще розуміння того, що програма повинна робити [21]. Візуальний контроль може значно пришвидшити роботу.

Об'єктно-орієнтоване програмування (ООП) - це парадигма програмування, яка представляє поняття як “об'єкти”, які мають поля даних (атрибути) і пов'язані з ними процедури, відомі як методи. Об'єкти, які зазвичай є зразками класів, і пов’язані між собою використовуються для розробки додатків і комп'ютерних програм. Такі мови програмування як C ++, Objective-C, Smalltalk, Java, C #, Perl, Python, Ruby і PHP є прикладами об'єктно-орієнтованих мов програмування.

Об'єктно-орієнтоване програмування є практичною і корисною методологією програмування, яка заохочує модульний дизайн та повторне використання програмного забезпечення.

SIMULA була першою мовою програмування яка використовувала об’єктно-орієнтований підхід. Вона була використаний для моделювання. Алан Кей, який в той час знаходився в Університеті штату Юта, мав бачення персонального комп'ютера, який забезпечував би графічно орієнтовані програми, і він вважав, що мова, подібна до SIMULA, стане хорошим способом для неспеціалістів створювати ці програми. Він продав своє бачення Xerox Parc, а на початку 1970-х років, команда на чолі з Аланом Кей в стінах компанії Xerox Parc створила перший персональний комп'ютер який мав назву Dynabook.

На початку 1990-х років група Sun на чолі з Джеймсом Гослін розробила спрощену версію C ++, яка називалася Java, яка повинна була бути мовою програмування для відео додатків. Цей проект нікуди не йшов, доки група не переорієнтувала свій фокус і не продавала Java як мову для програмування інтернет-додатків. Ця мова набула широкої популярності, оскільки Інтернет процвітав, хоча його проникнення на ринок було обмежено його неефективністю.

Існують чотири фундаментальних принципів ООП, а саме: успадкування, абстракція, інкапсуляція та поліморфізм. Успадкування членів від батьківського класу це процес, за допомогою якого об'єкти одного класу набувають властивості об'єктів інших класів. Він підтримує концепцію ієрархічної класифікації. Абстракція це процес приховування деталей і викриття лише суттєвих ознак певного поняття або об'єкта без включення до них деталей чи пояснення. Інкапсуляція це приховування внутрішніх елементів класу. Об'єкти розкривають функціональність лише за допомогою методів, властивостей, подій і приховують внутрішні деталі, такі як стан і змінні від інших об'єктів. Це полегшує оновлення або заміну об'єктів, якщо їх інтерфейси сумісні, не впливаючи на інші об'єкти та код. Поліморфізм це грецький термін означає здатність брати більше однієї форми. Операція може демонструвати різну поведінку в різних випадках. Поведінка залежить від типів даних, що використовуються в операції.

3 РОЗРОБКА МОДЕЛІ ДВИГУНА ВИСНОВКІВ

3.1 План алгоритму роботи

Алгоритм роботи двигуна висновків повинен складатися з декількох функціональних компоненті, які повинні забезпечити максимальну ефективність роботи двигуна висновків. Функціональними елементами алгоритму є:

– завантаження правил;

– побудова спрямованого ациклічного графа;

– дискримінація непотрібних правил;

– побудова інструкцій виконання;

– запуск правил.

Спочатку система повинна завантажити всі правила з бібліотеки в масив. Таким чином можна отримати каталог правил де у кожного правила є свій унікальний індекс. Використовуючи цей індекс можна в одну дію знайти правило в масиві. Альтернативним варіантом є формування масиву безпосередньо в бібліотеці правил. Це є досить логічним виходячи з того, що бібліотека правил має всі необхідні для цього данні. Правила при цьому повинні відповідати єдиним стандартам реалізуючи єдиний для всіх правил інтерфейс.

Одразу після завантаження правил двигун висновків повинен використовуючи масив правил створити мережу правил у вигляді спрямованого ациклічного графа. Це дозволить проводити швидкий пошук правил пов’язаних між собою зв’язками вхідних та вихідних об’єктів. При побудові графа варто використати перевірку на наявність можливих циклів в середині графа. Кожен вузол графу може мати безліч вхідних та вихідних ребер. Виходячи з того, що граф спрямований існує декілька структур даних яким він може бути представлений. Перевагу слід віддати масиву суміжних вершин тому що в ході роботи двигуна висновків не має потреби у швидкому пошуку суміжності для якоїсь окремої вершини. А тому не має потреби представляти граф у вигляді матриці. Збудований

граф актуально зберігати до тих пір поки не виникне потреба змінити набір правил. Якщо набір правил буде змінено граф потрібно буде перебудувати.

Правил в бібліотеці може бути дуже багато, а вхідні інформаційні об’єкти не можуть бути використані всіма правилами бібліотеки. Тому виникає потреба провести дискримінацію правил, тобто зменшити кількість правил до кількості необхідної для прийняття рішень виходячи із наявних вхідних фактів. Правила які не можуть бути використані для обробки вхідних інформаційних об’єктів повинні бути відсіяні. Правила, що залишилися треба представити у вигляді САГ. Це дозволить побудувати топологію графа. Отриманий граф можна зберегти в пам’яті для кожного набору вхідних об’єктів. Це дозволить швидко знаходити потрібні правила і не витрачати час на повторну дискримінацію правил.

Для правильного послідовного виконання правил, потрібно залучити інструкції виконання. Інструкції виконання повинні містити інформацію про порядок виконання правил, а також про взаємодію правил між собою. Для цього можна використати властивості САГ. Для спрямованого ациклічного графу можна побудувати топологію графа. Тобто правильну послідовність вузлів графа. Це дозволить виконувати правила в тій послідовності в якій не може виникнути ситуація коли правило виконується раніше за створення факту який воно використовує. Також для того, щоб знати в які саме правила додавати їх продукції потрібний масив суміжних вершин. Його можна використати напряму, бо в даній структурі даних зберігається граф.

Кінцевою частиною роботи алгоритму повинен бути запуск всіх правил в порядку черги, а також реалізація взаємодії між пов’язаними правилами. Спочатку повинні бути знайдені правила які потребують вхідних інформаційних фактів. До кожного такого правила повинні бути додані ті факти яких вони потребують. Після цього правила повинні бути виконані згідно з топологією графа. Отриману топологію графа варто також зберегти в пам’яті. Це дозволить також пришвидшити виконання при повторному запиті до даного набору правил. Після того як правило завершило свою роботу його продукт додається до правил, що його потребують. Результат виконання зберігається в масив результатів.

3.2 Структура та робота правил двигуна висновків

Вхідною інформацією для алгоритму є початкова ситуація, яка складається з набору фактів. Для обробки початкової ситуації двигун висновків використовує набір правил. Результатом роботи алгоритму є кінцева ситуація (7).

$$R(S) \to Q \tag{7}$$

де R – набір правил;

S – початкова ситуація;

Q – кінцева ситуація.

Інформація зберігається у вигляді фактів *f*. За допомогою правил *r* із вже існуючих фактів створюються нові факти. Факти поділяються на два типи: факти які входять до початкової ситуації *sf* та факти які продукуються правилами *rf*.

Правило має простий алгоритм роботи. Спочатку відбувається накопичення необхідних для роботи правила фактів в масив *WM*. Коли в наявності є всі необхідні факти відбувається процес виконання правила. Правило тестує *WM* на відповідність предикату *P(WM)*, якщо предикат повертає істину на об’єктах *WM* виконується функція *F(WM)*, яка продукує *rf*. У іншому випадку правило продукує модель за замовчуванням rf_d. Загальна форма алгоритму роботи правила може бути представлена формулою (8).

$$P(WM)\begin{cases} false \to F(WM) \to rf_d \\ true \to F(WM) \to rf \end{cases} \tag{8}$$

Факти типу *rf* можуть входити до *WS* правил, це дозволяє побудувати мережу правил пов'язаних між собою зв’язками вхідних та вихідних фактів. Факт який продукує правило не може бути частиною його *WM*. Така заборона дозволяє для заданого набору правил *R*, побудувати направлений ациклічний граф *G*.

3.3 Завантаження правил та побудова графа

Правила повинні зберігатися у вигляді бібліотек класів об’єктно-орієнтованої мови програмування. Бібліотека класів повинна містити спеціальний клас провайдер, який надає всі правила у вигляді масиву правил. Спочатку провайдер створює об’єкт класу правила потім додає його до масиву. Цю операцію він проводить для кожного правила. Алгоритм роботи провайдера змальовано на рисунку 3.1. Складність алгоритму складає O(R), де *R* кількість правил.

```
ALGORITHM #1
1.    while rules exists
2.          create object of rule
3.          add rule object to R
4.    return R
```

Рисунок 3.1 Псевдокод алгоритму роботи провайдера бібліотеки

При старті системи двигун висновків звертається до провайдера і той надає йому всі наявні в бібліотеці правила. Далі з масиву правил створюється граф. . Алгоритм цього процесу представлено на рисунку 3.2.

```
ALGORITHM #2
1.    Procedure(R)
2.    initialize G
3.    for each ruleX in R
4.          for each ruleY in R
5.                if ruleX.rf in ruleY.WM then
6.                      G add [ruleX, ruleY]
7.    return G
```

Рисунок 3.2 Псевдокод алгоритму побудови графа

Граф формується у вигляді масиву суміжних вершин. При цьому процесі кожне правило перевіряється на зв’язок між вхідними та вихідними об’єктами. У разі наявності такого зв'язку додається суміжна вершина. Складність алгоритму складає $O(R^2)$, де *R* кількість правил.

3.4 Обробка та виконання правил

Бібліотека правил може містити безліч правил. Але для обробки вхідної інформації від користувача не завжди можуть знадобитися всі правила. Якщо бібліотека буде містити десятки тисяч правил *R*, а для обробки запиту та формування експертного висновку потрібно лише сто з них, то розгляд і намагання виконати непотрібні правила, це втрата часу та комп'ютерних ресурсів. Правила які не можуть бути використані для обробки вхідних фактів ситуації повинні бути відсіяні. Правила, що залишилися повинні бути представлені у вигляді САГ. Представлення правил у вигляді спрямованого ациклічного графу є дуже важливим бо він може бути використаний для побудови його топології. Тобто правильної послідовності виконання правил. Дискримінація правил може бути виконана алгоритмом змальованим на рисунку 3.3.

```
ALGORITHM #3
1.    Procedure(S, G)
2.    init  Rd
3.    for each sf in S
4.          Rd add ← dfs(G, sf)
5.    return  Rd
```

Рисунок 3.3 Псевдокод алгоритму дискримінації правил

До МДВ надходить ситуація *S*. МДВ використовує ситуацію для знаходження тільки тих правил які можуть бути використані для її обробки *S*. Для цього можна буде використати алгоритми пошуку вглибину графа. Пошук в глибину графа потрібно провести для кожного *sf* з множини *S*. Результати пошуку потрібних правил необхідно зберегти у набір в якому немає дублікатів правил R_d. На основі R_d потрібно побудувати новий більш компактний граф (див. рис.3.2). Швидкість пошуку правил на основі алгоритму пошуку вглибину графа складає O(S(R+E)). Граф збережений у форматі списку суміжних вершин є базовою структурою даних алгоритму для збереження правил.

Умовно алгоритм складається з двох частин. Перша формує набір інструкцій які відповідають на питання в якій саме послідовності повинні активізуватися правила. Друга частина алгоритму безпосередньо активізує правила використовуючи послідовність знайдену в першій частині алгоритму. Поділ алгоритму на дві частини створює можливість повторного використання послідовності активації правил. Складність алгоритму пошуку інструкцій $O(R_d^2)$. На рисунку 3.4 представлений псевдокод алгоритму пошуку інструкцій.

```
ALGORITHM #4
1.    Procedure(Rd)
2.    init Graph
3.    for each x in Rd
4.          for each y in Rd
5.                if x.rf in y.WM then
6.                      Graph add [x, y]
7.    Topology ← Graph.getTopology()
8.    Adjacency ← Graph.getAdjacency()
9.    return [Topology, Adjacency]
```

Рисунок 3.4 Псевдокод алгоритму пошуку інструкцій

Перша частина алгоритму з набору правил які пройшли дискримінацію R_d формує направлений ациклічний граф (лінія 2 - 6). Вершинами графу слугують правила. Ребро графу формується якщо *rf* одного правила є частиною WM іншого правила (лінії 5 - 6). Напрям ребра графу вказує від правила що продукує до правила що споживає. Структура граф формується за допомогою пар правил. Першим елементом слугує правило що продукує другим правило що споживає.

Для знаходження порядку обробки правил потрібно побудувати топології графа. Алгоритм побудови топології графа в цій роботі не розглядається. Одним з можливих алгоритмів побудови топології може слугувати алгоритм Кана. Крім топології графа у вигляді інструкції використовується мапа суміжних вершин, вона вказує на всі вершини які можна досягти з заданої. Цю мапу використовує друга частина алгоритму розв'язання задач з не заданим кінцевим станом. Однією з переваг роботи алгоритму Кана є перевірка графу на ациклічність. Якщо граф циклічний його топологію буде не можливо збудувати. Результатом роботи першої

частини алгоритму є топологія графа та мапа суміжних вершин. На рисунку 3.5 представленні псевдокод вищерозглянутого алгоритму.

```
ALGORITHM #5
1.    Procedure(Rd, S, Topology, Adjacency)
2.    init Q
3.    for each index in topology
4.          r ← Rd[index]
5.          for each sf in S
6.                if sf in r.WM then
7.                      r.WM add sf
8.          rf ← r.run()
9.          Vertex_Adjacency ← Adjacency[index]
10.         if Vertex_Adjacency not empty
11.               for each i in vertex_adjacency
12.                     forwardRule ← Rd[i]
13.                     forwardRule.WM add rf
14.         else
15.               Q add rf
16.   return Q
```

Рисунок 3.5 Псевдокод алгоритму обробки ситуації

Друга частина алгоритму приймає на вхід масив правила, топологію графа, мапу суміжних вершин та ситуацію. Кожне правило активується в порядку описаному в топології графа. Якщо правило потребує фактів ситуації як вхідний, ситуація додається до правила (лінії 5 – 7) алгоритму. Потім правило активується. За допомогою мапи суміжних вершин продукт роботи правила додається до *WM* інших вузлів. У випадку якщо правило не має суміжних вершин його продукт додається до масиву результатів *Q*. Масив результатів може бути використаний для побудови експертного висновку. Перед повторним запитом користувача до системи масив результатів повинен бути очищений від результатів попереднього запиту.

Швидкість роботи алгоритму запуску правил можна оцінити як $O(R_d+E_d)$, де R_d це кількість вершин, E_d кількість ребер між R_d. У найгіршому випадку кількість ребер графу може складати не більше ніж R_d^2, у найліпшому випадку кількість ребер САГ може бути (R_d - 1). Тому у найгіршому випадку складність алгоритму становить $O(R_d+R_d^2)$. Таким чином алгоритм виконання правил є максимально швидкий алгоритм при умові що немає можливості стискати правила.

3.5 Оцінка швидкості роботи алгоритму

Двигун висновків в ході виконання своєї функції виконує декілька послідовних операцій: завантаження правил з бібліотеки, дискримінація правил які не можуть бути теоретично виконані, побудова інструкцій виконання і запуск правил. Схема виконання правил двигуном висновків представлена на рисунку 3.6.

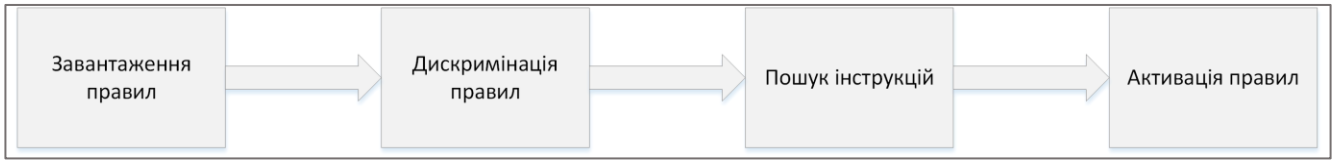

Рисунок 3.6 Схема роботи двигуна висновків.

Сукупний час роботи алгоритму можна оцінити як сума виконання всіх етапів роботи двигуна висновків. Швидкість роботи алгоритму залежить від кількості ребер графу, що представляє мережу правил. Зв'язки між правилами можуть бути непрогнозовані, тому доцільно подати максимально можливу (9) та мінімально можливу кількість операцій за один запит (10).

$$N_{max} = (\mathrm{R} + R^2) + \mathrm{S}(\mathrm{R} + \mathrm{E}) + R_d^2 + (R_d + R_d^2), \quad (9)$$

$$N_{min} = (\mathrm{R} + R^2) + \mathrm{S}(\mathrm{R} + \mathrm{E}) + R_d^2 + (2R_d - 1) \quad (10)$$

де R – загальна кількість правил;

S – кількість вхідних фактів;

E – кількість зв'язків між всіма правилами;

R_d – кількість обраних правил.

При умові використання пам'яті можна значно скоротити час виконання алгоритму до оптимального $O(R_d+E_d)$, де R_d – кількість необхідних для прийняття експертного рішення правил, E_d – кількість зв'язків між цими правилами. Для цього потрібно організувати структуру даних здатну зберігати "ключ – значення". При першому запиті до двигуна можна визначити набір інструкцій.

4 ІМПЛЕМЕНТАЦІЯ ТА ТЕСТУВАННЯ

4.1 Вибір мови програмування для імплементації моделі

При виборі мови для імплементації повинні бути враховані декілька умов. По-перше, мова програмування повинна бути об’єктно-орієнтована. Тобто за її допомогою можна буде створити об’єкти моделей які будуть мати тільки стан виражений у полях класу та об’єкти правил які повинні містити бізнес-логіку обробки моделей. По-друге, цією мовою програмування повинні бути вже реалізовані експертні системи та їх двигуни висновків, що дасть змогу порівняти розроблювану модель з аналогами та таким чином довести її ефективність і швидкість роботи. Потрете мова програмування повинна бути досить поширена, щоб не виникало проблеми знайти спеціаліста який би підтримував бібліотеки експертної системи. По-четверте мова програмування не повинна залежати від платформи, що повинно підвищити популярність моделі в майбутньому. Для прийняття рішення ці умови представлені у вигляді таблиці 1.

Таблиця 1 – Порівняльна таблиця мов програмування

№	Назва мови програмування	Відсоток	ООП	Кросплатформеність
1	Java	16.00%	+	+
2	C	14.24%	+	-
3	C++	8.10%	+	-
4	Python	7.83%	+	+
5	Visual Basic .NET	5.19%	+	-

Згідно з індексом TIOBE за травень 2019 року до п’ятірки найпопулярніших мов програмування входять Java, C, C++, Python, Visual Basic .NET. Всі ці мови програмування підтримують об’єктно-орієнтований підхід для програмування, але не всі вони є незалежними від апаратної частини, чи операційної системи. Такі мови програмування як C та C++ залежні від апаратної частини. Якщо розробляти бібліотеки правил на бозі апаратної цих мов програмування треба буде компілювати

окремі варіанти бази знань для різних апаратних комп'ютерних систем. Visual Basic .NET залежить від операційної системи Windows і хоча існують шляхи портуваня на операційні системи Linux такі системи досі не використовуються для серйозних проектів. Python це скриптова мова програмування яка не залежить від платформи та має інтерпретатори для широкого кола як апаратного забезпечення так і для операційних систем. Вона достатньо поширена. Але завдяки тому, що Python скриптова мова програмування. Швидкість її виконання в порівнянні з мовами програмування які компілюються не дуже висока. Тому це значно знижує потенціал моделі.

Java вже давно займає місце найпоширенішої мови програмування. Це мова яка компілюється в байт код який використовує Java Runtime Environment для побудови прямих інструкцій процесору мовою програмування Assembler. Існує багато версій JRE для різних платформ та операційних систем. JRE підтримує виконання старого коду тобто виконується девіз “написано один раз виконується завжди.” Java є найпопулярнішою мовою програмування для корпоративних додатків. Вона широко використовується завдяки безпрецедентній стабільності та низької ціни модернізації додатків які на ній написані. Бази знань реалізовані мовою програмування Java не доведеться оновлювати з виходом нової версії мови програмування. Це знизить вартість підтримки бази знань. Існує досить велика кількість двигунів висновків розроблених цією мовою програмування таких як Drools, OpenRules, EasyRules, JLisa, Jess. Декілька з них можуть бути обрані для порівняльного тестування для визначення переваг та недоліків.

Таким чином для імплементації моделі була обрана мова програмування Java. Доведення можливості імплементації моделі цією мовою програмування повинно стати відправною точкою для подальших досліджень і розробки більш досконалих і реальних програмних продуктів. Потрібно порівняти ефективність розробленого алгоритму роботи з реальними системами двигунів висновку. Це дасть можливість оцінити слабкі та сильні сторони розробленої моделі продукційної системи. Слабкі сторони продукту повинні бути враховані при подальшому вдосконаленню моделі продукційної системи.

4.2 Розробка UML моделі

Перед початком імплементації потрібно розробити модель компонентів фреймворку та UML модель інтерфейсів. Це необхідно для чіткого і зрозумілого процесу імплементації. Оскільки фреймворк складається з декількох незалежних логічних частин доцільно поділити його на три модулі. Перший модуль це набір інтерфейсів бібліотеки правил. Другий модуль це набір інтерфейсів двигуна висновків. Третій модуль це безпосередньо імплементація двигуна висновків. Такий поділ дозволить використовувати компоненти які складають з інтерфейсів окремо для використання при розробці програмних продуктів. Модель компонентів та їх взаємодія змальована на рисунку 4.1.

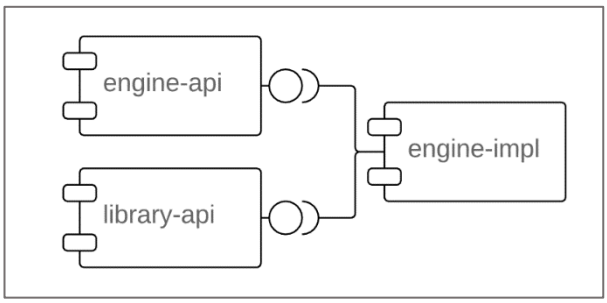

Рисунок 4.1 Модель компонентів двигуна висновків

Модуль бібліотеки правил складається з інтерфейсів Data, Provider, Rule. Інтерфейс Data не містить ніяких методів це просто інтерфейс який маркує клас моделі. Моделі мають тільки стан і не мають ніякої логіки. Інтерфейс Rule відбиває логіку поведінки правила яке буде використовувати двигун висновків. Він має чотири метода: add(), run(), getOutput(), isInput(). Метод add() приймає на вхід та додає об’єкт моделі до правила. Метод run() валідує всі класи моделі та виконує логіку правила “якщо-то”. Метод getOutput() повертає назву класу моделі яка є продуктом роботи правила. Метод isInput() приймає на вхід клас моделі і повертає істину якщо об’єкт класу моделі може бути оброблений правилом. Інтерфейс Provider має тільки один метод getAll(). Він повертає список об’єктів всіх правил

бібліотеки. На рисунку 4.2 представлена повна UML діаграма можливої імплементації інтерфейсів розробленої моделі та їх взаємодія.

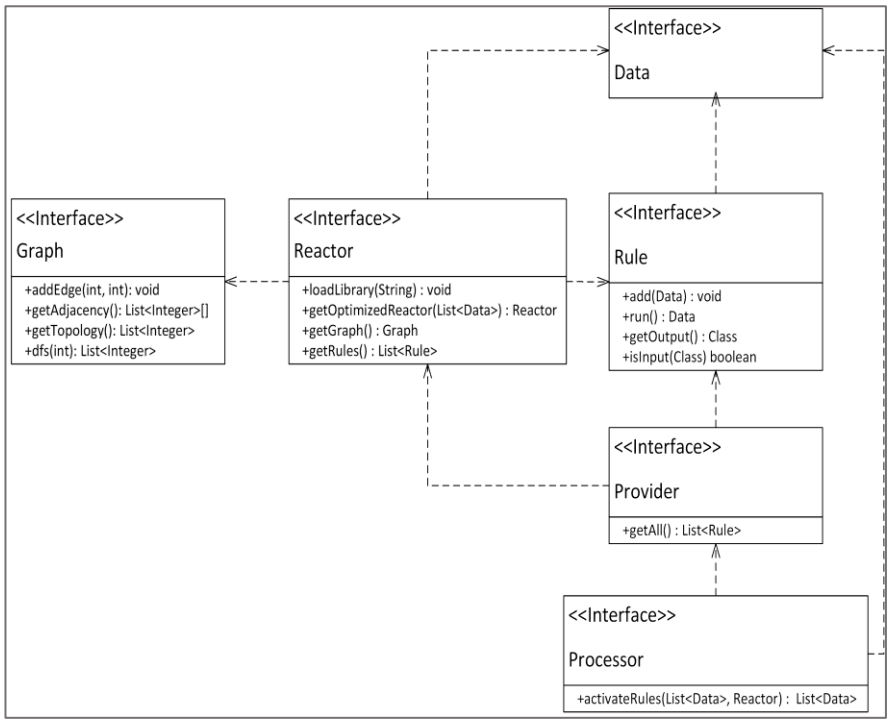

Рисунок 4.2 UML модель двигуна висновків

Модуль інтерфейсів двигуна висновків складається з наступних інтерфейсів: Graph, Processor, Reactor. Інтерфейс Graph містить в собі логіку необхідну для створення такої структури даних як спрямований ациклічний граф та використання деяких його якостей. Інтерфейс містить чотири методи addEdge(), getAdjacency(), getTopology(), dfs(). Метод addEdge() приймає на вхід та додає спрямоване ребро графа у вигляді пар елементів початкової вершини та кінцевої вершини. Метод getAdjacency() повертає список суміжних вершин для кожного вузла графа. Метод getTopology() повертає топологію графу. Метод dfs() приймає на вхід початкову вершину для пошуку та повертає всі вершини графу які були знайдені в процесі пошуку в глибину графа. Інтерфейс Reactor складається з наступних методів: loadLibrary(), getOptimizedReactor(), getGraph(), getRules(). Метод loadLibrary() приймає на вхід бібліотеку правил та зберігає у вигляді списку всі правила у вигляді спрямованого ациклічного графа всі взаємодії між правилами зумовлених моделями їх вхідних та вихідних об’єктів. Метод getOptimizedReactor() приймає на

вхід масив моделей які утворюють вхідну ситуацію для експертної системи. Для кожної моделі проводиться пошук в глибину графа з метою знайти тільки ті вузли графа, тобто правила, які можуть бути використані для його обробки. Метод повертає об’єкт класу Reactor який має набір вузлів які можуть бути використані для обробки вхідної ситуації. Метод getGraph() повертає граф який міститься в об’єкті класу Reactor. Метод getRules() повертає список об’єктів всіх правил які містяться в об’єкті класу Reactor.

4.3 Розробка підходів до тестування

Оскільки мовою імплементації моделі для тестування була обрана Java потрібно обрати декілька двигунів висновків написаних цією мовою програмування які реалізують різні алгоритми продукційних систем. Найпопулярнішими двигунами висновків написаними на Java є Drools, OpenRules, EasyRules, JLisa, Jess. Drools найпопулярніша система керування бізнес-правилами, яка забезпечує основний движок бізнес-правил, програму для створення веб-сторінок і управління правилами та додаток Eclipse IDE для розробки основних компонентів системи. OpenRules пропонує загальну систему управління рішеннями на основі бізнес-правил. EasyRules надає абстракцію правила для створення правил з умовами та діями, та двигун висновків API RulesEngine, який проходить через набір правил для оцінки умов і виконання дій. Правила пишуться мовою програмування Java. JLisa це CLISP-подібний движок правила доступний з Java з повною потужністю Common Lisp. Jess невеликий, легкий і швидкий двигун висновків і сценаріїв, написані повністю на Java.

Для проведення порівняльного тестування були обрані два кандидати Drools та EasyRules. Drools був обраний тому що він є найпопулярнішим на даний момент двигуном висновків. Drools споживає правила написані на спеціальному діалекті який дозволяє у якості моделей використовувати об’єкти класів написаних па мові

програмування Java. Він вимушений використовувати спеціальний інтерпретатор для виконання правил. Drools використовує модифікований rete-алгоритм для оптимізації та стиснення правил, а також для побудови порядку виконання правил. EasyRules був обраний тому, що використовує у якості правил об’єкти Java класів і не має інтерпретатора тому теоретично повинен працювати швидше ніж Drools. EasyRules використовує у вигляді продукційного алгоритму правила маркова. Тобто є необхідність прописувати чітку послідовність виконання правил, бо в інакшому випадку продукт роботи правила може з’явитися пізніше ніж правило, що його потребує буде застосовано. Тому теоретично двигун висновків повинен бути трохи повільнішим за модель, що розроблюється.

Всі тести були проведені в однакових умовах апаратного та програмного середовища. Апаратні умови виконання тестування проходили в наступних умовах. Процесор Intel Core i7-8550U 4-cores 8-threads 4GHz, оперативна пам’ять DDR4 - 2133 МГц 16GB. Для отримання виконання коду та отримання результатів швидкодії був обраний бенчмарк фреймворк спеціально розроблений для тестування швидкодії виконання Java коду JMH - Java Microbenchmark Harness. Рекомендованим способом запуску тесту JMH є використання збірника пакетів Maven для налаштування автономного проекту, який залежить від файлів jar бібліотек програми. Такий підхід гарантує, що контрольні показники будуть правильно ініційовані та результати будуть надійними. Можна запускати тести з наявного проекту, і навіть з самого середовища, однак налаштування є складнішим і результати менш надійні. Бібліотеки jar будуть запускатися з ключем “-prof gc”. Це дозволить знайти одразу декілька параметрів за якими можна скласти повну картину ефективності виконання коду. Цими параметрами являються наступні: швидкість операцій в одиницю часу, об’єм виділеної оперативної пам’яті, час роботи збірника сміття. В ході дослідження буде проведено серію однакових тестів для кожного з продуктів. Тести запускали послідовне виконання різної кількості правил, від 1-го до 10-ти. Кожне правило, виконуючись виробляє певний факт. Цей факт використовує наступне правило.

4.4 Результати тестування

Для кожного з двигунів висновків які тестувалися була створена найліпша імплементація виконуваного сценарію тестування. Для проведення тестування були зібрані три jar файли які можуть бути запущені з командної строки з тестами продуктивності JMH. Тести продуктивності були запущений з наступними параметрами. Виконання тестів продуктивності проводилося один раз. Розігрів JVM для виконаного коду проводився 5 п’ять разів до заміру.

Для забезпечення комплексної картини роботи двигунів висновків для кожного сценарію проводилися заміри трьох показників: кількість операцій в секунду, виділена оперативна пам’ять, та час роботи збірника сміття. Кількість операцій в секунду показує скільки разів в секунду може бути виконаний код який тестується. Чим вище цей показник тим краще. Наступний показник, виділена оперативна пам’ять за операцію, яка кількість оперативної пам’яті виділяє віртуальна машина Java під час виконання операції. Цей показник характеризується чим менше тим краще. Показник час роботи збірника сміття дуже важливий бо він стосується роботи самої віртуальної машини. Збірник сміття впливає на час роботи додатку в цілому. Збірник сміття у віртуальній машині Java повинен періодично самостійно видаляти об’єкти з оперативної пам’яті на які немає посилань в додатку. Чим менший цей показник тим краще.

Результати тестування двигунів висновків показали, що як і передбачалося двигун висновків Drools буде найповільнішим рішенням. Це пов’язано з необхідністю співставляти вхідні інформаційні об’єкти із мережею предикатів бізнес-правил. Трохи швидшим виявився двигун висновків EasyRules. Його швидкість забезпечують написані на мові програмування Java правила, але його вповільнює процес розбору правила за допомогою рефлексії та продукційна модель Маркова, що лежить в його основі. Найшвидшою виявилася імплементація розроблюваної моделі *X*. Особливо видатні результати показала модель з використанням пом’яті *X+memory*. Результати тестування приведені у таблиці 2.

Таблиця 2 – Результати тестування

Кількість правил	Тип вимірювань	од. в.	Drools	EasyRules	X	X+Memory
1	Операцій в секунду	опр/с	16818,185	99238,699	736625,634	10961690,648
	Виділення пам’яті операції	Б/опр	55969,118	25248,007	2168,001	328,000
	Час роботи збірника сміття	мс	517,000	87,000	86,000	83,000
2	Операцій в секунду	опр/с	15503,489	53274,904	678570,837	8742129,192
	Виділення пам’яті операції	Б/опр	59120,029	52352,014	2280,001	304,000
	Час роботи збірника сміття	мс	450,000	35,000	85,000	77,000
3	Операцій в секунду	опр/с	14385,258	38063,044	616252,997	5726800,778
	Виділення пам’яті операції	Б/опр	62272,369	80992,020	2424,001	352,000
	Час роботи збірника сміття	мс	425,000	90,000	86,000	83,000
4	Операцій в секунду	опр/с	14918,214	27452,328	557227,832	4170245,239
	Виділення пам’яті операції	Б/опр	65427,381	111984,030	2576,001	392,000
	Час роботи збірника сміття	мс	421,000	71,000	80,000	83,000
5	Операцій в секунду	опр/с	13751,477	22811,659	507763,142	3547440,565
	Виділення пам’яті операції	Б/опр	68575,722	140704,036	2752,001	440,000
	Час роботи збірника сміття	мс	386,000	89,000	77,000	81,000
6	Операцій в секунду	опр/с	12951,352	18427,959	428370,421	3171464,059
	Виділення пам’яті операції	Б/опр	71728,192	173584,045	2904,001	480,000
	Час роботи збірника сміття	мс	383,000	92,000	74,000	81,000
7	Операцій в секунду	опр/с	12365,963	15952,784	383623,525	2652088,745
	Виділення пам’яті операції	Б/опр	74880,355	202080,053	3080,001	528,000
	Час роботи збірника сміття	мс	368,000	59,000	77,000	79,000
8	Операцій в секунду	опр/с	11840,686	14452,374	359272,199	2341127,073
	Виділення пам’яті операції	Б/опр	78032,478	238440,060	3232,001	568,000
	Час роботи збірника сміття	мс	355,000	85,000	68,000	82,000
9	Операцій в секунду	опр/с	11593,190	12663,882	322725,793	2132158,441
	Виділення пам’яті операції	Б/опр	81184,555	273200,067	3440,001	616,000
	Час роботи збірника сміття	мс	347,000	68,000	67,000	79,000
10	Операцій в секунду	опр/с	11233,706	11044,308	327139,400	1832249,018
	Виділення пам’яті операції	Б/опр	84626,503	307896,077	3592,001	656,000
	Час роботи збірника сміття	мс	368,000	92,000	70,000	73,000

За допомогою отриманих тестових даних було побудовано діаграму використання пам’яті для двигунами висновків за кожну тестову операцію. Діаграма представлена на рисунку 4.3.

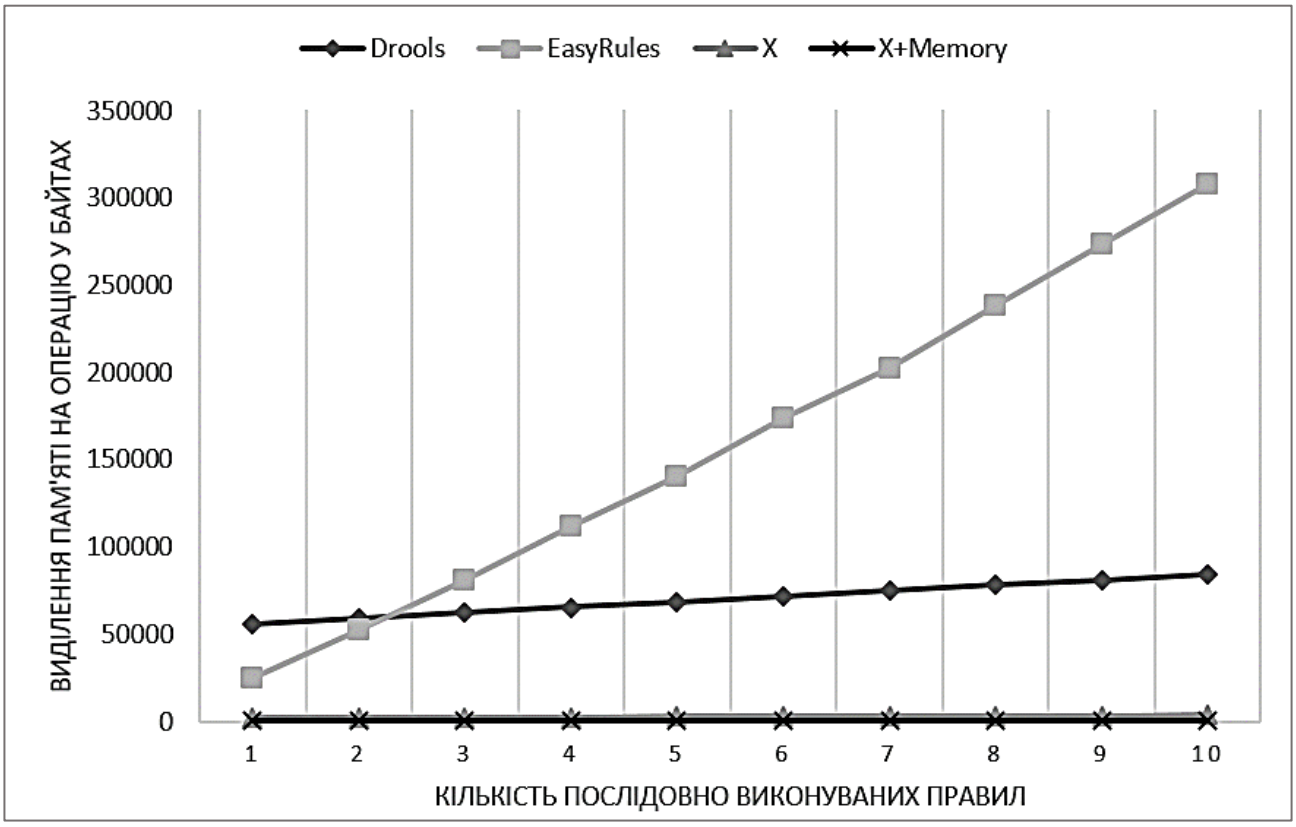

Рисунок 4.3 Діаграма використання пам’яті

Представлена діаграма наочно демонструє лінійну залежність виділення пам’яті від кількості виконуваних правил. Досліджуючи діаграму візуально можна прийти до наступного ряду висновків. По-перше можна оцінити кут нахилу моделі. Чим менший кут нахилу до осі кількості правил, тим менша залежність виділеної пам’яті від кількості виконуваних послідовно правил. За цим критерієм найгіршим виявився двигун висновків EasyRules. Набагато ліпшим за нього виявився двигун висновків Drools. Розроблена модель показала найліпший результат. Також графічно можна оцінити кількісні показники використання пам’яті. Порівнюючи двигуни висновків Drools та EasyRules можна сказати, що останній втрачає переваги вже після виконання двох послідовних правил. Порівняно з ними розроблена модель двигуна висновків демонструє перевагу у декілька порядків.

Отримані результати тестування дозволили накопичити достатньо даних для проведення математичного моделювання. Математична модель повинна показати залежність швидкості роботи двигуна правил від кількості послідовно виконаних правил. Для цього кожен з двох наборів даних був перевірений на відповідність одній з трьох основних математичних моделей: лінійної моделі (11), гіперболічної моделі (12) та параболічної моделі (13).

$$y = a + bx\ , \tag{11}$$

$$y = a + b/x\ , \tag{12}$$

$$y = a + bx + cx^2 \tag{13}$$

де a – вплив незалежний від змінної;

b, c – вплив залежний від змінної;

x – незалежна змінна.

Моделювання проводилося в MATLAB за допомогою вільної бібліотеки APMonitor Optimization Suite. Результати представлені у вигляді рівнянь. Для двигуна правил Drools найкращим чином підійшла лінійна модель (14). Для двигуна правил EasyRules найкращим чином підійшла гіперболічна модель (15). Для розробленої моделі без пам’яті (16). Для розробленої моделі з пам’яттю (17).

$$y_{Drools} = 16865{,}20 - 605{,}36x \tag{14}$$

$$y_{\text{EasyRools}} = 2606{,}41 + 98093{,}22/x \tag{15}$$

$$y_X = 760469.73 - 48856.93x \tag{16}$$

$$y_{X+memory} = 1415242.92 + 10626595.48/x \tag{17}$$

де x – кількість правил.

Для оцінки якості отриманих моделей використовувався коефіцієнт детермінації між реальними даними, зазначеними в таблиці 1, і розрахунковими даними моделі. Отримані коефіцієнти детермінації для моделей Drools і EasyRules склали 0,976 і 0,998 відповідно. Для розроблюваної моделі X, та її варіації з

пам’яттю X+Memory коефіцієнти кореляції склали 0,983 та 0,960 відповідно. Це свідчить про високу якість знайдених моделей. На рисунку 4.4 представлені діаграми знайдених моделей.

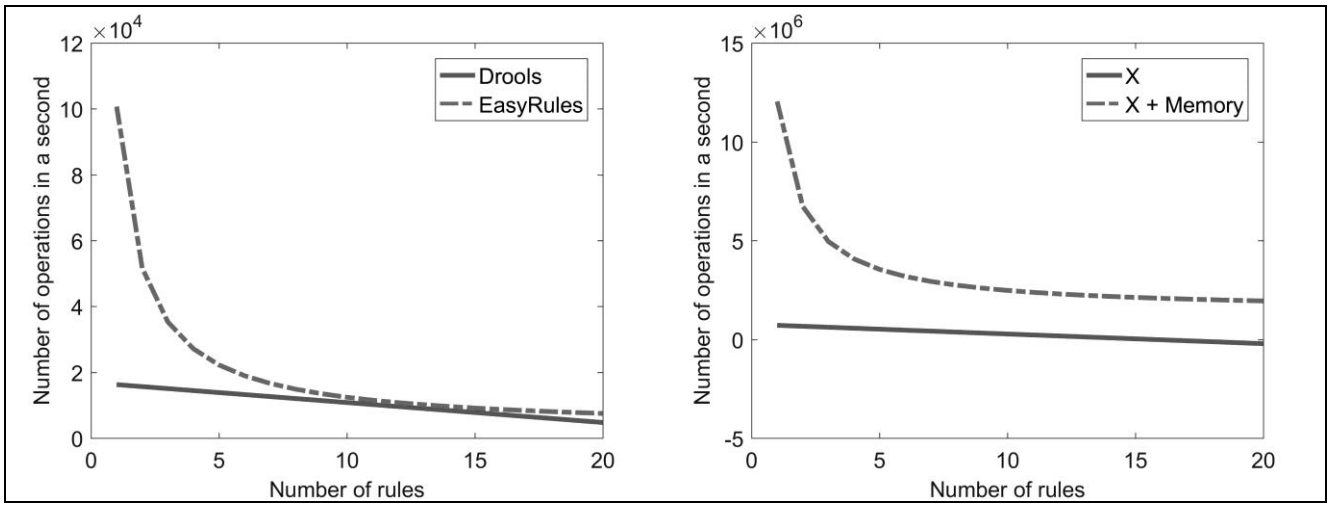

Рисунок 4.4 Діаграма математичних моделей двигунів висновків

Виходячи з результатів порівняльного дослідження найгірший результат показав двигун висновків Drools. Не дивлячись на схожість математичних моделей Drools та розробленої моделі без використання пам’яті, остання набагато швидша, та менш залежна від кількості правил. При використанні пам’яті в розробленій моделі її результат значно покращується. При цьому багатократне пришвидшення спостерігається при кількості правил від одного до п’яти. Модель з пам’яттю схожа за своєю поведінкою з двигуном висновків EasyRules, але набагато перевершує її в швидкості обробки бізнес-правил.

Таким чином в результаті проведеного дослідження було доведено, що розроблена модель керування правилами та розв’язання задач без заданого кінцевого стану набагато перевищує за швидкодією деякі з існуючих на ринку систем керування бізнес-правилами. При цьому використовуючи значно меншу кількість оперативної пам’яті. Отримані математичні моделі доводять розробленої моделі двигуна висновків щодо кількості послідовно виконуваних правил, особливо при їх невеликій кількості. Вона може бути використана у багато навантажених проектах де швидкість системи має велике значення.

5 ОГЛЯД РОЗРОБЛЕНОЇ МОДЕЛІ

5.1 Огляд переваг та недоліків розробленої моделі

Основною перевагою розробленої моделі є швидкість роботи. Швидкість роботи двигуна висновків порівнюється зі швидкістю виконання байт коду Java. Всі покращення віртуальної машини Java будуть відбиватися на швидкості роботи двигуна висновків. На відміну від класичних моделей розроблена в розрізі парадигми програмування потоку даних, тому не містить робочої пам’яті яка повинна зберігати всі проміжні факти які були знайдені під час виконання правил. Модель може бути реалізовані без використання сторонніх бібліотек і складається з шести інтерфейсів і трьох імплементацій. У порівнянні з іншими двигунами висновків розроблені фреймворки на базі цієї моделі будуть мати невеликий розмір.

Бази знань представлені у вигляді бібліотек класів автоматично здобувають всі переваги пов’язані з цим. З’являється можливість використовувати версії бібліотек. Версії бази знань надають можливість створювати нові версії бібліотек і переходити від старої до нової версії тільки при досягненні її стабільної роботи. Також це надає можливість легко створювати нові бібліотеки на базі старих бібліотек правил. При цьому версії бібліотек жорстко пов’язані та комплексні бібліотеки працюють стабільно. Також при розробці правил можна використовувати всі можливості мови програмування і її бібліотек.

Моделі та правила задаються однією мовою програмування. Різні бази знань можуть легко поділяти бібліотеки моделей. Крім того правила з різних бібліотек можуть працювати разом якщо вони завантажені у двигун висновків. Двигун висновків автоматично поєднає правила які можуть працювати разом.

Таким чином модель має багато переваг над вже існують розробленими двигунами висновків. Вона швидша, використовує менше апаратних ресурсів та може обробляти складну бізнес логіку додатків безпосередньо в межах розробленого двигуна висновків. Не зважаючи на переваги моделі у швидкості роботи в порівнянні з іншими аналогами потрібно підкреслити, що розроблена

модель має ряд функціональних недоліків якими довелося пожертвувати завдяки прискоренню швидкості роботи та використанню меншої кількості пам’яті. Першим недоліком моделі є відмова від правил які могли бути легко прочитані та відредаговані людиною без спеціальних навиків програмування. Одним зі шліфів розв'язання цієї проблеми є системи візуального програмування. Теоретично їх можна використати для побудови правил без залучення програміста. Крім того системи візуального програмування можуть візуально представляти мережу правил, що значно полегшить розробку складної бази знань з великою кількістю пов’язаних між собою правил.

Одним із недоліків системи є неможливість виконання одного і того самого правила декілька разів у межах одного експертного рішення. Виконання циклів можна вирішити шляхом внутрішньої рекурсії в межах одного правила.

Залежність бази знань від мови на якій вона реалізована теж є суттєвим недоліком. Для розв'язання цієї проблеми можна використовувати конвертори між різними об’єктно орієнтованими мовами програмування. Проте ця проблема навряд може виникнути для корпоративних додатків, більшість з них реалізована на мові програмування Java яка є те факто стандартом для великих корпоративних додатків із-за стабільності роботи і відсутності складності переходу на нові версії.

Також потенціальною проблемою є складність побудови бібліотеки правил залежної від великої кількості бібліотек які можуть теж залежати від великої кількості бібліотек. Цю проблему можна вирішити використовуючи збірники пакетів для платформи Java. Наприклад, Maven це – інструмент для збирання Java проектів: їх компіляції, створення jar файлів, створення дистрибутиву програми, генерації документації тощо. Прості проекти можна зібрати в терміналі. Але якщо збирати великі проекти з термінала, то команда для збору буде дуже довгою, тому вона може бути записана в bat скрипт. Скрипи залежні від платформи на якій вони запускаються. Для того, щоб відмовитися від цієї залежності, і заповнити написаний скрипт використовують інструменти для збірки проектів. Вони надають можливість автоматизувати збірку проектів з усіма потрібними бібліотеками.

5.2 Огляд шляхів подальшого розвитку та вдосконалення моделі

Одним з першочергових шляхів вдосконалення моделі є поліпшення швидкості її роботи. Загальний алгоритм роботи двигуна висновків складається з декількох незалежних алгоритмів які розв’язують різні математичні проблеми. Наприклад, в розробленій моделі використовується алгоритм Кана для побудови топології графа. Однак при проектуванні моделі не були розглянуті всі можливі алгоритми побудови топології. Порівняння декількох реалізованих алгоритмів на різних мовах програмування можуть мати різний ефект. Підвищення швидкості побудови топології графа, пришвидшить алгоритм в цілому. Також перспективним напрямком поліпшення моделі є можлива адаптація алгоритму до мов програмування задля швидшої імплементації двигуна висновків.

Бібліотеки в розробленій моделі можуть надавати двигуну висновків тільки масив правил. Це зроблено для того щоб правила могли бути обрані з декількох бібліотек, а потім з всіх правил разом був створений загальний граф правил. Проте у випадку якщо двигун використовує тільки одну бібліотеку, вона одразу повертала заздалегідь створений граф. Таке нововведення повинно значно прискорити старт системи особливо при великій кількості правил. Моделі непотрібно буде будувати граф кожного разу при старті системи. Однак такий підхід має і недоліки. Всю відповідальність за створення графа бере на себе розробник бібліотеки правил, а це означає можливу наявність помилок. Для розв'язання цієї проблеми можна буде використовувати спеціально розроблений валідатор графів.

Під час своєї роботи експертні системи стикаються з великою кількістю однотипних запитів, на які продукуються однакові відповіді. Тому для підвищення швидкодії експертної системи будованої на базі розробленої моделі можна використовувати хешування запитів і вироблених відповідей. В розробленій моделі існує досить велика кількість місць в яких може бути застосований такий підхід. Можна зберігати топологію, дискримінований граф правил, або навіть безпосередньо запити. Хешування активно використовується у вже розроблених

двигунах висновків. Наприклад, двигуну висновків Drools хешування дозволяє миттєво повертати вже розрахований результат у випадку коли до системи надходить однотипний запит. На жаль у хешування існує ряд обмежень. При кожній зміні набору правил хешування втрачає свою актуальність. Тому такий підхід не може бути застосований в системах з динамічною зміною правил.

Одним зі слабких місць моделі є процес дискримінації правил. Для цього використовується пошук в глибину графа. Однак це викликає неоднозначну поведінку моделі. Пошук в глибину знаходить всі вершини до яких можна дістати з заданої. Але може виникнути ситуація коли у знайденої вершини існують один чи декілька вхідних фактів які не є вхідними фактами до системи або не можуть бути розраховані з вхідних фактів. На даному етапі розвитку моделі передбачається, що правило має базову модель вхідного факту із самого початку свого існування. Але подібна поведінка суперечна. Вона нав’язана неякісною роботою алгоритму пошуку в глибину для знаходження правил. Майбутні варіанти моделі повинні вирішити недоліки використання алгоритму пошуку в глибину, або вдосконаливши його для вирішення цієї задачі, або використавши новий алгоритм.

Крім вищезгаданих варіантів покращення роботи моделі, можна розглянути розгортання комплексної інфраструктури навколо неї, яка дозволить розробникам та користувачам отримувати максимум від її використання. Для розробників бібліотек правил будуть дуже корисні інструменти тестування правил, знаходження помилок в них та перевірку коректності їх взаємодії. Існує багато можливостей для реалізації цих ідей. Від розробки спеціальних тестових бібліотек до реалізації розширень для популярних додатків розробника. Для користувачів може бути запропонований спеціальний веб ресурс на якому будуть зберігатися вже розроблені бібліотеки правил. Такий підхід дозволить як користувачам так і розробникам використовувати завжди актуальні версії бібліотек правил.

Таким чином існує дуже багато шляхів покращення ефективності роботи розробленої моделі двигуна висновків. Вони стосуються як покращення алгоритму продукційної системи, так і підходів до його імплементації.

5.3 Потенційні галузі застосування розробленої моделі

Пер за все розроблена модель може бути використана при побудові експертних систем. Властивості моделі дозволяють використовувати їх для створення швидких експертних систем які використовують мінімальну кількість апаратних ресурсів. Це може значно розширити коло їх застосування. Наприклад велика швидкість роботи може стати в нагоді при розробці високонавантажених веб додатків які повинні обробляти вхідну інформацію на основі відомого набора правил. Наприклад, такими сервісами можуть бути онлайн сервіси виправлення граматичних помилок, онлайн сервіси розрахунку маршрутів, банківські онлайн сервіси, сервіси планування розкладів тощо.

Окрім швидкості розроблена модель характеризується дуже незначним використанням оперативної пам’яті і може бути використана для створення систем здатних працювати в умовах сильного обмеження апаратних ресурсів. Розроблена модель може бути використана для створення мобільних додатків здатних реалізовувати експертні системи які не будуть потребувати виходу в інтернет і зв’язку з додатковими сервісами для отримання експертного рішення. Такі системи зможуть бути використані незалежно не впливаючи на швидкість роботи мобільного пристрою і не заважаючи іншим додаткам.

Маленький бібліотеки здатний знайти ще одну галузь використання систем на основі розробленої моделі. Такі системи можуть бути застосовані для створення розумних речей, здатних використовувати логіку експерта, та приймати складні галузеві рішення. Наприклад вони можуть бути використані в медичній галузі для створення розумних протезів, де логіка руху людини може бути представлена у вигляді правил. Або їх можна застосувати при розробці розумних побутових пристроїв які будуть автоматизувати роботу приміщення на основі складних бізнес-правил замість використання прописаних в коді алгоритмів.

Таким чином розроблена модель завдяки своїм властивостям може не тільки замінити вже існуючі продукти, а й відкрити нові галузі застосування.

ВИСНОВКИ

В ході виконання науково-дослідницької роботи було проведено дослідження об’єкта магістерської атестаційної роботи, а саме одного із різновидів програм штучного інтелекту - експертної системи, яка здатна розв'язувати завдання з не визначеним кінцевим станом. Були досліджені різноманітні підходи та алгоритми роботи систем розв’язання, що імітують людський підхід до розв’язання проблем.

Завдяки детальному аналізу попередніх досліджень були виявлені сильні та слабкі сторони алгоритмів на яких базуються сучасні двигуни висновків експертних систем. Була поставлена мета дослідження - розробка моделі двигуна висновків яка б використовувала невеликий об’єм апаратних ресурсів і при цьому не поступалася, а бажано перевищувала, за швидкодією сучасні двигуни висновків.

Для виконання розробки моделі були зібрані та проаналізовані теоретичні матеріали та підходи з різних галузей знань, а також враховані позитивні сторони існуючих алгоритмів продукційних систем двигунів висновків. Одними із ключових отриманих знань стосувалися властивостей спрямованого ациклічного графа і парадигми програмування потоку даних. Розроблена модель була представлена у вигляді набору послідовних алгоритмів, які поетапно описують роботу моделі двигуна висновків. Була побудована та описана UML модель, яка може бути реалізована на будь якій об’єктно-орієнтованій мові програмування. Була створена тестова імплементацій для порівняння розробленої моделі із сучасними аналогами. Результати показали значні переваги в швидкості роботи при мінімальному використанні комп’ютерної оперативної пам’яті. Подальший розвиток дослідження полягає в поліпшенні алгоритму продукційної системи.

Таким чином в ході проведеної науково дослідницької роботи була розроблена та протестована модель двигуна висновків експертної системи з новим алгоритмом роботи продукційної системи. Були розв’язані задачі зменшення кількості використовуваних апаратних ресурсів, швидкодії експертної системи та одночасно виконання складної бізнес-логіки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is a Business rule management system? *Medium* : веб-сайт. URL: https://medium.com/@ryanjollyyoung/what-is-a-business-rule-management-system-39ebcb7900c7 (Дата звернення: 15.05.2019)

2. Georgiana M. Decision support systems. *Romanian Economic Business Review*. 2019. №. 2. P. 513-520.

3. Nižetić I., Fertalj K., Milašinović B. An Overview of Decision Support System Concepts. *Proceedings of the 18th International Conference on Information and Intelligent Systems*. Varaždin : 2007. P. 251-256.

4. Wright A., Sittig D. A framework and model for evaluating clinical decision support architectures. *Journal of Biomedical Informatics*. 2008 Vol. 41, №. 6. P. 982-990. doi:10.1016/j.jbi.2008.03.009

5. Zhang S.X., Babovic V. An evolutionary real options framework for the design and management of projects and systems with complex real options and exercising conditions. *Decision Support Systems*. 2011. Vol. 51, №. 1. P. 119-129. doi:10.1016/j.dss.2010.12.001.

6. Poblete R.E., Fuente, D.D., Alonso M. Using Cloud Computing with RETE Algorithms in a Platform as a Service (PaaS) for Business Systems Development. *Proceedings of the 2011 International Conference on Artificial Inte*lligence. Las Vegas : Vol. 2, 2011. P. 654-658.

7. Cadet, a computer based decision support system for early cancer detection: a performance evaluation. *International Society for Preventive Oncology* : веб-сайт. URL: http://www.cancerprev.org/Journal/Issues/26/101/1291/4513 (Дата звернення: 16.05.2019)

8. Forsyth R.S. The Architecture of Expert Systems. Expert Systems: Principles & Case Studies / ed. R.S. Forsyth, London, 1984. – P. 9-17.

9. Inference engine. *EXPERT SYSTEMS* : веб-сайт. URL: https://brasil.cel.agh.edu.pl/~15sdrzeznik/en/system-wnioskujacy.html (Дата звернення: 16.05.2019)

10. Джарратано Дж., Райли Г. Экспертные системы: принципы разработки и программирование: 4-е издание. Москва : Вильямс, 2007. – 1152 с

11. Rete algorithm. *Wikipedia* : веб-сайт. URL: https://en.wikipedia.org/wiki/Rete_algorithm (Дата звернення: 16.05.2019)

12. Artificial intelligence and expert system. *PerfectLogic* : веб-сайт. URL: http://www.perfectlogic.com/articles/AI/ExpertSystems/ExpertSystems.html (Дата звернення: 17.04.2019)

13. World's fastest rules engine. *JAVAWORLD* : веб-сайт. URL: https://www.javaworld.com/article/2078197/world-s-fastest-rules-engine.html (Дата звернення: 16.05.2019)

14. Вакарь Л.Г. Концепція побудови продукційної системи для двигуна висновків експертних систем. *Реформування та розвиток гуманітарних та природничих наук* : матеріали наук.-практ. конф. 24-25 травня 2019 р. Херсон : Молодий вчений, 2019. С. 109-111.

15. How the Rete Algorithm Works? *Sparkling Logic* : веб-сайт. URL: https://www.sparklinglogic.com/rete-algorithm-demystified-part-2/ (дата звернення: 10.05.2019)

16. Bang-Jensen J. Digraphs: Theory, Algorithms and Applications : monography. London : Springer, 2008. – 795 p

17. Adamchik V. Graph Theory [Електронний ресурс]: лекції/ Carnegie Mellon University. URL: https://www.cs.cmu.edu/~adamchik/21-127/lectures/graphs_1_print.pdf (Дата звернення: 17.04.2019)

18. Mitrani I. Simulation techniques for discrete event systems : monography. Cambridge : Cambridge University Press, 1983. – 196 p

19. Kahn A. B. Topological sorting of large networks. *Communications of the ACM*. 1962. №. 11. P. 558–562.

20. Sedgewick R., Wayne K. Algorithms : 4th edition. Boston : Pearson Education, 2011. – 969 p

21. Sousa T. Dataflow Programming Concept, Languages and Applications. *Doctoral Symposium on Informatics Engineering 26-27 January 2012*. Porto : Faculdade de Engenharia da Universidade do Porto, 2012. P. 323-335.

22. Johnston W., Hanna P., Millar R. Advances in Dataflow Programming Languages. *ACM Computing Surveys*. 2004. March. P. 1-34.