ДИПЛОМНА РОБОТА
БАКАЛАВРА
ДРБ. ІП - 43.00.00.000 ПЗ
Група ІП-18-1
Фисун Тетяна
2022

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

УДК 004.4

ДИПЛОМНА РОБОТА БАКАЛАВРА

Тема: "Імплементація онтологічних моделей в контексті інженерії вимог"

(назва згідно з наказом ректора)

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДРБ.ІП - 43.00.00.000 ПЗ

(позначення)

Студент

ІП-18-1 /Т.Ю.Фисун/
(шифр групи) (підпис) (дата)(розшифровка підпису)

Керівник роботи

проф., д.т.н. /М.С.Пасєка/
(посада) (підпис) (дата) (розшифровка підпису)

Консультанти:

з питань нормоконтролю

доц., к.т.н /В.Я. Піх/
(посада) (підпис) (дата)(розшифровка підпису)

Допускається до захисту
Завідувач кафедри

проф., д.т.н. /В.І.Шекета/
(посада) (підпис) (дата) (розшифровка підпису)

2022
(рік)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**

проф. **В.І. Шекета**

“ ____ ” _____2022 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Фисун Тетяні Юріївні

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Імплементація онтологічних моделей в контексті інженерії вимог"

керівник проекту (роботи) Пасєка Микола Степанович, д.т.н., професор

затверджені наказом вищого навчального закладу від **“ __ ”** _травня 2021 р. № ____

2. Строк подання студентом проекту (роботи) 15 червня 2022 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1 Вступ до проблеми

2 Огляд існуючих концепцій , рішень та сервісів в даній області

3 Побудова моделі або алгоритму власного рішення

4 Документування реалізації власного оригінального рішення вибраними засобами

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 Інтерфейс запропонованої діалогової системи (рис. 3.1, ст. 48)

2 Класи, створені для запропонованої моделі онтології (рис. 3.2, ст. 49)

3 Властивості об’єкта, створені для запропонованої моделі онтології (рис. 3.3, ст. 49)

4 Властивості даних, створені для запропонованої моделі онтології (рис. 3.4, ст. 50)

5 Примірники , створені для запропонованої моделі (рис. 3.6, ст. 51)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **25 травня 2022 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми роботи	28.05.2022	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	31.05.2022	виконано
3	Побудова моделі або алгоритму власного рішення	03.06.2022	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	08.06.2022	виконано
5	Оформлення пояснювальної записки кваліфікаційної роботи	13.06.2022	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

В дипломній роботі бакалавра запропоновано підхід до інтерактивного виявлення вимог. Він використовує онтологію для представлення знань, пов’язаних із розробкою вимог, що спрямовує діалогову систему заповнення слотів для спілкування з клієнтами. За допомогою цього методу користувачі можуть налаштувати вимоги програми, які задовольняють їхні вимоги, взаємодіючи з машинами, при цьому забезпечується повнота та узгодженість налаштування.

КЛЮЧОВІ СЛОВА: ОНТОЛОГІЯ, ДІАЛОГОВА СИСТЕМА, ЛЮДИННО-МАШИННА ВЗАЄМОДІЯ, ІНЖЕНЕРІЯ ВИМОГ.

ANNOTATION

The bachelor's thesis proposed an approach to the interactive identification of requirements. It uses an ontology to represent knowledge related to requirements development, which guides a slot-filling dialogue system to communicate with customers. With this method, users can configure the application requirements that meet their requirements while interacting with the machines, while ensuring the completeness and consistency of the configuration.

KEY WORDS: ONTOLOGY, DIALOGUE SYSTEM, HUMAN-MACHINE INTERACTION, REQUIREMENTS ENGINEERING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 6

ВСТУП 7

1 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА СЕРВІСІВ В ПРЕДМЕТНІЙ ОБЛАСТІ 10

1.1 Лінія програмного забезпечення 10

1.2 Сервісно-орієнтована архітектура ..... **Помилка! Закладку не визначено.**

1.3 Інтерактивна розробка вимог 13

1.4 Система діалогу 14

1.5 Розробка вимог на основі знань 18

1.6 Онтологія 19

2 АЛГОРИТМІЧНІ ТА МАТЕМАТИЧНІ РІШЕННЯ ДЛЯ ЗАДАЧ ДАНОГО КЛАСУ 25

2.1 Вступ 25

2.2 Поняття 25

2.3 Відносини 27

2.4 Правила 31

2.5 Примірник онтології 33

2.6 Структура запропонованої системи діалогу 35

2.7 Процес визначення вимог 36

2.8 Налаштування вимог користувача 39

2.9 Результати вилучення вимог 40

2.10 Рекомендації щодо впровадження системи 46

3 ВЛАСНЕ ОРИГІНАЛЬНЕ АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ РІШЕННЯ ЗАДАЧІ 48

3.1 Реалізація 48

3.2 Приклад побудови рішення 51
3.3 Вилучення вимог 55
3.4 Генерація екземплярів проблеми 61
3.5 Експеримент із запропонованим методом 63
ВИСНОВКИ 66
СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА 69
БІБЛІОГРАФІЧНА ДОВІДКА

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ТЗ – Технічне завдання

БД – База даних

СУБД - Система управління базами даних

ПЗ - Програмне забезпечення

ВСТУП

Розробка лінійки програмних продуктів — це парадигма для розробки програмних додатків із багаторазовими програмними активами, які адаптуються до індивідуальних потреб клієнтів . Очікується, що за рахунок повторного використання засобів програмної інженерії (наприклад, програмних компонентів), а не розробки їх з нуля, програмні системи будуть налаштовані, а витрати можна ефективно скоротити. Тим часом, з тією ж основною метою повторного використання програмного забезпечення, сервіс-орієнтована архітектура (SOA) розділяє функціональні можливості системи на слабо пов’язані й повторно використовувані служби, які спілкуються один з одним за допомогою автономних повідомлень. Хоча SOA відрізняються, оскільки це різні парадигми програмної інженерії, багато в чому вони фактично доповнюють один одного . Повторне використання сервісів і застосування методів на основі SOA в розробці, особливо методів семантичної веб-служби (наприклад, автоматичне виявлення та складання сервісів) , можна досягти, щоб автоматизувати розробку програмного забезпечення. Крім того, основна увага в інженерії перейде з повторюваного проектування системи та впровадження налаштувань функціональних можливостей (тобто послуг).

З іншого боку, для того, щоб реалізувати повністю автоматизовану розробку, необхідний підхід для управління людино-машинною взаємодією при налаштуванні програмних продуктів. Однак керувати складністю та мінливістю функцій продукту, властивими лінійкам програмних продуктів, є дуже складним завданням . Крім того, досі відсутній допоміжний інструмент для керування автоматичною та інтерактивною настройкою продукту .

Рішення щодо автоматичного виявлення вимог є критичним для реалізації автоматизованого рішення. Незважаючи на те, що все більше публікацій у SOA присвячені проблемі автоматичної реалізації системи, мало досліджень досліджують автоматизацію розробки вимог.

Для автоматизації процесу розробки вимог, перш за все, необхідний допоміжний інструмент для взаємодії людини і машини, який використовується для здійснення комунікацій під час виявлення вимог. Тим часом, він повинен бути здатним керувати знаннями, пов’язаними з розробкою вимог, таким чином знання можуть бути природним чином представлені користувачам. Крім того, інструмент повинен мати можливість генерувати сервісно-орієнтовані результати для автоматизації впровадження системи.

З іншого боку, знання для автоматичного виявлення вимог мають бути представлені у форматах, зрозумілих для машин. Іншими словами, потрібен семантичний спосіб представлення знань. Більше того, як рішення для розробки знань для інженерії , воно повинно бути придатним для опису загальних і змінних ознак інженерії вимог програмних систем, які надаються. Оскільки головна проблема, пов’язана з розробкою вимог полягає в підтримці повноти та узгодженості продуктів із вимогами, необхідно правильно їх вирішувати. І останнє, слід дослідити підхід до вираження знань про взаємодію людини та машини.

Щоб полегшити реалізацію автоматизованого рішення на основі SOA, ця робота представляє інтерактивний підхід на основі діалогу для керівництва налаштуванням програмного продукту. Розроблено абстрактну модель вимог на основі онтології, яка представляє знання про особливості продукту, а також їх бізнес-логіку. Крім того, на основі моделі знань розроблено систему діалогу на основі фрейму. Вона допомагає виявити вимоги користувачів, а потім виводить сервісно-орієнтований опис системи для реалізації програм-кандидатів. Пропонований підхід розроблений спеціально для налаштування додатків SOA.

1 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА СЕРВІСІВ В ПРЕДМЕТНІЙ ОБЛАСТІ

1.1 Лінія програмного забезпечення

Ключова ідея інженерії SPL полягає в зборі, аналізі та повторному використанні артефактів програмної інженерії тісно пов'язаних програмних систем. Ці багаторазові артефакти надають варіанти розробки на кожному етапі розробки програмного забезпечення. Отже, діяльність з розробки програмного забезпечення буде в основному зосереджена на налаштуванні системи, а не на створенні.

Для розробки програми з фреймворком SPL існують два процеси: розробка домену та розробка додатків [1]. Під час процесу розробки домену визначаються спільні риси та різновиди артефактів, які можна використовувати повторно. Між артефактами різних етапів розробки програмного забезпечення встановлюються вертикальні зв’язки, які можна скласти. На стадії розробки додатків розробляються додатки. Ідентифіковано зміни, які прив’язуються до заявки-кандидата. Потім на основі цих загальних і змінних артефактів здійснюється розробка програми.

У SPL продукти процесу розробки домену повинні використовуватися повторно. Таким чином, ціна розробки нового додатка в основному стягується на етапі розробки програми. Завдяки налаштуванню, а не створенню, розробка додатків є економічно ефективною в порівнянні з традиційними підходами до розробки програмного забезпечення. Тому, поки процес розробки домену піддається контролю, SPL може ефективно зменшити витрати, час і людські зусилля в розробці програмного забезпечення. Тим не менш, Рабізер та ін. [6] зазначають, що порівняно з зусиллями, які витрачаються на розробку та моделювання ліній програмних продуктів, доступна незначна підтримка для підвищення їхнього використання на практиці. Без ефективних підходів до використання ліній продукції, зокрема автоматизованих, SPL не міг би бути

широко прийнятим у промисловості. Іншими словами, вони матимуть більшу академічну цінність, ніж практичну Рабізер та ін. далі визначити 6 вимог для полегшення застосування ліній продукції на практиці:

- Автоматизоване та інтерактивне дозвіл змінності
- Адаптивність і розширюваність
- Підтримка управління вимогами програми
- Гнучкі та специфічні для користувача візуалізації мінливості
- Інструкція для кінцевого користувача
- Підтримка управління проектами

Ідея інтерактивного виявлення вимог надихається цими пропозиціями. За допомогою автоматизованого та інтерактивного рішення для виявлення вимог мінливістю вимог додатків можна автоматично керувати та вилучати їх в інтерактивному режимі. Побічний продукт бази знань вимог додатково дає змогу адаптивно та широко застосовувати лінійки продуктів. Отже, підхід до підвищення практичної цінності. Рекомендується SPL.

1.2 Сервісно-орієнтована архітектура

Оскільки SOA охоплює багато передових практик з попереднього досвіду розробки програмного забезпечення та робить бізнес-системи більш гнучкими та придатними для повторного використання, за останнє десятиліття вона набула все більшої популярності в промисловості, а також у наукових спільнотах. Технічно кажучи, SOA являє собою модель, в якій програмні системи розкладаються на слабко пов’язані блоки функціональних можливостей (тобто послуги), при цьому кожен з цих блоків повинен бути автономним, багаторазовим, відкритим і здатним спілкуватися з іншими блоками за допомогою автономних повідомлень [2]. Таким чином, підрозділи можна розподіляти та співпрацювати за допомогою обміну повідомленнями.

Як правило, в розробці SOA беруть участь три ролі: постачальник, брокер і запитувач. Постачальник послуг розробляє та публікує послуги шляхом реєстрації описів послуг, а також відповідної інформації про доступ у депозитарії брокера послуг. Потім запитувач послуг намагається знайти послуги, звернувшись до брокера послуг. Брокер послуг узгоджує вимоги запитувача з послугами, зареєстрованими в депозитарії, і повертає запитувачу відповідну інформацію про доступ до послуг. Пізніше запитувач відвідує та отримує послуги постачальника відповідно до інформації про доступ. Таким чином, щоб отримати доступ до певних послуг, запитувач повинен спочатку знайти послуги з реєстру брокера, а потім зв'язатися з постачальником, щоб викликати та створити, якщо необхідно, послуги.

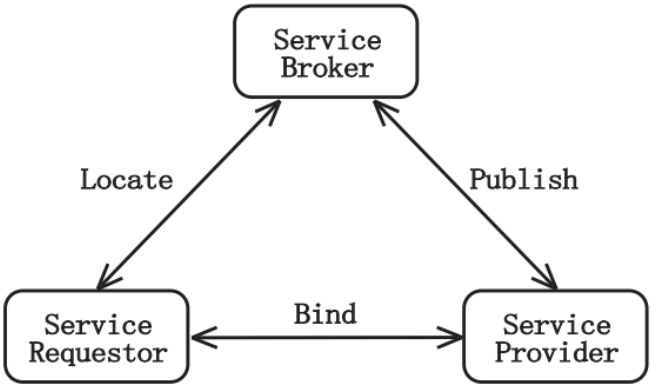


Рисунок 1.1 - Модель SOA

Виявлення сервісів – це процес пошуку відповідних послуг із реєстру брокерів. Традиційно механізм UDDI застосовується у службах публікації, зіставлення та пошуку. Однак він визначає лише набір синтаксичних критеріїв пошуку. Відповідність на семантичному рівні не підтримується, що призводить до небажаних зворотних зв'язків. Натхненна розвитком Semantic Web, з'являється ідея Semantic Web Service. Підходи до семантичних веб-сервісів пропонують семантику веб-сервісам. Отже, вони самоописувані та машинообробні, і відкриття цих сервісів є більш перспективним представляє

рішення семантичного веб-сервісу, а також список його застосувань для вирішення проблеми автоматичного виявлення сервісів на основі пошуку партнерів.

Більше того, у цій дипломній роботі прийнято метод, який є інверсією складу сервісу, для інженерії домену SPL. Іншими словами, функціональні можливості систем SOA аналізуються шляхом декомпозиції. Потім результати декомпозиції далі використовуються для створення екземпляра запропонованої моделі вимог на основі онтології.

Склад служби дуже важливий в інженерії SOA. Реалізація складного веб-сервісу часто передбачає залучення інших служб. Однак послуги розповсюджуються. Їхня співпраця покладається на обмін автономними повідомленнями. Тому синтаксис, а також семантика повідомлень, зокрема порядок їх обміну, повинні бути чітко визначені [9]. Обмін повідомленнями в SOA називається композицією служби. Композиція сервісу являє собою процес об’єднання функціональних можливостей певних сервісів для реалізації функціональних можливостей складеного сервісу. Це може бути виконано шляхом створення примітивних або складених сервісів . У цій дипломній роботі складені послуги розкладаються на менш складні послуги, щоб отримати знання, пов’язані з функціональністю систем та бізнес-логікою.

Крім того, нині люди також працюють над автоматизацією процесу складання послуг. Методології планування Semantic Web Service плюс AI пропонують підходи до вирішення цієї проблеми. Незважаючи на відсутність комплексного рішення, було досягнуто успішного покращення .

Автоматичне виявлення та складання сервісів є вирішальними для реалізації автоматизованого SPL. Очікується, що інтерактивний метод виявлення вимог, запропонований у цій роботі, автоматизує процес розробки вимог, але автоматизовані методи SOA є найкращими рішеннями для автоматизації впровадження системи.

1.3 Інтерактивна розробка вимог

У звичайній розробці програмного забезпечення комп'ютери розглядаються просто як безособові машини, що забезпечують функції, об'єкти або моделі, в той час як їх особистість і характеристики нехтують. У [11] Кнаус стверджує: «В очах розробника програмного забезпечення комп'ютер поводиться більше як людина з надзвичайними навичками та слухняністю. Далі він стверджує, що інтерактивна парадигма розробки програмного забезпечення, яка перерозподіляє обов'язки комп'ютерів, може подолати довгострокові проблеми розробки та обслуговування програмного забезпечення, що вкорінені в звичайних парадигмах програмування. Крім того, визначивши метафору для комп'ютера, побудувавши концептуальну модель як парадигму програмування та спроектувавши відповідний інтерфейс користувача, можна знайти таку інтерактивну парадигму.

Хоча Кнаус обіцяє світле майбутнє, прогресу мало. Завдання програмної інженерії дуже складне. Перевизначити обов'язки комп'ютерів дуже складно. Машини не можуть впоратися зі складністю певного програмного компонента. Тим часом для побудови концептуальної моделі потрібно багато зусиль як від розробки програмного забезпечення, так і від взаємодії людини та комп'ютера. Таким чином, як побудувати інтерактивну парадигму програмної інженерії, залишається питанням. На щастя, в недавніх дослідженнях з'явилися деякі надихаючі ідеї. SOA інкапсулює функціональні можливості програмного забезпечення у слабо пов'язані послуги, що допомагає інженерам з програмного забезпечення машин позбутися від складності нижнього рівня та спростити їх роботу. З іншого боку, у парадигмі SPL їх обов'язки додатково визначаються як керування змінними артефактами програмної інженерії. Таким чином, в інтерактивній інженерії програмного забезпечення машини можуть відігравати роль у спрямуванні користувачів до вибору програмних активів для

багаторазового використання та реалізації програми-кандидата шляхом створення замовлених послуг.

Оскільки тепер доступна відносно конкретна специфікація відповідальності машин в інтерактивній інженерії програмного забезпечення, у цій роботі додатково пропонується підхід до визначення вимог для інженерії SPL на основі SOA як моделі програмування для реалізації інтерактивної інженерії вимог. Як інтерфейс взаємодії використовується система діалогу на основі фрейму. Роботи, пов'язані з діалоговими системами, будуть представлені в наступному розділі.

1.4 Система діалогу

Системи діалогу — це різновид комп'ютерних систем, призначених для спілкування з людьми, вилучення й аналізу інформації з їхніх висловлювань на основі діалогу, щоб виконувати певні завдання (наприклад, обмін інформацією та надання послуг) відносно природними способами. Мова є найефективнішим способом обміну інформацією між людьми. Більшість людських комунікацій в історії засновані на діалогах. Таким чином система діалогу забезпечує більш природний, комфортний і зручний спосіб взаємодії людини і машини.

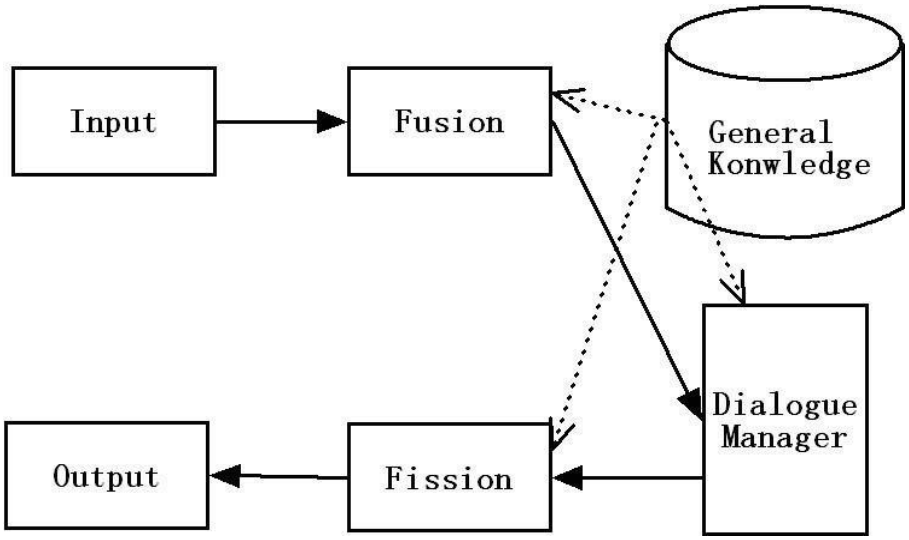


Рисунок 1.2 - Структура діалогової системи

Зазвичай система діалогів складається з шести компонентів (Рисунок 1.2): введення, злиття,

Менеджер діалогів, база знань, поділ та вихід.

Вхідні дані:

Обробляйте різні режими введення.

Fusion:

Витягувати, розпізнавати, інтерпретувати та об’єднувати інформацію з різних режимів введення.

Менеджер діалогів:

Контролюйте потік діалогу, вирішуючи, як система повинна реагувати на вхідні дані.

База знань:

Керуйте такою інформацією, як історія діалогів, знання завдань, загальні знання діалогів, знання предметної області та інформація про користувача.

поділ:

Трансформуйте відповіді на різні режими вихідних даних.

Вихід:

Обробляйте виходи.

Основними компонентами діалогової системи є менеджер діалогів і база знань. Менеджерів діалогу можна розділити на чотири категорії [7]:

Скінченний стан і на основі кадрів:

Менеджери діалогів на основі кінцевих станів є найпростішими менеджерами діалогів. Структура діалогу представлена у вигляді мережі переходу станів, а управління діалогом керується системою, і всі висловлювання системи наперед визначені [7]. Як розширення менеджерів діалогів на основі кінцевих станів, моделі на основі фреймів імітують підхід до заповнення форми, що забезпечує певний ступінь гнучкості. У цій дипломній роботі розроблена система діалогу на основі фрейму для інтерактивного проведення виявлення вимог.

Інформаційний стан та імовірнісний аналіз:

Для підходів на основі інформаційного стану попередньо визначена група станів, і стан діалогу буде динамічно змінюватися відповідно до певних стратегій взаємодії. Деякі ймовірнісні методи (наприклад, частково спостережуваний процес прийняття Маркова) застосовуються для управління стратегіями переходу.

На основі плану:

Менеджери діалогу на основі плану базуються на планових теоріях комунікативної дії та діалогу [2]. Вони складніші, ніж попередні менеджери діалогу.

На основі спільного агента:

Підходи, засновані на співпраці агентів, намагаються охопити мотивацію діалогу та механізми самого діалогу. В результаті менеджери, засновані на цих підходах, вносять свій внесок у найскладніші системи діалогу, які дають дуже високий рівень гнучкості.

Для джерела знань у системі діалогу зазвичай існує п’ять різних моделей знання :

Модель діалогу:

Модель діалогу містить загальну інформацію про побудову діалогу, яка використовується для керування діалогом. Граматичне моделювання та моделювання на основі плану є двома основними підходами до моделювання знань для моделі діалогу.

Історія діалогу:

Історія діалогу записує історію діалогу. Використовується для керування діалогом, роз’яснення залежних від контексту висловлювань та контексту

Модель домену:

Модель домену містить знання предметної області, які будуть посилатися в розмовах. Знання в моделі предметної області в основному використовуються для спрямування семантичної інтерпретації висловлювань користувача, пошуку

відповідних елементів і відносин, які обговорюються, надання відповідей за замовчуванням. Модель домену зазвичай містить структуру предметної області і містить підмножину загальних знань про світ. Його спрощенням є концептуальна модель, яка представляє концептуальні відносини між об'єктами в предметній області . Часто однієї лише концептуальної моделі достатньо для знання предметної області діалогової системи.

Модель завдання:

Модель завдання, яка часто складається з ієрархічної структури представлення, описує, як повинні виконуватися комунікативні та інші завдання системи.

Модель користувача:

Модель користувача представляє мету та плани, можливості, установки, знання та переконання користувача.

У цій роботі база знань пропонованої діалогової системи містить предметні знання щодо виявлення вимог і знання завдань, які допоможуть користувачам налаштувати певний тип програмного забезпечення.

Крім структури, враховуючи джерело інформації, що визначає взаємодію, завдання діалогових систем можна розділити на чотири категорії:

Завдання на заповнення слотів:

Користувач має свою мету і має інформацію про виконання завдання.

Завдання пошуку в базі даних:

Користувач має свою мету, але йому потрібно отримати інформацію для виконання завдання.

Завдання на пояснення:

Користувач не має або має мало знань про завдання.

Більш складні завдання:

Завдання є комбінацією трьох інших завдань.

У запропонованому методі прийнято діалогову систему на основі слотів, а виявлення вимог моделюється як група завдань на заповнення слотів. Ці

завдання виконуватимуться відповідно до знань, пов’язаних з інженерією вимог, яка вбудована в базу знань діалогової системи. Робота, пов’язана з розробкою вимог на основі знань, буде представлена в наступному розділі.

1.5 Розробка вимог на основі знань

Розробка вимог визнана найбільш критичною стадією всього процесу розробки програмного забезпечення . Як правило, понад 40% помилок у проекті програмного забезпечення є результатом розробки вимог, тоді як на їх виправлення потрібно в 10 разів більше витрат, ніж інші помилки .

Звичайні методи розробки вимог на основі процесів або сценаріїв попередньо визначають групу процесів та відповідні їм рекомендації. Відповідно, інженерно-технічні заходи та результати виконання вимог здійснюються відповідно до інструкцій . Однак дуже часто, коли процеси тривають, деяка важлива інформація ще недоступна. Тому інженерам доводиться повторювати процеси, що призводить до затримки проекту та додаткових витрат.

На відміну від традиційної розробки вимог, орієнтованої на процеси, розробка вимог на основі знань, як нова парадигма розробки вимог, проводиться під керівництвом предметних знань. Отже, інформація, прихована в домені, може можна отримати без особливої допомоги експертів домену. Далі інформація використовується для керівництва традиційним процесом розробки вимог. В результаті зберігається валідність, повнота, а також узгодженість продукту розробки вимог. Більше того, зміни в проекті розробки програмного забезпечення будуть виявлені та передбачені на ранній стадії, і буде зроблено менше зусиль. Нарешті, очікується, що результат проекту буде більш зрілим і повним, тоді як переробка може бути різко скорочена.

Крім того, серед групи методів розробки вимог, орієнтованих на знання, дуже популярна інженерія вимог на основі онтології. Це:

- Забезпечує офіційне представництво як документів вимог, так і знань.

Описує проблемну область з різним ступенем формалізації та виразності.

Добре підходить як еволюційний підхід.

Використовується для підтримки управління вимогами та покращення відстеження артефактів вимог.

Таким чином, він перевершує інші традиційні підходи, засновані на знаннях. На даний момент було запропоновано ряд підходів до розробки вимог на основі онтології.

Детальний вступ буде представлено в наступному розділі.

1.6 Онтологія

У теорії, онтологія є формальною, явною специфікацією спільної концептуалізації. Іншими словами, онтологія використовується для представлення загального знання в межах домену.

Причини розробки онтології можна грубо розділити на п’ять категорій

Поділитися спільним розумінням структури інформації між людьми або програмними агентами

Щоб увімкнути повторне використання знань домену

Щоб зробити припущення щодо домену явними

Відокремити предметні знання від операційних. Аналізувати знання предметної області

Усі вони тісно пов’язані з представленням знань предметної області. Як правило, онтологія надає спільний словник, який можна використовувати для моделювання домену або завдання. Тут «моделювання» означає побудову понять, об’єктів, а також їхніх властивостей і відношень, які існують у предметній області чи у розв’язанні задачі .

Зазвичай методи розробки знань, такі як логіка пропозицій, логіка предикатів та інші методи, засновані на правилах, в основному досліджують такі теми, як логіка, представлення знань, пошук тощо . Вони зосереджуються на

тому, як вирішити проблему, а не на самих знаннях. Тому отримані знання часто є неявними, і їх важко підтримувати, поширювати або використовувати повторно. Навпаки, головною турботою онтології є зміст знань і підходи до його накопичення. Це створює основу для загального знання.

Більше того, грубо кажучи, онтологія складається з онтології задач, яка характеризує обчислювальну архітектуру системи, заснованої на знаннях для певних завдань, та онтології предметної області, яка характеризує знання конкретної області завдання.

Розробка онтології, як правило, включає наступні кроки:

1. Визначте класи (поняття в предметній області) в онтології

2. Розташуйте класи в таксономічній (підклас-суперклас) ієрархії

3. Визначте слоти (властивості класів і екземплярів) та опишіть дозволені значення для цих слотів

4. Заповніть екземпляри

5. Заповніть значення для слотів екземплярів

Якщо всі класи, слоти, екземпляри та їх відносини визначені належним чином, створюється онтологія для знання завдання або домену.

У застосуванні онтології є багато важливих переваг. Перш за все, знання є формальними, явними та спільними, а це означає, що знання доступні кожному. За допомогою онтології загальні стандарти домену можуть встановити експерти. Люди з різним досвідом матимуть можливість здобувати знання без особливої професійної підготовки. Тим часом представлення на основі таксономії дуже стисле і зрозуміле, що зменшує неоднозначність і помилки. Нарешті, онтології є машинно-орієнтованими. Деякі з мов онтологій засновані на XML, які можна легко використовувати між різними машинами. Таким чином, на даний момент онтологія є одним із найпопулярніших і потужних методів розробки знань, широко застосовуваних у різних додатках.

У цій роботі прийнято мову веб-онтологій (OWL) , одну з найуспішніших мов онтологій, рекомендованих W3C. OWL використовує синтаксис XML і

частково зіставляється з логікою опису, яка є підмножиною логіки предикатів. Таким чином, OWL надає користувачам різноманітні можливості висновку. Насправді, реалізація деяких аргументів OWL базується на алгоритмах таблиці, що є алгоритмом для міркування логіки опису. OWL складається з трьох підмов: OWL Lite, OWL DL і OWL Full. OWL Full є найвиразнішим серед трьох. Але немає жодних аргументів, які підтверджують його висновок. Навпаки, обіцяючи розв'язність, виразність OWL DL та OWL Lite приноситься в жертву . Таким чином, була запропонована мова правил семантичної мережі (SWRL) [26], яка доповнює OWL DL і OWL Lite правилами. DL-безпечну версію SWRL також можна вирішити.

Більше того, як пояснювалося вище, онтологія має механізм для опису неявного знання. Насправді методи для отримання неявного знання засновані на онтологічних міркуваннях. Деякі з цих підходів є похідними від логіки опису. Наприклад, OWL DL заснований на логіці опису SHIQ. Таким чином алгоритми опису

Логічні міркування, такі як алгоритми таблиць, можна використовувати для висновку з онтологією OWL DL.

Крім того, для міркування OWL DL доступно багато стабільних аргументів. Наприклад, Protégé, платформа для розробки онтологій OWL, надає інтерфейси для підключення аргументів, таких як Pellet, FaCT++, Jena і RACER. У цій дипломній роботі Pellet використовується для онтологічних міркувань. Він підтримує міркування як з OWL DL, так і з DL-безпечною версією

SWRL.

Онтологія для розробки вимог

Як обговорювалося, визначення вимог на основі онтологій є популярною темою сьогодні. Однак існує довга історія застосування онтології для розробки вимог. Перші дослідницькі зусилля, присвячені використанню онтологій у розробці вимог, можна датувати початком 1980-х років . Відтоді було вивчено, розроблено та запропоновано низку підходів до розробки вимог на основі

онтології. Серед найвідоміших публікацій представлена модель вимог, заснована на онтології, яка полегшує виявлення неповноти та невідповідності артефактів вимог, вимірювання якості розробки вимог та прогнозування потенційних змін на наступних етапах розробки програмного забезпечення. У моделі визначено дуже повну групу онтологічних зв'язків, пов'язаних з інженерією вимог. Представлена мінімальна модель для опису знання вимог. Ціль, обмеження якості та програмна ціль пропонуються як основні концепції онтології в розробці вимог. Крім того, введено структуру для визначення вимог на основі онтологій. Типи функціональних вимог, а також їх взаємозв’язки, які сприяють виникненню вимог, окреслені в моделі онтології.

Тим часом представляє добре структуровану модель вимог на основі онтології під назвою SoftWiki, яка здатна фіксувати і керувати артефактами розробки вимог для всіх етапів розробки системи.

Хоча всі наведені вище підходи роблять великий внесок у розробку вимог на основі онтології, вони не підходять для представлення знань про вимоги для автоматизованого SPL акцентує увагу на перевірці артефактів, тоді як модель, запропонована, скоріше теоретична, ніж практична. Крім того, мета методу полягає в тому, щоб полегшити комунікацію між інженерами вимог і клієнтами під час виявлення вимог. Аналогічно, SoftWiki розроблено для підтримки співпраці всіх зацікавлених сторін на всіх етапах розробки програмного забезпечення.

Насправді, внески з більшості базованих на онтологіях інженерних досліджень поділяються на такі три категорії:

Підвищення якості артефактів розробки вимог .

Визначення спільного розуміння між інженерами та клієнтами.

Розробка нових методів розробки вимог, заснованих на знаннях.

Питання, які критично важливі для реалізації автоматизованого SPL на основі SOA, як-от надання автоматичного керівництва для налаштування

продукту та створення специфікації системи, орієнтованої на обслуговування, недостатньо охоплені цими підходами.

Онтологія для SOA

Як зазначалося в розділі II, ідея семантичного веб-сервісу пропонується для автоматизації діяльності із впровадження системи SOA. На відміну від Feature Driven Development і Model-Driven Achitecture , де функціональні можливості системи зіставляються з функціями системи та незалежними від платформи моделями, SOA інкапсулює функціональні можливості програми у слабо пов'язані служби. Таким чином, замість проектування та реалізації функцій чи моделей програмні додатки можуть бути реалізовані шляхом виявлення, створення та виклику служб у SOA. Більше того, методи семантичного веб-сервісу додатково специфікують описи веб-сервісів на семантичному рівні, таким чином пропонують рішення для автоматичного виявлення та компонування сервісів.

Підходи до семантичних веб-сервісів також засновані на онтології. Наразі для семантичної веб-сервісу розроблено в основному три онтології: моделювання веб-сервісів

Онтологія (WSMO), семантична розмітка для веб-сервісів (OWL-S) та онтологія SOA. ▪ WSMO - це концептуальна модель, пов'язана з семантичним веб-сервісом. Він підтримує розгортання та взаємодію семантичної веб-служби.

OWL-S також є онтологією для опису семантичної веб-служби. Він дозволяє автоматично виявляти, викликати, створювати та відстежувати веб-сервіси за певних обмежень.

"Онтологія SOA визначає поняття, термінологію та семантику SOA як у діловому, так і в технічному плані». Це створює основу для полегшення

Розуміння SOA, зв'язок, пов'язаний з SOA, і моделювання системи SOA.

Тим часом, це потенційно сприяє реалізації керованої моделі SOA .

Перші дві онтології є відносно низькорівневими. Це методи для опису конкретних семантичних веб-сервісів.

У цій дипломній роботі використовується OWL-S. OWL-S заснований на мові онтологій OWL. Це онтологія сервісів, яка робить можливим автоматичне виявлення та компонування сервісів. Екземпляри його класу ServiceProfile описують характеристики служб, які використовуються для узгодження запитів клієнтів, тоді як інформація про склад служби міститься в екземплярах класу ServiceModel. Під час виявлення сервісів сервісні профілі запитувачів автоматично узгоджуються з профілями послуг постачальника послуг за допомогою відповідності семантичних можливостей . Якщо узгодження вдалось, потрібні служби будуть знайдені. Тоді ServiceModels, пов’язані з виявленими службами, будуть нести інформацію про процес створення та виклику служб. Таким чином, обґрунтовуючи знання, що містяться в ServiceModels, буде виконана автоматична композиція сервісу і можна отримати бажані функціональні можливості . З іншого боку, хоча OWL-S надає описи функціональних можливостей веб-сервісів, у нього мало механізмів для опису нефункціональних сервісів. Пропонує розширення якості для OWL-S, щоб компенсувати цей недолік. Він успадковується від класу ServiceParameter, визначеного в OWL-S, який призначений для розширення OWL-S більш конкретними описами служб.

2 АЛГОРИТМІЧНІ ТА МАТЕМАТИЧНІ РІШЕННЯ ДЛЯ ЗАДАЧ ДАНОГО КЛАСУ

2.1 Вступ

У цьому розділі розповідається про модель вимог на основі онтології. На відміну від методів розробки вимог на основі онтології, які розглядаються в розділі III, модель, розроблена в цьому дослідженні, інтегрує знання інженерії вимог із знаннями, орієнтованими на обслуговування. У той час як ключові концепції та відносини, запропоновані в [7] і [8], зберігаються для підтримки повноти та узгодженості вимог до продукту, для створення екземплярів моделювання, а також для організації спільних рис і відмінностей використовується підхід декомпозиції, орієнтований на послуги. SPL. Крім того, інформація для визначення вимог, така як ранги вимог, також виражається в моделі онтології. Отже, очікується, що знання, що містяться в моделі, допоможуть автоматично налаштувати продукт і сприяти створенню сервісно-орієнтованої системної специфікації для впровадження системи.

У цьому розділі побудова моделі вимог на основі онтології представлена відповідно до кроків розробки онтології, запропонованих у [2]. Спочатку визначаються поняття моделі. По-друге, окреслюються відносини, що описують таксономічну ієрархію онтології. По-третє, як доповнення до онтології OWL, цей розділ пропонує групу правил SWRL для моделі. Нарешті, обговорюється створення екземплярів онтології.

2.2 Поняття

Поняття моделі проілюстровано на рисунку 2.1 з позначеннями діаграми класів. У сфері виявлення вимог, відповідно до їх різних ролей, вимоги можна в основному розділити на три категорії: функція, якість і програмна ціль. Між тим, Rank використовується для представлення важливості вимог щодо всього

програмного забезпечення продукту. Крім того, запропонована модель онтології також містить поняття для опису детальної інформації про вимогу, що допомагає клієнтам оцінити.

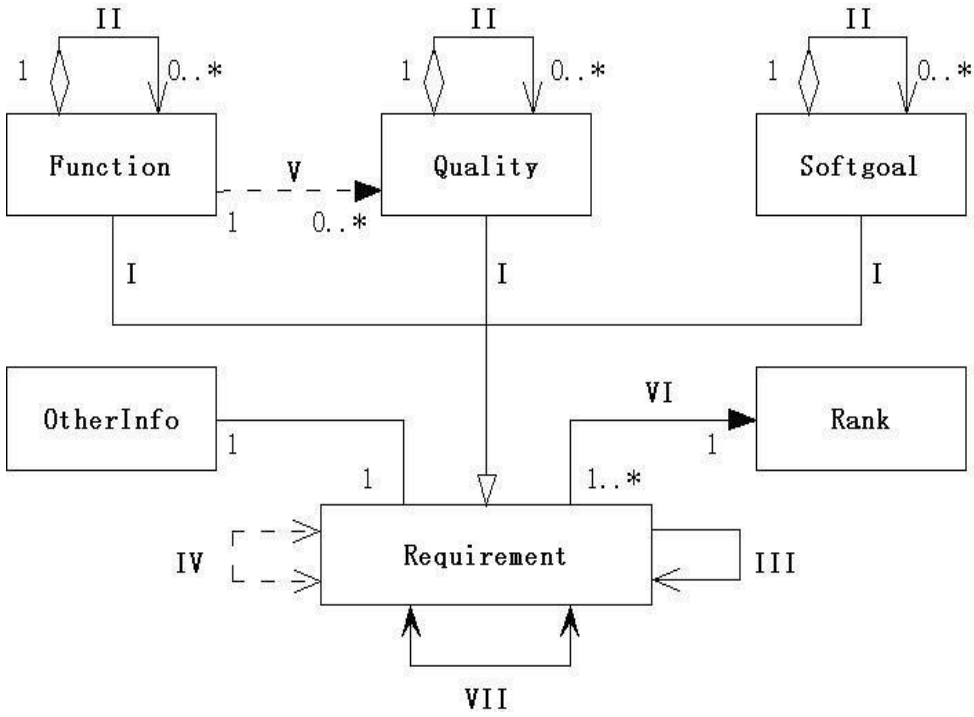


Рисунок 2.1 - Запропонована модель вимог на основі онтології

Вимога: екземпляр Requirement — це системна функція, яку користувачі можуть замовити. Існує три типи вимог: функція, якість і Softgoal.

Функція: екземпляр Function представляє функціональність, яку користувачі можуть замовити. Це може бути примітивна функція, яку пропонує програмне забезпечення продукту, або композиція з кількох примітивних функцій. З точки зору, орієнтованої на обслуговування, функція насправді є послугою. У цьому дослідженні функції організовані в структуру, схожу на ліс, де складені функції розкладаються на менш складні складені функції або примітивні функції.

Якість: екземпляр якості — це нефункціональне обмеження, накладене на функцію. В основному вони використовуються для подальшого визначення функціональності. Тому екземпляр якості завжди пов’язаний з певною

функцією. Якість не можна вибрати, якщо відповідну функцію відкидається. Крім того, якість також може бути додатково визначена за допомогою підякостей у дереві декомпозиції.

Softgoal: екземпляри Softgoal також є нефункціональними обмеженнями. Однак замість того, щоб нав’язувати певну функцію, вони описують глобальне середовище, в якому працює програмна система продукту. Вони часто дуже абстрактні і можуть бути пов’язані з підмножиною функцій і якостей кандидата, але всі вибрані вимоги повинні відповідати їм. Подібно до прикладів якості, програмні цілі також можна розкласти на підпрограмні цілі.

Ранг: екземпляри рангу представляють важливість вимог щодо потенційної системи. З іншого боку, вони також визначають порядок оцінки вимог у процесі виявлення вимог.

OtherInfo: Концепції, узагальнені з OtherInfo, можуть бути загальними концепціями, пов’язаними з розробкою вимог (наприклад, зацікавлені сторони) або конкретними концепціями в межах домену. Їх екземпляри використовуються для опису детальної інформації про вимоги. Користувачі можуть запитувати таку відповідну інформацію під час оцінки вимог.

2.3 Відносини

Зв'язки між поняттями зображено на рисунку 2.1, а їх позначення — на рисунку 2.2. У запропонованому методі ці взаємозв’язки не тільки дозволяють перевірити узгодженість і повноту налаштованих вимог, але й полегшують машинам керувати виявленням вимог.

I. Узагальнити:

Примірник Function, Quality або Softgoal також є екземпляром

Вимога. Узагальнення представляє відношення IS-A.

II. Розкласти:

Вимога x розкладається на менш складну вимогу у того ж типу. y описує частину характеристик x або є більш детальною альтернативою x. x називається батьківським, а y — дочірнім x.

$$REQ = FUN \cup QUA \cup SOF$$

I. $generalize(REQ, x),\ x \in \{FUN, QUA, SOF\}$

II. $decompose(x, y),\ x \in R,\ y \in R,\ R \in \{FUN, QUA, SOF\}$

III. $rely(x, y),\ x \in REQ,\ y \in REQ$

IV. $contradict(x, y),\ x \in REQ,\ y \in REQ$

V. $associate(x, y),\ x \in FUN,\ y \in QUA$

VI. $hasRank(x, r),\ x \in REQ,\ y \in RANK$

VII. $invalid(x, y),\ x \in REQ,\ y \in REQ$

Рисунок 2.2 - Позначення зв'язків

Вимога може розкластися на нуль дочірніх; інакше він повинен розкластися принаймні на двох нащадків. У дитини не більше одного батька. Логічно, вимога не може розкластися на більш складні вимоги (наприклад, її батьківську чи батьківську батьківську). Таким чином відношення decompose утворює дерева декомпозиції. На практиці можливо, що вимога бере участь у декомпозиціях кількох більш складних вимог. Однак, якщо вимогі дозволено мати двох батьків, коли вона буде підхоплена під час виявлення вимоги, склад, який вимога має приєднатися до програмного забезпечення продукту, буде невідомим. Отже, у цьому випадку для участі в двох декомпозиціях необхідні дві копії вимоги.

Коли функції розкладаються на підфункції, батьківські функції представляють функціональні можливості, які є результатами їхньої дочірньої композиції (тобто композиції служби). Іншими словами, батько має на увазі стратегію композиції, а не будь-які конкретні функції. Лише листки в дереві розкладання функцій є примітивними функціональними можливостями. Крім того, не обов’язково вибирати стратегію композиції, якщо лише потрібні примітивні функції.

У дереві розкладання якості або м’якої цілі діти позначають

уточнення для батьків. Таким чином, логічно, фактично не має ніякої різниці, якщо батьківська якість безпосередньо замінена його дочірніми в екземплярі моделі онтології. Відношення декомпозиції лише полегшує взаємодію виявлення вимог або роботу зі створення екземплярів онтології.

Реалізація вимоги x залежить від реалізації вимоги y. Якщо x замовлено клієнтами, потрібно також вибрати y; інакше отримана система не працюватиме належним чином.

Таким чином, Rely описує повноту продукту виявлення вимог. Крім того, коли вимога спирається на дві інші вимоги, це означає, що їй потрібні обидві. На практиці можливо, що вимога вимагає лише одного з них. У цьому випадку створюється два екземпляри однієї і тієї ж вимоги і між ними встановлюються відносини Contradict; тоді кожна з двох копій покладається на одну з двох необхідних вимог.

Батьківська функція, що спирається на своїх нащадків або нащадків своїх нащадків, означає, що стратегія композиції вимагає залучення відповідних дочірніх. Якщо дочірня функція покладається на свого батька або батьківську функцію, це означає, що дочірня функція спеціально розроблена для композиції. Зазвичай функція, яка спирається на іншу функцію, означає, що вхід однієї функції є виходом іншої функції.

Коли батьківська якість або програмна ціль покладається на своїх дочірніх, це означає, що діти є важливими для батьківського обмеження. У цьому дослідженні дитячі якості та м’які цілі мають покладатися на їхніх батьків. Це обіцяє, що під час процесу визначення вимог, якості нащадків та м’які цілі не будуть досліджуватися, якщо їх батьків покинуть.

Якості та м’які цілі можуть покладатися на функції. Це говорить про реалізацію

обмеження вимагає реалізації деяких функцій. Для належної роботи функціям також можуть знадобитися певні обмеження якості або softgoal. Більше

того, якщо функція покладається на якість, вона також покладається на функцію, пов’язану з цим обмеженням якості.

Вимога x суперечить вимозі y. Вимога x і вимога y не повинні одночасно реалізовуватися в програмному забезпеченні продукту. Contradict описує узгодженість викликаних вимог. Цей зв’язок симетричний і не відбиває. Вимога не може суперечити своїм дочірнім, батьківським або вимогам, на які вона спирається. Функція не повинна суперечити пов’язаним з нею обмеженням якості.

Зазвичай, якщо дві вимоги відіграють однакову роль у програмі-кандидаті, що означає, що вони представляють ту саму функціональність або обмеження, між ними існує відношення Contradict. Крім того, якщо дві вимоги не можуть бути виконані одночасно в програмному забезпеченні продукту, вони суперечать одна одній.

Функція x пов'язана з обмеженням якості— це обмеження якості, яке можна накласти на функцію x.

Оскільки якість не може бути реалізована в налаштованому програмному забезпеченні, якщо не реалізована пов’язана з ним функція. Асоційовані відносини також припускають, що обмеження якості покладається на відповідну функцію. Більше того, з тією ж проблемою та розв’язком, що й відношення Decompose, дві функції не повинні бути пов’язані з однаковою якістю. Якщо складена функція пов’язана з якістю, це означає, що обмеження якості накладається на композицію, а не на будь-які примітивні функціональні можливості. Обмеження для примітивних функцій повинні бути безпосередньо пов’язані з конкретними функціями. Більше того, оскільки дитячі якості є уточненням для їхніх батьків, якщо функція пов’язана з батьківською якістю, вона також пов’язана з відповідними дитячими якостями. Отже, якщо між функцією та якістю чітко визначено зв’язок асоційованого, функція пов’язана з усім піддеревом декомпозиції якості, що ґрунтується на якості. Між тим,

батьківський елемент кореневої якості, якщо він є, не повинен мати зв’язок асоційованого з жодними функціями.

V. має ранг: Вимога x має ранг r. Вимога може мати рівно один ранг. Під час процесу визначення вимоги вимоги з вищими рангами будуть запропоновані користувачам для оцінки перед тими, хто має нижчі ранги. Отже, якщо вимога має сильний вплив на заявку кандидата чи інші вимоги, їй слід присвоїти високий ранг. Крім того, вимоги батьків завжди повинні мати вищі ранги, ніж їхні діти. Якщо кілька вимог тісно пов’язані між собою і повинні оцінюватися одна за одною, вони повинні мати однаковий ранг.

Крім того, вимоги найвищого рангу розглядаються як істотні вимоги. Вони представляють спільні риси артефактів SPL. Як результат, вони будуть вибрані в обов’язковому порядку перед оцінкою будь-яких інших несуттєвих вимог. Крім того, жодна вимога не повинна суперечити істотним вимогам.

Існує недійсний зв’язок між вимогою x і вимогою y. Недійсні відношення використовуються для позначення недійсності в екземплярі моделі онтології. Він застосовується з правилами і може бути узагальнений на типи більш визначених Недійсних відносин. Види будуть представлені в наступному розділі.

2.4 Правила

Рисунок 2.3 ілюструє групу застосованих у дослідженні правил SWRL. Вони мають рогоподібний вигляд. Розмірковуючи з цими правилами, можна отримати неявні знання для виявлення вимог і створення екземплярів онтології. Нижче наведено пояснення до правил.

ii. Відношення декомпозиції не відображають.

iii Відношення опори є перехідним.

iv. Протилежні відносини симетричні.

v. Відношення Contradict і Rely розрізнені.

vi. Відношення Contradict і Decompose роз’єднані. vii. Конфліктні відносини та відносини асоційованого є роз’єднаними. viii. Відношення декомпозиції є асиметричним. ix Відношення декомпозиції є транзитивним.

x Відношення декомпозиції є зворотним функціональним. xi Асоційований зв'язок є обернено-функціональним.

xii Якість залежить від відповідної функції.

xiv Якщо вимога x спирається на вимогу y, x суперечить вимогам, яким суперечить y.

xv Якщо функція z пов’язана з якістю x, z асоціюється з дочірніми функціями x. xvi Вимоги не можуть суперечити вимогам вищого рангу.

i. $contradict(x,x) \Rightarrow invalid(x,x)$
ii. $decompose(x,x) \Rightarrow invalid(x,x)$
iii. $rely(x,y) \wedge rely(y,z) \Rightarrow rely(x,z)$
iv. $contradict(x,y) \Rightarrow contradict(y,x)$
v. $contradict(x,y) \wedge rely(x,y) \Rightarrow invalid(x,y)$
vi. $contradict(x,y) \wedge decompose(x,y) \Rightarrow invalid(x,y)$
vii. $contradict(x,y) \wedge associate(x,y) \Rightarrow invalid(x,y)$
viii. $decompose(x,y) \wedge decompose(y,x) \Rightarrow invalid(x,y)$
ix. $decompose(x,y) \wedge decompose(y,z) \Rightarrow decompose(x,z)$
x. $decompose(x,y) \wedge decompose(z,y) \Rightarrow invalid(x,y) \wedge invalid(z,y)$
xi. $associate(x,y) \wedge associate(z,y) \Rightarrow invalid(x,y) \wedge invalid(z,y)$
xii. $decompose(x,y) \wedge x \in R \wedge y \in R \wedge R \in \{QUA, SOF\} \Rightarrow rely(y,x)$
xiii. $associate(x,y) \Rightarrow rely(y,x)$
xiv. $rely(x,y) \wedge contradict(y,z) \Rightarrow contradict(x,z)$
xv. $decompose(x,y) \wedge associate(z,x) \Rightarrow associate(z,y)$
xvi. $contradict(x,y) \wedge hasRank(x,r) \wedge r = TopRank \Rightarrow invalid(x,y)$

Рисунок 2.3 - Правила для запропонованої моделі онтології

Фактично, правила i-xi можуть бути виражені елементами OWL. Однак деякі з них неможливо аргументувати наявними аргументами. Навіть якщо аргументи можуть з ними впоратися, аргументи не будуть чітко вказано на недійсні відносини. Тому тут застосовуються правила.

Правила iii, iv, xiii і xiv відображають характер відповідних відносин, тоді як правила xii використовуються для полегшення процесу виявлення вимог.

Крім того, правила i, ii, v, vi, vii, viii, ix, x, xi, xv і xvi використовуються для перевірки дійсності екземплярів моделі вимоги. Під час виявлення вимоги чітко визначені зв'язки «Розкласти» та «Асоціювати» визначають порядок оцінки вимоги. Таким чином, правило xv не активується в процесі виявлення вимог. Більше того, правило ix суперечить правилу x, а також порушує структуру дерева декомпозиції. Але правило ix полегшує виявлення недійсності того, що батьківські обєкти не можуть суперечити діям своїх нащадків. Отже, воно застосовується, але не активується разом із правилом x або в процесі виявлення вимоги.

2.5 Примірник онтології

Створення екземплярів моделі онтології насправді є процесом розробки предметної області SPL інженерія. Перш ніж створити екземпляр моделі онтології для певного типу програмних систем, розробники вимог повинні проаналізувати цей тип сервіс-орієнтованих додатків. Розкладіть ці системи на примітивні послуги та з'ясуйте спільні риси та відмінності. Потім вони можуть створити екземпляр моделі згідно з такою процедурою, представленою на рисунку 2.4.

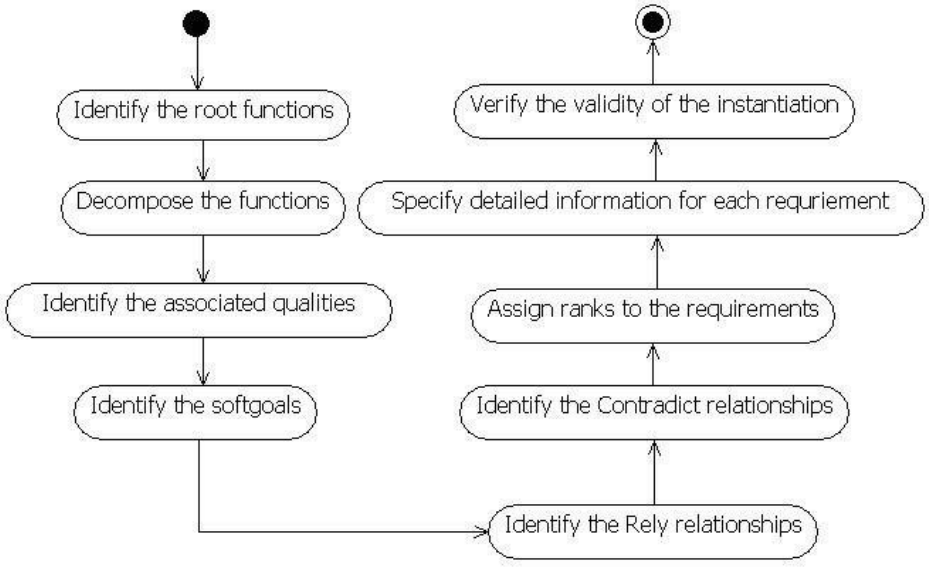


Рисунок 2.4 - Процедура створення екземпляра моделі онтології

1. Визначте основні функції, які є коренями дерев розкладання.

2. Для кожного з коренів, якщо він представляє певні стратегії композиції, визначте дочірніх, які сприяють створенню композиції. Потім установіть зв’язок Decompose між батьком і дітьми. Якщо діти також розкладаються, повторіть цей процес розкладання, поки не будуть виявлені всі примітивні функції.

3. Знайдіть відповідні обмеження якості, які можна накласти на функції. Організуйте якості за допомогою дерев розкладу. Оскільки всі обмеження якості, що містяться в піддереві декомпозиції, пов’язані з однією і тією ж функцією, установіть зв’язок асоційованого між відповідною функцією та коренем піддерева. Передбачається, що асоційовані відносини між дітьми та відповідна функція не повинні бути поясненими. У процесі виявлення вимоги, коли функцію вибирають користувачі, її пов’язані якості будуть негайно оцінені. Проте очікується, що якості нащадків будуть досліджені пізніше.

4. Визначте м’які цілі та за потреби розкладіть їх.

5. Встановіть зв'язок Rely для вимог.

6. Встановіть суперечливі відносини.

7. Визначте суттєві вимоги та призначте вимогам ранги відповідно до їх важливості та очікуваних позицій у процесі виявлення.

8. Укажіть докладні описи для кожної вимоги. Зв’яжіть кожну вимогу з відповідним описом сервісу, який буде використовуватися для створення сервісно-орієнтованого результату. Подробиці будуть розглянуті в розділі V.

9. Перевірте дійсність інстанційованої моделі онтології. За потреби внесіть зміни. Крім того, загальновизнані факти, як-от вимога, яка не може розкластися на одну дитину, також слід перевіряти вручну.

Потім будується дійсний екземпляр моделі онтології. Як правило, для такого типу програмних систем середнього розміру, як онлайн-сервіс покупок книг, будуть десятки вимог і більше сотні зв’язків, створених у екземплярі моделі онтології.

У цій дипломній роботі розроблено систему діалогу на основі фрейму, яка бере інстанційовану модель онтології як базу знань. Він використовується для виявлення запитів користувачів за допомогою взаємодії людини і машини. Хоча підтримувати повноту та узгодженість налаштованих вимог дуже складно і вимагає онтологічних міркувань, взаємодії для виявлення вимог насправді є групою завдань із заповненням слотів. Питання, наприклад, чи потрібна користувачам конкретна вимога, будуть запропоновані машиною, і користувачі відповідатимуть своїми рішеннями щодо цієї вимоги. Таким чином, користувачі знають, що вони збираються робити і як це буде зроблено, а це означає, що процес виявлення вимог можна моделювати як набір підзадач, що заповнюють слоти, тоді як висловлювання, слоти, а також параметри значення для кожного слота буде витягнуто з бази знань, отже, система діалогу на основі фрейму здатна обробляти взаємодії для виявлення вимог, незважаючи на обмежену комунікаційну здатність.

У цьому розділі буде обговорено структуру системи діалогу, процес визначення вимог та результати виявлення.

2.6 Структура запропонованої системи діалогу

Розроблена в даному дослідженні фреймова діалогова система складається з чотирьох компонентів: інтерфейсу, контролера вводу-виводу, менеджера діалогів та бази знань.

Інтерфейс діалогу заснований на тексті. Він відображає згенеровані машиною висловлювання та надає одне місце, яке користувачі можуть заповнити. Як правило, висловлювання будуть такими, як «Ви хочете вибрати вимогу…?». Очікується, що користувачі відповідатимуть «Так» або «Ні». Тоді відповідь користувачів буде передана на контролер вводу-виводу. Він намагатиметься узгодити введені дані з набором попередньо визначеної інформації. Якщо зіставлення не вдається, контролер вводу-виводу генерує

повідомлення, яке просить користувачів виправити свою відповідь, надсилається в інтерфейс і відображатиметься як результат поточної взаємодії. В іншому випадку введені дані будуть перетворені у формат, які можна обробити на машинах, і передані в менеджер діалогів. Тоді менеджер діалогів знає рішення користувачів щодо вимоги, що зараз оцінюється. Він проконсультується з базою знань про онтологію з рішенням і налаштує вимоги на основі відповідних знань вимог, а також вхідних даних. Після цього диспетчер діалогів згенерує висновок відповідно до результату налаштування та відправить на контролер вводу/виводу. Контролер вводу-виводу перетворить вихід на природну мову та відобразить його в інтерфейсі, що ініціює наступний раунд взаємодії.

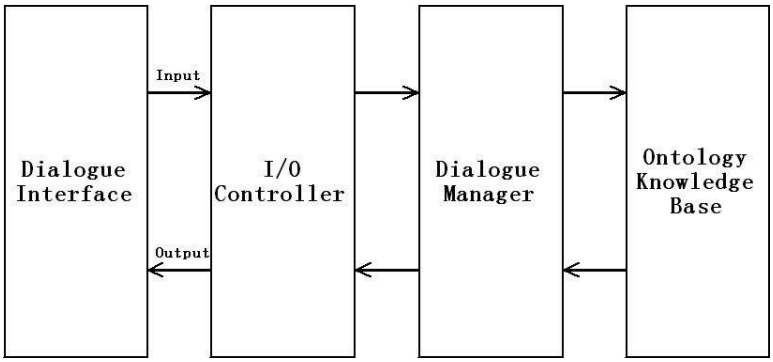


Рисунок 2.5 - Структура запропонованої діалогової системи

2.7 Процес визначення вимоги

Перед початком виявлення вимоги неявні знання (напр.непрямі зв'язки), що містяться в екземплярі моделі онтології, буде витягнуто шляхом міркування.

Процес визначення вимоги поділяється на два етапи. По-перше, визначення вимог буде проводитися під керівництвом машини. Тоді користувачі матимуть можливість змінити свої рішення, прийняті на першому етапі, і надалі налаштувати програмне забезпечення продукту.

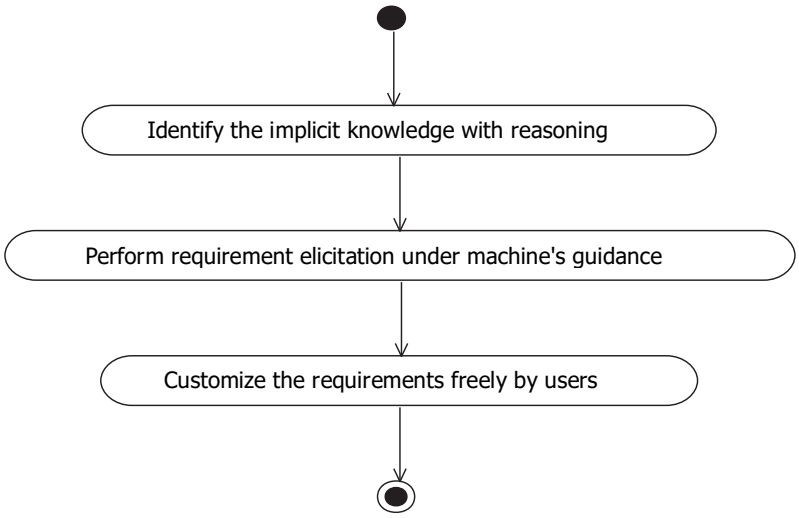


Рисунок 2.6 - Процес інтерактивного виявлення вимог

Машинно-спрямоване визначення вимог.

На першому етапі кожна вимога буде по черзі пропонуватися користувачам для оцінки. На початку всі основні вимоги вибиратимуться автоматично без оцінки. Потім будуть оцінені функції, а потім – оцінка якостей. Нарешті, програмні цілі будуть налаштовані. Серед вимог цієї ж категорії буде оцінюватися один з вищим рангом перед тими, хто має нижчі ранги.

На Рисунок 5.3 представлений псевдокод для оцінки вимог. При оцінці вимоги може бути чотири випадки.

1. Якщо вимога R є важливою для системи, будуть виконані дії для вибору вимоги.

2. Якщо вимога R є несуттєвою та попередньо обраною, будуть виконані дії щодо вибору вимоги. Ці дії включають виклик selectRequirement для вибору R; викликати preSelectRequirement, щоб попередньо вибрано вимоги, на які покладається R; виклик preDropRequirement, щоб мати вимоги, які

R суперечить pre-dropped; викликати preEvaluateRequirement, щоб отримати вимоги, які R розкладає на попередньо оцінені; і якщо R є функцією, викличте preEvaluateRequirement, щоб попередньо оцінити якості, з якими R асоціюється.

```
1.  FOR each requirement R to be evaluated
2.      IF R is essential to the system THEN
3.          CALL PerformRequirementSelecting with R
4.      ELSE IF R is pre-selected THEN
5.          CALL PerformRequirementSelecting with R
6.      ELSE IF R is pre-dropped THEN
7.          CALL PerformRequirementDropping with R
8.      ELSE
9.          CALL evaluateRequirement with R
10.         IF R is to be selected THEN
11.             CALL PerformRequirementSelecting with R
12.         ELSE
13.             CALL PerformRequirementDropping with R
14.         END IF
15.     END IF
16. END FOR

17. PerformRequirementSelection with R
18.         CALL selectRequirement with R
19.         CALL preSelectRequirement with the requirements R relies on
20.         CALL preDropRequirement with the requirements R contradicts
21.         CALL preEvaluateRequirement with the requirements R decomposes into
22.         IF R is a function THEN
23.             CALL preEvaluateRequirement with the qualities R is associated with
24.         END IF

25. PerformRequirementDropping with R
26.         CALL dropRequirement with R
27.         CALL preDropRequirement with the requirements that relies on R
```

Рисунок 2.7 - Псевдокод для процесу оцінки вимог

1. Якщо вимога R є несуттєвою та попередньо відкинута, буде виконана дія для відкидання вимоги. Ці дії включають вимогу до відкидання виклику для відкидання R і вимогу до відкидання виклику, щоб вимоги, які покладаються на R, попередньо відкидалися.

2. Якщо вимога R є несуттєвою і не була попередньо вибрана або не вилучена, буде викликано оцінку вимоги, щоб R оцінили користувачі.

Тоді, якщо користувачі вибирають вибрати R, будуть виконані дії для вибору вимоги. В іншому випадку будуть виконані дії для відкидання вимоги.

Нижче наведено пояснення до підпрограм, які використовуються в псевдокоді.

select Requirement помістить вимогу до набору, щоб він був позначений як «вибрано», якщо він не позначений.

droptRequirement поставить вимогу до набору, щоб він був позначений як "dropped", якщо він не позначений.

Вимога оцінки представить вимогу користувачам через інтерфейс діалогу. Користувачі можуть вибрати або відмовитися від вимоги, або запитати детальний опис вимоги, перш ніж прийняти рішення. Pre Select Requirement поставить вимогу до набору, щоб він був позначений як «попередньо вибраний», якщо він не позначений. Pre Drop Requirement поставить вимогу до набору, щоб він був позначений як «попередньо скинутий», якщо він не позначений pre Evaluate Requirement спочатку викличе evaluateRequirement, щоб оцінити вимогу, якщо вона ще не була оцінена. Потім на основі вибору користувачів буде викликано вимога попереднього вибору або вимога до відкидання. Крім того, якщо вимога має бути попередньо обрана, попередньо виберіть вимоги, на які вона спирається, і попередньо відкиньте вимоги, які їй суперечать. В іншому випадку попередньо відкиньте вимоги, що спираються на це.

Під час процесу визначення вимоги всі вимоги будуть розширені щонайбільше один раз (під час попередньої або офіційної оцінки). Тут розширення вимоги означає отримання детальної інформації про вимогу. Крім того, кожне стосунки Decompose та Associate відвідуватимуть батьківські обєкти не більше одного разу. Кожне відношення Rely та Contradict буде відвідано щонайбільше двічі двома залученими вимогами. Отже, нехай V — кількість екземплярів вимог в онтології, а E — число чотирьох відношень. Тоді складність алгоритму O(V+E).

2.8 Налаштування вимог користувача

На другому етапі користувачі можуть замовити машину вибрати або відкинути довільну вимогу. Іншими словами, вони можуть змінити свої рішення,

прийняті на першому етапі. Якщо вибрану вимогу потрібно відкинути, вибрані вимоги, які на неї покладаються, також будуть вилучені. Якщо потрібно вибрати вилучену вимогу, вибрані вимоги, які їй суперечать, будуть вилучені, а вилучені вимоги, на які вона спирається, будуть вибрані. Таким чином, повнота та послідовність налаштування зберігаються на всіх двох етапах.

2.9 Результати вилучення вимог

Щоб створити програмне забезпечення за допомогою методів SOA, необхідно спочатку виявити служби. Таким чином, результатом процесу виявлення вимог є набір описів послуг, які можна використовувати для виявлення послуг, що задовольняють вибраним вимогам.

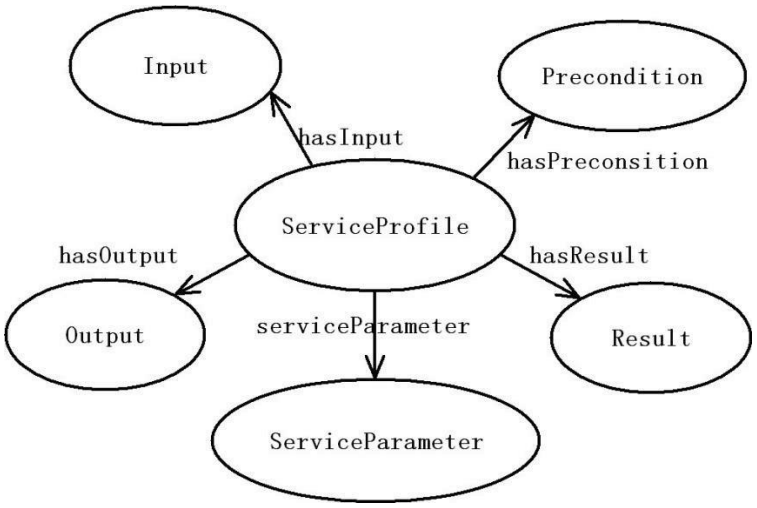


Рисунок 2.8 - Вибрані класи та властивості в описі функціональності OWL-S

У цьому дослідженні OWL-S використовується для опису послуг. OWL-S використовує екземпляр профілю служби для представлення інформації, необхідної для виявлення служби.

Профіль служби має чотири властивості, пов’язані з функціональністю: має вхід, має вихід, має передумову та має результат. Вони пов’язують екземпляр профілю служби з відповідними екземплярами входу, виходу, попередньої умови та результату. І екземпляр Input, Output, Precondition і Result відповідно представлятиме: інформацію, яку служба вимагає для роботи, повідомлення, яке повертає служба, умову, в якій служба виконується належним чином, а також наслідки, а також результати виконання служби.

Обмеження якості послуг явно не визначені в OWL-S. Розширення, запропоноване в [4], використовується для опису якостей і програмних цілей. У цьому розширенні для представлення обмеження використовується властивість Quality, яка узагальнена з параметра служби класу OWL-S. Для цих вимірюваних якостей екземпляри атрибута, успадкованого від якості _ властивості, можуть використовуватися для вираження їх, а також їхніх показників. Для цих абстрактних обмежень (наприклад, програмних цілей) для визначення їхніх стандартів можна використовувати модель якості _, яка з’єднує якість _ властивість через властивість, визначену _ за допомогою. Примірники профілю служби пов’язані з екземплярами якості _ Властивість через службу властивостей Параметр.

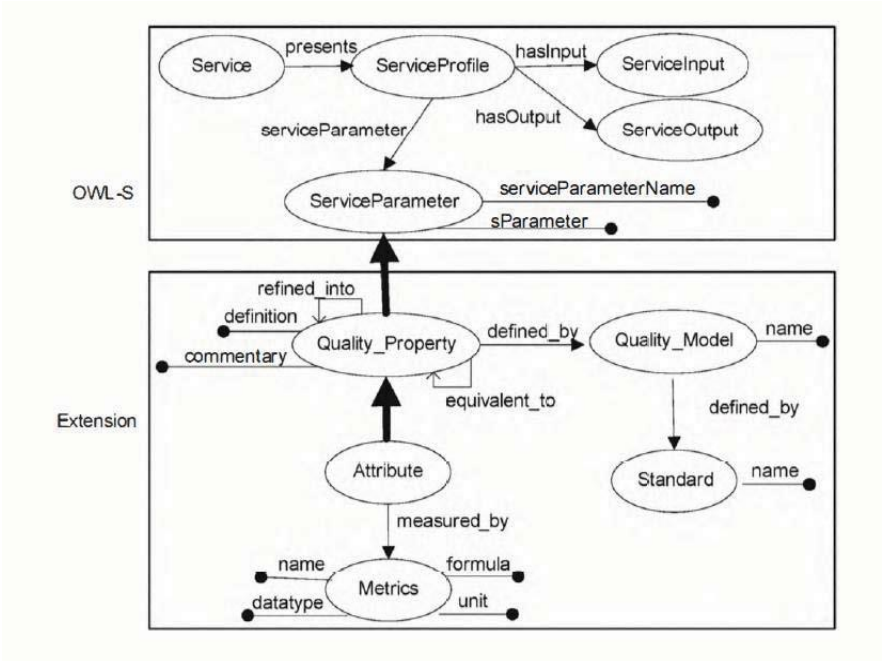


Рисунок 2.9 - Розширення якості OWL-S

Таким чином, для кожної вимоги вона пов’язана з частиною інформації про виявлення служби: або функціональними можливостями, представленими за допомогою Input, Output, Precondition і Result, або обмеженнями якості, представленими Quality_Property. Перетворення вимог в описи послуг — це фактично об’єднання інформації, яка належить до вибраних вимог. Оскільки вимоги організовані в деревах декомпозиції, вибрані вимоги також утворюють групу вибраних піддерев. Процес інтеграції може здійснюватися шляхом об’єднання вузлів у вибраних піддеревах.

Генерування вихідних даних

Процес формування вихідної продукції (рисунок 2.10) поділяється на чотири фази.

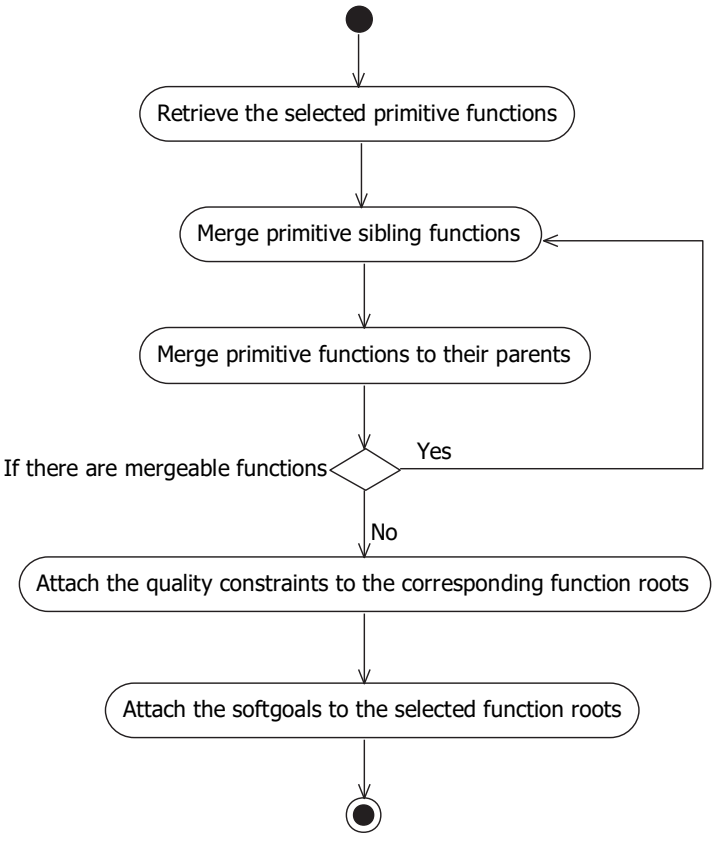


Рисунок 2.10 - Процедура формування опису послуги

Отримати вибрані примітивні функції. Зазвичай примітивні функції мають інформацію про вхід і вихід служби. Якщо між двома вибраними примітивними функціями-побратимами існує зв'язок Rely, це означає, що деякі вхідні дані однієї примітивної функції надходять з виходів іншої функції. Тоді ці відповідні вхідні та вихідні дані не обов'язково виражаються в описі послуги. Тому об'єднайте двох і за алгоритмом, показаним на Рисунок 2.11.

```
1.  //Let f1 and f2 be the selected siblings
2.  //Let f1 rely on f2
3.  //Let f3 be the result of the merge
4.  CREATE function f3
5.  FOR each f2's input, precondition or result
6.      IF f3 doesn't have it THEN
7.          Attach it to f3's inputs, preconditions or results
8.      END IF
9.  END FOR
10. FOR each f1' output, precondition or result
11.     IF f3 doesn't have it THEN
12.         Attach it to f3's outputs, preconditions or results
13.     END IF
14. END FOR
15. FOR each f2's output
16.     IF the output is not an input of f1
17.         AND f3 doesn't have it
18.     THEN
19.         Attach it to f3's outputs
20.     END IF
21. END FOR
22. FOR each f1's input
23.     IF the input is not an output of f2
24.         AND f3 doesn't have it
25.     THEN
26.         Attach it to f3's inputs
27.     END IF
28. END FOR
29. Establish Decompose relationship between f1, f2's parent and f3
30. FOR each Rely relationship f1 and f2 participate in
31.     IF the relationship is not between f1 and f2
32.       AND it involves another requirement r4
33.     THEN
34.         Establish the same relationship between f3 and r4
35.     END IF
36. END FOR
37. Remove f1 and f2 from the selected requirement set
38. Put f3 to the selected requirement set
```

Рисунок 2.11 - Псевдокод для злиття

Алгоритм спочатку створює нову функцію. Потім приєднайте входи, виходи, передумови та результати двох і до нової функції та видаліть непотрібні входи та виходи. Встановіть ті самі відносини Decompose і Rely, в яких беруть участь два брата і сестри для нової функції, тоді як відносини між двома братами і сестрами повинні бути усунені. Потім замініть двох і новою функцією у відповідному вибраному піддереві.

Повторюйте процес злиття і , доки більше примітивних і не можна буде об’єднати.

1. Об’єднайте примітивні дочірні функції з їхніми батьками у вибраних піддеревах функцій. Складені функції зазвичай не містять інформації про вхід або вихід. Оскільки вони представляють композиції стратегії, вони пов’язані з передумовами та результатами. нащадків слід об’єднати з батьками, щоб описи композицій були повними. Алгоритм об'єднання нащадків у батьківський представлений. Алгоритм спочатку прикріплює вхідні дані, вихідні дані, передумови та результати дитини до батьків. Потім установіть ті самі відносини Rely, в яких бере участь дитина, для батьків, і вилучіть дитину з вибраного набору вимог.

Можливо, вибрано батьківську функцію, але жодну з її примітивних дочірніх функцій не вибрано. Якщо це відбувається, і жодні інші вимоги не залежать від батьківської функції, передбачається, що користувачам ця функція насправді не потрібна. Отже, він буде видалений так, ніби його ніколи не вибирали. Якщо деякі вибрані вимоги залежать від нього, його неможливо видалити безпосередньо. У цьому випадку для композитної функції будуть попередньо визначені деякі примітивні функціональні описи за замовчуванням. За допомогою цих описів складову функцію можна розглядати як примітивну функцію.

Після того, як примітивні діти об’єднуються зі своїми батьками, об’єднаний батько формує нові «примітивні» функції. Також можливо, що нові примітивні функції покладаються на деяких зі своїх і . У цьому випадку

запустіть для них алгоритм злиття. Повторюйте об'єднання нових примітивних нащадків у батьків, поки не залишиться лише коріння вибраних піддерев.

```
1.  //Let p1 be the selected parent
2.  //Let c2 be the selected child
3.  FOR each c2's input, output, precondition or result
4.      IF p1 doesn't have it THEN
5.          Attach it to p1's inputs, outputs, preconditions or results
6.      END IF
7.  END FOR
8.  FOR each Rely relationship c2 participates
9.      IF the relationship is not between p1 and c2
10.       AND it involves another requirement r3
11.     THEN
12.         Establish the same relationship between p1 and r3
13.     END IF
14. END FOR
15. Remove c2 from the selected requirement set
```

Рисунок 2.12 - Псевдокод для злиття

1. Приєднайте вибрані якості листя до відповідних функціональних коренів. Батьківські якості не є конкретними обмеженнями, тому вони не несуть ніякої інформації, пов'язаної з послугами. Для кожної якості листка спочатку знайдіть вибрану функцію, яка з нею пов'язана. Потім простежте корінь вибраного піддерева, яке містить функцію. Додайте службову інформацію про якість листка до опису служби кореня. Якщо вибрано батьківську якість, але жоден з її дочірніх елементів не вибрано, для її обробки буде застосовано те саме рішення для функції.

2. Приєднайте вибрані програмні цілі листка до коренів функції. Батьківські програмні цілі також не містять інформації, пов'язаної з послугами. Додайте інформацію про програмні цілі листка до всіх вибраних коренів функції. Якщо вибрано батьківську програмну ціль, але жоден з її дочірніх не вибрано, буде застосовано те саме рішення щодо функції та якості.

Нарешті, інтегровані описи служб (тобто файли OWL-S), що передаються коренями вибраних піддерев функцій, утворюють вихідні дані виявлення вимог. Коли служба для кореневої функції виявлена, описи примітивних служб, на яких

побудований корінь, також переносяться специфікацією цієї служби. Таким чином, під час генерації вихідних даних службова інформація для примітивних функцій зливається в описи кореневих функцій, і немає необхідності описувати примітивні функції окремо. З іншого боку, об'єднання описів служб із кореневими функціями є необов'язковим. Користувачі можуть вибрати, щоб обмеження якості та softgoal були приєднані безпосередньо до примітивних функцій, а потім виявляти та оцінювати вибрані примітивні функції окремо. В результаті замість того, щоб застосовувати композиції послуг, які пропонують постачальники послуг, вони можуть самостійно визначати стратегії складання послуг.

2.10 Рекомендації щодо впровадження системи

Впровадження сервісно-орієнтованих систем включає два етапи: виявлення сервісу та складання сервісу. Рішення щодо автоматичного виявлення сервісів і компонування за допомогою OWL-S обговорюються в [8]. У цьому розділі представлено короткий огляд.

На відміну від UDDI, який підтримує пошук служби на основі ключових слів, OWL-S може описати семантику служб. За допомогою відповідності семантичних можливостей можна подолати обмеження синтаксичного зіставлення служб, і виявлення сервісів стане більш розумним. По суті, існує два типи представлення можливостей служби: перший полягає у використанні розширеної ієрархії класів для визначення детальних функцій; інший — визначити перетворення стану, що є результатом виконання послуги. OWL-S підтримує обидві презентації [4], що означає, що підбір можливостей з OWL-S є більш перспективним. Крім того, в літературі запропоновано безліч алгоритмів збігу, а також інших алгоритмів виявлення сервісів для автоматизації виявлення сервісів на основі OWL-S (перелік алгоритмів див. у [8]). Таким чином, за

допомогою вихідних даних виявлення вимог на основі OWL-S відповідні послуги можна знайти автоматично.

Коли виявлені послуги на основі OWL-S, їх композицію слід виконати, щоб забезпечити потрібну функціональність. У OWL-S композиція описується за допомогою екземплярів ServiceModel, які будуть отримані, коли служби будуть виявлені. Кожна ServiceModel фактично представляє моделі процесу складання та виконання відповідної служби. Моделі процесів можна використовувати для побудови загальних процедур і планів реалізації функціональних можливостей. Для автоматичної композиції послуг на основі OWL-S було запропоновано кілька методів планування ШІ [8]. Незважаючи на те, що більшість підходів ще недостатньо зрілі і все ще потрібна подальша робота для реалізації автоматичної композиції сервісу OWL-S, автоматизована композиція для служб збору інформації на основі OWL-S вже може бути виконана [4].

На даний момент представлено повний процес від визначення вимог до реалізації системи. З підходом SPL сервісно-орієнтоване програмне забезпечення замовляється і очікується, що воно буде реалізовано автоматично. Крім того, безпосередньо взаємодіючи з текстовою діалоговою системою, розробленою в цій дипломній роботі, користувачі можуть належним чином налаштувати очікуваний програмний продукт без будь-яких помилок.

3 ВЛАСНЕ ОРИГІНАЛЬНЕ АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ РІШЕННЯ ЗАДАЧІ

3.1 Реалізація

Система діалогу розроблена на Java 6.0 на платформі x86 з операційною системою Windows. Eclipse IDE (3.6) використовувався для полегшення кодування та налагодження.

На Рисунок 3.1 зображено інтерфейс діалогової системи. Він розділений на три частини.

Висловлювання, згенеровані менеджером діалогів, відображаються у верхньому лівому текстовому полі. Користувачі можуть ввести свою відповідь у нижньому лівому текстовому полі. Тим часом три списки праворуч містять вибрані, вилучені та оцінені вимоги відповідно.

Онтологія побудована за допомогою редактора онтології Protégé , а Pellet використовується як резонатор онтології. Pellet підтримує міркування як з OWL DL, так і з DL-безпечним SWRL.

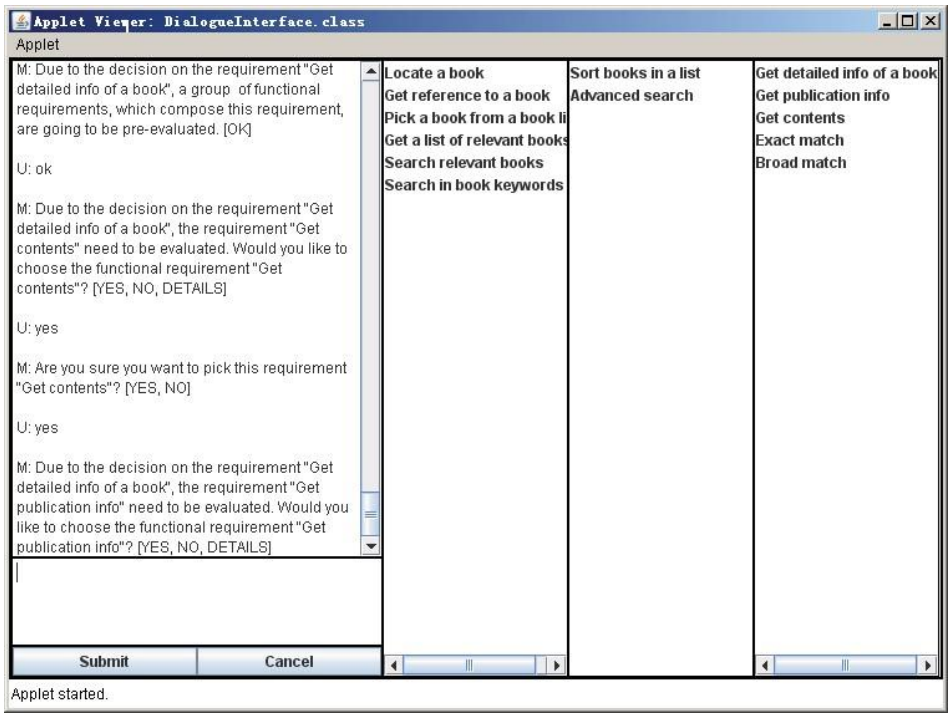


Рисунок 3.1 - Інтерфейс запропонованої діалогової системи

Нижче наведено поняття, об’єктні зв’язки, зв’язки з даними, правила та екземпляри моделі вимог, створених за допомогою редактора Protégé. Рисунок 3.2 ілюструє концепції для онтологічної моделі на основі онтології, а зв'язки між цими поняттями представлені на Рисунок 3.3. На Рисунок 3.4 створена група зв'язків даних, що описують поняття з примітивними типами даних. Між тим, правила SWRL, розроблені для моделі вимог, наведені на Рисунок 3.5, а потім на Рисунок 3.6, де екземпляри для вивчення прикладу, які будуть розглянуті в наступному розділі, будуються з моделлю онтології.

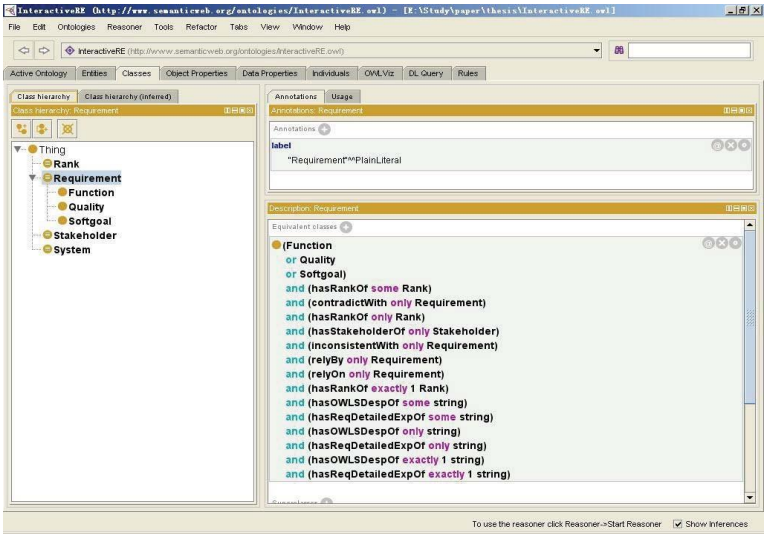


Рисунок 3.2 - Класи, створені для запропонованої моделі онтології

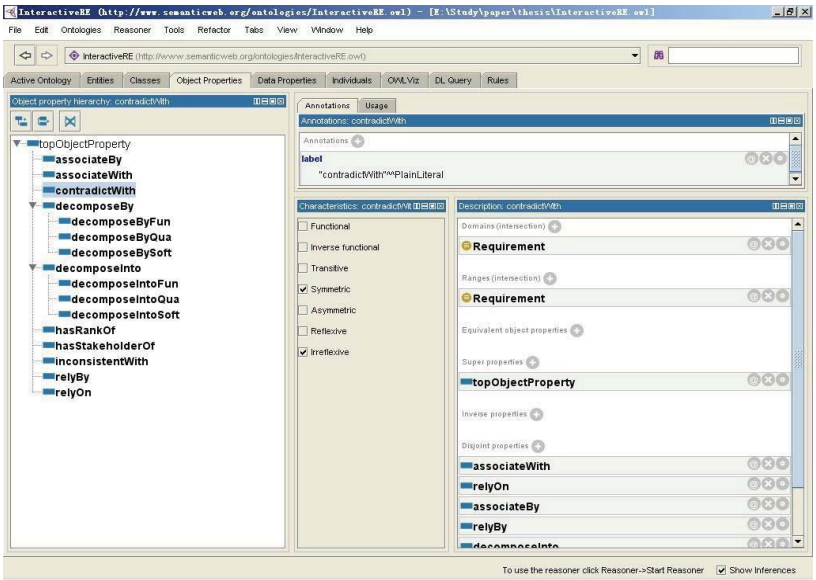


Рисунок 3.3 - Властивості об'єкта, створені для запропонованої моделі онтології

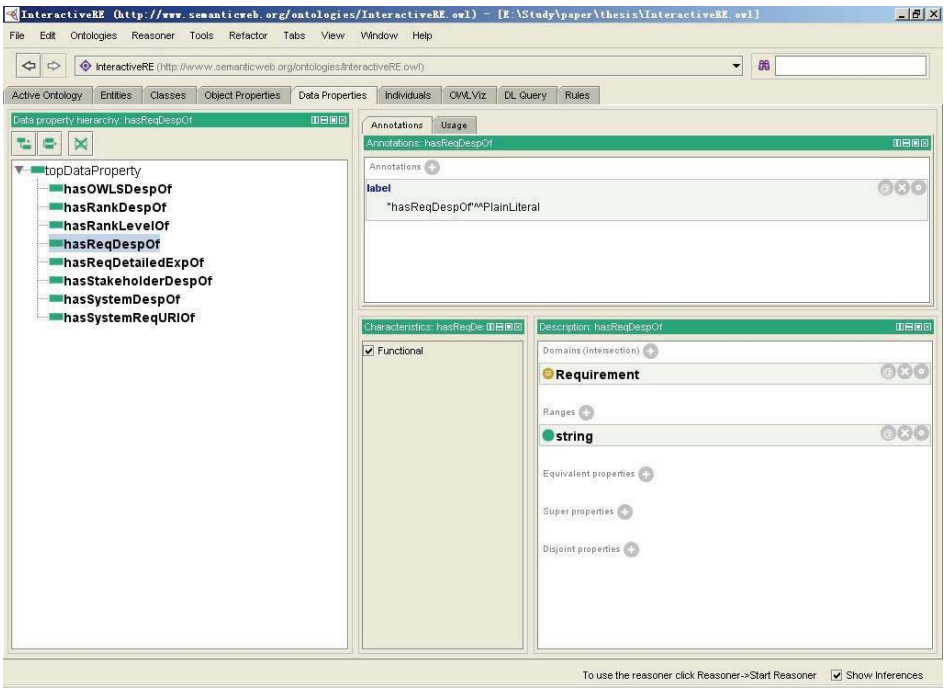


Рисунок 3.4 - Властивості даних, створені для запропонованої моделі Онтології

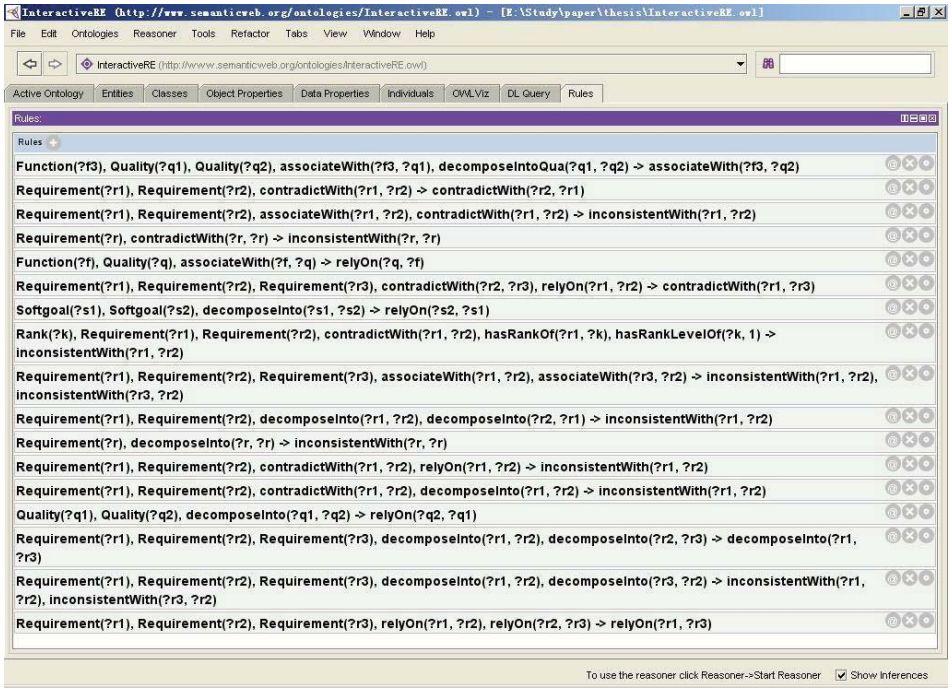


Рисунок 3.5 - Правила, створені для запропонованої моделі онтології

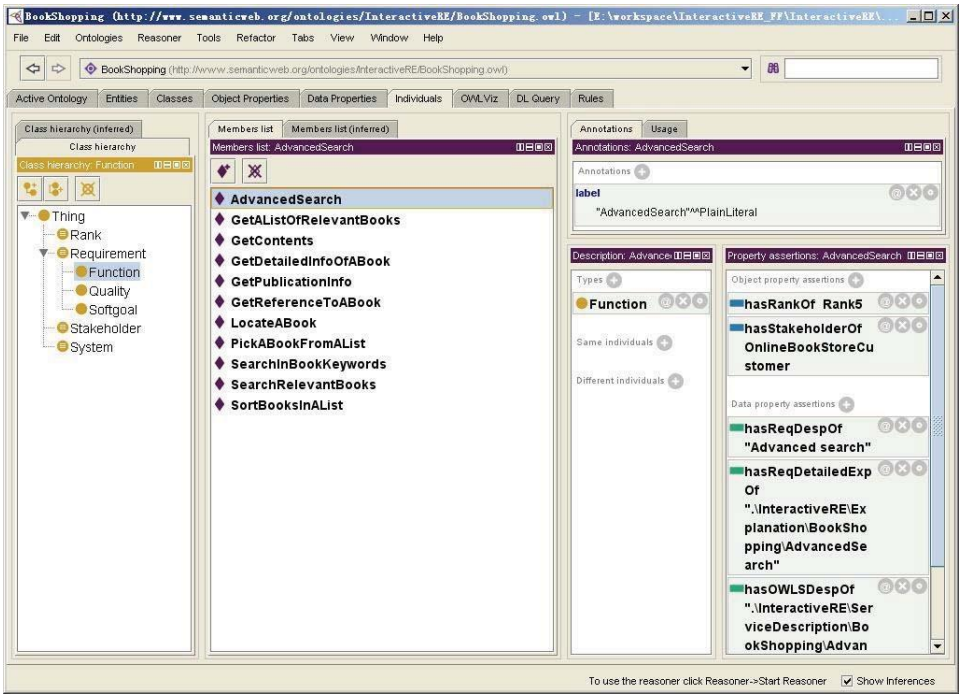


Рисунок 3.6 - Примірники, створені за допомогою запропонованої моделі онтології

3.2 Приклад побудови рішення

Як приклад у цьому дослідженні використано систему онлайн-магазину книг. Структура типової, але спрощеної системи покупок книг в Інтернеті проілюстрована на рисунку 3.7. В основному є чотири модулі: пошук книги, керування кошиком, керування обліковим записом та розміщення замовлень. Модуль пошуку книг відповідає за пошук книг та отримання книжкової інформації; модуль керування кошиком надає список, де користувачі можуть зберігати посилання на книги, які вони хочуть купити; модуль керування обліковим записом керує особистою інформацією користувачів, інформацією про доставку та оплату; Модуль розміщення замовлення збирає таку інформацію, як список покупок, оплата, доставка та загальна ціна, і допомагає користувачам розмістити замовлення. Передбачається, що керування обліковими записами не є необхідним для системи покупок книг онлайн.

Користувачі можуть вказати необхідну інформацію для кожної покупки, не зберігаючи її в книжковому інтернет-магазині.

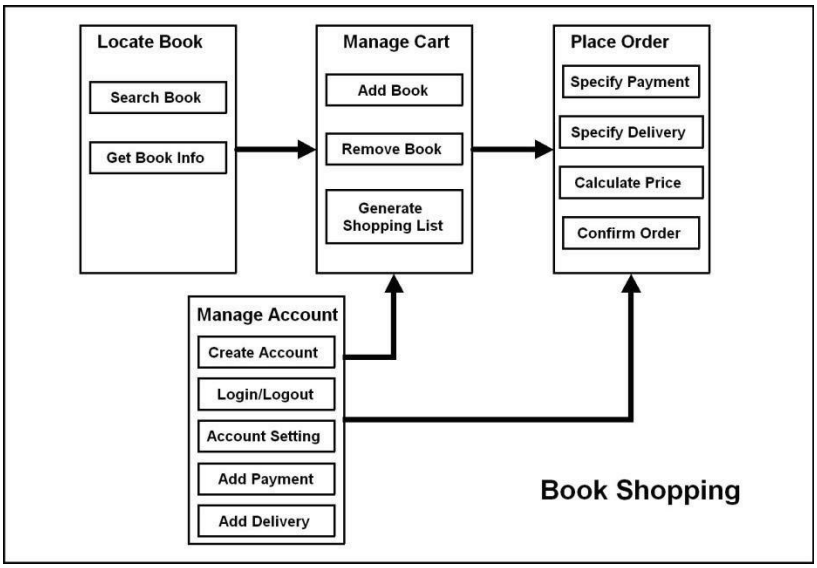


Рисунок 3.7 - Функціональні можливості системи інтернет-магазину книг

Пояснити всі деталі надто складно. У цій дипломній роботі процес налаштування модуля пошуку книг із запропонованим підходом детально представлений як тематичний приклад.

Служба пошуку книг

1. Функція декомпозиції

Спочатку модель онтології створюється з вимогами модуля пошуку книги. По суті, модуль пошуку книг надає дві функції: отримати посилання на книгу та отримати детальну інформацію про книгу. Деталі книги можуть включати інформацію про публікацію, зміст, зразки розділів тощо. Тут як приклади використано інформацію та зміст публікації. Крім того, для того, щоб отримати посилання на книгу, найпоширенішим методом є пошук. Пошук поверне список релевантних книг, тому користувачам потрібно вказати саме книгу зі списку. Крім того, щоб полегшити користувачам пошук книги серед списку книг, можна застосувати сортування. Таким чином, отримання списку книг може містити два

підпроцеси: пошук книги та сортування результатів пошуку. Крім того, є два способи пошуку книг. Один із них — узгодження введених користувачів із попередньо визначеними ключовими словами книг. Інший — розширений композиційний пошук. Користувачі можуть надавати детальну інформацію, таку як автори та публікації, щоб звузити область пошуку. Існує два рівні відповідності ключових слів: широка та точна відповідність. Точна відповідність намагається повернути результати, які є найбільш релевантними для введених даних, тоді як широка відповідність дозволяє повернути щось схоже на вхідні дані, але не зовсім пов’язані з вхідними параметрами. З іншого боку, широка відповідність може дати щось несподіване, але цікаве. Таким чином, вони є двома різними рівнями обмежень якості пошуку та взаємовиключними.

Тепер пошук книг повністю розкладено на примітивні функції та обмеження якості. Нижче наведено опис кожної з функцій.

Пошук у книзі за ключовими словами: input – фрази (від користувачів); вихід – список відповідних книг і список посилань на книги

Розширений пошук: введення – поля для відповідності та фрази для кожного поля (від користувачів); вихід – список відповідних книг і список посилань на книги

Пошук релевантних книг: введення – (з підфункцій); вихід – (з підфункцій)

Сортувати книги у списку: input – порядок сортування (за користувачами) та список посилань на книги (з Пошуку релевантних книг); вихід – список відповідних книг і список посилань на книги

Отримати список відповідних книг: введення – (з підфункцій); вихід – (з підфункцій)

Виберіть книгу зі списку: input – індекс книги у списку (від користувачів) та список посилань на книги (з Отримати список релевантних книг); вихід – посилання на книгу

Отримати посилання на книгу: введення – (з підфункцій); вихід – (з підфункцій)

Отримати інформацію про публікацію: вхід – посилання на книгу (з Отримати посилання на книгу); вихід – інформація про публікацію книги

Отримати вміст: input – посилання на книгу (з Отримати посилання на книгу); вихід – зміст книги

Отримати детальну інформацію про книгу: введення – (з підфункцій); вихід – (з підфункцій)

Знайдіть книгу: вхід – (з підфункцій); вихід – (з підфункцій)

2. Примірник онтології

Широка і точна відповідність є взаємовиключними, тому між ними існує взаємозв'язок Contradict. Вони є обмеженням якості для пошуку в книзі за ключовими словами. У результаті вони пов'язані з пошуком у книзі за ключовими словами через зв'язок асоційованого. Тим часом, пошук за ключовими словами книги є важливим для пошуку релевантних книг; Пошук релевантних книг необхідний для отримання списку релевантних книг; Отримати список релевантних книг і вибрати книгу зі списку необхідні для отримання посилання на книгу; Щоб знайти книгу, необхідно отримати посилання на книгу. Таким чином, існують відносини Rely, що вказують від батьків до нащадків. Крім того, сортування книг у списку залежить від результатів пошуку релевантних книг; Вибір книги зі списку покладається на вихідні дані Отримати список відповідних книг; Отримання детальної інформації про книгу покладається на вихідні дані Отримати посилання на книгу. Отже, між кожною парою існують відносини Rely. Тим часом вибір книги зі списку може лише сприяти отриманню посилання на книгу; Отримати посилання на книгу можна лише для пошуку книги.

В результаті виникають стосунки Rely від двох нащадків до двох батьків.

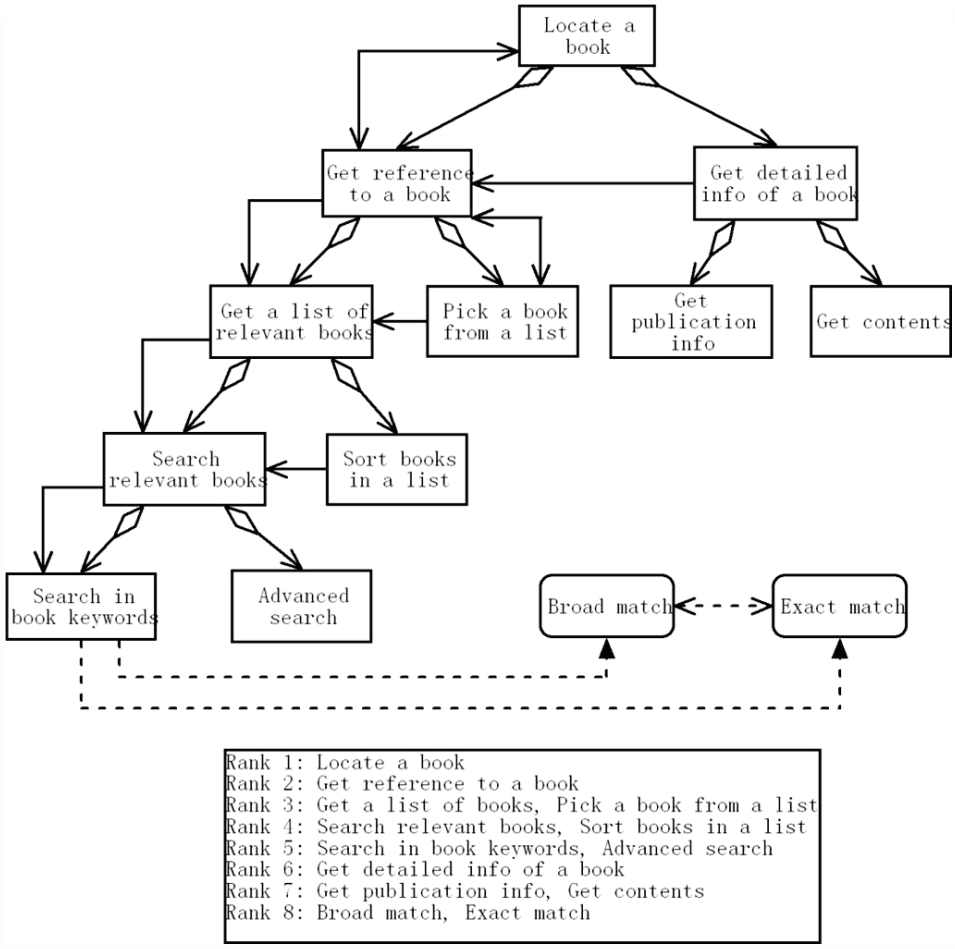


Рисунок 3.8 - Модель вимоги, створена за допомогою служби пошуку книги

Ранги вимог також показані на рисунку 3.8. Оскільки ядро Locate a book полягає в тому, щоб отримати посилання на книгу, очікується, що вимоги, що описують Отримати посилання на книгу, будуть оцінені першими. Таким чином, отримати детальну інформацію про книгу, а також її дочірніх є відносно низький ранг.

3.3 Вилучення вимог

Коли модель онтології готова, можна виконати виділення вимог. У цьому випадку користувачам потрібна система програмного забезпечення для

отримання детальної інформації про книгу. Тому вони хочуть лише шукати книгу з точним відповідністю та отримати всю доступну інформацію про книгу.

На рисунку 3.9 користувачів спочатку вітають та повідомляють, що суттєва вимога Знайти книгу вибирається обов'язково. Потім слід попередньо оцінити дві підвимоги Отримати посилання на книгу та Отримати детальну інформацію про книгу. Але Отримати посилання на книгу покладається на функцію Знайти книгу. Таким чином, він уже був попередньо обраний, і його попередня оцінка пропущена. Отримати детальну інформацію про книгу попередньо оцінено, але користувачам потрібні додаткові пояснення щодо неї. Після того як користувачі дізнаються про роль Отримати детальну інформацію про книгу в системі, яку потрібно створити, вони вибирають її.

На рисунку 3.10 показано оцінку Отримати посилання на книгу, а також на двох її дочірніх. Оскільки відношення Rely є перехідним, усі вони спираються на функцію Locate a book і були попередньо вибрані. На Рисунок 3.11 оцінюються Пошук релевантних книг і Сортування книг у списку. Потім настає черга оцінити вимогу пошуку за ключовими словами книги та попередньо оцінити два пов'язані з ними обмеження якості.

Під час цього процесу користувачі вводять деяку інформацію, яку машина не може обробити (Рисунок 3.12). Потім машина сповіщає користувачів, щоб вони виправили свою відповідь. За допомогою взаємодії, представленої на Рисунок 3.13 – Рисунок 3.15, користувачі вибирають необхідні функції та обмеження якості. Після цього вони не бажають змінювати свої рішення, і процес виявлення вимог завершується.

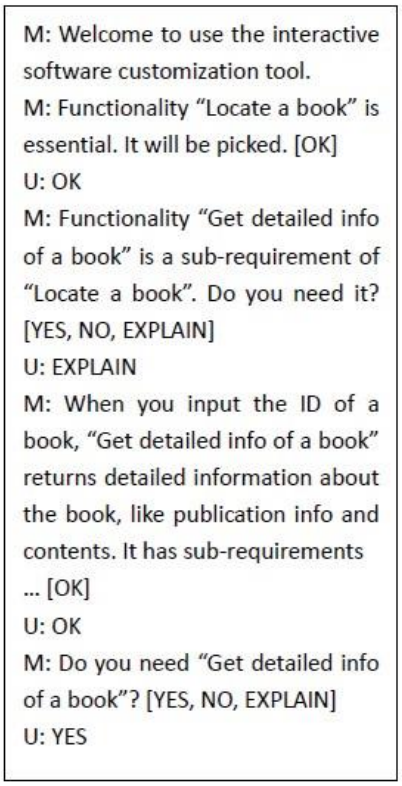


Рисунок 3.9 - Діалогічні висловлювання

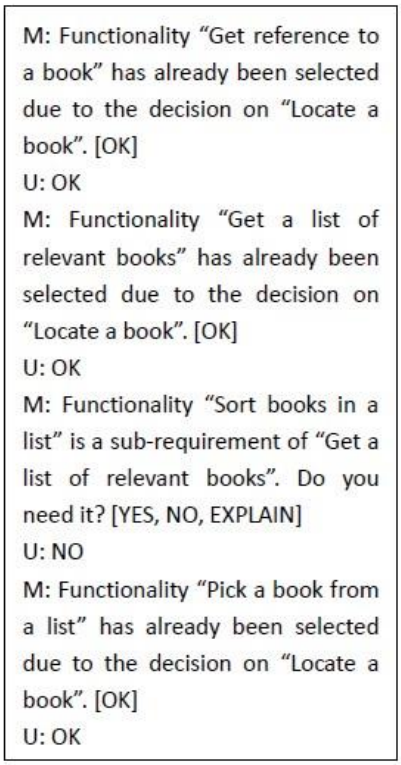


Рисунок 3.10 - Діалогічні висловлювання

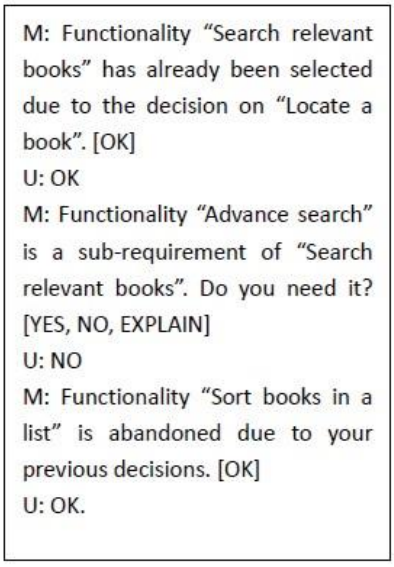


Рисунок 3.11 - Діалогічні висловлювання для налаштування розташування книг служба (частина 3)

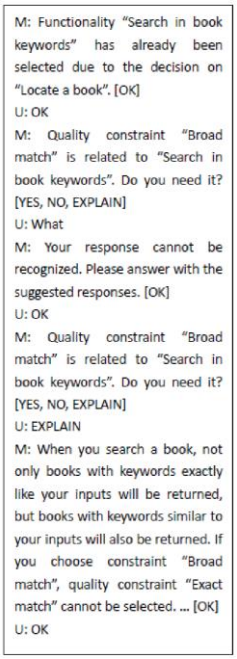


Рисунок 3.12 - Діалогові висловлювання для налаштування послуги пошуку книги (частина 4)

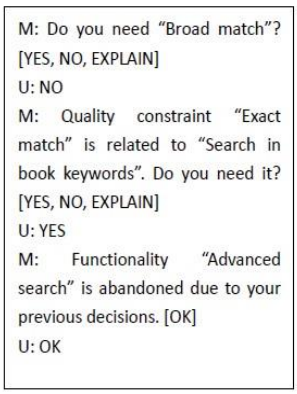


Рисунок 3.13: Діалогові висловлювання для налаштування розташування книги для Сервіс (частина 5)

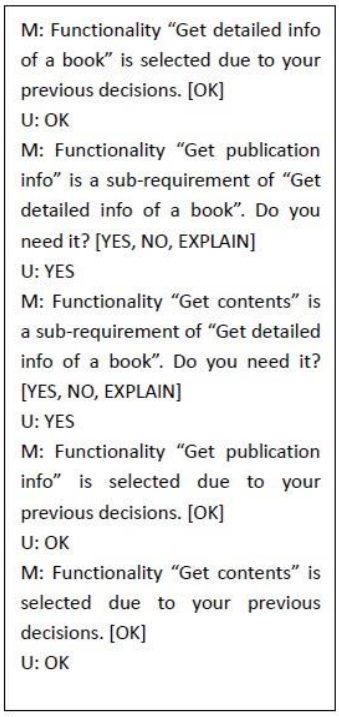


Рисунок 3.14: Діалогові висловлювання

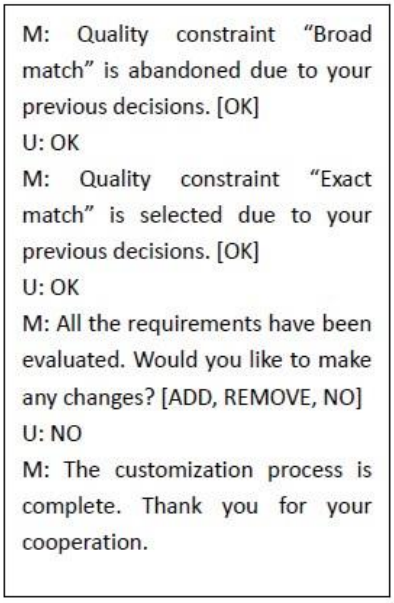


Рисунок 3.15: Діалогові висловлювання

Генерація вихідних даних

Нарешті, вимоги Шукати за ключовими словами книги, Шукати релевантні книги, Отримувати список релевантних книг, Виберіть книгу зі списку, Отримайте посилання на книгу, Отримайте інформацію про публікацію, Отримайте вміст, Отримайте детальну інформацію про книгу, Знайдіть книгу та Вибирається точна відповідність.

Тоді вихідні дані будуть згенеровані таким чином, щоб об'єднати піддерево, засноване на Отримати посилання на книгу, що включає злиття і між Отримати список релевантних книг і Вибрати книгу зі списку, а також об'єднання Отримати інформацію про публікацію, Отримати вміст у Отримати детальну інформацію про книгу. В результаті залишиться три функції: Отримати посилання на книгу, Отримати детальну інформацію про книгу та Знайти книгу. Оскільки Отримати детальну інформацію про книгу покладається на вихідні дані Отримати посилання на книгу, вони будуть об'єднані, щоб утворити нову функцію, яка негайно об'єднається в Знайти книгу.

Нарешті, якість Точна відповідність буде додано безпосередньо до Знайти книгу. Вихід BookShoppingProfile.owl виглядає як документ OWL-S, представлений на Рисунок 3.13. Примірники вхідних даних, вихідних даних і якостей визначені в документах BookShoppingProcess.owl і BookShoppingQuality.owl. Вони імпортуються профільним документом.

Тепер, з цим описом OWL-S, очікується, що служби будуть виявлені шляхом відповідності семантичних можливостей. Тоді складання та виконання послуг буде виконуватися на основі відповідної інформації про склад послуг, запропонованої постачальниками послуг.

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:profile="http://www.daml.org/services/owl-s/1.2/Profile.owl#"
         xmlns:owl="http://www.w3.org/2002/07/owl#">

<owl:Ontology rdf:about="">
         <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
         <owl:imports rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns"/>
         <owl:imports rdf:resource="ttp://www.w3.org/2002/07/owl"/>
         <owl:imports rdf:resource="http://www.semanticweb.org/ontologies/
                InteractiveRE/BookShoppingProcess.owl"/>
         <owl:imports rdf:resource="http://www.semanticweb.org/ontologies/
                InteractiveRE/BookShoppingQuality.owl"/>
</owl:Ontology>

<profile:Profile rdf:ID="Locate_a_book">
         <profile:textDescription>Locate a book</profile:textDescription>

         <profile:has_process ref:resource="http://www.semanticweb.org/ontologies/
                InteractiveRE/BookShoppingProcess.owl#LocateABookProcess"/>

         <profile:hasInput rdf:resource="http://www.semanticweb.org/ontologies/
                InteractiveRE/BookShoppingProcess.owl#PhrasesFromUserInput"/>
         <profile:hasInput rdf:resource="http://www.semanticweb.org/ontologies/
                InteractiveRE/BookShoppingProcess.owl#BookIndexInTheList"/>

         <profile:hasOutput rdf:resource="http://www.semanticweb.org/ontologies/
                InteractiveRE/BookShoppingProcess.owl#ListOfRelevantBooks"/>
         <profile:hasOutput rdf:resource="http://www.semanticweb.org/ontologies/
                InteractiveRE/BookShoppingProcess.owl#BookContents"/>
         <profile:hasOutput rdf:resource="http://www.semanticweb.org/ontologies/
                InteractiveRE/BookShoppingProcess.owl#BookPublicationInfo"/>

         <profile:serviceParameter rdf:resource="http://www.semanticweb.org/ontologies/
                InteractiveRE/BookShoppingQuality.owl#ExactMatch"/>

</profile:Profile>
</rdf:RDF>
```

Рисунок 3.16 - Опис послуги виведення

У запропонованому підході витягуються неявні знання, приховані в моделі онтології, щоб спрямувати процес виявлення вимог. Тим часом підхід використовує стратегію, яка завжди підтримує повноту та узгодженість результатів виявлення. Коли вимога обрана, будуть вибрані всі інші вимоги, на які вона спирається, а ті, які їй суперечать, будуть видалені. Коли вимога відкидається, вимоги, які на неї покладаються, також відкидаються. З іншого боку, без неявних знань і вказівок людям зазвичай доводиться вирішувати неповноту та непослідовність крок за кроком. У цій дипломній роботі були проведені імітаційні експерименти для порівняння ефективності запропонованого методу з ненаправленим методом.

3.4 Генерація екземплярів проблеми

Створення екземпляра проблеми означає побудову лісу декомпозиції вимог. Підмножина вимог, що містяться в деревах, буде обрана як очікувані вимоги.

У цьому дослідженні екземпляри проблем генеруються випадковим чином. Функції, якості та програмні цілі не розмежовуються. Усі вони розглядаються просто як вимоги. Під час побудови екземпляра проблеми кількість вимог n і кількість зв'язків Rely, Contradict m, які потрібно створити, є двома вхідними параметрами для екземпляра проблеми.

Спочатку під час створення i-го вузла генерується випадкове ціле число j в діапазоні від [0, i-1]. j визначає позицію лісу для додавання i-го вузла. Якщо він дорівнює нулю, створіть i-ий вузол як корінь. В іншому випадку i-ий вузол приєднується як дочірній до попередньо згенерованих j-их вузлів. Якщо j-й вузол є листком, коли до нього приєднується i-ий вузол, до нього також приєднується (i+1)-й вузол. Це тому, що вимога не повинна розкладатися лише на одну дитину. Якщо j-ий вузол не є листком, до нього приєднується лише i-ий вузол. Якщо настала черга створити n-ий вузол, оскільки вузлів більше немає, його не можна приєднати до аркуша. Його можна створити як корінь або приєднати до нелисткового вузла в дитинстві. Потім генерується ліс розкладання, і передбачається, що ранг i-го вузла дорівнює i.

```
1.  //Let n be the number of requirements to be generated
2.  i ← 1
3.  WHILE i <= n
4.      j ← a random number from [0, i)
5.      IF j == 0 THEN
6.          Create the ith requirement as a root
7.          i ← i+1
8.      ELSE IF the jth requirement is not a leaf THEN
9.          Create the ith requirement as a child of the jth requirement
10.         i ← i+1
11.     ELSE IF i != n
12.         Create the ith requirement as a child of the jth requirement
13.         Create the (i+1)th requirement also as a child of the jth requirement
14.         i ← i+2
15.     END IF
16. END WHILE
```

Рисунок 3.17 - Псевдокод для побудови декомпозиції

Після цього повинні бути згенеровані відносини m Rely і Contradict. Кожного разу випадковим чином вибираються два вузли з лісу, і Rely або Contradict випадковим чином вибираються як відносини-кандидати між ними. Достовірність лісу буде перевірена за допомогою онтологічних міркувань після того, як відношення-кандидат буде приєднано до лісу. Якщо відношення-кандидат є дійсним і не є дублюваним, згенеруйте наступне зв'язок. В іншому випадку видаліть зв'язок-кандидат з лісу та створіть нове.

Коли генерується m дійсних і недублюваних зв'язків, створюється екземпляр проблеми. Підмножина з n вимог буде випадковим чином обрана як вимоги, які очікують вибрати користувачі. Ця підмножина вимог буде перемішана у списку. Передбачається, що вимога з нижчим індексом у списку має вищий очікуваний пріоритет, що означає, що вона більш затребувана. Часто правда, що ця підмножина вимог не узгоджена чи не повна одна з одною. Користувачі повинні віддавати перевагу вимогам з вищими очікуваними пріоритетами, що означає, що вони відмовляться від очікуваної вимоги, якщо вона суперечить вимогам з вищими очікуваними пріоритетами. Відповідно до цієї інструкції користувачі намагатимуться проводити менше раундів взаємодії для виконання завдань і вибирати якомога більше очікуваних вимог, зберігаючи вибрані вимоги повними та узгодженими один з одним.

3.5 Експеримент із запропонованим методом

Запропонований підхід призначений для оцінки вимог по черзі, від найвищого рангу до найнижчого. Під час експериментів користувачі, які моделюються, повинні були оцінити вимоги. Під час оцінки вимоги, якщо вона не очікується, користувачі пройдуть через очікувані вимоги від найвищого очікуваного пріоритету до найнижчого. Якщо оцінювана вимога спирається на очікувану вимогу, але вона не суперечить очікуваним вимогам з більш високим очікуваним пріоритетом, вона буде вибрана. Інакше він буде скинутий. Якщо

вимога очікується, перегляньте очікувані вимоги від найвищого очікуваного пріоритету до себе. Якщо це не суперечить вимогам з вищим очікуваним пріоритетом, виберіть його, інакше відкиньте його. Як пояснюється вибір або відмова від вимоги із запропонованим підходом призведе до відповідного оброблення всіх відповідних вимог.

Таблиця 3.1 - Результати для першої групи експериментів (n:m=10:20-100:20)

A	B	C	D	E	F	G
50 / 10	22	80.9%	2	0	0	0
50 / 20	16	50.5%	3	1	2	0
50 / 30	5	30.2%	3	0	0	0
50 / 40	6	6.8%	2	3	0	0
50 / 50	2	6.4%	2	0	0	0
50 / 60	2	0.0%	1	1	0	0
50 / 70	0	-	-	-	-	-
50 / 80	0	-	-	-	-	-
50 / 90	0	-	-	-	-	-
50 / 100	0	-	-	-	-	-

Таблиця 3.2 - Результати для другої групи експериментів (n:m=50:10-50:100)

A	B	C	D	E	F	G
10 / 10	12	58.3%	5	1	0	1
20 / 20	8	0.0%	4	1	1	0
30 / 30	8	5.9%	2	1	0	1
40 / 40	5	11.5%	1	2	1	0
50 / 50	0	-	-	-	-	-
60 / 60	3	6.8%	0	1	2	0
70 / 70	0	-	-	-	-	-
80 / 80	0	-	-	-	-	-
90 / 90	0	-	-	-	-	-
100 / 100	0	-	-	-	-	-

Нижче наведено пояснення до колонок.

Колонка A: кількість вимог / кількість зв’язків

Стовпець B: кількість успішних ітерацій, виконаних ненаправленим методом

Стовпець C: середній відсоток більшої кількості взаємодій запропонований метод був заряджений у порівнянні з ненаправленим методом (відсоток розраховується таким чином: ^кількість взаємодій запропонований

метод був заряджений¯ кількість взаємодій ненаправлений метод був заряджений∨`кількість взаємодій ненаправлений метод стягнуто × 100%)

Колонка D: кількість успішних ітерацій, під час яких запропонований метод вибрав більше очікуваних вимог

Стовпець E: кількість успішних ітерацій, під час яких ненаправлений метод вибрав більше очікуваних вимог

Стовпець F: кількість успішних ітерацій, під час яких запропонований метод вибрав рівну кількість очікуваних вимог як ненаправлений метод і менше несподіваних вимог

Стовпець G: кількість успішних ітерацій, у яких ненаправлений метод вибрав рівну кількість очікуваних вимог як запропонований метод і менше несподіваних вимог

З усіх результатів можна помітити, що ненаправлений підхід не може обіцяти повних і послідовних результатів, тоді як запропонований підхід призначений для подолання цієї проблеми. У найкращих випадках, протягом 22 з 25 ітерацій, дійсні результати можуть бути отримані ненаправленим методом. Більше того, у значній кількості успішних ітерацій запропонований метод мав більш очікувані вимоги. З іншого боку, більшість вимог було оцінено принаймні один раз за запропонованим підходом, тоді як ненаправлений підхід стосується лише очікуваних вимог. Таким чином, запропонований підхід часто вимагає більшої кількості раундів взаємодії.

У першій групі вимог, коли n збільшується, а m залишається незмінним,

непрямий підхід працював все краще і краще. Для виконання завдань потрібно було все менше і менше раундів взаємодій, а рівень успіху зростав дуже швидко. При цьому запропонований метод так і не зазнав поразки. Завжди були деякі успішні ітерації, в рамках яких запропонований метод отримував більш очікувані вимоги.

Для другої групи вимог, коли m зростало і n було фіксовано, продуктивність ненаправленого методу різко знижувалася. Коли було більше 40

зв’язків, це навряд чи могло дати якийсь дійсний результат. Варто зазначити, що, хоча запропонований метод міг легко перемогти ненаправлений метод, у рідкісних випадках успішних ітерацій неорієнтований метод виявляв більш очікувані вимоги. В основному, цей результат був зумовлений стратегією моделювання, а не запропонованим методом виявлення. Змодельований користувач завжди намагається отримати очікувану вимогу з найвищим очікуваним пріоритетом. Іноді досягається найбільш затребувана вимога, але іншими очікуваними вимогами нехтують.

У третій групі, навіть якщо n і m підвищувалися з однаковою швидкістю, продуктивність ненаправленого підходу зменшувалася дуже швидко зі збільшенням складності задачі. При цьому відстань між кількістю взаємодій, які вимагає запропонований метод, і кількістю взаємодій, необхідних для ненаправленого підходу, скорочується, коли проблема ускладнюється.

На завершення, ці експерименти показують, що виявлення вимог – непросте завдання. Без неявних знань і належного керівництва майже неможливо отримати дійсний і очікуваний результат. Проте метод, запропонований у цій дипломній роботі, може успішно допомогти вирішити цю проблему.

ВИСНОВКИ

В дипломній роботі з метою реалізації автоматизованого рішення за допомогою сервіс-орієнтованих методів, запропоновано підхід до інтерактивного виявлення вимог. Він використовує онтологію для представлення знань, пов’язаних із розробкою вимог, що спрямовує діалогову систему заповнення слотів для спілкування з клієнтами. За допомогою цього методу користувачі можуть налаштувати вимоги програми, які задовольняють їхні вимоги, взаємодіючи з машинами, при цьому забезпечується повнота та узгодженість налаштування. Упорядковані вимоги будуть потім перетворені в описи послуг на основі онтології для впровадження системи. У цій роботі наведено приклад для підтвердження доцільності запропонованого методу, та для перевірки його ефективності та надійності були проведені імітаційні експерименти.

З іншого боку, незважаючи на те, що в цій роботі було докладено зусиль для досягнення автоматизованого визначення вимог, запропонована модель вимог все ще є попередньою та малозваженою. Оскільки модель статична, її неможливо застосувати в динамічному середовищі. Більше того, щоб уникнути додаткової складності, не всі онтологічні відносини, пов’язані з інженерією вимог, безпосередньо описуються. Нарешті, підхід до визначення вимог, що повідомляється, підтримує лише налаштування вимог на основі знань, якими володіє машина. Користувачі не можуть замовити нічого невідомого машині, що не завжди буває на практиці.

Для майбутніх робіт, по-перше, з метою впровадження автоматизованого рішення, будуть додатково досліджені підходи, пов’язані з автоматичною реалізацією додатків, такі як автоматичне виявлення сервісів, їх композиція та доставка. При цьому необхідно оптимізувати модель онтології (наприклад, покращити її виразність і розширити її властивостями домену). Крім того, варто провести аналіз інших методів для використання результатів виявлення вимог та опису абстрактної інформації (наприклад, програмних цілей).

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1 Baeza-Yates R., Ribeiro-Neto B. Modern Information Retrieval, Addison-Wesley,1999

2 Chen L., Sycara K. Web Mate: A Personal Agent for Browsing and Searching. In proceedings of the 2nd International Conference on Autonomous Agents and Multi Agent Systems, AGENTS '98, ACM, pp. 132 – 139, 1998

3 Chen C., Chen M., Sun Y., PVA: A Self-Adaptive Personal View Agent. Journal of Intelligent Information Systems, p.173-194, 2002

4 Dumais S., Cutrell E., Cadiz JJ., Jancke G., Sarin R., Robbings D., Stuff I've Seen: A System for Personal Information Retrieval and Re-use. In preceedings of SIGIR 2003, Toronto, Canada

5 Frakes W., Baeza-Yates R. Information Retrieval Data Structures & Algorithms. Prentice Hall, 1992

6 Gauch S., Madrid J., Induri S., Ravindran D., and Chadlavada, S., KeyConcept: A Conceptual Search Engine, Information and Telecommunication Technology Center, Technical Report: ITTC-FY2004-TR-8646-37, University of Kansas.

7 Gauch S., Chaffee J., Pretschner A. Ontology Based Personalized Search. Web Intelligence and Agent Systems (in press).

8 Gruber T. Toward principles for the design of ontologies used for knowledge sharing. Presented at the Padua workshop on Formal Ontology, March 1993

9 Kim H., Chan P., Learning Implicit User Interest Hierarchy for Context in Personalization. In Proceedings of the 2003 International Conference on Intelligent user interfaces 2003, Miami, Florida

10 Kurki T., Jokela S., Sulonen R., Turpeinen M. Agents in delivering personalized content based on semantic metadata. In Proceedings 1999 AAAI Spring Symposium Workshop on Intelligent Agents in Cyberspace, Stanford, USA, 1999, pp. 84-93, As cited in [Pretschner99]

11 Lieberman H. Letizia: An Agent That Assists Web Browsing. In Proceedings of the International Joint Conference on Artificial Intelligence, Montreal, CA, 1995

12 Luke S., Spector L., Rager D., Hendler J. Ontology-Based Web Agents. In Proceedings of First International Conference on Autonomous Agents (AutonomousAgents97), W. L. Johnson (ed.), New York: Association for Computing Machinery, pp 5966, 1997.

13 McGowan JP., Kushmerick N., Smyth B.,Who do you want t o be today? Web Personae for personalized information access. In Proc. International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (Malaga), 2002.

14 Middleton S., De Roure D., Shadbolt N. Capturing knowledge of user preferences: ontologies in recommender systems. In Proceedings of the 1 st International Conference on Knowledge Capture (K-Cap2001), Victoria, BC Canada, 2001

15 Middleton S., Alani H., Shadbolt N., De Roure D. Exploiting Synergy Between Ontologies and Recommender Systems. ACM, Semantic Web Workshop 2002 At the Eleventh International World Wide Web Conference Hawaii, USA, 2002 18

16 Mitchell M. An Introduction to Genetic Algorithms. “A Bradford Book”. The MIT Press, 1996. ISBN 0-262-63185-7

17 Mizzaro S., Tasso C., Ephemeral and persistent personalization in adaptive information access to scholarly publications on the Web. Artificial Intelligence Laboratory Department of Mathematics and Computer Science , 2002

18 Mladenic D. Text-learning and related intelligent agents. Revised version In IEEE

19 Expert special issue on Applications of Intelligent Information Retrieval, 1999

20 Mobasher B., Dai H., Luo T., Nakagawa M., Sun Y., Wiltshire J., Discovery of

21 Aggregate Usage Profiles for Web Personalization. In Proceedings of Web Minning for E-commerce Workshop (KDD-2000), Boston, MA 2000 OBIWAN Project. “OBIWAN – University of Kansas”. http://www.ittc.ku.edu/obiwan/. 1999

22 Open Directory Project. http://dmoz.org, April 2002.

23 Pazzani M., Muramatsu, Billsus D. Syskill&Webert: Identifying interesting Web sites. In Proceedings of 13 th National Conference of Artificial Intelligence, Portland, OR, 1996

24 Pretschner A. Ontology based personalized search. Master's thesis, The University of Kansas, Lawrence, KS, 1999

25 Rich E. User modeling via stereotypes. Cognitive Science, 3:329-354, 1979.

26 Salton G. Automatic Text Processing. Addison-Wesley, 1989. ISBN 0-201-12227-8

27 Tanudjaja F., Mui L. Persona: A contextualized and personalized Web search. In Proceedings of the 35 th Annual Hawaii International Conference on System Sciences (HICSS'02), Big Island, Hawaii, 2002, volume3 pp. 53

28 Thomas C., Fischer G. Using Agents to Personalize the Web. In Proceedings of the 2

29 Nd International Conference on Intelligent User Interfaces, Orlando, Florida, USA, 1997, pp.53-60

30 Trajkova, J. Improving Ontology-Based User Profiles, M.S. Thesis, EECS, University of Kansas, August, 2003

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи бакалавра: "Імплементація онтологічних моделей в контексті інженерії вимог"

Обсяг пояснювальної записки: 70 аркушів

Дата закінчення дипломної роботи 15 червня 2022р.

Підпис студента _____