МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 21.00.00.000 ПЗ

Група ІПм-23-3

Решетнік Олексій

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Решетнік Олексій Олександрович

(прізвище, ім'я, по батькові)

УДК 004.94

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі та методи доменно специфікованої мови для застосувань Інтернету речей

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Решетнік О.О.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Піх Володимир Ярославович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

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

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Решетніку Олексію Олександровичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі та методи доменно специфікованої мови для застосувань Інтернету речей”

керівник проекту (роботи) Піх Володимир Ярославович, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних та програмних технологій Інтернету речей

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

1. Аналіз концепцій використання фреймворку авоматизації технологій інтернету речей
2. Підхід локального хмарного середовища автоматизації в IoT
3. Дослідження доменно специфікованої мови в контексті застосувань Інтернету речей
4. Імплементація методів доменно специфікованої мови для застосувань Інтернету речей

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

1. Обмін послугами між постачальником та споживачем (рис. 1.1)
2. Слабка зв’язаність в SOA (рис. 1.2)
3. SoS у сфері охорони здоров’я (рис. 1.3)
4. Локальні хмари, що діють по принципу SoS (System of Systems) (рис. 1.4)
5. Сумісна з Arrowhead Framework система з трьома основними ядрами (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій використання фреймворку авоматизації технологій інтернету речей	29.09.2024	виконано
3	Підхід локального хмарного середовища автоматизації в IoT	15.10.2024	виконано
4	Дослідження доменно специфікованої мови в контексті застосувань Інтернету речей	08.11.2024	виконано
5	Імплементація методів доменно специфікованої мови для застосувань Інтернету речей	20.11.2024	виконано
6	Реалізація функціональності запропонованої інформаційної технології	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 79 с., 27 рис., 2 табл., 50 джерел.

Тема: Моделі та методи доменно специфікованої мови для застосувань Інтернету речей

Об'єкт дослідження: процес моделювання та інтеграції IoT-систем у межах архітектури Arrowhead Framework.

Мета роботи: імплементація доменно-специфічної мови (DSL) для фреймворку, що забезпечить моделювання, автоматизацію генерації коду та інтеграцію з існуючими системами Інтернету речей.

Предмет дослідження: доменно-специфічна мова (DSL) для моделювання та генерації коду в контексті застосування Інтернету речей.

Результати дослідження

В роботі досліджено процеси імплементації доменно-специфічної мови для моделювання в рамках Arrowhead Framework, що інтегрує елементи фізичних та логічних сутностей систем IoT.

Висновок

Результати дослідження можуть бути застосовані для автоматизації розробки IoT-систем на базі Arrowhead Framework. Розроблений DSL та інструмент генерації коду дозволять скоротити час на розробку, підвищити продуктивність та знизити кількість помилок при створенні IoT-рішень.

ІНТЕРНЕТ РЕЧЕЙ, ДОМЕННО-СПЕЦИФІКОВАНА МОВА, МОДЕЛЮВАННЯ СИСТЕМ, ГЕНЕРАЦІЯ КОДУ, АВТОМАТИЗАЦІЯ, ЖИТТЄВИЙ ЦИКЛ, СЕМАНТИКА

ABSTRACT

Master Thesis: 79 pp., 27 fig., 2 tab., 50 sources.

Thesis Subject: Domain-Specific Language Models and Methods for Using the Internet of Things

Research object: the process of modeling and integrating IoT systems within the Arrowhead Framework architecture.

The goal of the work: implementation of a domain-specific language (DSL) for a framework that provides modeling, automation of code generation, and integration with existing Internet of Things systems.

Research subject: domain-specific language (DSL) for modeling and code generation in the context of Internet of Things applications.

Research results

The paper examines the process of implementing a domain-specific language for modeling within the Arrowhead Framework, which integrates elements of the physical and logical entities of the IoT system.

Conclusion

The research results can be applied to automate the development of an IoT system based on the Arrowhead Framework. The developed DSL and code generation tool allows you to reduce development time, increase productivity and reduce the number of errors when creating IoT solutions.

INTERNET OF THINGS, DOMAIN-SPECIFIC LANGUAGE, SYSTEM MODELING, CODE GENERATION, AUTOMATION, LIFE CYCLE, SEMANTICS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CPS - Cyber Physical Systems

IBM - International Business Machines

IoT - Internet of Things

OCL - Object Constraint Language

SoS - System of Systems

VTL - Velocity Template Language

DSL - Domain Specific Language

SOA - Service Oriented Architecture

QoS - Quality of Service

SysML - SystemsModeling Language

BDD - Block Definition Diagram

IBD - Internal Block Diagram

MOF - Meta Object Facility

ВСТУП

Актуальність теми.

З розвитком технологій Інтернету речей (IoT) зростає потреба у впровадженні надійних та ефективних інструментів для моделювання та інтеграції складних систем автоматизації. Arrowhead Framework є одним із ключових підходів до вирішення завдань сумісності та інтеграції IoT-систем, що вимагає розробки доменно-специфічної мови (DSL) для забезпечення ефективної взаємодії між компонентами системи. Створення DSL для Arrowhead Framework є важливим для підвищення продуктивності розробників і автоматизації процесу генерації коду, що робить тему дослідження надзвичайно актуальною.

Розвиток технологій Інтернету речей (IoT) призводить до стрімкого зростання кількості пристроїв, підключених до мережі, та необхідності ефективної інтеграції цих пристроїв у складні автоматизовані системи. Це включає не тільки передачу даних між різними елементами, але й забезпечення безперебійної взаємодії між різними компонентами, які можуть використовувати різні платформи та протоколи. У цих умовах Arrowhead Framework стає важливим інструментом для забезпечення сумісності та ефективної інтеграції між різними IoT-рішеннями. Arrowhead Framework забезпечує стандартизовану архітектуру для управління сервісами в IoT-системах, що значно спрощує процес їх інтеграції та експлуатації.

Однак, сучасні підходи до розробки IoT-систем часто стикаються з труднощами у вигляді складних і не завжди стандартизованих методів розробки, що потребують значних ресурсів і часу. Відсутність уніфікованих інструментів та засобів автоматизації ускладнює процес створення та підтримки таких систем. У цьому контексті розробка доменно-специфічних мов (DSL) для моделювання IoT-систем стає одним із ключових напрямів досліджень, оскільки вони дозволяють спростити процес розробки та підвищити ефективність впровадження нових рішень.

Доменно-специфічні мови (DSL) надають можливість створювати більш точні та адаптивні моделі IoT-систем для конкретних завдань і платформ, забезпечуючи одночасно зручність використання та можливість автоматичної генерації коду для обраних платформ. Для Arrowhead Framework розробка такої мови дозволяє підвищити рівень автоматизації в процесі створення систем, прискорити процес розробки, знизити ймовірність помилок, а також поліпшити інтеграцію з іншими системами та технологіями.

Актуальність дослідження також підкріплюється зростаючим попитом на IoT-рішення в промислових, медичних, транспортних та інших секторах економіки, де необхідна ефективна інтеграція великої кількості різнорідних пристроїв. Створення доменно-специфічної мови для Arrowhead Framework дозволить спростити розробку складних систем IoT, покращити їхню гнучкість і масштабованість, а також забезпечить нові можливості для автоматизації розробки і впровадження рішень у різних галузях.

Таким чином, актуальність теми дослідження полягає в необхідності розробки спеціалізованих інструментів для спрощення моделювання, інтеграції та впровадження IoT-систем, що базуються на Arrowhead Framework, а також у підвищенні ефективності розробки за рахунок автоматизації процесу генерації коду.

Мета дослідження - імплементація доменно-специфічної мови (DSL) для фреймворку, що забезпечить моделювання, автоматизацію генерації коду та інтеграцію з існуючими системами Інтернету речей

Об’єкт дослідження - процес моделювання та інтеграції IoT-систем у межах архітектури Arrowhead Framework

Предмет дослідження - є доменно-специфічна мова (DSL) для моделювання та генерації коду в контексті застосування Інтернету речей

Відповідно до мети роботи було сформовано наступні **задачі:**

- Провести аналіз існуючих підходів до використання Arrowhead Framework у системах автоматизації.
- Дослідити методи створення доменно-специфічних мов та інструментів моделювання.
- Визначити вимоги до розробки DSL для Arrowhead Framework.
- Виконати імплементацію DSL для Arrowhead Framework, що включатиме фізичні та логічні сутності, зв'язки, атрибути і процеси.
- Створити інструмент для автоматичної генерації коду на основі DSL.
- Перевірити розроблену мову та інструмент шляхом генерації і тестування коду у середовищі Arrowhead Framework.

Методи дослідження.

Для досягнення поставлених завдань у дослідженні використовувалися наступні методи:

- Огляд і аналіз літератури з доменно-специфічних мов, IoT та Arrowhead Framework.

- Моделювання і розробка DSL.

- Генерація коду з використанням шаблонних мов.

- Верифікація створеної моделі за допомогою обмежень OCL та тестування в Arrowhead Framework.

Наукова новизна отриманих результатів

Досліджено процеси імплементації доменно-специфічної мови для моделювання в рамках Arrowhead Framework, що інтегрує елементи фізичних та логічних сутностей систем IoT. Крім того, розроблено механізм автоматичної генерації коду з використанням цієї мови, що підвищує ефективність розробки IoT-рішень.

Практичне значення магістерської роботи

Результати дослідження можуть бути застосовані для автоматизації розробки IoT-систем на базі Arrowhead Framework. Розроблений DSL та інструмент генерації коду дозволять скоротити час на розробку, підвищити продуктивність та знизити кількість помилок при створенні IoT-рішень.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 79 сторінок, і містить 27 рисунків, 2 таблиці, список використаних джерел із 50 найменувань.

РОЗДІЛ 1. АНАЛІЗ КОНЦЕПЦІЙ ВИКОРИСТАННЯ ФРЕЙМВОРКУ АВОМАТИЗАЦІЇ ТЕХНОЛОГІЙ ІНТЕРНЕТУ РЕЧЕЙ

1.1. Опис предмету дослідження

Arrowhead Framework є платформою автоматизації IoT, яка сприяє взаємодії та інтеграції Інтернету речей (IoT), абстрагованих у системи та послуги. Архітектура Arrowhead Framework складається з локальних хмар автоматизації, що включають основні послуги, які використовуються застосуванням SoS [4]. Вона спрямована на спільну автоматизацію та промисловий IoT шляхом побудови систем систем (SoS) з компонентів IoT та забезпечення взаємодії між ними [2].

Ця платформа також допомагає створювати складні взаємодіючі системи IoT. Існує потреба в тому, щоб ці складні критичні системи в Arrowhead Framework були змодельовані та валідовані перед фактичним розвитком цих систем, щоб уникнути помилок або вразливостей. Мова специфічної предметної області (DSL) є типом формальної мови, що складається з елементів певної предметної області. DSL спрощує процес моделювання, використовуючи елементи специфічної предметної області, а не елементи загального призначення мови моделювання.

Необхідно створити DSL для Arrowhead Framework, що охоплює всю його структуру та елементи в цілому. Цей встановлений профіль, що складається з конструкцій Arrowhead Framework, слугуватиме основою для інженерії систем на основі моделей (MBSE), переходячи від обміну інформацією на основі документів до ефективного обміну інформацією на основі моделей. Насправді будь-яка складна система автоматизації IoT в Arrowhead Framework може бути змодельована з кількома поглядами, що охоплюють вимоги, поведінку, структуру та параметри системи. Інструмент для створення DSL вибирається на основі вимог до процесу створення та валідації DSL. Наприкінці процесу вибору, коли було оцінено кілька

інструментів, Cameo Systems Modeler було обрано як відповідний інструмент для цього завдання. DSL для Arrowhead Framework потім створюється методом розширення UML за допомогою профілів, що складаються зі стереотипів та позначених значень. Створюється повний DSL, що складається з основних елементів платформи та різних систем і послуг у платформі.

Цей DSL перевіряється на відповідність формі за допомогою обмежень Object Constraint Language (OCL). За допомогою цієї платформи будується модель, і для цієї конкретної моделі генерується код для валідації. Згенерований код потім перевіряється в середовищі Arrowhead Framework, що в свою чергу перевіряє створений DSL. Насправді цей створений DSL покращує комунікацію та розуміння, що в свою чергу економить час і підвищує продуктивність.

«Індустрія 4.0» — це індустріальний зрушення, що зараз розвивається, яке зосереджується на автоматизації традиційних промислових систем і налагодженні взаємодії між пристроями на нижчих рівнях підприємства зі складними корпоративними системами вищого рівня. Такі технології, як IoT і CPS, повсюдно використовуються в Industry 4.0 для підвищення гнучкості. Стандартні протоколи, які використовуються для зв’язку в Industry 4.0, відкривають шлях для встановлення взаємодії між, можливо, будь-якими системами чи пристроями на виробництві. Незважаючи на те, що це забезпечує кілька переваг, це створює проблеми з точки зору безпеки та надійності. Щоб вирішити ці проблеми та покращити масштабованість, гнучкість виробництва та автоматизацію , було запропоновано поняття Arrowhead Framework.

Arrowhead Framework — це спільний дослідницький проект, над яким працюють кілька європейських компаній. Ця структура є стандартизацією, що зараз розвивається та розвивається, спрямована на взаємодію та створення промисловості 4.0 у системах промислової автоматизації. Це структура автоматизації IoT, яка допомагає у взаємодії та інтеграції IoT,

абстрагованих у системи та служби [2]. Основна мета Arrow head Framework полягає в тому, щоб надати можливість усім користувачам працювати єдиним чином, що забезпечує високий рівень сумісності. Крім того, він націлений на спільну автоматизацію та промисловий IoT шляхом побудови SoS з компонентів IoT і забезпечення взаємодії між ними [2].

Незважаючи на те, що Arrowhead Framework забезпечує спосіб створення складних сумісних систем автоматизації, ці системи та їх відповідна взаємодія мають бути безпечними та надійними. Для забезпечення надійності та сумісності ці системи мають бути змодельовані та перевірені перед фактичним впровадженням. Щоб змоделювати ці складні системи автоматизації в Arrowhead Frame для роботи зі специфічними конструкціями Framework, необхідно використовувати DSL. Використання DSL допомогло б у стислому вираженні моделей, використовуючи специфічні для домену концепції, що, у свою чергу, призводить до кращого розуміння та покращення комунікації.

Крім того, це підвищує продуктивність, якість і надійність моделей. IncQuery Labs створила базовий профіль для Arrowhead Framework для вирішення цих проблем [17]. Однак він не охоплює всі аспекти цього Framework, що постійно розвивається. Крім того, моделі, створені з використанням конкретних елементів домену, повинні бути перевірені за допомогою інструментів, специфічних для цього домену. Таким чином, потрібен DSL для Arrowhead Framework, який охоплює всі концепції та елементи цієї Framework. Крім того, моделі, створені з цього DSL, також повинні бути перевірені в середовищі Arrowhead Framework.

Хоча базовий профіль SysML або DSL для Arrowhead Framework існує, він не охоплює всі аспекти цього фреймворку, що постійно розвивається. Слід створити повний профіль SysML або DSL, який охоплює структуру систем і служб Arrowhead Framework, а також взаємодію між ними. Профіль стане основою для моделювання складних систем IoT, великих автоматизованих систем та інших систем на основі Arrowhead Framework.

Крім того, сама по собі Arrowhead Framework потребує достатньо часу та зусиль для розуміння. Тим, хто має обмежені знання про мови моделювання, було б простіше легко моделювати складні системи Інтернету речей у цьому фреймворку за допомогою цих специфічних конструкцій. Крім того, на початку цього завдання не було жодних існуючих методів перевірки цих моделей у Arrowhead Framework. Це додатково створює потребу в генерації спеціального коду Arrowhead Framework з цих моделей, який можна перевірити в середовищі Arrowhead Framework. Таким чином, по суті, повний робочий процес створення DSL для Arrowhead Framework, побудови моделі з використанням елементів із цього DSL, генерації спеціального коду Arrowhead Framework із цієї побудованої моделі та перевірки цієї моделі шляхом перевірки коду, згенерованого з неї, у середовищі Arrowhead Framework , необхідно завершити.

Виявлення прогалин у поточній установці є першим кроком цілісної реалізації установки, яка допоможе досягти мети. Це передбачає формулювання дослідницьких питань і пошук різних підходів до їх вирішення.

1. Чи має SysML достатньо семантики для створення DSL для -роботи Arrowhead Frame?

2. Як створити профіль SysML для Arrowhead Framework? Які елементи Framework можна профілювати, а які ні?

Метою цього проекту є вдосконалення існуючого профілю Arrowhead Framework шляхом розширення його до нового повного DSL для Framework. Створений DSL забезпечить спосіб моделювання складних систем IoT у Framework, що, у свою чергу, допоможе подолати розрив у комунікації, обміні знаннями та розумінні.

З використанням цього DSL було створено модель, щоб зрозуміти інтуїтивність систем моделювання у Framework за допомогою цього DSL.

1.2. Дослідження архітектури фреймворку автоматизації систем Інтернету речей

Першим кроком для створення доменно-специфічної мови для фреймворку є повне розуміння її архітектури. У цьому розділі повністю розглядається архітектура Arrowhead Framework. У ньому надається детальна інформація про різні концепції, на основі яких було створено Framework.

Архітектура Arrowhead Framework складається з систем автоматизації на основі Інтернету речей, яка наголошує на:

• сумісності широкого спектру IoT і застарілих систем.

• обробці даних у режимі реального часу для швидшого зв'язку та керування обчисленнями.

• забезпеченні кращого захисту систем і даних.

• інженерії систем автоматизації.

• масштабованості систем автоматизації, що дозволяє працювати з великими інтегрованими системами автоматизації.

Цей фреймворк базується на фреймворку Spring-Boots у веб-сервісах Java та REST [16]. Крім того, проект Arrowhead Framework пропонує клієнтський скелетний код і приклади на різних мовах програмування, таких як Java, Python, C++ і NodeJs [14]. Останнім доповненням до цього списку є Kalix, яка є бібліотекою Java для розробки систем eclipse arrowhead [24].

Архітектура Arrowhead Framework заснована на парадигмі SOA. SOA була визначена Джеймсом Біном [25] як: «Сервісно-орієнтована архітектура (SOA) — це комбінація споживачів і служб, які співпрацюють, підтримується керованим набором можливостей, керується принципами та регулюється допоміжними стандартами».

Обмін інформацією між системою-провайдером і системою-споживачем представлений сутністю, яка називається сервісом [6]. Сервіси є основними блоками SOA [6]. Цей обмін послугами між системою постачальника та системою споживача зображено на рисунку 1.1. Послуга

провайдер реєструє свою послугу в системі реєстру. Зареєстровану послугу виявляє споживач послуги за допомогою процесу виявлення послуги. Ця виявлена послуга потім споживається споживачем послуги.

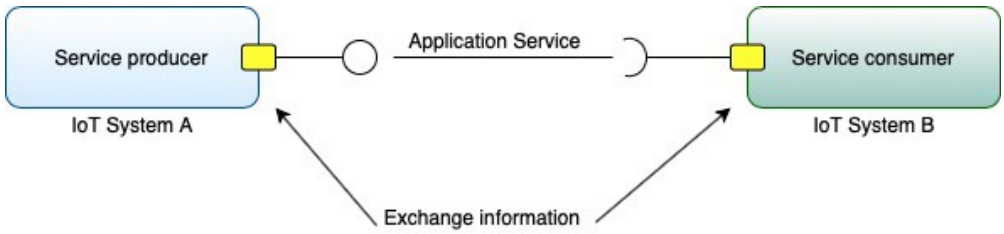

Рис. 1.1. Обмін послугами між постачальником та споживачем

Архітектура Arrowhead Framework дотримується основних властивостей SOA, таких як:

1. Слабка зв’язаність.

Сервіси розподілені між кількома системами та пристроями. Для того, щоб відбувся обмін даними під час виконання, системі не обов’язково знати іншу систему під час проектування. Підтримуються реєстр послуг і механізми виявлення, коли нова послуга реєструється в реєстрі послуг самостійно. Після реєстрації цей сервіс стає доступним для інших систем. Система, якій потрібно споживати цю послугу, може використовувати механізм виявлення служби, щоб споживати її [26]. Цей слабкий зв’язок між системами представлений на рисунку 1.2 [4].

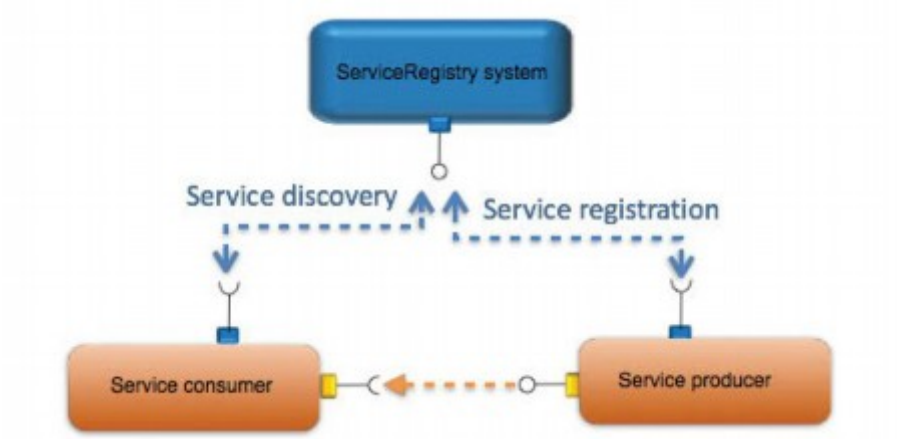

Рис. 1.2. Слабка зв’язаність в SOA

2. Пізня зв'язність.

Система може підключатися до інших систем у будь-який час, коли їй потрібні дані, тобто обмін даними між двома системами відбувається під час виконання. Для встановлення цієї пізньої зв'язності між системами можна використовувати метод оркестрації взаємодії [4].

3. Автономія.

Системи володіють своїми даними і відповідають за прийняття рішень про те, з якими системами вони можуть обмінюватися даними. Кожна система може діяти самостійно, незалежно від інших систем. Більше того, у разі встановлення обміну послугами між двома системами, цей обмін буде неконтрольованим, тобто не потрібно буде втручання жодних інших підтримувальних послуг/функціональностей [4].

4. Поведінка "push" і "pull":

Обмін послугами може бути ініційований або системою, що споживає послугу, або системою, що виробляє послугу. Поведінка "pull" виникає, коли обмін послугами ініціюється споживачем послуги, а поведінка "push" виникає, коли обмін послугами ініціюється виробником послуги [4].

1.3.Підхід локального хмарного середовища автоматизації в IoT

Як згадувалося раніше, архітектура Arrowhead Framework складається з SoS (System of Systems) автоматизації на основі IoT. SoS можна визначити як велику мережу складних і територіально розподілених систем, що функціонують разом для досягнення спільної мети, де ці складові системи є незалежними та значущими самі по собі [27]. Приклад SoS у сфері охорони здоров’я показано на рисунку 1.3 для подальшого розуміння. На цьому рисунку 1.3, різні системи, такі як система керування пацієнтами, лабораторна система, система керування зображеннями, телеметрична система та аптечна система, виконують різні власні функції. Ці системи

співпрацюють разом, щоб сформувати систему охорони здоров'я, яка має певну мету виконувати функції, пов'язані з охороною здоров'я.

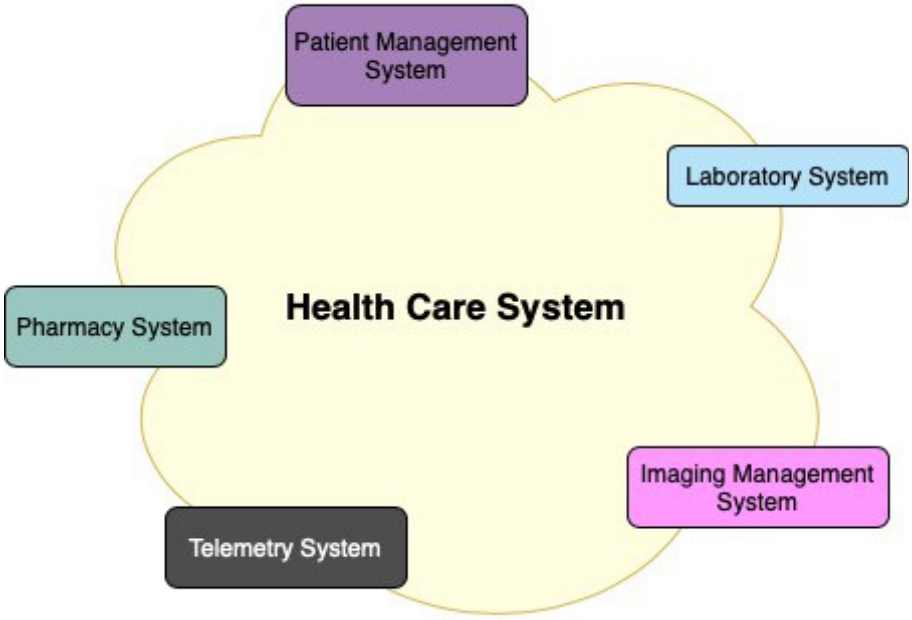

Рис. 1.3. SoS у сфері охорони здоров'я

Існує п'ять основних характеристик SoS, які допомагають визначити його [27]:

1. Операційна незалежність компонентів.

У випадку SoS, що складається з систем компонентів, які розбираються, кожна з цих систем компонентів повинна мати здатність працювати автономно для корисної мети. По суті, це гарантує, що SoS складається з компонентних систем, які є повністю автономними та корисними самі по собі [27].

2. Управлінська незалежність елементів.

Однією з необхідних умов є те, що складові системи повинні працювати незалежно. Незважаючи на те, що здійснюється окреме придбання та інтеграція систем компонентів, вони все ще підтримують безперервне функціонування, яке не залежить від SoS [27].

3. Еволюційний розвиток.

SoS може здатися не повністю сформованим, і його існування та розвиток є прогресивним, де додавання, видалення та модифікація цілей і функцій здійснюється з плином часу [27].

Кілька форм хмарних обчислень стають помітними в галузі. Відкрита глобальна Інтернет-хмара має кілька обмежень і тому не підходить для цілей автоматизації. Завдання автоматизації мають жорсткі вимоги до безпеки, операцій, зв'язку та обчислень у реальному часі, технічного обслуговування тощо. Як наслідок, було розроблено поняття локальних хмар, яке відповідало б вимогам вищевказаного завдання автоматизації. Ця локальна хмара формується шляхом включення пристроїв і систем зі службами, які необхідні для виконання необхідних завдань автоматизації. По суті, ця локальна хмара утворює межу, що містить необхідні системи, і ця межа захищає системи всередині від будь-якого вхідного чи вихідного зв'язку через зовнішній відкритий Інтернет [28].

Локальні хмари в архітектурі Arrowhead Framework зображені на рисунку 1.4. Ці локальні хмари складаються із систем, що надають і споживають основні послуги, які співпрацюють як SoS. По суті, локальна хмара діє як SoS, де набір систем співпрацює між собою для створення набагато складнішої системи автоматизації. Крім того, характеристики локальної хмари подібні до вищезгаданих характеристик SoS. Таким чином, локальна хмара сама по собі діє як SoS [3].

Архітектура Arrowhead Framework складається з локальних хмар автоматизації, які гарантують певні функції, пов'язані з глобальною хмарою, наприклад:

• інфраструктуру хмарного сервісу.

• послуги з підтримки автоматизації.

• властивості затримки можна гарантувати або передбачити дуже добре.

• основні властивості безпеки, такі як авторизація та автентифікація.

• масштабованість систем автоматизації, що веде до великих інтегрованих комплексних систем автоматизації.

• простота розробки додатків.

• інтеграція багатьох зацікавлених сторін.

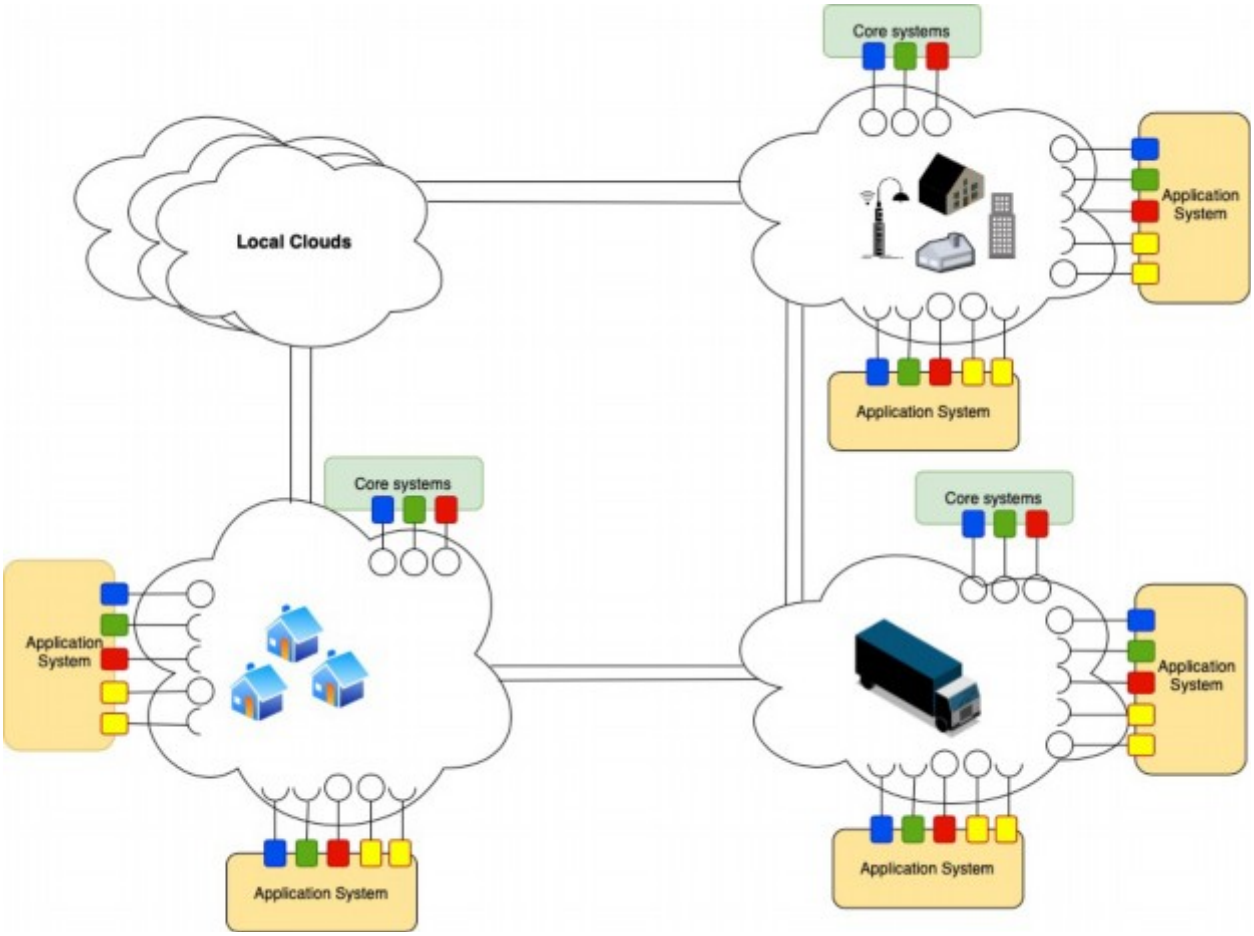

Рис. 1.4. Локальні хмари, що діють по принципу SoS (System of Systems) [4]

У деяких випадках під час створення великих систем автоматизації, якщо необхідна інформація недоступна в конкретній локальній хмарі, може виникнути безпечна взаємодія між хмарами. У цій взаємодії одна локальна хмара спілкується з іншими локальними хмарами, які дотримуються вказівок Arrowhead Framework для обміну інформацією. Взаємодія між хмарами допомагає створювати складні масштабовані системи автоматизації.

Основними властивостями базової локальної хмари автоматизації в Arrowhead Framework є:

1. Самодостатність

Локальна хмара охоплює ресурси, необхідні для встановлення, що забезпечує автономну локальну роботу. Це усуває залежність від інших зовнішніх ресурсів [28]. Основні зовнішні ресурси, необхідні для створення локальної хмари:

• Реєстри. Допомога у збереженні записів про пристрої, системи та служби, розгорнуті в локальній хмарі. Крім того, це встановлює можливість виявлення вищезазначеного. У локальній хмарі Arrowhead Framework потрібні три основні реєстри:

- Реєстр послуг.
- Системний реєстр.
- Реєстр пристроїв.

• Оркестровка служб, де центральний об’єкт може координувати взаємодію між різними службами.

• Автентифікація послуги та авторизація для безпечної перевірки споживача послуги/об’єкта та безпечного контролю доступу/споживання послуги відповідно.

2. Підтримка вимог автоматизації

• Підтримка проектування, налаштування, розгортання, експлуатації та обслуговування систем автоматизації.

• Увімкнення обміну послугами на основі подій.

• Включення аудиту сервісного обміну.

• Підтримка зв'язку QoS.

• Зберігання аудиторської інформації та історичних даних [28].

3. Забезпечте захист зовнішніх мереж

Наявність необхідних ресурсів, що містяться в локальній хмарі, усуває необхідність зовнішнього зв’язку з іншими зовнішніми ресурсами. Отже, фізичний комунікаційний рівень не пов’язаний з жодними іншими зовнішніми мережами [28].

4. Безпечне завантаження та оновлення програмного забезпечення.

Розгортання та оновлення пристроїв, систем і послуг має бути безпечним.

5. Підтримка метаданих пристроїв, систем і послуг

6. Підтримка прозорості протоколу та семантики

Сумісність систем IoT вимагає протокольної та семантичної прозорості, що призводить до масштабованості [28].

7. Підтримка безпечного адміністрування та обміну даними із зовнішніми ресурсами

У разі виникнення потреби в обміні між хмарними службами має бути можливим вищезазначене адміністрування обміну послугами, наприклад виявлення служби, автентифікація служби, авторизація служби та оркестровка служби. Крім того, обмінювані дані корисного навантаження служби повинні мати наскрізне шифрування [28].

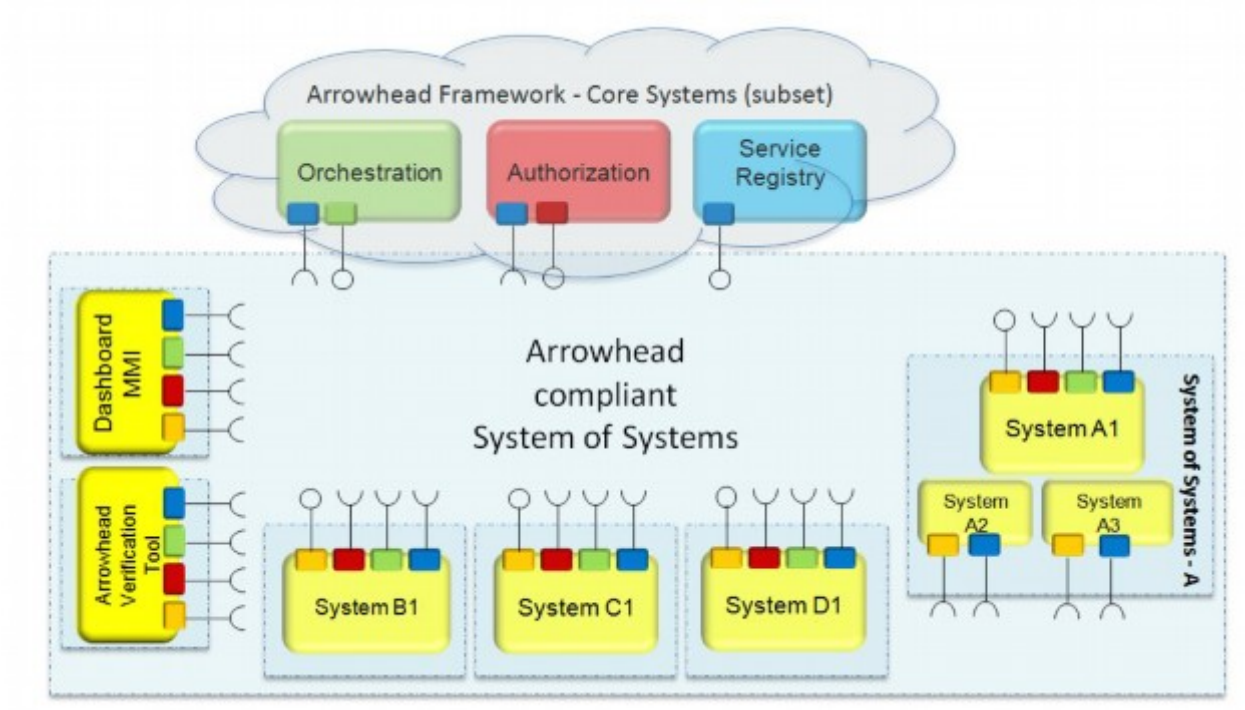

Рис. 1.5. Сумісна з Arrowhead Framework система з трьома основними ядрами

Інтеграція вищезазначених властивостей SOA з цими властивостями локальної хмари забезпечує необхідність певних важливих аспектів у локальній хмарі. Ці аспекти включають реєстрацію систем і послуг у хмарі,

виявлення зареєстрованих послуг, авторизацію систем перед реєстрацією їхніх послуг та організацію обміну послугами між виробником послуг і споживачем послуг. Зрештою це призвело до формування первинних систем, необхідних для формування локальної хмари Arrowhead Framework [4]. Три основні системи ядра в Arrowhead Framework — це система реєстру послуг, система авторизації та система оркестровки. Локальна хмара Arrowhead Framework, яка діє як SoS з трьома основними системами ядра, показана на рисунку 1.5.

1.4. Дослідження структури та моделі фреймворку автоматизації IoT

Поки визначаються основні системи, необхідні для створення локальної хмари, потрібен набір суворих принципів проектування для Arrowhead Framework. Це забезпечить встановлення чіткого розмежування між системами, сумісними з Arrowh ead Framework, та іншими системами. Крім того, це допомагає краще зрозуміти структуру для дизайнерів і розробників, що допоможе їм у розробці, розгортанні, підтримці та управлінні цими системами.

Дизайн Arrowhead Framework базується на наступному [4]:

1. Arrowhead Framework дотримується вищезазначених властивостей SOA.
2. Сумісна з інфраструктурою SoS відповідає вищезазначеним характеристикам SoS.
3. Локальна хмара стає сумісною з Arrowhead Framework, лише якщо вона відповідає вищезазначеним основним властивостям локальної хмари автоматизації.
4. Фреймворк надає можливість динамічного створення та нових послуг, споживання цих новостворених і видалення існуючих послуг.
5. Крім того, вона підтримує як підхід "push", так і підхід "pull".

6. Крім того, він підтримує підхід публікації та підписки.

7. Система, що створює послугу, має початкові повноваження своєї власної служби.

8. Тільки на рівні сервісного обміну інформація гарантується.

9. Мережа в структурі більше орієнтована на інформацію.

Наведені вище вимоги до дизайну Arrowhead Framework задовольняються набором із трьох основних обов'язкових послуг, які допомагають у формуванні базової SoS та сукупності служб підтримки автоматизації. Таким чином, структура складається з двох наборів служб, а саме, основних служб і служб додатків. Локальні хмари в цій структурі керуються деякими основними службами. Основні послуги далі поділяються на дві, а саме обов'язкові основні послуги та основні послуги підтримки автоматизації [4]. Три обов'язкові основні послуги: послуги реєстру послуг, послуги авторизації та послуги оркестровки.

1.4.1. Структура фрейворку

Основна структура Arrowhead Framework складається з локальної хмари, яка діє як SoS. Ця локальна хмара Arrowhead Framework складається з трьох основних будівельних блоків, а саме:

1. Пристрої. Будь-яка сутність, яка є апаратним забезпеченням або машиною, яка має можливість обчислювати, обмінюватися даними та зберігати дані, а також розміщувати одну чи декілька систем, сумісних із Arrowhead Framework разом із їхніми службами, є пристроєм Arrowhead Framework [4].

2. Системи. Система в Arrowhead Framework — це об'єкт у локальній хмарі, який реєструється в обов'язковому системному реєстрі та здатний виробляти або споживати послуги, сумісні з Arrowhead Framework, відмовляючись від своїх повноважень над послугами, що надаються нею [4].

3. Послуги. Обмін інформацією між виробником і споживачем представлений сутністю, що називається Сервісом [31]. Його можна розуміти

як програмний блок, який є логічним відображенням конкретного завдання або містить певну інформацію [30].

Будь-яка система в цій локальній хмарі може бути розміщена на апаратному пристрої. Система в структурі або надає, або споживає одну чи більше послуг, або вона може як надавати, так і споживати послуги [4].

Arrowhead Framework надає набір основних систем, де обов’язкові базові системи забезпечують незамінні послуги в локальній хмарі, а базові системи підтримки допомагають у підтримці автоматизації. Щоб побудувати масштабовані комплексні системи автоматизації з кращою безпекою, можливістю взаємодії між несумісними системами, обміном послугами між різними локальними хмарами та кращим початковим завантаженням, ці системи були представлені в Arrowhead Framework [4]. Зараз зростає кількість цих базових систем підтримки, які розробляються. Перелічені нижче основні системи підтримки або зараз розробляються, або вже розроблені [4, 32]:

- Система реєстру послуг (обов’язкова основна система)
- Система авторизації (обов'язкова основна система)
- Система оркестровки (обов'язкова основна система)
- Система системного реєстру
- Система реєстрації пристроїв
- Система обробки подій
- Система Gatekeeper
- Система шлюзів
- Система керування QoS
- Система хореографа
- Система бортового контролера
- Система перекладу

Ці базові системи разом із прикладними системами співпрацюють одна з одною для створення складної системи автоматизації, утворюючи SoS. Мінімальною вимогою для функціонування локальної хмари автоматизації є

взаємодія трьох обов’язкових основних систем із прикладними системами, як показано на рисунку 1.6 [4].

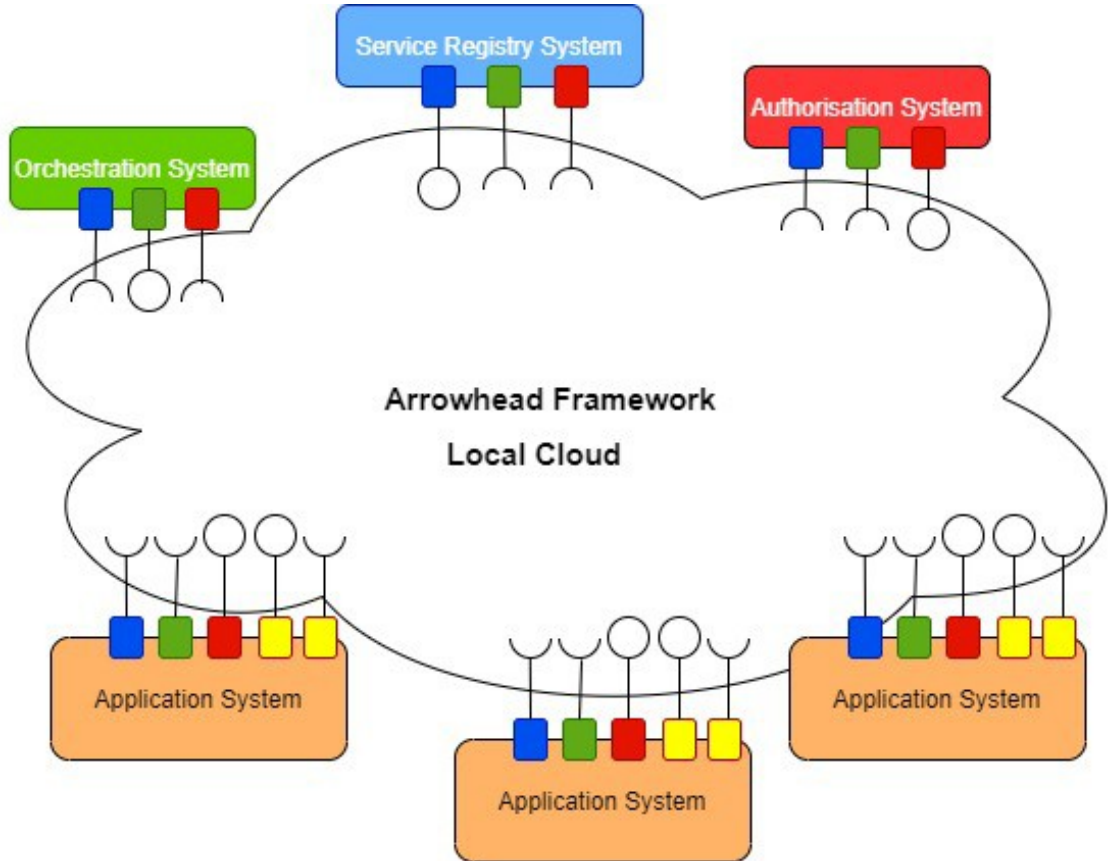

Рис. 1.6. Локальна хмара, що функціонує з мінімальним набором трьох обов'язкових базових систем

1.4.2. Модель документації

Модель документації Arrowhead Framework надає інформацію про те, як слід описувати SoS, системи та служби. Він складається з документації рівня SoS, документації рівня системи та документації рівня обслуговування [2]. Документація рівня обслуговування складається з чотирьох документів, а саме опису служби (SD), опису дизайну інтерфейсу (IDD), профілю зв’язку (CP) і семантичного профілю (SP) [4]. Зв’язок між цими документами Arrowhead Framework зображено на рисунку 1.7 [4]. Крім того, зв’язок між частиною документів документації рівня обслуговування та їх залежністю від технологічної інформації зображено на рисунку 1.8.

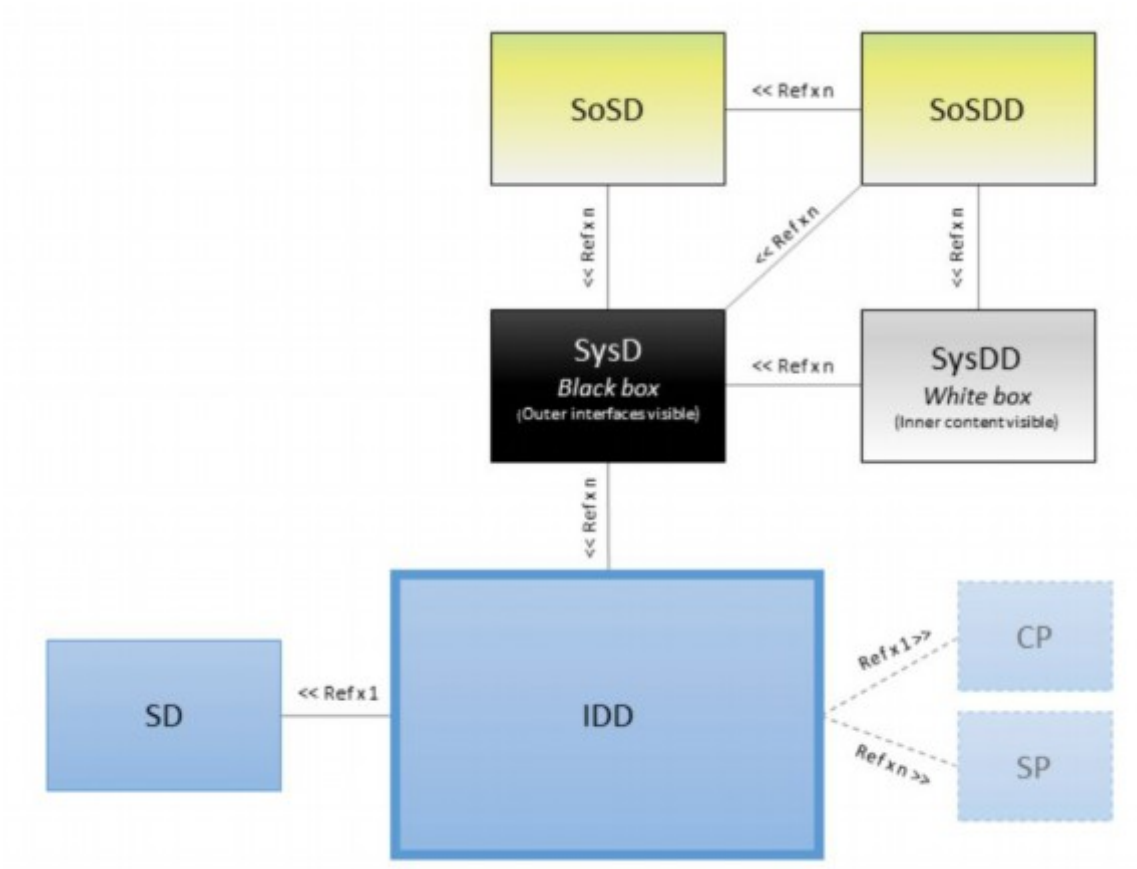

Рис. 1.7. Взаємозв’язок між специфікаціями Arrowhead Framework

Рис. 1.8. Взаємозв'язки між документами в документації рівня послуг

Arrowhead Framework — це стандартизаційна та дослідницька робота, що зараз розвивається та розвивається, спрямована на взаємодію та створення систем промислової автоматизації Industry 4.0. За словами

команди розробників Arrowhead Framework, для повної стандартизації цього фреймворку ще потрібно виконати певну роботу.

Висновки до розділу

У цьому розділі було розглянуто основні концепції, на яких базується Arrowhead Framework. Обговорювалися ключові принципи, такі як сервіс-орієнтована архітектура (SOA) та система систем (SoS), які лежать в основі цього фреймворку. Окрему увагу приділено підходу до реалізації локальної хмари в Arrowhead Framework та її перевагам для інтеграції систем. Крім того, розглядалися принципи проєктування, які використовуються для створення систем у межах цього фреймворку. Описано також обов’язкові та допоміжні основні системи, які забезпечують роботу та гнучкість Arrowhead Framework.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ПРОЦЕСІВ МОДЕЛЮВАННЯ ТА ДОМЕННО СПЕЦИФІКОВАНОЇ МОВИ В КОНТЕКСТІ ЗАСТОСУВАНЬ ІНТЕРНЕТУ РЕЧЕЙ

2.1. Опис особливостей мови побудови візуальних моделей

Systems Modeling Language (SysML) — це мова графічного моделювання, яка використовується для цілей системної інженерії або спеціальна мова для створення візуальних моделей складних систем [7]. Це допомагає в проектуванні, специфікації, аналізі, верифікації та перевірці широкого спектру систем і SoS [7]. Дизайн SysML гарантує, що потужні конструкції, які не є громіздкими, можуть бути використані для моделювання широкого діапазону додатків системної інженерії [21]. У цьому розділі описується початок SysML, його взаємозв'язок з UML, набір діаграм, які він надає, і його корисність. Крім того, обговорюється методологія SYSMOD. Крім того, у цьому розділі розглядається концепція DSL і необхідність DSL для Arrowhead Framework.

SysML був спочатку використовувалась як налаштування UML для цілей системної інженерії [23]. UML — це мова моделювання розвитку загального призначення, що використовується в інженерії програмного забезпечення, яка допомагає специфікувати, візуалізувати та документувати моделі разом із їхньою структурою та дизайном, особливо для програмних систем. UML — це мова моделювання, яка використовується переважно в області розробки програмного забезпечення [22]. Потреба в мові моделювання загального призначення в галузі системної інженерії зі спеціальними конструкціями призвела до появи SysML. І UML, і SysML були прийняті як стандарт Object Management Group (OMG) [23]. SysML було опубліковано як стандарт ISO/IEC 19514:2017 (інформаційна технологія – Мова моделювання груп систем управління об’єктами) [9]. SysML був

прийнятий OMG у 2006 році, і з цього року він був опублікований як OMG SysML [2].

2.1.1. Дослідження взаємозв'язку між SysML і UML

Як згадувалося раніше, UML — це мова моделювання, яка використовується переважно в області розробки програмного забезпечення. Важливою особливістю цієї мови є те, що вона надає варіанти її розширення для цілей, специфічних для домену. Згодом SysML було розроблено шляхом розширення UML для цілей системної інженерії. SysML використовує підмножину UML з власним набором розширень [8].

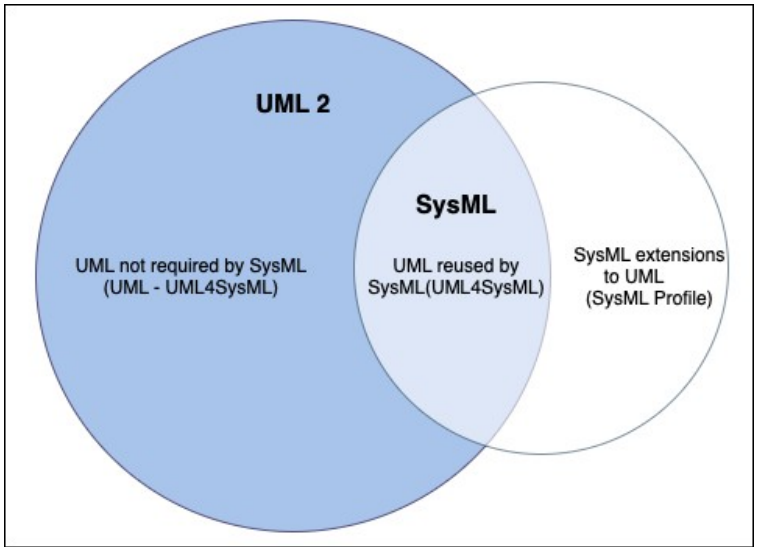

Рис. 2.1. Представлення взаємозв'язку між SysML і UML

Цей зв'язок між UML і SysML можна візуалізувати на діаграмі Венна (рис. 2.1. Ця діаграма складається з трьох частин. Розділ, виділений синім кольором із текстом «UML не вимагається SysML», є першою частиною. Це частина UML, яка не потрібна для реалізації SysML. Другою частиною є розділ, створений перетином двох кіл і виділений обома кольорами з текстом «UML повторно використаний SysML». Це частина UML, необхідна для реалізації SysML, і вона ілюструє конструкції моделювання UML, які повторно використовуються SysML. Розділ, виділений білим кольором з

текстом «SysML extensions to UML», є третьою частиною. Це частина SysML, яка представляє нові конструкції моделювання, визначені для SysML, які не мають еквівалентних конструкцій в UML або тих, що замінюють конструкції UML [21].

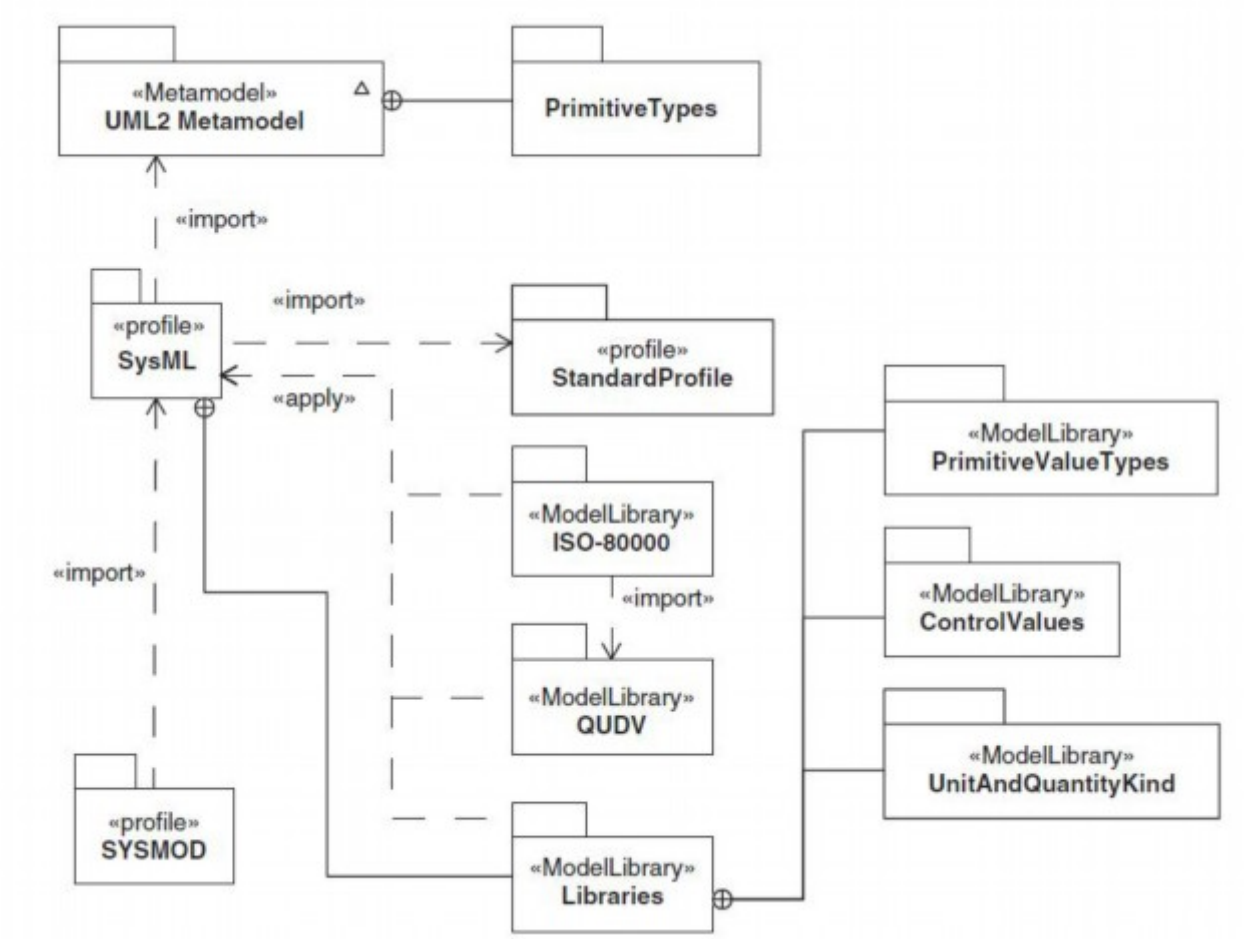

Рис. 2.2. Пакети архітектури SysML

Крім того, рис 2.2 показує зв'язок між SysML і UML разом з іншими пакетами моделювання. Як показано на цій діаграмі, метамодель UML імпортується SysML. SysML використовує стереотипи, які розширюють необхідні елементи метамоделі UML. Крім того, SysML також використовує інші бібліотеки моделей, зображені на цій діаграмі, які надають деякі елементи, необхідні для одиниць, розмірів, кількостей, значень, типів значень тощо [36].

2.1.2. Опис діаграм побудованих засобами SysML

Нижче наведено дев’ять різних типів діаграм SysML. Ці дев'ять діаграм представляють різні аспекти системи, тобто структуру, вимоги та поведінку:

1. Діаграма визначення блоку (BDD)

BDD є основною діаграмою в SysML, яка допомагає передати структурну структуру системи. Крім того, це допомагає виразити обидва можливі типи структур, які можуть існувати, а саме структуру, яка існує всередині системи, і зовнішню структуру в середовищі системи. Крім того, він надає варіанти для вираження типів обмежень, яким має відповідати кожна структура, і типів значень, які також можна використовувати в операційній системі. Блок — це основна структурна одиниця в SysML, яка використовується в BDD. Для представлення зв’язку між різними блоками можна використовувати декілька типів зв’язків. Крім того, обмін послугами, даними, енергією, речовиною тощо між блоками може бути представлений за допомогою портів [7].

2. Внутрішня блок-схема (IBD)

IBD допомагає представити внутрішню структуру блоку з точки зору конкретних частин, які існують у побудованій системі, її властивостей і з’єднувачів між цими частинами. Одним із важливих аспектів цієї діаграми є те, що інформація, яка передається в IBD, доповнює інформацію в BDD, і фактично ці дві діаграми створюються одночасно.

3. Діаграма варіантів використання

Погляд з висоти пташиного польоту на певну поведінку системи, яку можна спостерігати ззовні системи як одиниці істотної роботи, представляє варіант використання. Діаграма варіантів використання допомагає визначати та переглядати варіанти використання та їхній зв’язок із тими учасниками, які взаємодіють із нею та отримують певну цінність від системи. Він надає чорну скриньку для перегляду послуг, що надаються системою, і взаємодії з акторами, які отримують цю послугу. По суті, він діє як контекстна діаграма

системи, що забезпечує уявлення про систему, необхідне зацікавленим сторонам на ранньому життєвому циклі.

4. Діаграма діяльності

Діаграма активності SysML є розширенням діаграми активності UML. Це допомагає передавати зацікавленим сторонам поведінку системи з часом. Він використовує послідовність дій для представлення поведінки блоку або будь-якої іншої структури. Крім -того, наголос цієї діаграми полягає в передачі безперервної системи поведінки та потоку елементів. Потік предметів, який можна представити, включає інформацію, дані, енергію, послуги або будь-яку іншу фізичну сутність, яку можна виробити, спожити або навіть передати. Крім того, це допомагає представити як послідовний, так і одночасний потік.

5. Діаграма послідовності

Подібно до діаграми послідовності UML, діаграма послідовності SysML також представляє інформацію про поведінку системи з часом, зосереджуючись більше на повідомленнях зв’язку між різними частинами всередині системи. Крім того, це допомагає передати повну та однозначну поведінку системи, наприклад, яка частина системи виконує або викликає кожну поведінку в системі та порядок виникнення дій. Це ефективно допомагає використовувати дану діаграму на етапі розробки життєвого циклу системи.

6. Діаграма державного автомата

Діаграма кінцевого автомата допомагає представити складну поведінку системи. Зокрема, це допомагає передати інформацію про весь хід системного об’єкта, який має кілька можливих життєвих циклів, які важко зрозуміти. Крім того, це допомагає представити різні стадії або стани, через які сутність може пройти протягом усього свого курсу або життєвих циклів. Крім того, це допомагає у представленні умов і застережень лише за яких об’єкт переходить з одного стану в інший.

7. Параметрична діаграма

Параметрична діаграма допомагає створити математичну модель системи, яка складається з набору обмежень, переважно набору математичних рівнянь і нерівностей. Це допомагає визначити ті значення параметрів, які дійсні в мінімальній операційній системі. Крім того, можна провести аналіз критичних параметрів системи з інженерної точки зору. Це включає оцінку важливих показників, таких як надійність системи, продуктивність та інші. Цілісно ця діаграма слугує мостом між вимогами та проектом системи, допомагаючи комбінувати ці моделі, фіксуючи обмеження, засновані на складних математичних співвідношеннях.

8. Діаграма упаковки

Пакетна діаграма надає положення для ілюстрації складної організації моделі в прості зрозумілі репозиторії. Це допомагає розробнику моделей передати логічне угруповання складних структур моделі, що, врешті -решт, допомагає зацікавленим сторонам орієнтуватися в структурі моделі, щоб знайти елементи на детальному рівні. Це групування високого рівня може базуватися на кількох критеріях, таких як легкість навігації, контроль доступу, керування конфігурацією або рівень залежності.

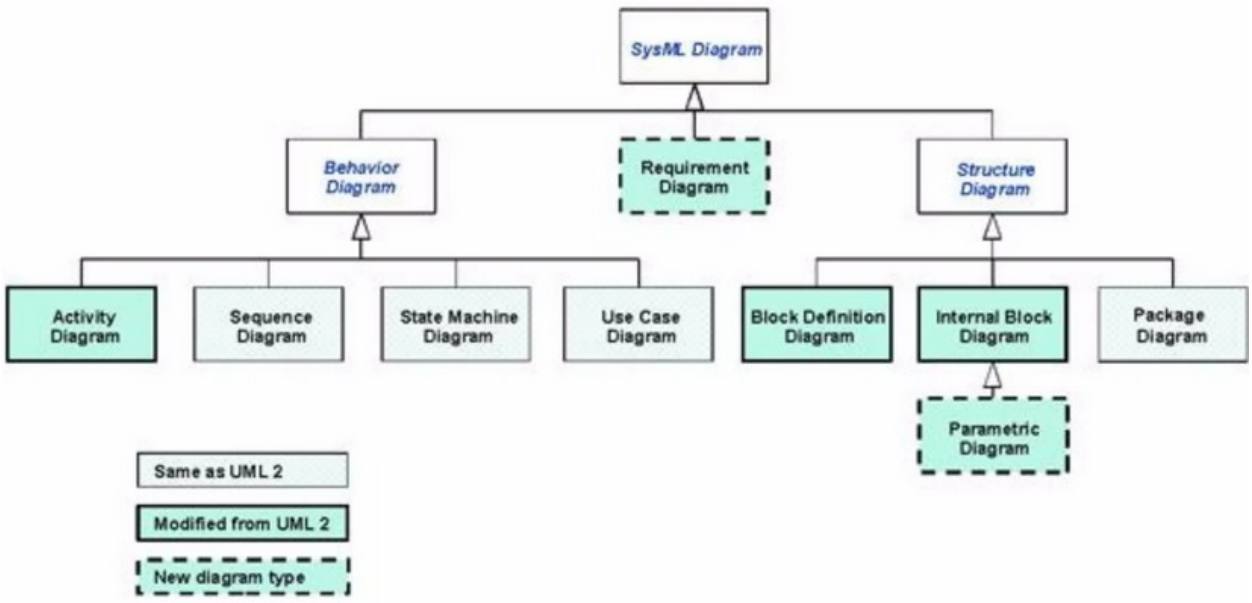

Рис. 2.3. Класифікація діаграм SysML

8. Діаграма вимог

Діаграма вимог використовується для передачі вимог до дизайну системи та зв’язку між ними. Крім того, це допомагає відстежувати вимоги до структури та поведінки системи, представленої на вищезгаданих діаграмах. По суті, це допомагає зацікавленим сторонам зрозуміти системні вимоги та відстежити їх також у моделі системи [7].

Діаграма моделі системи представляє лише один із видів моделі, але сама по собі моделлю не є. Класифікація цих діаграм на основі представлення певного аспекту системи зображена на рисунку 2.3.

2.1.3. Принципи проектування засобами SysML

Основні принципи проектування для SysML:

1. Керується вимогами

• Вимоги UML для цілей системної інженерії задовольняє SysML.

2. Повторне використання UML

• UML можна повторно використовувати в SysML скрізь, де це прагматично необхідно для задоволення вимог. Однак, якщо виникає потреба в модифікації, до мови можна вносити лише номінальні зміни. По суті, це полегшило б реалізацію SysML у тих середовищах, де вже підтримується UML.

3. Розширення UML

• UML надає методи розширення за допомогою механізму профілю, який допомагає розширити сутності в UML для певних цілей. Таким чином, щоб задовольнити вимоги системної інженерії, SysML розширює UML там, де це необхідно.

4. Пакетність

• Основною одиницею розділення в SysML був би пакет. Крім того, щоб обмежити циклічні залежності між елементами моделі, пакети поділяють елементи моделі на логічні групи [21].

5. Багатошаровість

• Пакети SysML визначені як рівень розширення метамоделі UML.

6. Сумісність

• UML має можливість для обміну XMI. Цю можливість успадкував SysML. Крім того, для підтримки взаємодії з іншими інженерними інструментами очікується, що стандарт обміну даними ISO 10303-233 підтримуватиме SysML.

Є кілька мов, доступних для різних цілей моделювання. UML є видатною мовою моделювання, яка переважно використовується в області розробки програмного забезпечення. Однак це призначення вимагає мови моделювання, придатної для системної інженерії, що робить SysML прийнятним варіантом. SysML надає діаграму вимог, яка не є частиною UML, що допомагає фіксувати вимоги як на вищому, так і на детальному рівнях. Однак UML пропонує лише діаграму варіантів використання, яка допомагає переважно охопити вимоги високого рівня. Крім того, SysML пропонує параметричну діаграму, яка допомагає вводити рівняння та обмеження, що допомагає створювати добре сформовані моделі. Ця діаграма недоступна в UML. Щоб змоделювати систему, першим кроком є порівняння вимог, потім створення структури системи, а потім поведінка системи, яка потім добре сформована за допомогою специфікацій параметричних обмежень. SysML пропонує цю можливість моделювати систему, охоплюючи її вимоги, структуру, поведінку та параметри.

2.2. Методологія побудови системи на основі V-моделі

V-модель є візуальним представленням життєвого циклу розробки системи. Він представляє дії, які необхідно виконати, і відповідні результати, яких необхідно досягти протягом життєвого циклу розробки системи. Ця V-модель зображена на рис 2.4 . Він має різні етапи, починаючи з лівої сторони з порівняння вимог та аналізу цих вимог, системного аналізу та проектування на основі цих зібраних вимог, детального проектування з логічними та

фізичними структурами з потоком інформації, реалізації детального проектування, а потім блоку тестування. Права сторона V-моделі включає інтеграцію різних модулів і їх тестування, тестування інтеграції систем і підсистем і прийняття системи.

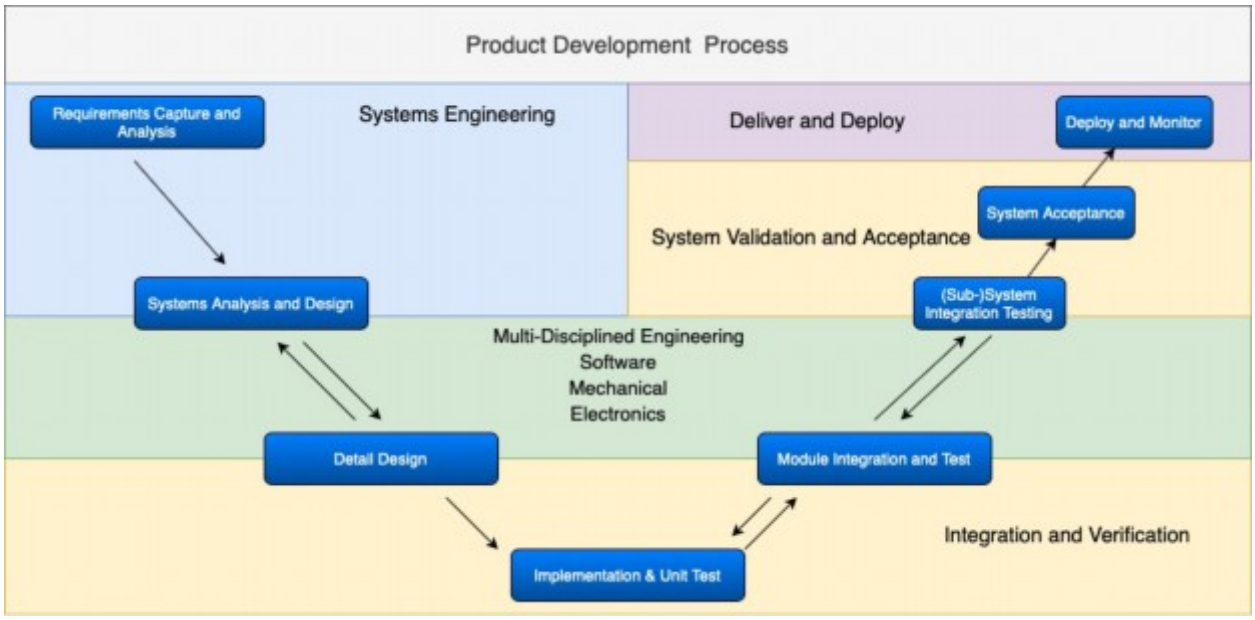

Рис. 2.4. Життєвий цикл розробки системи V-моделі

Однак у цієї V-моделі є деякі недоліки. Наприклад, якщо на етапі тестування виявлені будь-які помилки або вразливості, весь цикл повторюється повторно для перепроектування системи для усунення дефектів. Це відбувається через завершення тестування модулів і інтеграції системи після фази впровадження. Зрештою, для уникнення цих проблем виникає необхідність у ранній перевірці або безперервній перевірці. Одним із запропонованих підходів є «V-в-модель», де віртуальна інтеграція елементів системи завершується на ранніх стадіях. Це дозволяє проводити віртуальне інтеграційне тестування та перевірку кожного компонента системи, потоку інформації та пов’язаних елементів для виявлення несправностей ще до етапу впровадження. Цей підхід «V у V-моделі» зображено на рисунку 2.5. У наступному розділі пояснюється методологія SYSMOD, яка являє собою

набір рекомендацій, які можна використовувати для розробки системи на основі моделі з використанням вищезгаданого підходу V-моделі.

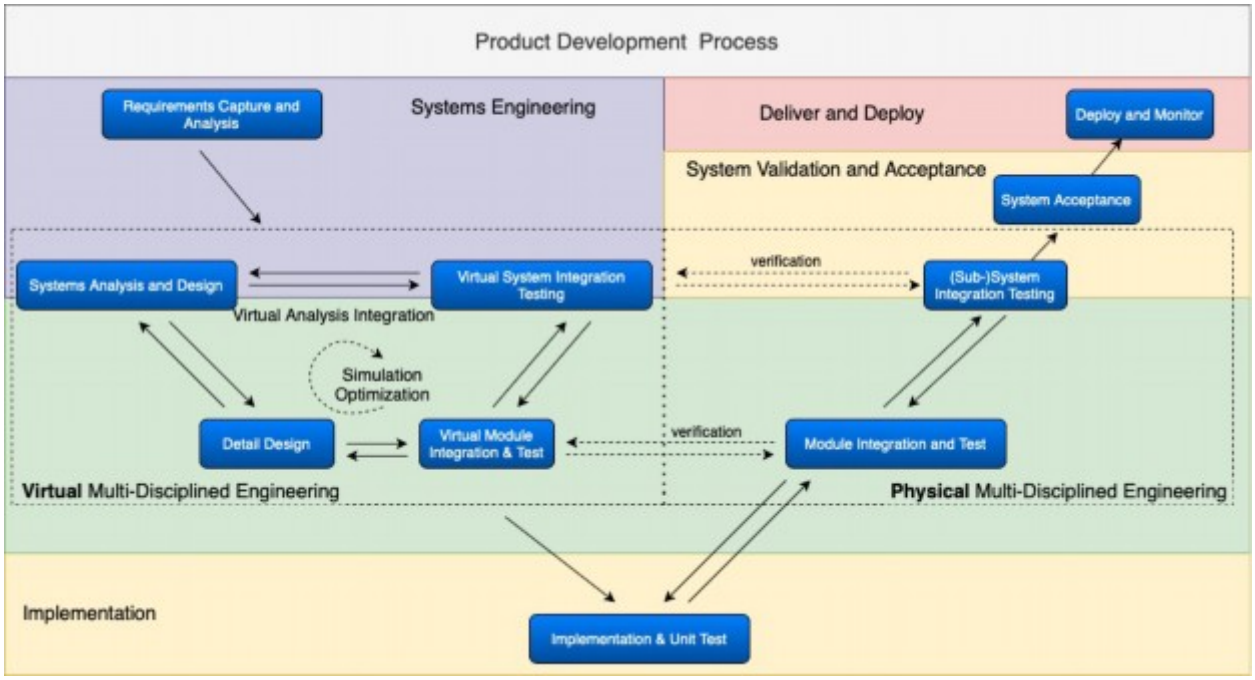

Рис. 2.5. Життєвий цикл розробки системи "V у V-моделі"

SYSMOD — це термін, введений в [27], який походить від «Моделювання систем». Це розшифровується як Systems Modeling Toolbox, який переважно використовується для розробки системи, де він допомагає реалізувати ідею системи за допомогою проектування системи. Це логічний підхід, який охоплює як цілісний, так і детальний рівень розуміння вимог до розробки системи. Таким чином, це підхід, орієнтований на високі вимоги. Він забезпечує набір завдань, рекомендації та найкращі практики моделювання систем. Різні завдання, охоплені цією методологією:

- Опис контексту проекту

Це текстовий опис цілей проекту, над якими команда працюватиме разом. Він включає в себе перший крок визначення ідей і цілей системи..

- Визначення зацікавлених сторін

Особи в організації або за її межами, зацікавлені в системі, які можуть вплинути на вимоги системи або зробити свій внесок у них, вважаються

зацікавленими сторонами. Наступним кроком буде визначення цих осіб із ширшої сфери та визначення пріоритетів цих зацікавлених сторін. Це допомагає зібрати різні погляди на систему, що зрештою призведе до моделювання детального рівня вимог системи.

- Моделювання вимог.

Необхідно змоделювати різний набір вимог у кількох категоріях. Для кращої ясності ці вимоги можна класифікувати на різній основі. Однією з прикладів моделі може бути класифікація вимог на основі характеристик якості, операцій, безпеки та політики. Іншим прикладом моделі вимог є модель FURPS, яка складається з:

- Функціональні вимоги
- Вимоги до зручності використання
- Вимоги до надійності
- Вимоги до виконання
- Вимоги до підтримки

- Моделювання контексту системи

На користь зацікавлених сторін зібрані вимоги переведено в графічний опис логічної та фізичної структур системи. Це контекст системи. Він моделюється на основі цілісного уявлення про систему, а окремі компоненти моделюються в підконтексти для представлення детального рівня деталізації. Крім того, можуть бути створені контекстні точки зору, щоб розмежувати занепокоєння різних зацікавлених сторін на основі їхніх вимог [40]. Крім того, на цьому етапі методології SYSMOD завершується процес ідентифікації акторів системи [37]. Крім того, контекст системи можна сприймати як представлення найвищого рівня абстракції структури системи.

- Моделювання інформаційного потоку

Моделювання потоку інформації між різними компонентами системи робить її зрозумілою структурою. Фактично, моделювання інформаційного потоку стає частиною системного контексту [40]. Крім того, це моделювання допомагає чітко зрозуміти предметну область і служить основою для

розуміння можливих випадків використання в системі. Крім того, це допомагає ідентифікувати точки взаємодії в системі [37].

- Моделювання випадків використання

Варіанти використання, які є наслідком зіставлених вимог, представляють різні функціональні можливості служби, які надає система. Першим кроком на цьому етапі є визначення варіантів використання. Під час ідентифікації для моделювання цих варіантів використання використовується низхідний підхід, де спочатку моделюються варіанти використання вищого рівня, а потім — варіанти використання нижнього рівня [37]. Подібно до того, як контекст системи представляє найвищий рівень абстракції структури системи, варіанти використання можна розуміти як представлення найвищої абстракції поведінки системи.

- Опис системних процесів

Залежності між варіантами використання визначаються системними процесами. Системні процеси представляють логічний потік у різних варіантах використання, що ефективно допомагає визначити залежності потоків. Крім того, вони допомагають представити агреговану поведінку системи [37].

- Моделювання знань предметної області

Модель знань предметної області створюється для представлення структур предметної області в проекті. Це представляє узгоджене структурне представлення специфічних для предметної області онтологій у проекті, що робить його легшим для розуміння зацікавленими сторонами [37].

- Моделювання структури та поведінки системи

На основі порівняних вимог і змодельованого контексту системи моделюється повна структура системи разом із її компонентами, з’єднаннями, портами та інтерфейсами, операціями, властивостями тощо. Крім того, поведінка системи разом із обмеженнями та параметрами моделюється для реалізації випадків використання, змодельованих на попередніх етапах.

- Моделювання моделі

Моделі моделюються в середовищах виконання в інструменті моделювання. Моделювання виконується для виявлення недоліків у проектних рішеннях, що корисно на ранніх етапах, оскільки вартість виправлення буде мінімальною. Крім того, генератори коду можуть бути використані для створення виконуваної моделі змодельованої системи, яку можна використовувати як для моделювання, так і для тестування [40].

- Тестування моделі

Тестування моделі в тестовому середовищі допомагає перевірити модель системи, щоб перевірити, чи реалізовано зіставлені вимоги щодо структури та поведінки системи. Крім того, може існувати різна основа для проведення тестів для підтвердження моделі для різних цілей, таких як аналіз безпеки, аналіз узгодженості та інші. По суті, це допомагає виявити помилки та вразливі місця в моделі системи. При ідентифікації ці несправності можна усунути з мінімальними витратами, що зрештою прокладе шлях до вдосконалення моделі системи.

Незважаючи на те, що методологію SYSMOD можна застосовувати за допомогою переважно використовуваної мови моделювання для систем, якою є OMG SysML (мова системного моделювання групи керування об’єктами), її також можна використовувати для інших мов моделювання, таких як OMG UML (група керування об’єктами). Unified Modeling Language) та ін.

2.3. Дослідження переваг використання доменно спеціалізованої мови

DSL можна розуміти як формальну мову, яка є сукупністю виразів, символів або речень на основі певного формалізму, де ці елементи представляють сутності, специфічні для домену. Одне з визначень DSL взято з джерела [41]: «Доменно-орієнтована (спеціалізована) мова (DSL) — це

невелика, як правило, декларативна мова, яка пропонує експресивну силу, зосереджену на конкретній проблемній області».

Є кілька переваг використання DSL перед мовою загального призначення:

1. Висловлювання з використанням предметних понять, що веде до кращого розуміння та покращення спілкування;
2. Покращена продуктивність, якість, довговічність даних, надійність тощо;
3. Продуктивний інструментарій і дуже стислий.

2.3.1. Чотирирівнева архітектура UML

DSL, який використовується в цьому призначенні, призначений для цілей моделювання, і тому його можна назвати Domain Specific Modeling Language [35]. Є кілька можливих методів з UML для створення нового DSL. Щоб краще зрозуміти ці методи, необхідні знання про архітектуру UML.

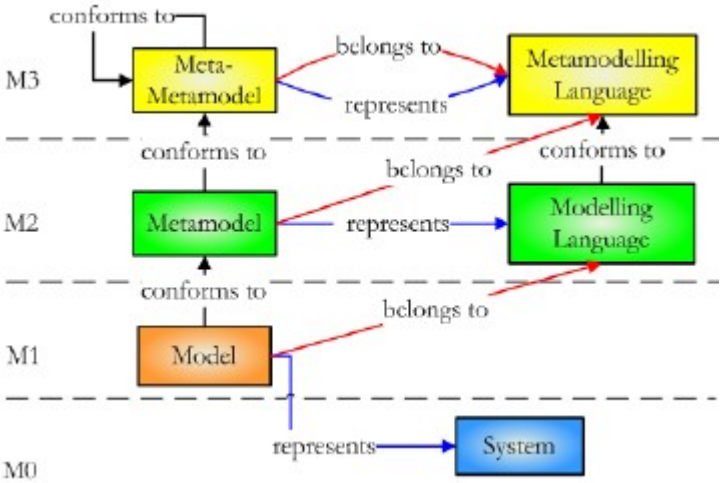

Рис. 2.6. Чотирирівнева архітектура UML

UML має чотирирівневу архітектуру, де кожен із цих рівнів представляє різні рівні абстракції:

1. M3: рівень мета-мета-моделі

На цьому найвищому рівні визначається мова для визначення метамоделей. Наприклад, цей рівень містить мову MOF, яку можна використовувати для створення структури метаданих. Формалізми моделювання UML визначені єдиним чином за допомогою цього MOF.

2. M2: рівень метамоделі

Цей рівень можна сприймати як екземпляр рівня мета-мета-моделі, який охоплює визначення структури метаданих. Метамоделі в цьому шарі створюються з моделі шару M3. Метамодель UML, створена з MOF, належить до цього рівня [20].

3. M1: Рівень моделі

Цей рівень можна сприймати як екземпляр рівня метамоделі, який охоплює визначення об’єктів або даних на рівні M0. Крім того, він визначає мову для специфікації конструкцій, які можна використовувати для конкретного домену. Метамоделі рівня M2 описують структуру сутностей рівня M1. Моделі, побудовані з використанням UML, належать до цього рівня [20].

4. M0: рівень моделі об’єкта або даних

Цей шар є екземпляром рівня M1: model. Він складається з даних або об’єктів [20].

2.3.2. Методи створення DSL

Існують різні варіанти визначення DSL. Кожен із цих методів має свої переваги та недоліки. Метод, обраний для впровадження DSL, базується на вартості та простоті впровадження. Є три основні методи, доступні для визначення DSL [35]:

1. Визначення нової мови
2. Розширення та модифікація існуючої метамоделі
3. Удосконалення або розширення існуючої мови моделювання загального призначення для включення конструкцій, специфічних для домену

Незважаючи на те, що перший метод є прямим підходом, він має ряд недоліків і труднощів у реалізації. Цей варіант потребує нового інструменту моделювання для підтримки цієї нової мови з її конструкціями. Крім того, врахувати складну та заплутану семантику, що стоїть за цими мовними конструкціями, дорого.

Другий метод передбачає модифікацію існуючої мета-моделі UML шляхом додавання нових елементів і концепцій у мета-модель. Це важкий метод розтягування, який має такі ж труднощі, як і перший метод.

Третій метод — це легкий метод розширення, який використовує положення в UML, що дозволяє розширенню UML додавати специфічні для домену конструкції. Крім того, він дозволяє повторно використовувати існуючі інструменти, що підтримують UML. Крім того, цей метод надає можливість використовувати існуючі елементи моделі UML. По суті, DSL можна створити за допомогою цього методу розширення UML шляхом ретельного повторного використання елементів UML, де це можливо, і створення нових елементів, специфічних для домену, лише там, де це необхідно [35]. Третій спосіб є найефективнішим за вартістю та простотою реалізації, і саме цей метод був обраний для даного завдання.

Висновки до розділу

У цьому розділі було розглянуто мову моделювання систем (SysML), її виникнення, взаємозв’язок з UML та різні типи діаграм, які вона пропонує. Також було детально висвітлено життєвий цикл розробки продукту за моделлю V та методологію SYSMOD. Окрема увага була приділена концепції предметно-орієнтованих мов (DSL), а також потребі в розробці DSL для Arrowhead Framework. Визначено, що хоча необхідність у створенні DSL є очевидною, цей DSL має забезпечувати можливість використання спільно з існуючими мовами моделювання, такими як SysML. Це дозволить застосовувати існуючі інструменти, що підтримують мови моделювання

загального призначення, для моделювання з використанням елементів DSL. Окрім цього, елементи SysML можуть бути повторно використані для представлення загальних структур у поєднанні з елементами DSL. Метод розширення профілю UML та різні елементи створеного DSL будуть докладно описані в наступному розділі.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МОДЕЛЕЙ ТА МЕТОДІВ ДОМЕННО СПЕЦИФІКОВАНОЇ МОВИ ДЛЯ ЗАСТОСУВАНЬ ІНТЕРНЕТУ РЕЧЕЙ

3.1. Методика використання доменно специфікованої мови

Було встановлено потребу в доменно специфікованій мові (DSL) для Arrowhead Framework. Поки потреба встановлена, необхідно вибрати інструмент, необхідний для цього завдання, виходячи з вимог цього завдання. Одним із головних завдань для цього завдання був вибір відповідного інструменту для створення та перевірки DSL. У цьому розділі міститься інформація про інструмент, вибраний для призначення, і методологія створення DSL. У цьому розділі далі розглядаються елементи, створені в профілі Arrowhead Framework, а також семантика та обґрунтування створених елементів. Крім того, також було з'ясовано елементи в профілі, які використовуються для створення бібліотек моделей, і створені елементи бібліотеки моделей.

Щоб побудувати DSL, що охоплює набір елементів у домені, який би широко використовувався, існує кілька кроків для реалізації проекту DSL. Розглядаючи Arrowhead Framework, модель предметної області має бути визначена такими елементами:

- Елементи в DSL мають бути основним набором мовних конструкцій, що представляють основні поняття в домені.
- Зв'язки між основними поняттями або конструктами.
- Набір атрибутів цих основних понять.
- Послідовне графічне представлення конструкцій у моделі.
- Семантика репрезентації моделі.

Першим кроком для створення DSL є визначення понять у домені. Щоб ідентифікувати концепції, слід провести широке дослідження домену. Наступним кроком є визначення проблемного простору в домені. Це

передбачає взаємодію з командою Arrowhead Framework, щоб зрозуміти їхню потребу в профілі. Після ідентифікації проблемного простору наступним кроком є визначення мовних понять. Слід провести широке дослідження різних концепцій мета-моделі UML. Цей крок допоможе визначити мовні поняття. Наприкінці цього етапу можна визначити відповідні поняття домену шляхом зіставлення ідентифікованих понять домену з ідентифікованими мовними поняттями. Цей крок, можливо, можна виконати протягом кількох ітерацій. Наприкінці цього кроку буде завершено початкове відображення між концепціями предметної області та концепціями мови [46]. Це слугуватиме основою для подальшого відображення концепцій домену на різних рівнях абстракції. Наприкінці цього останнього етапу всі концепції домену на різних рівнях абстракції будуть зіставлені з різними мовними концепціями.

Для вибору інструменту було розглянуто та оцінено декілька версій різних інструментів моделювання. Корпоративні або пробні версії цих різних інструментів були встановлені, інтегровані із зовнішніми генераторами коду, а потім оцінені. Серед розглянутих інструментів були IBM Rational Software Architect Designer [13], IBM Rational Rhapsody Architect for Software [5], Cameo Systems Modeler і MagicDraw [15]. Ці інструменти були оцінені на основі вимог цього завдання. Ці вимоги до інструменту моделювання полягали в забезпеченні підтримки методу розширення профілю UML, підтримки моделювання за допомогою SysML, підтримки мов обмежень моделі, таких як OCL, підтримки вбудованого спеціального коду або створення шаблонів, підтримки експорту даних у термінах побудованих моделей, створені профілі та згенерований код, підтримка редагування й оновлення моделей і час, необхідний для початку роботи з інструментом на основі поточної доступності ліцензії. Це порівняння різних інструментів на основі вимог до призначення DSL Arrowhead Framework показано в таблиці 3.1.

Таблиця 3.1.

Порівняння різних інструментів на основі вимог до призначення доменно-специфічної мови Arrowhead Framework

№	Вимоги	IBM Rational Software Architect Designer	IBM Rational Rhapsody Architect for Software	Cameo Systems Modeler	MagicDraw
1	Підтримка методу розширення профілю UML	так	так	так	так
2	Підтримка моделювання за допомогою SysML	немає	так	так	так
3	Підтримка мов модельних обмежень, таких як OCL	так	немає	так	так
4	Підтримка вбудованої генерації спеціального коду/шаблону	так	немає	так	так
5	Підтримка експорту даних (моделі, згенерований код, профіль)	так	так	так	так
6	Редагування та оновлення моделей	так	так	так	так
7	Готовий до використання з точки зору наявності	так	так	так	немає

Спочатку рішення схилялося до IBM Rational Rhapsody Architect for Software, оскільки він задовольняв кілька вимог. Однак він не мав функції власного генератора коду/шаблону, яка б допомагала генерувати код, сумісний із Arrowhead Framework. Інструменти генератора коду, такі як MDWorkbench, які інтегруються з цим інструментом, також були встановлені та оцінені. Однак, оскільки IBM Rational Rhapsody Architect for Software не підтримував OCL, цей інструмент не був правильним варіантом для цього призначення. З іншого боку, Cameo Systems Modeler і MagicDraw задовольняли всім основним вимогам. Однак ліцензія на Cameo Systems Modeler вже була доступна, тому її було обрано як правильний інструмент.

3.2. Механізм розширення в метамоделі UML

UML надає можливість розширення існуючих елементів UML. Компонент профілю UML, який є підмножиною метамоделі UML, надає цю опцію. Цей метод профілю UML не створює нових елементів у метамоделі. Навпаки, він містить положення, які б розширювали існуючий елемент UML у специфічну конструкцію домену [1]. Профіль UML складається з трьох основних компонентів для розширень, і вони представлені на рисунку 3.1.

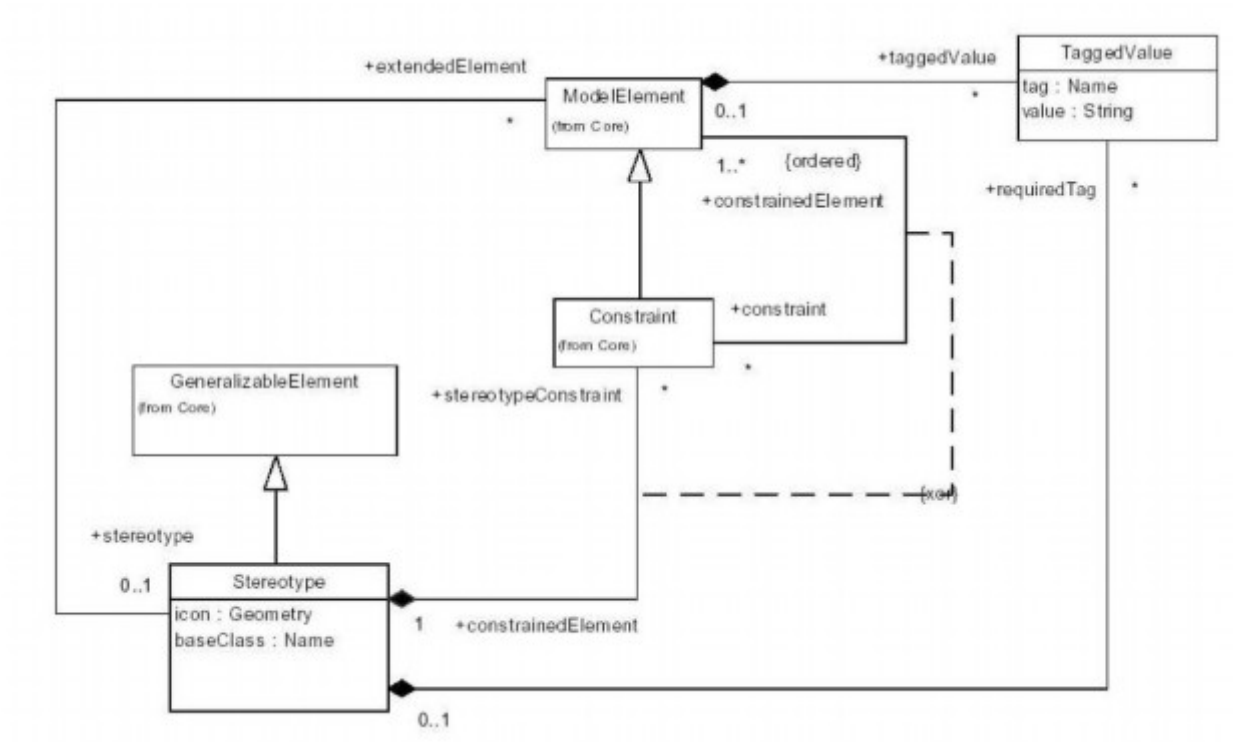

Рис. 3.1. Механізми розширення в метамоделі UML

Ці компоненти наступні:

1. Стереотипи.

Стереотипи — це елементи UML, які розширюють існуючий елемент UML для створення нового елемента в певній області, що представляє концепцію, фізичний або логічний елемент, відношення або будь-який інший елемент у цій області. Стереотип представлений ключовим словом «стереотип», назвою та відповідним метаелементом, від якого він походить.

2. Значення з тегами

Значення з тегами використовуються для визначення нової або додаткової інформації в стереотипі, по суті представляючи атрибути, пов’язані з цим стереотипом. Позначені тегами значення представлені назвою та типом.

3. Обмеження

Обмеження — це правила, які можна використовувати для обмеження функціонування чи поведінки цих нових стереотипних елементів залежно від контексту домену. По суті, це допомагає зробити модель правильної форми. Обмеження можуть бути виражені певними мовами, такими як OCL, або навіть природною мовою [1].

3.2.1. Елементи в профілі Arrowhead Framework

Профіль — це тип пакету, який складається зі стереотипів, які налаштовують UML для певної мети. Команда IncQuery Labs вже створила базовий профіль Arrowhead Framework [17]. Цей профіль охоплює досить багато важливих понять у Framework. Однак цей профіль не охоплює всі аспекти цієї структури, і є деякі концепції, які необхідно чітко визначити. Наприклад, їхній профіль складається з деяких елементів, які використовуються для представлення двох різних концепцій. З одного боку, ці елементи використовувалися для представлення різних документів, які є частиною моделі документації у Framework. З іншого боку, ці самі елементи представляють такі концепції в рамках, як системи, служби та інші. Використання того самого елемента для представлення двох різних концепцій робить це досить неоднозначним для модельєра. Крім того, бракує чіткого уявлення про те, як модель документації пов’язана з основними поняттями в профілі.

Повний Arrowhead Framework DSL складається з трьох частин. Перша частина — це профіль, що складається з елементів домену з точки зору стереотипів і позначених значень. Друга частина — обов’язкова базова

бібліотека моделей, яка складається з часто використовуваних обов'язкових базових систем і служб. Третя частина — це бібліотека основних моделей підтримки, яка складається з часто використовуваних основних систем і служб підтримки. Початкове відображення між двома концепціями, зображене в таблиці 3.2 за основу побудували елементи в профілі. Було ідентифіковано різні елементи, що входять до складу концепцій Arrowhead Framework, і вони визначені в профілі. Це був ітеративний процес, і врешті-решт було визначено всі елементи профілю.

Таблиця 3.2.

Початкове відображення специфічних концепцій Arrowhead Framework на концепції UML

Arrowhead FrameworkConcepts	UML Concepts
System of Systems	Class
Local Cloud	Class
System	Class
Device	Class
Service(Includes the technological information)	Class
Deployed Elements (Deployed element specific information)	Property, Artifact
HTTP Method (POST, GET, PUT, DELETE, PATCH)	Operation
Data Semantics	Class
ServiceExchange	Connector, InformationFlow
Input and Output Parameters	Class
Arrowhead Framework Description Documents (Blackbox & Whitebox)	Artifact

Будь-яка сутність, яка є апаратним забезпеченням або машиною, яка має можливість обчислювати, передавати та зберігати дані, а також розміщувати одну або багато сумісних систем arrowhead framework разом із їхніми службами є пристроєм arrowhead framework.

Крім того, пристрій стане сумісним зі структурою стрілок, лише якщо:

• Його можна завантажувати в локальну хмару фреймворка зі стрілками

• Він може динамічно встановлювати нові системи та їхні служби

• Його можна зареєструвати в системі DeviceRegistry

• Його можна динамічно налаштовувати

Профільна діаграма, що зображує зв'язок між вищезгаданими стереотипами, показана на рисунку 3.2.

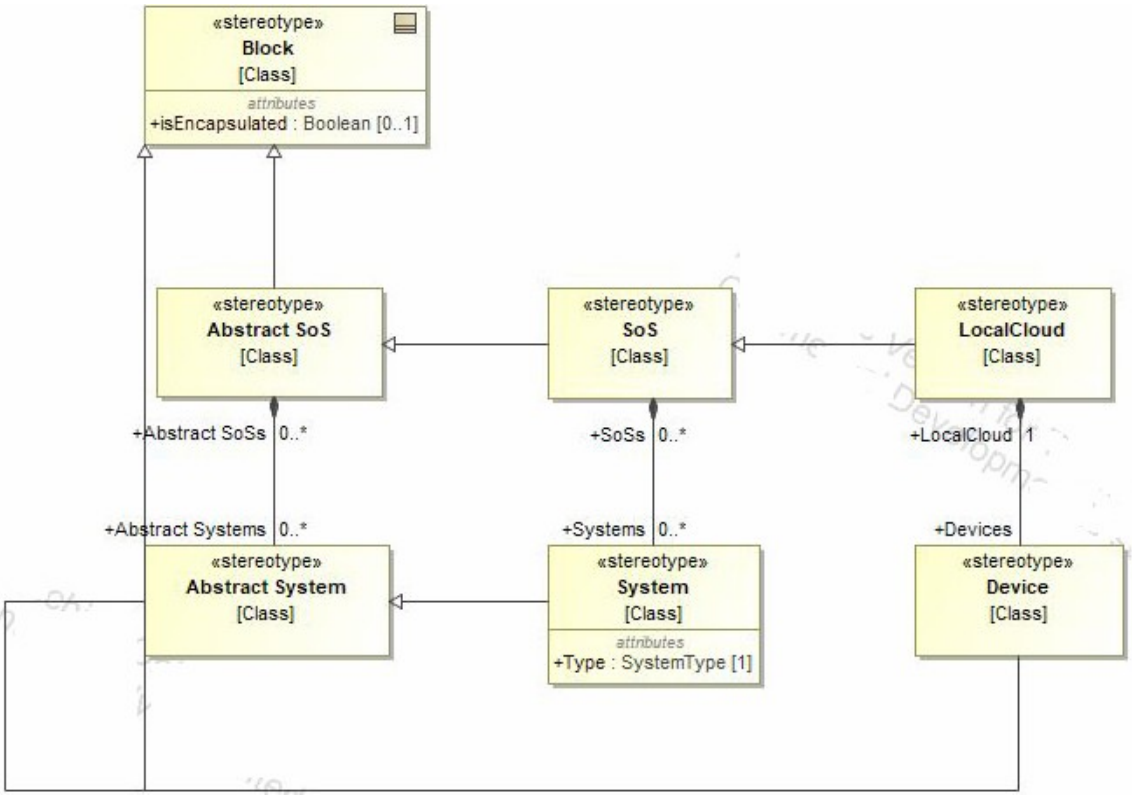

Рис. 3.2. Діаграма профілю, що включає стереотипи SoS, локальну хмару, систему та пристрій IoT

3.2.2. Концепція сервісу в Arrowhead Framework

Як згадувалося раніше, обмін інформацією між виробником і споживачем представлений сутністю, яка називається Сервіс [31]. Його можна розуміти як програмну одиницю, яка є логічним представленням конкретного завдання або містить певну інформацію [30]. Крім того, послуга є багаторазовою, незалежною від контексту та повинна мати можливість виявлення іншими системами. Крім того, сервіс у Arrowhead Framework міг

мати пов’язані метадані та підтримувати нефункціональні вимоги, такі як різні рівні надійності, безпеки та умови реального часу [3].

Для того, щоб служба була сумісною з Arrowhead Framework, вона повинна мати можливість зареєструватися в обов’язкових основних системах і створюватися та/або споживатися системою, сумісною з Arrowhead Framework. Крім того, конкретна послуга може бути створена однією системою або навіть кількома різними системами [4]. Послуга, наприклад дані датчиків, може пропонуватися декількома системами, які можуть використовуватися іншими системами.

Діаграма профілю, що зображує зв’язок між стереотипами служби та абстрактної служби, показана на рисунку 3.3.

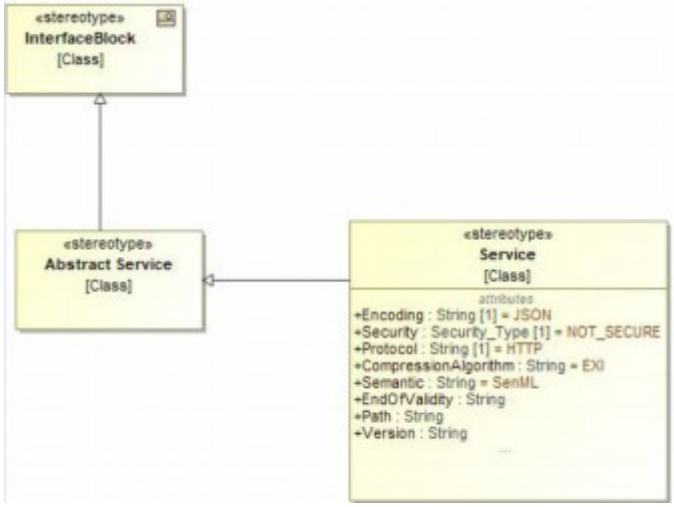

Рис. 3.3. Діаграма профілю, що складається з абстрактних послуг і стереотипів послуг

3.2.3. Концепція використання локальної хмари

DeployedLocalCloud є розширенням метакласу Property та Artifact, і він успадковує стереотип DeployedEntity. Він представляє концепцію локальної хмари, розгорнутої разом із фреймворком. Таким чином, цей стереотип можна застосувати до будь-якої локальної хмари, розгорнутої за допомогою фреймворку. Він має теговане значення під назвою GatekeeperSystemName, яке представляє назву системи Gatekeeper у цій розгорнутій локальній хмарі

для міжхмарного зв’язку. Це теговане значення може приймати будь-які рядкові значення.

Діаграма профілю, що зображує зв’язок між розгорнутою сутністю, розгорнутою локальною хмарою, розгорнутим пристроєм і стереотипами розгорнутої системи, показана на рисунку 3.4.

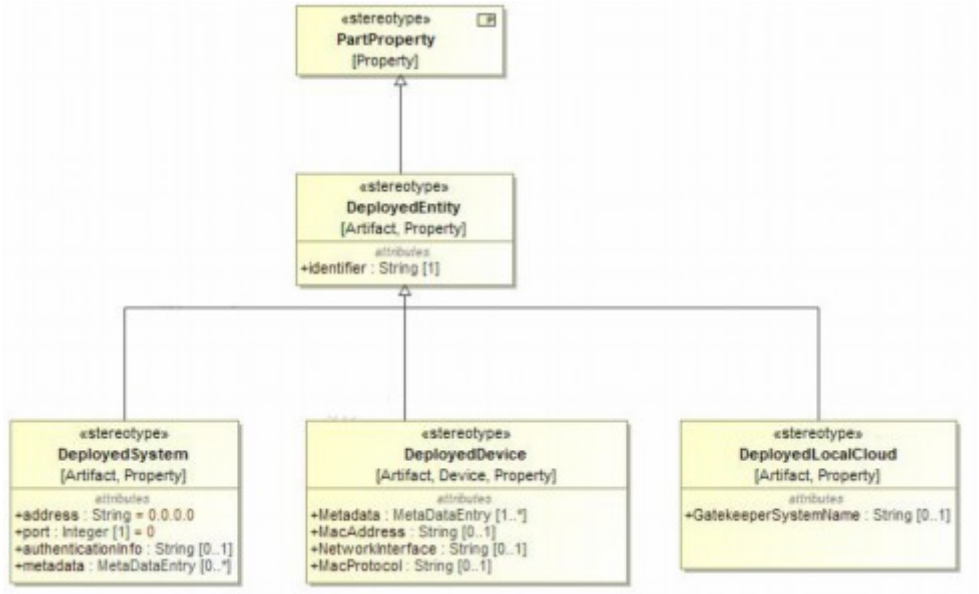

Рис. 3.4. Діаграма профілю, що складається зі стереотипів розгорнутих сутності, локальної хмари, системи та пристрою

3.2.4. Реалізація та опис стереотипів метакласів

Стереотип POST є розширенням метакласу Operation і успадковує стереотип HTTP Method. Він представляє будь-який метод HTTP POST, який використовується в операції. Стереотип GET є розширенням метакласу Operation і успадковує стереотип HTTP Method. Він представляє будь-який метод HTTP GET, що використовується в операції. Стереотип PUT є розширенням метакласу Operation і успадковує стереотип HTTP Method. Він представляє будь-який метод HTTP PUT, який використовується в операції

Стереотип PATCH є розширенням метакласу Operation і успадковує стереотип HTTP Method. Він представляє будь-який метод HTTP PATCH, який використовується в операції.

Діаграма профілю, що зображує зв’язок між стереотипами різних методів роботи показана на рисунку 3.5.

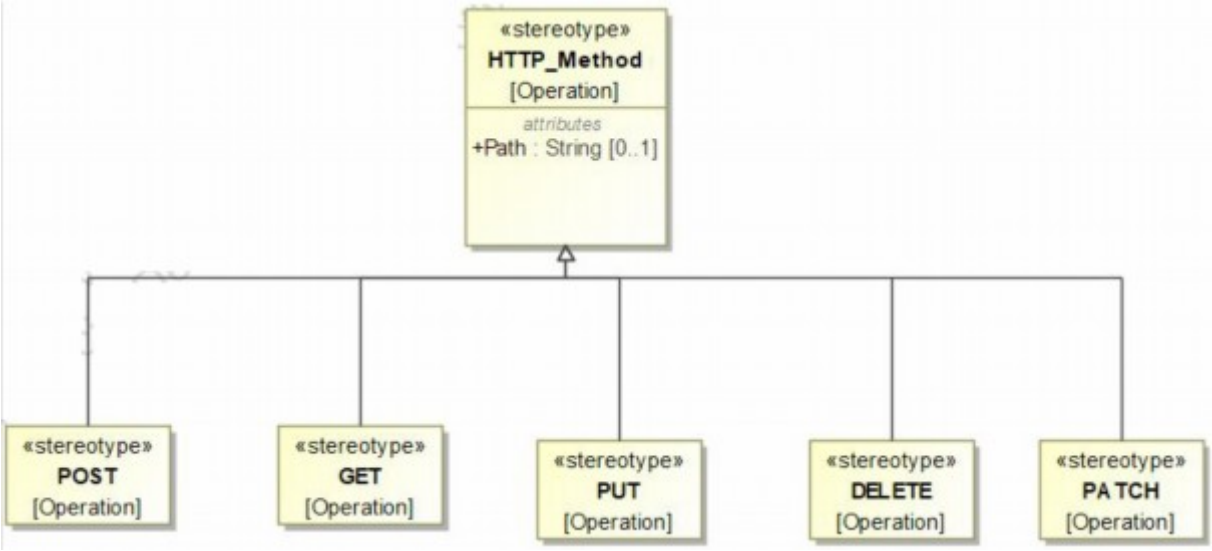

Рис. 3.5. Діаграма профілів , що містить стереотипи різних методів роботи

Стереотип SoSDD є розширенням метакласу Artifact, який представляє документ з описом дизайну SoS. Опис проектування системи (SoSDD) — це документація рівня White Box SoS. Він представляє опис реалізації для конкретного сценарію, що включає повний опис точних технологій, що використовуються, і відповідних налаштувань. Крім того, цей документ містить опис логічної реалізації основних функціональних можливостей з точки зору варіантів використання, структурних і поведінкових діаграм. Крім -того, цей документ охоплює детальний опис фізичної реалізації, включаючи детальний рівень інформації, такої як фізичні обмеження та фізичне розташування пристроїв тощо. Крім того, він охоплює опис реалізації реалізованих нефункціональних вимог.

Стереотип SySD є розширенням метакласу Artifact, який представляє документ System Description. Документ «Опис системи» — це документація на системному рівні Black Box. Цей документ містить інформацію щодо визначення послуг, що надаються та споживаються, і відповідних інтерфейсів, без опису точної технічної реалізації. Крім того, цей документ

змушує використовувати формальні або напівформальні моделі для розміщення описів, що охоплюють семантичну інформацію про системи, які прокладуть шлях до оцінки сумісності різних систем.

Стереотип SySDD є розширенням метакласу Artifact, який представляє документ Опис дизайну системи. Документ System Design Description — це документація на системному рівні White Box. Цей документ містить опис технологічної реалізації каркасних систем стрілчастих наконечників. Цей документ складено необов'язковий, тому у випадках, коли фактичне впровадження неможливо оприлюднити або коли знання компанії, що впроваджує, можуть бути розкриті, цього документа можна уникнути. У цих випадках буде достатньо лише використання SysD.

Стереотип IDD є розширенням метакласу Artifact, який представляє документ з описом дизайну інтерфейсу. Опис дизайну інтерфейсу містить детальний опис фактичної реалізації послуги. Ідентифікатори послуг для конкретних реалізацій послуг визначено в цьому документі. Крім того, у ньому міститься посилання на документ профілю зв'язку для визначення інформації про протокол зв'язку та використаних методів кодування, шифрування та стиснення. Крім того, він посилається на документ семантичного профілю для визначення семантики інформації та даних, що використовуються [33].

Стереотип SD є розширенням метакласу Artifact, який представляє документ Service Description. Документ з описом послуги містить абстрактний опис потреб системи для надання або споживання послуги. Крім того, він забезпечує основні цілі та функціональні можливості служби разом із її абстрактними інтерфейсами. Розробники системи, сумісної зі стрілочною структурою, створюють цей документ, який описує служби додатків. Кожна служба має набір нефункціональних вимог, таких як використання ресурсів, час відгуку, надійність, якість обслуговування (QoS) та інші, які також повинні бути вказані з детальним описом.

Стереотип CP є розширенням метакласу Artifact, який представляє документ Profile Communication. Документ «Профіль зв'язку» є частиною документації рівня обслуговування та містить технічну інформацію, що стосується кодування даних, стиснення даних, шифрування даних і протоколу зв'язку, який використовується під час фактичної реалізації.

стереотип SP є розширенням метакласу Artifact, який представляє документ Semantic Profile. Документ семантичного профілю також є частиною документації рівня обслуговування та містить інформацію, що стосується семантики інформації або даних, які використовуються в фактичній реалізації.

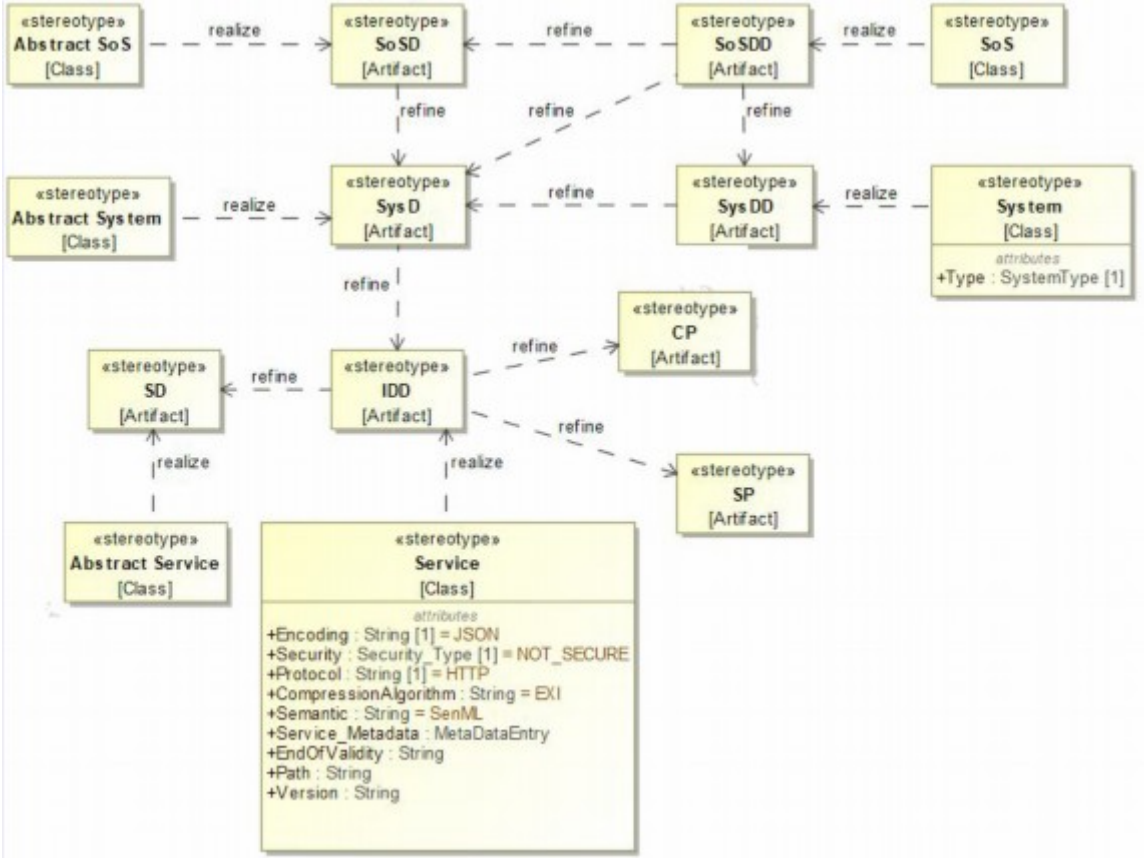

Рис. 3.6. Діаграма профілю , що складається з різних стереотипів документа Arrowhead Framework

Діаграма профілю, що зображує зв'язок між різними стереотипами документа Arrowhead Framework, показана на рисунках 3.6 і 3.7. На цій

діаграмі використано два основних зв’язки, а саме «реалізувати» та «уточнити». Відношення реалізації в UML було визначено в специфікації OMG UML 2.5 як «Реалізація — це спеціалізований абстрактний зв’язок між двома наборами Елементів моделі, один з яких представляє специфікацію (постачальник), а інший — реалізацію останнього (клієнт)» [22].

SysDD — це документ з описом проекту системи, який містить деталі реалізації конкретної системи, а Система є фактичною реалізацією на основі деталей, указаних у SysDD. Отже, SysDD діє як постачальник, а System — як клієнт.

Інший зв’язок, який використовується на діаграмі, — уточнення, який є типом зв’язку залежності. Це співвідношення означає, що елемент на стороні клієнта є конкретним, а елемент на стороні постачальника є абстрактним [7]. Цей зв’язок використовується між конкретними та абстрактними документами. Наприклад, SoSDD — це конкретний документ «білої скриньки», який охоплює кожну деталь реалізації та функціональності, тоді як SoSD — це лише абстрактний документ «чорної скриньки», який охоплює основні деталі архітектури без уточнення будь-якої технологічної інформації.

Цей профіль, що складається з цих різних елементів, перетворюється на спільний пакет у Cameo Systems Modeler. Це дозволить перевірити наявність звичайних і циклічних залежностей у профілі перед перетворенням його на спільний пакет. Після перетворення в спільний пакет цей профіль можна імпортувати в будь-який проект, а його елементи можна використовувати для моделювання.

3.3. Бібліотеки моделей для основних систем

Бібліотека моделей - це тип пакету, який складається з елементів, які зазвичай призначені для частого повторного використання в моделях [7]. Елементи в профілі Arrowhead Framework використовуються для створення елементів у бібліотеках моделей. У цьому призначенні є дві основні

бібліотеки моделей, а саме одна для обов'язкових базових систем, а друга – для базових систем підтримки.

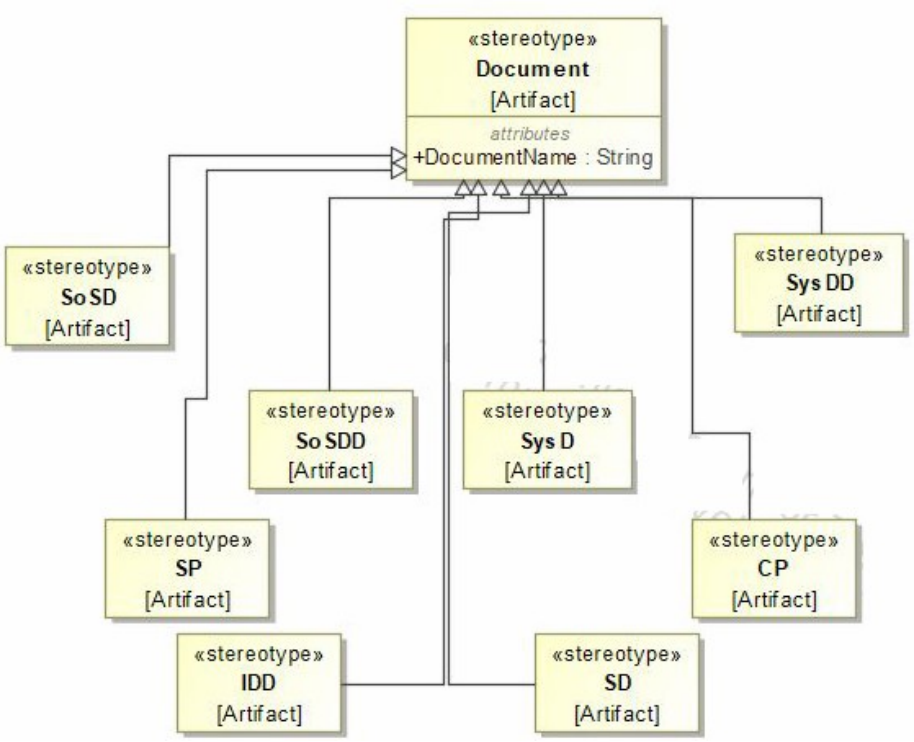

Рис. 3.7. Діаграма профілю , що складається з різних стереотипів документа Arrowhead Framework

Нижче наведено короткий опис того, як було змодельовано елементи в бібліотеках моделей. Спочатку визначаються різні основні системи та послуги, які вони надають і споживають. Основні операції, які є частиною служб, ідентифікуються разом із їхніми вхідними та вихідними параметрами та типами операційних методів. Абстрактна система і системні стереотипи були використані для представлення абстрактної та конкретної системи. Назви системи вказані у визначеннях тегів. Подібним чином абстрактні послуги та стереотипи послуг використовуються для представлення послуг, які пропонує система. Інформація про послуги вказується в значеннях тегів. Різні операції, які є частиною сервісу, додаються в абстрактний стереотип сервісу без уточнення будь-яких технологічних деталей. Технологічні деталі операції прописані лише в сервісному стереотипі. Під час визначення

операцій, які є частиною служби, типи методів визначаються з використанням різних стереотипів методів HTTP. Під час визначення вхідних і вихідних параметрів до них було застосовано стереотипи InputParameter і OutputParameter на основі їхніх ролей. Крім того, семантика даних, яка використовується в цих параметрах, визначається за допомогою стереотипу семантики даних. Ідентифікуються надані та спожиті послуги системи. Ці служби представлені як порти в системі. Якщо певна послуга використовується системою, порт сполучається, щоб відобразити споживання послуги цією системою. Якщо системою надається кілька послуг, стереотипи послуг створюються для кожної наданої послуги. Крім того, кожна з цих служб додається до системи як порти. Дві бібліотеки моделей основних систем детально описані в наступних розділах.

Система оркестровки відповідає за пізнє зв'язування між прикладними системами, тобто за надання інформації про оркестровку прикладним системам, яка допомагає їм отримати інформацію про підключення, щоб споживати ті послуги, які їм потрібні [4]. Правила оркестрів містять інформацію про [32]:

- Доступ до інформації постачальника послуг щодо мережевої адреси та номера порту
- Інформація про специфікацію екземпляра послуги, що надається системою провайдера. Ця інформація включає детальну інформацію про базову URL-адресу, специфікацію опису дизайну інтерфейсу (IDD) та інші пов'язані метадані.
- Інформація про авторизацію щодо маркерів доступу та підписів
- Будь-яка інша додаткова інформація, необхідна для встановлення взаємодії

Система Orchestration допомагає контролювати розгортання систем і взаємозв'язки між ними. З точки зору SOA, це та система, яка допомагає у створенні інших систем, забезпечуючи напрямок , контроль і координацію.

Крім того, це дозволяє динамічно повторно використовувати існуючі системи та сервіси для створення нових сервісів і функцій.

3.4. Використання доменно спеціалізованої мови для моделювання системи IoT

Arrowhead Framework DSL використовується з метою моделювання систем IoT у Arrowhead Framework за допомогою елементів Framework. Таким чином, модель має бути побудована за допомогою створеного DSL, щоб зрозуміти інтуїтивність моделювання за допомогою DSL. Крім того, створена модель також має бути перевірена в середовищі Arrowhead Framework. Це диктує необхідність генерації коду для побудованої моделі. Крім того, цей згенерований код має бути специфічним для Arrowhead Frame. Крім того, цей згенерований код потрібно перевірити в Arrowhead Framework. У цьому розділі описано модель реєстрації автомобіля, побудовану з використанням елементів Arrowhead Framework DSL, і коду, створеного для цієї моделі.

Для розуміння простоти моделювання за допомогою створеного DSL була створена проста модель реєстрації автомобіля. Для цієї моделі профіль Arrowhead Framework разом із бібліотеками моделей було імпортовано до проекту моделі. Після імпортування елементи в бібліотеках профілів і моделей були доступні для використання.

Нижче наведено короткий опис моделі реєстрації автомобіля. Ця модель складається з локальної хмари, що складається з двох розгорнутих у ній систем. Перша система – це система автопровайдера, яка надає послугу CarRegistration. Друга система – це автомобільна система споживача, яка споживає надану послугу. Сервіс CarRegistration складається з двох операцій, а саме: створення автомобіля та отримання автомобіля. Операція create-car створює набір заданих автомобілів. Операція get-car вибирає ті автомобілі зі списку, які відповідають наданим критеріям.

Для системи постачальника автомобіля створюється діаграма визначення блоку. Стереотип абстрактної системи додано до діаграми, а його ім’я встановлено на систему Car Provider. Потім на діаграму додається стереотип системи, а його ім’я встановлюється на HTTP Car Provider System. Потім встановлюється зв’язок узагальнення між системою Car Provider System і HTTP Car Provider System. На тій самій діаграмі додано стереотип абстрактної служби, а її назву встановлено на службу реєстрації автомобіля. Крім того, на діаграму додається стереотип служби з назвою служби HTTPCarRegistration. Потім операції create-car і get-car додаються до HTTP-CarRegistration Service. Після створення стереотипу послуги в системі створюються порти для кожної послуги, що надається системою. Система Car Provider зі стереотипом Abstract System матиме один порт, відповідний одній абстрактній службі - CarRegistration. Подібним чином система HTTP Car Provider зі стереотипом System матиме один порт, відповідний службі – HTTPCarRegistration.

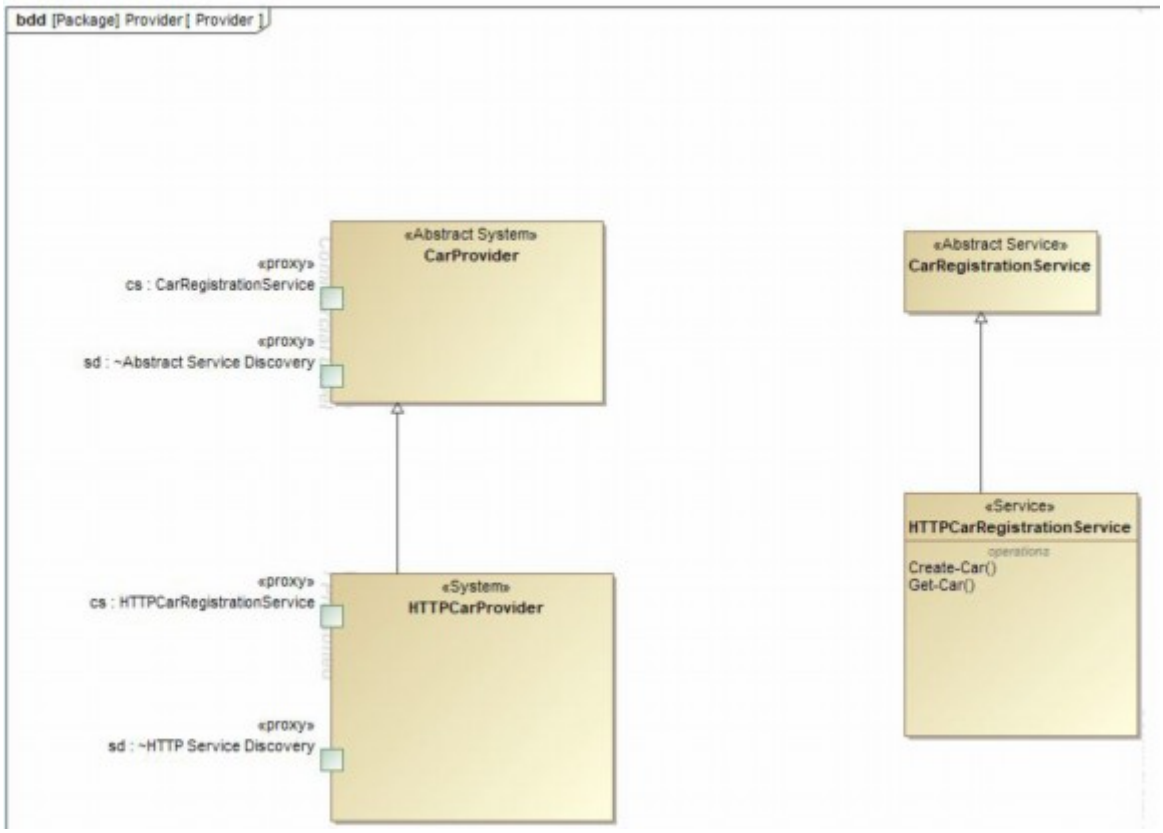

Рис. 3.8. Блок-схема системи постачальника послуг

Блок-схема системи постачальника послуг (автомобіля) зображена на рисунку 3.8. Подібно до системи постачальника автомобіля, для системи споживача автомобіля створено іншу діаграму визначення блоку з двома стереотипами, а саме абстрактною системою та стереотипами системи. Крім того, споживана служба та абстрактна служба створюються як сполучений порт у системі та абстрактній системі відповідно. Блок-схема споживчої системи автомобіля зображена на рисунку 3.9. Третя діаграма визначення блоку створюється для LocalCloud.

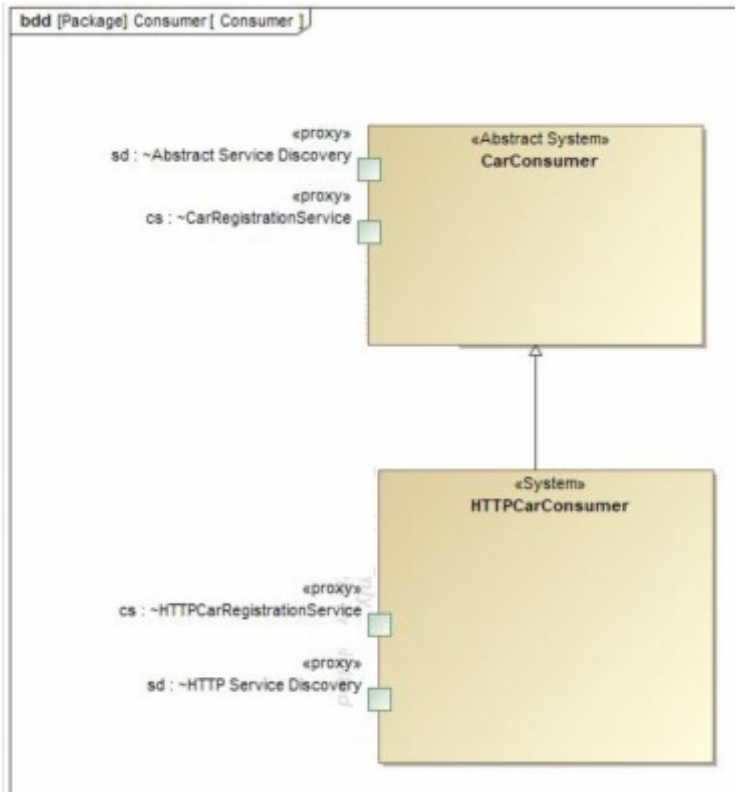

Рис. 3.9. Блок-схема системи споживача послуг

Для локальної хмари створюється внутрішня блок-схема. Внутрішня частина локальної хмари складається з двох систем. Стереотип DeployedSystem додається до діаграми з назвою CarProviderSystem і типом HTTP Car Provider System. З іншого боку, інший стереотип DeployedSystem додається до діаграми з назвою CarConsumerSystem і типом встановлено HTTP Car Consumer System. Сервісні порти в обох системах потім

з’єднуються за допомогою з’єднувача зі стереотипом ServiceExchange, щоб відобразити обмін послугами між двома системами. Ця внутрішня блок-схема локальної хмари зображена на рисунку 3.10.

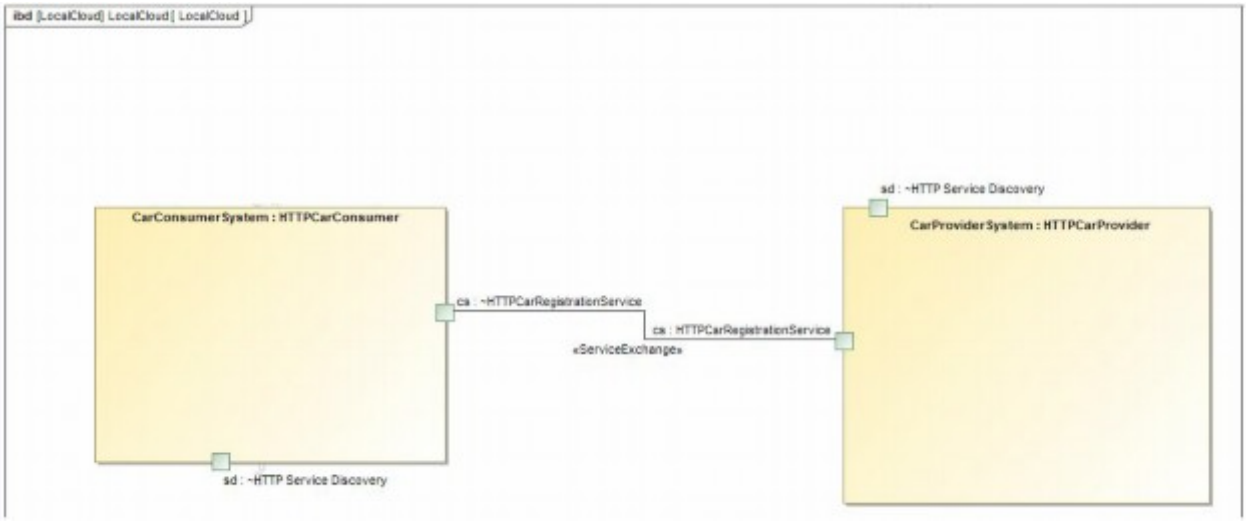

Рис. 3.10. Внутрішня блок-схема локальної хмари

Для генерації спеціального коду Arrowhead Framework з моделей використовується функція генерації шаблону в інструменті. Першим кроком генерації коду є вивчення та розуміння існуючого специфічного коду Arrowhead Framework. Також була обрана мова, якою має бути згенерований код. Для цього призначення було вибрано реалізацію Java специфічного коду Arrowhead Framework.

Функція генерації шаблону в інструменті є рудиментарною і підтримує генерацію коду лише певною мірою. Крім того, для кожного файлу коду, який потрібно згенерувати, слід надати окремий файл шаблону з кодом VTL як вхідні дані для інструменту. Після надання файлу шаблону слід вибрати область моделі, де можна вибрати всю модель або окремий пакет у моделі. Написаний код VTL може бути специфічним для всієї моделі або для пакета в моделі. Щоб зробити цей процес генерації коду ефективним, файли коду, які є загальними та ті, які є специфічними для моделі, були ідентифіковані та розділені. Для цього процесу створення коду було вибрано лише ті розділи у файлі коду, які є специфічними для моделі. Для решти коду у файлі коду

використовувалися шаблони коду. Крім того, код Arrowhead Framework для вибраної демонстраційної моделі автомобіля вже існує. Таким чином, були ідентифіковані елементи Arrowhead Framework і відповідні їм властивості та значення в моделі, які повинні використовуватися в згенерованому коді. Було розроблено код VTL для вилучення властивостей і значень різних елементів моделі в моделі. Залишок коду, згаданого вище, буде шаблонами коду з наданого коду демо-автомобіля, який не був проаналізований движком швидкості. Це був найкращий спосіб генерації коду за допомогою вбудованого інструменту створення шаблонів у Cameo Systems Modeler.

Одним із основних операторів, які використовуються в коді VTL для створення файлів коду, є оператор нерозібраного коду, який включатиме загальний вміст у певний файл коду. Наприклад, у спеціальному файлі коду Ar rowhead Framework більшість файлів загальної бібліотеки, доданих на початку коду, будуть загальними та повторюваними для інших моделей. По суті, у файлі шаблону, наданому як вхідні дані, ці бібліотечні файли будуть додані як нерозібраний код, щоб вони були доступні у згенерованому коді. Приклад нерозібраного коду у VTL наведено в лістингу 3.1 . Вихід цього коду VTL показано в лістингу 3.2 . Таким чином, шаблони коду клієнта з Arrowhead Framework були вказані у файлі шаблону швидкості для створення цих шаблонів коду разом із об’єктами та значеннями, характерними для моделі.

Лістинг 3.1. Код мови шаблону швидкості введення з нерозібраним розділом

```
#[[
import java.util.List;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import eu.arrowhead.client.library.ArrowheadService;
]]#
```

Лістинг 3.2. Вихідний код Arrowhead Framework

```
import java.util.List;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import eu.arrowhead.client.library.ArrowheadService;
```

Лістинг 3.3. Код мови шаблону швидкості введення

```
#literal()
final List<CarRequestDTO> carsToCreate = List.of(
#end
#foreach ($c in $System)
#foreach($vp in $report.filterElement($c.ownedAttribute, [Value Property]))
#if($vp.name)
#if($velocityCount == 1)
#literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "),
#end
#if($velocityCount == 2)
#literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "),
#end
#if($velocityCount == 3)
 #literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "),
#end
#if($velocityCount == 4)
 #literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "));
#end
#end
#end
#end
```

Деякі значення, отримані з моделі за допомогою VTL, включають системні імена та їхні атрибути, такі як адреса та номер порту, атрибути служби, такі як тип кодування, тип протоколу та інші. Зразок коду VTL і відповідний згенерований спеціальний код Arrowhead Framework показано в лістингу 3.3.

Отже, детально описано, як використовувати створений DSL для моделювання робочої прикладної системи Arrowhead Frame. Обговорено різні елементи та діаграми частини моделі і описано функцію генерації шаблону, яка є частиною інструменту Cameo Systems Modeler, і мова VTL, яку він використовує для цього процесу генерації.

3.5. Перевірка системних і сервісних аспектів

Деякі концепції систем і сервісів були визначені в розділі Arrowhead Framework - DSL. Це основні концепції, які необхідно перевірити для функціональної системи та служб, сумісних із Arrowhead Framework. Ця перевірка певним чином підтвердить, чи були прийняті рішення щодо дизайну правильними. Деякі з визначених концепцій полягають у тому, що система повинна мати можливість надавати одну або більше послуг. Це підтверджено, і це показано на рисунку 3.11.

Рис. 3.11. Одна система, що надає одну або декілька послуг

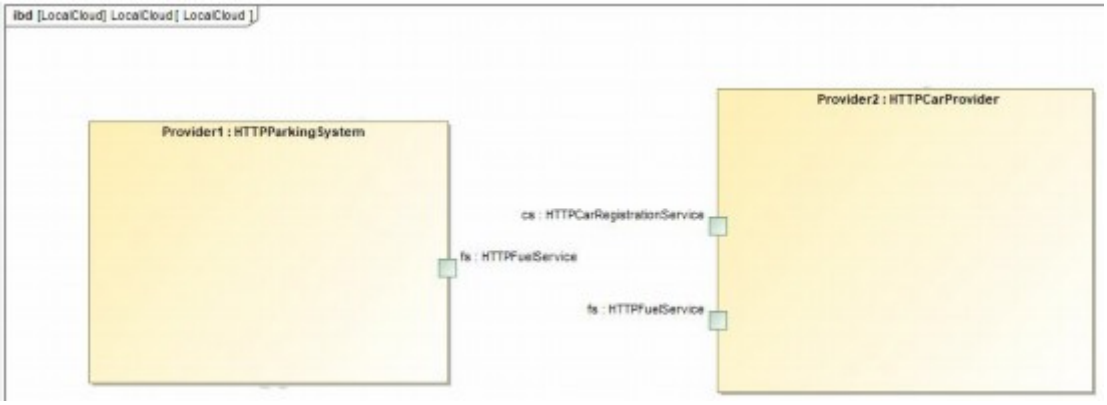

Рис. 3.12. Одна послуга, що надається більш ніж однією системою

Крім того, в концепції послуги було зазначено, що послуга може надаватися однією або кількома системами. Це також підтверджено, і це показано на рисунку 3.12.

Мова обмежень об’єктів (OCL) — це тип мови обмежень, який використовується для визначення правил у моделях UML і SysML. Обмеження використовуються в DSL для покращення сформованості моделей. OCL містить інваріанти, попередні та післяумови, вирази тіла та інші оператори для визначення обмежень у моделі. Він переважно використовується для визначення незмінних умов для стереотипів у цьому завданні. Першим кроком для використання OCL у моделі є визначення стереотипів, до яких необхідно застосувати обмеження. Під час ідентифікації розробляються обмеження, які необхідно визначити для стереотипів. Потім ці обмеження розробляються на спеціальній мові обмежень. Потім обмеження перевіряються в інструменті моделювання.

Наступним кроком перевірки є перевірка згенерованого коду з модуля шаблону звіту в інструменті. Файл шаблону надається як вхідні дані інструменту для створення файлу коду. Цей крок повторювався, доки не було згенеровано всі файли коду. Цей проект Java, що складається зі згенерованих файлів коду, було виконано в середовищі Arrowhead Framework.

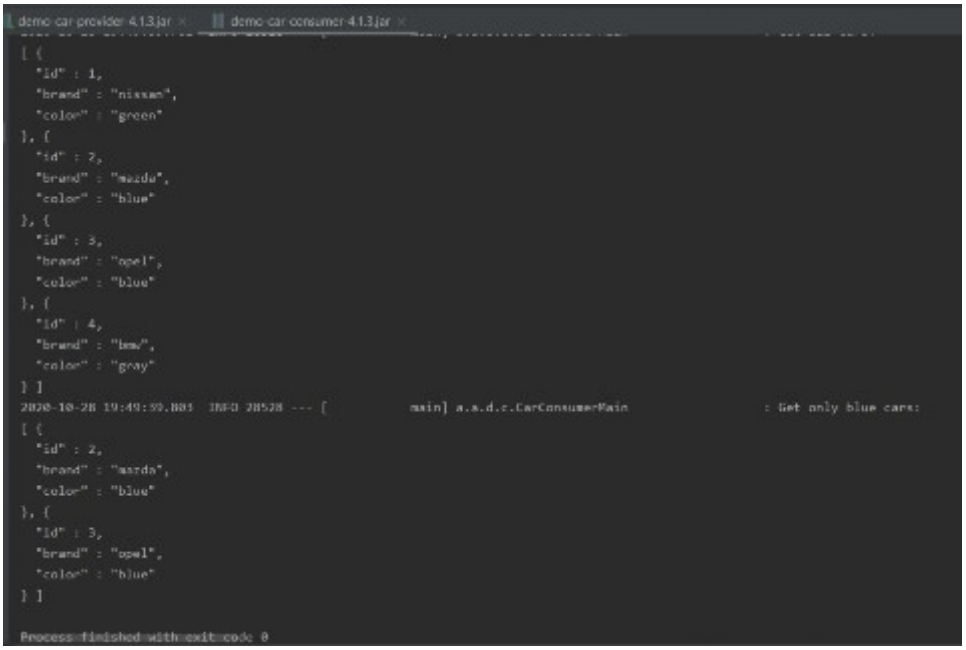

Рис. 3.13. Перевірка операцій create- car і get-car

Під час виконання системи споживача Car перевіряються операції create-car і get-car. Операція create-car створює набір марок автомобілів і кольорів, указаних у моделі. Крім того, операція get-car отримує набір марок автомобілів із вказаним синім кольором. Ця перевірка операцій create-car і get-car у середовищі Arrowhead Framework зображена на рисунку 3.13.

Висновки до розділу

У цьому розділі детально розглянуто методологію розробки предметно-орієнтованої мови (DSL). Представлено аналіз створених у DSL елементів, логіку їх формування, їх значення та можливості застосування у моделі. Окрему увагу було приділено елементам UML, які слугували основою для розширення елементів DSL. Також визначено бібліотеки моделей для двох основних категорій систем: обов’язкових базових систем та базових систем підтримки, побудованих на основі елементів профілю. Наступний розділ присвячено застосуванню профілю для побудови моделі та процесу генерації коду на основі конкретної моделі.

Було здійснено перевірку розробленого DSL для Arrowhead Framework. DSL було верифіковано за допомогою обмежень OCL, а кроки перевірки в інструменті Cameo Systems Modeler детально описані. Загалом, перевірка різних аспектів системи та сервісів, а також згенерований код підтверджують коректність DSL для Arrowhead Framework, розробленого в рамках даного дослідження.

ВИСНОВКИ

У результаті проведеного дослідження були розглянуті моделі та методи доменно специфікованої мови для застосувань Інтернету речей. Першим важливим кроком є проведення ретельного огляду літератури, щоб глибше зрозуміти концепції Arrowhead Framework, оцінити його значення та користь з точки зору сумісності, інтеграції складних систем автоматизації та спрощення впровадження. Це дослідження також допоможе усвідомити потребу в створенні доменно-специфічної мови (DSL), її застосування та ефективність. Огляд наявних DSL для цієї архітектури дозволить вивчити існуючі рішення та подальші напрями досліджень.

Другим важливим етапом є дослідження різних мов моделювання для систем, методів створення DSL, процесів його побудови та методів перевірки. Це допоможе вибрати найбільш підходящі інструменти та підходи для розробки власного DSL.

На основі цих досліджень буде визначено функціональні вимоги до інструменту моделювання. Оцінка та вибір інструментів дозволить знайти найкращий інструмент для підтримки вибраної мови моделювання, забезпечення можливості побудови DSL та генерації настроюваного коду. Важливою умовою є також можливість експорту цього коду для подальшого використання.

Далі було уточнено основні концепції Arrowhead Framework, та створено базу даних для управління зібраною інформацією про елементи цієї архітектури, зокрема концепції, фізичні та логічні сутності, зв'язки та атрибути. Ці елементи використовуватимуться для створення бібліотек профілів і моделей. На основі зібраної інформації розроблено DSL, який інтегрує елементи Arrowhead Framework. Наступним кроком була ідентифікація конкретного застосування, для якого необхідно створити модель, використовуючи елементи профілю.

Далі важливо визначити шаблонну мову для генерації коду, розуміти її концепції та синтаксис, а також розробити код для генерації зі створеної моделі. Це дозволить реалізувати кінцевий продукт, інтегрований з Arrowhead Framework.

На завершальному етапі розроблений DSL буде перевірено за допомогою обмежень OCL, а згенерований код протестовано в середовищі Arrowhead Framework. Цей процес підтвердить правильність створеної моделі та DSL, що використовується для її побудови.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Fuentes-Fernández, L. and Vallecillo-Moreno, A., 2004. An introduction to UML profiles. UML and Model Engineering, 2(6-13), p.72.
2. Arrowhead Project, Arrowhead Framework-This is it, Arrowhead Project, viewed February 2020, <https://arrowhead.eu/arrowheadframework/this-is-it>
3. Varga, P., Blomstedt, F., Ferreira, L.L., Eliasson, J., Johansson, M., Delsing, J. and de Soria, I.M., 2017. Making system of systems interoperable–The core components of the arrowhead framework. Journal of Network and Computer Applications, 81, pp.85-95.
4. Delsing, J. ed., 2017. Iot automation: Arrowhead framework. CRC Press.
5. IBM Knowledge Center, Rational Rhapsody 8.4.0, IBM, viewed March 2020, URL <https://www.ibm.com/support/knowledgecenter/SSB2MU_8.4.0/com.ibm.rhp.homepage.doc/helpindex_rhapsody.html>
6. Blomstedt, F., Ferreira, L.L., Klisics, M., Chrysoulas, C., de Soria, I.M., Morin, B., Zabasta, A., Eliasson, J., Johansson, M. and Varga, P., 2014, October. The arrowhead approach for SOA application development and documentation. In IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society (pp. 2631-2637). IEEE.
7. Delligatti, L., 2013. SysML distilled: A brief guide to the systems modeling language. Addison-Wesley.
8. SysML Org., What is SysML? Who created SysML?, SysML, SysML Org., viewed March 2020, <https://sysml.org/>
9. ISO, 2017, ISO/IEC 19514:2017 [ISO/IEC 19514:2017] Information technology — Object management group systems modeling language (OMG SysML), ISO, viewed March 2020, <https://www.iso.org/standard/65231.html>
10. Nagorny, K., Harrison, R., Colombo, A.W. and Kreutz, G., 2013, July. A formal engineering approach for control and monitoring systems in a

service-oriented environment. In 2013 11th IEEE International Conference on Industrial Informatics (INDIN) (pp. 480-487). IEEE.

11. Arrowhead Project, 2019, Support Core Systems and Services, Fusion Forge, viewed February 2020, <https://forge.soa4d.org/plugins/mediawiki/wiki/arrowheadf/index.php/Support_Core_Systems_and_Servicess>
12. Lollini, P., Mori, M., Babu, A. and Bouchenak, S., 2016. Amadeos sysml profile for sos conceptual modeling. In Cyber-Physical Systems of Systems (pp. 97-127). Springer, Cham.
13. IBM Knowledge Center, Rational Software Architect 9.7.0, IBM, viewed March 2020, URL <https://www.ibm.com/support/knowledgecenter/SS8PJ7_9.7.0/com.ibm.rsa_base.nav.doc/topics/c_node_overview.html>
14. Arrowhead Project, Arrowhead Consortia, Github, viewed March 2020, <https://github.com/arrowhead-f/>
15. No Magic Inc., MagicDraw 18.5 Documentation, MagicDraw, viewed March 2020, URL <https://docs.nomagic.com/display/MD185/MagicDraw+Documentatio>
16. Arrowhead Project, Arrowhead-f/core-java-spring, GitHub Repository, viewed March 2020, <https://github.com/arrowhead-f/core-java-spring/blob/master/documentation/development/developer_tmp.txt>
17. Arrowhead Project IncQuery Labs, IncQueryLabs/arrowhead-tools, GitHub Repository, viewed March 2020, <https://github.com/IncQueryLabs/arrowhead-tools>
18. Wikipedia, Profile(UML), Wikipedia, viewed March 2020, <https://en.wikipedia.org/wiki/Profile(UML)>
19. UML Diagrams Org., UML Profile, UML Diagrams Org., viewed January 2020, <https://www.uml-diagrams.org/profile.html>
20. Alhir, S.S., 2006. Guide to Applying the UML. Springer Science Business Media.
21. Object Management Group. OMG Systems Modeling Language TM(OMG SYSML), 2019. URL <https://www.omg.org/spec/SysML/1.6/PDF>

22. Object Management Group. OMG Unified Modeling Language TM (OMG UML), 2015. URL <https://www.omg.org/spec/UML/2.5/PDF>
23. Wikipedia, Systems Modeling Language, Wikipedia, viewed September 2020, <https://en.wikipedia.org/wiki/Systems_Modeling_Language>
24. Palm, E., Bodin, U. and Schelén, O., 2020. Kalix: A Java 11 Library for DevelopingEclipse Arrowhead System-of-Systems. In 2020 IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, September 8-11.
25. Bean, J., 2009. SOA and web services interface design: principles, techniques, and standards. Morgan Kaufmann.
26. Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. ACM Computing Surveys (CSUR), 37(4), 316-344.
27. Kelly, S., & Tolvanen, J.-P. (2008). Domain-specific modeling: Enabling full code generation. John Wiley & Sons.
28. Fowler, M. (2010). Domain-specific languages. Addison-Wesley.
29. Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: An annotated bibliography. ACM Sigplan Notices, 35(6), 26-36.
30. Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2008). EMF: Eclipse Modeling Framework. Addison-Wesley.
31. Harel, D., & Gery, E. (1997). Executable object modeling with statecharts. Computer, 30(7), 31-42.
32. Bjoerner, D. (2006). Domain-specific languages for system modeling. Formal Methods at the Crossroads: From Panacea to Foundational Support, 98-124.
33. White, J., Benavides, D., Dougherty, B., & Schmidt, D. C. (2010). Automated diagnosis of feature model configurations. Journal of Systems and Software, 83(7), 1094-1107.
34. Zeng, D., Guo, S., Cheng, Z., & Zhang, J. (2011). The web of things: A survey. Journal of Communications, 6(6), 424-438.

35. Abie, H., Balasingham, I. (2012). Risk-based adaptive security for the internet of things in eHealth. Proceedings of the 7th International Conference on Body Area Networks, 269-275.
36. Saxena, N., Roy, A., & Kim, T. H. (2016). Efficient IoT gateway over 5G wireless: A comprehensive survey. IEEE Internet of Things Journal, 4(5), 90-108.
37. Waher, P. (2015). Learning Internet of Things. Packt Publishing Ltd.
38. Aazam, M., & Huh, E. N. (2016). Fog computing and smart gateway based communication for cloud of things. Proceedings of the IEEE International Conference on Future Internet of Things and Cloud, 464-470.
39. Haller, S., Karnouskos, S., & Schroth, C. (2009). The Internet of Things in an enterprise context. Future Internet Symposium, 14-28.
40. Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645-1660.
41. Qin, Y., Gong, X., & Wang, H. (2013). A domain-specific modeling approach for IoT middleware. International Journal of Distributed Sensor Networks, 2013, 1-12.
42. Patel, K. K., & Patel, S. M. (2016). Internet of Things-IoT: Definition, characteristics, architecture, enabling technologies, application & future challenges. International Journal of Engineering Science and Computing, 6(5), 6122-6131.
43. Weyrich, M., & Ebert, C. (2016). Reference architectures for the Internet of Things. IEEE Software, 33(1), 112-116.
44. Belli, F., Beyazıt, M., & Linschulte, M. (2016). Model-based testing of Internet of Things systems: A survey. IEEE Internet of Things Journal, 3(6), 645-658.
45. Rahmani, A. M., Liljeberg, P., & Preden, J. S. (2018). Fog computing in the Internet of Things: Intelligence at the edge. Springer.

46. Shafagh, H., & Hithnawi, A. (2016). Poster: Security in the Internet of Things. Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, 305-306.
47. Serpanos, D., & Wolf, M. (2017). Internet of Things (IoT) systems: Architectures, algorithms, methodologies. Springer.
48. Margaria, T., & Steffen, B. (2000). Lightweight Co-design flow using domain-specific languages. Proceedings of the IEEE International Conference on Computer Design, 115-120.
49. Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. IEEE Computer, 39(2), 25-31.
50. Charif, H., & Sabri, A. (2019). Domain-specific language design for IoT applications: A review. International Journal of Computer Applications, 178(17), 1-8.