МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 18.00.00.000 ПЗ

Група ІПм-23-2

Павликівський Любомир

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Павликівський Любомир Миколайович

(прізвище, ім'я, по батькові)

УДК 004.9

(індекс)

МАГІСТЕРСЬКА РОБОТА

Методологія визначення шару конверсії між рівнем моделей та рівнем імплементації

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Павликівський Л.М.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Вовк Роман Богданович, 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

Павликівському Любомиру Миколайовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Методологія визначення шару конверсії між рівнем моделей та рівнем імплементації”

керівник проекту (роботи) Вовк Роман Богданович, 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 технологій візуалізації моделей

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

1. Дослідження предметної області моделювання систем на основі перетворення моделей
2. Дослідження методів для побудови шару конверсії рівнів моделей та рівнів імплементації
3. Побудова архітектури системи для перетворення моделі
4. Розробка методології визначення шару конверсії між рівнем моделей та рівнем імплементації

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

1. Зв'язок між UML і SysML (рис. 1.1)
2. Інтерфейс середовища проектування IBM Rational Rhapsody (рис. 1.2)
3. Інтерфейс інструменту 3D-візуалізації Unity (рис. 1.3)
4. Системне визначення шару конвертації (рис. 1.4)
5. Стислий огляд концепцій, пов'язаних із системою (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 та алгоритмів предметної області	29.09.2024	виконано
3	Дослідження предметної області моделювання систем на основі перетворення моделей	15.10.2024	виконано
4	Дослідження методів для побудови шару конверсії рівнів моделей та рівнів імплементації	08.11.2024	виконано
5	Побудова архітектури системи для перетворення моделі	20.11.2024	виконано
6	Розробка методології визначення шару конверсії між рівнем моделей та рівнем імплементації	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 80 с., 53 рис., 53 джерела.

Тема: Методологія визначення шару конверсії між рівнем моделей та рівнем імплементації

Об'єкт дослідження: процес моделювання систем та інтеграції моделей з візуалізацією у 3D-середовищі.

Мета роботи: розробка методології визначення шару конверсії між рівнем моделей та рівнем імплементації, що забезпечує інтеграцію модельно-орієнтованої системної інженерії з візуалізацією у середовищі Unity для підвищення ефективності проектування та реалізації складних систем.

Предмет дослідження: методи та підходи до перетворення моделей для створення інтерактивних візуалізацій систем.

Результати дослідження

В роботі розроблено методологію визначення шару конверсії між моделями та візуалізацією в середовищі Unity, яка дозволяє автоматизувати процес передачі та перетворення даних між IBM Rhapsody та Unity. Визначено ефективні підходи до обробки та абстрагування даних, що сприяють підвищенню точності візуалізації та інтуїтивного розуміння моделей.

Висновок

Запропоновані в роботі методи та концепція дозволяють знизити складність процесу інтеграції між моделями та їх реалізацією у вигляді інтерактивних візуалізацій.

МОДЕЛЬНО-ОРІЄНТОВАНА СИСТЕМНА ІНЖЕНЕРІЯ, UNITY, КОНВЕРСІЯ МОДЕЛЕЙ, ВІЗУАЛІЗАЦІЯ, UML, 3D-ВІЗУАЛІЗАЦІЯ, ІНТЕГРАЦІЯ МОДЕЛЕЙ, ПЕРЕТВОРЕННЯ МОДЕЛЕЙ

ABSTRACT

Master Thesis: 87 pp., 53 fig., 55 sources.

Thesis Subject: Methodology for determining the conversion layer between the level of models and the level of implementation

The object of research: the process of modeling systems and integrating models with visualization in a 3D environment.

The purpose of the work: development of a methodology for determining the conversion layer between the level of models and the level of implementation, which ensures the integration of model-oriented system engineering with visualization in the Unity environment to increase the efficiency of the design and implementation of complex systems.

Research subject: methods and approaches to transforming models to create interactive visualizations of systems.

Research results

The work has developed a methodology for defining the conversion layer between models and visualization in the Unity environment, which allows automating the process of data transfer and conversion between IBM Rhapsody and Unity. Effective approaches to processing and abstracting data, which contribute to increasing the accuracy of visualization and intuitive understanding of models, have been determined.

Conclusion

The methods and concept proposed in the work allow reducing the complexity of the process of integration between models and their implementation in the form of interactive visualizations.

MODEL-BASED SYSTEMS ENGINEERING, UNITY, MODEL CONVERSION, VISUALIZATION, UML, 3D VISUALIZATION, MODEL INTEGRATION, MODEL TRANSFORMATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

MDD - Model-Driven Development

SysML - System Modeling Language

MDE - Model-driven engineering

MDSE - Model Driven System Engineering

OMG - Object Managment Group

INCOSE - International Council on Systems Engineering

SE DSIG - Systems Engineering Domain Special Interest Group

BDD - Block De_nition diagram

IBD - Internal block diagram

SoS - System of Systems

ВСТУП

Актуальність теми дослідження.

У сучасній інженерії постійно зростає складність проектування та розробки систем, що вимагає використання більш ефективних підходів до моделювання та візуалізації. Модельно-орієнтована системна інженерія (MDE) дозволяє створювати формалізовані моделі складних систем, що значно спрощує процеси проектування та тестування. Проте виникає необхідність у побудові зв'язку між створеними моделями та їх подальшою реалізацією в інтерактивних середовищах, таких як Unity. Це дозволяє здійснити наочне уявлення функціонування системи на основі моделей та забезпечити ефективну інтеграцію між інструментами моделювання та 3D-візуалізації. Актуальність дослідження обумовлена потребою в розробці ефективних методів і підходів до перетворення моделей у віртуальне середовище, що сприяє зменшенню ризиків та підвищенню точності розробки складних систем.

Важливим аспектом для інженерів та розробників є можливість перегляду та оцінки створених моделей у вигляді візуально зрозумілих прототипів та симуляцій. Технології 3D-візуалізації, зокрема середовище Unity, надають можливість створення інтерактивних середовищ, де моделі можуть бути представлені у зрозумілому тривимірному форматі. Це дозволяє знизити ризики на етапах впровадження, забезпечити точніший контроль за розробкою та полегшити комунікацію між розробниками та іншими учасниками процесу.

Актуальність дослідження полягає у вирішенні проблеми ефективної інтеграції модельно-орієнтованих моделей з інструментами візуалізації, що є ключовим фактором для забезпечення наочності та доступності створених систем. Розробка методології перетворення моделей з IBM Rhapsody до Unity сприяє зниженню складності процесів проєктування та реалізації, що важливо для скорочення витрат часу та ресурсів на розробку. Це дозволяє

підвищити конкурентоспроможність розробок та забезпечити якісніший продукт за рахунок оптимізації процесів створення і тестування систем на основі моделей.

Крім того, дослідження відповідає потребам сучасного ринку у забезпеченні гнучкої інтеграції різних програмних платформ, що використовується у проєктуванні та візуалізації складних систем. Розробка інноваційних підходів до візуалізації моделей дозволяє розширити можливості моделювання, зробити процеси розробки більш адаптивними та гнучкими. Це має особливе значення в умовах зростаючої потреби в технологіях віртуальної реальності та їх застосуванні у процесі навчання, симуляцій та випробувань.

Мета дослідження - розробка методології визначення шару конверсії між рівнем моделей та рівнем імплементації, що забезпечує інтеграцію модельно-орієнтованої системної інженерії з візуалізацією у середовищі Unity для підвищення ефективності проектування та реалізації складних систем.

Об’єкт дослідження - процес моделювання систем та інтеграції моделей з візуалізацією у 3D-середовищі.

Предмет дослідження - методи та підходи до перетворення моделей для створення інтерактивних візуалізацій систем.

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

- Дослідити особливості модельно-орієнтованої системної інженерії та інструментів моделювання, таких як IBM Rhapsody та Unity.
- Проаналізувати методи та алгоритми побудови шару конверсії між рівнем моделей та імплементації.

- Розробити архітектуру системи для перетворення моделей з IBM Rhapsody до середовища Unity.
- Описати процеси обробки та передачі даних між компонентами системи.
- Реалізувати методику візуалізації моделей у середовищі Unity на основі створених моделей.
- Оцінити ефективність розробленої методології та визначити її застосування в інженерних проєктах

Методи дослідження.

- Математичне моделювання для аналізу структур та процесів.

- Аналіз та порівняння методів побудови діаграм та моделей на основі SysML та UML.

- Розробка та тестування програмного забезпечення для реалізації перетворення моделей у середовищі Unity.

- Візуалізація даних та моделей за допомогою 3D-графіки.

Наукова новизна отриманих результатів.

Розроблено методологію визначення шару конверсії між моделями та візуалізацією в середовищі Unity, яка дозволяє автоматизувати процес передачі та перетворення даних між IBM Rhapsody та Unity. Визначено ефективні підходи до обробки та абстрагування даних, що сприяють підвищенню точності візуалізації та інтуїтивного розуміння моделей. Запропоновані методи дозволяють знизити складність процесу інтеграції між моделями та їх реалізацією у вигляді інтерактивних візуалізацій

Практичне значення магістерської роботи.

Розроблена методологія може бути використана для створення інтерактивних візуалізацій складних інженерних систем у різних галузях та освітніх проєктах. Це дозволяє підвищити якість та швидкість розробки,

знизити витрати на тестування та зменшити ризики при впровадженні систем. Візуалізація моделей у 3D-середовищі забезпечує інтуїтивне сприйняття складних концепцій та спрощує процеси комунікації між розробниками та зацікавленими сторонами

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 80 сторінок, і містить 53 рисунки, перелвк використаних джерел із 53 позицій.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ МОДЕЛЮВАННЯ СИСТЕМ НА ОСНОВІ ПЕРЕТВОРЕННЯ МОДЕЛЕЙ

1.1. Опис теми дослідження

З недавнім технологічним прогресом системи стають більшими та складнішими. Це призводить до того, що розробник не може зрозуміти складні системи. Системи сьогодні часто виражаються за допомогою мови моделювання систем (SysML). Інструменти CASE, в яких розробляються моделі SysML, пропонують розробнику двовимірний погляд на моделі. Це може бути обмеженням для розуміння складної системи. Системна модель має кілька діаграм, які всі взаємопов'язані, це не може бути належним чином візуалізовано за допомогою цього двовимірного вигляду.

Цей проект представляє спосіб перетворення моделі SysML на 3D-представлення. Це буде зроблено за допомогою так званого "шару конвертації" між інструментом CASE SysML та інструментом 3D-візуалізації. Ми представимо прототип, щоб показати, що можна зробити з використанням конвертованих даних моделі SysML у 3D-середовищі. Кінцева мета полягає в тому, щоб представити структуру та поведінку моделей SysML у 3D-середовищі, щоб складність моделей могла бути зрозуміла ефективніше, ніж у 2D-середовищі.

За останні пару років моделі набули більшого значення в інженерному світі. Розробка на основі моделі (MDD) — це спосіб роботи, який починається з вищого рівня абстракції. Розробка на основі моделі — це швидкий і економічно ефективний підхід, тому MDD швидко стає стандартним способом розробки. MDD використовується в багатьох різних дисциплінах, наприклад, розробка програмного забезпечення, автомобільна та системна інженерія.

Для розробки моделей використовуються засоби CASE, що розшифровується як Computer-aided software engineering tools. Прикладами деяких інструментів CASE є: TogetherSoft, MySQL Workbench і IBM Rational Rhapsody. Ці інструменти часто великі та складні.

Зростання технологічного прогресу призводить до складних систем. Оскільки ці системи настільки складні, моделі цих систем неминуче стають великими та складнішими. Ця складність може зробити системні моделі складними для розуміння, що може призвести до провалу проекту.

Мова моделювання системи (SysML) — це складна мова моделювання загального призначення для проектування систем. Він підтримує широкий спектр різноманітних діаграм із окремим використанням. SysML тісно пов’язаний з уніфікованою мовою моделювання (UML), оскільки використовує деякі з її основних діаграм. Зв'язок між SysML і UML представлений на рисунку 1.1. SysML використовує діаграми з UML і розширено додатковими діаграмами. Ці діаграми часто стають складними, тому важливо підтримувати структуру та поведінку моделей керованими.

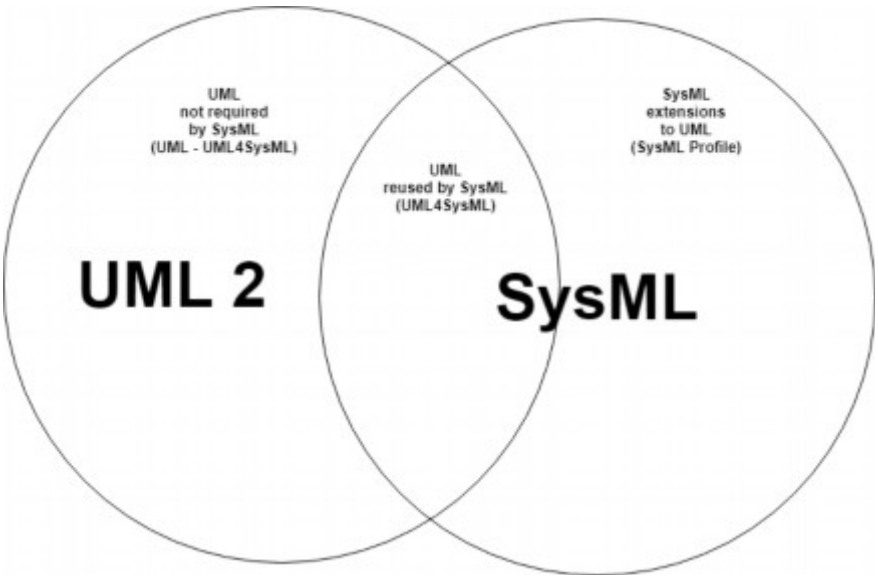


Рис. 1.1. Зв'язок між UML і SysML

Інструменти CASE надають розробнику 2-вимірний вигляд для ілюстрації моделей. Цього часто недостатньо для розуміння всієї структури

та поведінки системи. Мета цього проекту — створити структуру між інструментом CASE та інструментом 3D-візуалізації, щоб краще розуміти складність моделей SysML.

1.2. Представлення інструментів моделювання як об’єкту дослідження

Цей проект вивчатиме можливості та корисність візуалізації моделей SysML у інструменті 3D-візуалізації. Для цього слід вибрати інструмент CASE, який підтримує моделі SysML. У цьому проекті використовується IBM Rational Rhapsody.

1.2.1. Середовище проектування IBM Rational Rhapsody

IBM Rational Rhapsody — це середовище для спільного проектування та розробки для системних інженерів і розробників програмного забезпечення для створення, тестування та документування систем і програмного забезпечення в реальному часі або вбудованих систем.

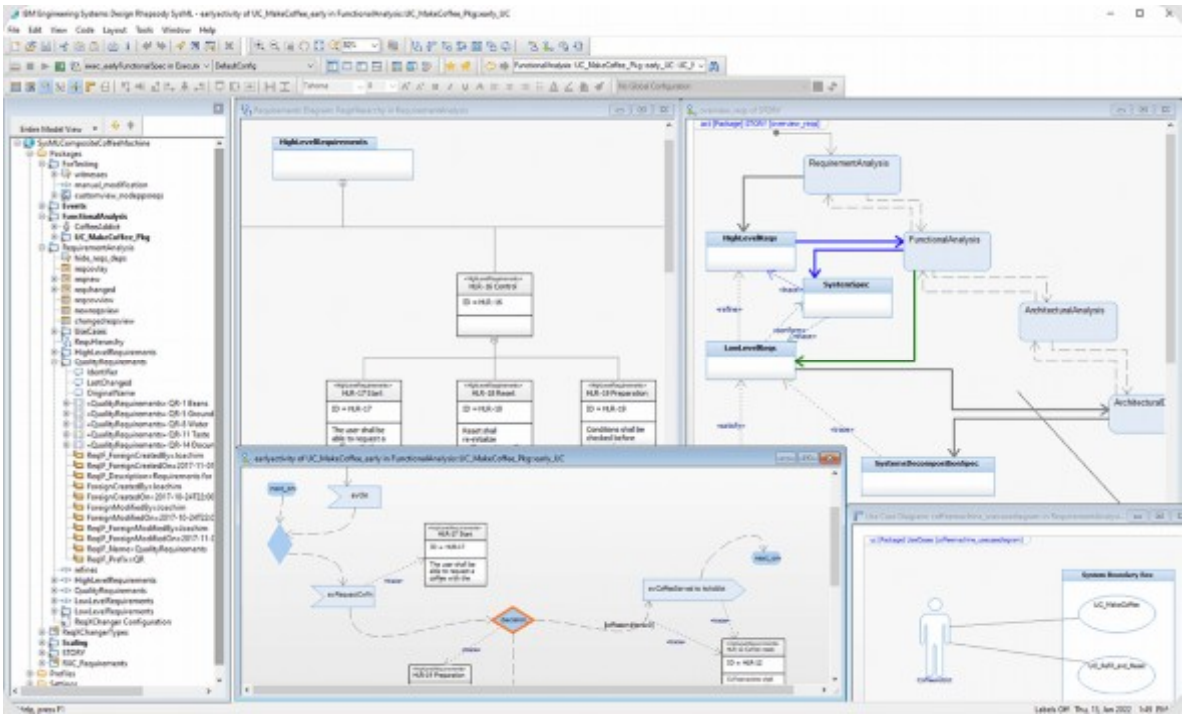


Рис. 1.2. Інтерфейс середовища проектування IBM Rational Rhapsody

IBM Rational Rhapsody — це інструмент CASE (Computer-Aided Software Engineering), який використовується для розробки вбудованих систем. Він підтримує різні методики розробки, включаючи UML (Unified Modeling Language), SysML (Systems Modeling Language) та SDL (Specification and Description Language). Rational Rhapsody пропонує широкий спектр функцій, які допомагають розробникам створювати, аналізувати, тестувати та документувати вбудовані системи.

Ключові особливості IBM Rational Rhapsody:

- Підтримка різних методів розробки: Rational Rhapsody підтримує UML, SysML та SDL, що дозволяє розробникам використовувати різні підходи до моделювання та аналізу систем.

- Інтегроване середовище розробки: Rational Rhapsody пропонує інтегроване середовище розробки, яке включає в себе редактор діаграм, редактор коду, дебагер та інші необхідні інструменти.

- Генерація коду: Rational Rhapsody може автоматично генерувати код на основі моделей, що скорочує час розробки та знижує ризик помилок.

- Аналіз моделей: Rational Rhapsody дозволяє проводити різні аналізи моделей, такі як аналіз залежностей, аналіз впливу змін та аналіз продуктивності.

- Тестування моделей: Rational Rhapsody підтримує різні методи тестування моделей, включаючи тестування на відповідність вимогам та тестування на продуктивність.

- Інтеграція з іншими інструментами: Rational Rhapsody може бути інтегрований з іншими інструментами IBM Rational, такими як Rational DOORS та Rational Team Concert, що дозволяє забезпечити повний життєвий цикл розробки.

IBM Rational Rhapsody є потужним інструментом для розробки вбудованих систем, який пропонує широкий спектр функцій та переваг. Він може допомогти розробникам підвищити продуктивність, якість та ефективність своєї роботи

1.2.2. Інструментом 3D-візуалізації Unity

Інструментом 3D-візуалізації, який використовується в цьому проекті, є Unity, яка є платформою розробки в реальному часі. Unity переважно використовується для ігор, але відповідає вимогам, необхідним для 3D-візуалізації моделі SysML. Unity – це один з найпопулярніших і універсальних ігрових двигунів, який широко використовується не тільки для розробки відеоігор, але й для створення інтерактивних досвідів, симуляцій, віртуальної та доповненої реальності. Його потужні можливості 3D-візуалізації роблять його незамінним інструментом для кого кола розробників.

Основні сфери застосування Unity:

- Розробка відеоігор;
- Віртуальна реальність (VR) і доповнена реальність (AR);
- Симуляції; прототипування.

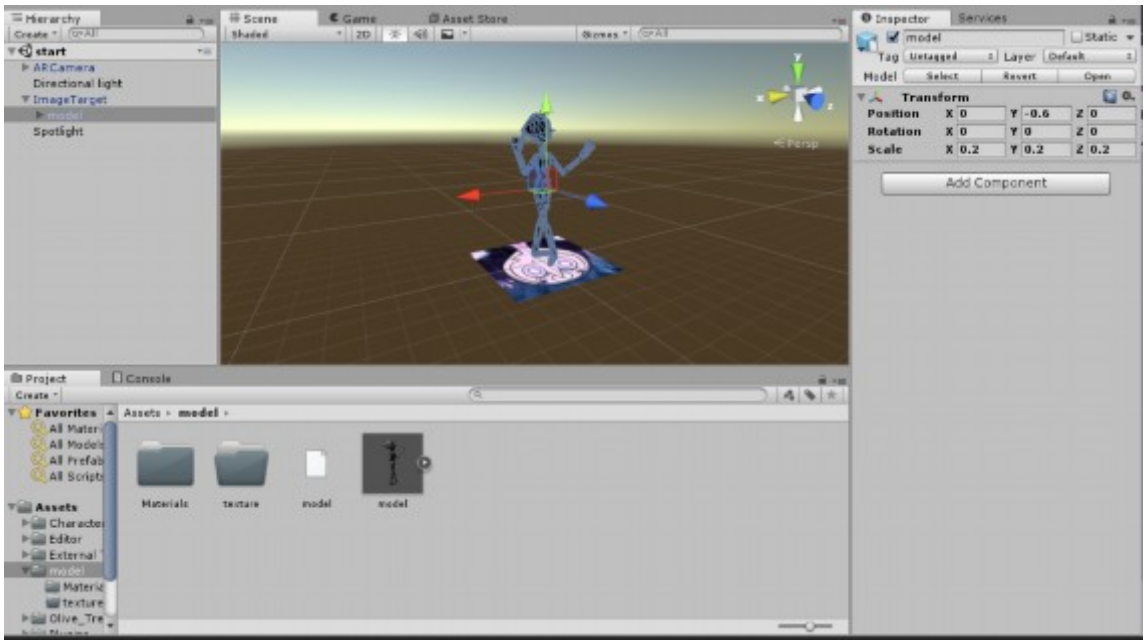


Рис. 1.3. Інтерфейс інструменту 3D-візуалізації Unity

Unity – це крос-платформний ігровий двигун, що дозволяє розробникам створювати 2D і 3D ігри та інтерактивні досвіді для різних платформ, включаючи ПК, мобільні пристрої, консолі, веб-браузери та VR/AR пристрої.

Ключові особливості Unity для 3D-візуалізації:

- Потужний редактор: Інтуїтивно зрозумілий редактор Unity дозволяє легко створювати 3D-сцени, моделювати об'єкти, налаштовувати освітлення, матеріали та ефекти.

- Різноманітні інструменти моделювання: Unity пропонує широкий спектр інструментів для створення 3D-моделей, включаючи інструменти для скульптування, текстурування та анімації.

- Фізика: Реалістична фізика в Unity дозволяє створювати динамічні ігри та симуляції з реалістичними взаємодіями між об'єктами.

- Скриптування: За допомогою мови програмування C# розробники можуть створювати складну поведінку ігрових персонажів, взаємодії з оточенням та інші інтерактивні елементи.

- Висока продуктивність: Unity оптимізований для забезпечення високої продуктивності на різних платформах, що дозволяє створювати вимогливі до ресурсів 3D-ігри.

- Підтримка VR/AR: Unity надає потужні інструменти для розробки VR і AR додатків, дозволяючи створювати інтерактивні досвіді, які стирають межі між віртуальним і реальним світом.

- Активи та ресурси: Величезний магазин Asset Store пропонує безліч готових активів, моделей, скриптів та інших ресурсів, що прискорюють процес розробки.

- Крос-платформна розробка: Unity дозволяє легко експортувати свої проекти на різні платформи, що дозволяє досягти широкої аудиторії.

За допомогою цих інструментів проект можна розділити на три етапи:

• отримання даних про моделі SysML із бази даних IBM Rational Rhapsody.

• візуалізація отиманих моделей SysML в Unity.

• представлення корисності досягнутої 3D візуалізації.

Перші два кроки пов’язані з розробленою системою, яку ми називаємо рівнем перетворення. Огляд рівня перетворення можна побачити на рисунку 1.4.

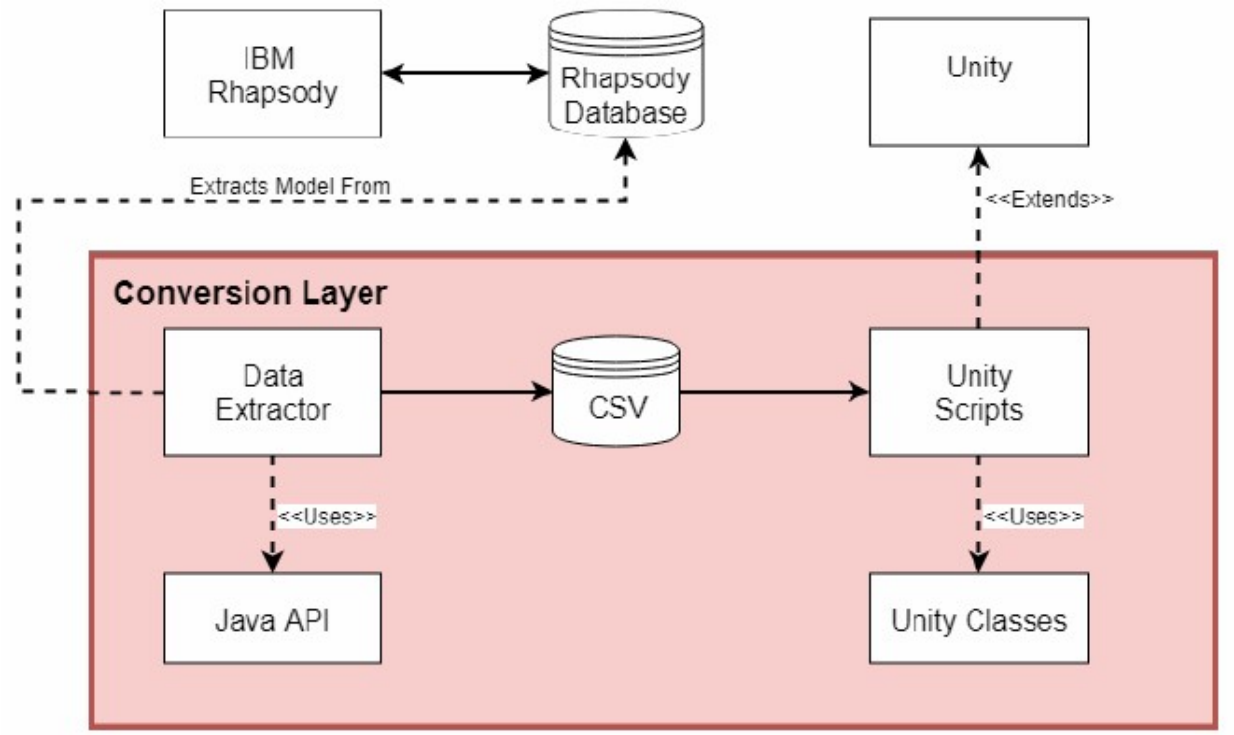


Рис. 1.4. Системне визначення шару конвертації

Моделі SysML створюються в IBM Rhapsody, ці моделі зберігаються в базі даних Rhapsody. Щоб отримати дані про моделі SysML, ми створюємо Data Extractor. Цей екстрактор даних використовує дані з бази даних Rhapsody. IBM Rhapsody має розширену бібліотеку API, яку можна використовувати для отримання доступу до моделей SysML. API досить складний і не призначений безпосередньо для вилучення даних, а призначений для маніпулювання запущеною програмою IBM Rhapsody.

Data Extractor зберігає свої дані в кількох файлах із значеннями, розділеними комами (CSV) . Після чого витягнуті дані візуалізуються в Unity. Це робиться за допомогою сценаріїв Unity, написаних мовою C#, які розширюють середовище Unity. Ці сценарії використовують попередньо визначені класи Unity. Деякі сценарії Unity зчитують дані з файлу CSV і

зберігають їх у структурі даних. Інші сценарії Unity забезпечують правильну візуалізацію моделі.

1.3. Основні поняття розробки на основі моделі

Існує кілька визначень розробки на основі моделі, наприклад:

• «Розробка на основі моделі — це просто уявлення про те, що ми можемо створити модель системи, яку потім можемо перетворити на реальну річ» [13].

• «Інженерія, керована моделями (MDE) — це методологія розробки програмного забезпечення, яка зосереджена на створенні та використанні моделей предметної області (тобто абстрактних представлень знань і дій, які керують конкретною областю застосування), а не на обчисленнях (або алгоритмічних)) концепції" [10].

Загалом, не існує загальноприйнятого визначення розвитку, керованого моделлю. Це спосіб розробки системи, який використовує рівні абстракції, які пропонують моделі для візуалізації та передачі певних ідей і концепцій системи. Основна ідея MDD полягає в тому, щоб підвищити продуктивність, і MDD має два основні переваги [4]:

• «Це покращує продуктивність розробників у короткостроковій перспективі, збільшуючи значення основних артефактів програмного забезпечення з точки зору того, скільки функціональних можливостей воно забезпечує».

• «Це підвищує продуктивність розробників у довгостроковій перспективі, зменшуючи швидкість, з якою основний програмний артефакт застаріває».

Зважаючи на ці переваги, залишається питання: як слід визначити інфраструктуру MDD? Згідно з документом «Model-Driven Development: A Metamodeling Foundation», існують деякі технічні вимоги, які повинна визначати інфраструктура підтримки MDD [4]:

1. Доступні поняття для створення моделей і правила їх використання.

2. Позначення для використання при зображенні моделей.

3. Як елементи моделі представляють елементи реального світу, включаючи артефакти програмного забезпечення.

4. Концепції для полегшення динамічних розширень користувача для концепцій моделі, нотації моделі та моделей, створених з них.

5. Концепція для полегшення обміну поняттями моделі та позначеннями, а також моделями, створеними з них.

6. Концепції для полегшення визначених користувачем відображень від моделей до інших артефактів, включаючи код.

Ми можемо зробити висновок, що моделі є рівнем абстракції між реальним світом і технічними деталями системи, що робить їх ідеальними для здатності визначати складні системи на високому рівні абстракції.

1.4. Особливості модельно-орієнтованої системної інженерії

Модельно-орієнтована системна інженерія (MDSE) – це сучасний підхід до розробки складних систем, який базується на використанні моделей як основного артефакту протягом усього життєвого циклу системи. Замість того, щоб починати з написання коду, інженери створюють абстрактні представлення системи у вигляді моделей. Ці моделі потім використовуються для генерації коду, документації та інших артефактів.

Перш ніж розпочати вступ до розробки систем, керованих моделями (MDSE), нам потрібно пояснити кілька понять. Почнемо з того, що таке система ? Існує багато визначень і поглядів на те, що саме можна описати як систему. У нас є визначення, яке визначає систему, як її використовуємо в мові [14]:

«Організований набір ідей чи теорій або певний спосіб робити щось».

або

«Група речей, частин обладнання тощо, які з'єднані або працюють разом».

З цих визначень можна зробити висновок, що система – це сукупність зв'язаних елементів. У ширшому контексті ці системи можуть мати зв'язки з іншими системами. Це призводить до складної мережі систем, з'єднаних одна з одною, це називається системою систем або скорочено SoS. Знову ж таки, існує багато різних визначень для SoS, але одне з цих формальних визначень таке [6]: «Системи систем (SoS) визначаються як взаємодіюча сукупність систем-компонентів, які дають результати, недосяжні окремими системами».

Включно з цим визначенням INCOSE узагальнює концепції, пов'язані з системою, як показано на діаграмі на рисунку 1.5.

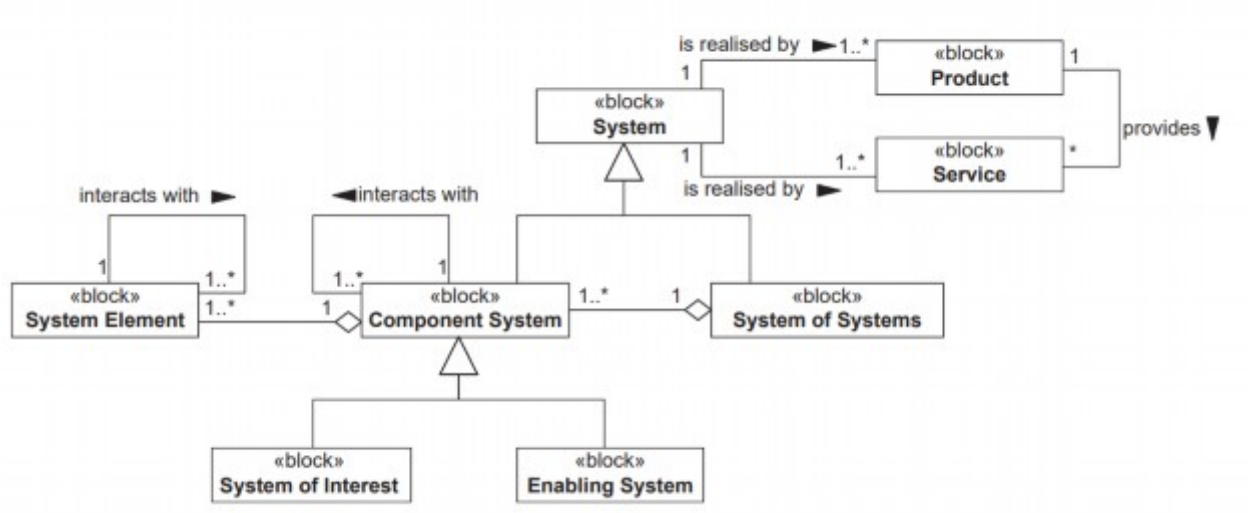


Рис. 1.5. Стислий огляд концепцій, пов'язаних із системою

Взявши концепції та визначення системи, системи систем і розробки на основі моделі в цілому, ми можемо зробити висновок, що системна інженерія на основі моделі є способом побудови складної системи або системи систем, який використовує абстракцію, яку надають моделі.

MDSE особливо ефективна для розробки великих і складних систем, де важлива висока якість, швидкість розробки та гнучкість. Цей підхід широко

використовується в таких галузях, як авіація, автомобілебудування, телекомунікації та програмна інженерія.

Основні інструменти MDSE:

- UML (Unified Modeling Language): Стандартна мова для візуального моделювання систем.

- SysML (Systems Modeling Language): Розширення UML для системної інженерії.

- Інструменти для моделювання: Rhapsody, Enterprise Architect, MagicDraw тощо.

Висновки до розділу

У цьому розділі було проведено аналіз предметної області моделювання систем, зокрема аспектів, що стосуються використання моделі як основи для розробки. Описано ключові інструменти моделювання, які використовуються в сучасних середовищах розробки, і розглянуто їхні особливості та можливості.

Зокрема, детально проаналізовано середовище проектування IBM Rational Rhapsody, яке є потужним інструментом для моделювання складних систем. Воно забезпечує підтримку різних підходів до системного проектування та моделювання, зокрема UML та SysML, що дозволяє забезпечити чітке та систематичне перетворення моделей у код. Це середовище надає інженерам можливості для моделювання, автоматизації процесів та перевірки функціональних характеристик розроблюваних систем, що є важливим для забезпечення якості та надійності розробок.

Інструмент 3D-візуалізації Unity був представлений як інструмент для створення інтерактивних моделей, особливо у випадках, коли візуалізація та інтерактивність є ключовими аспектами системи. Unity дає змогу створювати високоякісні візуальні компоненти та інтерактивні елементи, що важливо для розробки складних моделей, які потребують реалістичної візуалізації. Це

дозволяє інженерам не лише проектувати, але й тестувати моделі в реальному або віртуальному середовищі.

Розглянуто основні поняття розробки на основі моделі (Model-Driven Development, MDD) та модельно-орієнтованої системної інженерії (Model-Based Systems Engineering, MBSE). Виявлено, що ці підходи дозволяють значно підвищити ефективність розробки складних систем шляхом автоматизації процесів перетворення моделей у програмний код. MDD і MBSE спрощують процес розробки, забезпечуючи зв’язок між рівнем абстракції та реалізацією, що дозволяє зменшити кількість помилок та підвищити якість кінцевого продукту.

На основі проведеного аналізу встановлено, що використання модельно-орієнтованих підходів та інструментів моделювання значно сприяє підвищенню продуктивності процесу розробки складних систем. Це досягається завдяки автоматизації перетворення моделей та інтеграції різних компонентів у єдине середовище розробки. Такий підхід забезпечує високу гнучкість у проектуванні, дозволяючи адаптувати моделі до змін у вимогах та забезпечуючи оптимальний перехід від проектування до імплементації.

Отже, дослідження предметної області моделювання систем на основі перетворення моделей підкреслює важливість інструментів моделювання та сучасних підходів до проектування, що є невід'ємними для ефективної розробки та оптимізації складних систем.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ МЕТОДІВ ТА АЛГОРИТМІВ ДЛЯ ПОБУДОВИ ШАРУ КОНВЕРСІЇ МІЖ РІВНЕМ МОДЕЛЕЙ ТА РІВНЕМ ІМПЛЕМЕНТАЦІЇ

2.1. Фактори складнощі в застосуванні модельно-орієнтованої системної інженерії

Багато проектів зазнають невдачі та завершуються катастрофічними наслідками. Ідеї не можуть бути належним чином розроблені, а люди не розуміють ідей один одного. Згідно з працею "SysML для системної інженерії: модельний підхід" [7], причинами провалу проектів є три основні фактори: складність, відсутність розуміння та проблеми комунікації. Ці концепції взаємопов'язані, як показано на рисунку 2.1.

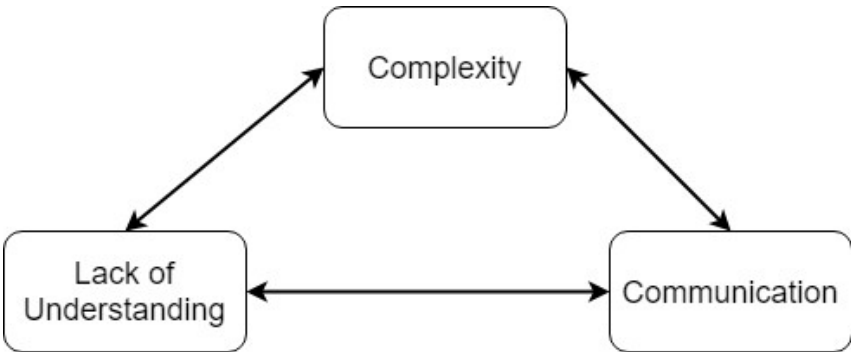


Рис. 2.1. Фактори провалів проектів

Складність важко визначити, існують різні рівні складності, але складність не може бути виміряна. Приклад наведено на рисунку 2.2, який фокусується на взаємозв’язках між певними елементами. Метою цього прикладу є показати, що навіть якщо ми маємо повне розуміння окремих елементів, система все одно може бути складною.

Якщо ми розглянемо рисунок 2.2 (а), то побачимо 5 різних блоків. Припустимо, що ми повністю розуміємо всі окремі блоки, тоді ця система не є складною. Якщо ми розглянемо рисунок 2.2 (б), то там присутні ті ж самі

окремі блоки, отже, складність всередині блоків не змінилася. Проте складність системи (б) є вищою через зв’язки між окремими блоками. Тепер очевидно, що рисунок 2.2 (с) є складнішим за свого попередника через зростання кількості зв’язків між блоками. Навіть якщо є повне розуміння певних елементів у системі, це не означає, що система як ціле не є складною.

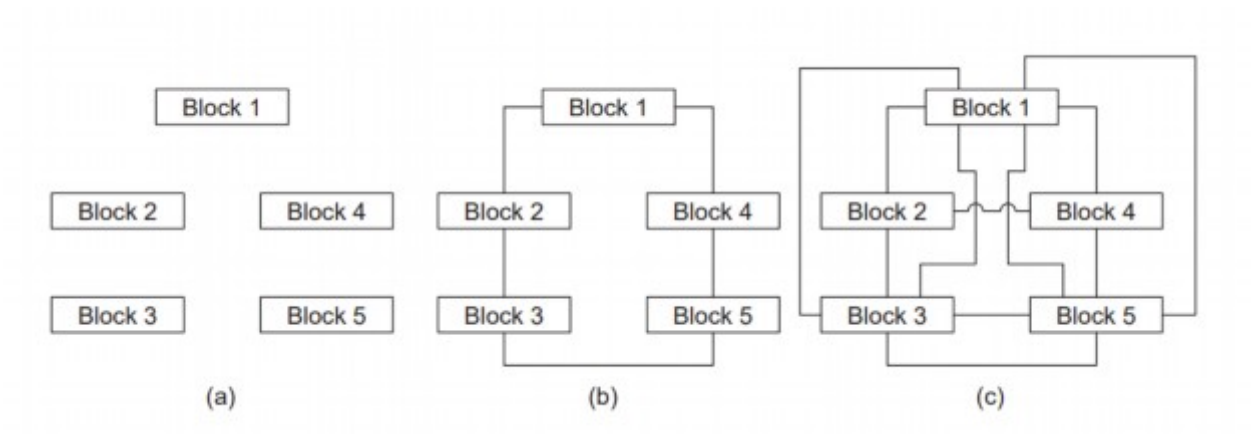


Рис. 2.2. Схеми систем різного рівня складності

Відсутність розуміння може виникнути в будь-який момент життєвого циклу системи. Розглянемо кілька прикладів [7]:

Відсутність розуміння може виникнути на етапі концепції проекту, під час процесу, пов'язаного з вимогами. Якщо вимоги не сформульовані чітко та однозначно (або, в ідеалі, наскільки це можливо однозначно), то ця відсутність розуміння поширюється на весь проект. Загальновизнано, що помилки, допущені на ранніх етапах життєвого циклу, обходяться в багато разів дорожче для виправлення на пізніших етапах життєвого циклу, тому має сенс все робити правильно якомога раніше.

Відсутність розуміння може виникнути на етапі розробки проекту, під час процесу, пов'язаного з аналізом. Наприклад, може бути відсутність знань про впровадження SysML та системного моделювання під час аналізу, що може призвести до неправильних припущень або навіть до повністю неправильних результатів через недостатність знань. Знову ж таки,

невиправлені помилки на цьому етапі призведуть до більших проблем на подальших етапах життєвого циклу розробки.

Відсутність розуміння може виникнути на етапі експлуатації проекту, під час процесу, пов'язаного з експлуатацією. Неправильне використання системи може призвести до збою системи або катастрофи. Наприклад, люди, які не дотримуються правил безпеки або не використовують правильні інструменти.

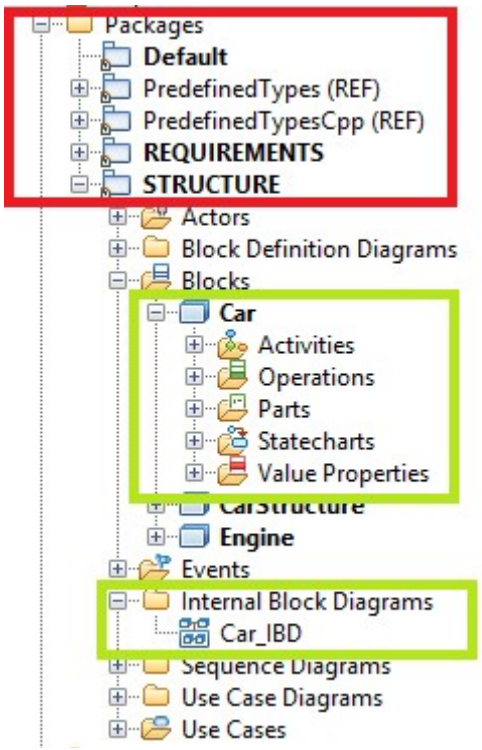


Рис. 2.3. Корисна інформація (виділено зеленим) у незрозумілій структурі (виділено червоним)

Комунікація є однією з сильних сторін, якими наділені люди. Неефективна комунікація може призвести до провалу проекту. Проблеми комунікації можуть виникати на будь-якому рівні організації або проекту, вони можуть бути особистісними, груповими, міжорганізаційними, система-система або будь-якою комбінацією цих рівнів комунікації. Не лише рівень відрізняється, але й сфера, це може бути мовна різниця або технічна специфікація, яка не була добре комунікована.

2.2. Методи вирішення незрозумілості моделей

Три фактори, розглянуті в попередньому розділі, не можуть бути ізольовані, вони з'єднані в трикутник, як показано на рисунку 2.1. Проблема в одному з трьох факторів впливає на два інших. Єдине, що можна зробити, це звернути увагу на кожен з цих факторів окремо.

2.2.1. Абстрагування від деталей

Однією з переваг розробки на основі моделі є те, що можна абстрагуватися від технічних деталей, щоб зробити речі більш зрозумілими. Зі зростанням складності систем ця абстракція втрачена. Важливі речі, які слід показати, оточені незрозумілою структурою, що призводить до старої приказки «Не видно лісу за його деревами». Приклад цього представлено на рисунку 2.3. Ми маємо корисну інформацію в зеленому полі, блок автомобіля з усіма його властивостями, який має внутрішню блок-схему «Дочірня» в нижньому зеленому полі. Ця інформація неочевидна, оскільки вона має надзвичайно незрозумілу структуру.

2.2.2 Використання 3D середовища

Наразі елементи моделі SysML відображаються у 2-вимірному середовищі. Однак можна стверджувати, що двовимірного зображення недостатньо для розуміння складних систем. Оскільки тепер у нас є потужні комп'ютери, які можуть генерувати тривимірні візуалізації в режимі реального часу, цікаво дослідити, що ця тривимірна візуалізація може зробити для відображення елементів моделі та зв'язків. Як зазначалося раніше, моделі часто стають великими та складними. Одна система може мати багато різних діаграм, які всі пов'язані одна з одною. У попередньому розділі було описано, як можна збільшити складність шляхом збільшення зв'язків. Це тому, що зв'язки між елементами стають менш чіткими, і ми не маємо чіткого огляду.

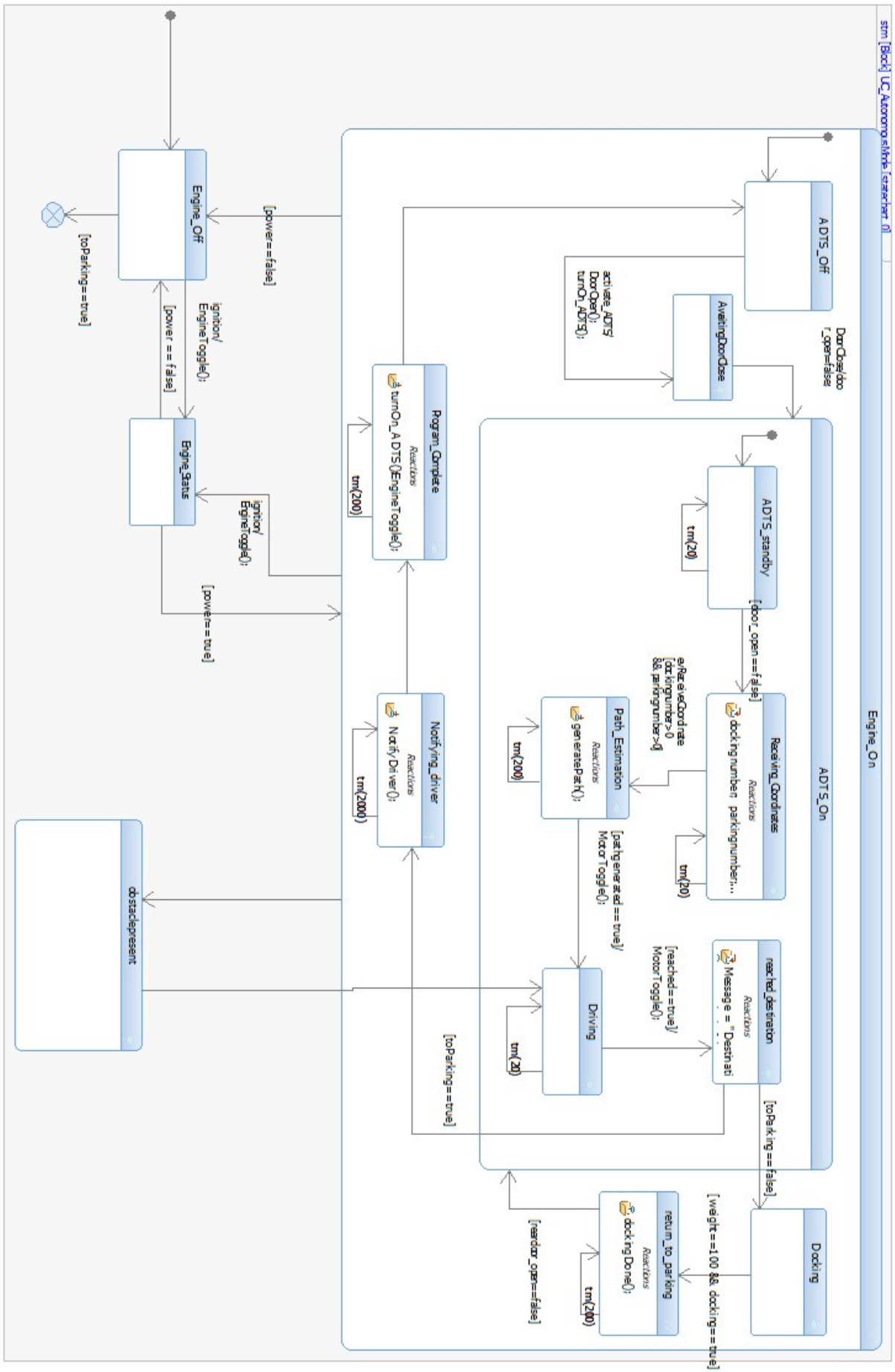


Рис. 2.4. Представлення складності у діаграмі варіантів використання

На рисунку 2.4 представлено приклад складності діаграми кінцевого автомата. На цій діаграмі багато вузлів і шляхів. На цьому складність не закінчується, оскільки ця діаграма не лише має внутрішні шляхи, але й пов’язана з іншими діаграмами в моделі. У двовимірному середовищі важко представити всю цю інформацію в загальному огляді. Таким чином, двовимірне середовище обмежує користувача, що можна вважати проблемою, яку слід вирішити.

З технологічним прогресом, який ми досягаємо, 3D візуалізація стає все доступнішою. В [2] показали, що перегляд 3D-графіка у віртуальній реальності краще, ніж використання 2D-діаграми. Цікаво, що може зробити тривимірна візуалізація для розуміння моделі програмного забезпечення чи системи.

У [12] досліджено цю 3D-візуалізацію та отримано кілька цікавих результатів. На рисунку 2.5 ми можемо побачити діаграму класів UML, відображену в 3D, усі різні з’єднання та елементи мають різну форму та колір, завдяки чому з одного погляду стає зрозуміло, які речі пов’язані. Це переконлива концепція на шляху до візуалізації 3D-моделі.

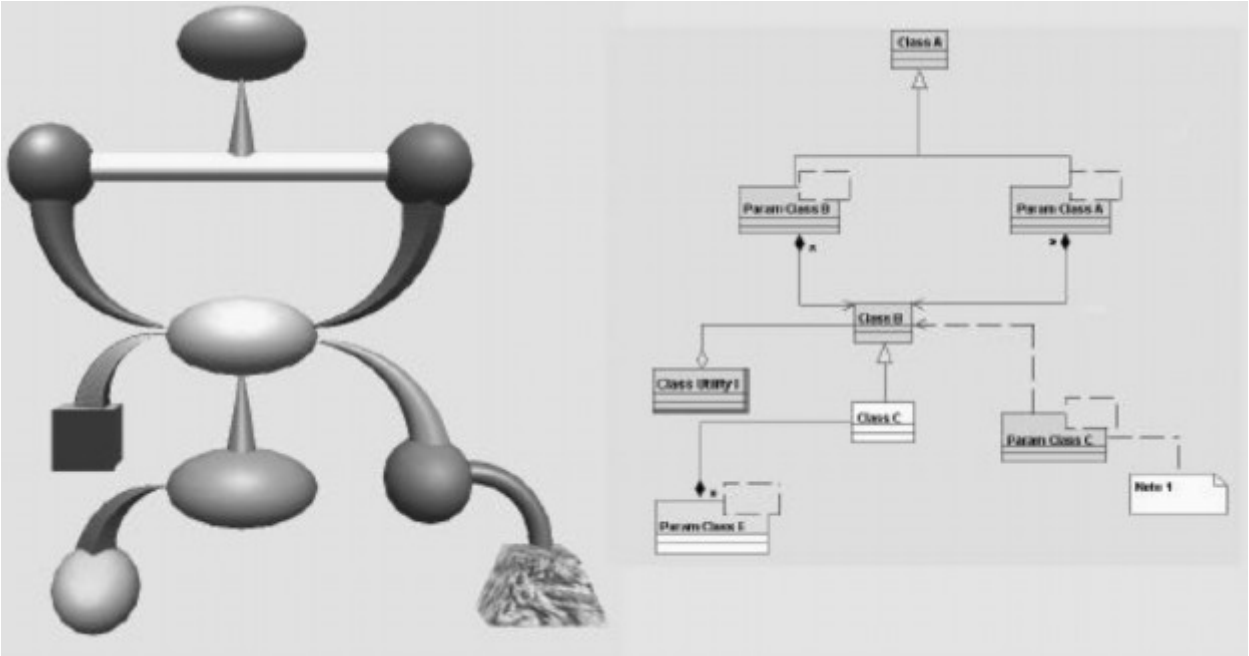


Рис. 2.5. Тривимірна візуалізація діаграми класів UML [12]

У роботі [9], досліджується багатовимірний підхід UML. Передбачається, що створений інструмент буде схожий на спільні онлайн-інструменти, такі як Google Document. Метою цього підходу є зменшення складності UML-моделей і підвищення ефективності роботи. Підхід до тривимірної візуалізації полягає у використанні двовимірних діаграм UML на взаємопов'язаних шарах. Користувач може обертати цю тривимірну структуру та бачити зв'язки не лише самих діаграм, але й взаємозв'язки між діаграмами. Користувачі також можуть спілкуватися один з одним, і коли один користувач вносить зміни в модель, вона оновлюється в реальному часі для іншого користувача. Знімок екрана демонстраційного інструменту, наданого в цій роботі представлено на рисунку 2.6.

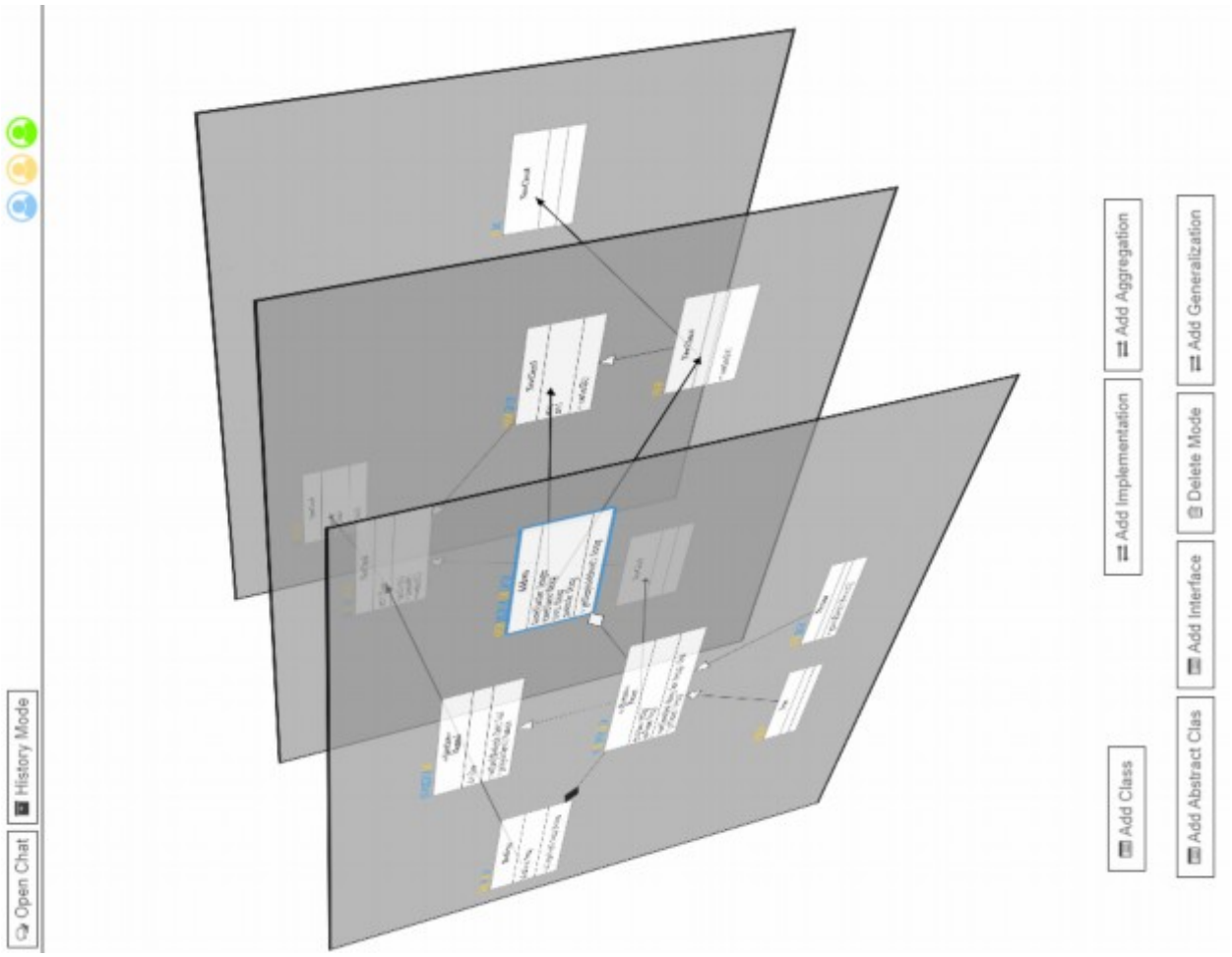


Рис. 2.6. Інструмент для спільного 3D UML-моделювання [9]

Ми визначили, що мета розробки, керованої моделлю, полягає в покращенні продуктивності розробників як у короткостроковій, так і в довгостроковій перспективі. У системній інженерії цей спосіб розробки використовується для побудови складних систем з використанням моделей. Багато проектів зазнають невдачі через три фактори: складність, брак розуміння та спілкування. Ми дослідили деякі обмеження двовимірного середовища. Використовуючи переваги інших 3D-візуалізацій модельних проектів, можливим рішенням є фільтрування корисної інформації від незрозумілої структури та використання методів 3D-візуалізації.

У цій роботі буде показано, як ці рішення можна реалізувати шляхом створення 3D-візуалізації відфільтрованих даних моделі SysML. Це буде початковий крок для візуалізації 3D моделі. Одним із наступних кроків може бути створення 3D-копії моделі, яку можна буде запустити в програмі 3D-моделювання, яка показує не лише структуру, але й поведінку.

2.3. Дослідження та опис архітектури мови системного моделювання

Мова системного моделювання (SysML) — це експресивна графічна мова моделювання, яка належить і публікується Object Managment Group, Inc. (OMG) [11]. SysML — це похідна від уніфікованої мови моделювання (UML), яка була створена в 1990-х як мова загального призначення для розробки програмного забезпечення. У 1997 році UML було прийнято як стандарт тією ж групою керування об’єктами (OMG), яка створила SysML. Зараз UML визнається стандартною мовою моделювання загального призначення всіма розробниками програмного забезпечення. Системним інженерам потрібна була мова загального призначення, така як UML, для системної інженерії, тому Міжнародна рада з системної інженерії (INCOSE) прийняла рішення про ініціативу SysML. Після цього рішення INCOSE та OMG об’єдналися, щоб створити OMG Systems Engineering Domain Special

Interest Group (SE DSIG). Після цього було створено OMG SysML, що містить 9 типів діаграм, деякі з яких узяті з UML.

2.3.1. Порівняння SysML з UML

Взаємозв’язок між UML2 та SysML можна побачити на діаграмі, що представлена на рисунку 2.7. Ми бачимо, що UML2 та SysML мають великий перетин, який описується як "UML, повторно використаний у SysML (UML4SysML), це діаграми та метакласи, які використовуються як UML2, так і SysML. Область, описана як "UML, не потрібна для SysML (UML-UML4SysML), це діаграми та метакласи, які є в UML2, але не використовуються SysML. І область, описана як "Розширення SysML до UML (SysMLProfile), це діаграми та метакласи, які були нещодавно введені SysML.

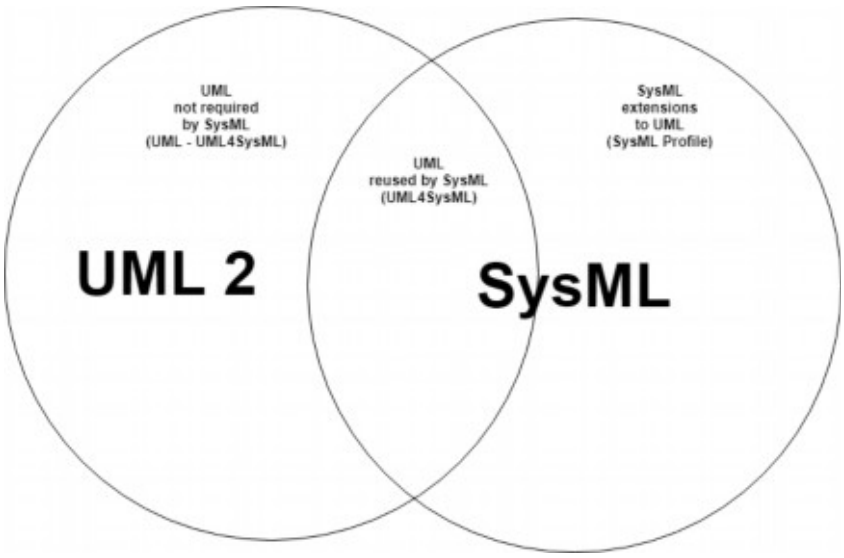


Рис. 2.7. Діаграма взаємозв’язку SysML і UML

2.3.2. Принципи проектування

Основні принципи проектування для SysML надані групою керування об’єктами (OMG) у специфікації мови моделювання систем OMG [11]:

• Керований вимогами – SysML призначений для задоволення вимог UML для SE RFP.

• Повторне використання UML – SysML повторно використовує UML, де це можливо, щоб задовольнити вимоги RFP, і коли потрібні зміни, вони виконуються таким чином, щоб мінімізувати зміни базової мови. Відповідно, SysML призначений для відносно легкого впровадження для постачальників, які підтримують UML 2.

• Розширення UML – SysML розширює UML за потреби, щоб задовольнити вимоги RFP. Основним механізмом розширення є механізм профілю UML 2.

• Розбиття на розділи - пакет є основною одиницею розділення в цьому міжнародному стандарті ¬. Пакети розбивають елементи моделі на логічні групи, які мінімізують циклічні залежності між ними.

• Розрівнювання — пакети SysML визначено як рівень розширення метамоделі UML.

• Взаємодія - SysML успадковує можливість обміну XMI від UML. SysML також має підтримуватися стандартом обміну даними ISO 10303-233 для підтримки взаємодії з іншими інженерними інструментами.

2.3.3. Загальний огляд діаграм

У цьому розділі описано, які саме діаграми SysML існують і що вони роблять [5]. Існує дев'ять типів діаграм SysML:

• Діаграма визначення блоку (BDD): відображає такі елементи, як блоки та типи значень. BDD можна порівняти з діаграмою класів UML, де блоки представляють класи.

• Внутрішня блок-схема (IBD): визначає внутрішню структуру блоку. Показує з’єднання та інтерфейси між усіма частинами в одному блоці.

• Діаграма варіантів використання: діаграма варіантів використання має ті самі функції в SysML, що й діаграми UML. Він служить для надання графічного огляду всіх функцій, наданих у системі, і учасників, які можуть використовувати ці функції.

• Діаграма діяльності: визначає поведінку за допомогою дій і вхідних і вихідних даних. Діаграма діяльності — це діаграма поведінки, що означає, що поведінка системи виражається за допомогою діаграми діяльності.

• Діаграма послідовності: діаграма поведінки, яка може бути використана для моделювання того, як частини блоку взаємодіють одна з одною за допомогою викликів операцій і сигналів. Часто діаграми послідовності використовуються для визначення тестових випадків.

• Діаграма кінцевого автомата: діаграма поведінки для вираження певних станів системи. Система може переходити в інший стан за допомогою подій.

• Параметрична діаграма: параметрична діаграма використовується для зв’язку обмежень із властивостями системи.

• Пакетна діаграма: пакетна діаграма — це структурна діаграма, яка показує, як виглядає ієрархія в системі. Він використовується для моделювання системи з високого рівня.

• Діаграма вимог: діаграма вимог — це діаграма, яка визначає всі вимоги, які має мати певна система, ці вимоги можуть мати зв’язки між собою.

Огляд усіх категорій і зв'язків між діаграмами представлено на рис. 2.8.

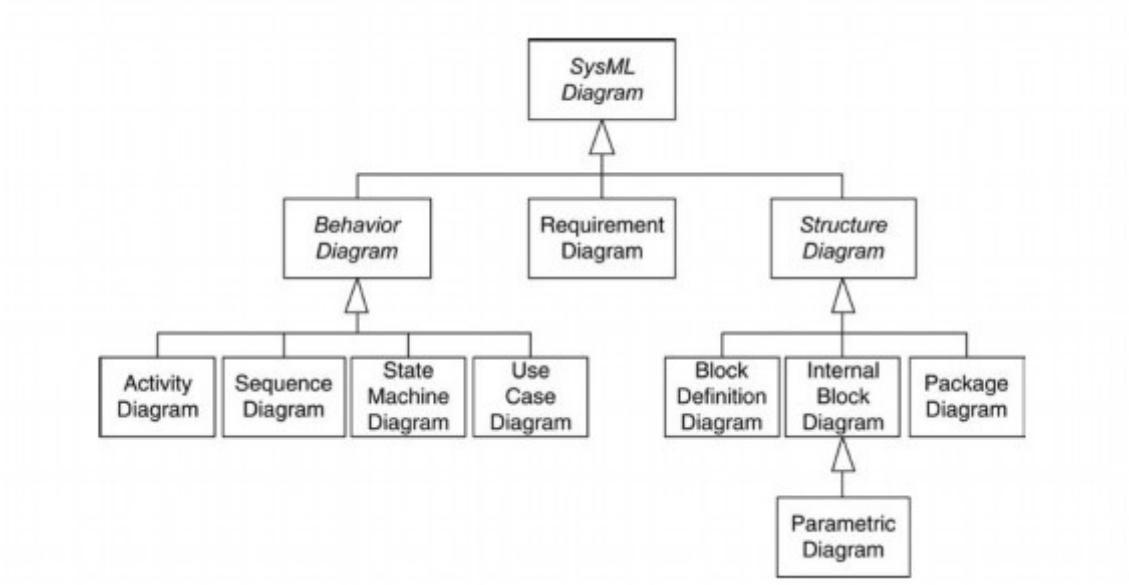


Рис. 2.8. Таксономія діаграм SysML

2.4. Особливості процесу моделювання з використанням SysML

Як зазначалося раніше, SysML є універсальною графічною мовою для моделювання, специфікації, аналізу, проектування та верифікації. Щоб змоделювати систему, ми повинні почати з вимог до системи, з яких ми можемо показати структуру, поведінку та параметри системи. Вимоги є будівельними блоками системи. Вони визначають, на що повинна бути здатна система, і часто пов'язані з компонентами системи. SysML використовує діаграму вимог для показу вимог системи та зв'язку з іншими вимогами та елементами моделі. Приклад моделювання вимог наведено на рисунку 2.9.

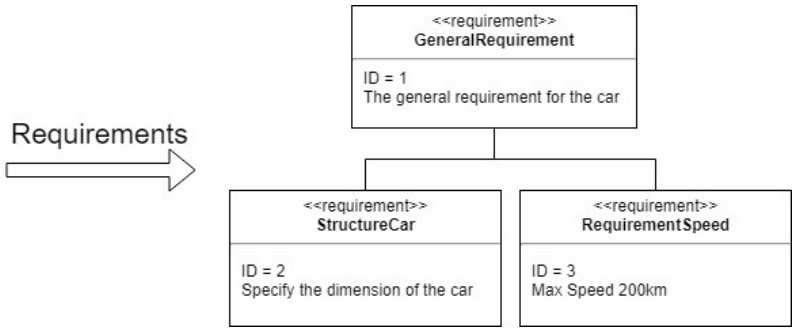


Рис. 2.9. Вимоги до простої системи автомобіля

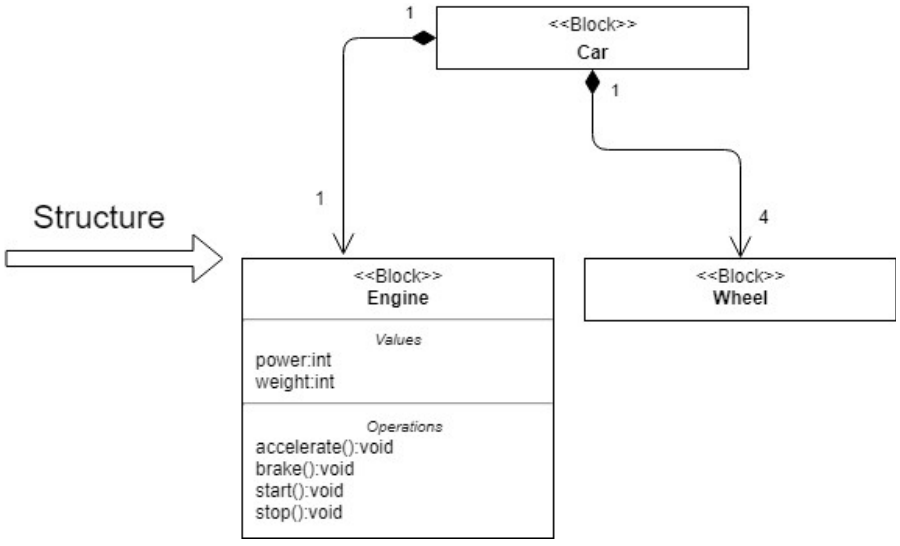


Рис. 2.10. Структура (схема визначення блоку) простої системи автомобіля

Структура системи визначає, які елементи існують у системі та як вони з’єднуються один з одним. У SysML ми використовуємо діаграму визначення блоку, внутрішню блок-схему та діаграму пакета для визначення структури в системі. На рисунку 2.10 представлено приклад структури в SysML.

Поведінка системи охоплює функціональні можливості системи або її частини. У SysML вони моделюються за допомогою діаграм діяльності, діаграм послідовностей, діаграм кінцевого автомата та діаграм варіантів використання. На рисунку 2.11 представлено приклад поведінки в SysML (State Machine Diagram).

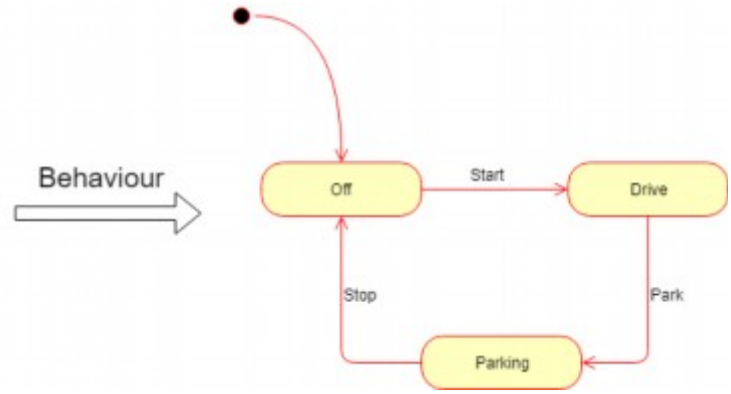


Рис. 2.11. Поведінка простої системи автомобільного контролера

Параметри в системі використовуються для вираження рівнянь і обмежень за допомогою параметрів. У SysML ми використовуємо параметричну діаграму для визначення таких рівнянь і обмежень у внутрішніх блоках. На рисунку 2.12 показана параметрична діаграма для обмеження швидкості автомобіля.

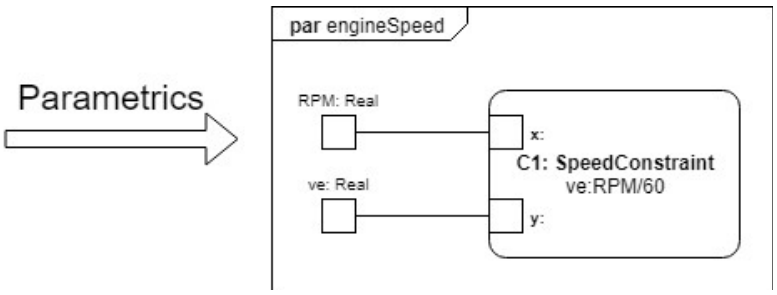


Рис. 2.12. Параметри простого системного обмеження

2.5. Дослідження концепцій інструментаріїв моделювання

2.5.1. Опис IBM Rhapsody

IBM Rational Rhapsody — це візуальне середовище розробки для системних інженерів і розробників програмного забезпечення, які створюють системи та програмне забезпечення в режимі реального часу або вбудовані системи. Він підтримує широкий спектр мов моделювання, таких як: UML, SysML, AUTOSAR, MARTE, DDS, MODAF, UPDM, multicore, MISRA-C, MISRA-C++. Для цього проекту IBM Rhapsody буде використано як інструмент моделювання SysML. Цей інструмент відповідає вимогам цього проекту, це графічне середовище у 2D, де підтримуються всі діаграми, які ми хочемо дослідити.

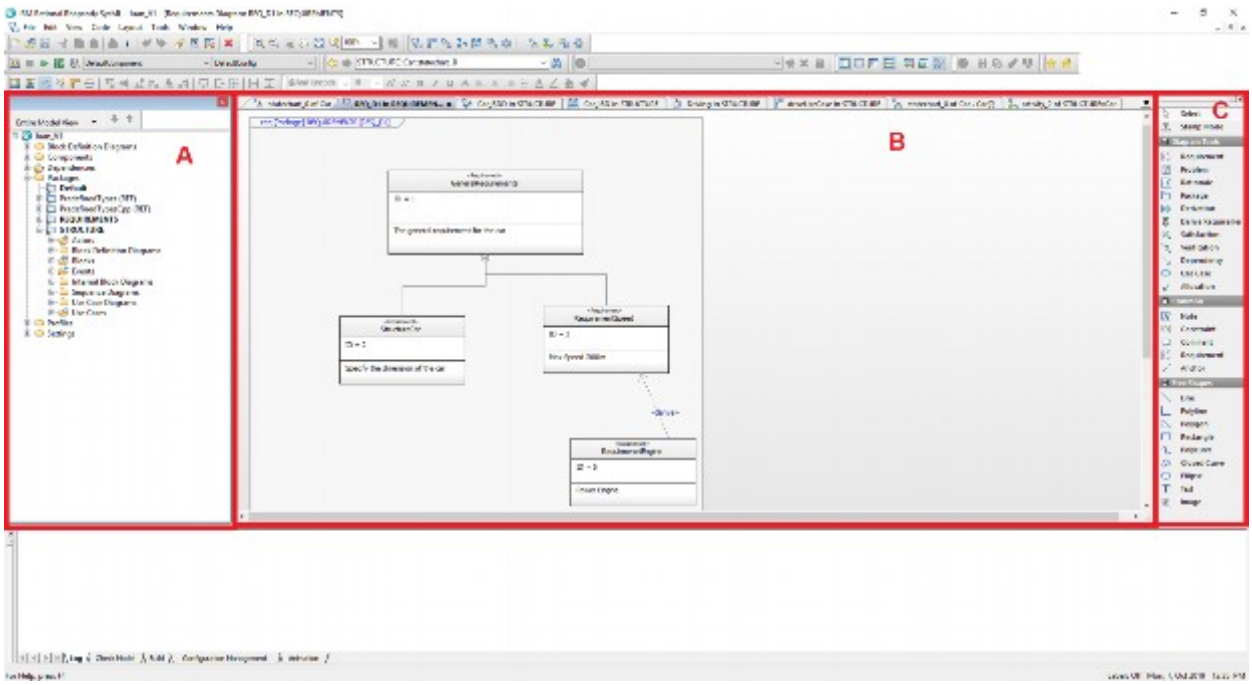


Рис. 2.13. Огляд IBM Rhapsody A) Браузер проекту, B) Переглядач діаграм, C) Панель інструментів

Огляд інтерфейсу IBM Rhapsody представлено на рисунку 2.13. Інтерфейс розділений на три важливі частини:

A) Браузер проекту: інтерфейс користувача, який дозволяє користувачеві керувати проектом. Він представляє пакети, діаграми та елементи. Відображення цього браузера є у форматі дерева, що означає, що папка чи пакет мають дочірні елементи, які можна згорнути та розгорнути.

B) Переглядач діаграм: двовимірний графічний інтерфейс, де діаграму можна відкривати та редагувати. У верхній частині може бути кілька вкладок для навігації по всіх відкритих діаграмах. У цьому поданні можна прокручувати по горизонталі та по вертикалі, щоб розширити подання.

C) Панель інструментів: містить значки та кнопки, які є елементами поточного типу моделі. Наприклад, у діаграмах визначення блоків панель інструментів складатиметься з блоків, асоціацій тощо. У той час як у діаграмі вимог він існуватиме з вимог, залежностей тощо.

Оскільки в проекті SysML багато діаграм, це подання досить обмежене. Як пояснювалося раніше, у браузері проекту є незрозуміла структура. Засіб перегляду діаграм може одночасно відображати лише обмежену кількість діаграм і не показує зв’язки між діаграмами. Розуміти проекти таким чином досить складно і часто потребує додаткових пояснень, що може призвести до провалу проекту, якщо його не виконати належним чином.

2.5.2. Платформа Unity

Unity [1] – це тривимірна платформа для розробки ігор і симуляцій. У цьому розділі представлено середовище Unity. Ми обговоримо, що таке Unity, пояснимо деякі особливі елементи Unity та пояснимо, чому Unity було обрано як інструмент. У цьому проекті Unity використовується для представлення нашої моделі SysML як тривимірної візуалізації.

Unity є провідним ігровим рушієм і на цій платформі створено багато великих ігор, таких як Hearthstone: Heroes of Warcraft, Pokemon GO, Escape from Tarakov та багато інших. Unity визначається як платформа створення в реальному часі, що означає, що досвід роботи в реальному часі негайно реагує на введення користувача. Ця техніка реального часу також популярна

для створення CG-анімації у фільмах. Перевагою, наприклад, фільмів є те, що проект «живий» відразу після початку виробництва. Оскільки створення відбувається в реальному часі, усі елементи анімації можна змінювати на льоту, наприклад положення камери або рух.

Елементи в Unity GameObject

GameObject — це стандартна сутність в Unity, наприклад персонажі, світло та реквізит. GameObject сам по собі нічого не робить, додаючи компоненти до GameObject, ми повертаємо його до життя. Компоненти — це поведінка GameObjects, до яких вони приєднані. Приклад GameObject з деякою поведінкою можна побачити на рисунку 2.14.

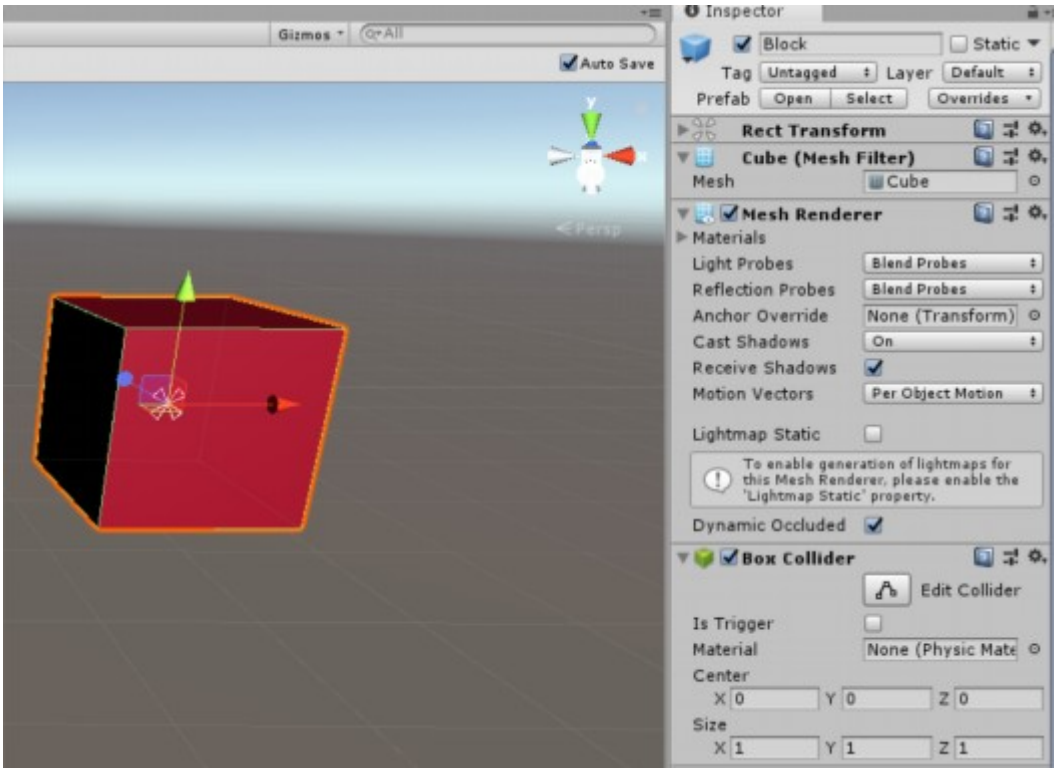


Рис. 2.14. Простий блок GameObject

Prefab Prefab — це шаблон GameObject, до якого вже можуть бути приєднані властивості та сценарії. Його можна створити в сценаріях у певній позиції сцени.

Scene A – це середовище, в якому відбувається тривимірна анімація. Проект може мати кілька сцен. В іграх сцена часто використовується для «рівня» або, наприклад, іншої сцени «меню» або сцени «Гра закінчена». Коротше кажучи, всі сцени можуть мати різні елементи та об’єкти, включаючи види камери. Спочатку кожна створена сцена порожня, лише основна камера. Приклад сцени показано на рисунку 2.15.

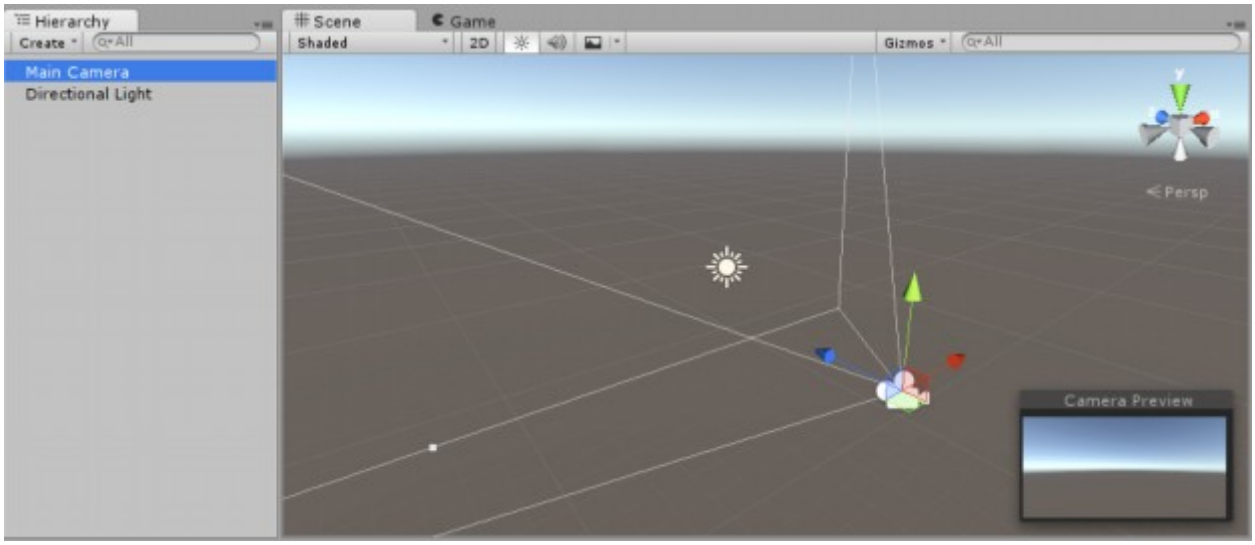


Рис. 2.15. Приклад порожньої сцени

Сценарії в Unity — це фрагменти коду C#, які можна приєднати до GameObjects. Більшість сценаріїв, які пов’язані з GameObjects, походять від класу MonoBehaviour, це клас Unity, який містить багато функцій. Наприклад, клас MonoBehaviour має функції Start() і Update(). Функція Start() виконує код, коли приєднаний GameObject генерується на сцені, тоді як функція Update() виконує код у функції кожного кадру сцени.

Незважаючи на те, що Unity здебільшого використовується як механізм 3D-ігор, він визначається як платформа для 3D-розробки в реальному часі. Це означає, що Unity може мати більше випадків використання, ніж просто створення ігор. Деякі приклади того, що було створено в Unity, що не пов’язано безпосередньо з грою:

- Модель віртуальної реальності, що імітує роботу реальних автомобілів [3];

- Система тривимірного моделювання для навігації та керування мініатюрними безпілотними літальними апаратами в середовищі без GPS [15];

- Система імітації навчання водінню [8].

Unity має чітку документацію на своєму веб-сайті. Згідно документації сценарій надзвичайно важливий, оскільки ми хочемо створити програму, яка витягує дані з файлів CSV і створює структуру даних. Підсумовуючи, Unity ідеально підходить для пропонованого проекту.

Висновки до розділу

У другому розділі було проведено аналіз методів і алгоритмів, які використовуються для побудови шару конверсії між рівнем моделей та рівнем імплементації в модельно-орієнтованій системній інженерії (MBSE). Розглянуто ключові фактори, що ускладнюють застосування MBSE, а також запропоновано способи подолання цих викликів, що є важливими для ефективної реалізації модельно-орієнтованих підходів.

Зокрема, виявлено, що серед основних факторів складності у застосуванні MBSE є високий рівень деталізації моделей та необхідність точного узгодження між абстрактними моделями і конкретними вимогами реалізації. Це потребує розробки чітких методів, які забезпечують перехід від моделей до коду. Як вирішення цих проблем розглянуто методи абстрагування від деталей та використання 3D-середовищ для візуалізації моделей, що дозволяють спростити сприйняття складних моделей та зменшити розрив між моделями та їх реалізацією.

Дослідження архітектури мов системного моделювання, зокрема SysML, дало змогу оцінити їхню придатність для забезпечення ефективної комунікації між різними рівнями абстракції. Було проведено порівняння

SysML з UML, що дозволило виділити переваги SysML для системного інженерного моделювання, зокрема його здатність більш точно відображати системні вимоги і функціональність. Описано принципи проектування на основі цих мов та проведено огляд основних типів діаграм, що використовуються для представлення системних моделей.

Розглянуто особливості процесу моделювання з використанням SysML, що включають формалізацію системних вимог та їх зв’язок із процесами реалізації. Застосування SysML сприяє забезпеченню чіткого переходу від концептуальних моделей до конкретної реалізації, що є важливим для побудови ефективного шару конверсії.

Крім того, у розділі досліджено концепції інструментаріїв для моделювання та їхні особливості. Розглянуто можливості середовища IBM Rational Rhapsody як інструмента для розробки систем з використанням SysML, а також потенціал платформи Unity для реалізації тривимірних візуалізацій моделей. Це дослідження підкреслює важливість вибору відповідного інструментарію для реалізації складних системних моделей та їхнього подальшого перетворення в програмний код.

РОЗДІЛ 3. РОЗРОБКА МЕТОДОЛОГІЇ ВИЗНАЧЕННЯ ШАРУ КОНВЕРСІЇ МІЖ РІВНЕМ МОДЕЛЕЙ ТА РІВНЕМ ІМПЛЕМЕНТАЦІЇ

3.1. Побудова архітектури системи для перетворення моделі

У цьому розділі представлено архітектуру всієї системи. Створена система є рівнем перетворення між IBM Rhapsody та Unity. Система в цілому представлена на рисунку 3.1. IBM Rhapsody та її базу даних можна описати як джерело даних, де зберігаються моделі SysML. Сценарії екстрактора, зв'язку та обробки даних можна описати як рівень перетворення. Тривимірну візуалізацію в Unity можна описати як продукт візуалізації.

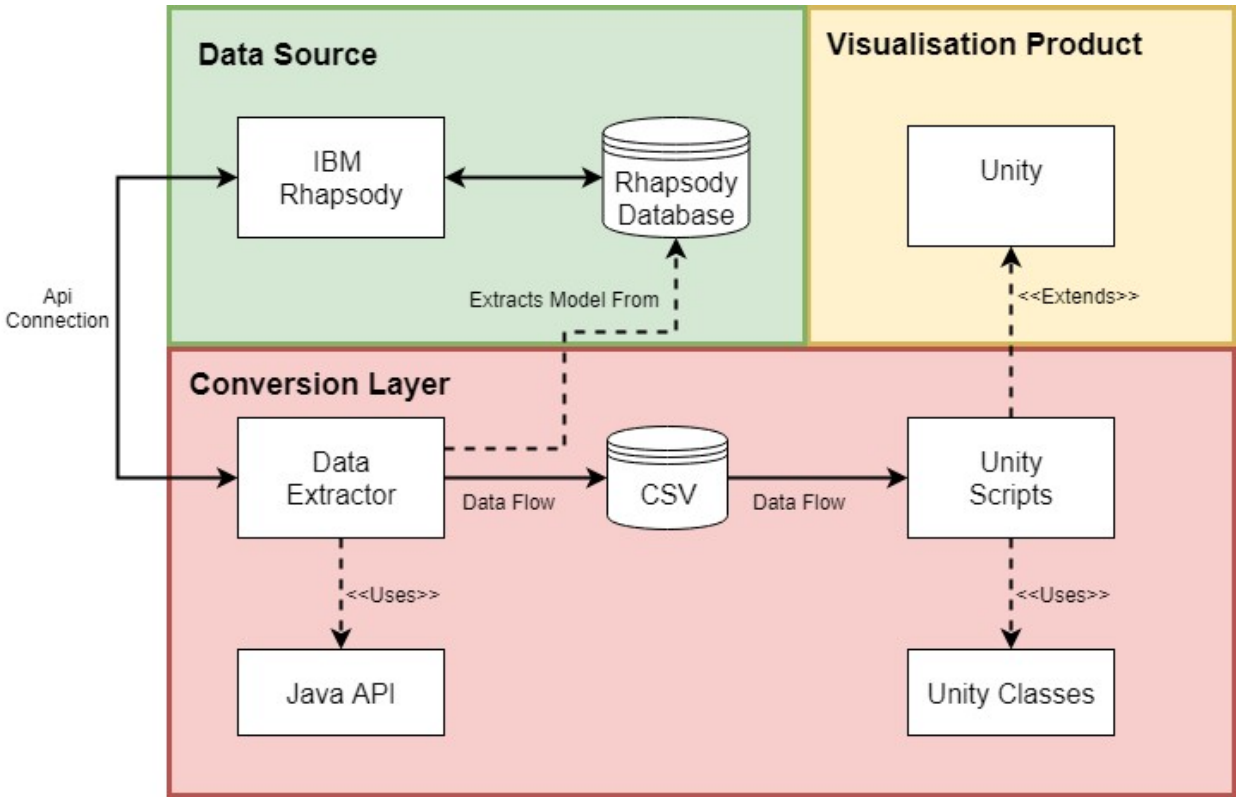


Рис. 3.1. Огляд архітектури системи: джерело даних, продукт візуалізації та рівень перетворення.

У цьому розділі описується рівень перетворення в трьох розділах, які представляють різні частини шару:

• Вилучення даних: описує, як здійснюється підключення до IBM Rhapsody і які дані витягуються. Цей розділ відповідає блоку Data Extractor.

• Комунікація: описує спосіб передачі даних від Data Extractor до Unity. Цей розділ відповідає сховищу даних CSV.

• Обробка даних: описує структуру даних, яка використовується для вираження даних моделі та сценаріїв візуалізації. Цей розділ відповідає блоку сценаріїв Unity.

3.2. Опис етапу та методу видобування даних для моделі

Цей розділ розділений на дві частини. У першій частині обговорюється, які дані важливо отримати. У другій частині буде представлено метод видобування даних моделі з IBM Rhapsody. Предметна область – авто.

Модель — це представлення, яке використовується для кращого розуміння системи та візуалізації системи. Важливо, щоб ці моделі не заплутували. Відображення всієї інформації в складній системі зменшить шанс повного розуміння важливих частин системи. Отже, щоб зробити моделі зрозумілими, візуалізація повинна бути абстрактною. За допомогою тривимірної візуалізації ми робимо більш чітке уявлення про зв’язки між елементами. Основна інформація має бути зосереджена на зв’язках. Тому важливі дані елемента містять спосіб ідентифікації певного елемента та інших елементів, з якими він пов’язаний. Здебільшого буде достатньо назви елемента, типу та його зв’язків з іншими елементами. Але тут ми розглянемо всі типи даних, які ми будемо видобувати, пояснимо їхнє призначення та обговоримо, чому вони будуть або чому не будуть вилучені.

Блоки є основною концепцією в моделі SysML, їх можна порівняти з класами на діаграмі класів UML. Блок завжди має назву та може мати опис, значення, операції, частини та порти. Назва важлива для ідентифікації блоку в 3D-моделі, інші частини та порти важливі, але витягуються самостійно, а

також частина внутрішньої блок-схеми, про яку йдеться в наступному розділі.

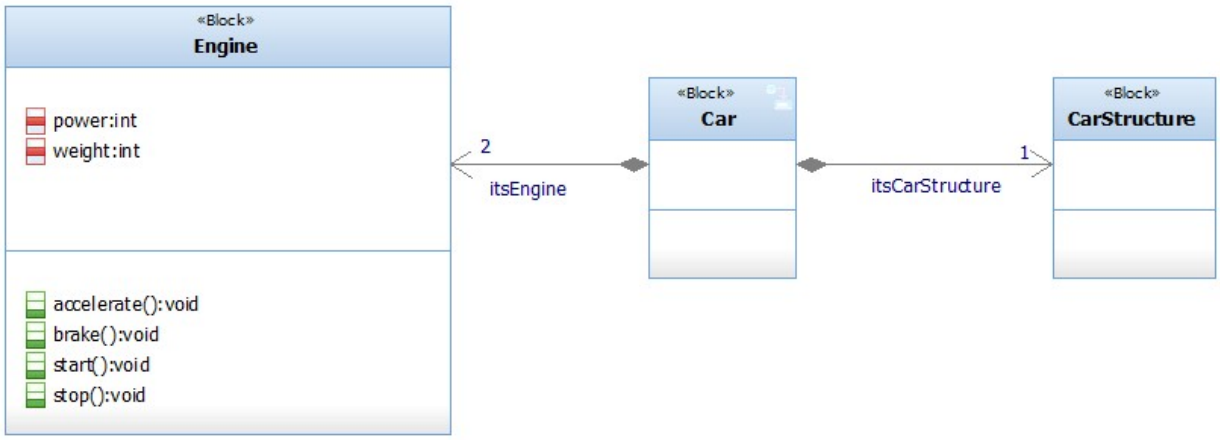


Рис. 3.2. Приклад діаграми структури блоку

Асоціація означає, що зв'язок може існувати між двома примірниками блоків. На рисунку 3.3 представлено приклад асоціації між двома блоками.

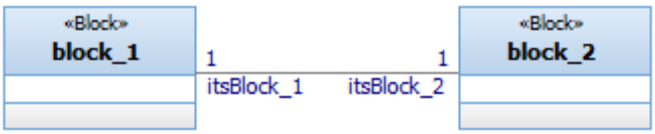


Рис. 3.3. Приклад асоціації

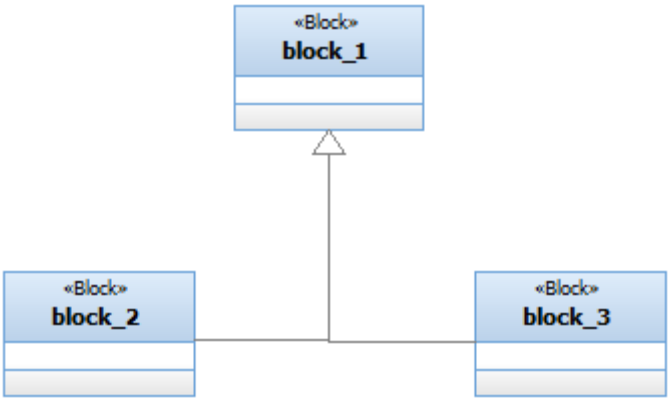


Рис. 3.4. Приклад узагальнення. Блоки 2 і 3 є конкретними реалізаціями блоку 1

Узагальнення - це відношення об’єднує подібні блоки в більш загальний блок. Наприклад, загальний блок може бути банківським рахунком, тоді як блоки спеціалізації можуть бути ощадним рахунком і особистим рахунком. На рисунку 3.4 представлено узагальнення двох блоків і загального блоку.

Зв'язки мають джерело, ціль, тип, назву та, можливо, мітку. Беручи до уваги можливість відстеження, для експорту в 3D-модель важливі лише джерело, ціль і тип.

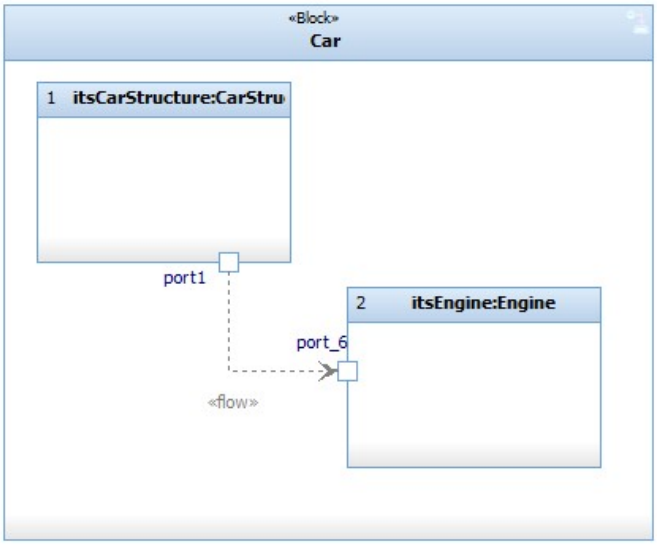


Рис. 3.5. Приклад внутрішньої блок-схеми

Властивість частини — це внутрішня структура блоку. Це означає, що блок має право власності на певну частину. Єдина інформація, яку ми отримуємо з частини, це ім’я та блок власника.

Порт має назву та власника. Для порту витягуємо ім’я та власника (Блок). Існує багато різних типів портів, але це недостатньо релевантно для відстеження, щоб отримати цю інформацію з 3D-моделі.

Потік представляє тип даних, який протікає між двома структурами. У випадку систем ці «дані» також можуть бути фізичними типами потоку, як-от речовина чи енергія. Потік визначається між двома портами, джерелом і

цільовим. Він також може мати опис і властивості, але для відстеження важливими є лише ім’я, джерело та ціль, тому вилучаються лише вони.

Вимога в SysML має досить тривіальне представлення, вона має назву, опис та ID. Усе це витягнуто з бази даних (рис. 3.6).

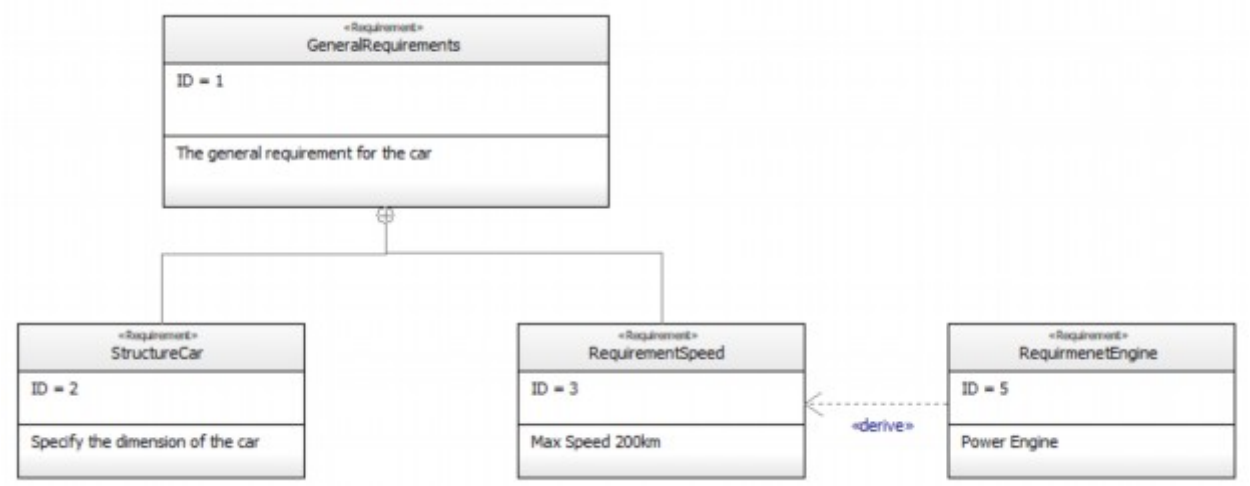


Рис. 3.6. Приклад діаграми вимог

Залежність – це відношення між двома вимогами, вона має джерело та ціль. Далі він має тип. Задля відстеження ми виділяємо лише джерело та цільову вимогу.

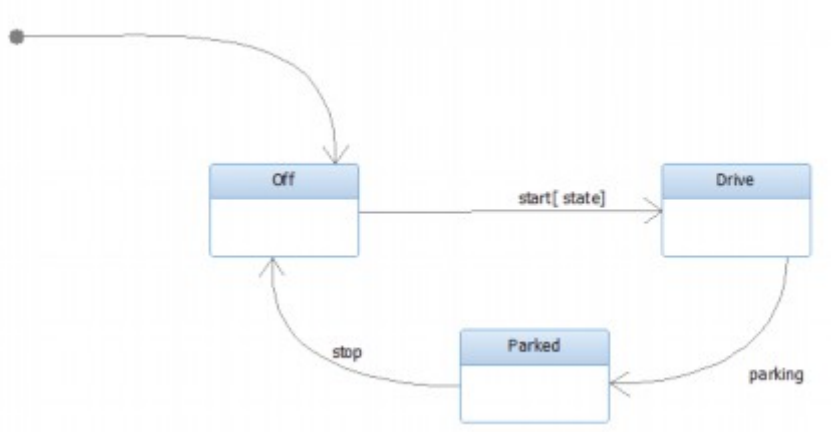


Рис. 3.7. Приклад діаграми кінцевого автомата

Стан — це особлива умова, у якій може перебувати система. Стан має назву й опис, але буде вилучено лише назву.

Перехід стану — це перехід між двома станами. Він має стан джерела та призначення та може мати дію, захист, мітку та тригер. Уся інформація витягується, але не використовується безпосередньо в 3D-моделі.

Випадок використання – це послуга системи, наприклад «запуск контролера» або «зупинка двигуна». Варіант використання може мати назву, опис, значення, операції та порти. Для сфери відстеження вилучається лише назва.

Актор — це роль, яка є користувачем або системою, яка може взаємодіяти з суб'єктом, тобто випадком використання. Актор має ті самі властивості, що й варіант використання, а саме ім'я, опис, значення, операції та порти. Для сфери відстеження вилучається лише назва.

Асоціація показує зв'язок між варіантами використання або між актором і варіантом використання. Асоціація має назву, джерело та ціль. Усе це витягнуто з даних.

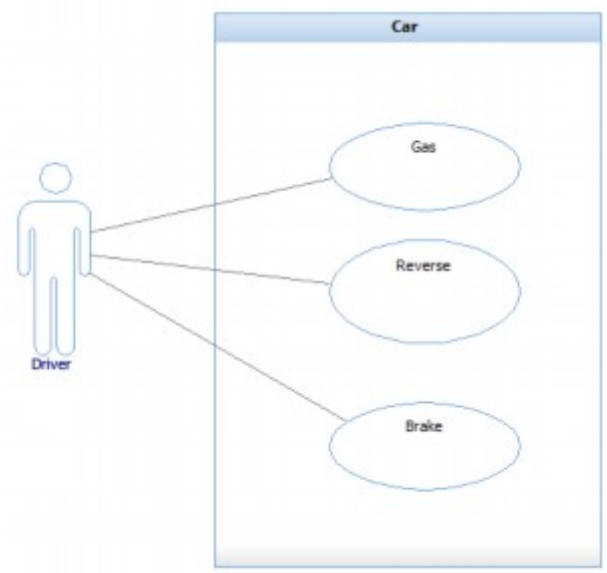


Рис. 3.8. Приклад діаграми варіантів

Використовуючи Rhapsody Java API, ми можемо витягувати дані з бази даних Rhapsody. Щоб використовувати Rhapsody API, ми повинні мати запущену активну програму Rhapsody. У java ми створюємо клас IRPApplication, який ініціалізується як наша активна програма Rhapsody:

```
IRPApplication app;
app = RhapsodyAppServer.getActiveRhapsodyApplication();
```

Тепер, коли активну програму rhapsody підключено до нашої програми Java, ми можемо встановити параметри нашого файлу проекту та встановити його як файл проекту:

```
IRPProject prj;
app.openProject("..\\JUAN\\Juan_V1.rpy");
prj = app.activeProject();
```

Rhapsody API розбивається на об’єкти, повний список представлений у лістингу 3.1. Нас переважно цікавлять IRPModelElements, які відповідають елементам, що розглянуті раніше.

Лістинг 3.1. Фрагмент коду Rhapsody API

```
IRPApplication
IRPASCIIFile
IRPCollection
IRPExternalCodeGenerator
IRPExternalCodeGeneratorInvoker
IRPFlow
IRPGraphElement
    IRPGraphEdge
    IRPGraphNode
IRPGraphicalProperty
IRPModelElement
    IRPAction
    IRPAnnotation
        IRPComment
        IRPConstraint
        IRPRequirement
    IRPAssociationRole
    IRPClassifierRole
    IRPCollaboration
    IRPComponentInstance
    IRPConfiguration
    IRPDependency
        IRPHyperLink
    IRPEnumerationLiteral
    IRPExecutionOccurrence
    IRPFileIRPGeneralization
    IRPGuard
    IRPInteractionOccurrence
    IRPInterfaceItem
```

```
            IRPUseCase
        IRPComponent
        IRPDiagram
            IRPCollaborationDiagram
            IRPComponentDiagram
            IRPDeploymentDiagram
            IRPObjectModelDiagram
            IRPSequenceDiagram
            IRPStatechart
            IRPFlowchart
            IRPStructureDiagram
            IRPUseCaseDiagram
        IRPPackage
            IRPProfile
            IRPProject
        IRPRelation
            IRPInstance
            IRPBlock
            IRPModule
            IRPPort
    IRPVariable
        IRPArgument
        IRPAttribute
        IRPTag
        IRPTemplateParameter
```

Фактичне видобування даних досить просте: ми спочатку витягуємо всі пакети, потім проходимо цикл по пакетах і витягуємо діаграму за допомогою команди get. Наприклад, згідно наступного алгоритму видобування даних:

```
ArrayList packages = project.getPackages();
foreach pack in packages do
    ArrayList useCaseDiagrams = pack.getUseCaseDiagrams();
    foreach diagram in useCaseDiagrams do
        print diagram elements;
    end
end
```

3.3. Представлення методики передачі даних від Data Extractor до Unity

Модуль комунікації в конверсійному шарі надсилає вихідні дані модуля вилучення даних до модуля обробки даних. Викликом у цій комунікації є те, що вона є міжмовна. Модуль вилучення даних написаний на Java, а модуль обробки даних написаний на C#. Ці мови не були обрані довільно. Модуль вилучення даних побудований на Rhapsody Java API, тому

сама програма повинна бути написана на Java. Крім того, модуль обробки даних розширює середовище Unity. Середовище Unity приймає лише C#, UnityScript і Boo. C# є найбільш використовуваною мовою і має найбільше класів розширення. Також документація Unity в основному написана для C#, тому C# є логічним вибором. Огляд ідеї комунікації представлено на рисунку 3.9.

Рис.3.9. Зв’язок між модулем вилучення даних і модулем обробки даних

Вибір між активною та пасивною комунікацією має бути зроблений. При активній комунікації ми маємо на увазі реальне часове з'єднання між модулем вилучення даних і модулем обробки даних. При пасивній комунікації ми маємо на увазі структуру, де дані зберігаються в якомусь місці, щоб модуль вилучення даних і модуль обробки даних могли бути виконані асинхронно. У цьому проекті були досліджені обидва ці підходи і будуть детально розглянуті в наступних підрозділах.

3.3.1. Активний взаємозв’язок між модулями

При активному зв’язку має бути встановлено пряме з’єднання між модулем вилучення даних і модулем обробки даних. Оскільки вони написані іншою мовою програмування, варіанти обмежені. У цьому проекті було розглянуто та протестовано TCP-з'єднання між двома модулями. У такому підході модуль вилучення даних діятиме як TCP-сервер, а модуль обробки даних — TCP-клієнт. Перевага цього підходу полягає в тому, що зміни в діаграмі SysML у реальному часі будуть негайно передані Unity. Це корисно, якщо в Unity відображатиметься не лише структура, але й поведінка.

Недоліком використання цього підходу є те, що дві програми, модуль вилучення даних і модуль обробки даних, завжди мають працювати одночасно. Схема цього підходу представлена на рис 3.10.

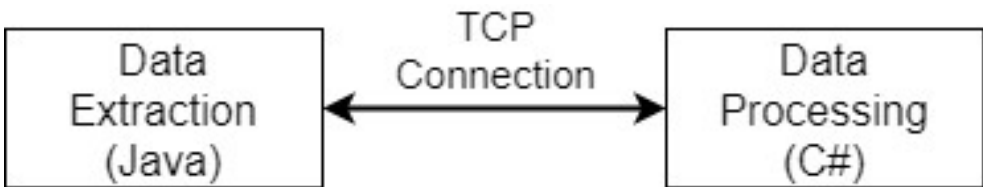


Рис. 3.10. Активний взаємозв’язок між модулем видобування і модулем обробки даних

3.3.2. Пасивний взаємозв’язок між модулями

Для пасивного зв’язку ми повинні знайти спосіб зберігати дані, отримані з моделі SysML, щоб модуль обробки даних міг читати ці дані. Для цього підійде багато різних методів. У цьому проекті ми використовуємо файли зі значеннями, розділеними комами (CSV), оскільки єдиними даними, які надсилаються, є рядки. Ці файли легко програмувати, їх можна писати та читати як Java, так і C#. Схема цього підходу представлена на рис 3.11.

Рис. 3.11. Пасивний взаємозв’язок між модулем видобування і модулем обробки даних

Дві переваги цього підходу полягають у тому, що його легко програмувати, і обидва модулі (вилучення даних і обробка даних) можна запускати асинхронно. Спосіб експорту для кожного типу елемента створюється окремий файл CSV. Цей файл має певну структуру. Як приклад візьмемо довільну діаграму вимог, представлену на рисунку 3.12.

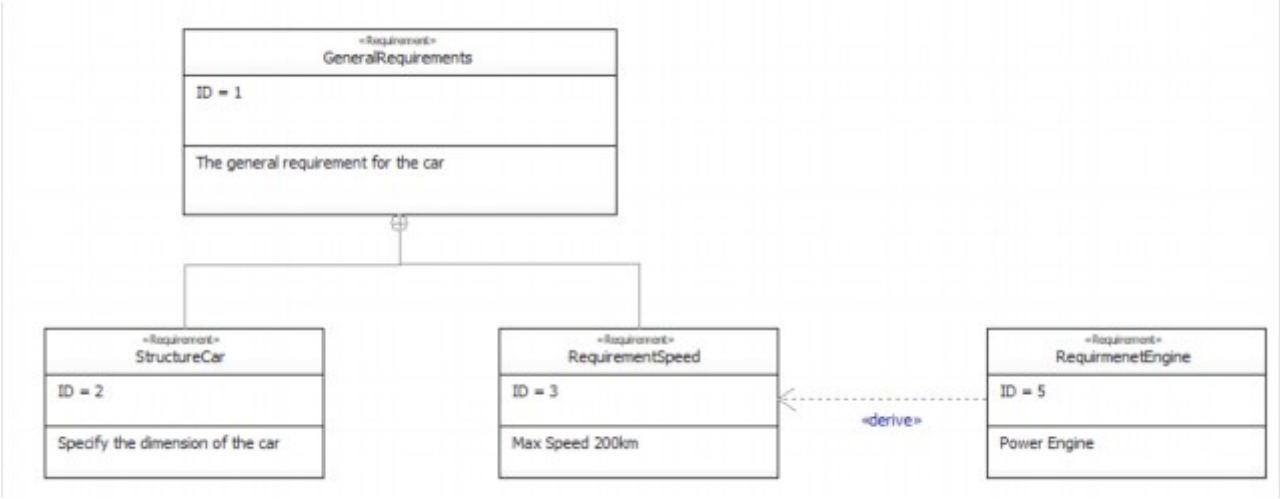


Рис. 3.12. Діаграма вимог

Ця діаграма має два типи елементів: вимоги та залежності вимог. Файл вимог має структуру «ім'я, опис, ідентифікатор», і кожен рядок представляє новий елемент. Переклад вимог до діаграми, представленої на рисунку 3.12 до файлу CSV представлено на рисунку 3.13.

	A
1	GeneralRequirements,The general requirement for the car,1
2	StructureCar,Specify the dimension of the car,2
3	RequirementSpeed,Max Speed 200km,3
4	RequirmenetEngine,Power Engine,5

Рис. 3.13. Приклад файлу CSV для елементів типу Requirement

Залежності вимог мають окремий файл, який містить підключені вимоги. Цей файл має структуру «Від, До» з вихідною вимогою та цільовою вимогою. Трансляція залежностей вимог на діаграмі, представленій на рисунку 3.12 у файл CSV представлено на рисунку 3.14 .

	A
1	RequirmenetEngine,RequirementSpeed

Рис. 3.14. Приклад файлу CSV для елементів типу Requirement Dependency

3.4. Процеси обробки та абстрагування даних

У цьому розділі представлено структуру сценаріїв обробки даних, розширених до Unity. Ці сценарії використовують дані, доступні у файлах CSV, і створюють їх структуру даних. По суті, усі діаграми складаються виключно з вузлів і шляхів. Ці вузли та шляхи мають різні назви, властивості та форми, але вони мають спільне, що свідчить про абстрактне представлення рівності. Крім того, цю абстракцію можна використовувати для розширення програми для легкої обробки більшої кількості типів діаграм. Розділ обробки даних розділений на дві частини. Спочатку представлена абстракція структури програмного забезпечення. По-друге, ми поговоримо про структуру даних для всіх діаграм. Включені діаграми: діаграма кінцевого автомата, діаграма вимог, діаграма визначення блоку, внутрішня блок-схема та діаграма варіантів використання. Ми також пояснимо, що робить кожна функція в класі.

По суті, усі діаграми складаються виключно з вузлів і шляхів. Представлено класи абстракції для вузла, шляху та діаграми. Це абстрактні класи, що означає, що кожен клас у специфікації може мати власні властивості та 3D-візуалізацію.

Вузли представляють усі «Об’єкти» на діаграмах. В Unity всі вузли мають спільне ім’я String і префаб GameObject. Усі вони повинні мати функції для обох. Усі вузли на нашій діаграмі в Unity мають мати своє ім’я над GameObject, тому всі вузли мають функцію drawtext(). Діаграма класів для цієї абстракції представлена на рисунку 3.15.

На діаграмі показано, які об’єкти класифікуються як вузли, але вони будуть перераховані нижче для кожної діаграми відповідно:

- BDD: Блоки.
- IBD: частини та порти.
- StateMachineElements: діаграма кінцевого автомата.
- UseCaseElements: діаграма варіантів використання.

• Requirement Diagram: вимоги.

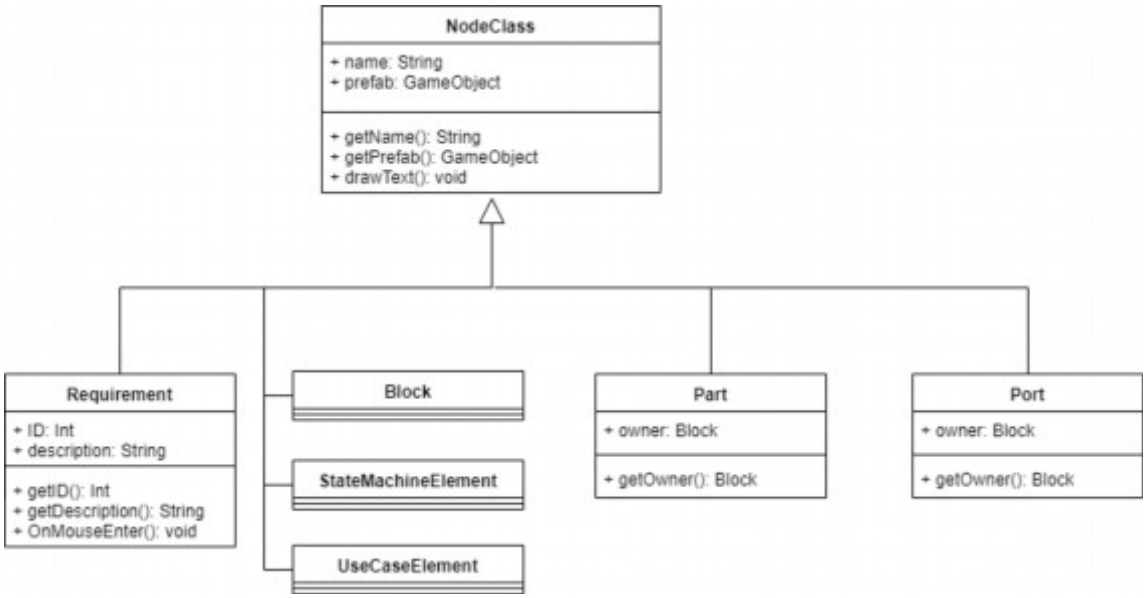


Рис. 3.15. Абстракція класу вузла

Шляхи представляють зв’язки між вузлами на діаграмах. Шляхи NodeClass від і NodeClass до якого є джерелом і метою шляху. Потім, як і Node Class, вони мають префаб GameObject для відображення шляху. PathClass має лише такі функції get, як getFrom(), getTo(), getGameObject(). Діаграма класів для цієї абстракції представлена на рисунку 3.16.

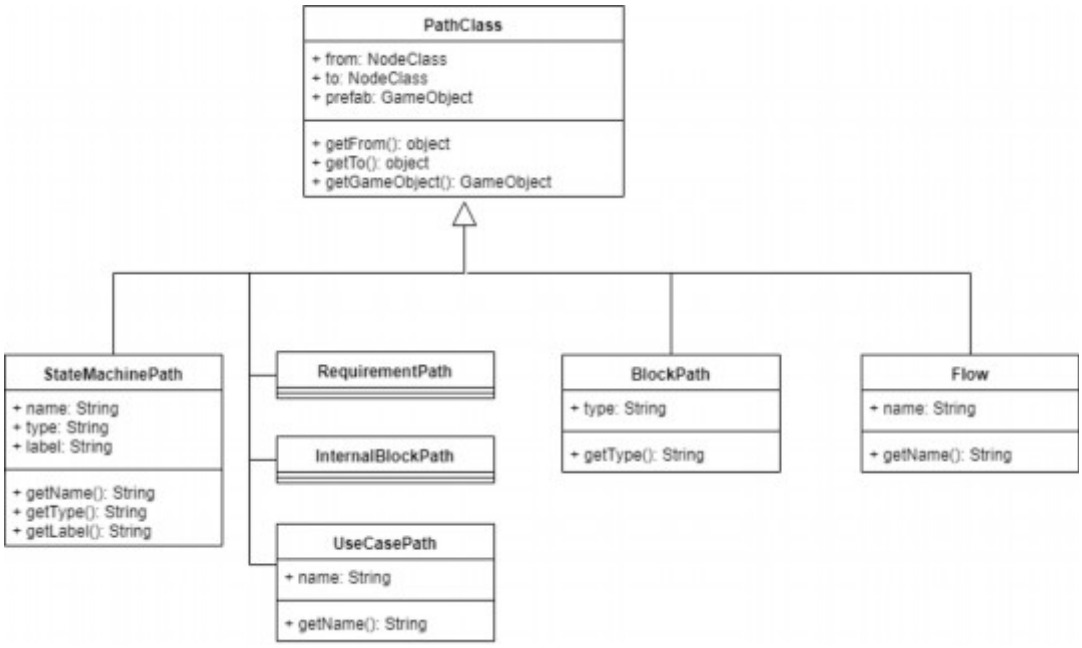


Рис. 3.16. Абстракція класу переходу

На діаграмі показано, які об’єкти класифікуються як шляхи, але вони будуть перераховані нижче для кожної діаграми відповідно:

• BDD: BlockPath.

• IBD: Flow і InternalBlockPath.

• State Machine Diagram: StateMachinePath.

• Use Case Diagram: UseCasePath.

• Requirement Diagram: RequirementPath.

Діаграми — це об’єкти, які містять усі об’єкти NodeClass і TransitionClass свого типу. Тому клас DiagramClass складається з двох вузлів ArrayLists і переходів, які містять усі об’єкти NodeClass і TransitionClass. Крім того, вони мають функції додавання та отримання, а саме addNode(NodeClass), addTransition(TransitionClass), getNodes() і getTransitions(). Діаграма класів для цієї абстракції представлена на рис. 3.17.

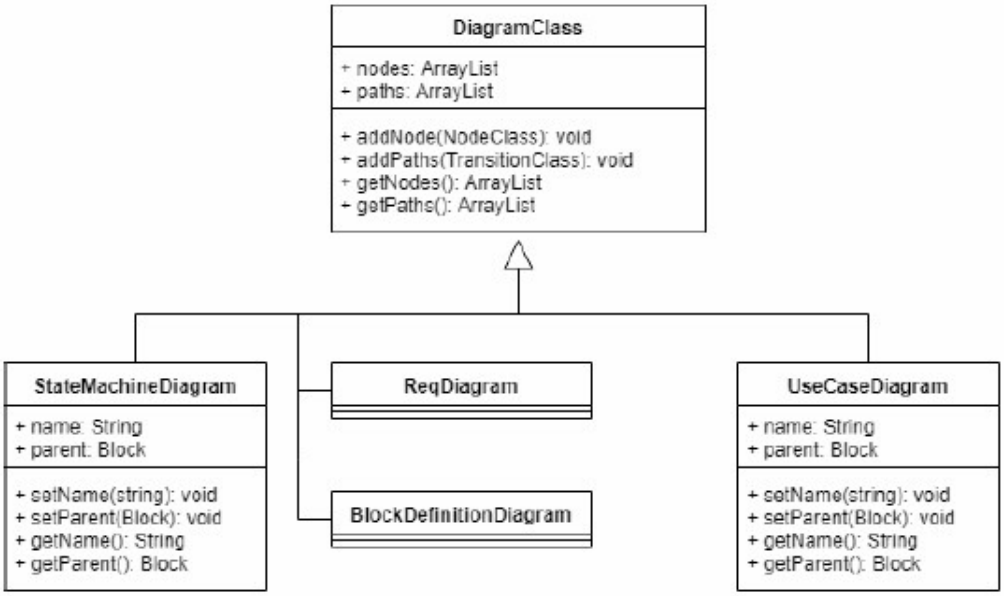


Рис. 3.17. Абстракція класу діаграми

3.5. Опис структур даних та представлення діаграм класів

Щоб створити діаграму в Unity, ми повинні створити порожній GameObject. До GameObject додається сценарій, який обробляє вилучення

даних із файлів CSV. Для кожної групи сутностей на діаграмі ми створюємо окремий клас. Наприклад, у діаграмі визначення блоку ми маємо класи Block і BlockRelation , які є розширеними з класів абстракції, про які йшлося раніше. Для кожної сутності автоматично створюється попередньо сконфігурований елемент.

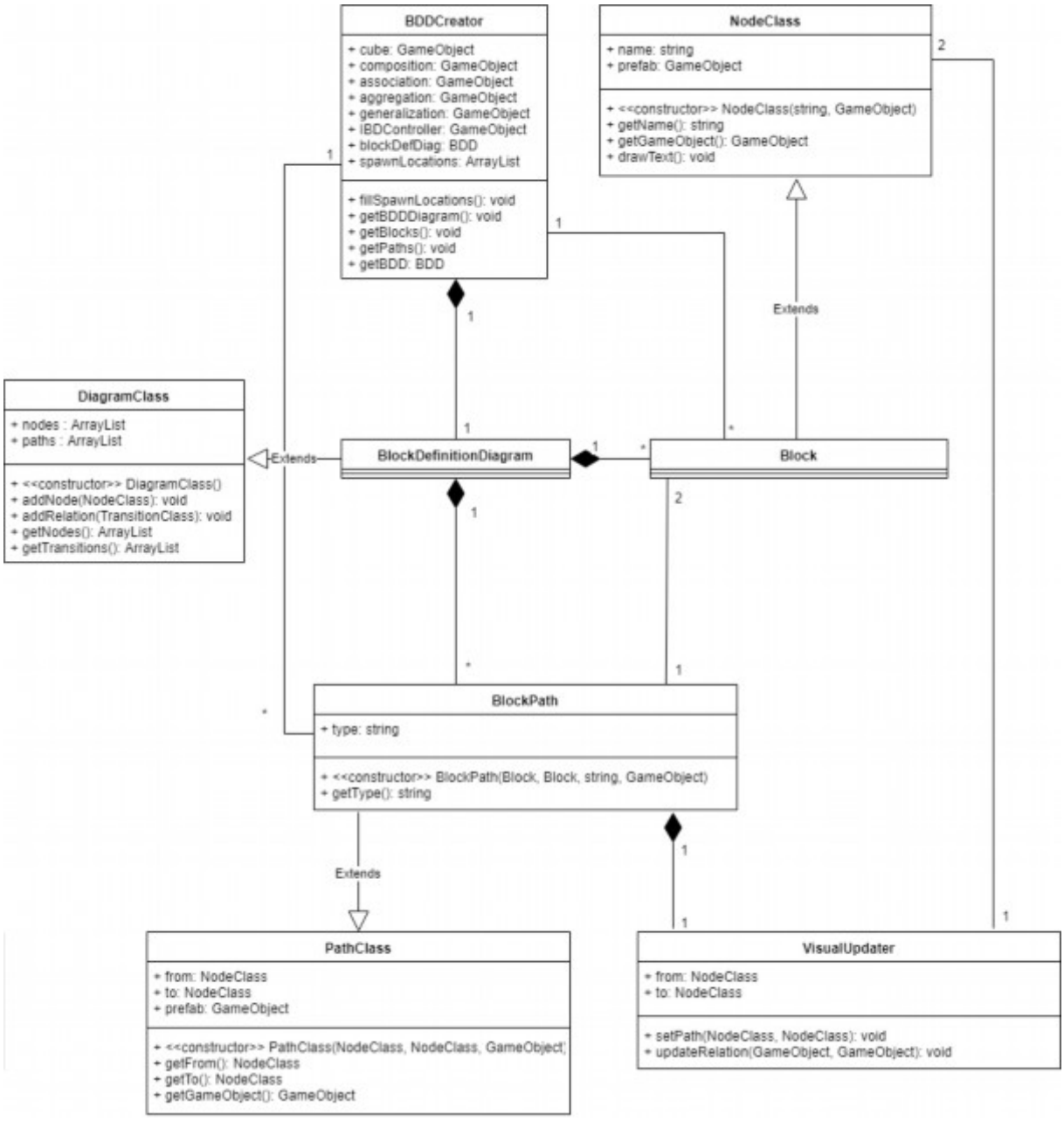


Рис. 3.18. Діаграма класів діаграми визначення блоку в Unity

Для кожної діаграми ми створюємо структуру класу, яка охоплює отримання даних із файлів CSV, створення об’єктів із витягнутих даних і

тривимірну візуалізацію витягнутих даних. Діаграма класів для BDD представлена на рисунку 3.18.

BDDCreator отримує дані з файлів CSV і створює діаграму визначення блоку. Кожен елемент у діаграмі визначення блоку ініціалізується відповідним префабом. У нас є різні префаби для куба, асоціації, композиції, узагальнення, агрегації та залежності. Вилучення даних виконується за допомогою StreamReader, який є стандартною функціональністю в C#. Ми визначаємо розташування файлу .csv і записуємо його в потік.

Кожен рядок у файлі csv відповідає блоку. З цими даними виконуються наступні кроки:

1. Створення екземпляру GameObject із префабом «Блок» у випадковому місці сцени.

2. Створення нового блоку, що містить дані з файлу csv і раніше створеного об’єкта GameObject, щоб вони були пов’язані разом.

3. Додавання блоку до діаграми визначення блоку за допомогою addBlock (Block).

Структура шляху різна. У блоку зберігається лише ім’я, тоді як у шляхах є ім’я , блок « від » і «до» . Після вилучення даних із файлу csv виконуються такі кроки:

1. Перегляд всіх блоків чиє ім’я відповідає імені «від» і «до» у файлі csv, якщо так, то встановлюється блок «від» і «кому» на відповідний блок.

2. Якщо «Від» і «До» не дорівнюють нулю, то перевіряється тип зв’язку (композиція, агрегація, асоціація, залежність, узагальнення). Якщо жоден тип не відповідає, використовується регістр за замовчуванням «Асоціація».

3. Створення екземпляра GameObject з відповідним типом відношення Prefab.

4. Створення нового BlockRelation з Block, to Block і назвою та об’єктом GameO.

5. Додавання BlockRelation до діаграми визначення блоку за допомогою addRelation(BlockRelation.

BDD — це клас даних, який містить інформацію про всі блоки та шляхи блоків. Клас є розширенням DiagramClass і тому має ArrayList вузлів і шляхів.

• addNode(Block) додає блок до ArrayList.

• addPath(BlockRelation) додає відношення до ArrayList.

• getPaths() повертає відношення ArrayList.

Клас Block безпосередньо пов’язаний з Block GameObject. Він зберігає назву блоків і малює текст над GameObject. Клас є розширенням NodeClass, тому має назву (String) і префаб (GameObject). Конструктор встановлює назву та GameObject. Відразу після ініціалізації він викликає drawText(). Block має три функції, всі три походять від NodeClass:

• getName() повертає назву блоку.

• getGameObject() повертає GameObject, пов’язаний із блоком.

• drawText() малює текст безпосередньо над GameObject у сцені.

Клас BlockPath безпосередньо пов’язаний із типом Relation GameObject. Клас є розширенням PathClass . Він зберігає вихідний блок і цільовий блок. Крім того, він має тип шляху, який відповідає типу префабу. У конструкторі BlockPath(Block, Block, string, GameObject) встановлюється вихідний і цільовий блок, встановлюється тип і зв’язок GameObject зв’язується з класом. Далі конструктор встановлює ціль джерела та тип для класу «VisualUpdater», який обробляє 3D-візуалізацію та положення шляху. Block-Path має чотири функції, з яких getFrom(), getTo() і getGameObject() успадковані від PathClass:

• getFrom() повертає вихідний блок.

• getTo() повертає цільовий блок.

• getType() повертає тип BlockRelation.

• getGameObject() повертає GameObject, пов’язаний із BlockRelation.

Решта діаграм представлено нижче.

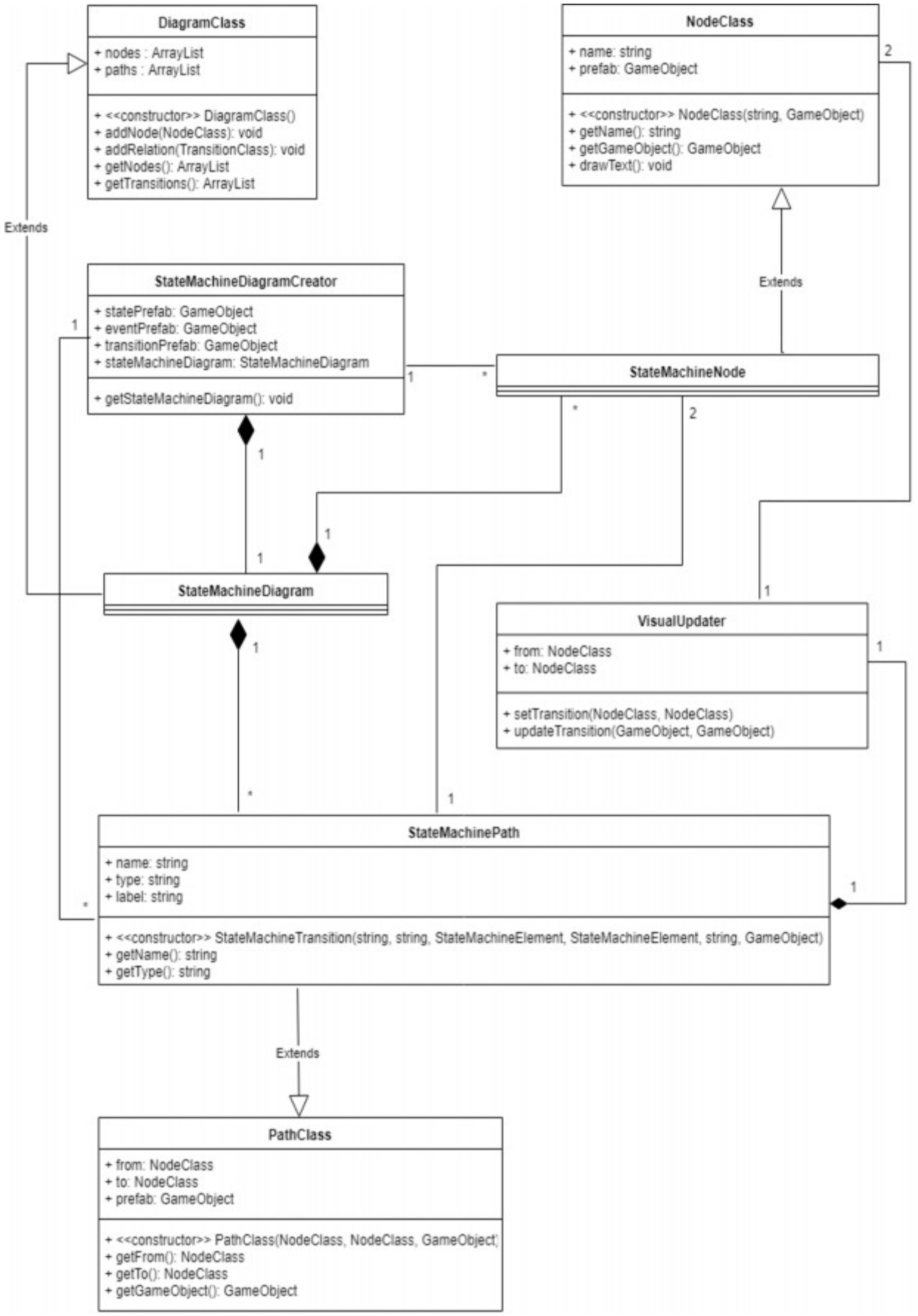


Рис. 3.19. Діаграма класів діаграми кінцевого автомата в Unity

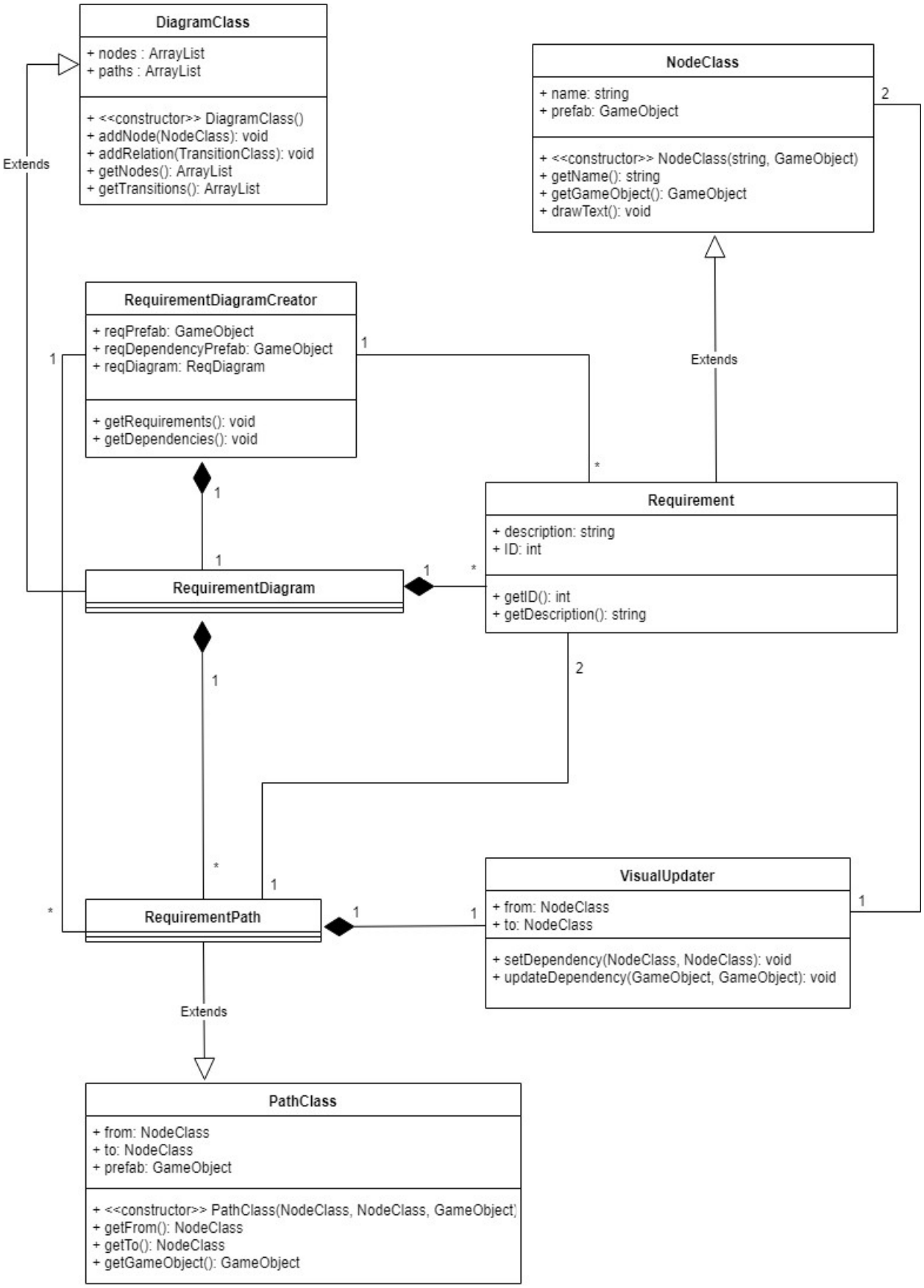


Рис. 3.20. Діаграма класів діаграми вимог в Unity

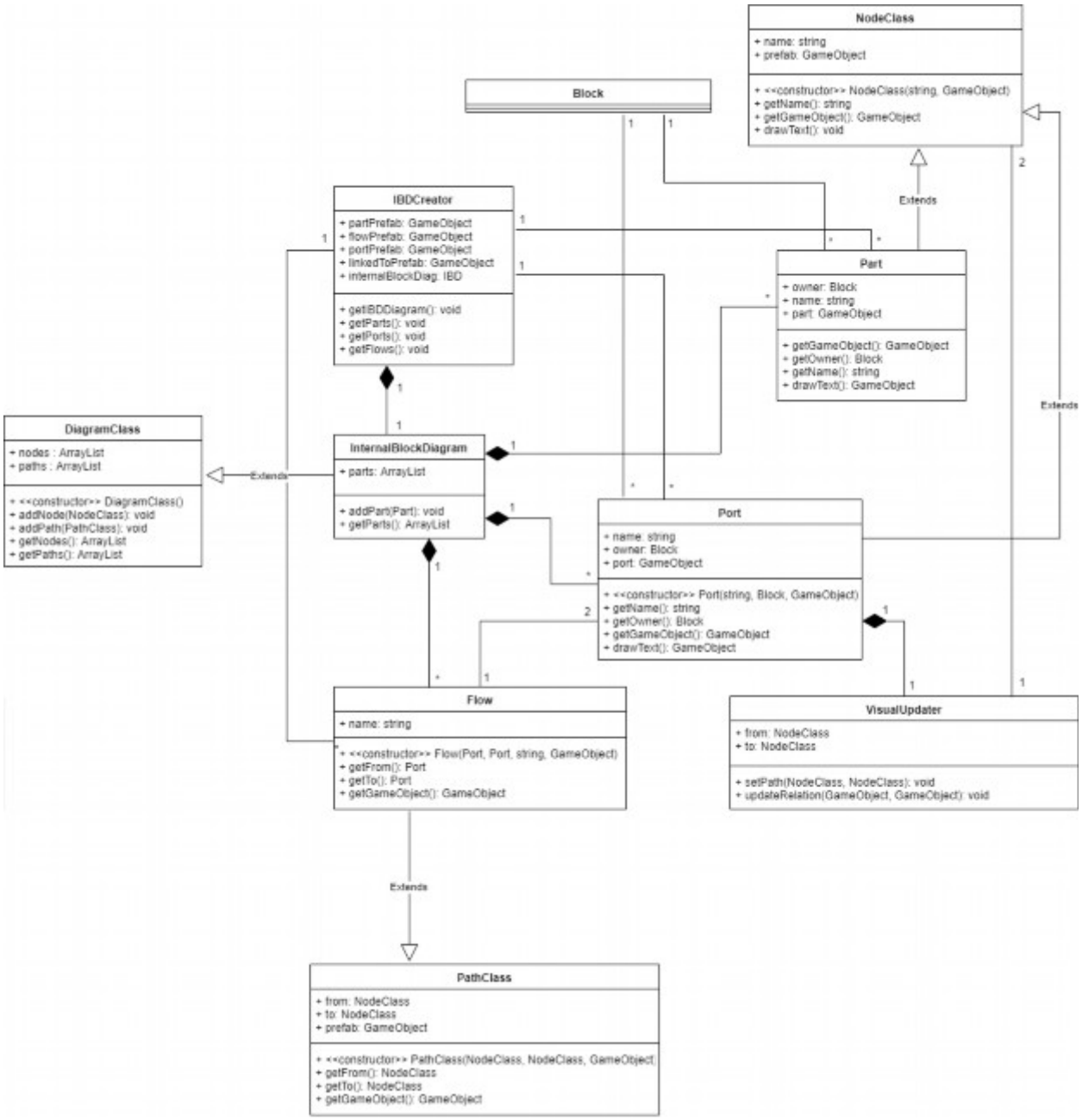


Рис. 3.21. Діаграма класів внутрішньої блок-схеми в Unity

Клас InternalBlockDiagram містить інформацію про всі порти, частини та потоки. Це зберігається в трьох ArrayLists, які ініціалізуються в конструкторі. Клас є абстрактованим від DiagramClass. Внутрішня блок-схема має не лише вузли та шляхи, а й частини, це атрибут розширення у InternalBlockDiagram. Цей клас має шість функцій, з яких addNode(), addPath(), getNodes() і getPaths() успадковані від DiagramClass:

- addNode(Port) додає порт до ArrayList.
- addPath(Flow) додає Flow до ArrayList.

• addPart(Part) додає Частину до ArrayList.

• getNodes() повертає вузли ArrayList .

• getPaths() повертає шляхи ArrayList .

• getParts() повертає частини ArrayList .

Клас Port пов’язаний з Port GameObject і містить інформацію про Port. Він походить від NodeClass, тому має ці атрибути та операції. Крім того, порт має блок власника. Клас Port має чотири функції, з яких getName(), getGameObject() і drawText() успадковані від NodeClass:

• getOwner() повертає власника (блок) цього порту.

• getName() повертає цю назву портів.

• getGameObject() повертає GameObject, пов’язаний із цим портом.

• drawText() малює текст безпосередньо над GameObject у сцені.

Клас Flow пов’язаний з Flow GameObject і містить інформацію Flows. Цей клас є розширеним класом PathClass. Він зберігає вихідний порт і цільовий порт. Крім того, потік може мати назву. Клас Flow має чотири функції, з яких getFrom(), getTo() і getGameObject() успадковані від PathClass:

• getName() повертає назву потоку.

• getFrom() повертає вихідний порт.

• getTo() повертає цільовий порт .

• getGameObject() повертає GameObject, пов’язаний із потоком.

Клас VisualUpdater обробляє візуалізацію всіх шляхів. Той самий клас використовується в BlockDefinitionDiagram і має таку саму функціональність.

Отже, ми визначили структуру системи і показали, як можна отримати дані з інструменту CASE, такого як Rhapsody. Велике значення має релевантність тих чи інших даних. Це вилучення даних можна розширити, використовуючи той самий метод, що представлений на різних типах моделей або елементах моделі. Далі ми обговорили спосіб зв’язку від модуля «Вилучення даних» до модуля «Обробка даних». Основний вибір – між пасивним і активним спілкуванням, причому проблема полягає в тому, що обидва модулі мають різні мови програмування. Для цього проекту ми

використовуємо пасивний зв’язок із файлами CSV. Є кілька цікавих переваг і випадків використання активного спілкування, які можна вивчити в майбутній роботі. Окрім вилучення та передачі даних, ми представили структуру модуля «Обробка даних». Цю структуру можна використовувати для розширення модуля «Обробка даних» за допомогою додаткових діаграм.

3.6. Реалізація структури шару конверсії моделей між IBM Rhapsody та Unity

У цьому розділі розглядається структура рівня перетворення між IBM Rhapsody та Unity. Також представлено деякі результати прототипу 3D візуалізації. Однак основна мета прикладів — перевірити рівень перетворення. Метою є не створення ідеальної 3D-візуалізації моделі. Для отримання результатів ми візьмемо запущений приклад моделі, розглянутої в розділі 2 і перетворіть це на 3D-модель.

Для зручності витягування даних представимо перетворення пакета Rhapsody у файл .xlsx. На рисунку 3.22 показано пакет у Rhapsody. Витягнувши дані, ми створюємо файл, представлений на рисунку 3.23.

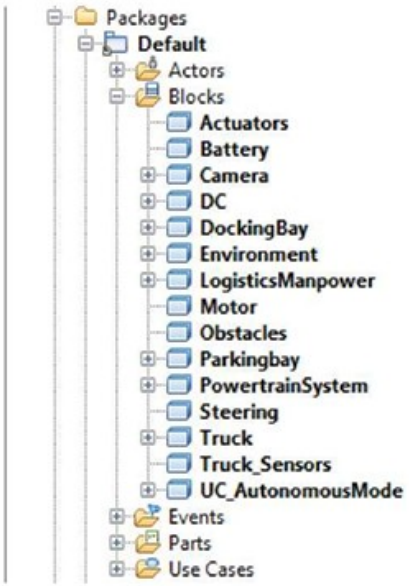


Рис. 3.22. Пакет структури в Rhapsody

	A	B	C	D
1	Name	Metaclass	Owner/parent	Identifier
2	ROOT	State	statechart_0	com.telelogic.rhapsody.core.RPState@a336fa
3	itsEnable Autonomous Mode	AssociationEnd	Driver	com.telelogic.rhapsody.core.RPRelation@fb3d5ba8
4	actor_14	Actor	Default	com.telelogic.rhapsody.core.RPActor@e8deb29f
5	Truck_actor	Actor	Default	com.telelogic.rhapsody.core.RPActor@a1eab7b3
6	itsEnable Autonomous Mode	AssociationEnd	Truck_actor	com.telelogic.rhapsody.core.RPRelation@bef5faee
7	itsTurn Power Off	AssociationEnd	Truck_actor	com.telelogic.rhapsody.core.RPRelation@f755486a
8	itsDisable autonomous mode	AssociationEnd	Truck_actor	com.telelogic.rhapsody.core.RPRelation@ace57898
9	itsAvoid Obstacles	AssociationEnd	Truck_actor	com.telelogic.rhapsody.core.RPRelation@6833f73
10	DC_Actor	Actor	Default	com.telelogic.rhapsody.core.RPActor@cd264b9e
11	itsParking	AssociationEnd	DC_Actor	com.telelogic.rhapsody.core.RPRelation@a59c01e8
12	itsDriving	AssociationEnd	DC_Actor	com.telelogic.rhapsody.core.RPRelation@ab8f2764
13	Engine_Off	State	statechart_0	com.telelogic.rhapsody.core.RPState@23f752d6
14	itsDocking	AssociationEnd	DC_Actor	com.telelogic.rhapsody.core.RPRelation@aa1dce4b
15	Environment_Actor	Actor	Default	com.telelogic.rhapsody.core.RPActor@87b08729
16	itsDetect Obstacles	AssociationEnd	Environment_Actor	com.telelogic.rhapsody.core.RPRelation@27ee565b
17	Camera_Actor	Actor	Default	com.telelogic.rhapsody.core.RPActor@40fec365
18	itsDetect Obstacles	AssociationEnd	Camera_Actor	com.telelogic.rhapsody.core.RPRelation@14f7b3f3
19	Engine_Status	State	statechart_0	com.telelogic.rhapsody.core.RPState@b52b2cc6
20	Engine_On	State	statechart_0	com.telelogic.rhapsody.core.RPState@db5f994b
21	obstaclepresent	State	statechart_0	com.telelogic.rhapsody.core.RPState@d500cde4
22	0	DefaultTransition	statechart_0	com.telelogic.rhapsody.core.RPTransition@72a074be
23	1	Transition	statechart_0	com.telelogic.rhapsody.core.RPTransition@830537b8

Рис. 3.23. Частина файлу вилучення даних із стандартного пакета

Під час вилучення діаграм усі елементи зберігаються в іншому файлі. Давайте візьмемо діаграму визначення блоку на рисунку 3.24.

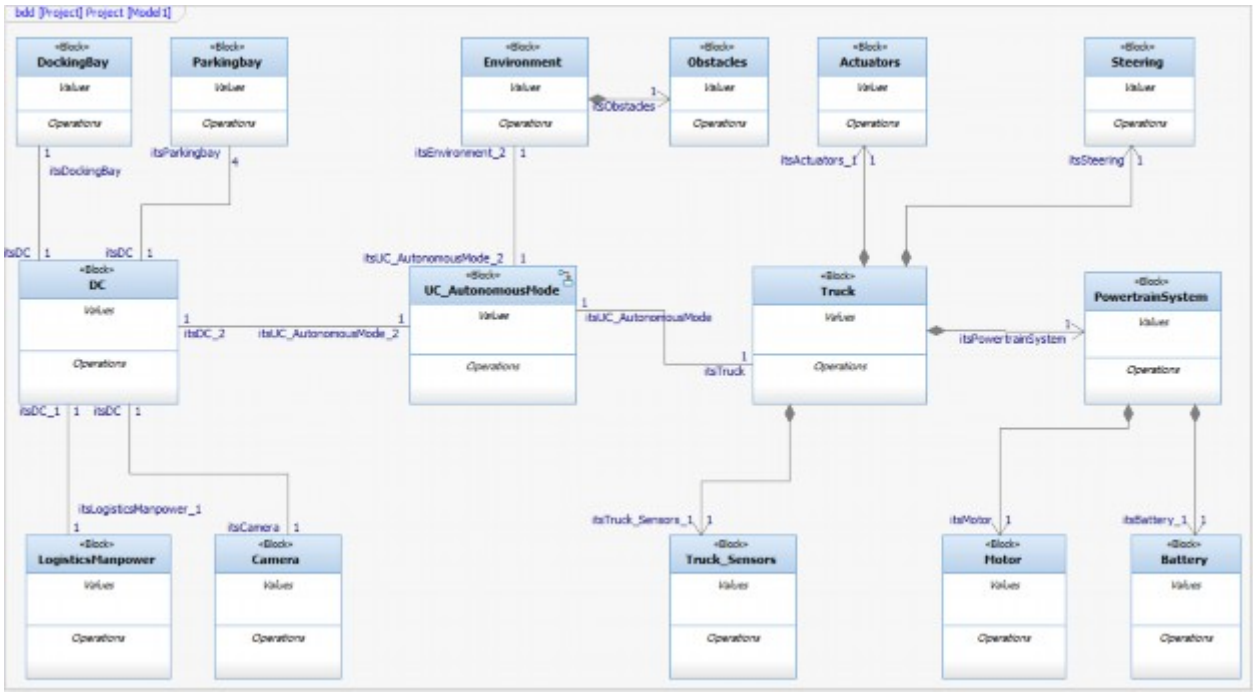
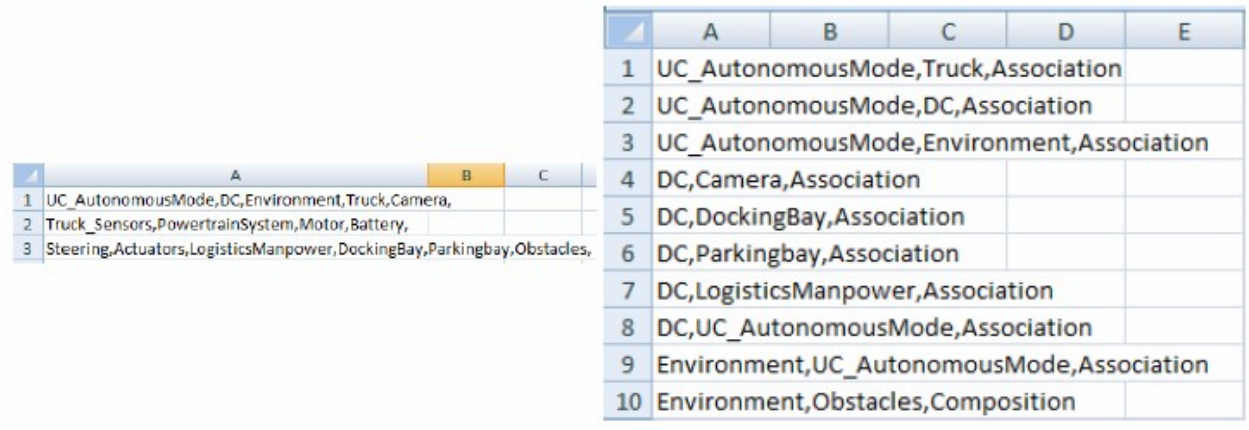


Рис. 3.24. Приклад використовуваної блок-схеми

На цій діаграмі є блоки та шляхи блоків. Файл Blocks містить просто ім’я, розділене комами, як показано на рисунку 3.25a. Кожен рядок файлу

BlockPaths містить інший шлях. Структура цього рядка: вихідний блок, цільовий блок, тип шляху. Файл шляху представлено на рисунку 3.25 б. Це працює однаково для всіх інших діаграм, але інші діаграми можуть мати дещо іншу структуру файлу через різну видобуту інформацію.



	A
1	UC_AutonomousMode,DC,Environment,Truck,Camera,
2	Truck_Sensors,PowertrainSystem,Motor,Battery,
3	Steering,Actuators,LogisticsManpower,DockingBay,Parkingbay,Obstacles,

	A	B	C	D	E
1	UC_AutonomousMode,Truck,Association				
2	UC_AutonomousMode,DC,Association				
3	UC_AutonomousMode,Environment,Association				
4	DC,Camera,Association				
5	DC,DockingBay,Association				
6	DC,Parkingbay,Association				
7	DC,LogisticsManpower,Association				
8	DC,UC_AutonomousMode,Association				
9	Environment,UC_AutonomousMode,Association				
10	Environment,Obstacles,Composition				

а) Блокує файл CSV б) Частина CSV файлу

Рис. 3.25. Файли діаграми визначення блоку

3.6.1. Процес візуалізації даних моделі

Тепер розглянемо процес візуалізації даних реалізованого в Unity. Сама тривимірна візуалізація складається виключно з простих кубів для вузлів і конусів для шляхів. Для цього прикладу буде використана модель автомобіля. Модель складається з діаграми визначення блоку, діаграми вимог, діаграми варіантів використання та діаграми кінцевого автомата.

На діаграмі визначення блоку блок відображається як тривимірний куб. Для доріжки використовуються різні збірні конструкції, які можуть відрізнятися за кольором або формою. Збірні конструкції представлені на рисунку 3.26.

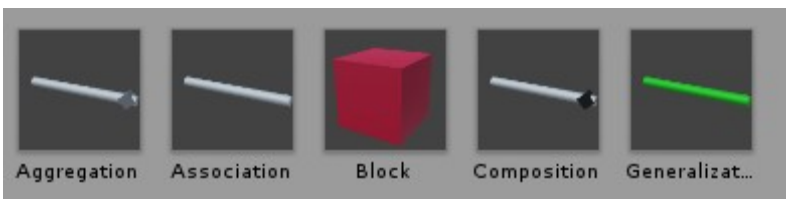


Рис. 3.26. Збірні блоки діаграми визначення блоку

3D-кубики розміщуються у випадковій сітці на сцені, щоб блоки не мали однакового положення. Ці блоки можна перетягувати за допомогою миші, щоб створити оптимальний вигляд системи. Діаграма визначення блоку на рисунку 3.24 перетворюється на 3D-модель. Це показано на рисунку 3.27.

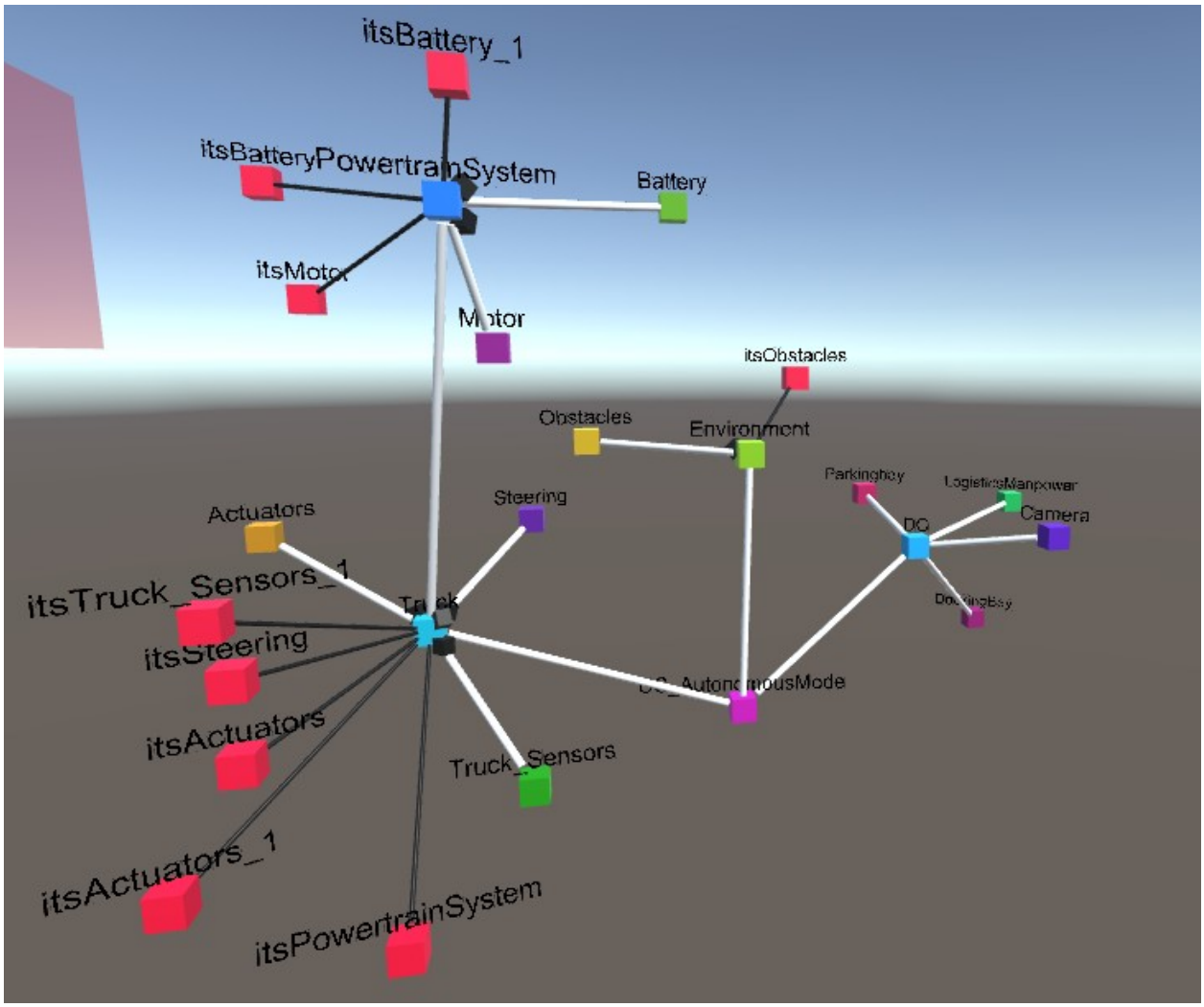


Рис. 3.27. Приклад діаграми визначення блоку в Unity

3.6.2. Візуалізація діаграми вимог

Діаграми вимог зазвичай представляють абстрактні концепції, взаємозв'язки та функціональні вимоги системи. Вони зазвичай зображуються у двовимірному просторі за допомогою таких нотацій як UML (Unified Modeling Language), BPMN (Business Process Model and Notation) та інших

Діаграма вимог складається з вимог і шляхів вимог. Префаби діаграми вимог визначаються так само, як діаграма визначення блоку, але вона має лише один тип префабів шляху та різні кольори. Збірні конструкції представлені на рисунку 3.28.

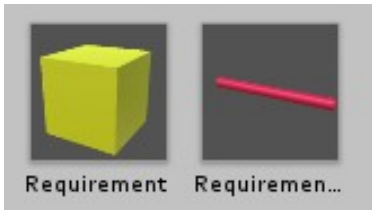
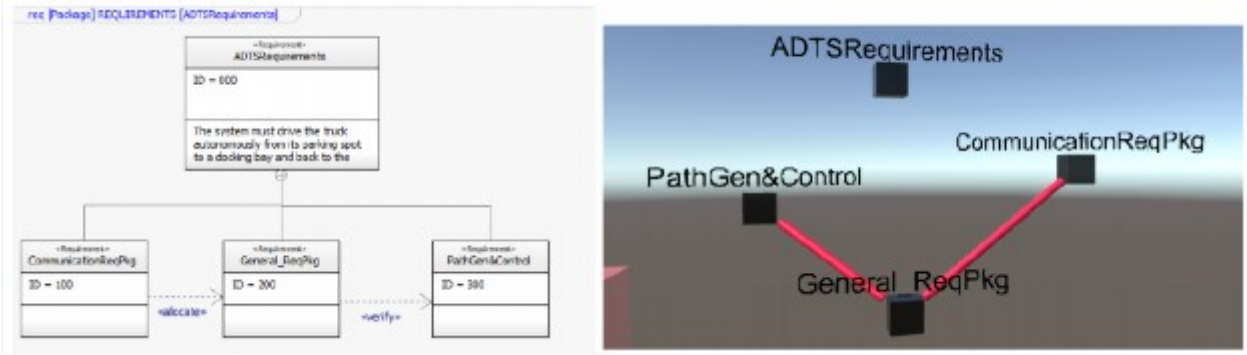


Рис. 3.28. Збірні конструкції діаграми вимог

Усі шляхи розподілу, перевірки та задоволення вважаються RequirementPath. Це означає, що всі вони матимуть однаковий збірний модуль у 3D-моделі. Для прикладу перетворення ми візьмемо діаграму вимог, представлену на рисунку 3.29а і 3D-перетворення в Unity представлено на рисунку 3.29б.



а) Приклад діаграми вимог б) Візуалізація діаграми вимог

Рис. 3.29. Перетворення діаграми вимог

3.6.3. 3D візуалізація діаграми варіантів використання

Приклад діаграми варіантів використання представлено на рисунку 3.30. Перетворення прикладу діаграми варіантів використання представлено на рисунку 3.31.

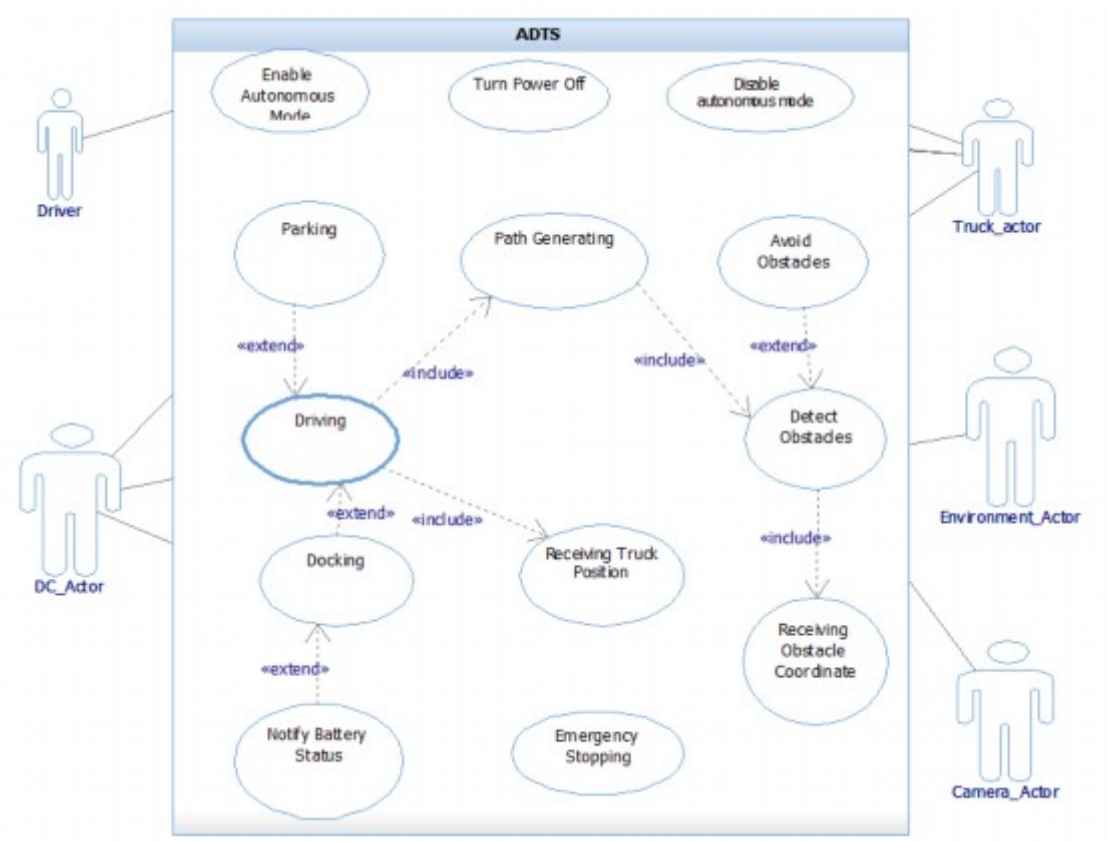


Рис. 3.30. Приклад діаграми варіантів використання

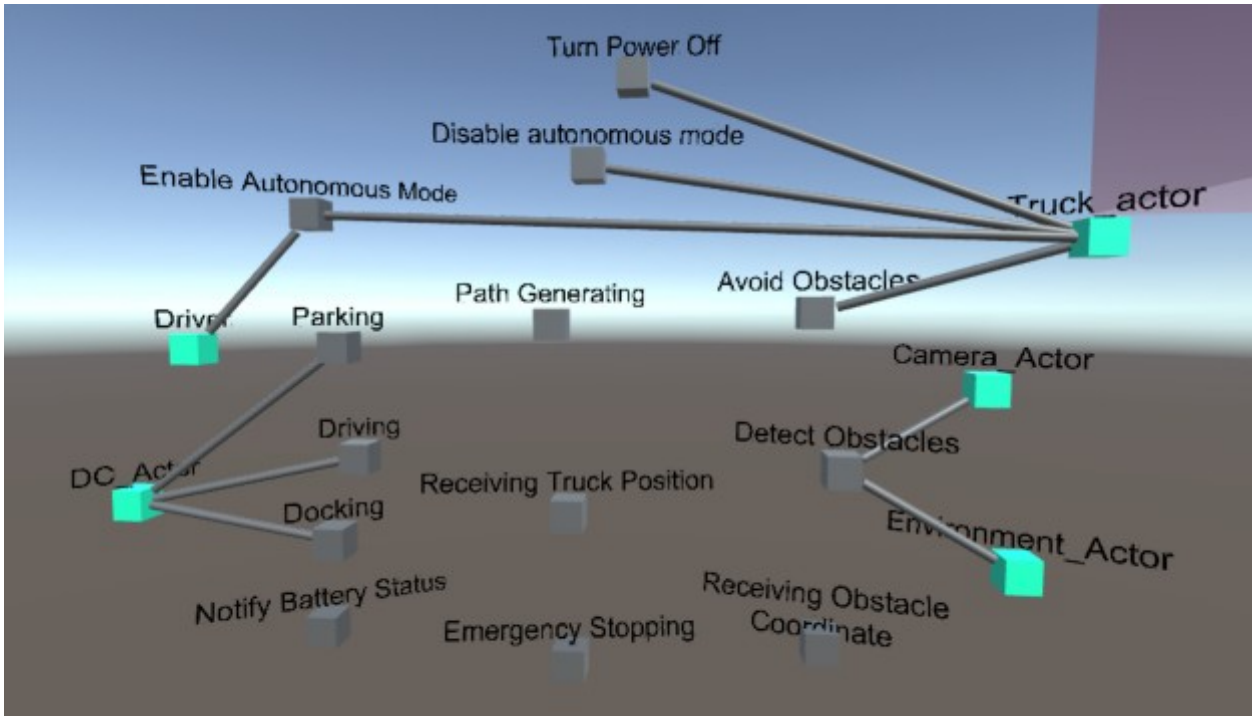


Рис. 3.31. Перетворена діаграма варіантів використання в Unity

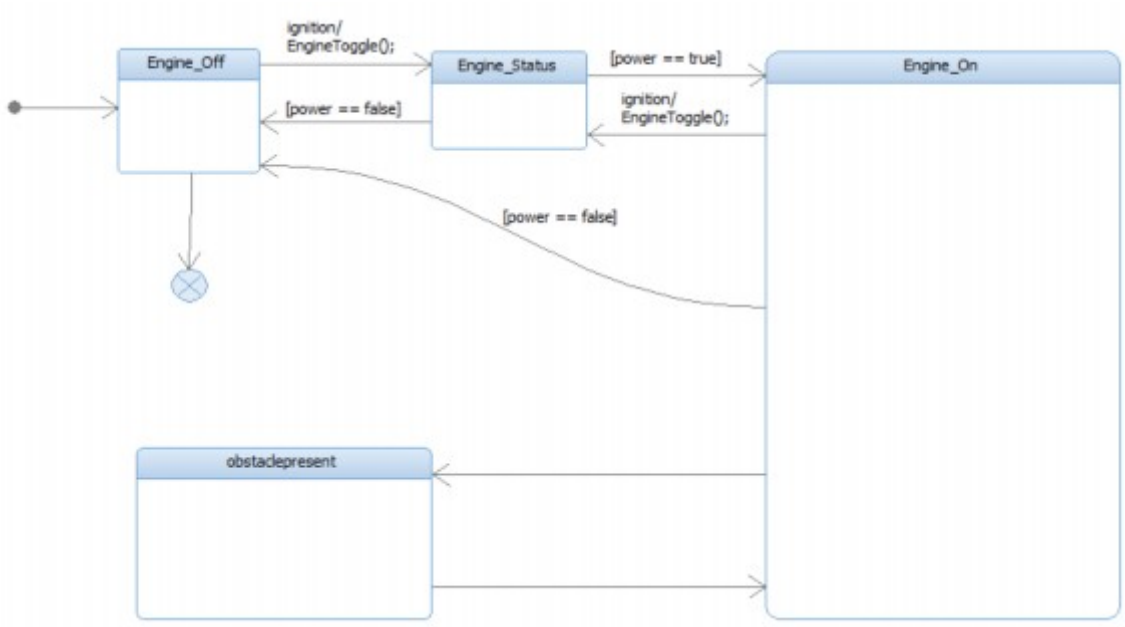


Рис. 3.32. Приклад діаграми кінцевого автомата

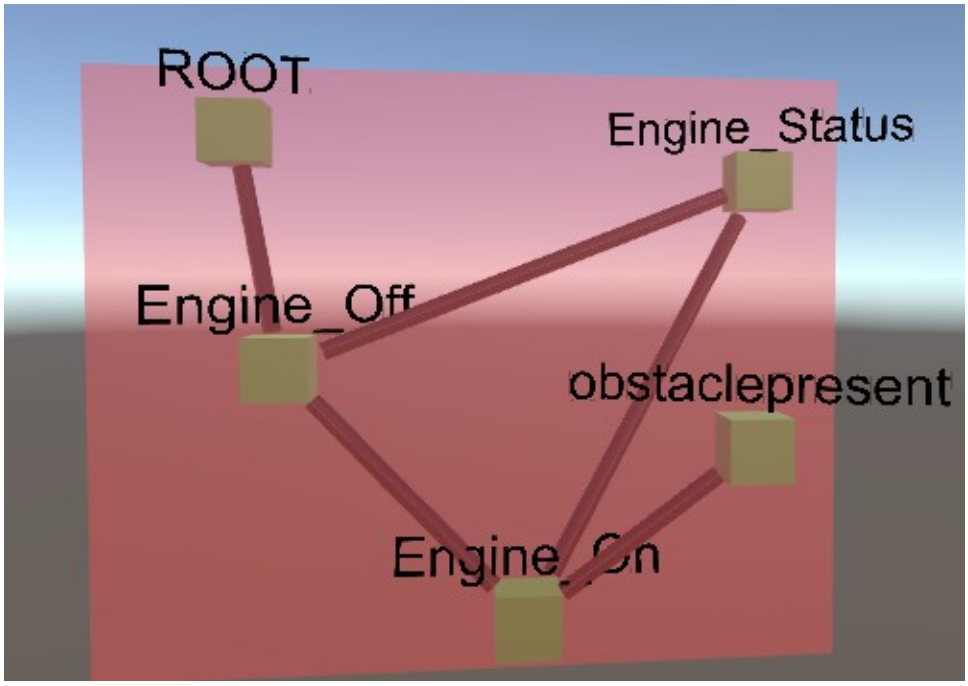


Рис. 3.33. Перетворена схема кінцевого автомата в Unity

Висновки до розділу

У третьому розділі розроблено методологію визначення шару конверсії між рівнем моделей та рівнем імплементації, що є ключовим аспектом інтеграції моделей у системи віртуальної реальності на базі Unity. Здійснено

побудову архітектури системи для перетворення моделі, яка визначає принципи взаємодії між моделями, що створюються у IBM Rhapsody, та середовищем Unity. Це забезпечує ефективний перехід від моделювання до реальної імплементації.

На етапі видобування даних описано процеси, які забезпечують коректний збір та обробку даних, необхідних для подальшого перетворення. Описано методи передачі даних між Data Extractor та Unity, а також розглянуто активні та пасивні взаємозв’язки між модулями, що сприяє гнучкій взаємодії системних компонентів.

Проаналізовано процеси обробки та абстрагування даних, що дозволяють ефективно використовувати структуровану інформацію для візуалізації моделей. Представлено структури даних та діаграми класів, що забезпечують чітке уявлення про організацію даних у системі.

Розглянуто реалізацію структури шару конверсії моделей між IBM Rhapsody та Unity, що охоплює не лише передачу даних, але й їхню подальшу візуалізацію. Зокрема, описано процес візуалізації діаграм вимог та 3D візуалізації варіантів використання, що забезпечує інтуїтивне уявлення про функціонування системи та її компонентів.

Таким чином, розроблена методологія дозволяє інтегрувати моделі у середовище Unity з високим ступенем точності, що сприяє більш ефективному процесу розробки та впровадження програмних рішень на основі моделювання.

ВИСНОВКИ

У ході магістерської роботи було проведено комплексне дослідження методології перетворення моделей, що базується на інтеграції модельно-орієнтованої системної інженерії (MDE) та засобів 3D-візуалізації. Дослідження включає аналіз інструментів моделювання, особливості процесу моделювання на основі моделей, а також розробку підходів для побудови ефективного зв’язку між рівнем моделей та рівнем імплементації.

У першому розділі здійснено аналіз предметної області моделювання систем на основі перетворення моделей. Описано основні поняття MDE та особливості модельно-орієнтованої системної інженерії, що є основою для проектування складних систем. Представлено огляд ключових інструментів моделювання, таких як IBM Rational Rhapsody, що використовується для проектування системних моделей, та Unity, яке забезпечує 3D-візуалізацію моделей, створюючи основу для реалізації інтуїтивно зрозумілих та візуально привабливих інтерфейсів.

На основі проведеного аналізу можна зробити висновок, що побудова шару конверсії між рівнем моделей та рівнем імплементації потребує комплексного підходу, який включає використання адаптивних методів моделювання, вибір відповідних мов моделювання та інструментів, а також реалізацію ефективної комунікації між моделями та їхньою реалізацією. Це дозволяє забезпечити цілісність системи, точність реалізації та високу якість кінцевого продукту.

Другий розділ присвячено дослідженню методів та алгоритмів, що забезпечують побудову шару конверсії між рівнем моделей та рівнем імплементації. Визначено фактори складності при використанні модельно-орієнтованої системної інженерії, такі як незрозумілість моделей та потреба в абстрагуванні. Досліджено можливості використання 3D-середовища для вирішення цих проблем. Проведено порівняння різних підходів до системного моделювання, зокрема SysML та UML, що дозволило сформувати

оптимальні принципи проектування для побудови структурованих діаграм та моделей.

У третьому розділі розроблено методологію визначення шару конверсії між рівнем моделей та імплементації. Побудовано архітектуру системи для перетворення моделі та описано процеси видобування, передачі та обробки даних, що є ключовими для забезпечення інтеграції між IBM Rhapsody та Unity. Розглянуто активний та пасивний взаємозв'язок між модулями, що дозволяє досягти гнучкої інтеграції. Детально описано структури даних та їх візуалізацію, що дозволяє забезпечити наочне уявлення про моделі у 3D-середовищі.

Таким чином, результати роботи дозволяють вирішити актуальні проблеми інтеграції модельно-орієнтованої системної інженерії з візуалізацією у середовищі Unity, що сприяє підвищенню точності розробки та зниженню ризиків на етапах впровадження. Розроблена методологія може бути використана як основа для подальших досліджень та впроваджень у сфері інтеграції моделей та візуалізації у різних галузях інженерії.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Unity documentation. https://docs.unity3d.com/Manual/index.html.
2. G. F. C Ware. Viewing a graph in virtual reality display is three times as good as a 2d diagram, 1994.
3. Chi-Wen Yang, Tsung-HanLee, Chien-Lung Huang, Kuei-Shu Hsu. Unity 3d production and environmental perception vehicle simulation platform, 2016.
4. T. K. Colin Atkinson. Model-driven development: A metamodeling foundation.
5. L. Delligatti. SysML Distilled, A Brief Guide to the Systems Modeling Language. Addison-Wesley Professional, 2013. ISBN 9780321927866.
6. INCOSE. INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities. Wiley, 2015. ISBN 9781118999400.
7. S. P. Jon Holt. SysML for systems engineering : a model based approach. London : Institution of Engineering and Technology, 2013. ISBN 9781849196512.
8. J. J. Kuang Yang. The designing of training simulation system based on unity 3d, 2011. URL https://ieeexplore.ieee.org/abstract/document/5750762.
9. J. V. Matej Ferenc, Ivan Polasek. Collaborative modeling and visualization of software systems using multidimensional uml, 2017.
10. M. R. N Md Jubair basha, Salman Abdul Moiz. Model based software develeopment: Issues challenges, 2012. URL https://arxiv.org/ abs/1203.1314.
11. Object Management Group. OMG Systems Modeling LanguageTM, 2017. URL https: //www.omg.org/spec/SysML/1.5/PDF.
12. C. W. Pourang Irani. Diagramming information structures using 3d perceptual primitives, 2003.

13. T. F. Stephen J. Mellor, Anothony N. Clark. Model-driven development: guest editors' introduction., 2003. URL http://eprints.mdx.ac.uk/6083/1/s5014.pdf.
14. Atkinson, C., & Kühne, T. (2003). Model-driven development: A metamodeling foundation. IEEE Software.
15. Mellor, S. J., Clark, A. N., & Futagami, T. (2003). Model-driven development: guest editors' introduction. IEEE Software.
16. Bézivin, J., & Gerbé, O. (2001). Towards a precise definition of the OMG/MDA framework. Proceedings of the 16th IEEE international conference on Automated software engineering.
17. Schmidt, D. C. (2006). Model-driven engineering. IEEE Computer.
18. France, R. B., & Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. Future of Software Engineering.
19. Stahl, T., & Völter, M. (2006). Model-driven software development: technology, engineering, management. John Wiley & Sons.
20. Favre, J. M. (2004). Foundations of model (driven) (reverse) engineering: Models—episode I: Stories of the fidus papyrus and of the solarus. Lecture Notes in Computer Science.
21. Bézivin, J., & Jouault, F. (2005). Model transformation in practice. Proceedings of the OOPSLA Workshop on Model-Driven Development (MDD).
22. OMG. (2003). Model Driven Architecture (MDA) Guide. Object Management Group.
23. Kleppe, A. G., Warmer, J., & Bast, W. (2003). MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley.
24. Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., & Völkel, S. (2009). Design guidelines for domain specific languages. Proceedings of the 9th OOPSLA workshop on domain-specific modeling.
25. Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. IBM Systems Journal.

26. Selic, B. (2003). The pragmatics of model-driven development. IEEE Software.
27. Brown, A. W. (2004). Model Driven Architecture: Principles and Practice. Software and Systems Modeling.
28. Broy, M., & Stølen, K. (2001). Specification and development of interactive systems: focus on streams, interfaces, and refinement. Springer.
29. Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: An annotated bibliography. ACM SIGPLAN Notices.
30. Hutchinson, J., Whittle, J., Rouncefield, M., & Kristoffersen, S. (2011). Empirical assessment of MDE in industry. Proceedings of the 33rd International Conference on Software Engineering.
31. Greenfield, J., Short, K., Cook, S., & Kent, S. (2004). Software factories: Assembling applications with patterns, models, frameworks, and tools. John Wiley & Sons.
32. Giese, H., & Wagner, R. (2006). From model transformation to incremental bidirectional model synchronization. Software & Systems Modeling.
33. Mens, T., & Gorp, P. V. (2006). A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science.
34. Seidewitz, E. (2003). What models mean. IEEE Software.
35. Mohagheghi, P., Dehlen, V., & Neple, T. (2009). Model-driven architecture for real-time systems: from models to code. Real-Time Systems.
36. Thomas, D. (2004). MDA: Revenge of the modelers or UML utopia?. IEEE Software.
37. Kent, S. (2002). Model Driven Engineering. Proceedings of the Third International Conference on Integrated Formal Methods.
38. Uhl, A., & Guttman, M. (2011). Model-driven architecture: Foundations and applications. Springer.
39. Tisi, M., Jouault, F., Cabot, J., & Fraternali, P. (2012). Advanced model transformation with ATL. Springer.

40. Rensink, A., & Warmer, J. (2008). Model transformation with QVT. Lecture Notes in Computer Science.
41. Cabot, J., Clarisó, R., & Riera, D. (2014). Verification and validation of UML/OCL models. Lecture Notes in Computer Science.
42. Guerra, E., de Lara, J., Kolovos, D. S., Paige, R. F., & dos Santos, O. M. (2013). A comparative analysis of language support for model management. Science of Computer Programming.
43. Bézivin, J., & Vignollet, L. (2005). Model engineering for interoperability. Lecture Notes in Computer Science.
44. Liu, D., & Özsu, M. T. (2009). Model-driven development in database design. IEEE Data Engineering Bulletin.
45. Clarke, P. J., & Carroll, J. L. (2013). Towards a methodology for model-driven development of adaptive systems. Journal of Systems and Software.
46. Blouin, A., Bourdeau, F., & Barais, O. (2014). Combining model-driven engineering and rule-based programming. Journal of Object Technology.
47. Jackson, M. (2012). Software Requirements & Specifications: A Lexicon of Practice, Principles, and Prejudices. Addison-Wesley.
48. Morin, B., Barais, O., & Fleurey, F. (2010). Models@run.time: a guided tour of the state of the art and research challenges. Lecture Notes in Computer Science.
49. Kelsen, P., & Ma, Q. (2009). A model transformation approach for model-based design. Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems.
50. Langer, P., Wimmer, M., & Kappel, G. (2013). Model versioning and model differencing. Proceedings of the 9th International Workshop on Model-based Methodologies for Pervasive and Embedded Software.
51. Kolovos, D. S., Rose, L. M., & Paige, R. F. (2008). A lightweight approach for model management. Lecture Notes in Computer Science.

52. Poizat, P., & Royer, J. C. (2012). Contract-based integration of web services into software architecture models. Journal of Software and Systems Modeling.
53. Heckel, R., & Thöne, S. (2008). Towards automated model evolution. Lecture Notes in Computer Science.