БАКАЛАВРСЬКА РОБОТА

БР. ІП - 16.00.00.000 ПЗ

Група ІП-21-4

Блиха Давид

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Блиха Давид Олегович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Методологія інтерпретації програмних парадигм на основі засобів UML

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

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

Здобувач освітнього рівня Блиха Давид Олегович

(підпис, ініціали та прізвище здобувача)

Науковий керівник Шекета Василь Іванович, д.т.н., професор

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

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

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

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Блисі Давиду Олеговичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Методологія інтерпретації програмних парадигм на основі засобів UML”

керівник проекту (роботи) Шекета Василь Іванович, професор

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

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

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

1. Аналіз предметної області опису програмних парадигм за допомогою UML
2. Представлення модельних трансформацій програмних парадигм на основі засобів UML
3. Трансформація моделей компонентів та комунікації у представлення засобами UML
4. Представлення методології інтерпретації програмних парадигм на основі засобів UML
5. Трансформація поведінкових аспектів та потоків взаємодії програмних парадигм

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

1. Діаграма станів компоненти Filter (рис. 1.1)
2. Діаграма станів компоненти Buffer (рис. 1.2)
3. Лінійна архітектура конвеєра для моделі Paradigm (рис. 1.7)
4. Комунікації від деталізованої STD до співпраці (рис. 1.8)
5. Діаграма послідовності комунікацій фільтра (рис. 2.1)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області опису програмних парадигм за допомогою UML	04.05.2025	виконано
2	Представлення модельних трансформацій програмних парадигм на основі засобів UML	15.05.2025	виконано
3	Трансформація моделей компонентів та комунікації у представлення засобами UML	25.05.2025	виконано
4	Представлення методології інтерпретації програмних парадигм на основі засобів UML	01.06.2025	виконано
5	Трансформація поведінкових аспектів та потоків взаємодії програмних парадигм	05.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 79 сторінок, 42 рисунків, список використаних джерел із 38 найменуваннями.

Метою даної дипломної роботи є розробка та обґрунтування методології інтерпретації програмних парадигм на основі засобів UML, що забезпечує коректну трансформацію структурних і поведінкових аспектів моделей парадигми координації.

Об'єкт дослідження - процес моделювання програмних парадигм засобами формальних та візуальних мов моделювання.

Предмет дослідження - методика трансформації моделей координації, представлених мовою Paradigm, у діаграми UML.

В першому розділі проаналізовано особливості Paradigm як формальної мови для моделювання координації та побудовано специфікації ключових компонентів систем.

В другому розділі наведено детальну методику трансформації моделей Paradigm у різні типи UML-діаграм, зокрема діаграми станів і компонентів.

В третьому розділі представлено повну методологію інтерпретації структурних і поведінкових аспектів програмних парадигм у вигляді UML-діаграм, що дає змогу забезпечити цілісне уявлення про функціонування системи

Висновок: запропоновано комплексний підхід до трансформації як структурних, так і поведінкових аспектів програмних парадигм у UML-діаграми (станів, компонентів, класів, пакетів, розгортання, часу та варіантів використання).

КЛЮЧОВІ СЛОВА: UML, ПРОГРАМНІ ПАРАДИГМИ, МОДЕЛЮВАННЯ, КООРДИНАЦІЯ, ДІАГРАМИ СТАНІВ, КОМПОНЕНТІВ, ТРАНСФОРМАЦІЯ МОДЕЛЕЙ, МЕТОДОЛОГІЯ ІНТЕРПРЕТАЦІЇ.

ANNOTATION

The bachelor's thesis contains 79 pages, 42 figures, a list of used sources with 38 names.

The purpose of this thesis is to develop and substantiate a methodology for interpreting software paradigms based on UML tools, which ensures the correct transformation of structural and behavioral aspects of coordination paradigm models.

The object of research is the process of modeling software paradigms using formal and visual modeling languages.

The subject of research is a methodology for transforming coordination models, represented in the Paradigm language, into UML diagrams.

The first section analyzes the features of Paradigm as a formal language for coordination modeling and builds specifications for key system components.

The second section provides a detailed methodology for transforming Paradigm models into various types of UML diagrams, in particular state and component diagrams.

The third section presents a complete methodology for interpreting structural and behavioral aspects of software paradigms in the form of UML diagrams, which allows providing a holistic view of the functioning of the system.

Conclusion: a comprehensive approach to transforming both structural and behavioral aspects of software paradigms into UML diagrams (states, components, classes, packages, deployment, timing, and use cases) is proposed.

KEYWORDS: UML, SOFTWARE PARADIGMS, MODELING, COORDINATION, STATE MACHINE DIAGRAMS, COMPONENT DIAGRAMS, MODEL TRANSFORMATION, INTERPRETATION METHODOLOGY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

UML - Unified Modeling Language (Уніфікована мова моделювання)

STD - State Diagram(s) (Діаграми станів) - основний тип діаграм у Paradigm.

ST - Set of States (Набір станів) - використовується у формальному визначенні STD.

AC - Set of Actions (Набір дій) - використовується у формальному визначенні STD.

TR - Set of Transitions (Набір переходів) - використовується у формальному визначенні STD

ВСТУП

У сучасному програмному забезпеченні стрімко зростає складність систем, що вимагає чітких і формалізованих підходів до моделювання архітектури та поведінки компонентів. Різноманітність програмних парадигм, таких як об’єктно-орієнтована, функціональна, процедурна чи парадигма координації, створює потребу в універсальних засобах для їх опису, порівняння та інтеграції. Мова моделювання UML (Unified Modeling Language) є галузевим стандартом для візуального представлення структури та поведінки програмних систем. Однак її застосування як інструмента інтерпретації між різними парадигмами потребує чітко визначеної методології. Актуальність даної роботи полягає у створенні формального підходу до інтерпретації та трансформації концептів програмних парадигм у засоби UML, що сприятиме покращенню якості проектування, уніфікації моделей та підвищенню сумісності між різними мовами опису систем.

Сучасна розробка програмного забезпечення характеризується широким використанням різноманітних підходів, які ґрунтуються на різних програмних парадигмах. Кожна з парадигм — об'єктно-орієнтована, процедурна, функціональна, логічна, парадигма координації тощо — формує власне бачення структури програм, способів управління даними, взаємодії між компонентами та механізмів синхронізації. Зі збільшенням складності програмних систем виникає потреба у гнучких і уніфікованих засобах, які дозволяють моделювати системи незалежно від обраної парадигми або забезпечувати перехід між ними.

Одним із таких засобів є UML (Unified Modeling Language) — стандартна мова візуального моделювання, яка дозволяє описувати структурні й поведінкові аспекти програмного забезпечення. UML широко застосовується в індустрії для проєктування, аналізу та документування програмних систем. Проте, на практиці UML використовується переважно у

контексті об’єктно-орієнтованої парадигми, що не завжди дозволяє коректно представити інші типи моделей або виконати інтеграцію парадигм.

Особливої уваги потребує парадигма координації — зокрема, формальна мова Paradigm, яка забезпечує опис взаємодії компонентів, процесів синхронізації та узгодження в складних системах, таких як конвеєрні архітектури або розподілені обчислення. Paradigm дозволяє моделювати компоненти типу Filter і Buffer, які відіграють ключову роль у таких системах, однак її формалізм не завжди інтуїтивно зрозумілий для розробників, які працюють із UML.

Актуальність роботи

Зростання складності інформаційних систем вимагає ефективних засобів для опису як внутрішніх механізмів їхньої роботи, так і взаємодії компонентів. Paradigm надає потужний формалізм для координації поведінки систем, однак його практична застосовність у розробці обмежена складністю сприйняття та недостатньою інтеграцією з візуальними стандартами. UML, натомість, забезпечує інтуїтивне середовище моделювання, однак не охоплює специфіку поведінкової координації на тому ж рівні формальності. Саме тому актуальним є дослідження методів трансформації моделей Paradigm у діаграми UML, що дозволить поєднати точність формалізму з наочністю і поширеністю візуального опису, зробивши моделі більш зрозумілими та доступними для практичного використання в інженерії програмного забезпечення.

У зв’язку з цим виникає потреба в методології, яка дозволяє здійснювати трансформацію моделей, побудованих у рамках Paradigm, у еквівалентні представлення на основі UML. Такий підхід відкриває можливості для візуалізації, перевірки узгодженості, автоматизації процесів проєктування та формування документації у зрозумілому й стандартизованому вигляді.

Метою даної дипломної роботи є розробка та обґрунтування методології інтерпретації програмних парадигм на основі засобів UML, що забезпечує коректну трансформацію структурних і поведінкових аспектів моделей парадигми координації.

Для досягнення цієї мети було виконано **наступні задача**:

- аналіз предметної області та концепцій мови Paradigm;

- специфікацію типових компонентів систем;

- формалізацію правил трансформації моделей у відповідні UML-діаграми;

- опис процесу побудови UML-моделей для різних аспектів (станів, класів, компонентів, розгортання тощо).

Об’єкт дослідження - процес моделювання програмних парадигм засобами формальних та візуальних мов моделювання.

Предмет дослідження - методика трансформації моделей координації, представлених мовою Paradigm, у діаграми UML.

Методи дослідження:

- Формальний аналіз структур моделей мовою Paradigm;

- Методологія трансформацій моделей;

- Засоби візуального моделювання UML;

- Порівняльний аналіз моделей;

- Прикладне моделювання компонентів і комунікацій.

Наукова новизна

Запропоновано системну методику трансформації моделей Paradigm у UML, що поєднує формальність опису координаційних аспектів із широким спектром візуального представлення структур та поведінки програмних систем.

Практичне застосування

Результати роботи можуть бути використані при проєктуванні складних програмних систем, у процесі верифікації поведінкових моделей, у

навчальному процесі з моделювання та розробки ПЗ, а також як інструмент інтеграції формального й візуального підходів до опису систем.

Бакалаврська робота містить 79 сторінок, 42 рисунки, 3 розділи список використаних джерел із 38 найменуваннями.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ОПИСУ ПРОГРАМНИХ ПАРАДИГМ ЗА ДОПОМОГОЮ UML

1.1. Задача і мета дипломної роботи

Дана дипломна робота присвячена дослідженню можливостей представлення специфікацій програмних систем за допомогою різних мов моделювання. У роботі розглянуто специфікацію типового архітектурного стилю "конвеєр" (pipeline), змодельованого засобами мови координації Paradigm. Обраний приклад лінійного конвеєра, що складається з взаємодіючих компонентів фільтра та буфера, які функціонують за патерном "виробник-споживач", слугує для ілюстрації та практичної демонстрації підходів до моделювання та трансформації моделей.

Основною задачею даної дипломної роботи є дослідження та практична реалізація трансформації специфікацій конвеєра, представлених за допомогою діаграм мови Paradigm, у відповідний набір діаграм уніфікованої мови моделювання (Unified Modeling Language - UML). Вибір UML як цільової мови представлення обґрунтовується тим, що UML, маючи значно багатші виразні засоби, ширший спектр представлення різних аспектів системи (структурних, поведінкових) та будучи визнаним стандартом моделювання, може забезпечити більш повне та деталізоване представлення систем, початково описаних у Paradigm.

В рамках даної роботи ставиться завдання продемонструвати можливість та визначити особливості перекладу моделей, розроблених у Paradigm на прикладі конвеєра, у весь визначений набір з тринадцяти типів діаграм UML, а також проаналізувати отримані результати з точки зору адекватності та повноти представлення моделі конвеєра засобами UML. Практична частина роботи покаже, як саме можна здійснити таку трансформацію та які аспекти моделі при цьому відображаються.

1.2. Взаємодія мов моделювання Paradigm та UML у специфікації систем

Мова моделювання Paradigm є специфічним формалізмом, призначеним для моделювання та аналізу координації між компонентами складних розподілених або багатоагентних систем. В її основі лежить використання діаграм станів (State Diagrams - STD) як фундаментального елемента для опису поведінки компонентів та їхньої взаємодії. Paradigm реалізує два основні типи комунікаційних механізмів: вертикальну та горизонтальну комунікацію. Вертикальна комунікація забезпечує зв'язок та обмін інформацією між компонентом та його зовнішнім інтерфейсом, представленим у вигляді рольового порту, а також між цим портом та відповідним йому дзеркальним ролем іншого компонента. Горизонтальна комунікація, у свою чергу, моделює безпосередню взаємодію між самими компонентами або між їхніми дзеркальними ролями, відображаючи потоки даних чи керування на рівні архітектури системи.

Однією з ключових особливостей моделей, створених у Paradigm, є їхня здатність до динамічної адаптації та реконфігурації під час виконання. Ця функціональність, реалізована, зокрема, за допомогою спеціального компонента McPal, дозволяє системі еволюціонувати від її початкового стану (модель AsIs) до нового, адаптованого стану (модель ToBe). Динамічна адаптація включає можливості додавання нових компонентів, модифікації існуючих або зміни зв'язків між ними, що робить Paradigm придатним для моделювання систем з високими вимогами до гнучкості та автономності.

Концептуальний апарат Paradigm базується на п'яти основних поняттях: Діаграма станів (STD), Фаза, Пастка (Trap), Роль STD та Правило узгодженості. Хоча більшість цих понять (STD, Фаза, Пастка, Роль STD) мають безпосереднє візуальне представлення на основі STD, Правило узгодженості є винятком, не маючи власної явної графічної нотації у рамках

Paradigm. Ця обставина може створювати певні складнощі для повноцінної візуалізації та розуміння логіки підтримання цілісності координації програмної системи.

Варто зазначити, що існують вже наявні точки дотику та можливості інтеграції між Paradigm та іншими мовами моделювання, зокрема Unified Modeling Language (UML). Наприклад, для більш детального опису статичної структури системи, змодельованої у Paradigm, можуть використовуватися діаграми компонентної структури UML, що дозволяє уточнити зв'язки та залежності між компонентами на більш низькому рівні абстракції. Крім того, для наочного представлення послідовності виконання та логіки Правил узгодженості Paradigm ефективно застосовуються діаграми активності UML. Це свідчить про потенційну сумісність та доцільність використання UML для доповнення та уточнення моделей Paradigm.

Систематична візуалізація всіх ключових понять Paradigm за допомогою відповідних діаграм UML представляє значний інтерес для подальшого розвитку підходів до моделювання координації. Оскільки UML є універсальною та найбільш поширеною мовою моделювання у світі, представлення концепцій Paradigm через призму його діаграм дозволить значно підвищити зрозумілість моделей Paradigm для широкого кола фахівців, полегшить використання стандартних інструментів моделювання та аналізу, а також сприятиме обміну моделями.

Крім того, багатство виразних можливостей UML, що охоплюють різні аспекти системи (структуру, поведінку, взаємодію, розгортання), дозволяє представити моделі Paradigm більш всебічно, враховуючи деталі, які можуть бути неочевидними або непредставленими при використанні лише STD. Такий підхід може не тільки спростити розуміння Paradigm, але й відкрити нові можливості для аналізу, верифікації та потенційного покращення самих моделей координації за рахунок використання потужного аналітичного апарату, що асоціюється з UML.

1.3. Основні концепції та формальні визначення мови моделювання координації Paradigm

Архітектура системи в рамках формалізму Paradigm організована відповідно до певних аспектів або напрямків співпраці, які називаються розділами (π). Розділ представляє собою логічно визначений набір підповедінок (сукупність локальних поведінкових фрагментів) компонента, що декомпозує його повну поведінку. Він визначає послідовність фаз (S), через які проходить компонент, беручи участь у конкретній кооперації або взаємодії.

На більш високому рівні архітектури, компонент взаємодіє з іншими компонентами через свою роль (Z(π)). Роль є абстрактним представленням набору фаз компонента в контексті певного розділу, фокусуючись на точках взаємодії між фазами.

Оскільки зміни стану всередині конкретної фази є суто локальними для компонента, ключовим концептуальним розмежуванням у Paradigm є відмінність між переходом фази (S~S′) та переходом стану (xax′). Перехід стану відбувається між станами в межах однієї і тієї ж фази, відображаючи внутрішню локальну динаміку компонента. Натомість, перехід фази являє собою зміну між різними фазами (які визначаються як набори станів з пов'язаними пастками), забезпечуючи механізм для моделювання глобальних змін у координації між компонентами. Цей підхід дозволяє одночасно і незалежно моделювати як локальні зміни поведінки в рамках окремих компонентів, так і узгоджені глобальні зміни, що відбуваються по всій архітектурі системи [1].

Формальні визначення ключових понять мови моделювання Paradigm, представлені нижче у структурованому вигляді для підтримки вищезазначених концептуальних положень:

Формальні визначення понять Paradigm:

- Діаграма станів (State Diagram - STD):

STD Z визначається як трійка $Z=\langle ST,AC,TR\rangle$, де: ST — скінченний набір станів, що містить один спеціальний стартовий стан. AC — скінченний набір дій.

$TR \subseteq ST \times AC \times ST$ — набір переходів. Позначення xax' означає, що існує перехід $(x,a,x') \in TR$ з стану x у стан x' за дії a.

- Фаза (Phase):

Фаза $S=\langle st,ac,tr\rangle$ STD $Z=\langle ST,AC,TR\rangle$ є сутностю, яка сама є STD, такою що: $st \subseteq ST$ — підмножина станів з Z. $ac \subseteq AC$ — підмножина дій з Z. $tr \subseteq \{(x,a,x') \in TR \mid x,x' \in st, a \in ac\}$ — підмножина переходів з Z, що з'єднують стани лише всередині st з діями з ac.

- Пастка (Trap):

Пастка t фази $S=\langle st,ac,tr\rangle$ STD Z — це непорожня підмножина станів $t \subseteq st$ даної фази. Якщо t=st, пастка називається тривіальною.

- Зв'язок на основі пасток / Перехід фази (Trap Connection / Phase Transition):

Пастка t фази S STD Z з'єднує фазу S з фазою $S'=\langle st',ac',tr'\rangle$ STD Z, якщо множина станів пастки t є підмножиною множини станів фази S', тобто $t \subseteq st'$. Такий зв'язок між двома фазами S та S' на основі пастки t називається переходом фази та позначається як $S \sim_t S'$.

- Розділ (Section):

Розділ $\pi=\{(Si,Ti) \mid i \in I\}$ STD $Z=\langle ST,AC,TR\rangle$, де I — непорожній індексний набір, є набором пар (Si,Ti). Кожна пара складається з фази $Si=\langle sti,aci,tri\rangle$ STD Z та набору Ti пасток фази Si.

- Роль STD (Role STD):

Роль $Z(\pi)$ на рівні розділу $\pi=\{(Si,Ti) \mid i \in I\}$ STD $Z=\langle ST,AC,TR\rangle$ є STD $Z(\pi)=\langle ST,AC,TR\rangle$, де: $ST \subseteq \{Si \mid i \in I\}$ — набір станів, що є підмножиною фаз

з розділу. AC⊆∪i∈ITi — набір дій, що є підмножиною об'єднання всіх пасток з розділу. TR⊆{Si~,tSj ∣ i,j∈I,t∈AC} — набір переходів фази. STD Z називається деталізованою STD, що лежить в основі глобальної STD Z(π), яка є її роллю на рівні розділу π.

- Правило узгодженості (Consistency Rule):

Правило узгодженості ρ для набору ролей Z1(π1),…,Zk(πk) — це механізм синхронізації переходів, що беруть участь у правилі, головним чином переходів фази з ролей в певному ансамблі. Таке правило узгодженості ρ позначається як рядок, що починається символом "*", за яким слідує непорожній список переходів фази, взятих з різних ролей ансамблю. Рядок може бути опціонально префіксований одним переходом стану з недеталізованої (нерольової) STD Z. У присутності переходу стану з недеталізованої STD Z може бути включена так звана змінна клауза Z:[y:=expr], яка дозволяє перезаписати значення змінної y, доступної для STD Z, значенням виразу expr відповідного типу. Якщо змінна клауза включена, список переходів фази може бути порожнім. STD Zk, що зустрічається у списку переходів фази, називається учасником ρ; якщо перехід недеталізованої STD Z зустрічається у ρ, Z називається диригентом ρ. Правило узгодженості з диригентом також називається кроком оркестрації; правило узгодженості без диригента називається кроком хореографії.

- Модель Paradigm (Paradigm Model):

Модель Paradigm — це набір STD, відповідних їм ролей та правил узгодженості.

- Протокол (Protocol):

Підмножина P правил узгодженості з моделі Paradigm називається протоколом P, якщо для будь-якої ролі Zi(πi), яка зустрічається у правилі з P, ця роль Zi(πi) не зустрічається у жодному правилі узгодженості, що не належить до P. Будь-яке правило узгодженості ρ∈P називається кроком протоколу P. Протокол P називається хореографією, якщо всі правила

узгодженості в P є кроками хореографії. Протокол, який не є хореографією, називається оркестрацією. Диригент кроку оркестрації в оркестрації P також називається диригентом P.

1.4. Специфікація компонентів Filter та Buffer мовою моделювання

У даному підрозділі представлена детальна специфікація двох ключових компонентів лінійного конвеєра — фільтра та буфера — з використанням формалізму мови моделювання координації Paradigm. Наведений приклад базується на модифікованому варіанті моделі і слугує для ілюстрації практичного застосування концепцій Paradigm до конкретних архітектурних елементів.

1.4.1. Специфікація компонента Filter

Компонент фільтра, позначений як $Filter_i$, моделюється у вигляді Діаграми станів (STD), представленої на рисунку 1.1.

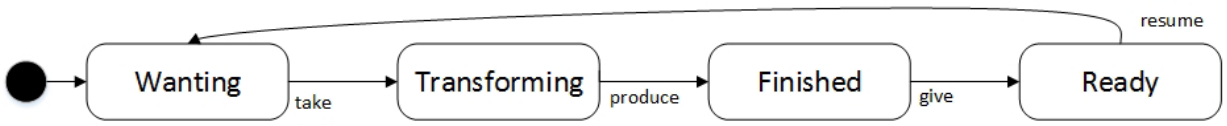


Рисунок 1.1 – Діаграма станів компоненти $Filter_i$

STD Filteri складається з чотирьох станів (STFilter ={Wanting,Transforming,Finished,Ready}) та чотирьох дій (ACFilter ={take,produce,give,resume}). Функціонально фільтр реалізує процес обробки елементів, що включає їх отримання (споживання) та видачу (виробництво), здійснюючи один повний цикл трансформації елемента. Ця діяльність координується через його ролі: роль Prod відповідає за аспекти, пов'язані з

"виробництвом" та видачею обробленого елемента, а роль Cons — за аспекти, пов'язані зі "споживанням" та отриманням вхідного елемента.

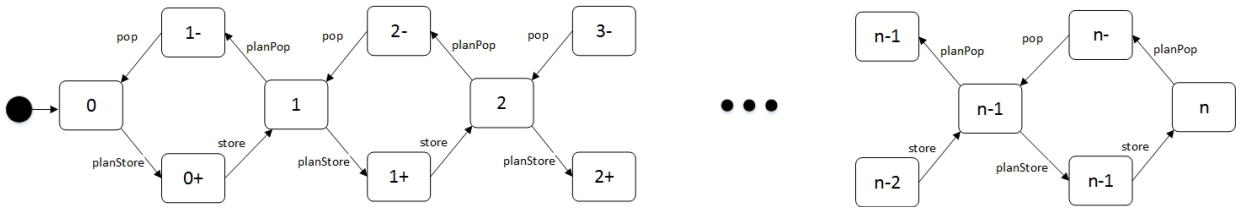
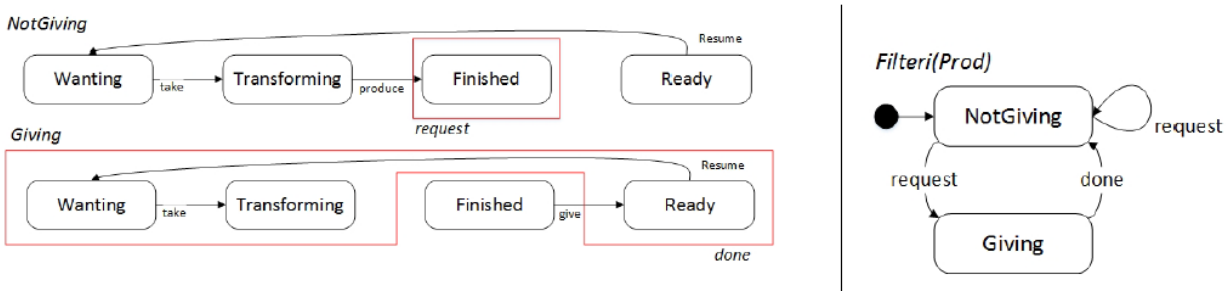


Рисунок 1.2 – Діаграма станів компоненти $Buffer_i$

Поведінка компонента Filteri структурована за розділами, що відповідають його ролям. Розділ та рольова поведінка для ролі Prod візуалізовані на рисунку 1.3.



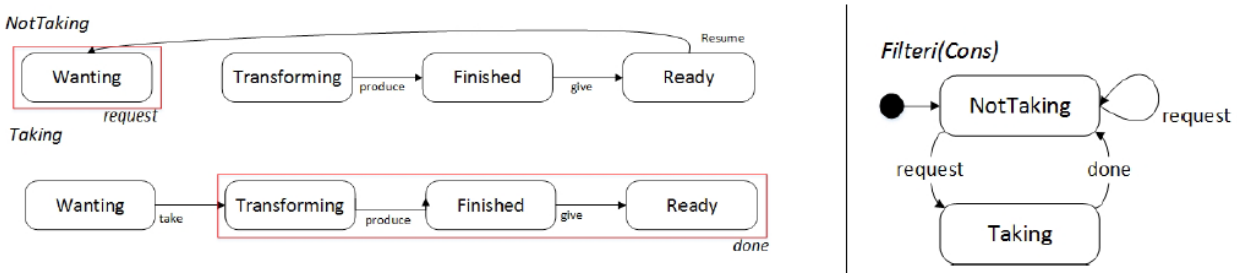
а) б)

Рисунок 1.3 – Розділ а) і роль б) для компоненти $Filter_i$ (Prod)

Рисунок 1.3 а) ілюструє розділ Prod, що включає дві фази: NotGiving та Giving. Кожна фаза визначає набір станів компонента Filteri та асоційовані з ними пастки. У фазі NotGiving стан Finished включено до пастки request. Перебування у пастці означає, що компонент у цьому стані сигналізує про готовність до взаємодії з оточенням, необхідну для продовження локальної поведінки або переходу до іншої фази/стану. У даному випадку, пастка request у фазі NotGiving відображає готовність фільтра передати (віддати) оброблений елемент. Фаза Giving включає стани Wanting,Transforming,Ready

у пастці done, що вказує на завершення циклу "віддачі" елемента та готовність до подальших операцій. Відповідна рольова поведінка Filter_i (Prod), візуалізована на рисунку 1.3 б), відображає абстракцію поведінки компонента як послідовності переходів між цими фазами. Перехід фази відбувається від фази NotGiving до фази Giving через пастку request, символізуючи ініціацію акту "віддачі" елемента, а зворотний перехід від Giving до NotGiving через пастку done — завершення цієї операції та повернення до стану очікування.

Аналогічно, для ролі Cons компонента $Filter_i$, що відповідає за отримання елемента, визначено розділ та рольову поведінку, представлені на рисунку 1.4.



а) б)

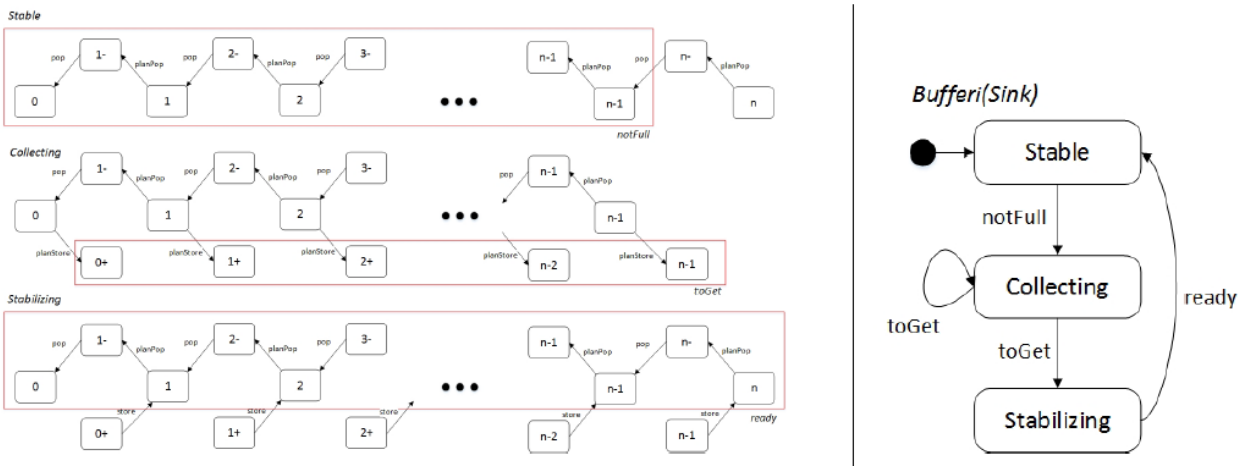
Рисунок 1.4 - Розділ і роль для $Filter_i$ (Cons)

Рисунок 1.4 а) показує розділ Cons з двома фазами: NotTaking та Taking. У фазі NotTaking стан Wanting знаходиться в пастці request, сигналізуючи про бажання фільтра "взяти" новий елемент. У фазі Taking стани Transforming,Finished,Ready включено до пастки done, що вказує на процес обробки або завершення циклу "взяття". Рольова поведінка Filter_i (Cons), представлена на рисунку 1.4 б), ілюструє переходи фази: від NotTaking до Taking через пастку request (символізує ініціацію "взяття" елемента) та від Taking до NotTaking через пастку done (символізує завершення операції "взяття").

1.4.2. Специфікація компонента Buffer

Компонент буфера, позначений як Bufferi, представлений як STD на рисунку 1.2. STD Bufferi моделює циклічний буфер обмеженої місткості n. Стани STD від 0 до n представляють кількість елементів, що наразі зберігаються у буфері, причому стан 0 відповідає порожньому буферу, а стан n — повній місткості (для натуральних чисел). Додатково введені проміжні стани, такі як 0+ (для додавання елемента) та 1− (для видалення елемента), використовуються для моделювання дворівневого процесу операцій збереження та видалення. Операція збереження реалізована як послідовність двох дій: planStore (ініціація збереження, пов'язана зі станом 0+) та store (виконання збереження, перехід до стану 1). Аналогічно, операція видалення складається з дій planPop (ініціація видалення, пов'язана зі станом 1−) та pop (виконання видалення, перехід до стану 0). Ці двоетапні цикли дозволяють моделювати більш складні протоколи взаємодії з буфером.

Взаємодія з буфером координується через його ролі: роль Sink відповідає за операції збереження елементів, а роль Source — за операції видалення. Розділ та рольова поведінка для ролі Sink візуалізовані на рисунку 1.5.

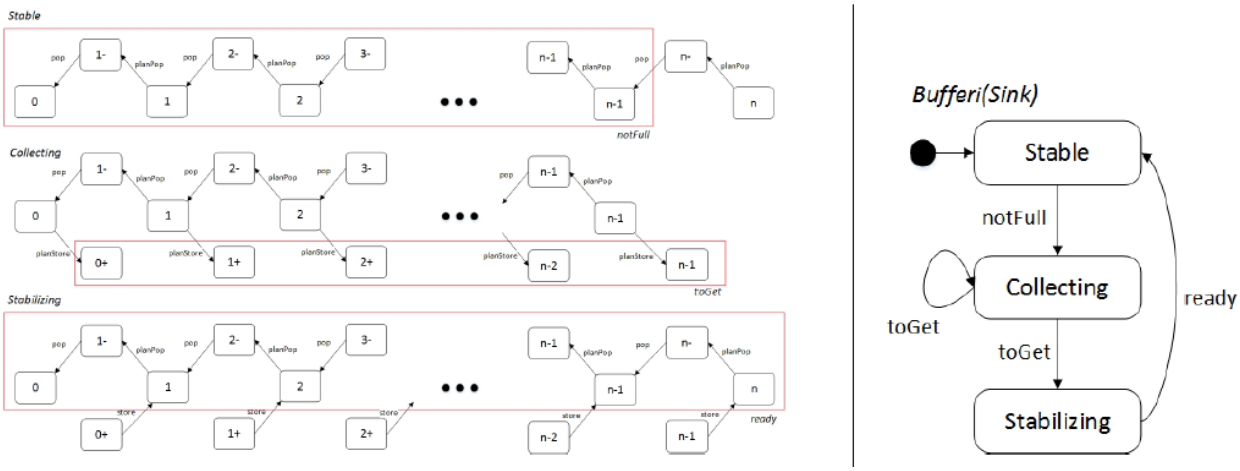


а) б)

Рисунок 1.5 - Розділ і роль для $Buffer_i$ (Sink)

Рисунок 1. 5а показує розділ Sink, що включає фази Stable, Collecting, Stabilizing. Фаза Stable відповідає стану, коли буфер не здійснює активних кроків у циклі збереження. Фаза Collecting асоціюється з початком циклу збереження (дія planStore). Фаза Stabilizing відповідає завершальному етапу збереження (дія store). Важливо, що в будь-якій з цих фаз (Stable, Collecting, Stabilizing) поведінка буфера дозволяє здійснювати будь-які кроки, пов'язані з циклом видалення (через іншу роль), але обмежує виконання до не більше одного кроку циклу збереження в межах даної фази.

Відповідно, розділ та рольова поведінка для ролі Source, що керує видаленням елементів з буфера, візуалізовані на рисунку 1.6.



а) б)

Рисунок 1.6 - Розділ і роль для $Buffer_i$ (Source)

Рисунок 1.6 а) ілюструє розділ Source з фазами Stable, Providing, Stabilizing. Фаза Stable відповідає стану відсутності активних кроків у циклі видалення. Фаза Providing асоціюється з початком циклу видалення (дія planPop). Фаза Stabilizing відповідає завершальному етапу видалення (дія pop). Аналогічно до ролі Sink, у будь-якій з цих фаз (Stable, Providing, Stabilizing) можливі кроки, пов'язані з циклом збереження (через іншу роль), але обмежується виконання до не більше одного кроку циклу видалення в межах даної фази.

Специфікації компонентів Filter та Buffer, представлені у вигляді STD з визначеними розділами та ролями, формують основу для побудови та аналізу взаємодії цих компонентів у складі лінійного конвеєра як архітектурного стилю в Paradigm.

1.5. Комунікаційні механізми та правила узгодженості у моделі лінійного конвеєра Paradigm

У даному підрозділі представлена архітектурна конфігурація лінійного конвеєра, змодельована із застосуванням мови координації Paradigm, та описані механізми взаємодії його компонентів.

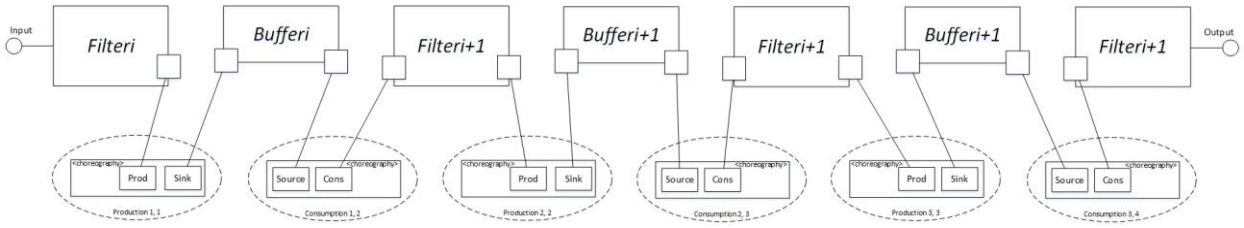


Рисунок 1.7 - Лінійна архітектура конвеєра для моделі Paradigm

Базова архітектура, візуалізована на рисунку 1.7, складається з послідовно з'єднаних компонентів фільтрів (Filteri) та буферів (Bufferi). У типовій секції конвеєра компонент Filteri функціонує як виробник, передаючи елементи до Bufferi, тоді як Filteri+1 виступає як споживач, отримуючи елементи з того ж Bufferi. В контексті такої двокомпонентної взаємодії (Filter↔Buffer), ролі розподіляються наступним чином: роль Prod належить компоненту фільтра, що є виробником для буфера (Filteri у парі Filteri→Bufferi), ролі Source та Sink належать компоненту буфера (Bufferi), а роль Cons належить компоненту фільтра, що є споживачем з буфера (Filteri+1 у парі Bufferi→Filteri+1). Ця структура повторюється вздовж конвеєра.

Взаємодія між компонентами у Paradigm моделюється з використанням специфічних комунікаційних механізмів. Рисунки 1.8 – 1.11 ілюструють процес вертикальної асинхронної комунікації, що відбувається між деталізованою STD компонента, його рольовим портом та відповідним дзеркальним рольовим портом у контексті співпраці. Асинхронний характер комунікації означає, що акт відправки інформації не блокує відправника і її отримання може відбутися з деякою затримкою.

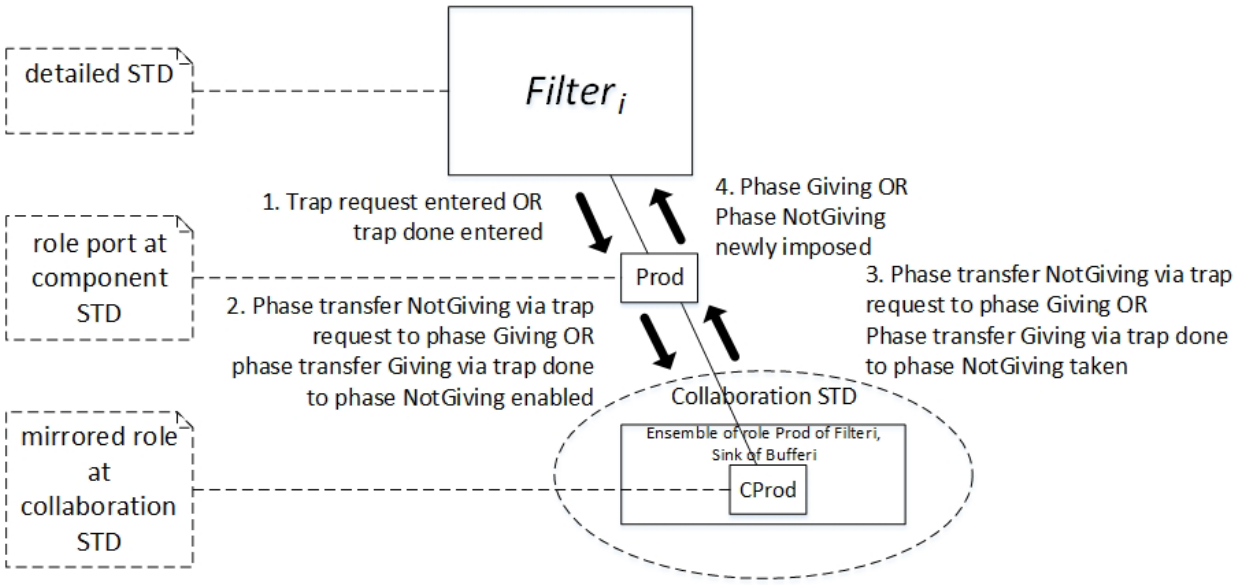


Рисунок 1.8 - Комунікації від деталізованої STD до співпраці

Цикл вертикальної комунікації, який відповідає одному переходу фази, складається з чотирьох основних кроків, візуалізованих на Рисунку 1.8 (для Filter Prod) та аналогічно для інших ролей на Рисунках 1.9 – 1.11 (Filter Cons, Buffer Sink, Buffer Source). Хоча ці схеми є специфічними для моделі Paradigm і не є стандартними діаграмами UML, вони надають чітке уявлення про послідовність обміну інформацією:

Крок 1 (STD → Role Port): Деталізована STD компонента сигналізує про вхід у певну пастку. Ця інформація передається до відповідного рольового порту компонента.

Крок 2 (Role Port → Mirror Role Port): Інформація про вхід у пастку передається від рольового порту компонента до його дзеркального рольового порту, розташованого у просторі співпраці (кооперації) між компонентами.

Крок 3 (Mirror Role Port → Role Port): Залежно від логіки співпраці та синхронізації, дзеркальний рольовий порт надсилає інформацію про необхідну зміну фази назад до рольового порту компонента.

Крок 4 (Role Port → STD): Рольовий порт передає інформацію про нову фазу до деталізованої STD компонента, "нав'язуючи" їй цю фазу як поточну.

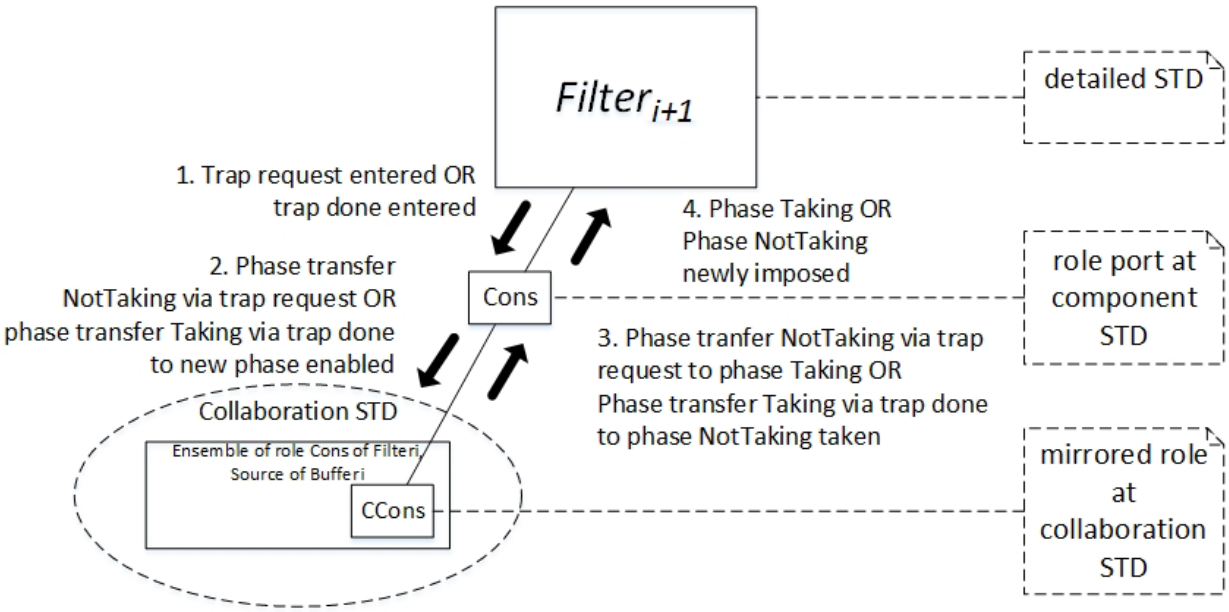


Рисунок 1.9 - Комунікації від деталізованої STD до співпраці

Асинхронні комунікації для буфера Sink між STD Buffer$_i$, де i=1, портом компонента Sink і портом співробітництва CSink наведені на рисунку 1.10, де спосіб такий же, як на рисунку 1.8 і рисунку 1.9. Оскільки буфер має три зміни фази, він має три цикли, один цикл для однієї зміни фази. Таким чином, він пройде три рази через кроки один, два, три і чотири.

Перші два кроки цього циклу пов'язані з передачею інформації про стан пастки, тоді як третій та четвертий кроки забезпечують передачу інформації про зміну фази. Рисунок 1.8 детально показує, як вхід у пастку

'request' у фазі NotGiving для ролі Filter (Prod) може ініціювати цикл, що завершиться нав'язуванням фази Giving. Оскільки буфер, як описано раніше, має три фази у своїх розділах Sink та Source, повний цикл взаємодії з буфером (наприклад, збереження або видалення елемента) може включати проходження через три цикли комунікації, що відповідають змінам його фаз.

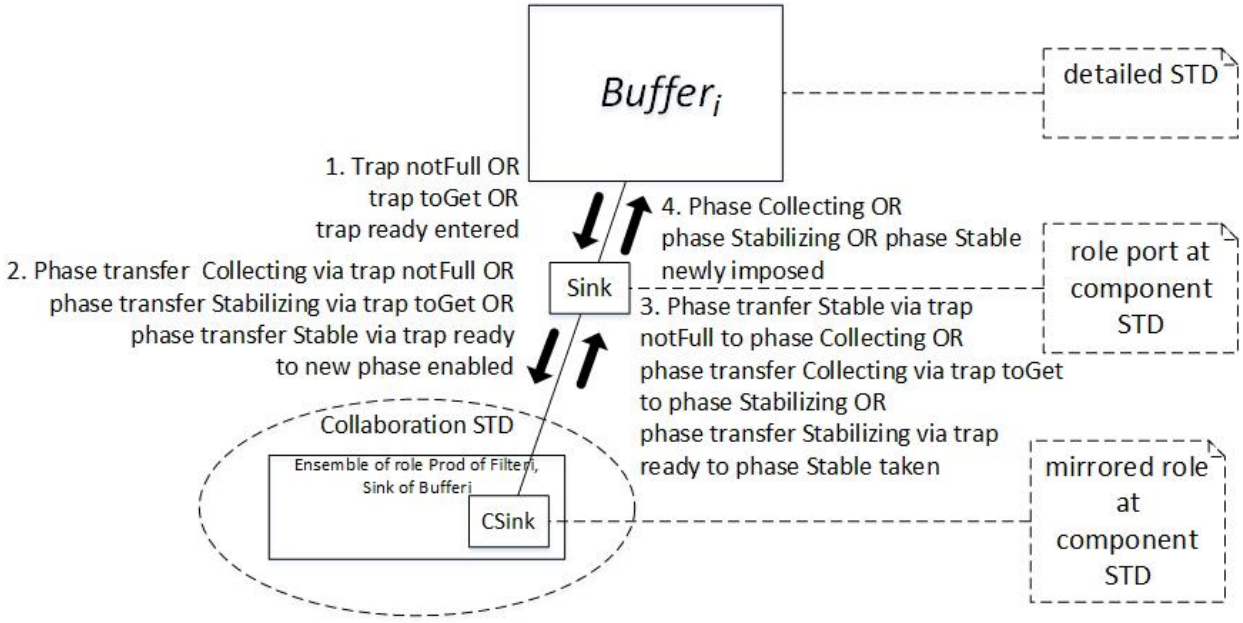


Рисунок 1.10 - Комунікації від деталізованої STD до співпраці

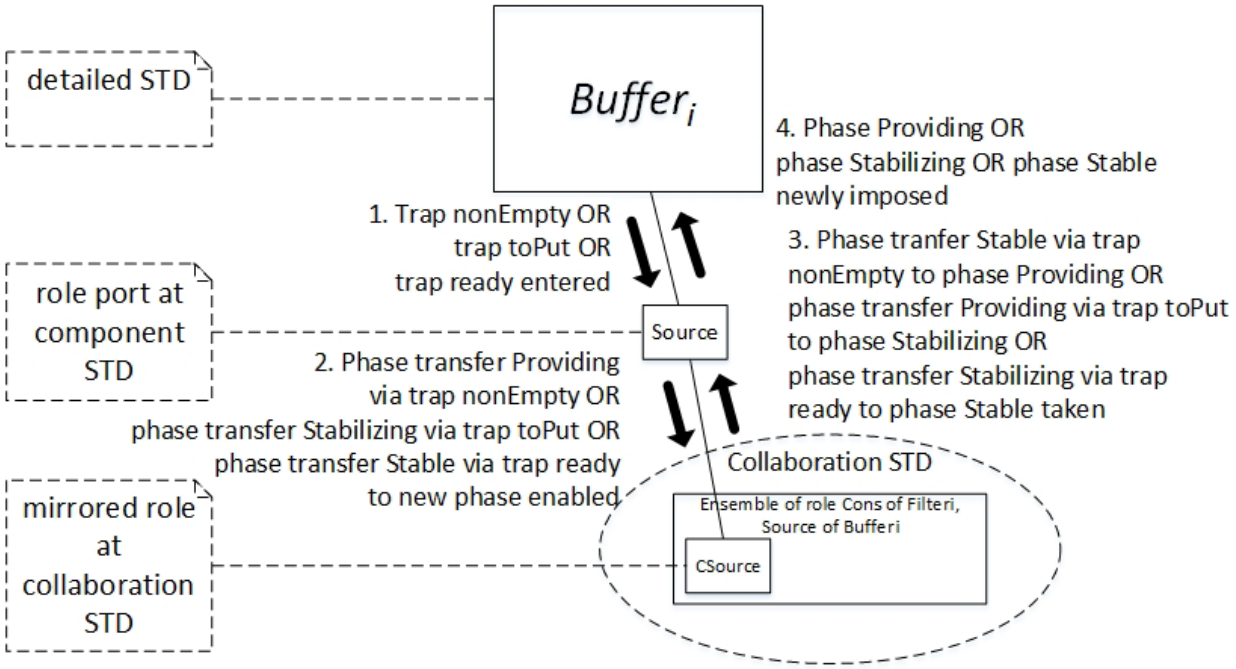


Рисунок 1.11 - Комунікації від деталізованої STD до співпраці

Синхронізація взаємодії між компонентами конвеєра здійснюється за допомогою правил узгодженості. Ці правила визначають, які переходи фаз у різних ролях мають відбуватися одночасно. Нижче наведено правила узгодженості для взаємодії між роллю виробника (Filter i (Prod)) та роллю приймача (Buffer i (Sink)).

Правило узгодженості 1: * Filter_i(Prod) : NotGiving *request* NotGiving, Buffer_i(Sink) : Stable *notFull* Collecting Це правило синхронізує одночасне настання двох подій: перехід фази ролі Filter i (Prod) з NotGiving до NotGiving, спричинений пасткою 'request', та перехід фази ролі Buffer i (Sink) зі Stable до Collecting, спричинений пасткою 'notFull'.

Правило узгодженості 2: * Filter_i(Prod) : NotGiving *request* Giving, Buffer_i(Sink): Collecting *notIer* Collecting Це правило синхронізує перехід фази ролі Filter i (Prod) з NotGiving до Giving через пастку 'request' з переходом фази ролі Buffer i (Sink) з Collecting до Collecting через пастку 'notIer'.

Правило узгодженості 3: * Filter_i(Prod) : Giving *done* NotGiving, Buffer_i(Sink): Collecting *notIer* Stabilizing Це правило синхронізує перехід фази ролі Filter i (Prod) з Giving до NotGiving через пастку 'done' з переходом фази ролі Buffer i (Sink) з Collecting до Stabilizing через пастку 'notIer'.

Правило узгодженості 4: * Buffer_i (Sink) : Stabilizing *modq* Stable Це правило синхронізує перехід фази ролі Buffer i (Sink) зі Stabilizing до Stable через пастку 'modq'.

Правила узгодженості 5-8 (аналогічно до 1-4, але не наведені детально) описують процес синхронізації між роллю джерела буфера (Buffer i (Source)) та роллю споживача фільтра (Filter i+1 (Cons)), використовуючи відповідні фази та пастки цих ролей.

РОЗДІЛ 2. ПРЕДСТАВЛЕННЯ МОДЕЛЬНИХ ТРАНСФОРМАЦІЙ ПРОГРАМНИХ ПАРАДИГМ НА ОСНОВІ ЗАСОБІВ UML

2.1. Опис концепцій UML

UML має деякі різні діаграми, які насправді означають те ж саме. UML — це сімейство графічних позначень, підтримуваних однією метамоделлю, які допомагають описувати та проектувати програмне забезпечення. Також, визначено, що UML — це універсальна візуальна мова моделювання, яка використовується для специфікації, візуалізації, конструювання та документування артефактів програмного забезпечення. Вона фіксує рішення та розуміння систем, які потрібно побудувати. Вона призначена для використання з усіма методами розробки, етапами життєвого циклу, областями застосування та середовищами.

UML — це широко застосовна мова моделювання. Вона не є лише мовою моделювання для програмних систем, але також застосовна для моделювання бізнес-процесів та непрограмних систем.

UML має тринадцять різних типів діаграм. Нижче в таблиці 2.1 наведено з коротким описом призначення та використання конкретної діаграми UML. Таблиця підсумовує перекладені діаграми, використані в цій роботі, та в контексті вищезгаданого прикладу конвеєра. Зверніть увагу, що діаграма станів поміщена як перша діаграма. Це тому, що діаграма станів дуже важлива, оскільки моделі Paradigm насправді є діаграмами станів.

Таблиця 2.1 - Діаграми UML, які використовуються для візуалізації поведінки конвеєра, заданого в Paradigm

Діаграма	Призначення та короткий опис
Діаграма станів	Локальна поведінка об'єкта. Дає детальний огляд станів і переходів. Усі діаграми станів ґрунтуються на STD. Конвеєр у Paradigm складається з STD; Filteri, Bufferi, Filteri роль Prod, Filteri роль Cons, Bufferi роль Sink і Bufferi роль Source. Також ролі в співпрацях є STD, CProd, CCons,

Діаграма	Призначення та короткий опис
	CSink і CSource.
Діаграма комунікацій	Показує шляхи комунікації між учасниками. Учасники в даному випадку для Filteri: Filteri, Prod і CProd. Для Filteri+1: Filteri+1, Cons і CCons. Для Bufferi: Bufferi, Sink і CSink і останній для Bufferi: Bufferi, Source, CSource. Це означає, що для кожного Prod, Cons, Sink і Source представлений шлях комунікації.
Діаграма послідовності	Взаємодія між учасниками. Діаграма послідовності ґрунтується на комунікації в діаграмі комунікацій, але тепер зроблена в діаграмі послідовності. Як і діаграма комунікацій, діаграма послідовності не нумерується. Діаграма послідовності ґрунтується на її вертикальному читанні. Стрілка на кінці переходу повідомлення показує, чи повідомлення синхронне чи асинхронне.
Діаграма компонентної структури	Показує структуру конвеєра з фільтрами, буферами та рольовими портами. Виглядає як лінійний потік даних від частини та рольового порту до наступного рольового порту та частини.
Діаграма співпраці	Співпраця — це підмова та насправді належить до діаграми компонентної структури. Діаграма компонентної структури використовує цю співпрацю. У співпраці вказані ролі CProd, CCons і CSink, CSource. З допомогою співпраці фільтр спілкується через роль CProd і CSink з буфером. Він також спілкується через буфери CSource з фільтром CCons.
Діаграма компонентів	Структура та з'єднання компонентів, що використовуються з портами. Компонент представляє фільтр і буфер. Співпраця також створюється як компонент для візуалізації зв'язку CProd, CSink і CSource, CCons, який називається компонентом співпраці. Фільтр має наданий інтерфейс, де порт має необхідний інтерфейс. Такий же спосіб використовується для буфера.
Діаграма класів	Показує вхідні та вихідні сигнали класу. Клас — це один STD, що означає, що є клас для Filteri, Bufferi, Prod, Cons, Sink, Source, CProd, CCons, CSink і CSource. Також є Prod і Cons, представлені з композицією. Для Prod це CProd і CSink, а для Cons композиція до CSource і CCons. Де використовується композиція, клас розглядається як співпраця. Це означає, що Prod і Cons належать до цього класу через композицію.
Діаграма пакетів	Діаграма пакетів представляє пакети для Filteri, Bufferi та один пакет співпраці. Вкладені пакети для Filteri — Prod і Cons. Вкладені пакети для Bufferi — Sink і Source. Для співпраць також створено два пакети. Пакети співпраць тоді мають для Prod: CProd і CSink і для Cons: CSource і CCons.
Діаграма розгортання	Показує фізичні аспекти конвеєра. Тут є фільтр, буфер, співпраця Prod і співпраця Cons.
Діаграма активності	Глобальна поведінка серед класів і об'єктів. Об'єкти можуть бути компонентами або пакетами. Лінії плавання показують, які об'єкти щось роблять, іноді разом. Атрибути (дані) та назви дій, повідомлень і

Діаграма	Призначення та короткий опис
	процедур. Це означає, що діаграма активності може дійсно моделювати активність, яку здійснює задана модель.
Діаграма огляду взаємодії	Це змішання діаграм послідовності та активності. Діаграми послідовності, які використовуються для візуалізації, наприклад, потоку Filteri, Prod і CProd, є однією діаграмою послідовності для Prod. Також для діаграми послідовності Cons, Sink і CSink. Правила узгодженості забезпечують паралельний потік Prod, Sink і Cons, Source. Взаємодія ґрунтується на діаграмі послідовності, поведінці правил узгодженості та діаграмі активності.
Діаграма часу	Показує огляд взаємодії між учасниками. Учасник, як раніше сказано, наприклад, Filteri, Prod і CProd. Filteri має деякі стани, те ж саме для Prod і CProd, які також мають стани. Діаграма часу показує поведінку цих станів і як довго вони залишаються в поточному стані.
Діаграма випадків використання	Як користувачі взаємодіють з системою. Показує, як конкретні користувачі, названі акторами, використовують систему. Діаграма випадків використання ґрунтується на Filteri і Bufferi з використаними в цих моделях станами як випадки використання. Опис випадку використання уточнює діаграму випадків використання.

2.2. Трансформація моделей компонентів та комунікації у представлення засобами UML

Даний підрозділ присвячений процесу трансформації моделей компонентів лінійного конвеєра, розроблених у формалізмі Paradigm (зокрема, моделей фільтрів Filteri та буферів Bufferi з відповідними портами та ролями, описаних у першому розділі), у комплексний набір діаграм уніфікованої мови моделювання (UML). Метою є представлення всіх аспектів Paradigm-моделі за допомогою тринадцяти типів діаграм UML.

2.2.1. Трансформація STD, розділів та ролей

Основними елементами, що підлягають трансформації, є діаграми станів (STD) компонентів Filter (Filteri, де i=1,2 або 3) та Buffer (Bufferi, де i=1 або 2). Ці деталізовані STD компонентів трансформуються у відповідні Діаграми станів UML. Аналогічно, поведінка рольових портів (Prod, Cons,

Sink, Source) компонента також представляється у вигляді Діаграм станів UML, що моделюють їхню реакцію на вхідні та вихідні повідомлення.

Взаємодія між компонентами та їхніми ролями у моделях Paradigm значною мірою базується на механізмах вертикальної комунікації. Як було описано раніше, вертикальна комунікація відбувається між деталізованою STD компонента, його рольовим портом та відповідним дзеркальним рольовим портом у контексті співпраці (collaboration). Цей тип комунікації, візуалізований у Paradigm-схемах (рис. 1.8 – 1.11), що ілюструють взаємодію між Filteri (STD), портами Prod/Cons та співпрацями CProd/CCons, а також між Bufferi (STD), портами Sink/Source та співпрацями CSink/CSource, представляє послідовність обміну інформацією, що реалізує рольову поведінку.

Комунікація між дзеркальними портами співпраць (наприклад, CProd і CSink, CSource і CProd) у Paradigm називається горизонтальною комунікацією.

Вертикальна комунікація між деталізованою STD компонента, його рольовим портом та дзеркальним рольовим портом співпраці може бути представлена за допомогою стандартних діаграм взаємодії UML, зокрема діаграм комунікацій та діаграм послідовності.

2.2.2. Трансформація аспектів Filter Prod

Розглянемо трансформацію моделі вертикальної комунікації для ролі Prod компонента Filter, візуалізовану у Paradigm на рисунку 1.8. Ця комунікація представлена у вигляді діаграми комунікацій UML на рисунку 2.1 та діаграми послідовності UML на рисунку 2.2. Учасники комунікації — деталізована STD компонента Filteri (для i=1,2 або 3), його рольовий порт Prod та дзеркальний рольовий порт співпраці CProd — представлені як лінії життя (lifelines) на цих діаграмах.

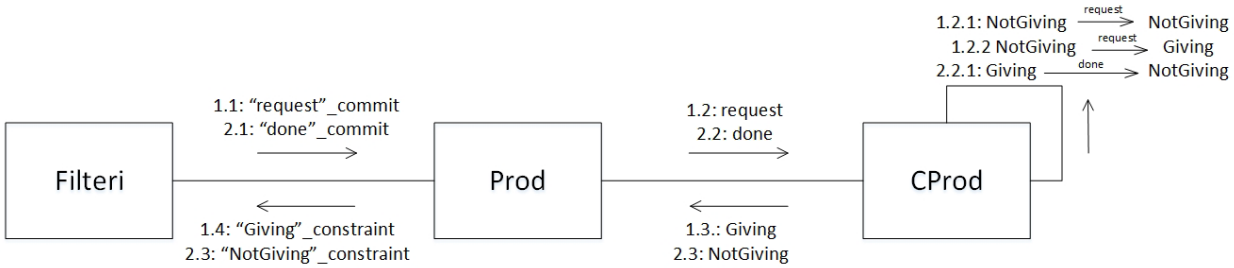


Рисунок 2.1 – Діаграма послідовності комунікацій фільтра

Діаграми взаємодії (рисунки 2.1 та 2.2) ілюструють послідовності повідомлень, що відповідають двом повним циклам переходів фази ролі Prod: перший цикл відображає перехід з фази NotGiving до фази Giving, а другий — зворотний перехід з фази Giving до фази NotGiving. Кожен цикл включає обмін вісьмома повідомленнями у Діаграмі комунікацій.

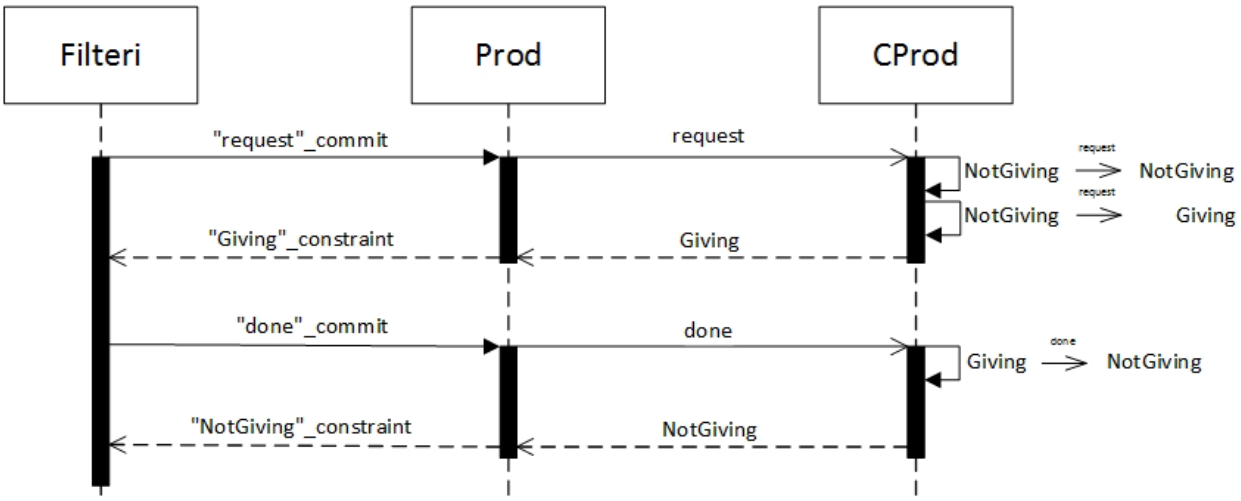


Рисунок 2.2 – Діаграма послідовності комунікацій фільтра

На діаграмі послідовності (рис. 2.2) тип повідомлення (синхронне або асинхронне) визначається стилем стрілки: суцільна зафарбована стрілка позначає синхронне повідомлення, тоді як незафарбована стрілка з відкритою головкою — асинхронне. Наприклад, перше повідомлення від Filteri до Prod є синхронним, тоді як друге повідомлення від Prod до CProd є асинхронним.

Важливо розрізняти повідомлення, що відображають вхід у пастку та повідомлення, що ініціюють зміну фази. Наприклад, повідомлення з мітками типу 1.2.1 NotGiving → request NotGiving, 1.2.2 NotGiving → request Giving та 2.2.1 Giving → request NotGiving (як показано на рис. 2.1) ілюструють передачу інформації про вхід у пастку ('request' або 'done') та потенційні фази, які можуть бути активовані. Сама зміна фази (наприклад, перехід до фази Giving або NotGiving) моделюється іншими повідомленнями в послідовності, що відповідають крокам 3 та 4 вертикальної комунікації (наприклад, мітки 1.3 та 2.3 на рис. 2.1). Ці повідомлення відображають нав'язування нової фази компоненті та її рольовому порту після узгодження на рівні співпраці, що може бути обумовлене правилами узгодженості (наприклад, Правилом 1, згаданим раніше). На Діаграмі послідовності (рисунок 2.2) такий процес узгодження на рівні співпраці CProd може бути візуалізований як "самовиклик" (self-call), де обробляється отримана інформація про пастку перед тим, як ініціювати зворотну комунікацію для зміни фази.

2.2.3. Трансформація деталізованої STD у діаграму станів UML

Базові деталізовані STD компонентів Filteri (для i=1,2 або 3), як візуалізовано на рис 1.1, трансформуються у діаграми станів UML. Діаграма станів UML для Filter візуалізує ті ж чотири стани, що і в оригінальній STD Paradigm (Wanting, Transforming, Finished, Ready), та відповідні переходи між ними.

Перехід у Діаграмі станів UML формально визначається структурою: тригер [умова] / поведінка_переходу. Не всі три частини обов'язково присутні для кожного переходу. Рисунки 2.3 – 2.5 ілюструють поступове уточнення представлення переходів фільтра в UML:

Рисунок 2.3 - Діаграма станів фільтра з поведінкою переходу. На цьому рисунку представлена спрощена візуалізація переходів, де вказано лише поведінку переходу (transition behavior). Наприклад, перехід від стану

Wanting до Transforming асоційований з поведінкою take. Ці поведінки відповідають діям компонента у Paradigm.

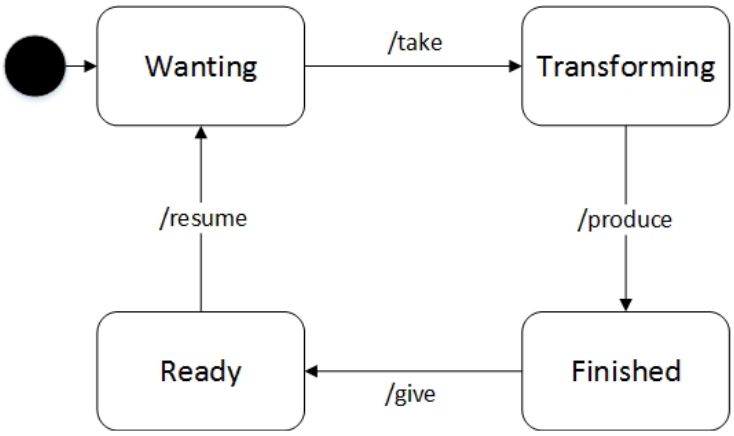


Рисунок 2.3 - Діаграма станів фільтра з поведінкою переходу

Рисунок 2.4 - Діаграма станів фільтра з умовами. Додано умови (guards) до переходів. Умови типу [Phase NotGiving is imposed] або [Phase Giving is imposed] використовуються для моделювання того, що перехід можливий лише тоді, коли компоненту нав'язана відповідна фаза згідно з логікою координації. Перехід відбувається лише тоді, коли його умова істинна.

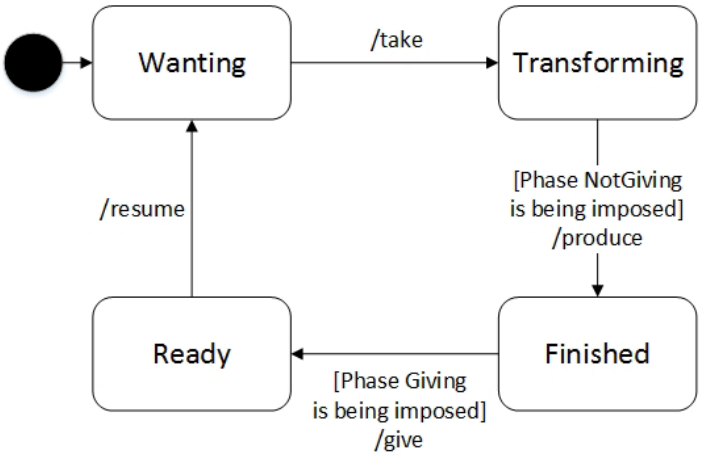


Рисунок 2.4 - Діаграма станів фільтра з умовами

Рисунок 2.5 - Діаграма станів фільтра з умовами та поведінкою переходу. Надає повне представлення переходів, комбінуючи умови та

поведінку переходу. Крім того, тут візуалізовано дії відправки (send actions) як частину поведінки переходу. Дія відправки в UML позначається символом ^ або ! перед назвою дії і має формат порт.повідомлення. Наприклад, дія відправки Prod.request означає надсилання повідомлення request до порту Prod. Ці дії відправки моделюють передачу інформації про пастки та фази з деталізованої STD до рольових портів, що є першим кроком вертикальної комунікації.

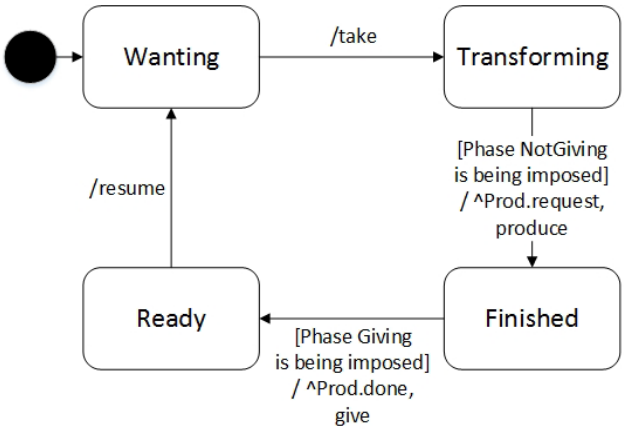


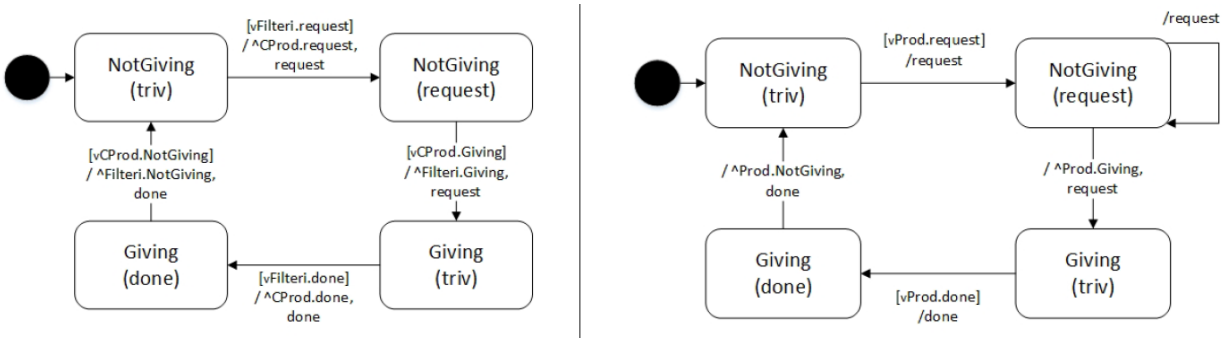
Рисунок 2.5 - Діаграма станів фільтра з умовами та поведінкою переходу

2.2.4. Трансформація рольових портів у діаграми станів UML

Поведінка рольових портів компонента Prod та дзеркального рольового порту співпраці CProd додатково описана і візуалізована у вигляді діаграм станів UML на рисунку 2.6 (рисунок 2.6 а для порту компонента Prod, рисунок 2.6 б для порту співпраці CProd). Ці діаграми моделюють стани самих портів, які відображають їхню участь у процесі комунікації та координації (наприклад, стани типу NotGiving(triv), NotGiving(request), Giving(triv), Giving(done)).

Тригерами переходів на діаграмах станів портів є дії прийому (receive actions), часто інтегровані в умови переходу (використовуючи, наприклад,

позначення типу V в умові). Дії відправки (позначені ∧ або !) є поведінкою переходу.



а) Рольовий порт компонента Prod б) Рольовий порт співпраці CProd

Рисунок 2.6 – Діаграми станів рольових портів компонента та співпраці

Розглянемо, як чотири кроки вертикальної комунікації для ролі Prod (рис. 1.8) реалізуються через взаємодію цих трьох діаграм станів UML (рис. 2.5, 2.6 а, 2.6б):

- Крок 1 (STD → Role Port): Коли у Діаграмі станів Filter і (рис. 2.5) виконується перехід, що має дію відправки, наприклад, ^Prod.request, ця дія надсилає повідомлення request до порту Prod.

- Крок 2 (Role Port → Mirror Role Port): Діаграма станів порту Prod (рис. 2.6 а), отримавши повідомлення request, переходить у стан, що відповідає обробці цієї пастки (наприклад, NotGiving(request)). Як поведінку переходу, порт Prod може виконати дію відправки, наприклад, ^CProd.request, передаючи інформацію про пастку до дзеркального порту CProd.

- Крок 3 (Mirror Role Port → Role Port): Діаграма станів порту CProd (рис. 2.7 б), отримавши повідомлення від Prod (наприклад, request), обробляє його (можливо, з урахуванням правил узгодженості) і виконує перехід, який може включати дію відправки, наприклад, ^Prod.Giving або ^Prod.NotGiving. Це повідомлення містить інформацію про нову фазу, яку має прийняти компонент.

- Крок 4 (Role Port → STD): Діаграма станів порту Prod (рис. 2.6 а), отримавши повідомлення про нову фазу від CProd (наприклад, Giving), виконує перехід у відповідний стан і надсилає повідомлення назад до деталізованої STD Filter i (рис. 2.5), наприклад, дією відправки ^Filter_i.Giving. Ця дія моделює нав'язування нової фази (Giving) компоненті Filter i. Діаграма станів Filter i реагує на це "вхідне" повідомлення про фазу, можливо, як на тригер або умову для подальших внутрішніх переходів станів.

Таким чином, діаграми станів UML для компонента та його портів, разом із Діаграмами взаємодії, забезпечують деталізоване представлення поведінки та комунікації, змодельованих у Paradigm, використовуючи стандартизовану нотацію UML.

Продовження трансформації моделей компонентів та комунікації Paradigm у представлення засобами UML 2.0

Цей розділ продовжує опис трансформації моделей компонентів лінійного конвеєра з формалізму Paradigm у представлення засобами UML 2.0, зосереджуючись на ролі споживача фільтра, об'єднаному представленні фільтра та ролі приймача буфера.

2.3. Трансформація аспектів Filter Cons

Аналогічно до представлення комунікації для ролі виробника, вертикальна комунікація для ролі споживача компонента Filter (Filteri Cons), візуалізована у Paradigm на рисунку 1.9, трансформована у діаграму комунікацій UML (рис. 2.7) та діаграму послідовності UML (рис. 2.8). Учасниками комунікації є деталізована STD компонента Filteri (i=1,2 або 3), його рольовий порт Cons та дзеркальний рольовий порт співпраці CCons. Ці

діаграми також ілюструють два повних цикли переходів фази ролі Cons: від NotTaking до Taking та від Taking назад до NotTaking. Кожен цикл представлений послідовністю з восьми повідомлень на діаграмі комунікацій.

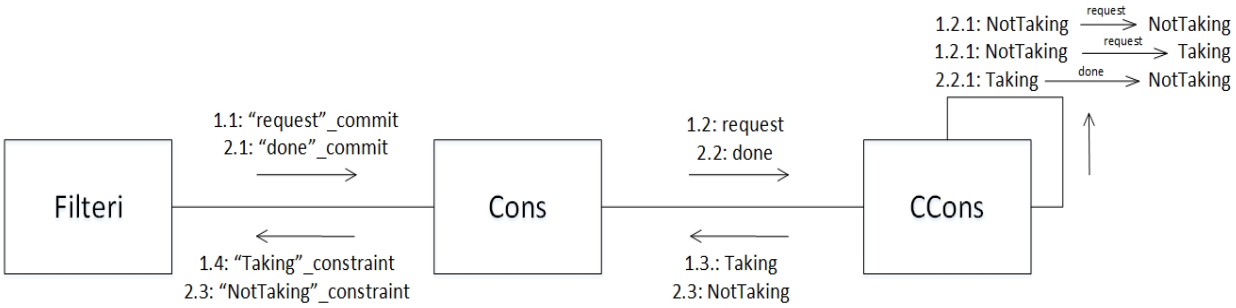


Рисунок 2.7 - Діаграма комунікацій фільтра

Діаграма послідовності (рис. 2.8) відображає послідовність обміну повідомленнями. Перше повідомлення від Filteri до Cons є синхронним, тоді як друге повідомлення від Cons до CCons є асинхронним, відповідно до загальних правил візуалізації UML. Послідовність повідомлень відображає логіку комунікації, яка може бути пов'язана з правилами узгодженості, наприклад, Правилом 5 (згаданим у попередньому розділі).

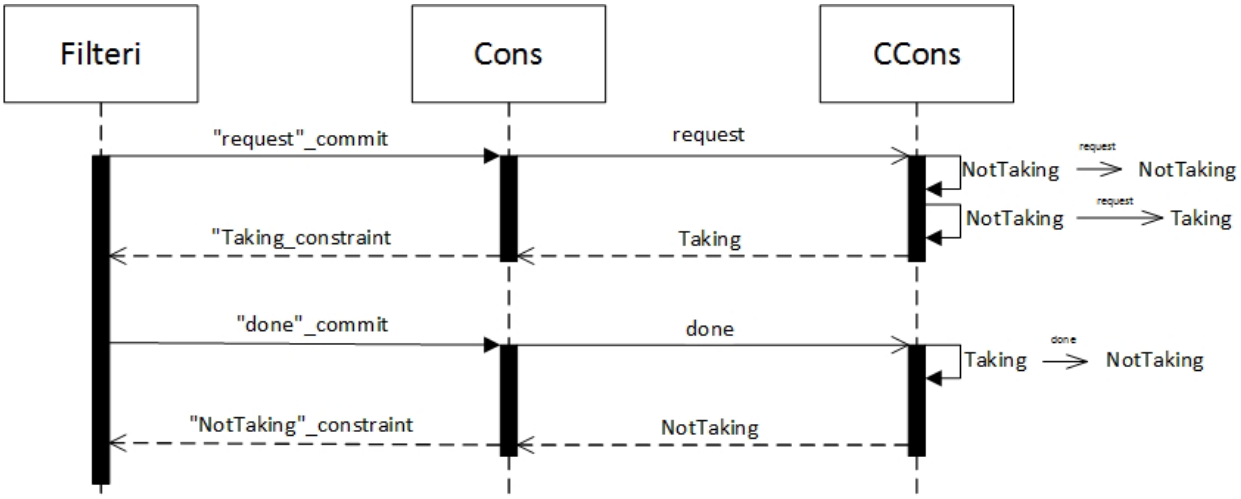


Рисунок 2.8 - Діаграма послідовності фільтра

Як і у випадку з роллю Prod, на діаграмі послідовності може бути візуалізовано самовиклик на лінії життя CCons, що ілюструє внутрішню обробку повідомлення (наприклад, обробку сигналу NotTaking), яка передує надсиланню відповіді. Завершення першого циклу призводить до нав'язування фази Taking, активація якої ініціює другий цикл комунікації, що відповідає поверненню до фази NotTaking.

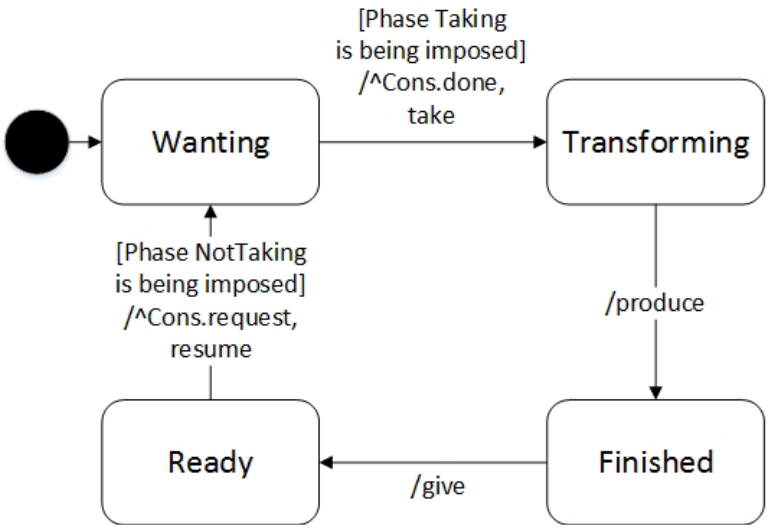
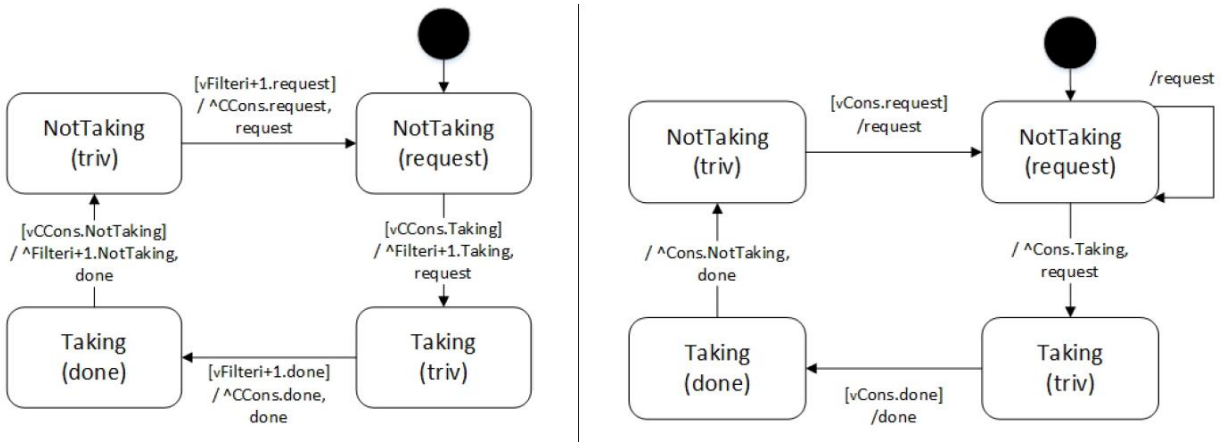


Рисунок 2.9 - Діаграма станів фільтра з умовами та поведінкою переходу

Деталізована STD компонента Filteri, що функціонує як споживач, представлена у вигляді діаграми станів UML на рисунку 2.9. Ця діаграма представляє повну специфікацію переходів фільтра в контексті його ролі споживача. Хоча набір станів та базова поведінка переходів (наприклад, take, produce) є спільними для обох ролей фільтра (виробника та споживача), Діаграма станів для ролі Cons включає умови (guards), специфічні для фаз цієї ролі ([Phase NotTaking is imposed], [Phase Taking is imposed]), та дії відправки (send actions), що відображають комунікацію з портом Cons (наприклад, ^Cons.request, ^Cons.done). Ці умови та дії відправки визначають, коли можливе виконання тих чи інших локальних переходів станів компонента в залежності від поточної нав'язаної фази та необхідності сигналізувати про вхід у пастки, асоційовані з роллю Cons.

Процес вертикальної комунікації для ролі Cons, як і для ролі Prod, реалізується через взаємодію між діаграмами станів Filter STD (рис. 2.9), рольового порту Cons (аналогічно рис. 2.6 а) та дзеркального рольового порту CCons (аналогічно рис. 2.6 б). Наприклад, коли умова [Phase NotTaking is imposed] стає істинною в діаграмі станів Filter i (рис. 2.9), і виконується відповідний перехід (наприклад, з Wanted до Transforming), як поведінка цього переходу може бути виконана дія відправки ^Cons.request.



а) Рольовий порт компонента Cons б) Рольовий порт співпраці CCons

Рисунок 2.10 - Діаграми станів рольових портів компонента та співпраці

Ця дія відправки ініціює перехід у діаграмі станів порту Cons (рис. 2.10а - схожа структура), який, отримавши повідомлення request, може виконати дію відправки ^CCons.request. Це повідомлення надходить до діаграми станів порту CCons (рис. 2.10 б - схожа структура), де воно обробляється. Після обробки (можливо, після синхронізації за правилами узгодженості), порт CCons надсилає повідомлення про зміну фази назад до порту Cons (наприклад, ^Cons.Taking). Порт Cons, отримавши це повідомлення, переходить у новий стан (наприклад, Taking(triv)) і надсилає повідомлення про нав'язування фази до STD Filter i (наприклад, ^Filter_i.Taking), що завершує цикл комунікації та нав'язує Filter i нову фазу

Taking. Аналогічна послідовність повідомлень відбувається при переході фази назад до NotTaking.

2.3.1. Об'єднане представлення Filter STD у UML

Оскільки компонент Filteri в моделі Paradigm (рис. 1.1) інтегрує поведінку, необхідну як для ролі виробника, так і для ролі споживача, його деталізована STD включає всі дії, що використовуються обома ролями (take, produce, give, resume).

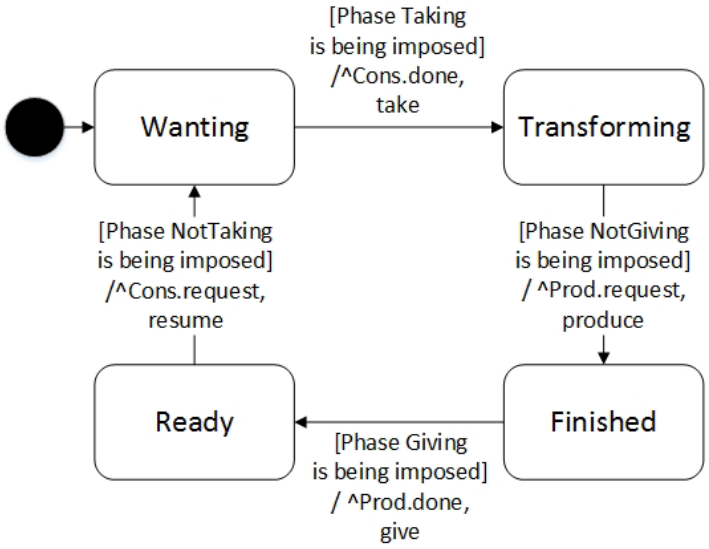


Рисунок 2.11 - Об'єднана діаграма станів фільтрів виробництва та споживання

Відповідно, діаграма станів UML для Filteri, представлена на рисунку 2.11, об'єднує переходи, умови та дії відправки з діаграм станів, орієнтованих окремо на ролі Prod (рис. 2.5) та Cons (рис. 2.9). Це означає, що на одній діаграмі станів Filter і (рис. 2.11) візуалізовано переходи, які можуть бути активовані в залежності від нав'язаної фази для ролі Prod (з умовами типу [Phase Giving is imposed]) або для ролі Cons (з умовами типу [Phase Taking is imposed]), а також дії відправки, адресовані як порту Prod (^Prod.message), так і порту Cons (^Cons.message). Таким чином, Діаграма станів Filter і у

UML відображає повну локальну поведінку компонента з урахуванням його участі в різних співпрацях через відповідні ролі.

2.3.2. Трансформація аспектів Buffer Sink

Трансформація вертикальної комунікації для ролі приймача компонента Buffer (Bufferi Sink), візуалізованої у Paradigm на рисунку 1.10, здійснюється аналогічно до трансформації комунікацій фільтрів. Ця комунікація представлена у вигляді діаграми комунікацій UML (рис. 2.12) та діаграми послідовності UML (рис. 2.13).

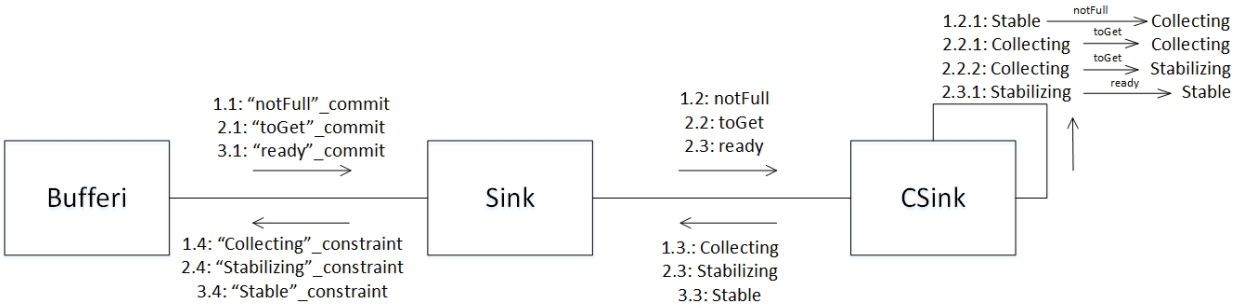


Рисунок 2.12 - Діаграма комунікацій буфера

Учасниками є деталізована STD компонента Bufferi (i=1 або 2), його рольовий порт Sink та дзеркальний рольовий порт співпраці CSink. Відмінністю від комунікацій фільтрів є кількість циклів, які ілюструють ці діаграми: для ролі Sink, що має три фази (Stable, Collecting, Stabilizing) у відповідному розділі, візуалізуються три цикли переходів фази, що включають загалом дванадцять повідомлень на Діаграмі комунікацій. Перший цикл відповідає переходу зі Stable до Collecting, другий — з Collecting до Stabilizing, а третій — зі Stabilizing назад до Stable.

Діаграма послідовності (рис. 2.13) також використовує три лінії життя для учасників Buffer i, Sink та CSink і візуалізує послідовність обміну повідомленнями, включно з синхронними та асинхронними повідомленнями, та можливими самовикликами на лінії життя CSink для обробки отриманої

інформації про пастку (наприклад, обробка сигналу notFull, що може бути пов'язано з Правилом узгодженості 2). Ці цикли комунікації відповідають послідовним змінам фаз ролі Sink, що відображають процес додавання елемента до буфера.

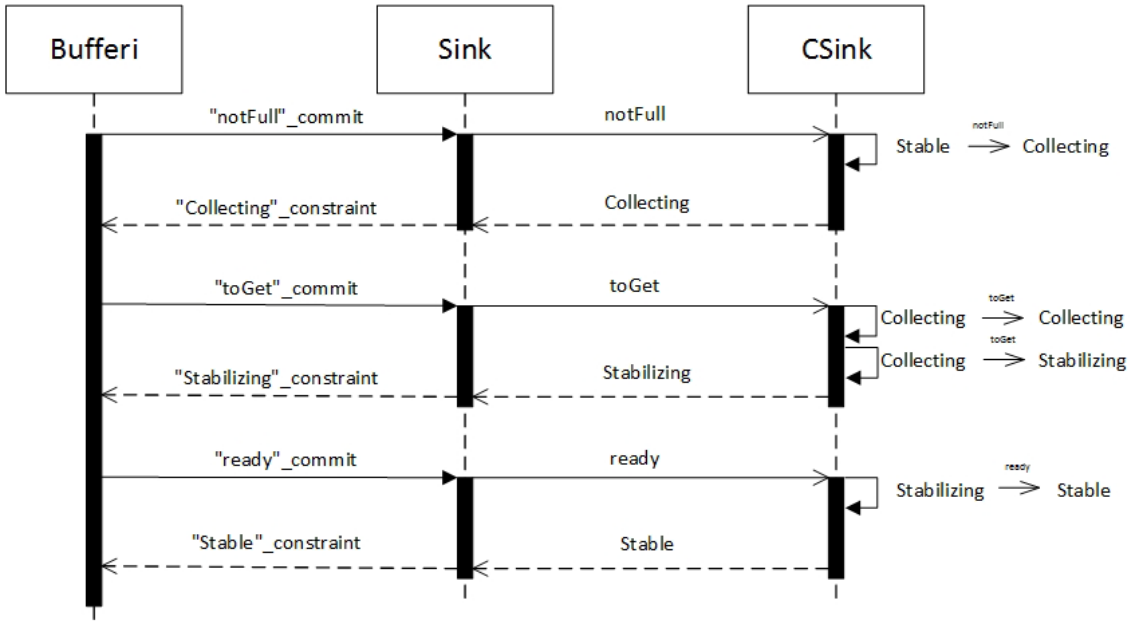


Рисунок 2.13 – Діаграма послідовності комунікацій буфера

Деталізована STD компонента Bufferi у контексті ролі Sink представлена у вигляді діаграми станів UML на рисунку 2.14. Ця діаграма моделює стани буфера, що відображають його місткість (від 0 до n) та проміжні стани, пов'язані з операціями додавання/видалення (0+,1− згідно з рис. 1.2). Переходи на Діаграмі станів включають умови, що ґрунтуються на нав'язаних фазах ролі Sink ([Phase Collecting is imposed], [Phase Stabilizing is imposed], [Phase Stable is imposed]), та дії відправки, що сигналізують про вхід у пастки, асоційовані з цими фазами та діями збереження (наприклад, ^Sink.toGet при переході до стану 1+, ^Sink.ready при переході до стану 1). Важливо, що переходи, пов'язані з діями planPop та pop (які стосуються ролі Source), також присутні на цій Діаграмі станів компонента Buffer, але їх активація залежить від інших умов, пов'язаних з роллю Source.

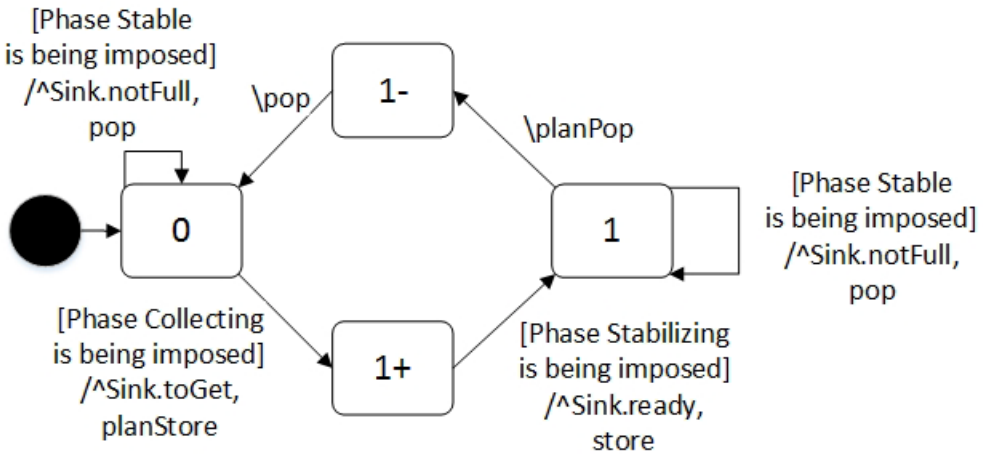
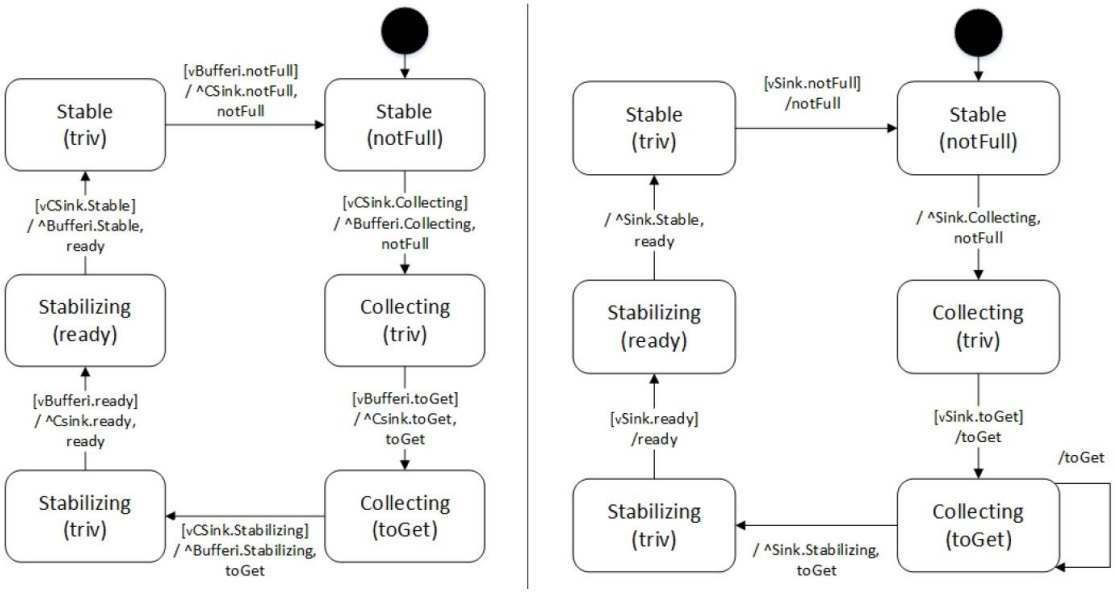


Рисунок 2.14 - Діаграма станів буфера з умовами, відправками та діями

Процес вертикальної комунікації для ролі Sink (рисунок 1.10) та відповідні зміни станів Діаграм станів (рис. 2.14 для STD Buffer та рис. 2.15 для портів Sink та CSink) детально ілюструють, як відбувається зміна фаз ролі Sink. Наприклад, коли нав'язана фаза Stable ([Phase Stable is imposed]) істинна, і буфер готовий прийняти елемент (наприклад, його місткість менша за n), може бути виконана дія відправки ^Sink.notFull.



а) Рольовий порт компонента Sink б) Рольовий порт співпраці CSink

Рисунок 2.15 - Діаграми станів рольових портів компонента та співпраці

Це повідомлення приймається діаграмою станів порту Sink (рисунок 2.15 а), яка, у свою чергу, взаємодіє з діаграмою станів порту CSink (рисунок 2.15б). Після обробки на рівні співпраці CSink, порт CSink надсилає повідомлення про зміну фази до порту Sink (наприклад, ^Sink.Collecting). Порт Sink приймає це повідомлення, змінює свій стан (наприклад, на Collecting(triv)) і надсилає відповідне повідомлення до STD Buffer i (рисунок 2.14), нав'язуючи йому нову фазу Collecting. Аналогічні послідовності повідомлень та змін станів відбуваються при переходах до фази Stabilizing та поверненні до Stable.

2.4. Трансформація аспектів Buffer Source засобами UML

Цей розділ завершує опис трансформації моделей компонентів лінійного конвеєра з формалізму Paradigm у представлення засобами UML, деталізуючи трансформацію ролі джерела буфера та об'єднаного представлення STD буфера.

Аналогічно до представлення комунікації для ролі приймача буфера, вертикальна комунікація для ролі джерела компонента Buffer (Bufferi Source), візуалізована у Paradigm на рисунку 1.11, трансформована у діаграму комунікацій UML (рисунок 2.16) та діаграму послідовності UML (рисунок 2.17).

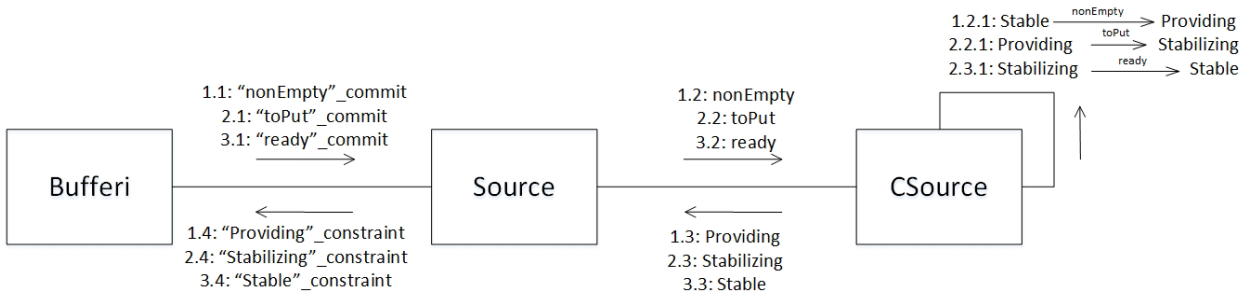


Рисунок 2.16 - Діаграма комунікацій буфера

Учасниками комунікації є деталізована STD компонента Bufferi (i=1 або 2), його рольовий порт Source та дзеркальний рольовий порт співпраці CSource. Як і у випадку з роллю Sink, ці діаграми ілюструють три повних цикли переходів фази для ролі Source (Stable, Providing, Stabilizing), що включають загалом дванадцять повідомлень на Діаграмі комунікацій. Ці цикли відповідають послідовності дій, пов'язаних з видаленням елемента з буфера.

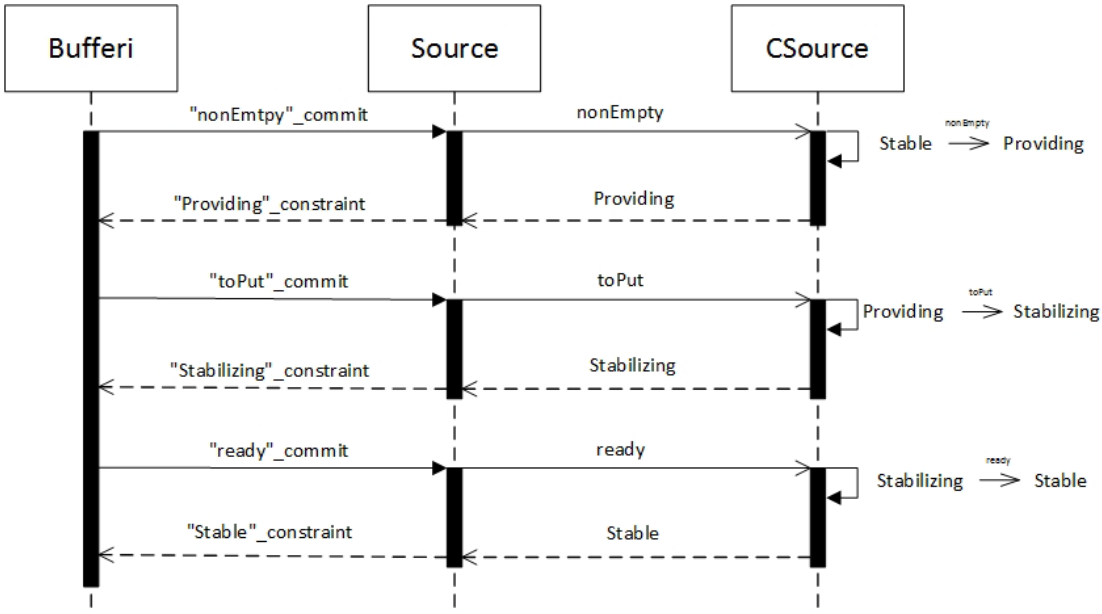


Рисунок 2.17 – Діаграма послідовності комунікацій буфера

Діаграма послідовності (рисунок 2.17) використовує три лінії життя для учасників Buffer і, Source та CSource і візуалізує послідовність обміну повідомленнями, включно з синхронними та асинхронними повідомленнями. На відміну від деяких попередніх прикладів, на цій діаграмі послідовності відсутній самовиклик на лінії життя дзеркального порту співпраці CSource, що може відображати іншу логіку обробки повідомлень на рівні співпраці для ролі Source. Ці цикли комунікації відображають процес переходу фаз ролі Source, що ініціюється активацією відповідної фази та подальшим обміном повідомленнями. Завершення першого циклу призводить до нав'язування фази Providing, активація якої ініціює другий цикл, що

відповідає переходу до фази Stabilizing. Завершення другого циклу активує третій цикл, що відповідає поверненню до фази Stable.

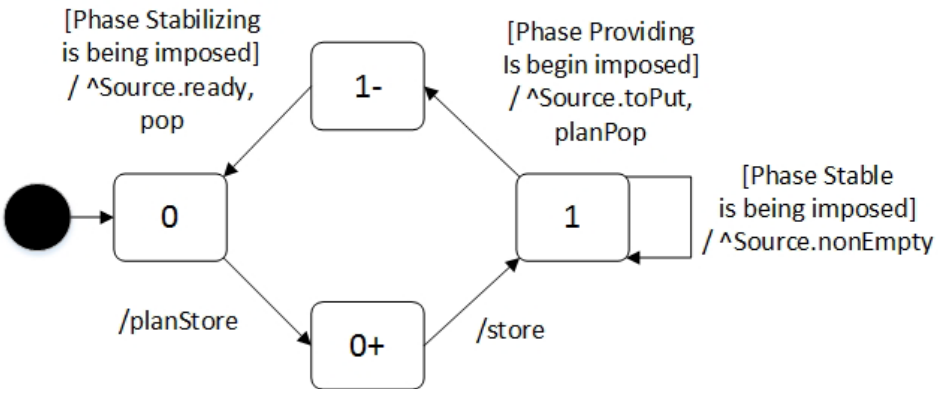
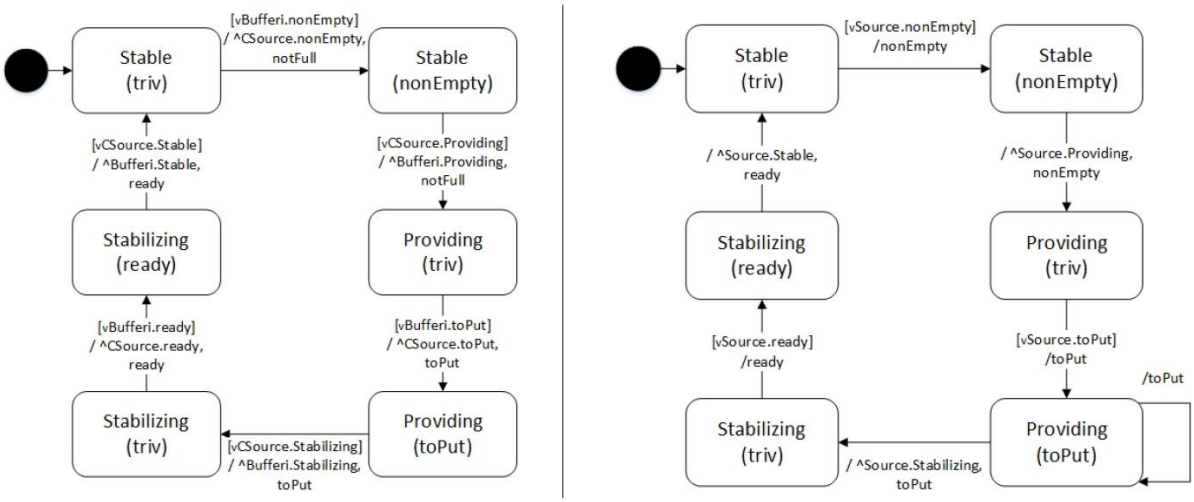


Рисунок 2.18 - Діаграма станів з умовами, відправками та діями

Деталізована STD компонента Bufferi у контексті ролі Source представлена у вигляді Діаграми станів UML на рисунку 2.18. Ця діаграма представляє специфікацію переходів буфера, релевантних для операцій видалення елементів. Вона включає стани, що відображають місткість буфера (0..n) та проміжні стани (0+,1− згідно з рисунком 1.2), а також переходи, що відображають процес видалення (наприклад, перехід зі стану 1 до стану 0 за дії pop). Переходи на цій Діаграмі станів включають умови, що ґрунтуються на нав'язаних фазах ролі Source ([Phase Stable is imposed], [Phase Providing is imposed], [Phase Stabilizing is imposed]), та дії відправки, що сигналізують про вхід у пастки, асоційовані з цими фазами та діями видалення (наприклад, ^Source.nonEmpty при активації фази Stable, ^Source.toPut при активації фази Providing, ^Source.ready при активації фази Stabilizing). Переходи, пов'язані з діями planStore та store (які стосуються ролі Sink), також присутні на цій Діаграмі станів компонента Buffer, але їх виконання не обмежене умовами, що базуються на фазах ролі Source.

Комунікаційний цикл для ролі Source компонента Buffer (аналогічно описаному для ролі Sink) та відповідні зміни станів діаграм станів (рисунок 2.18 для STD Buffer та рисунок 2.19 для портів Source та CSource) детально ілюструють, як відбувається зміна фаз ролі Source. Наприклад, коли нав'язана фаза Stable ([Phase Stable is imposed]) істинна і буфер містить елементи (тобто його місткість >0), може бути виконана дія відправки ^Source.nonEmpty.



а) Рольовий порт компонента Source б) Рольовий порт співпраці CSource

Рисунок 2.19 - Рольові порти та уточнена роль

Це повідомлення приймається діаграмою станів порту Source (рисунок 19 а), яка взаємодіє з діаграмою станів порту CSource (рисунок 2.19 б). Після обробки на рівні співпраці CSource, порт CSource надсилає повідомлення про зміну фази до порту Source (наприклад, ^Source.Providing). Порт Source приймає це повідомлення, змінює свій стан (наприклад, на Providing(triv)) і надсилає відповідне повідомлення до STD Buffer і (рисунок 2.18), нав'язуючи йому нову фазу Providing. Аналогічні послідовності повідомлень та змін станів відбуваються при переходах до фази Stabilizing та поверненні до Stable.

Як і у випадку з компонентом Filter, компонент Bufferi у моделі Paradigm (рисунок 1.2) інтегрує поведінку, необхідну як для ролі приймача (Sink), так і для ролі джерела (Source). Його деталізована STD включає всі дії, що використовуються обома ролями (planStore, store, planPop, pop). Відповідно, діаграма станів UML для Bufferi, представлена на рисунку 2.20, є об'єднанням переходів, умов та дій відправки з діаграм станів, орієнтованих окремо на ролі Sink (рисунок 2.14) та Source (рисунок 2.18).

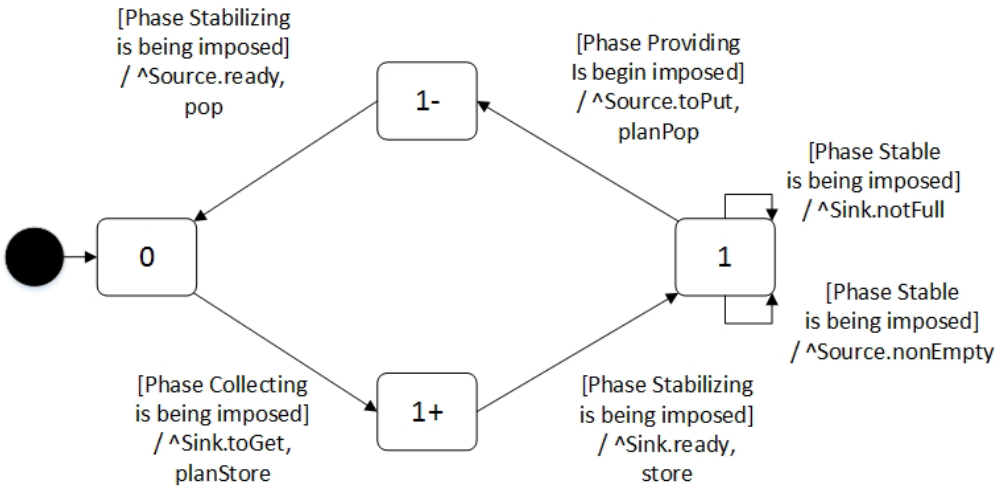


Рисунок 2.20 - Діаграма станів з умовами, відправками та діями

На цій об'єднаній діаграмі станів Bufferi візуалізовано переходи, які можуть бути активовані в залежності від нав'язаної фази для ролі Sink (з умовами типу [Phase Collecting is imposed]) або для ролі Source (з умовами типу [Phase Providing is imposed]), а також дії відправки, адресовані як порту Sink (^Sink.message), так і порту Source (^Source.message). Об'єднання поведінки різних ролей на одній Діаграмі станів компонента може призводити до ситуацій, коли з одного стану виходять кілька переходів, умови яких можуть перетинатися. У таких випадках для точного моделювання вибору між можливими переходами в UML можуть використовуватися точки вибору (choice pseudostates - візуалізуються як

ромб) або взаємовиключні умови (guards) на переходах. На Рисунку 31, наприклад, між станами 1+ та 1 візуалізовано вибір, що ілюструє ситуацію, коли рішення про подальший перехід залежить від виконання певних умов, пов'язаних з нав'язаними фазами (наприклад, [Phase Stabilizing is imposed] або [Phase Stable is imposed]), гарантуючи, що лише один перехід буде здійснено в певний момент часу. Таким чином, Діаграма станів Buffer і у UML відображає повну локальну поведінку компонента, що інтегрує функціональність обох ролей, визначену у відповідних розділах Paradigm.

РОЗДІЛ 3. ПРЕДСТАВЛЕННЯ МЕТОДОЛОГІЇ ІНТЕРПРЕТАЦІЇ ПРОГРАМНИХ ПАРАДИГМ НА ОСНОВІ ЗАСОБІВ UML

3.1. Процес трансформації структурних аспектів програмних парадигм та розгортання у представлення засобами UML

Цей розділ присвячений трансформації структурних та деплойментних (розгортання) аспектів моделей, розроблених у формалізмі мови координації Paradigm, у відповідні діаграми уніфікованої мови моделювання (UML) версії.

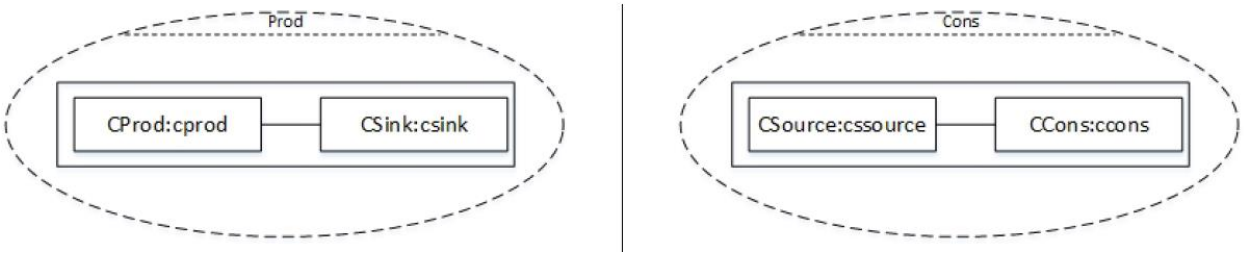
3.1.1. Трансформація структурних представлень

У Paradigm для візуалізації архітектури системи часто використовуються діаграми компонентної структури та діаграми співпраці. Paradigm-специфічна діаграма компонентної структури (рисунок 1.7) надає високоабстрактний огляд структури моделі, демонструючи компоненти та їхні зв'язки. Ці структурні представлення Paradigm трансформуються у відповідні діаграми компонентної структури UML (рисунок 3.1) та діаграми співпраці UML (рисунок 3.2), що відображають архітектуру конвеєра з використанням стандартизованої нотації.

Рисунок 3.1 - Архітектура конвеєра як діаграма компонентної структури

На діаграмі компонентної структури UML (рисунок 3.1) компоненти фільтрів та буферів представлені з їхніми портами (Prod, Cons для фільтрів; Sink, Source для буферів). Порти визначають точки взаємодії компонента із зовнішнім середовищем або іншими компонентами. Зв'язки між портами

різних компонентів моделюються за допомогою конекторів (connectors), що візуалізують асоціації або канали комунікації, необхідні для використання функціональності інших портів. Рольові порти Prod та Cons компонента фільтра призначені для забезпечення його участі у взаємодіях, пов'язаних відповідно з "виробництвом" та "споживанням" елементів. Рольові порти Sink та Source компонента буфера відповідають за обробку операцій збереження та видалення елементів, ініційованих фільтрами.



а) Співпраця Prod б) Співпраця Cons

Рисунок 3.2 - Співпраці Prod та Cons

Діаграми співпраці UML (рисунок 3.2) використовуються для визначення специфічних патернів взаємодії (співпраць) між набором ролей. Рисунок 3.2а візуалізує співпрацю Prod, а рисунок 3.2 б — співпрацю Cons. Кожна співпраця складається з набору ролей (roles), що виконуються певними класифікаторами під час виконання. Учасники співпраці позначаються як назва_ролі : Тип_класифікатора. Співпраця Prod включає роль CProd (типу cprod) та роль CSink (типу csink), моделюючи взаємодію між виробником та приймачем. Співпраця Cons складається з ролі CSource (типу csource) та ролі CCons (типу ccons), моделюючи взаємодію між джерелом та споживачем.

Інтеграція цих рівнів моделювання показана на Діаграмі компонентної структури з прив'язкою співпраць (рисунок 3.3). Ця діаграма ілюструє, як екземпляри співпраць (наприклад, екземпляр співпраці Prod) пов'язуються з

портами компонентів, які беруть участь у цій співпраці, використовуючи зв'язувачі ролей (role bindings). Наприклад, порт Prod компонента Filteri пов'язаний з роллю CProd у співпраці Prod. Порт Sink компонента Bufferi пов'язаний з роллю CSink також у співпраці Prod. Водночас, порт Source компонента Bufferi пов'язаний з роллю CSource у співпраці Cons, а порт Cons компонента Filteri+1 — з роллю CCons у співпраці Cons. Ці прив'язки визначають, які саме екз

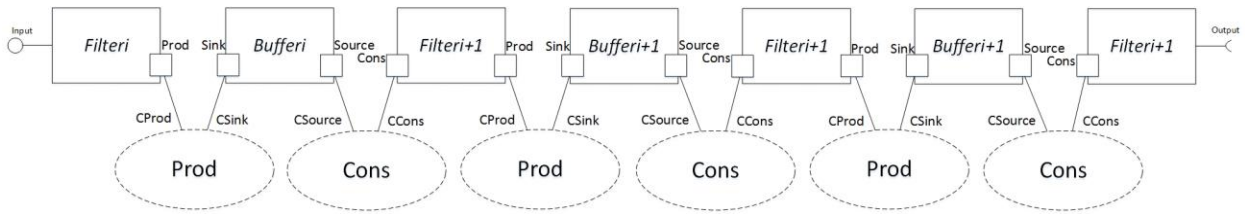


Рисунок 3.3 - Архітектура конвеєра в шести діаграмах співпраці

3.1.2. Трансформація у діаграму компонентів UML

Діаграма компонентів UML (рисунок 3.4) надає огляд фізичних або імплементаційних компонентів системи та їхніх залежностей. У даній моделі, компонент Filter і та компонент Buffer і представлені як композитні компоненти, що включають внутрішні частини (parts), які відповідають їхнім ролям (Prod та Cons для фільтра; Sink та Source для буфера). Співпраці Prod та Cons також представлені як компоненти, що містять частини, які відповідають їхнім ролям (CProd, CSink для співпраці Prod; CSource, CCons для співпраці Cons). Таке представлення ілюструє, які частини належать до певних компонентів або співпраць.

На цій діаграмі також візуалізовано інтерфейси, що надаються та вимагаються компонентами та їхніми частинами, використовуючи нотацію "куля-розетка" (provided/required interfaces). На основі об'єднаних Діаграм станів (рисунки 2.11 та 2.20), які специфікують поведінку компонентів з урахуванням їхніх ролей, компоненти Filter і та Buffer і вимагають або

надають відповідні інтерфейси через свої порти. Діаграма компонентів (рисунок 3.4) показує, як ці інтерфейси пов'язуються між собою. Наприклад, наданий інтерфейс на порту Prod компонента Filter і з'єднаний з необхідним інтерфейсом на відповідному порту компонента співпраці Prod, використовуючи зв'язувач. Делегуючі конектори (delegate connectors) можуть використовуватися для зв'язування зовнішніх портів композитного компонента з портами його внутрішніх частин, наприклад, зовнішнього порту Prod компонента Filter і з портом внутрішньої частини, що відповідає ролі Prod.

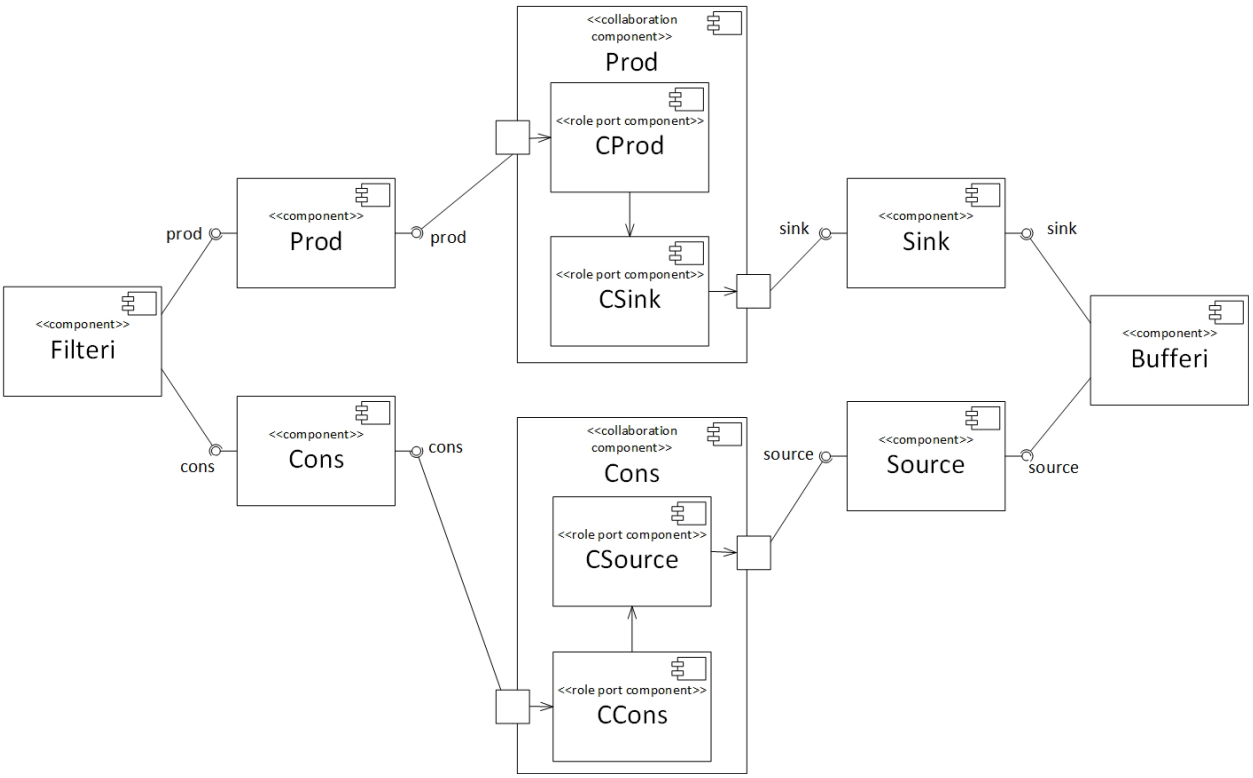


Рисунок 3.4 - Діаграма компонентів

Діаграма компонентів (рисунок 3.4) також відображає гнучкість архітектури конвеєра, де компоненти фільтрів та буферів можуть використовувати різні набори внутрішніх частин/ролей залежно від їхньої позиції. Наприклад, перший фільтр (Filter1) може використовувати лише

частину, що відповідає ролі Prod, тоді як проміжні фільтри (Filter2,Filter3) використовують обидві частини (Prod і Cons), а останній фільтр (Filter4) — лише частину Cons. Це демонструє, що конфігурація компонента може бути адаптована залежно від контексту його використання в архітектурі.

3.1.3. Трансформація у діаграму класів UML

Діаграма класів UML (рисунок 3.5) представляє логічну структуру моделі та визначає класифікатори та їхні взаємозв'язки.

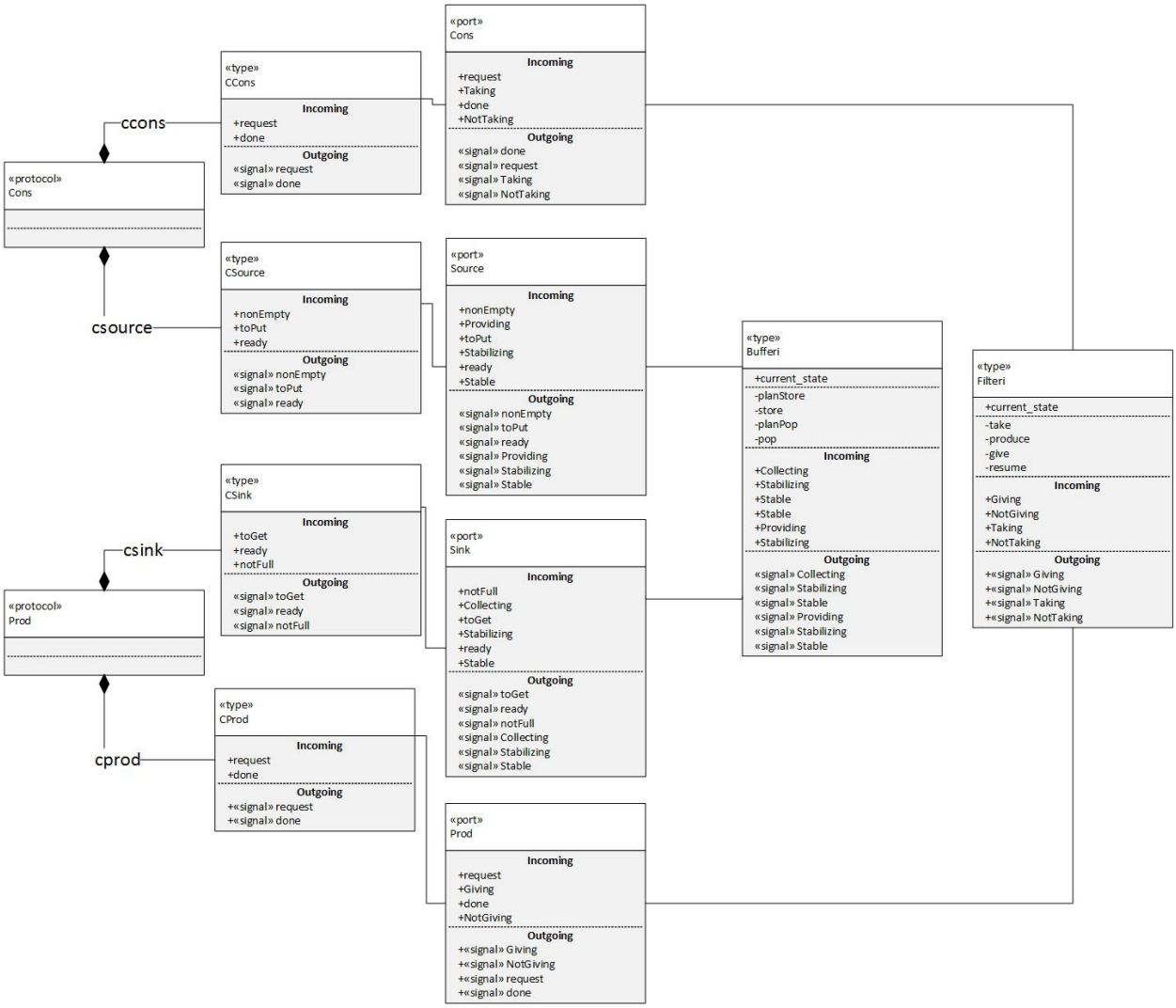


Рисунок 3.5 - Діаграма класів

Ця діаграма побудована на основі інформації з Діаграм компонентної структури, Діаграм співпраці та Діаграм станів. На Діаграмі класів

представлені класи, що відповідають компонентам (Filter і, Buffer і), ролям, портам та співпрацям. Використовуються стереотипи (stereotypes) для уточнення семантики цих класифікаторів. Наприклад, стереотип «protocol» застосовується до класів, що моделюють протоколи співпраці (наприклад, класи, що представляють співпраці Prod та Cons). Стереотипи «type» та «port» можуть застосовуватися до класів, що представляють типи елементів та їхні порти відповідно.

Класи Filter і та Buffer і асоційовані зі своїми порт-класами (Prod, Cons, Sink, Source). Класи співпраць (наприклад, протокол Prod) можуть бути пов'язані з класами, що представляють учасників співпраці (CProd, CSink), можливо, за допомогою композиції. Відповідно, специфікація класу-протоколу співпраці не містить специфікації внутрішніх вхідних чи вихідних дій, фокусуючись на зовнішній поведінці та взаємодії ролей.

Діаграма класів також відображає вхідні та вихідні сигнали (signals), що використовуються для комунікації між елементами, як атрибути або операції класів [11]. Наприклад, клас Filter і може мати атрибути або операції, що відповідають сигналам Giving, NotGiving (для ролі Prod) та Taking, NotTaking (для ролі Cons). Рольові порти (класи Prod, Cons, Sink, Source) також мають визначені вхідні та вихідні сигнали, що відповідають повідомленням, які вони можуть приймати або відправляти (наприклад, сигнали request, done для порту Prod). При об'єднанні ролей на рівні класу компонента (наприклад, клас Buffer і для ролей Sink та Source), можуть виникати ситуації, коли сигнали з однаковими назвами, але різною семантикою (належністю до різних ролей), визначені в одному класі.

3.1.4. Трансформація у діаграму пакетів UML

Діаграма пакетів UML (рисунок 3.6) використовується для організації елементів моделі у логічні групи (пакети) та візуалізації залежностей між ними. На цій діаграмі представлені пакети верхнього рівня для компонентів

(Filter i, Buffer i) та співпраць (Prod Collaboration, Cons Collaboration). Вкладені пакети використовуються для подальшої структуризації, наприклад, пакети Prod та Cons всередині пакету Filter i, або пакети CProd та CSink всередині пакету співпраці Prod.

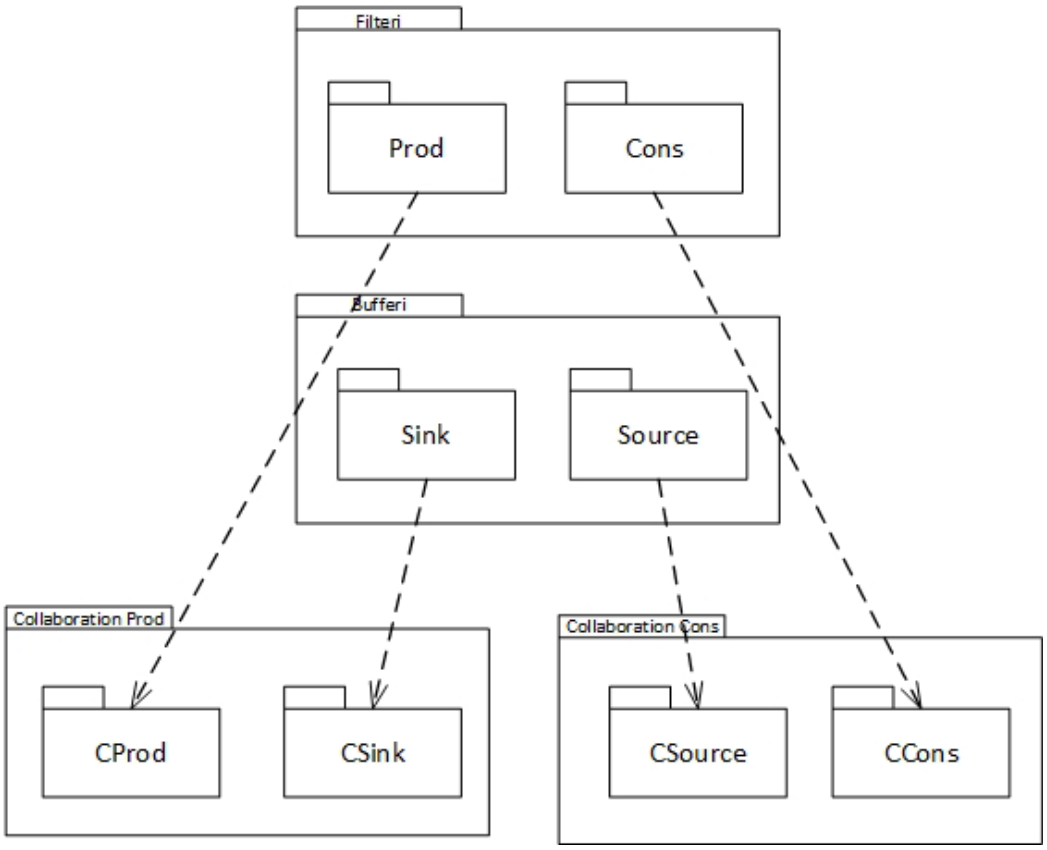


Рисунок 3.6 - Діаграма пакетів

Залежності (dependencies) між пакетами ілюструють, що елементи в одному пакеті залежать від елементів в іншому. Наприклад, залежність від вкладеного пакету CProd (в пакеті співпраці Prod) до вкладеного пакету Prod (в пакеті Filter i) може означати, що елементи в пакеті Prod Filter i використовують або посилаються на елементи, визначені в пакеті CProd. Така організація пакетів відображає, де розташована поведінка (в пакетах компонентів, що містять STD/Діаграми станів) та які елементи (наприклад, учасники співпраці) є необхідними для реалізації цієї поведінки та взаємодії.

3.1.5. Трансформація у діаграму розгортання UML

Діаграма розгортання UML (рисунок 3.7) візуалізує фізичне розміщення артефактів програмного забезпечення на вузлах (nodes) та пристроях (devices) виконавчого середовища. У контексті даної трансформації, вузол може представляти екземпляр виконавчого середовища (наприклад, процес або потік), на якому виконується відповідна STD компонента (Filter і або Buffer і). Пристрій може представляти фізичний апаратний ресурс або, як у даному випадку, логічне середовище, що забезпечує функціонування співпраць.

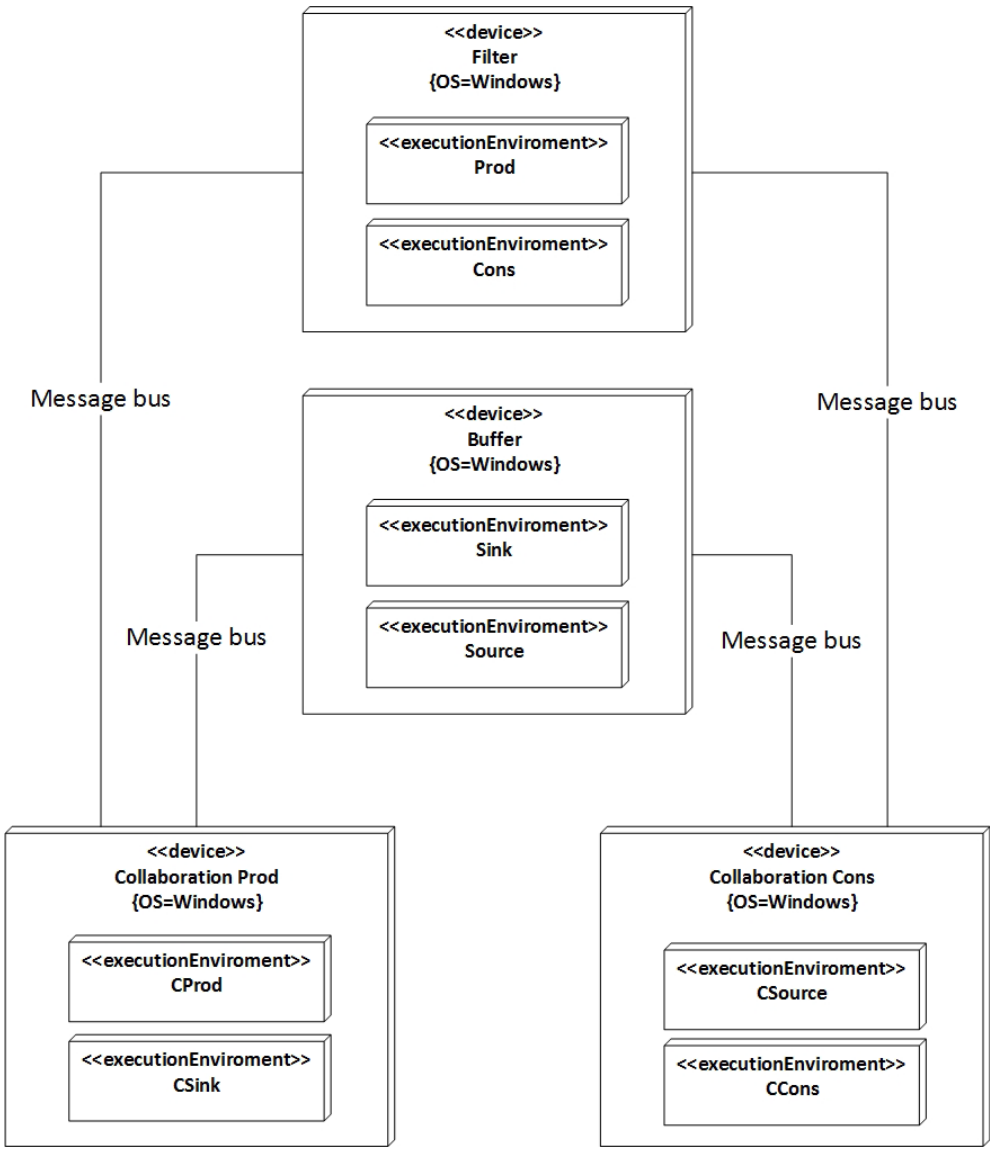


Рисунок 3.7 - Діаграма розгортання

На діаграмі розгортання (рисунок 3.7) вузли компонентів Filter і та Buffer і з'єднані з вузлами, що представляють середовище виконання співпраць (Prod Collaboration Device Node, Cons Collaboration Device Node), за допомогою комунікаційних шляхів (communication paths), що моделюють шини повідомлень. Це відображає архітектуру, де кожен компонент (STD) може виконуватися як окрема одиниця, а комунікація та координація між ними здійснюється через спеціалізовані середовища співпраці, які забезпечують передачу повідомлень через шини. Таке представлення ілюструє, як логічна структура моделі Paradigm відображається на фізичну інфраструктуру розгортання.

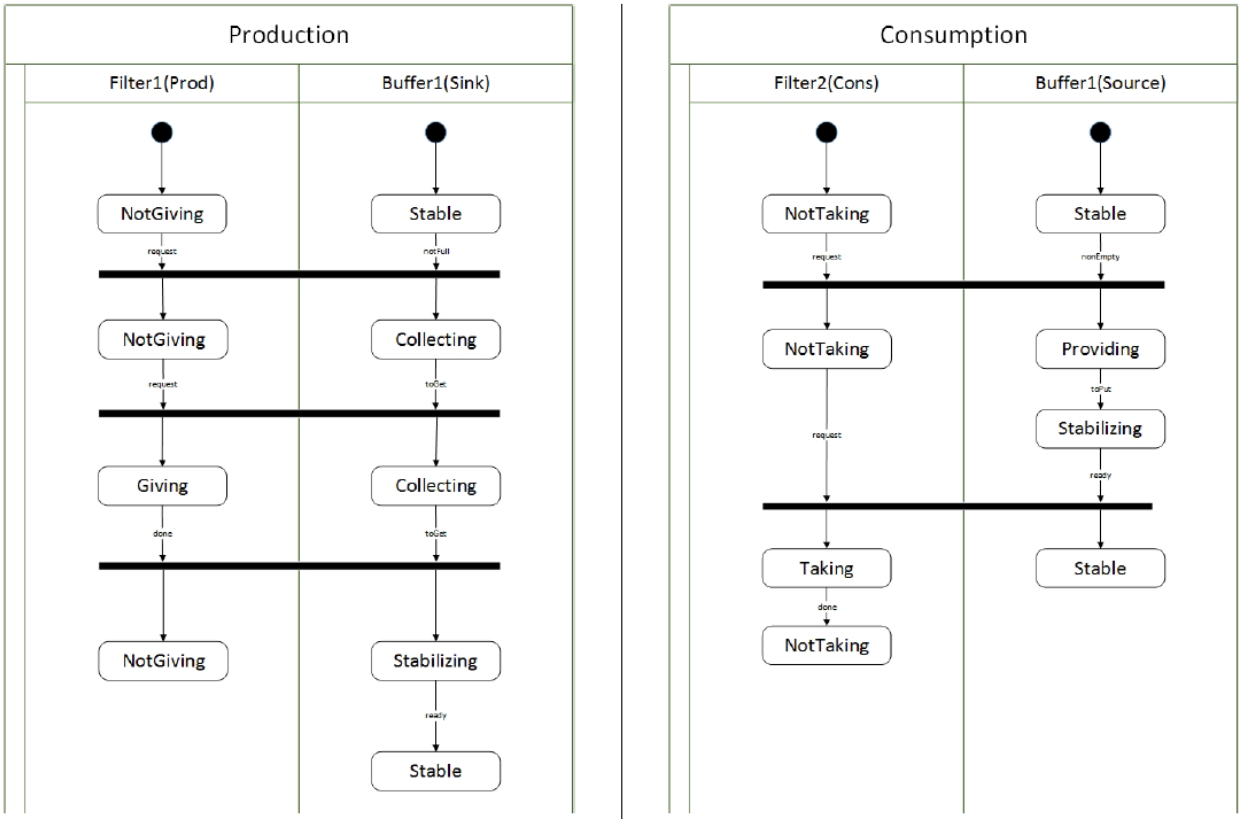
3.2. Трансформація поведінкових аспектів та потоків взаємодії програмних парадигм у представлення засобами UML

Цей підрозділ деталізує трансформацію поведінкових аспектів моделей компонентів та потоків взаємодії, розроблених у формалізмі Paradigm, у відповідні діаграми активності, діаграми огляду взаємодії, діаграми варіантів використання та діаграми часу уніфікованої мови моделювання (UML).

3.2.1. Трансформація у діаграми активності UML

Діаграми активності UML використовуються для моделювання потоку керування та даних, візуалізації бізнес-процесів або, в даному контексті, виконання координаційної логіки, визначеної правилами узгодженості. Рисунок 3.8 ілюструє високоабстрактні Діаграми активності, що моделюють процеси, асоційовані зі співпрацями Production (для взаємодії Filter Prod - Buffer Sink) та Consumption (для взаємодії Buffer Source - Filter Cons). Рисунок 3.8а базується на правилах узгодженості 1-4 (процес виробництва), а рисунок 3.8 б — на правилах 5-8 (процес споживання). Дії на цих діаграмах можуть представляти фази або ключові стадії взаємодії. Варто відзначити,

що повторювані назви дій (наприклад, NotGiving у смузі Filter i(Prod)) вказують на багаторазове входження в цю стадію в процесі виконання, а не на різні види дій.



а) Діаграма активності Prod

б) Діаграма активності Cons

Рисунок 3.8 - Діаграми активності правил узгодженості виробництва та споживанн

Деталізована діаграма активності (рисунок 3.9) надає більш детальне представлення потоку, інтегруючи учасників комунікації та логіку синхронізації. Діаграма використовує смуги плавання (swimlanes) для візуалізації діяльності різних учасників (наприклад, Filter i, Prod, CProd, Buffer i, Sink, CSink). Центральні смуги плавання представляють рольові порти співпраці (CProd, CSink), як це було показано на рисунку 3.8. Початковий вузол Діаграми активності може відповідати початковому стану

певної фази (наприклад, NotGiving(triv) для Filter, Stable(triv) для Buffer), як це відповідає рольовим поведінкам на рисунках 1.3б та 1.5б.

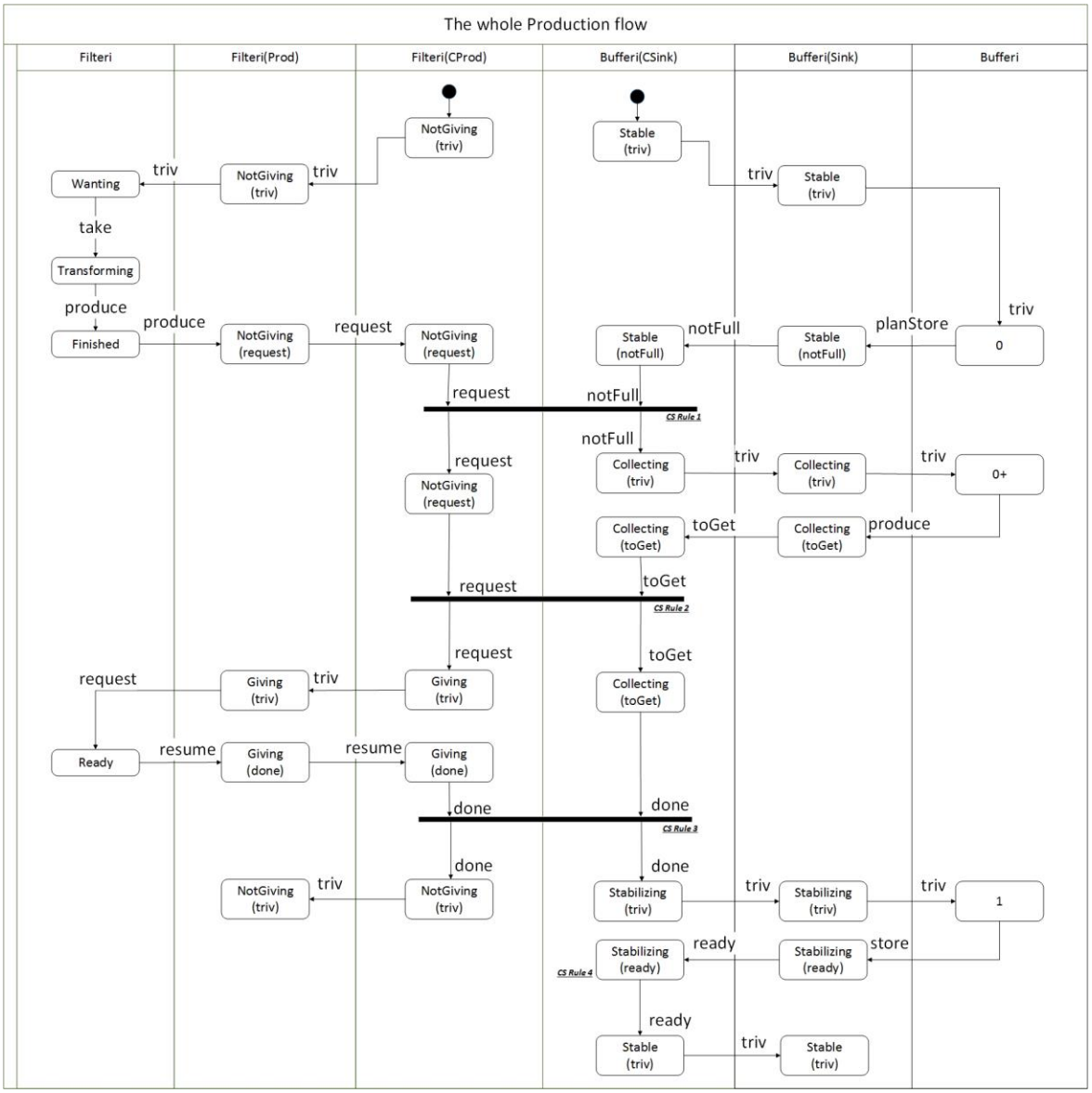


Рисунок 3.9 - Уточнена діаграма активності

Діаграма активності (рисунок 3.9) візуалізує взаємодію між компонентами, що реалізується через їхні рольові порти та співпраці. Вузли розгалуження (fork nodes) використовуються для представлення паралельної активації дій, а вузли злиття (join nodes) або вузли рішень/об'єднання (decision/merge nodes) - для синхронізації або вибору потоків. Наприклад, вузол розгалуження з приміткою "CS Rule 1" (правило узгодженості 1)

ілюструє, як активація цього правила синхронізує паралельні дії: вхід у пастку request у стані NotGiving для ролі Filter i(Prod) та вхід у пастку notFull у стані Stable для ролі Buffer i(Sink). Це означає, що перехід до наступних станів (NotGiving(request) та Collecting(triv)) відбувається лише після одночасної готовності обох учасників згідно з Правилом 1. Таким чином, Діаграма активності візуалізує, як правила узгодженості керують паралельним виконанням кроків комунікації та змін станів у різних компонентах та їхніх ролях, що є важливим для розуміння координації. Дії на цій діаграмі можуть відповідати входу в пастки (наприклад, NotGiving(request)), станам портів (наприклад, NotGiving(triv)) або крокам комунікації (наприклад, надсилання/прийом повідомлення). Діаграма активності також може включати візуалізацію дій triv, що представляють стани очікування або прості переходи без специфічних дій. Аналогічна Діаграма активності може бути побудована для процесу споживання (взаємодія Buffer Source - Filter Cons), що матиме схожу структуру, але використовуватиме відповідні дії та стани/фази ролей Source та Cons.

3.2.2. Трансформація у діаграму огляду взаємодії UML

Діаграма огляду взаємодії UML (рисунок 3.10) надає високоабстрактне представлення потоку взаємодії у системі, комбінуючи елементи Діаграм активності (керуючий потік) та Діаграм взаємодії (посилання на послідовності повідомлень). Діаграма візуалізує чотири Діаграми послідовності (позначені як sd): sd prod та sd sink для процесу виробництва, sd cons та sd source для процесу споживання.

Процес починається з початкового вузла, за яким слідує вузол рішення, що спрямовує потік до шляху Production або Consumption. Якщо обрано шлях Production, то Діаграми послідовності, що моделюють взаємодію ролей Prod (sd prod) та Sink (sd sink), виконуються паралельно, що ілюструється використанням вузлів розгалуження та злиття.

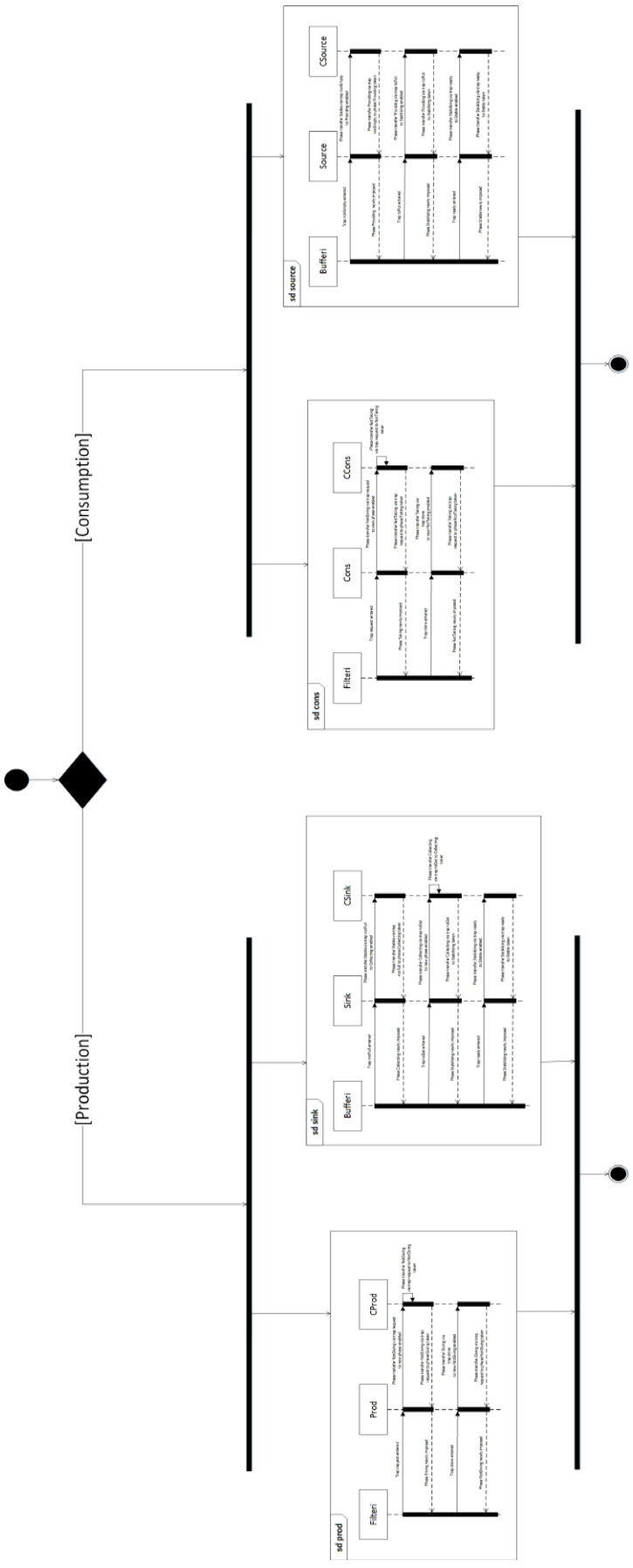


Рисунок 3.10 - Діаграма огляду взаємодії

Це відображає той факт, що виробництво елемента фільтром та його прийом буфером включають паралельні аспекти взаємодії. Завершення циклу виробництва відбувається лише після завершення обох паралельних послідовностей взаємодії. Аналогічно, для шляху Consumption, Діаграми послідовності sd cons та sd source виконуються паралельно. Діаграма огляду взаємодії слугує для моделювання взаємодії на високому рівні, показуючи залежності та паралельність між складнішими послідовностями взаємодії, деталізованими в інших діаграмах.

3.2.2. Трансформація у діаграму часу UML

Діаграма часу UML (рисунок 3.11) використовується для візуалізації змін станів класифікаторів (представлених лініями життя) протягом певного лінійного часового проміжку.

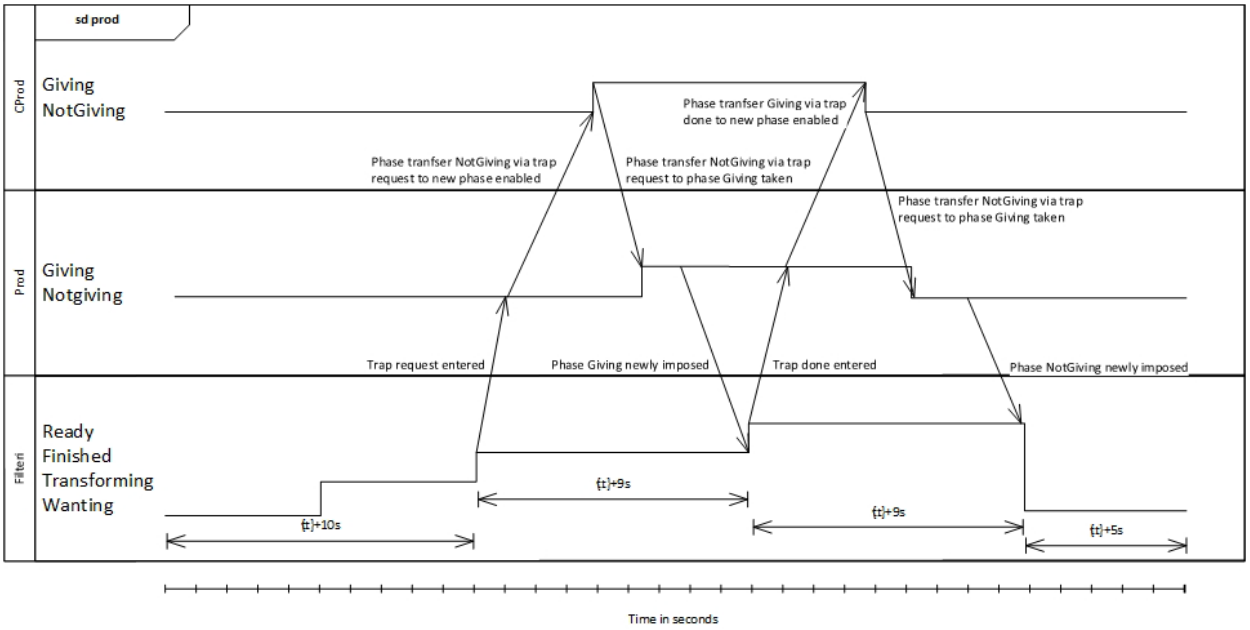


Рисунок 3.11 - Діаграма часу UML

Діаграма базується на інформації про послідовність повідомлень та зміни станів, отриманій з Діаграм комунікацій та Послідовності. Кожна горизонтальна лінія життя на Діаграмі часу представляє учасника

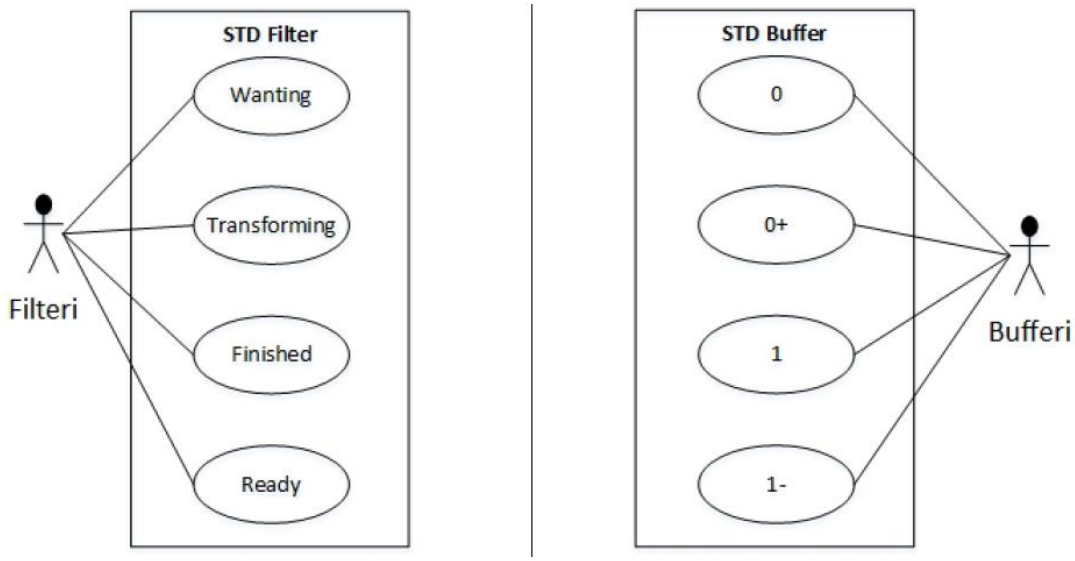
(наприклад, Filter i, рольовий порт Prod, рольовий порт співпраці CProd). Горизонтальні сегменти на лінії життя відображають проміжки часу, протягом яких учасник перебуває в певному стані (наприклад, стани Filter i: Ready, Finished, Transforming, Wanting з рисунка 1.1; стани ролей Prod/CProd: Giving, NotGiving з рисунка 1.3б). Вертикальні лінії або маркування позначають моменти зміни станів.

Діаграма часу ілюструє тривалість перебування учасників у певних станах та моменти, коли події (наприклад, отримання повідомлення) спричиняють зміни станів. Вона також наочно демонструє затримки, пов'язані з асинхронною комунікацією. Наприклад, затримка між відправленням повідомлення від порту Prod та його прийомом портом CProd візуалізується як часовий проміжок між відповідними подіями на лініях життя Prod та CProd. Тривалість станів та затримки на цій діаграмі є ілюстративними припущеннями, оскільки формалізм Paradigm не визначає часових характеристик виконання. Зміни станів на Діаграмі часу узгоджуються з логікою переходів, визначеною в Діаграмах станів та ініційованою повідомленнями з Діаграм взаємодії. Аналогічні Діаграми часу можуть бути побудовані для інших взаємодій конвеєра (Filter Cons, Buffer Sink, Buffer Source).

3.2.3. Трансформація у діаграму варіантів використання UML

Діаграма варіантів використання UML (рисунок 3.12) традиційно використовується для опису функціональних вимог до системи з точки зору акторів (користувачів або інших систем), що взаємодіють з системою для досягнення певних цілей. В даному випадку застосування діаграми варіантів використання для моделювання архітектури конвеєра є дещо нетрадиційним. Межа системи візуалізована як STD компонента (STD Filter, STD Buffer), а акторами виступають самі екземпляри компонентів (Filteri, Bufferi). Варіанти

використання (use cases) відповідають станам, визначеним у STD Filter (Wanting, Transforming, Finished, Ready) та STD Buffer (0, 0+, 1, 1-).



а) STD Filter б) STD Buffer

Рисунок 3.12 - Діаграма варіантів використання

Таке нетрадиційне відображення використовує Діаграму варіантів використання для представлення можливостей або функціональних станів, які кожен компонент (розглянутий як система) надає, та акторів (інших компонентів), які потенційно можуть "використовувати" або взаємодіяти з цими станами. Хоча це відхилення від стандартного застосування діаграми (яка зазвичай фокусується на зовнішній взаємодії з користувачем), воно дозволяє візуалізувати набір дискретних "функціональних станів", у яких може перебувати компонент.

Кожен "варіант використання" (стан) на діаграмі варіантів використання супроводжується детальним описом (таблиці 3.1 – 3.8). Ці описи варіантів використання уточнюють семантику кожного стану, включаючи його мету в контексті системи, передумови для досягнення стану, успішні та неуспішні кінцеві умови, основних та другорядних акторів, а

також тригер, що ініціює перехід до стану. Описи також включають основний потік дій, що відбуваються у цьому стані.

Таке деталізоване текстове пояснення доповнює візуальне представлення станів на діаграмі варіантів використання, надаючи повне розуміння функціонального значення кожного стану компонента в рамках моделі конвеєра.

Таблиця 3.1 - Опис випадку використання

Use case name		**Wanting**
Goal in context		The filter wants a new item
Preconditions		The filter has made a resume
Successful end condition		There is a item available from elsewhere
Failed end condition		There is no item available from elsewhere
Primary actors		Filter
Secondary actors		None
Trigger		Input in the filter
Main flow	**Step**	**Action**
	1	Looking for new input from elsewhere

Таблиця 3.2 - Опис випадку використання

Use case name		**Transforming**
Goal in context		The filter transforms an item
Preconditions		Gets the input in the form of a new item
Successful end condition		Transforms an item to a new item
Failed end condition		Does not transforms an item to a new item
Primary actors		Filter
Secondary actors		None
Trigger		Input to the filter
Main flow	**Step**	**Action**
	1	Transforms the gotten item in new item

Таблиця 3.3 - Опис випадку використання

Use case name		**Finished**
Goal in context		A new item is finished
Preconditions		Made available for being put elsewhere
Successful end condition		The filter shows availability
Failed end condition		The filter shows no availability
Primary actors		Filter
Secondary actors		None
Trigger		Input in the filter
Main flow	**Step**	**Action**
	1	Indicates availability

Таблиця 3.4 - Опис випадку використання

Use case name		**Ready**
Goal in context		A new item is ready
Preconditions		Puts new item to elsewhere
Successful end condition		Item has been produced or consumed
Failed end condition		Item has not been produced or consumed
Primary actors		Filter
Secondary actors		None
Trigger		Input in the filter
Main flow	**Step**	**Action**
	1	Filter done with producing and consuming

Таблиця 3.5 - Опис випадку використання

Use case name		**0**
Goal in context		The buffer is empty
Preconditions		There is no must be an item planned to be popped out
Successful end condition		The buffer is empty
Failed end condition		The buffer has an item
Primary actors		Buffer
Secondary actors		None
Trigger		Item from the filter
Main flow	**Step**	**Action**
	1	Buffer has popped an item

Таблиця 3.6 - Опис випадку використання

Use case name		**0+**
Goal in context		An item is planned to store in the buffer
Preconditions		Input from the filter
Successful end condition		The filter has made an item to be available in the buffer
Failed end condition		The filter has not made an item to be available in the buffer
Primary actors		Buffer
Secondary actors		None
Trigger		Item from the filter
Main flow	**Step**	**Action**
	1	Buffer is planning to store an item

Таблиця 3.7 - Опис випадку використання

Use case name		**1**
Goal in context		An item is stored in the buffer
Preconditions		There must be store planned
Successful end condition		An item has been stored in the buffer
Failed end condition		No item has been stored in the buffer
Primary actors		Buffer
Secondary actors		None
Trigger		Produce item from the filter
Main flow	**Step**	**Action**
	1	An item is stored in the buffer

Таблиця 3.8 - Опис випадку використання

Use case name		**1-**
Goal in context		An item is planned to be popped out of the buffer
Preconditions		There must be at least one item in the buffer
Successful end condition		The buffer has planned to pop out the item
Failed end condition		The buffer didn't planned to pop out the item
Primary actors		Buffer
Secondary actors		None
Trigger		Consume item from filter
Main flow	**Step**	**Action**
	1	An item is planned to be popped out

Отже, проведене дослідження було спрямоване на вивчення та демонстрацію можливостей трансформації моделей мови координації Paradigm у представлення засобами уніфікованої мови моделювання UML. Різноманітність підходів до перекладу, застосованих до моделі конвеєра з фільтром та буфером, та візуалізація різних аспектів цієї моделі за допомогою тринадцяти типів діаграм UML переконливо демонструють принципову можливість здійснення такого перекладу.

Мова моделювання Paradigm, хоча на перший погляд і здається простою завдяки використанню єдиного типу діаграм (діаграми станів - STD), є достатньо виразною для специфікації складних координаційних взаємодій. Її компактність полягає в тому, що єдина діаграма STD, доповнена концепціями фаз, пасток, ролей та правил узгодженості, дозволяє охопити аспекти, для моделювання яких в UML вимагається використання множини різних діаграм. Це дає підстави вважати Paradigm просунутим формалізмом з точки зору виразної сили на одиницю нотаційного елемента. Водночас, ключова перевага UML полягає у його широкій стандартизації та поширеності у галузі розробки програмного забезпечення. Трансформація моделей Paradigm у представлення засобами UML робить їх доступними для розуміння широким колом фахівців, знайомих з UML, та дозволяє використовувати багатий арсенал існуючих інструментів моделювання, аналізу та верифікації. Таке поєднання виразних можливостей Paradigm та екосистеми UML може створити потужний інструмент для специфікації та аналізу архітектур складних систем.

В ході дослідження було виявлено, що різні типи діаграм UML мають різну ефективність та корисність для представлення концепцій Paradigm:

- Діаграми станів UML є найбільш природним відповідником STD Paradigm та ефективно моделюють локальну поведінку компонентів та їхніх рольових портів.

- Діаграми компонентної структури, діаграми співпраці та діаграми композитної структури успішно відображають структурні аспекти архітектури Paradigm, включаючи компоненти, порти, ролі та зв'язування співпраць.

- Діаграми активності UML виявилися особливо ефективними для візуалізації виконання координаційної логіки та правил узгодженості. Вони дозволяють інтегрувати в одній діаграмі елементи поведінки компонентів та потоки комунікації між ролями, наочно представляючи синхронізацію, обумовлену правилами узгодженості.

- Діаграми комунікацій та діаграми послідовності деталізують послідовність обміну повідомленнями, що становить вертикальну комунікацію, надаючи альтернативну візуалізацію потоків взаємодії.

- Діаграми класів ефективно моделюють логічну структуру системи, класифікуючи елементи, що відповідають компонентам, ролям, портам та сигналам/подіям, і відображаючи взаємозв'язки між ними.

- Діаграми пакетів та діаграми розгортання надають цінні високоабстрактні представлення організаційної структури моделі та фізичного розміщення компонентів, доповнюючи більш детальні діаграми.

- Діаграми огляду взаємодії та діаграми варіантів використання, хоча і можуть бути побудовані на основі моделей Paradigm, виявилися менш інформативними або нетрадиційними для безпосереднього представлення сутності координаційних механізмів Paradigm порівняно з іншими типами діаграм UML.

Ключовим обмеженням є відсутність у стандартній нотації та семантиці UML безпосередньої підтримки для моделювання динамічної адаптації (самоадаптації) систем, що є важливою можливістю Paradigm, реалізованою за допомогою таких компонентів, як McPal. Стандартний UML не надає спеціалізованих діаграм чи елементів для візуалізації та

специфікації процесів динамічної реконфігурації архітектури під час виконання у такий спосіб, як це реалізовано в Paradigm.

Виявлене обмеження визначає перспективні напрямки для подальших досліджень. Зокрема, актуальним є дослідження можливостей моделювання компонента McPal та загалом механізмів самоадаптації засобами UML. Це може включати розробку розширень UML (наприклад, профілів або стереотипів) або специфічних патернів моделювання для представлення динамічної реконфігурації та процесів адаптації. Іншим напрямком є моделювання процесів динамічної зміни конфігурації архітектури (наприклад, додавання або видалення компонентів під час виконання) шляхом інтеграції моделей Paradigm (зокрема, з компонентом McPal) та розширених моделей UML. Таке дослідження дозволить розширити можливості моделювання адаптивних архітектур за допомогою стандартизованих засобів UML та інтегрувати динамічні аспекти, які є сильною стороною Paradigm, у більш широку екосистему моделювання UML.

ВИСНОВКИ

В дипломній роботі досліджено методологію інтерпретації програмних парадигм за допомогою засобів UML, що дозволило сформувати комплексний підхід до представлення структурних та поведінкових аспектів програмних систем. На основі аналізу предметної області було визначено ключові особливості мови моделювання Paradigm, її основні компоненти та механізми координації, що дозволило чітко ідентифікувати трансляційні зв’язки між концепціями Paradigm та відповідними елементами UML.

В результаті виконаних досліджень і трансформацій:

1. Було розроблено процедури відображення моделей компонентів Paradigm (зокрема Filter та Buffer) у відповідні діаграми UML, що забезпечує формальну сумісність між мовами.

2. Запропоновано підходи до трансформації елементів, таких як STD, ролі та аспекти координації, у діаграми станів UML, що дозволяє зберегти логіку поведінки системи під час переходу між мовами.

3. Детально розглянуто трансформації різних видів UML-діаграм — компонентів, класів, пакетів, розгортання, часових діаграм і діаграм варіантів використання — з урахуванням специфіки програмних парадигм, що демонструє універсальність UML як засобу моделювання.

4. Сформовано цілісну методологію, що дозволяє інтерпретувати програмні парадигми з урахуванням як структурної, так і поведінкової складової, забезпечуючи підтримку на етапах аналізу, проєктування та документації програмних систем.

Запропонована методологія дозволяє використовувати UML не лише як інструмент проєктування, але і як засіб уніфікованої інтерпретації різних програмних парадигм, що підвищує ефективність і зрозумілість процесу розробки складних програмних систем.

Отримані результати підтверджують доцільність та ефективність використання UML як універсального інструменту для інтерпретації та візуалізації концепцій різних програмних парадигм, що відкриває можливості для їх інтеграції у складні проєктні середовища.

Результати роботи можуть бути використані при розробці інструментів автоматичного моделювання та для покращення процесу проектування складних програмних систем.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Androva, A., Bureš, T., & Plášil, F. (2007). Formal foundation of the PARADIGM coordination language. Electronic Notes in Theoretical Computer Science, 177, 3-17.
2. Grunewegen, L., Androva, A., Bureš, T., & Plasil, F. (2008). Specifying and Executing Architectures with Dynamic Adaptation. In Component-Based Software Engineering (pp. 195-210). Springer Berlin Heidelberg.
3. Plášil, F., & Bureš, T. (2002). The SOFA/CDP Component Model. In Component-Based Software Engineering (pp. 88-105). Springer Berlin Heidelberg.
4. Bureš, T., Plášil, F., & Veselý, P. (2008). Modelling component behaviour and its coordination using state diagrams. International Journal of Parallel, Emergent and Distributed Systems, 23(2), 103-118.
5. Plášil, F., Bureš, T., Hnětynka, P., & Tůma, P. (2006). SOFA 2.0: Connecting Interconnectors. In Component-Based Software Engineering (pp. 36-51). Springer Berlin Heidelberg.
6. Androva, A., Bureš, T., & Plášil, F. (2009). Behavioural Specification of Coordination. In Coordination Models and Languages (pp. 18-35). Springer Berlin Heidelberg.
7. OMG. (2017). Unified Modeling Language (UML) Version 2.5.1. Object Management Group. (Офіційна специфікація UML)
8. Fowler, M. (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional.
9. Rumbaugh, J., Jacobson, I., & Booch, G. (2005). The Unified Modeling Language Reference Manual. Addison-Wesley Professional.
10. Kleppe, A., Warmer, J., & Bast, M. (2003). MDA Explained: The Model Driven Architecture™: Practice and Promise. Addison-Wesley Professional.

11. Czarnecki, K., & Helsen, V. (2006). Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3), 621-640.
12. Sendall, S., & Kozaczynski, W. (2003). Model transformation: the heart of model-driven development. IEEE Software, 20(5), 42-45.
13. Mens, T., & Van Gorp, P. (2006). A survey of model transformation languages. International Journal on Software Tools and Technology Transfer, 8(5), 523-542.
14. Küster, J. M., Ryndina, K., & Ehrig, K. (2007). Transformation of UML state machines to Petri nets. In International Conference on Model Driven Engineering Languages and Systems (pp. 178-192). Springer Berlin Heidelberg.
15. Van der Aalst, W. M. (1998). The application of Petri nets to workflow management. Journal of Systems and Software, 42(1), 1-21.
16. Wieringa, R. J. (2005). Requirements Engineering: Frameworks for Understanding. John Wiley & Sons.
17. Medvidovic, N., & Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering, 26(1), 70-93.
18. Shaw, M., & Garlan, D. (1996). Software architecture: Perspectives on an emerging discipline. Prentice Hall. (
19. Garlan, D., & Shaw, M. (1993). An introduction to software architecture. Advances in Software Engineering and Knowledge Engineering, 1, 1-39.
20. Wieringa, R. (2007). Design Methods for Reactive Systems: Net-Based Systems Design. Morgan Kaufmann.
21. Harel, D. (1987). Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8(3), 231-274.
22. Bokhari, F., & dign, M. (2013). A survey of coordination models and languages. International Journal of Distributed Sensor Networks, 9(6), 530259.

23. De Alfaro, L., & Venkatesh, S. (2001). Modeling Software Systems: The Unified Modeling Language. Prentice Hall.
24. Giese, H., & Vilbig, A. (2003). Model-driven development of component-based systems with dynamic reconfiguration. In International Workshop on Distributed Event-Based Systems (pp. 135-144). ACM.
25. Vilbig, A. (2004). Modelling dynamic architectures with statecharts. PhD thesis, University of Paderborn.
26. Traon, Y. L., & Jézéquel, J. M. (2005). Tracing software changes with UML models. Springer.
27. France, R. B., & Rumpe, B. (2007). Model-driven development of complex software systems. ACM Computing Surveys (CSUR), 39(4), 11.
28. Atkinson, C., & Kühne, T. (2003). Model-driven development: a metamodeling foundation. IEEE Software, 20(4), 36-41.
29. Bézivin, J. (2005). Model driven engineering: An emerging technical space. In Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (pp. 194-196). ACM.
30. Favre, J. M. (2004). Foundations of Model Driven Engineering. In Journée LMO (pp. 23-30).
31. Zdun, U., & Fricke, L. (2005). Modeling dynamic adaptations with process models. In International Workshop on Distributed Event-Based Systems (pp. 1-6). ACM.
32. Pezze, M., & Young, M. (2007). Software testing and analysis: process, principles, and techniques. John Wiley & Sons.
33. Holzmann, G. J. (2003). Spin model checking: Primer and reference manual. Addison-Wesley Professional.
34. Magee, J., & Kramer, J. (2006). Concurrency: State Models & Java Programs. John Wiley & Sons.

35. Falkner, K., & Striemer, R. (2005). Mapping Statechart Diagrams to Sequence Diagrams. In Australian Software Engineering Conference, 2005. ASWEC 2005. Proceedings (pp. 125-134). IEEE.
36. Snoeck, M., Haesen, R., & Dedene, G. (2002). Using UML class diagrams for static inter-model consistency checking. In International Conference on the Unified Modeling Language (pp. 327-341). Springer Berlin Heidelberg.
37. Baresi, L., Heckel, R., Thöne, S., & Varró, D. (2004). Tutorial on model transformation. In Joint European Conferences on Theory and Practice of Software (pp. 482-496). Springer Berlin Heidelberg.
38. Ehrig, H., Ehrig, K., Habel, A., & Taentzer, G. (2006). Graph transformations for software engineering. Springer Berlin Heidelberg.

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Методологія інтерпретації програмних парадигм на основі засобів UML ”

Обсяг пояснювальної записки: 79 аркушів.

Дата закінчення роботи: 10 червня 2025 р.

Підпис студента _____