Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ДОСЛІДЖЕННЯ ЗАСОБІВ ФОРМАЛІЗАЦІЇ ПАТЕРНІВ ПРОЕКТУВАННЯ

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
к.н., доц. Бублик В.В.
_____
(підпис)
“____” _____ 2021 р.

Виконав студент
Семенюк Х.Р.
“____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
доцент, к.н.
_____О. П. Жежерун
(підпис)
„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Семенюк Христині Романівні факультету інформатики 1-го курсу МП
ТЕМА Дослідження засобів формалізації патернів проектування
Вихідні дані:

-
-

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1 Патерни
2 Статичний диспетчер (Static Dispatcher)
3 Динамічний диспетчер (Dynamic Dispatcher)
Висновки
Список літератури

Дата видачі „___” _____ 2021 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: Дослідження засобів формалізації патернів проектування

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	січень – лютий 2021	
2.	Огляд технічної літератури за темою роботи	лютий – березень 2021	
3.	Створення практичної частини роботи	березень-квітень 2021	
4.	Написання текстової частини роботи	березень-квітень 2021	
5.	Надання роботи керівнику для перевірки	квітень 2021	
6.	Коригування роботи за результатами перевірки	квітень 2021	
7.	Подання роботи на кафедру для перевірки на плагіат	11.05.2021	
8.	Здача курсової роботи	17.05.2021	

Студент Семенюк Х.Р.

Керівник Бублик В.В.

“_____”_____

Зміст

Анотація

Роботу присвячено створенню та специфікації патернів на основі реалізації мультиметодів, розглянутої у книзі Александреску «Сучасне проектування на C++», а також вдосконаленню реалізації за допомогою можливостей нових версій C++. Проаналізовано підхід до патернів проектування та засоби формалізації, описані у книзі Джейсона Мак-Колм Сміта «Елементарні шаблони проектування». Проаналізовано реалізацію мультиметодів з книги Александреску. Наведено специфікації та декомпозиції патернів Статичний диспетчер та Динамічний диспетчер. Реалізовано статичну і динамічну подвійну диспетчеризацію з використанням списків типів на основі варіативних шаблонів та приклади їхнього застосування.

Вступ

Книга Джейсона Мак-Колм Сміта «Елементарні шаблони проектування» присвячена елементарним патернам, які, за словами автора, є основою для дослідження і застосування шаблонів у програмній інженерії. Елементарні патерни необхідні для формалізації проектування, бо вони, на відміну від більш абстрактних шаблонів, невеликі, точно визначені і повсюдно зустрічаються в об'єктно-орієнтованому програмуванні. Одне з призначень шаблонів проектування полягає в перекладі інтуїтивних і рефлекторних рішень в область свідомих концепцій, специфікації елементарних патернів повинні сприяти цьому процесу і спрощувати обговорення основних принципів і їх використання.

У книзі наведено пояснення понять патернів проектування та елементарних патернів. Окрім цього, визначено специфікації елементарних шаблонів виклику метода та чотирьох фундаментальних шаблонів об’єктно-орієнтованого програмування. Розглянуто систему позначень екземплярів шаблонів Pattern Instance Notation (PIN), представлено патерни за її допомогою, розглянуто декомпозицію патернів на складові патерни, у тому числі елементарні. Цей підхід – специфікації патернів, декомпозиція, зображення за допомогою системи PIN – було б корисно застосовувати до патернів високого рівня для формалізації проектування.

У книзі Андрія Александреску «Сучасне проектування на C++» розглянуто різні реалізації мультиметодів або множинної диспетчеризації – механізму, який надає можливість виклику різних реалізацій функції залежно від динамічних типів кількох об’єктів, наприклад аргументів функції. Цей механізм варто використовувати, коли операція маніпулює кількома поліморфними об'єктами за допомогою вказівників чи посилань на їхні базові класи, і її потрібно модифікувати залежно від динамічних типів цих об’єктів.

Цей механізм і реалізації з книги Александреску було б корисно дослідити і специфікувати як патерн, здійснити декомпозицію, використавши підхід з книги Сміта «Елементарні шаблони проектування». Окрім того, можна

вдосконалити реалізацію, розглянуту в книзі Александреску, скориставшись можливостями нових версій C++, зокрема варіативними шаблонами.

За мету цієї роботи було поставлено створення та специфікацію патернів на основі реалізації мультиметодів, розглянутої у книзі Александреску, а також вдосконалення реалізації за допомогою можливостей нових версій C++.

Щоб досягти поставленої мети, потрібно розв'язати такі завдання:

а) проаналізувати й описати підхід до патернів та засобів формалізації, описаний у книзі Джейсона Мак-Колм Сміта «Елементарні шаблони проектування»;

б) проаналізувати реалізацію мультиметодів з книги Александреску;

в) реалізувати мультиметоди з книги Александреску, скориставшись можливостями нових версій C++;

г) описати патерни на основі розглянутих і реалізованих варіантів мультиметодів.

Робота складається з трьох розділів.

Перший розділ присвячено загальному огляду поняття патернів проектування та засобів формалізації, описаних у книзі Джейсона Мак-Колм Сміта «Елементарні шаблони проектування».

У другому розділі наведено специфікацію та декомпозицію патерна Статичний диспетчер.

У третьому розділі наведено специфікацію та декомпозицію патерна Динамічний диспетчер.

Створено програмний продукт: реалізація статичної і динамічної подвійної диспетчеризації з використанням списків типів на основі варіативних шаблонів; приклад застосування – диспетчеризація викликів функцій для ієрархії класів з базовим класом Shape і трьома похідними.

1 Патерни

У книзі Джейсона Мак-Колм Сміта «Елементарні шаблони проектування» надано таке визначення: патерн проектування – загальне рішення загальної проблеми у конкретному контексті. Будь-який шаблон можна декомпонувати, тобто розкласти на складові частини, які також є шаблонами, і продовжувати декомпозицію, доки частини не будуть елементарними шаблонами. Книга Джейсона Сміта присвячена елементарним патернам, які, за словами автора, є основою для дослідження і застосування шаблонів у програмній інженерії. Патерни – це концепції, незалежні від мов програмування, а елементарні патерни є їхніми будівельними блоками. Елементарні патерни необхідні для формалізації проектування, бо вони, на відміну від більш абстрактних шаблонів, невеликі, точно визначені і повсюдно зустрічаються в об'єктно-орієнтованому програмуванні. Одне з призначень шаблонів проектування полягає в перекладі інтуїтивних і рефлекторних рішень в область свідомих концепцій, специфікації елементарних патернів повинні сприяти цьому процесу і спрощувати обговорення основних принципів і їх використання.

Згідно з визначенням Сміта, елементарні патерни – це ті, які містять лише одне відношення між двома сутностями. Сутності: типи, об’єкти, методи, поля. Класи не вважаються окремим типом сутностей, адже не у всіх об’єктно-орієнтованих мовах є класи, і формально класи є типами, якщо не брати до уваги статичних членів. Сутність, яка володіє елементами класу, тобто статичними, можна інтерпретувати як об’єкт і називати об’єкт-клас (class object). Отже, клас емулюється за допомогою типа і об’єкта. Автор наводить види взаємодій між сутностями. Для визначення елементарних шаблонів проектування достатньо всього чотирьох з них:

а) виклик метода: один метод залежить від іншого;

б) використання поля методом: метод залежить від поля;

в) задання поля методом: поле залежить від метода;

г) залежність одного поля від іншого.

У книзі розглянуті елементарні патерни, які містять відношення між сутностями виклик метода. У кожного такого відношення є компоненти: метод, який викликає, метод, який викликають, об’єкти (екземпляри або класи-об’єкти для статичних; простір імен можна трактувати як об’єкт), що містять ці методи, і їхні типи. Ці компоненти можуть мати різну схожість (similarity):

а) схожість між об’єктами: однакові чи різні;

б) схожість між типами: однакові, різні, однорівневі зі спільним предком, тип і підтип;

в) схожість між методами: однакові чи різні (однакові – це ті, які вирішують схожі завдання. Оскільки назви методів повинні вказувати на їхнє призначення, методи з однаковим назвами можна вважати однаковими).

Автор розглядає схожість як осі контексту, координатні осі тривимірного простору. Будь-який виклик метода можна помістити у цей простір, визначивши схожість його компонентів. Цей простір надає групу з дванадцяти елементарних патернів виклику метода: Delegation, Redirection, Conglomeration, Recursion, Revert Method, Extend Method, Delegated Conglomeration, Redirected Recursion, Trusted Delegation, Trusted Redirection, Deputized Delegation, Deputized Redirection. Вони розглянуті у книзі разом з чотирма фундаментальними шаблонами об’єктно-орієнтованого програмування: Create Object, Retrieve, Inheritance Relation, Abstract Interface.

У книзі визначено специфікації патернів у такому ж форматі, як і в книзі Gang of Four (GoF) «Шаблони проектування: Елементи повторно використовуваного об'єктно-орієнтованого програмного забезпечення»:

а) ім’я;

б) призначення;

в) мотивація (проблема, яку вирішує шаблон);

г) застосування (як шаблон варто чи не варто використовувати);

д) структура (у вигляді UML, PINbox);

е) учасники (сутності, які беруть участь у патерні; ролі);

є) відносини;

ж) результати;

з) реалізація.

У книзі Джейсона Сміта патерни зображені за допомогою графічної системи позначень екземплярів шаблонів – Pattern Instance Notation (PIN), яка дозволяє описувати шаблони та способи їхньої взаємодії. PIN може використовуватись з іншими системами позначень, наприклад UML. Компонент PINbox зображує окремий екземпляр шаблона і дозволяє вибирати рівень деталізації при його поданні.

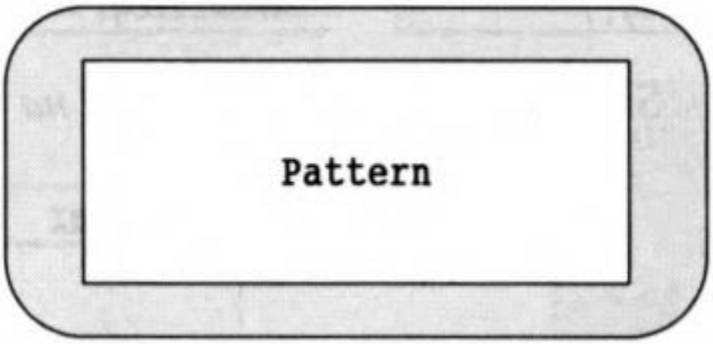

Рисунок 1.1 – Згорнутий компонент PINbox

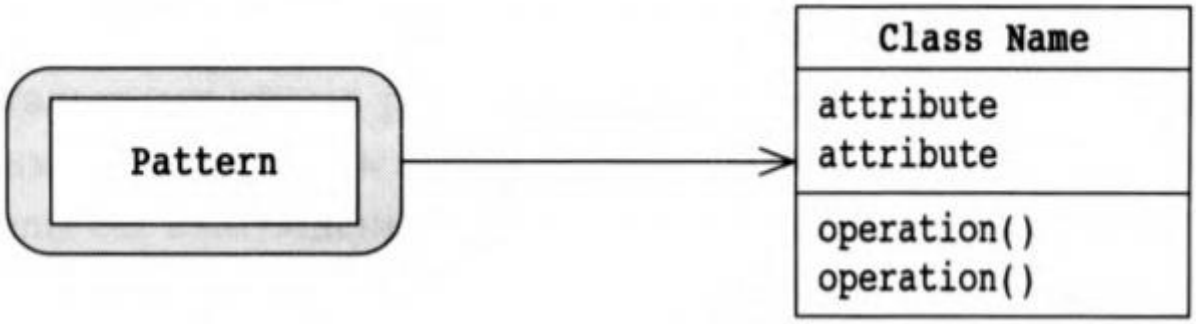

Рисунок 1.2 – Згорнутий компонент PINbox у якості анотації до діаграми UML

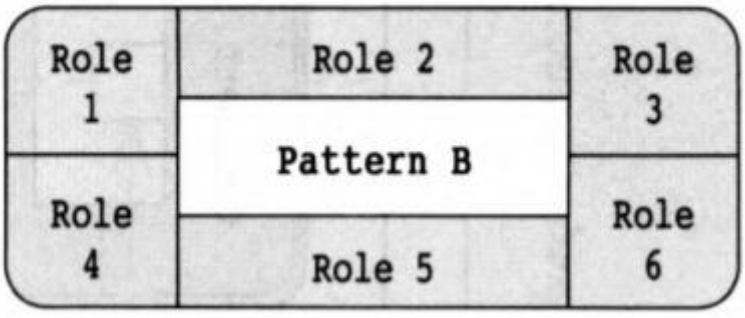

Рисунок 1.3 – Стандартний компонент PINbox

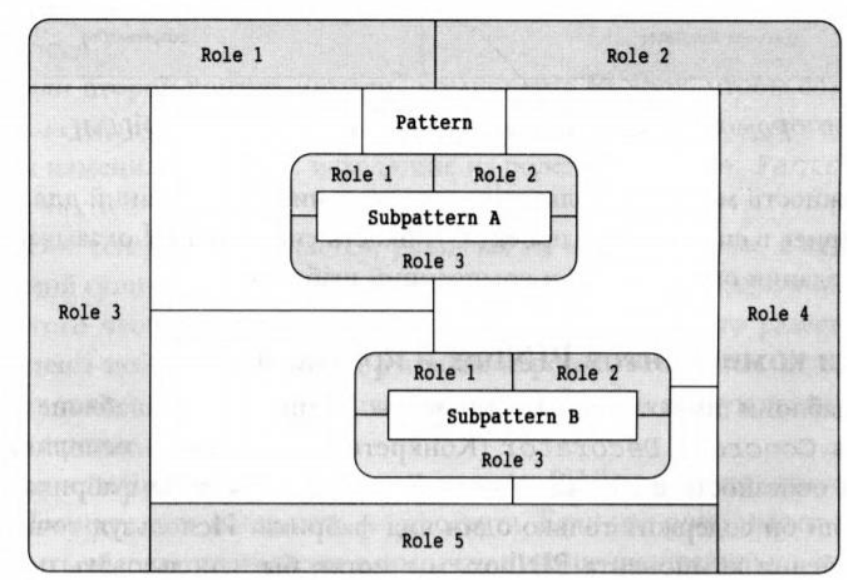

Рисунок 1.4 – Розкритий компонент PINbox

Компоненти можна розкривати на декілька рівнів, а також поміщати в них всередину діаграми UML.

Стеки компонентів можна використовувати для позначення патернів з кратними елементами (наприклад, декілька похідних класів).

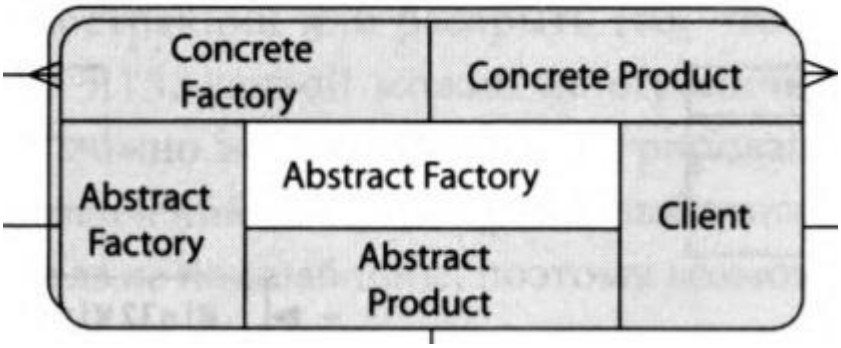

Рисунок 1.5 – Стек компонентів PINbox

Система PIN та специфікації є засобами формалізації патернів проектування. [2]

Отже, патерн проектування – загальне рішення загальної проблеми у конкретному контексті. Елементарні патерни є будівельними блоками шаблонів проектування і основою для їхнього дослідження і застосування. Елементарні шаблони необхідні для формалізації проектування, вони невеликі, точно визначені і повсюдно зустрічаються в об'єктно-орієнтованому програмуванні. Одне з призначень шаблонів проектування полягає в перекладі інтуїтивних і рефлекторних рішень в область свідомих концепцій, специфікації елементарних

патернів повинні сприяти цьому процесу і спрощувати обговорення основних принципів і їх використання. Елементарні патерни виклику метода: Delegation, Redirection, Conglomeration, Recursion, Revert Method, Extend Method, Delegated Conglomeration, Redirected Recursion, Trusted Delegation, Trusted Redirection, Deputized Delegation, Deputized Redirection. Фундаментальні шаблони об’єктно-орієнтованого програмування: Create Object, Retrieve, Inheritance Relation, Abstract Interface. Формат специфікації патернів:

а) ім’я;

б) призначення;

в) мотивація (проблема, яку вирішує шаблон);

г) застосування (як шаблон варто чи не варто використовувати);

д) структура (у вигляді UML, PINbox);

е) учасники (сутності, які беруть участь у патерні; ролі);

є) відносини;

ж) результати;

з) реалізація.

Патерни зображені за допомогою графічної системи позначень екземплярів шаблонів – Pattern Instance Notation (PIN), яка дозволяє описувати шаблони та способи їхньої взаємодії. Система PIN та специфікації є засобами формалізації патернів проектування.

2 Статичний диспетчер (Static Dispatcher)

2.1 Призначення

Призначений для диспетчеризації виклику функції залежно від динамічних типів кількох об’єктів. [1, 281]

2.2 Мотивація

Поліморфізм – концепція, яка означає, що виклик певної функції може бути пов’язаний з різними реалізаціями і вибір реалізації залежить від статичного

чи динамічного контексту, тобто від типів даних об’єктів, для яких застосовується функція. У мові C++ реалізовано два типи поліморфізму:

а) статичний (compile-time polymorphism), який здійснюється за допомогою перевантаження функцій (overloading) та шаблонних функцій;

б) динамічний (run-time polymorphism), який здійснюється за допомогою віртуальних функцій.

При виклику віртуальної функції вибір конкретної її реалізації здійснюється під час виконання програми залежно від динамічного типу об’єкта, для якого викликається ця функція. Наприклад, певна віртуальна функція оголошена у базовому класі з ключовим словом virtual і перевизначена у похідному класі. Якщо створити вказівник базового класу, який вказує на об’єкт похідного класу, і викликати віртуальну функцію із цього вказівника, то викличеться реалізація функції з похідного класу, а не з базового.

Проте конкретна реалізація віртуальної функції обирається залежно від динамічного типу лише одного об’єкта – того, з якого викликається функція. Механізм віртуальних функцій не масштабується для кількох об’єктів, проте у деяких ситуаціях потрібна можливість виклику різних реалізацій функції залежно від динамічних типів кількох об’єктів, наприклад аргументів функції. Зазвичай для таких ситуацій характерна наявність у програмі операції, яка маніпулює кількома поліморфними об'єктами за допомогою вказівників чи посилань на їхні базові класи, і яку потрібно модифікувати залежно від динамічних типів цих об’єктів. Зіткнення об’єктів (collisions) у відеогрі є одним з прикладів таких ситуацій.

Ще один приклад такої ситуації розглянуто у книзі Андрія Александреску «Сучасне проектування на C++». Є програма для малювання, у ній визначено ієрархію класів фігур: абстрактний клас Shape, похідні від нього класи Rectangle, Ellipse, Poly та інші. Потрібно реалізувати методи, які штрихують перетини двох фігур і приймають вказівники на об’єкти типу Shape. Методи мають бути спеціалізовані для кожної пари типів фігур, наприклад прямокутник-прямокутник, прямокутник-еліпс і т. д., адже один узагальнений алгоритм для

всіх типів фігур може бути складним і неефективним. У цій програмі, як і загалом у ситуаціях такого типу, потрібно застосувати механізм виклику різних реалізацій функції залежно від динамічних типів двох об’єктів. Цей механізм називається «мультиметоди» або «множинна диспетчеризація». Якщо об’єкти два – «подвійна диспетчеризація». [1, 282-283]

2.3 Застосування

Патерн Множинна диспетчеризація і варіант його реалізації Статичний диспетчер потрібно застосовувати у таких ситуаціях:

а) операція маніпулює кількома поліморфними об'єктами за допомогою вказівників чи посилань на їхні базові класи, цю операцію потрібно модифікувати залежно від динамічних типів цих об’єктів.

2.4 Структура

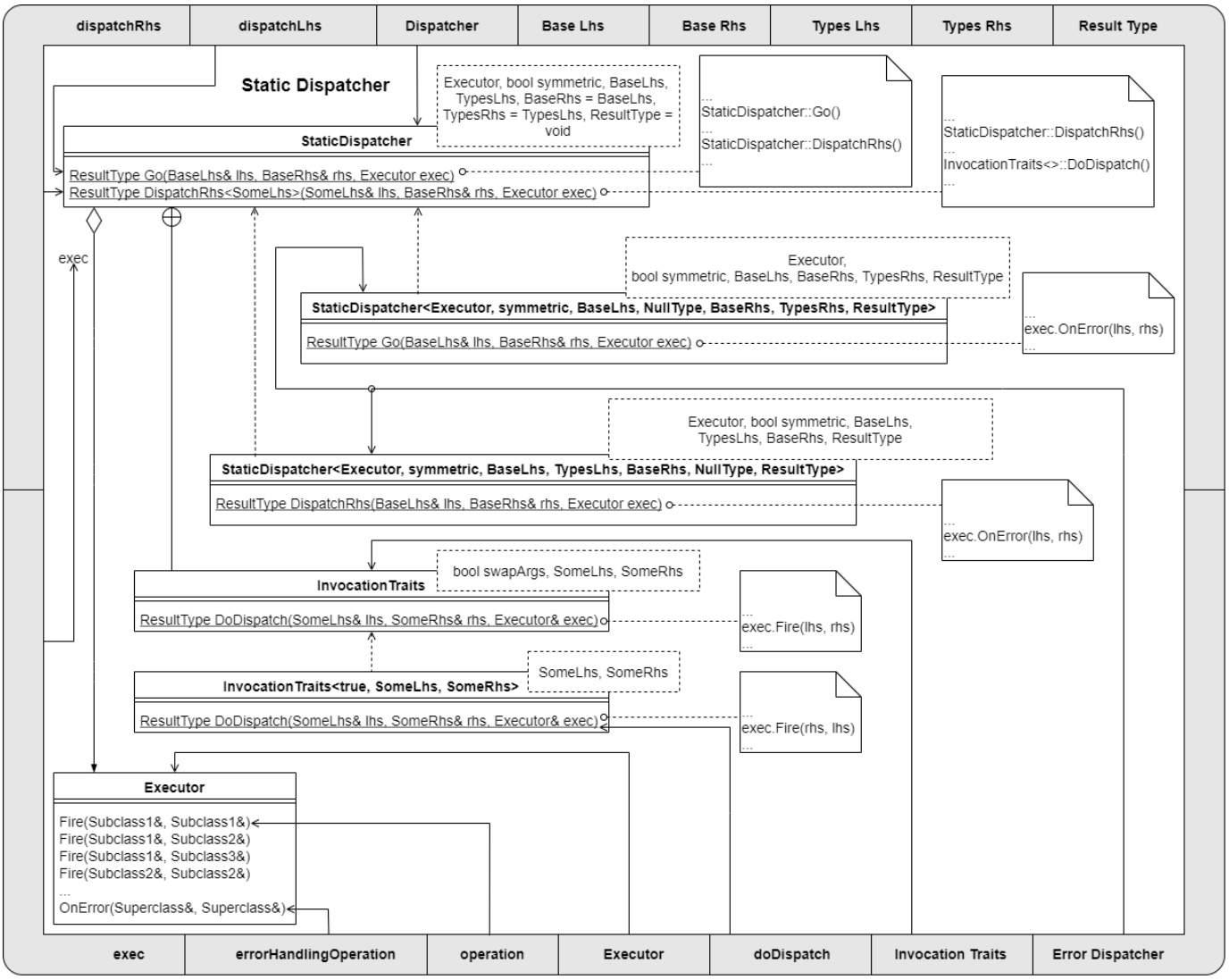

Рисунок 2.1 – Патерн Static Dispatcher у вигляді розкритого компонента PINbox та діаграми UML

2.5 Учасники

Dispatcher

Шаблонний клас, який виконує алгоритм визначення типів аргументів і викликає функцію operation з класу Executor з аргументами, зведеними до відповідного похідного типу.

dispatchLhs

Статична функція з класу Dispatcher, яка визначає тип першого, лівого аргументу lhs. Здійснює зведення адреси аргументу lhs до першого типу у списку

Types Lhs. Якщо зведення не виконалось, функція створює новий екземпляр шаблона класу Dispatcher і передає хвіст списку Types Lhs виклику функції dispatchLhs. Якщо зведення виконалось, викликає функцію dispatchRhs з першим аргументом, зведеним до відповідного похідного типу.

dispatchRhs

Статична функція з класу Dispatcher, яка визначає тип другого, правого аргументу rhs. Здійснює зведення адреси аргументу rhs до першого типу у списку Types Rhs. Якщо зведення не виконалось, функція створює новий екземпляр шаблона класу Dispatcher і передає хвіст списку Types Rhs виклику функції dispatchRhs. Якщо зведення виконалось:

а) визначає, чи потрібно поміняти аргументи місцями. Їх потрібно міняти, якщо індекс типу правого аргументу у списку типів Types Rhs менший за аналогічний індекс лівого аргументу і шаблонний булевий аргумент symmetric класу Dispatcher вказує, що мультиметод повинен бути симетричним;

б) встановлює перший шаблонний булевий аргумент класу Invocation Traits, який вказує класу Invocation Traits, чи потрібно поміняти аргументи місцями;

в) викликає функцію doDispatch з шаблонного класу Invocation Traits з аргументами, зведеними до відповідних похідних типів.

Executor

Клас, який містить перевантажені методи, що приймають по два аргументи похідних типів і виконують роботу над об’єктами.

exec

Об’єкт класу Executor.

operation

Перевантажені методи класу Executor, що виконують роботу над об’єктами.

errorHandlingOperation

Метод класу Executor для обробки помилок, який викликається у випадку, коли класу Dispatcher не вдалось визначити тип аргументу.

Error Dispatcher

Спеціалізації шаблонного класу Dispatcher для випадків помилок, коли у шаблонних параметрах класу замість списків похідних типів передається NullType – маркер кінця списку – тобто коли функція dispatchLhs або dispatchRhs не визначила тип аргументу. Викликають метод errorHandlingOperation з класу Executor.

Invocation Traits

Внутрішній шаблонний клас класу Dispatcher, потрібен для забезпечення симетрії. Симетрію мультиметода можна реалізувати, якщо для методів operation не важливий порядок слідування аргументів, і Base Lhs дорівнює Base Rhs та Types Lhs дорівнює Types Rhs. У такому випадку у клієнтському коді, тобто у класі Executor, можна буде реалізувати меншу кількість перевантажених методів operation, клас Dispatcher за допомогою внутрішнього класу Invocation Traits буде за потреби міняти аргументи місцями перед викликом одного з цих методів. Клас Invocation Traits має спеціалізацію для випадку, коли потрібно міняти аргументи місцями, і загальне визначення для випадку, коли не потрібно.

doDispatch

Метод класу Invocation Traits, який залежно від спеціалізації класу міняє порядок слідування аргументів і викликає відповідний перевантажений метод operation з класу Executor з аргументами, зведеними до відповідних похідних типів.

Base Lhs

Базовий тип лівого аргументу.

Base Rhs

Базовий тип правого аргументу, за замовчуванням дорівнює Base Lhs.

Types Lhs

Список типів, що складається з можливих похідних класів для лівого аргументу.

Types Rhs

Список типів, що складається з можливих похідних класів для правого аргументу. За замовчуванням дорівнює Types Lhs.

Result Type

Тип результату подвійної диспетчеризації. Має бути такий же, як у перевантажених методів operation. За замовчуванням дорівнює void.

2.6 Відносини

Шаблонний клас Dispatcher приймає у якості шаблонних параметрів клас Executor, який містить перевантажені функції, що виконують реальну роботу, булевий параметр, який вказує, чи мультиметод повинен бути симетричним, базові типи двох аргументів, списки типів, що складаються з можливих похідних класів для аргументів та тип результату подвійної диспетчеризації. Клас Dispatcher виконує визначення типів аргументів: спочатку лівого у методі dispatchLhs, потім правого у методі dispatchRhs. Цей механізм реалізовано через зведення типів, створення нових екземплярів шаблона класу і виклик цих самих методів у нових екземплярах. Якщо не вийшло визначити типи аргументів, методи викликають метод для обробки помилок з класу Executor. Після визначення типів аргументів клас Dispatcher за допомогою внутрішнього класу Invocation Traits реалізує симетрію, тобто визначає, чи варто міняти місцями аргументи, і міняє за потреби. Клас Invocation Traits викликає відповідний перевантажений метод operation з класу Executor з аргументами, зведеними до відповідних похідних типів.

2.7 Результати

Патерн виконує диспетчеризацію виклику функції залежно від динамічних типів двох об’єктів. Він здійснює лінійний розгорнутий пошук динамічних типів і відкладає перевантаження методів на період виконання програми.

Може виникнути помилка, якщо в ієрархії класів похідні типи мають свої похідні типи, наприклад Rectangle і RoundedRectangle, і якщо менш віддалений

нащадок базового класу (Rectangle), знаходиться ближче до початку у переданому списку типів (Types Lhs або Types Rhs), ніж більш віддалений нащадок. Помилка полягає у тому, що якщо один з аргументів має динамічний тип RoundedRectangle, то цей аргумент буде зведено до типу Rectangle. Адже у ланцюжку перевірок при визначенні типу тип Rectangle зустрінеться швидше, ніж RoundedRectangle, і зведення до цього типу виконається успішно. Щоб уникнути цієї помилки, потрібно, щоб найбільш віддалені нащадки базового типу були на початку списку типів. Можна залишити цю відповідальність на клієнтському коді або реалізувати механізм, який перед диспетчеризацією упорядковуватиме списки типів відповідно до порядку спадкування.

Цей варіант реалізації можна узагальнити для більшої кількості аргументів мультиметода. Для цього потрібно змінити реалізацію так, щоб клас Dispatcher приймав не два списки типів, а один список типів, елементами якого є списки типів для кожного аргументу. Перший тип кожного такого списка типів має бути базовим класом ієрархії, щоб не передавати їх окремо.

Переваги:

а) велика швидкість для невеликої кількості класів;

б) ненав’язливість, тобто не потрібно модифікувати ієрархію класів для використання цієї реалізації мультиметодів.

Недоліки:

а) невелика швидкість для великої кількості класів;

б) сильна залежність між класами (оскільки використано шаблонні класи, ця реалізація мультиметодів є програмою, яка генерує код на етапі компіляції. Згенерований код залежить від ієрархії класів);

в) вищеописана проблема зі спадкуванням і порядком типів у списках. [1]

2.8 Реалізація

Зведення аргументів до типів зі списків здійснюється за допомогою оператора dynamic_cast.

Шаблонний клас Dispatcher приймає у якості шаблонних параметрів два списки типів. Тут можна було б застосувати варіативні шаблони, але клас не може приймати два варіативні шаблони, і поєднати два списки типів у один у цій реалізації теж не можна. Хорошим виходом з цієї ситуації було б реалізувати клас TypeList, який працює з варіативними шаблонами, і передавати два його екземпляра шаблона класу Dispatcher у якості шаблонних параметрів. Приклад реалізації класу TypeList:

```
struct NullType {};

template<typename H, typename... T>
struct TypeList
{
	using Head = H;
	using Tail = TypeList<T...>;
};

template<typename H>
struct TypeList<H, NullType>
{
	using Head = H;
	using Tail = NullType;
};
```

2.9 Декомпозиція

Оглянувши діаграму UML патерна Статичний диспетчер, можна помітити патерни, які є його складовими частинами.

Один з таких патернів, Conglomeration – поведінковий елементарний шаблон, використовується в ситуаціях, коли один об’єкт повинен виконати велике завдання, розбите на підзавдання, кожне з яких виконується в окремому методі, і результати яких поєднуються в одне ціле. Шаблон полягає в тому, що метод об’єкта викликає інший метод цього ж об’єкта цього ж типу. Класифікація викликів метода: об’єкти: однакові; типи об’єктів: однакові; методи: різні. [2, 155]

У Статичному диспетчері цей патерн використовується, коли метод dispatchLhs викликає метод dispatchRhs. Це різні статичні методи. Кожен екземпляр шаблона класу має одну власну копію статичних елементів класу.

Сутність, яка володіє елементами класу, можна інтерпретувати як об’єкт і називати об’єкт-клас (class object). [2, 41] Метод dispatchLhs викликає метод dispatchRhs з того самого екземпляра шаблона класу, тобто з того самого об’єкта-класу.

Redirected Recursion – поведінковий елементарний шаблон, призначений для виконання певного завдання в кількох об’єктах одного типу, що несуть відповідальність за розподілення і активізацію цього завдання. Шаблон полягає в тому, що метод об’єкта викликає цей самий метод іншого об’єкта цього ж типу. Класифікація викликів метода: об’єкти: різні; типи об’єктів: однакові; методи: однакові. [2, 184]

У Статичному диспетчері цей патерн використовується, коли статичні методи dispatchLhs та dispatchRhs викликають самі себе з інших екземплярів шаблона класу, тобто з інших об’єктів-класів.

Delegation – поведінковий елементарний шаблон, призначений для того, щоб переслати чи делегувати частину роботи іншому методу і об’єкту. Шаблон полягає в тому, що метод об’єкта викликає інший метод іншого об’єкта іншого типу. Класифікація викликів метода: об’єкти: різні; типи об’єктів: різні; методи: різні. [2, 144]

У Статичному диспетчері цей патерн використовується, коли метод dispatchRhs викликає статичний метод doDispatch з класу Invocation Traits; коли методи dispatchLhs та dispatchRhs викликають метод errorHandlingOperation з об’єкта класу Executor; коли метод doDispatch з класу Invocation Traits викликає метод operation з об’єкта класу Executor.

Це декомпозиція патерна Статичний диспетчер на елементарні шаблони. На її основі можна зобразити патерн у вигляді розкритого компонента PINbox. Екземпляри патерна Delegation зображені здебільшого окремо, а не в стеках, бо вони не однорідні у цих випадках.

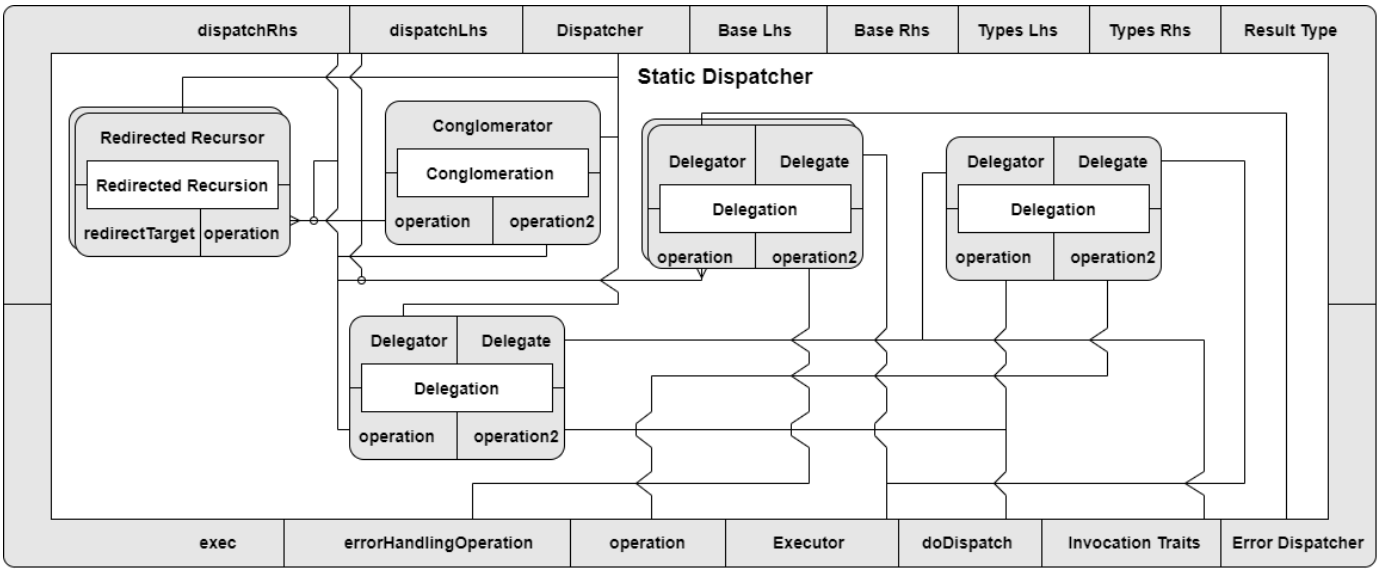

Рисунок 2.2 – Патерн Static Dispatcher у вигляді розкритого компонента PINbox

3 Динамічний диспетчер (Dynamic Dispatcher)

3.1 Призначення

Призначений для диспетчеризації виклику функції залежно від динамічних типів кількох об’єктів. [1, 281]

3.2 Мотивація

Така ж, як у Статичного диспетчера (див. підрозділ 2.2).

3.3 Застосування

Таке ж, як у Статичного диспетчера (див. підрозділ 2.3).

3.4 Структура

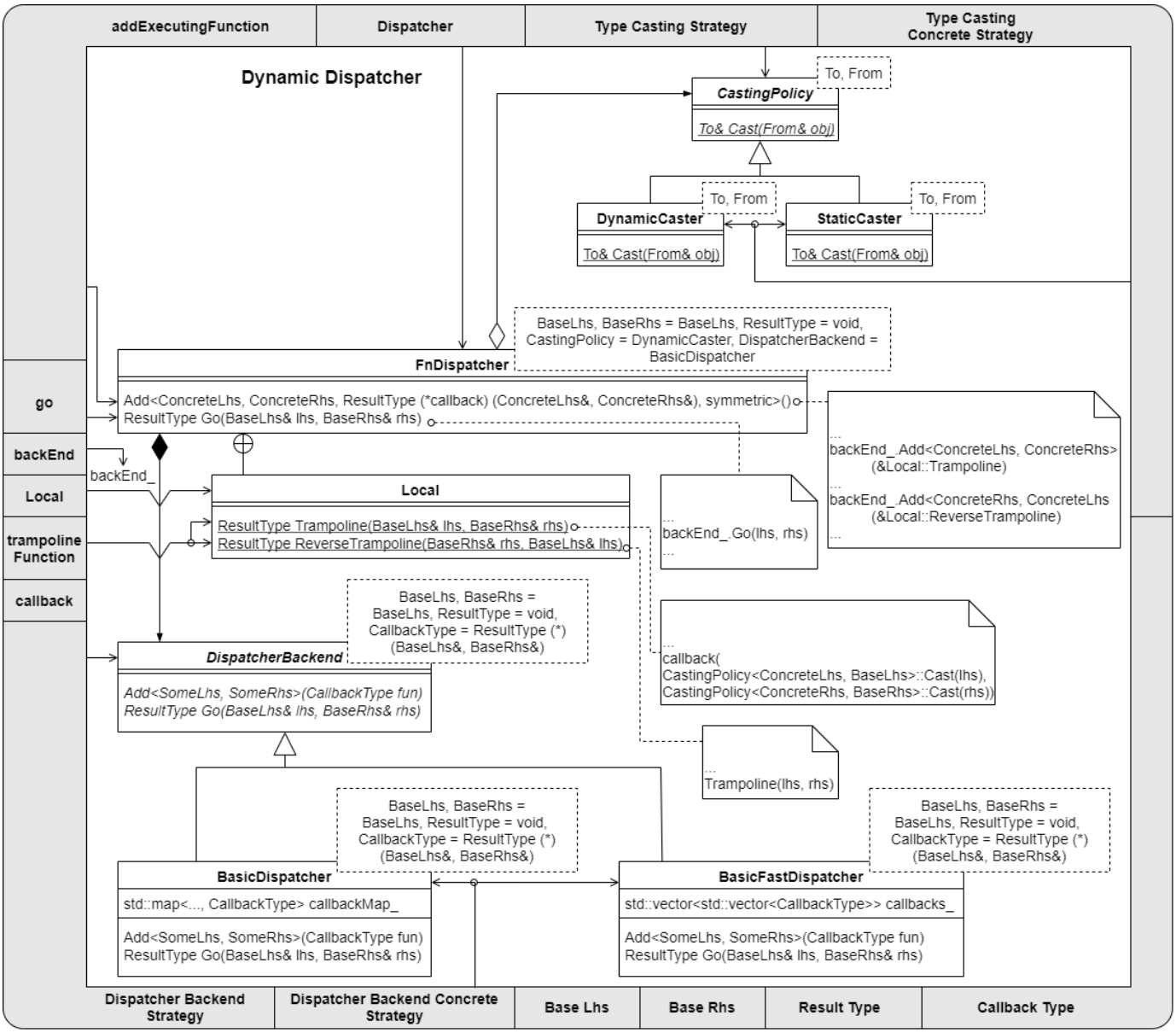

Рисунок 3.1 – Патерн Dynamic Dispatcher у вигляді розкритого компонента PINbox та діаграми UML

3.5 Учасники

Dispatcher

Шаблонний клас, який перевикликає функції addExecutingFunction та go з об’єкта backEnd класу Dispatcher Backend Strategy. Відповідає за симетрію і зведення типів (за допомогою Type Casting Strategy) з базових до відповідних похідних перед викликом клієнтських функцій.

Dispatcher Backend Strategy

Інтерфейсний шаблонний клас, стратегія, тип об’єкта backEnd, який містить функції для додавання в контейнер та отримування з контейнера об’єктів зворотного виклику: addExecutingFunction, go.

Dispatcher Backend Concrete Strategy

Шаблонний клас, конкретна стратегія, що реалізує інтерфейс Dispatcher Backend Strategy та містить контейнер для зберігання об’єктів зворотного виклику.

backEnd

Об’єкт класу Dispatcher Backend Strategy, змінна-член класу Dispatcher.

addExecutingFunction

Шаблонна функція, що міститься в класах Dispatcher, Dispatcher Backend Strategy. Додає об’єкти зворотного виклику в контейнер. Її шаблонні параметри – конкретні похідні типи аргументів для клієнтських функцій, вони використовуються в контейнері як ключі чи індекси (залежно від реалізації).

go

Функція, що міститься в класах Dispatcher, Dispatcher Backend Strategy. Визначає типи аргументів мультиметода, за цими типами отримує об’єкти зворотного виклику з контейнера, здійснює виклик, генерує помилку у разі невдачі.

trampolineFunction

Функції всередині локального класу Local (функція addExecutingFunction у класі Dispatcher). Відповідають за симетрію і зведення типів (за допомогою Type Casting Strategy) перед викликом клієнтських функцій.

Local

Локальний клас у функції addExecutingFunction у класі Dispatcher, містить trampolineFunction.

Type Casting Strategy

Інтерфейсний шаблонний клас, стратегія, відповідає за безпосереднє зведення типів за допомогою операторів static_cast, dynamic_cast.

Type Casting Concrete Strategy

Шаблонний клас, конкретна стратегія, що реалізує інтерфейс Type Casting Strategy.

Base Lhs

Базовий тип лівого аргументу.

Base Rhs

Базовий тип правого аргументу, за замовчуванням дорівнює Base Lhs.

Result Type

Тип результату подвійної диспетчеризації. Має бути такий же, як у клієнтських функцій-виконавців. За замовчуванням дорівнює void.

Callback Type

Тип вказівника на клієнтську функцію-виконавця, тобто тип зворотного виклику.

3.6 Відносини

Шаблонний клас Dispatcher приймає у якості шаблонних параметрів базові типи двох аргументів, тип результату подвійної диспетчеризації, класи конкретних стратегій зведення типів (Type Casting Concrete Strategy) та внутрішнього диспетчера (Dispatcher Backend Concrete Strategy). Клас Dispatcher містить приватну змінну-член backEnd типу Dispatcher Backend Strategy. Також Dispatcher містить функцію addExecutingFunction, яка:

а) містить шаблонні параметри: конкретні похідні типи аргументів для функції, вказівник на функцію, булевий параметр, який вказує, чи функція повинна бути симетричною (рішення про симетричність приймається для кожної функції окремо);

б) застосовує trampolineFunction для реалізації симетрії і зведення типів за допомогою Type Casting Concrete Strategy;

в) викликає функцію addExecutingFunction з об’єкта backEnd і передає їй вказівник на trampolineFunction, яка містить вказівник на клієнтську функцію. Функція addExecutingFunction додає об’єкт зворотного виклику в контейнер.

Функція go з класу Dispatcher викликає функцію go з об’єкта backEnd, яка визначає типи аргументів мультиметода, за цими типами отримує об’єкти зворотного виклику з контейнера, здійснює виклик, генерує помилку у разі невдачі.

Загалом клас Dispatcher Backend Strategy реалізує диспетчеризацію, а клас Dispatcher перевикликає його методи і додає до них реалізацію симетрії і зведення типів аргументів.

3.7 Результати

Патерн виконує диспетчеризацію виклику функції залежно від динамічних типів двох об’єктів. Диспетчеризація здійснюється під час виконання програми завдяки динамічним структурам і алгоритмам, зокрема механізму, який ідентифікує типи і відношення між ними під час виконання (runtime type information).

У цього варіанту реалізації мультиметодів є недолік: якщо зареєструвати функцію для певних типів аргументів, наприклад Rectangle і Poly, вона не буде викликатись для аргументів типів, похідних від них, наприклад RoundedRectangle і Poly. Об’єкти похідних типів не розглядаються як об’єкти їхніх базових типів, а це суперечить правилам спадкування. Диспетчер шукатиме функцію саме для похідних типів і видасть помилку, якщо її не буде.

Результати залежать від типів контейнерів для зберігання об’єктів зворотного виклику:

а) асоціативний масив – алгоритм розпізнавання типу виконує бінарний пошук за логарифмічний час;

б) матриця – алгоритм розпізнавання типу має сталий час виконання. Але для цієї реалізації потрібно додати певний макрос у кожен клас ієрархії, до якого застосовується диспетчеризація.

Цей варіант реалізації можна узагальнити для більшої кількості аргументів мультиметода. Для цього потрібно змінити насамперед розмірність контейнерів для зберігання об’єктів зворотного виклику.

Переваги:

а) непогана швидкість для невеликої кількості класів. Менша, ніж у Статичного диспетчера. Більша у реалізації з матрицею і сталим часом виконання, ніж у реалізації з асоціативним масивом і логарифмічним часом виконання;

б) хороша швидкість для великої кількості класів, особливо у реалізації з матрицею;

в) слабка залежність між класами (диспетчера від ієрархії класів);

г) можливість реєструвати в одному об’єкті диспетчера симетричні і несиметричні функції;

д) можливість реєструвати функції в одному об’єкті диспетчера у різний час і в різних місцях програми.

Недоліки:

а) вищеописана проблема зі спадкуванням: при диспетчеризації об’єкти похідних типів не розглядаються як об’єкти їхніх базових типів;

б) потреба модифікувати ієрархію класів (додавати макрос) для варіанта реалізації з матрицею. [1]

3.8 Реалізація

У реалізації, розглянутій у книзі Александреску, у ролі внутрішнього диспетчера (Dispatcher Backend Concrete Strategy) можуть виступати два класи:

а) BasicDispatcher (використовується за замовчуванням) зберігає об’єкти зворотного виклику в асоціативний масив std::map<..., CallbackType>. Ключ має тип std::pair<TypeInfo, TypeInfo>, де TypeInfo є класом-оболонкою навколо класу std::type_info, яка надає додаткові можливості і дозволяє зберігати свої об’єкти в стандартних контейнерах. У ключі зберігається інформація про конкретні похідні типи аргументів для клієнтської функції. У цій реалізації алгоритм розпізнавання типу виконує бінарний пошук за логарифмічний час;

б) BasicFastDispatcher зберігає об’єкти зворотного виклику в матриці std::vector<std::vector<CallbackType>>. Індекси в матриці – унікальні цілі числа,

що присвоюються класам ієрархії. Диспетчер сам здійснює індексування, але клас повинен зберігати індекс у статичній змінній і містити в макросі віртуальну функцію, яка повертає посилання на цю змінну. При диспетчеризації клас BasicFastDispatcher отримує індекси класів аргументів через віртуальні функції, отримує з матриці за індексами об’єкт зворотного виклику і викликає функцію через вказівник. Алгоритм розпізнавання типу має сталий час виконання.

У ролі диспетчера (Dispatcher) можуть виступати два класи:

а) FnDispatcher для диспетчеризації функцій. Клієнтські функції для FnDispatcher оголошені і визначені у безіменному просторі імен, щоб вони були глобальними і вказівники на них можна було передавати функції FnDispatcher::Add як шаблонний параметр без типу (non-type). Таким чином, компілятор володіє статичною інформацією про ці вказівники і фіксує адреси клієнтських функцій у коді трамплінних функцій (виконують роль trampolineFunction у патерні), тому в контейнері не потрібно зберігати вказівники на клієнтські функції, а лише на трамплінні. Трамплінні функції – це невеликі функції, що виконують налаштування перед викликами інших функцій.

б) FunctorDispatcher для диспетчеризації функторів. Функтори – це класи, які перевантажують оператор виклику функції operator()(), і таким чином імітують виклик функції. Функтори потрібні у випадку, коли треба, щоб об’єкт зворотного виклику зберігав стан (за допомогою змінних-членів). Адже функції можуть зберігати тільки значення статичних змінних. Структура класу FunctorDispatcher дуже схожа на структуру FnDispatcher. У ролі локального класу Local виступає не клас зі статичними трамплінними функціями, а клас, який зберігає об’єкт функтора як свою змінну-член, адже трамплінні функції не могли б зберігати стан функтора. У реалізації Александреску локальний клас є похідним від класу FunctorImplType, розглянутого у п’ятій главі його книги «Сучасне проектування на C++».

У диспетчері можна було б використовувати для зведення типів оператор static_cast або dynamic_cast, але було використано стратегію (Type Casting Strategy). Дві конкретні стратегії StaticCaster і DynamicCaster (використовується

за замовчуванням) здійснюють зведення типів за допомогою операторів static_cast і dynamic_cast відповідно. Це дає можливість у клієнтському коді обирати між операторами або використати свою власну стратегію, яка обирає оператор залежно від встановлених правил. Ця можливість корисна, адже в різних ситуаціях краще підходять різні оператори зведення типів:

а) dynamic_cast безпечніший і повільніший за рахунок додаткових перевірок;

б) static_cast швидший, але не працює при віртуальному та множинному спадкуванні. [1]

3.9 Декомпозиція

Оглянувши діаграму UML патерна Динамічний диспетчер, можна визначити патерни, які є його складовими частинами.

Один з таких патернів, Strategy – поведінковий шаблон, який визначає сімейство схожих алгоритмів для одного об'єкта, розміщує кожен з них у власному класі та робить їх взаємозамінними незалежно від об'єктів-клієнтів, які їх використовують.

У Динамічному диспетчері цей патерн використовується для зведення типів за допомогою різних алгоритмів (Type Casting Strategy) та внутрішнього диспетчера з різною реалізацією та однаковою поведінкою (Dispatcher Backend Strategy).

Redirection – поведінковий елементарний шаблон, призначений для створення запиту, за яким інший об’єкт повинен виконати підзавдання, тісно пов’язане з головним завданням, або саме завдання. Шаблон полягає в тому, що метод об’єкта викликає такий же метод іншого об’єкта іншого типу. Класифікація викликів метода: об’єкти: різні; типи об’єктів: різні; методи: однакові. [2, 149]

У Динамічному диспетчері цей патерн використовується, коли методи addExecutingFunction і go з об’єкта класу Dispatcher викликають такі ж методи з об’єкта backEnd класу Dispatcher Backend Strategy.

Патерн Delegation полягає в тому, що метод об’єкта викликає інший метод іншого об’єкта іншого типу. Класифікація викликів метода: об’єкти: різні; типи об’єктів: різні; методи: різні (див. підрозділ 2.9).

У Динамічному диспетчері цей патерн використовується, коли метод trampolineFunction класу Local викликає клієнтську функцію з безіменного простору імен, який можна трактувати як об’єкт. Також патерн Delegation використовується, коли метод trampolineFunction класу Local викликає метод для зведення типів із класу Type Casting Strategy; коли метод go з об’єкта класу Dispatcher Backend Strategy викликає trampolineFunction, яка зберігається у контейнері об’єктів зворотних викликів.

Це декомпозиція патерна Динамічний диспетчер на шаблони. На її основі можна зобразити патерн у вигляді розкритого компонента PINbox.

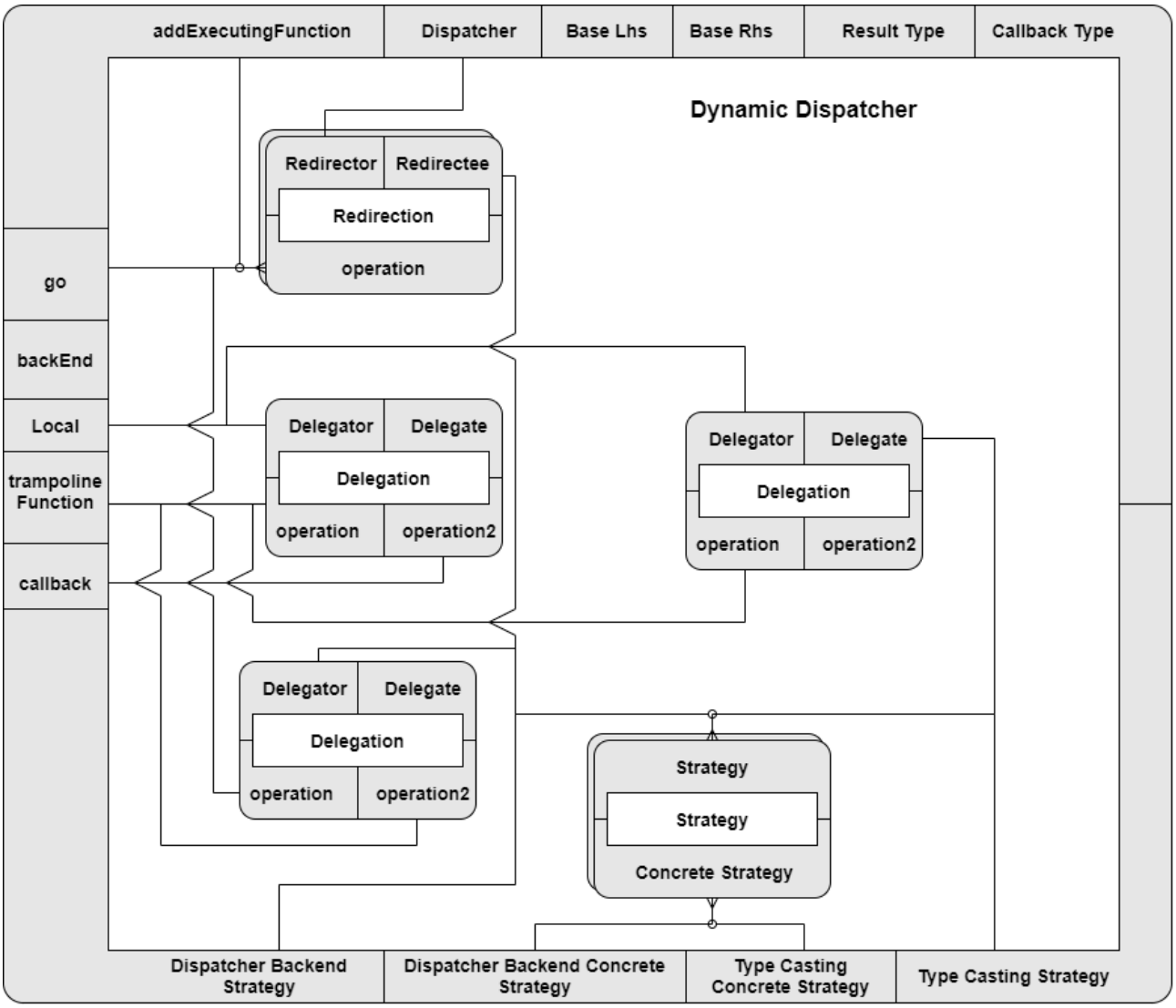

Рисунок 3.2 – Патерн Dynamic Dispatcher у вигляді розкритого компонента PINbox

Висновки

Отже, патерн проектування – загальне рішення загальної проблеми у конкретному контексті. Елементарні патерни є будівельними блоками шаблонів проектування і основою для їхнього дослідження і застосування. Елементарні шаблони необхідні для формалізації проектування, вони невеликі, точно визначені і повсюдно зустрічаються в об'єктно-орієнтованому програмуванні. Одне з призначень шаблонів проектування полягає в перекладі інтуїтивних і рефлекторних рішень в область свідомих концепцій, специфікації елементарних патернів повинні сприяти цьому процесу і спрощувати обговорення основних принципів і їх використання. Елементарні патерни виклику метода: Delegation, Redirection, Conglomeration, Recursion, Revert Method, Extend Method, Delegated Conglomeration, Redirected Recursion, Trusted Delegation, Trusted Redirection, Deputized Delegation, Deputized Redirection. Фундаментальні шаблони об’єктно-орієнтованого програмування: Create Object, Retrieve, Inheritance Relation, Abstract Interface. Формат специфікації патернів:

а) ім’я;

б) призначення;

в) мотивація (проблема, яку вирішує шаблон);

г) застосування (як шаблон варто чи не варто використовувати);

д) структура (у вигляді UML, PINbox);

е) учасники (сутності, які беруть участь у патерні; ролі);

є) відносини;

ж) результати;

з) реалізація.

Патерни зображені за допомогою графічної системи позначень екземплярів шаблонів – Pattern Instance Notation (PIN), яка дозволяє описувати шаблони та способи їхньої взаємодії. Система PIN та специфікації є засобами формалізації патернів проектування.

Патерни Статичний і Динамічний диспетчер призначені для диспетчеризації виклику функції залежно від динамічних типів кількох об’єктів. Потрібно застосовувати, коли операція маніпулює кількома поліморфними об'єктами за допомогою вказівників чи посилань на їхні базові класи, і цю операцію потрібно модифікувати залежно від динамічних типів цих об’єктів.

Патерн Статичний диспетчер здійснює лінійний розгорнутий пошук динамічних типів і відкладає перевантаження методів на період виконання програми.

Переваги Статичного диспетчера:

а) велика швидкість для невеликої кількості класів;

б) ненав’язливість, тобто не потрібно модифікувати ієрархію класів для використання цієї реалізації мультиметодів.

Недоліки Статичного диспетчера:

а) невелика швидкість для великої кількості класів;

б) сильна залежність між класами (оскільки використано шаблонні класи, ця реалізація мультиметодів є програмою, яка генерує код на етапі компіляції. Згенерований код залежить від ієрархії класів);

в) вищеописана проблема зі спадкуванням і порядком типів у списках.

У патерні Динамічний диспетчер диспетчеризація здійснюється під час виконання програми завдяки динамічним структурам і алгоритмам, зокрема механізму, який ідентифікує типи і відношення між ними під час виконання (runtime type information).

Переваги Динамічного диспетчера:

а) непогана швидкість для невеликої кількості класів. Менша, ніж у Статичного диспетчера. Більша у реалізації з матрицею і сталим часом виконання, ніж у реалізації з асоціативним масивом і логарифмічним часом виконання;

б) хороша швидкість для великої кількості класів, особливо у реалізації з матрицею;

в) слабка залежність між класами (диспетчера від ієрархії класів);

г) можливість реєструвати в одному об’єкті диспетчера симетричні і несиметричні функції;

д) можливість реєструвати функції в одному об’єкті диспетчера у різний час і в різних місцях програми.

Недоліки Динамічного диспетчера:

а) вищеописана проблема зі спадкуванням: при диспетчеризації об’єкти похідних типів не розглядаються як об’єкти їхніх базових типів;

б) потреба модифікувати ієрархію класів (додавати макрос) для варіанта реалізації з матрицею.

Створено програмний продукт: реалізація статичної і динамічної подвійної диспетчеризації з використанням списків типів на основі варіативних шаблонів; приклад застосування – диспетчеризація викликів функцій для ієрархії класів з базовим класом Shape і трьома похідними.

Перспективи покращення досягнутих результатів:

а) виправити недолік Статичного диспетчера, пов’язаний зі спадкуванням і порядком типів у списках;

б) виправити недолік Динамічного диспетчера, який полягає у тому, що об’єкти похідних типів не розглядаються як об’єкти їхніх базових типів;

в) усунути потребу модифікувати ієрархію класів (додавати макрос) для варіанта реалізації з матрицею.

Список літератури

1. Александреску, Андрей Современное проектирование на C++. Серия C++ In-Depth, т. 3.: Пер. с англ. – М. : Издательский дом «Вильямс», 2002. – 336 с. : ил. – Парал. тит. англ.

2. Смит, Джейсон Мак-Колм Элементарные шаблоны проектирования. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2013. — 304 с.: ил. — Парал. тит. англ.

3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001. – 368 с.: ил. (Серия «Библиотека программиста»)

4. Шаблони (C++) [Електронний ресурс] : Матеріал з Вікіпедії — вільної енциклопедії : Версія 29295080, збережена о 17:07 UTC 21 серпня 2020, Режим доступу: https://uk.wikipedia.org/w/index.php?title=%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%B8_(C%2B%2B)&oldid=29295080

5. www.ibm.com [Електронний ресурс] : [Інтернет-портал]. – Режим доступу: https://www.ibm.com/docs/en/zos/2.2.0?topic=only-static-data-members-templates-c – Назва з екрану.

6. www.tutorialspoint.com [Електронний ресурс] : [Інтернет-портал]. – Режим доступу: https://www.tutorialspoint.com/cplusplus/cpp_static_members.htm – Назва з екрану.