АВТОМАТИЗАЦІЯ ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ

УДК 004. 4 DOI: https://doi.org/10.32515/2664-262X.2022.5(36).1.98-104

О.С. Улічев, канд. техн. наук, **К.О. Буравченко,** канд. техн. наук, **Л.І. Поліщук,** викл.
Центральноукраїнський національний технічний університет, м.Кропивницький, Україна
e-mail: askin79@gmail.com, buravchenkok@gmail.com, pli_80@ukr.net

Об'єктно-орієнтований підхід в програмуванні. Композиційна взаємодія об'єктів

В даній статті проведено дослідження певних аспектів об'єктно-орієнтованого підходу в програмуванні. Основну увагу приділено поняттям взаємодії об'єктів та композиційної взаємодії. В статті розглядається ієрархія взаємодії та особливості конкретних способів її реалізації.

На основі опрацьованих матеріалів (навчальних посібників, книги з даної тематики, професійних дискусій розробників) запропоновано трактування основних типів взаємодій.

об'єктно-орієнтований підхід, клас, об'єкт, взаємодія, міжоб'єктні зв'язки

Постановка проблеми. Одна з популярних та актуальний сьогодні парадигм в програмуванні це об'єктно-орієнтована парадигма (підхід). Сам термін «ПАРАДИГМА» приписують авторові Т. Куну, термін був запропонований в його роботі «Структура наукових революцій» [7], але в розрізі програмування цей термін вперше застосував в 1978 році Р. Флойд. Хоча термін з'явився лише в 1978 році, самі парадигми утворилися істотно раніше, як тільки з'явилися перші програмовані комп'ютери, почали з'являтися і удосконалюватися підходи і методи. Кількість парадигм складно визначити однозначно, та й кордони між конкретними парадигмами бувають досить умовними. Так, наприклад, автор К.А. Хайдаров [8] визначає ряд парадигм, а зокрема об'єктно-орієнтовану парадигму програмування автор визначає наступним способом: «программа = несколько взаимодействующих объектов, функциональность (действия) и данные распределяются между этими объектами» (мовою оригіналу). Ключовим моментом є взаємодія об'єктів, в той же час відсутні чіткі правила визначення даної взаємодії.

Аналіз останніх досліджень і публікацій. Парадигма об'єктно-орієнтованого програмування базується на основі процедурного програмування. З моменту виникнення ідеї нового підходу (погляду) на програмування, що була запропонована Аланом Кеєм, ООП постійно еволюціонувала.

Базові поняття та їх трактування не набули суттєвих змін, але в деталях та нюансах виникає ряд протиріч. Протиріччя породжено різними факторами, зокрема особливостями реалізації ООП підходу в різних мовах програмування, і, як наслідок, трактуванням даних понять різними авторами, бо, здебільшого, автори опираються на певну мову програмування, а не розглядають ООП як абсолютну абстракцію.

На такі протиріччя вказують ряд зарубіжних та вітчизняних авторів. Зокрема В.В. Соколов в своїй статті[11] відмічає – «Засоби взаємодії об’єктів в ООП не визначені. Інколи взаємодію розглядають як обмін повідомленнями між об’єктами, хоча на практиці це звичайне використання даних та виклики функцій об’єктів.» Автор[11] наголошує на тому, що відсутність уніфікованих засобів взаємодії об’єктів є певним недоліком методології й вимагає формалізації.

В зв’язку з відсутністю чіткої формалізації понять взаємодії між класами та об’єктами, а також відсутності чіткої класифікації типів взаємодії, різні автори достатньо вільно трактують дані поняття. Наприклад, автор К.М. Лавріщева[12] визначає вирішальним фактором відмінності «композиції» та «агрегації» (різних типів відношення) час існування залежного та базового класів, а тип «асоціація» розглядається як окремий тип.

Ситуація неоднозначного трактування понять, що має місце в різних джерелах, породжує певні протиріччя та ускладнює розуміння даних аспектів ООП (типів зв’язків між класами).

Постановка завдання. Дана стаття має на меті розглянути типи взаємодії, а зокрема композиційної взаємодії між об’єктами, та визначити особливості окремих типів. Метою статті є з’ясування суті понять різних типів відношень між класами.

Викладки даної статті базуються на аналізі публікацій, як авторів-теоретиків[1, 2, 3, 8], так і публікацій та аналізу міркувань практиків в сфері прикладного застосування ООП та вибудовування архітектури ОО-програм (програм на основі об’єктно-орієнтованого підходу)[4, 5, 6, 10].

В статті запропоновано ряд тез, які ілюструють неоднозначність тлумачення та розуміння понять, що розглядаються. В подальших викладках різні типи взаємодії проілюстровано прикладами та запропоновано один з варіантів тлумачення та класифікації композиційної взаємодії.

Виклад основного матеріалу. Хоча досліджуване питання відноситься до основних понять ООП, проаналізувавши ряд літературних джерел і статей на професійних форумах, приходимо до висновку, що питання конкретизації типів взаємодії та визначення – який з типів має бути реалізований в тому чи іншому випадку, є досить складними, незважаючи на простоту свого формулювання.

Приведемо деякі тези:

1) На рівні основних визначень поняття ООП: композиція це один із видів взаємодії між об’єктами в ООП (поряд з агрегацією і асоціацією). Але навіть на цьому етапі були знайдені протиріччя, в деяких джерелах автори описують (підводять до думки), що агрегація і композиція це підвиди асоціації. Такий підхід є більш класичним і частіше зустрічається в літературі. В інших альтернативних джерелах розглядаються всі три типи як підвиди взаємодії з різною силою взаємодії.

2) Багато публікацій розглядають питання взаємодії об'єктів на рівні протиставлення «успадкування» і «композиції». І такий підхід передбачає розгляд взаємодії об'єктів вже на більш високому рівні абстракції, тут вже розглядаються не самі механізми ООП, а більше філософія парадигми програмування і об'єктний спосіб мислення.

3) Фактично відсутні чіткі формальні правила і обмеження (більшість сучасних ООП мов досить гнучкі) і конкретний підхід взаємодії між об'єктами при побудові об'єктної ієрархії обирає безпосередньо розробник (архітектор, програміст і т.д.). Питання відповідності тієї чи іншої конфігурації взаємодії об'єктів принципам ООП дуже не однозначне. Хоча ведуться дискусії про переваги того чи іншого способу організації взаємодії між класами (а в підсумку об'єктами), розробник вибирає той чи

інший шлях ґрунтуючись на особистому досвіді та стилі, але також враховує можливості і обмеження, що дають і накладають ці способи.

Виходячи з тез (наведених вище), найбільш доцільним видається спочатку вибрати один з варіантів визначень і чітко визначити - що ми буде розуміти під тим чи іншим терміном, а потім вказати на особливості.

Залежність є найпростішим видом відносин між об'єктами. Залежність вказує на те, що один об'єкт залежить від специфікації (технічних вимог) іншого. Специфікація, опис, визначення, оголошення, технічне завдання, технічні умови, або технічні вимоги - не що інше, як опис поведінки об'єкта. Якщо об'єкти знаходяться в відношенні залежності, то залежний об'єкт тісно прив’язаний до технічних вимог базового об’єкта. У випадку коли специфікація базового об’єкта зміниться, доведеться змінити і залежний об'єкт.

Композиція, агрегація і асоціація всі ці терміни визначають відносини між об'єктами або класами між собою.

В класичних роботах з теорії об’єктно-орієнтованого програмування пропонується п'ять основних типів відносин:

- асоціація;
- агрегація;
- композиція;

і ще два окремих типи:

- успадкування;
- реалізація.

Реалізація - цей тип відносин базується на інтерфейсах, тобто створюються інтерфейси, які основний клас повинен реалізувати.

Порівняння «композиції» та «успадкування», а також аналіз доцільності застосування тієї чи іншої ООП технології в даній статті не розглядаються. Метою статті є аналіз суті понять композиційної взаємодії між класами типу: «асоціація», «композиція», «агрегація», та визначення відмінності між цими типами.

Будемо дотримуватися тієї точки зору, що композиція, агрегація і асоціація все це типи відносин між об'єктами, які відрізняються тільки ступенем сили зв'язку (тобто - підхід де композиція і агрегація не є підтипами асоціації, а всі три типи, що розглядаються, реалізують окремий вид зв'язку).

Асоціація це такий тип, при якому об'єкти будуть посилатися один на одного. При цьому вони залишаються повністю незалежними один від одного.

Наприклад: «СТО» і «МЕХАНІК»

Агрегація це тип відносин, коли один об'єкт є частиною іншого. Агрегація утворює слабкий зв'язок між об'єктами. Всі залежні класи ініціалізуються поза основним об'єктом.

Наприклад: «АВТОМОБІЛЬ» і «ВОДІЙ»

Композиція це тип відносин, при яких один об'єкт може належати тільки іншому об'єкту і нікому іншому. При композиції утворюється сильний (абсолютно жорсткий) зв'язок між об'єктами. При такому типі зв’язку основний об'єкт повністю забезпечує життєвий цикл об'єктів від яких він залежить.

Наприклад: «АВТОМОБІЛЬ» і «ВИГУН».

Відштовхуючись від такого визначення відносин, визначимо деякі особливості «композиції».

Композиція - це концепція, яка моделює відносини. Вона дозволяє створювати складні типи, комбінуючи об'єкти інших типів. Це означає, що клас **Composite (клас-контейнер)** може містити об'єкт іншого класу **Component**. Клас-контейнер включає в

себе виклики конструкторів інших класів. В результаті виходить, що при створенні об'єкта класу-контейнера, також створюються об'єкти включених в нього класів-компонентів.

При взаємодії типу «композиція» можна виділити наступні особливості:

1) класи-компоненти не можуть існувати окремо від контейнера;

2) класи-компоненти не можуть бути створені окремо від класу-контейнера;

*3) при створенні класу-контейнера обов'язково створюються всі екземпляри класів-компонентів;

4) при видаленні класу-контейнера, попередньо, повинні бути видалені всі класи-компоненти

***Цей пункт по різному трактується в різних джерелах, фактично створення неповного набору класів-компонентів не суперечить фундаментально поняттю «композиція»*

Досить чітке розмежування відношень наводиться в нотації UML, для кожної взаємодії запропоновано свій графічний символ (рис. 1)

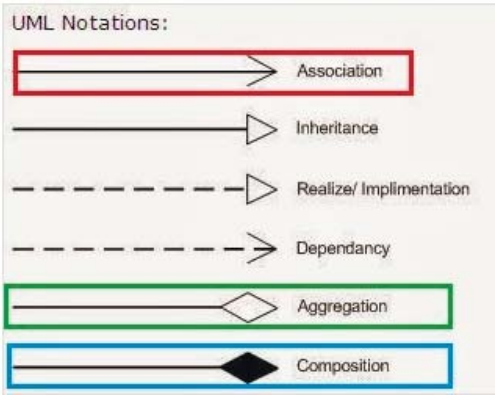

Рисунок 1 – Графічні позначення відношень в нотації UML

Джерело: [10]

UML представляє композицію таким чином:

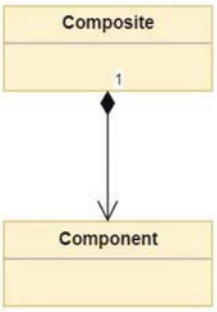

Рисунок 2 – Подання композиції в моделі UML

Джерело: розроблено автором

Проілюструємо композицію наступним прикладом

Питання: Нехай існує клас Family, в який композиційно входять ще три класи Husband, Wife і Child (рис. 3).

1) Чи може Family існувати з неповним набором композуючих класів або вони повинні бути всі в обов'язковому порядку (наприклад, тільки Husband і Wife або тільки Wife і Child)?

2) Чи може об'єкт будь-якого з композуючих класів бути знищений до знищення об'єкта-контейнера або композуючі об'єкти зобов'язані існувати поки існує об'єкт-контейнер?

3) Чи може композуючий клас бути створений не в момент створення об'єкту-контейнера а пізніше, наприклад, у разі виникнення певної умови або створений окремо?

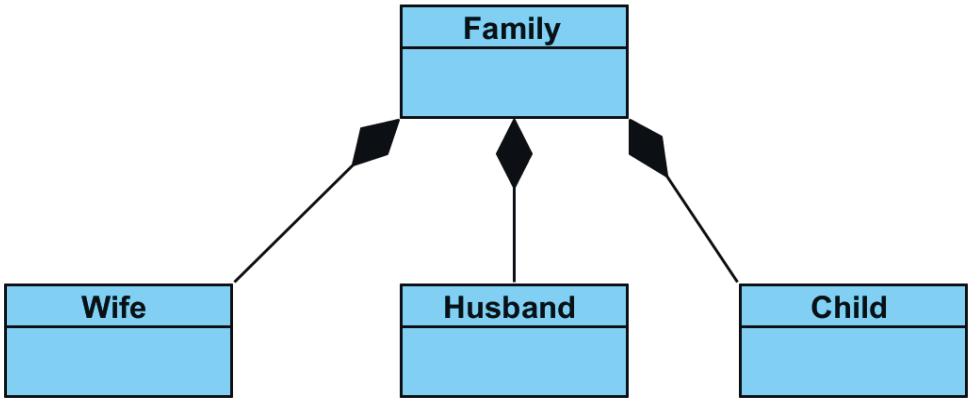

Рисунок 3 – Модель взаємодії Family

Джерело: розроблено автором

З визначення композиції - «включений» (композуючий) об'єкт може існувати тільки як частина контейнера. Якщо контейнер буде знищений, то і включений об'єкт теж буде знищений. Тобто поняття «композиції» однозначно визначає тільки момент знищення, але не момент створення об'єкта, що композиційно включається до класу.

Виходячи з останнього однозначно можна відповісти тільки на питання про творення об’єкту **окремо** - ні, об’єкт що входить в композицію окремо створений бути не може. Всі інші питання не однозначні.

Все залежить від логіки предметної області і, відповідно, конкретної реалізації в коді додатку. Якщо в додатку допускається існування неповних сімей, то звичайно може, а якщо неповні сім’ї заборонено (що досить нелогічно з точки зору моделювання реальної ситуації) - то ні. Але саме по собі використання композиції не накладає ніяких заборон на це. У прикладі з «сім'єю» доречніше використовувати агрегацію, а не композицію. Інакше кажучи об'єкти, що агрегуються (члени сім'ї) повинні створюватися поза об'єктом агрегату (сім'ї). Вони повинні передаватися йому ззовні (конструктори, присвоювання, додавання елемента в колекцію). Резюмуючи, можна сказати наступне: в композиції контейнер повністю керує життєвим циклом об'єктів, що входять в композицію і може створювати/не створювати/знищувати/динамічно замінювати їх в залежності від своєї логіки. При композиції "контейнер" може існувати без вмісту, а ось вміст (члени сім'ї в прикладі) без контейнера не можуть і "знищуються" разом з ним.

Висновки. Гнучкість ООП є і перевагою і недоліком парадигми одночасно, і це конкретне питання про взаємодію об'єктів в черговий раз демонструє цей факт.

У співтоваристві C++ термін «aggregation» використовувався для позначення класу, що визначає певний атрибут для посилання на об'єкти іншого незалежного класу (див. [9]), що в діаграмах класів UML визначено як «association». Термін «composition» використовувався для класів, які визначають об'єкти компонентів для своїх об'єктів, так що при знищенні складеного об'єкта ці об'єкти компонентів також знищуються.

У діаграмах класів UML і «aggregation», і «composition» були визначені як окремі випадки асоціацій, що представляють відносини між «частина»-«ціле». В нотації UML відмінність між «aggregation» і «composition» базується на критерії - чи дозволяється розділити «частину» між двома або більше «цілими». UML визначає «compositions» як такі, що не підлягають спільному використанню (ексклюзивні) частини, в той час як «aggregations» можуть ділитися своїми частинами. Крім того «compositions» визначають залежність життєвого циклу між «цілим» і його «частинами», так що «частини» не можуть існувати без «цілого».

Таким чином, хоча UML помістив терміни «aggregation» і «composition» в конкретний контекст (відносин «частина-ціле»), але авторам нотації не вдалося визначити їх чітким і недвозначним чином, вирішення питання здебільшого покладено на інтуїцію розробників. Однак це не дивно, тому що існує так багато різних властивостей (нюансів реалізації), які можуть мати ці відносини, і кожен з розробників має власну думку - як їх реалізувати.

Список літератури

1. Бублик В.В. Об’єктно-орієнтоване програмування: підруч. К.: ІТкнига, 2015. 624 с.
2. Вайсфельд, М. Объектно-ориентированное мышление. М.: Питер, 2014. 998 с.
3. Язык UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-ое изд. Джим Арлоу, Айла Нейштадт; Исд: Символ-Плюс, 2007.
4. Композиция или наследование: как выбрать, статья Хабр. Автор оригинала: Steven Lowe. URL: https://habr.com/ru/post/325478/ (дата звернення: 28.06.2021)
5. Композиция. Взаимное существование контейнера и композируемого объекта, форум Stack Overflow. URL: https://ru.stackoverflow.com/questions (дата звернення: 27.06.2021)
6. Наследование, композиция, агрегация, статья Хабр. Автор Gamos. URL: https://habr.com/ru/post/354046/ (дата звернення: 29.06.2021)
7. Кун Т. Структура научных революций. М., 2009. 310 с.
8. Хайдаров К.А. Объектно-ориентированное программирование. URL: http://bourabai.kz/alg/oop.htm (дата звернення: 28.06.2021)
9. Aggregation. december 7th, 2007 / last modified by alex on december 21st, 2020. URL: https://www.learncpp.com/cpp-tutorial/aggregation/ (дата звернення: 28.06.2021)
10. Difference between Association, Composition and Aggregation in Java, UML and Object Oriented Programming URL: https://javarevisited.blogspot.com/2014/02/ifference-between-association-vs-composition-vs-aggregation.html (дата звернення: 29.06.2021)
11. Соколов В. В. Применение функциональной и реляционной моделей в объектно-ориентированном программировании. *Information Technology and Security*. 2017. Vol. 5. Iss. 1 (8). С. 54-62.
12. Лавріщева К.М. ПРОГРАМНА ІНЖЕНЕРІЯ. К.: АкадемПеріодика, 2008. 319 с.

References

1. Bublyk, V.V. (2015). *Ob'iektno-oriientovane prohramuvannia* [*Object-oriented programming*] . Kyiv: ITknyha [in Ukraine].
2. Vajsfel'd, M. (2014). *Ob#ektno-orientirovannoe myshlenie* [*Object-oriented thinking*]. Moscow: Piter [in Russian].
3. Dzhim Arlou, Ajla Nejshtadt (2007). *UML 2 and the Unified Process. Practical object-oriented analysis and design*. (2d ed.) . Isd: Simvol-Pljus [in Russian].
4. Steven Lowe. Kompozicija ili nasledovanie: kak vybrat' [Composition or inheritance: how to choose]. Retrieved from https://habr.com/ru/post/325478[in Russian].
5. Kompozicija. Vzaimnoe sushhestvovanie kontejnera i kompoziruemogo ob#ekta, forum Stack Overflow [Composition. Mutual existence of container and composable object, Stack Overflow forum]. *ru.stackoverflow.com.* Retrieved from https://ru.stackoverflow.com/questions [in Russian].
6. Habr. Avtor Gamos. Nasledovanie, kompozicija, agregacija [Inheritance, composition, aggregation]. Retrieved from https://habr.com/ru/post/354046/ [in Russian].
7. Kun, T. (2009). *Struktura nauchnyh revoljucij* [*The structure of scientific revolutions*]. Moskow [in Russian].

8. Hajdarov, K.A. Ob#ektno-orientirovannoe programmirovanie [Object Oriented Programming]. Retrieved from http://bourabai.kz/alg/oop.htm [in Russian].
9. Aggregation. december 7th, 2007 / last modified by alex on december 21st, 2020. Retrieved from https://www.learncpp.com/cpp-tutorial/aggregation [in English].
10. Difference between Association, Composition and Aggregation in Java, UML and Object Oriented Programming. Retrieved from https://javarevisited.blogspot.com/2014/02/ifference-between-association-vs-composition-vs-aggregation.html [in English].
11. Sokolov, V.V. (2017). Primenenie funkcionalnoi i reliacinoi modelei v ob#ektno-orientirovannomprogrammirovanii [Application of functional and relational models in object-oriented programming]. *Information Technology and Security, Vol. 5, Iss. 1 (8)*, 54-62 [in Russian].
12. Lavrishcheva, K.M. (2008). *Programna ingeneriia* [*Software Engineering*]. Kyiv: AcademPeriodika [in Ukraine].

Oleksandr Ulichev, PhD tech. sci., **Kostiantyn Buravchenko,** PhD tech. sci., **Liudmyla Polishchuk,** Senior Lecturer
Central Ukrainian National Technical University, Kropyvnytskyi, Ukraine

Object-oriented Approach in Programming. Compositional in Teraction of Objects

This article aims to consider the types of interaction, in particular the compositional interaction between objects, and to identify the characteristics of individual types. The aim of the article is to clarify the essence of the concepts of different types of relationships between classes. The article proposes a number of theses that illustrate the ambiguity of the interpretation of concepts - types of connections. In the following calculations, different types of interaction are illustrated by examples and one of the options for interpretation and classification of compositional interaction is proposed.

Analysis of a number of literature sources and articles in professional forums leads to the conclusion that the question of specifying the types of interaction and determining - which of the types should be implemented in a given case, is quite complex, despite the simplicity of its formulation. Some theses: 1) At the level of the basic definitions of OOP: composition is one of the types of interaction between objects in OOP (along with aggregation and association). But even at this stage, contradictions were found, in some sources the authors describe (suggest) that aggregation and composition are subspecies of association; 2) Many publications consider the interaction of objects at the level of opposition of "inheritance" and "composition". And this approach involves considering the interaction of objects at a higher level of abstraction, it is not considered the mechanisms of OOP, but rather the philosophy of the programming paradigm and the objective way of thinking; 3) In fact, there are no clear formal rules and restrictions (most modern OOP languages are quite flexible) and a specific approach to interaction between objects in building the object hierarchy is chosen directly by the developer (architect, programmer, etc.). The question of compliance of one or another configuration of the interaction of objects with the principles of OOP is very ambiguous.

The flexibility of the OOP is both an advantage and a disadvantage of the paradigm at the same time, and this particular issue of the interaction of objects once again demonstrates this fact. There are two main interpretations of the relationship, one formed in the Society of C ++ developers (taking into account the peculiarities of OOP in this programming language), the other is based on the principles and specifications of UML. Thus, although UML placed the terms "aggregation" and "composition" in a specific context ("part-whole" relationship), but the authors of the notation failed to define them clearly and unambiguously, the decision is largely left to the intuition of developers.

Object-oriented programming, UML, aggregation, association, composition, objects relationship

Одержано (Received) 12.01.2022 *Прорецензовано (Reviewed) 25.01.2022*
Прийнято до друку (Approved) 31.03.2022