INFORMATION TECHNOLOGY

DOI 10.20535/2411-1031.2022.10.1.261040
УДК 004(942+415.2.043)

ВОЛОДИМИР СОКОЛОВ,
ДМИТРО ШАРАДКІН,
ОЛЕКСІЙ ЦАРЕНОК

МЕТОД ПРОЄКТУВАННЯ СИСТЕМИ КЛАСІВ МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ НА ОСНОВІ АРХІТЕКТУРИ ІНТЕГРАЛЬНИХ ОБ'ЄКТІВ

Представлено метод проєктування системи класів програмного забезпечення, що включає етапи концептуального, логічного та фізичного проєктування класів з реалізацією програмної системи в архітектурі інтегральних об'єктів. В якості вхідних даних для проєктування розглядається модель предметної області, яка отримана за результатами об'єктно-орієнтованого аналізу, що представляється концептуальною, математичною та семантичною моделями. Концептуальна модель визначає сутності та атрибути сутностей і розглядається як основа для проєктування властивостей класів. Математично модель, як динамічна модель поведінки сутностей, формально або неформально задається математичними формулами, алгоритмами, діаграмами або описом реалізації, що є основою для проєктування методів класів. Семантична модель визначає поняття предметної області відносно елементів її моделі і включає ідентифікатори, змістовні назви та описи, що використовуються під час проєктування, програмування та реалізації функціональності системи в термінах предметної області. Метод проєктування, що пропонується, базується на наступних принципах: нормалізація класів (приведення даних до атомарних значень, редукція аргументів функцій та видалення надлишкових функцій), заміна успадкування на логічну композицію та фізичну інтеграцію, декапсуляція (відокремлення функцій фізичної моделі від даних логічної моделі), віртуалізація класів (класи логічної моделі розглядаються як сукупність реляційної моделі даних та функціональної моделі методів фізичних класів), взаємозв'язок моделей (наскрізне відображення концептуальної моделі на логічну та фізичну модель, та реалізацію класів). Суть методу полягає у виконанні послідовних етапів проєктування: нормалізація концептуальної моделі, логічне проєктування класів (перетворення математичної моделі у нормалізовану функціональну модель, проєктування та аналіз характеристик класів), фізичне проєктування класів в архітектурі інтегральних об'єктів (створення класів, відображення логічних класів на фізичні класи, реалізація класів мовою програмування). Розглянуто приклад застосування методу для певної предметної області з демонстрацією результатів проєктування, а також показано підхід на основі мови запитів формулювання задач в термінах предметної області з генерацією вихідного коду мовою програмування. Застосування методу дозволяє підвищити ефективність створення програмних систем за рахунок не надлишковості системи класів та параметрів методів, повторного використання коду та вибору мінімально необхідної множини властивостей та методів під час розв'язання конкретних задач.

Ключові слова: проєктування класів, декапсуляція, модель предметної області.

Постановка проблеми. В теорії об'єктно-орієнтованого аналізу та проєктування програмного забезпечення (ПЗ) описано евристичний підхід щодо створення класів для майбутньої програмної системи, який в основному зосереджується на структурі окремих класів та їх відносин, приділяючи недостатньо уваги саме сукупності класів як системи, а також функціональній складовій класів, хоча саме методи класів визначають загальну функціональність майбутньої програмної системи. Це часто призводить до фрагментарності

та безсистемності проєктування, внаслідок чого збільшуються витрати як під час програмування (за рахунок необхідності допроєктування упущених рішень та дублювання окремих елементів), так і під час виконання (за рахунок завантаження в пам'ять зайвих даних та функцій, що не потрібні для рішення певної задачі). Іншою проблемою є те, що взаємодія між класами також часто розглядається виключно зі статичної точки зору (генералізація, композиція, агрегація), а не з функціональної. Наступною проблемою є факт ігнорування того, що об'єктно-орієнтована програма в решті решт будується з об'єктів, що взаємодіють, а не з класів, а саме цьому питанню приділяється взагалі мало уваги. У багатьох з тих, хто вперше знайомиться з об'єктно-орієнтованим програмуванням (ООП) та UML складається враження, що це окрема технологія, і діаграми класів достатньо для створення програми. Але для правильного розуміння ООП треба усвідомлювати, що це різновид процедурного модульного програмування, в основі якого лежать алгоритми – методи кожного окремого класу, що застосовуються до об'єктів, та алгоритми вищого рівня, які об'єднують певним чином методи класів, зовнішні функції, операції та інструкції в кінцевій програмі.

Звісно, що такі проблеми притаманні універсальному підходу із застосування об'єктно-орієнтованого аналізу та проєктування. Але у випадку використання певної цільової архітектури реалізації, як от архітектури інтегральних об'єктів (АІО), що є розробкою авторів [1], можливі більш продуктивні підходи, які враховують її особливості, що і є основою для створення цього методу. АІО є певним стилем застосування ООП, що поєднує реляційну модель даних та функціональну модель [2] з реалізацією в ООП, а також містить специфічний спосіб взаємодії об'єктів, що надає можливості зменшувати допоміжний програмний код для взаємодії об'єктів, використовувати декларативний підхід до формулювання задач та автоматичну генерацію коду. Саме основу цього методу проєктування і складає орієнтація на АІО в якості цільової архітектури, який реалізується класами активних динамічних сполук (АДС-класами).

Аналіз останніх досліджень та публікацій. Сучасні дослідження з моделювання предметної області та проєктування класів спрямовані на подальше удосконалення мови моделювання UML, створення графічно-текстових нотацій моделей, формальних підходів до проєктування класів та автоматизації генерації програмного коду з моделей вищого рівня.

У роботі [3] наведено узагальнений аналіз застосування виконуваного UML (xUML) для створення моделей, придатних для генерації коду, та зроблено висновки, що:

- для моделювання найзручнішими є діаграми класів, станів та діяльності;
- найпопулярнішими мовами для опису дій є дії UML, C++, Java;
- напрямками вдосконалення є підвищення зрозумілості виконуваних моделей, покращення контролю виконання моделі та пряма компіляція моделі у виконувані файли.

У [4] розглянуто декілька текстових та графічних інструментів для реалізації мови дії ALF для доповнення виконуваної семантики текстами програмного коду, що визначають структуру та поведінку моделі, з автоматичним генеруванням коду мовою Java.

Робота [5] присвячена застосуванню ООП з вбудованою мовою предметної області (EDSL), а також поєднанню ООП, функціональних мов програмування та мови запитів SQL на проміжному рівні моделювання, що покращує розуміння семантики програмного коду та надає можливості його автоматичної генерації, але задача високорівневого проєктування системи класів не розглядається.

У [6] представлено аналіз об'єктно-орієнтованого моделювання, визначено проблеми розуміння семантики діаграм UML під час моделювання предметної області та зроблено акцент на необхідності доповнення поведінкової моделі діаграми класів для більш ефективного застосування UML як засобу концептуального моделювання.

У [7] розв'язується проблема визначення основних класів програми, які складають її суть та визначають розуміння її семантики шляхом ранжування класів з урахуванням структур класів та їх зав'язків, що покращує розробку та супроводження системи класів, їх аналіз та удосконалення. Запропонований метод застосовується саме в процесі супроводження, а не проєктування систем.

У [8] розглядається метод перетворення семантики природньої мови, якою описана модель предметної області, в об'єктно-орієнтовану модель з використанням формальних граматик та семантичного графу, з якої можна отримати діаграму класів та згенерувати рамковий код програми мовою C++. Фактично в запропонованому методі розв'язується проблема опису предметної області та генерування об'єктно-орієнтованої моделі, але питання проєктування системи класів не розглядаються.

У [9] та [10] досліджуються проблеми предметно-орієнтованого проєктування із застосуванням предметно-орієнтованих мов опису моделей, відображення статичних та поведінкових моделей на об'єктно-орієнтовану модель з використанням діаграм UML та можливістю автоматичної генерації програмного коду прототипу системи. В цих роботах акцент робиться саме на перетворенні моделей після виконання проєктування, а власне питання проєктування не розглядаються.

Таким чином, аналіз останніх публікацій свідчить про актуальність та можливість вирішення проблеми проєктування системи класів для певних цільових архітектур.

Метою статті є підвищення ефективності процесів створення та виконання ПЗ на основі АІО за рахунок забезпечення не надлишковості множини класів та параметрів методів класів під час об'єктно-орієнтованого проєктування системи класів предметної області, а також використання мінімальної множини властивостей та методів класів під час виконання.

Виклад основного матеріалу дослідження. Розглянемо формальну постановку задачі проєктування та суть методу проєктування системи класів на основі АІО.

Постановка задачі проєктування. Формальна постановка задачі проєктування представляється в наступному вигляді.

Дано: Модель предметної області (МПО) D, яка задана наступним чином

$$D = \langle C, M, S \rangle, \tag{1}$$

де C – концептуальна модель;

M – математична модель;

S – семантична модель.

Концептуальна модель C задає статичну структуру даних наступним чином

$$C = \left\{ \langle E_i, A_i \rangle \mid i = \overline{1, I} \right\}, \tag{2}$$

де E_i – сутності МПО, що задані їх ідентифікаторами, унікальними в межах МПО;

$A_i = \left\{ A_{ij_i} \mid j_i = \overline{1, J_i} \right\}$ – атрибути сутностей, унікальних в межах сутностей.

Атрибути сутностей визначають склад властивостей майбутніх класів МПО. Для позначення складу атрибутів сутностей можна використовувати форму $E_i(\{A_{ij_i}\})$, наприклад, $Triangle(a,b,c)$ позначає сутність *Triangle* та його сторони *a, b, c*. Для позначення екземпляру (об'єкту) сутності можна використовувати форму *Triangle::T*, а посилання на окремі атрибути об'єкту – *T.a, T.b, T.c*.

Відношення між сутностями задаються виключно композицією (агрегацією), коли в якості атрибуту однієї сутності виступає один або більше об'єктів інших сутностей, наприклад, *Prism(Triangle::T,h,v)* визначає призму з основою трикутник *T*, висотою *h* та об'ємом *v*.

Множину сутностей МПО позначимо як $E = \bigcup_{i=1}^{I} E_i$, множину всіх атрибутів як $A = \bigcup_{i=1}^{I} A_i$.

Математична модель M задається як

$$M = \left\{ \langle E_i, M_i \rangle \mid i = \overline{1, I} \right\}, \tag{3}$$

де $M_i = \left\{ M_{ik_i}(O_{ik_i}) \mid O_{ik_i} \subseteq A_i, k_i = \overline{1, K_i} \right\}$ – множина співвідношень атрибутів O_{ik_i} i-сутності.

Математична модель визначає поведінку об'єктів, задається у формі алгоритмів, формул, функцій, описом реалізації тощо та визначає склад методів майбутніх класів МПО.

Семантична модель S задається наступним чином

$$S = \langle SD, SE, SA \rangle, \tag{4}$$

де $SD = \langle Model, ModelName, ModelDescr \rangle$ – семантика МПО, що містить її ідентифікатор $Model$, назву $ModelName$ та опис $ModelDescr$;

$SE = \left\{ \langle E_i, Ename_i, Edescr_i \rangle \mid i = \overline{1, I} \right\}$ – семантика сутностей, що для кожної i-сутності містить її ідентифікатор E_i, назву $Ename_i$ та опис $Edescr_i$;

$SA = \left\{ \langle E_i, A_{ij_i}, Aname_{ij_i}, Adescr_{ij_i} \rangle \mid i = \overline{1, I},\ j_i = \overline{1, J_i} \right\}$ – семантика атрибутів, що для кожного атрибуту сутності E_i містить ідентифікатор A_{ij_i}, назву $Aname_{ij_i}$ та опис $Adescr_{ij_i}$.

Семантична модель використовується на етапі формулювання вимог, в процесі проєктування та реалізації класів, а також для рішення задач в термінах МПО.

Потрібно:

1) визначити склад та структуру класів логічної об'єктно-орієнтованої моделі

$$Classes = \left\{ \langle ClassName_g, \Pr operties_g, Methods_g \rangle \mid g = \overline{1, G} \right\}, \tag{5}$$

де $ClassName_g$ – ідентифікатор g-класу;

$\Pr operties_g = \left\{ \Pr op_{gq_g} \mid q_g = \overline{1, Q_g} \right\}$ – множина властивостей g-класу;

$Methods_g = \left\{ Method_{gr_g} : \Pr opX_{gr_g} \to \Pr opY_{gr_g} \mid r_g = \overline{1, R_g} \right\}$ – множина методів g-класу, що містить ідентифікатор методу $Method_{gr_g}$, його вхідні $\Pr opX_{gr_g}$ та вихідні $\Pr opY_{gr_g}$ властивості.

2) визначити склад та структуру класів фізичної моделі в архітектурі АІО

$$AClasses = \left\{ \langle AClassName_v, AX_v, AF_v, AY_v \rangle \mid v = \overline{1, V} \right\}, \tag{6}$$

де $AClassName_v$ – ідентифікатор АДС-класу;

AX_v та AY_v – вхідні та вихідні конектори АДС-класу відповідно;

$AF_v : AX_v \to AY_v$ – метод (функція ядра) АДС-класу.

3) визначити спосіб відображення логічної моделі на фізичну модель $Classes \to AClasses$.

Принципи проєктування. Даний метод проєктування базується на наступних принципах.

1. *Нормалізація класів* – приведення властивостей базових класів до атомарних значень, видалення надлишкових аргументів методів та надлишкових методів класів.

2. *Заміна успадкування* на композицію, що реалізується через інтеграцію. Концептуальна модель має використовувати тільки один вид зав'язків між сутностями – композицію (включення одних об'єктів у склад інших), яка у фізичній моделі реалізується інтеграцією (агрегацією за посиланням на властивості об'єктів) згідно АІО. Цей принцип нівелює негативні властивості успадкування, відповідає АІО та підвищує ефективність управління пам'яттю.

3. *Декапсуляція* – відокремлення методів класів фізичної моделі від властивостей класів логічної моделі. В процесі проєктування спочатку створюється логічна структура звичайних класів (властивості та методи), а у фізичній моделі методи відокремлюються та реалізуються АДС-класами. Це забезпечує ефективну маніпуляцію даними та вибір обмеженої множини методів для рішення задач.

4. *Віртуалізація класів* – внаслідок декапсуляції фактично кожний логічний клас розбивається на дві складові: властивості (як реляційна модель даних, що допускає проекцію) та методи (як функціональна модель, що реалізується АДС-класами). Цей принцип орієнтований на рішення задач та дозволяє за рахунок застосування реляційної операції проекції обирати тільки потрібні, а не всі властивості об'єктів, та тільки потрібні методи для

рішення задач. Тобто логічні класи існують тільки віртуально та використовуються для формулювання задач, які розв'язуються композицією методів об'єктів АДС-класів фізичної моделі.

5. *Взаємозв'язок моделей* – формулювання задач може здійснюватися в термінах МПО з використанням семантичної моделі на логічному рівні та віртуальних класів з автоматичним або автоматизованим перетворенням задач у фізичну модель, або представленням задач безпосередньо на фізичному рівні шляхом створення схеми її рішення у вигляді композиту сполук об'єктів АДС-класів. Крім того, відображення методів віртуальних логічних класів на АДС-класи підвищує коефіцієнт повторного використання коду та зменшує витрати на розробку системи.

Метод проєктування. Метод проєктування системи класів МПО полягає у виконанні наступних етапів:

1. Нормалізація концептуальної моделі.
2. Логічне проєктування системи класів:
   2.1. Пертворення математичної моделі у функціональну модель (ФМ).
   2.2. Нормалізація ФМ.
   2.3. Проєктування класів.
   2.4. Аналіз логічної моделі.
3. Фізичне проєктування:
   3.1. Проєктування АДС-класів.
   3.2. Відображення логічної моделі на фізичну модель.
   3.3. Реалізація АДС-класів.

Розглянемо детально кожний з етапів методу проєктування.

1. Нормалізація концептуальної моделі. Суть нормалізації полягає в тому, щоби всі атрибути сутностей були атомарними з урахуванням композиції. Тобто потрібно всі групові атрибути виділити в окремі сутності та включити ці сутності цілком замість групових атрибутів. Така нормалізація дозволить підвищити коефіцієнт повторного використання сутностей та АДС-класів, шо відповідає АІО. В результаті нормалізації формується структура властивостей класів.

2. Логічне проєктування системи класів полягає у створенні структур класів, що складаються з властивостей (атрибутів концептуальної моделі) та методів, що походять з математичної моделі. Логічне проєктування включає наступні кроки.

2.1. Перетворення математичної моделі у функціональну модель – формування множини функцій для кожної сутності як основи майбутніх методів класів:

$$F = \left\{ \left\langle E_i, F_i \right\rangle \mid i = \overline{1,I} \right\}, \tag{7}$$

де $F_i = \left\{ F_{ik_i} : X_{ik_i} \to Y_{ik_i} \mid X_{ik_i} \subset A_i, Y_{ik_i} \subset A_i, k_i = \overline{1,K_i} \right\}$.

2.2. Нормалізація функціональної моделі – забезпечення повної функціональної залежності значень функцій від аргументів (редукція аргументів) та видалення надлишкових функцій за наступними правилами.

Правило 1: жоден з аргументів будь-якої функції не має залежати від підмножини цих аргументів в інших функціях цієї ж сутності (редукція з композицією). Наприклад,

$$F_i : a,b \to c \wedge F_j : a \to b \Rightarrow F_i : a \to c, \tag{8}$$

де у функції F_i видаляється зайвий аргумент *b*, який залежить від *a*, але функція F_i має включати F_j для обчислення необхідного значення *b*.

Правило 2: якщо права частина якоїсь функції залежить від підмножини її аргументів в іншій функції, то ця функція є надлишковою і має бути видалена. Наприклад,

$$F_i : a,b \to c \wedge F_j : a \to c \Rightarrow F_i = \varnothing. \tag{9}$$

У результаті нормалізації буде отримано ненадлишкову функціональну модель.

2.3. Проєктування класів зводиться до створення для кожної сутності нормалізованої концептуальної моделі класу, в якому поєднуються атрибути сутності (стають властивостями) з функціями цієї сутності (стають методами). З урахуванням (2) та (7) маємо систему класів:

$$Classes = \left\{ \left\langle E_i, A_i, F_i \right\rangle \mid i = \overline{1,I} \right\}. \quad (10)$$

2.4. Аналіз логічної моделі – визначення окремих характеристик моделі, таких як:

- *можливі ключі класів* – множина мінімальних наборів властивостей, що визначають всі властивості класу (мінімальні вхідні дані, по яких можна обчислити всю решту значень)

$$Key(E_i) = \left\{ Key_{it} \mid Key_{it} \to A_i, Key_{it} \subset A_i, t = \overline{1,T_i} \right\}; \quad (11)$$

- *входи класів* – множина властивостей, що не залежать від інших властивостей класу (не можуть бути обчислені з інших властивостей, тому можуть бути тільки вхідними значеннями)

$$Input(E_i) = \left\{ Inp_{ix} \mid \neg\exists Y \subset A_i, Y \to Inp_{ix}, Inp_{ix} \in A_i, x = \overline{1, X_i} \right\}; \quad (12)$$

- *виходи класів* – множина властивостей, що залежать від інших властивостей, але від них не залежить жодна інша властивість класу (можуть бути тільки обчислені з інших властивостей, тому не можуть бути вхідними даними)

$$Output(E_i) = \left\{ Out_{iy} \mid \neg\exists X \subset A_i, Out \to X, Out_{iy} \in A_i, y = \overline{1,Y_i} \right\}. \quad (13)$$

Ці характеристики можуть використовуватись для аналізу коректності постановок та оцінки можливості розв'язку задач.

3. Фізичне проєктування полягає у відокремленні (декапсуляції) методів класів логічної моделі від властивостей та їх представленні у вигляді АДС-класів фізичної моделі, а також у відображенні логічної моделі на фізичну та реалізації АДС-класів. Фактично фізична модель реалізуватиме функціональність елементів системи на основі віртуальних класів логічної моделі.

3.1. Проєктування АДС-класів полягає у тому, що кожна функція логічного класу (10) з *унікальним алгоритмом* представляється формальним АДС-класом без прив'язки до конкретного логічного класу з абстрактними ідентифікаторами класу та властивостей, функція ядра якого реалізує відповідний метод, та з урахуванням (6) має наступну структуру:

$$AClasses = \left\{ \left\langle AClassName_v, AX_v, AF_v, AY_v \right\rangle \mid AF_v \in F, v = \overline{1,V}, V \le I \right\}. \quad (14)$$

Така реалізація дозволить деякі АДС-класи використати повторно для тих функцій, що реалізують однаковий формальний алгоритм. Наприклад, якщо є дві функції з однаковим алгоритмом $F_1 : a = b + c$ та $F_2 : d = e + f$, то для них можна створити однин АДС-клас з абстрактною функцією $F : z = x + y$, яка реалізує той самий алгоритм для обох функцій при відповідній підстановці параметрів, що потенційно може зменшити кількість фізичних класів всієї моделі. Саме для цього і виконується наступний пункт проєктування.

3.2. Відображення логічної моделі класів на фізичну модель – створення таблиці відповідності між методами і їх параметрами логічних класів та АДС-класами і їх абстрактними ідентифікаторами (табл. 1). Таблиця має наступну структуру:

Таблиця 1 – Відображення методів логічної моделі на фізичну модель

E	F	$AClassName$	A_i			
			A_{i1}	A_{i2}	…	A_{iJ_i}
E_i	F_{i1}	$AClassName(F_{i1})$	$A(F_{i1}, A_{i1})$	$A(F_{i1}, A_{i2})$	…	$A(F_{i1}, A_{iJ_1})$
	F_{i2}	$AClassName(F_{i2})$	$A(F_{i2}, A_{i1})$	$A(F_{i2}, A_{i2})$	…	$A(F_{i2}, A_{iJ_2})$
	…	…	…	…	…	…
	F_{iK_i}	$AClassName(F_{iK_i})$	$A(F_{iK_i}, A_{i1})$	$A(F_{iK_i}, A_{i2})$	…	$A(F_{iK_i}, A_{i1J_{K_i}})$

де $AClassName(F_{ik_i})$ – ідентифікатор АДС-класу, що реалізує функцію F_{ik_i};

$$A(F_{ik_i}, A_{ij_i}) = \begin{cases} \textit{відповідний ідентифікатор конектору АДС – класу}; \\ \varnothing, \textit{якщо не використовується}. \end{cases}$$

Табл. 1 використовується для заміни звертання до методу логічного класу на звертання до відповідного об'єкту АДС-класу з правильною підстановкою вхідних та вихідних властивостей замість конекторів, а також для застосування семантики до фізичної реалізації.

3.3. Реалізація АДС-класів полягає у створенні АДС-класів заданою мовою програмування, де кожний клас має ідентифікатор, вхідні конектори, вихідні конектори та реалізує функцію класу в методі його ядра. Таким чином, по закінченню проєктування маємо:

- нормалізовану концептуальну модель;
- віртуальну логічну модель;
- фізичну модель.

Приклад застосування методу проєктування. Розглянемо МПО Geometry, концептуальна модель якої містить дві сутності – прямокутний трикутник та призму з основою прямокутний трикутник: $E = \{\langle RTriangle, a, b, c, p, s\rangle, \langle PRism, RTriangle :: T, h, v, Sp\rangle\}$.

Математична модель задана набором формул:

$$M = \left\{ \left\langle RTriangle, a^2 + b^2 = c^2, s = \frac{ab}{2}, p = a + b + c \right\rangle, \langle PRism, v = T.sh, Sp = 2T.s + T.ph\rangle \right\}.$$

Семантична модель:

$$SD = \langle Geometry, Геометрія, Геометричні\_фігури\rangle;$$

$$SE = \begin{Bmatrix} \langle RTriangle, Трикутник, Прямокутний\_трикутник\rangle, \\ \langle PRism, Призма, Призма\_з\_основою\_прямокутний\_трикутник\rangle \end{Bmatrix};$$

$$SA = \begin{Bmatrix} \langle RTriangle, a, Катет\_a, Катет\rangle, \\ \langle RTriangle, b, Катет\_b, Катет\rangle, \\ \langle RTriangle, c, Гіпотенуза\_c, Гіпотенуза\rangle, \\ \langle RTriangle, p, Периметр\_p, Периметр\rangle, \\ \langle RTriangle, s, Площа\_s, Площа\_трикутника\rangle \\ \langle PRism, h, Висота\_h, Висота\_призми\rangle, \\ \langle PRism, v, Об'єм\_v, Об'єм\_призми\rangle, \\ \langle PRism, Sp, Площа\_поверхні\_Sp, Площа\_поверхні\_призми\rangle \end{Bmatrix}.$$

Проєктування. Концептуальна модель прикладу вже нормалізована, тому розглянемо решту етапів проєктування.

Функціональна модель трикутника включатиме функції, що виводяться з формул (самі алгоритми функцій знадобляться на етапі фізичного проєктування):

1) $a^2 + b^2 = c^2 \Rightarrow \{ F_1 : a,b \to c;\ F_2 : a,c \to b;\ F_3 : b,c \to a \}$;

2) $s = \frac{ab}{2} \Rightarrow \{ F_4 : a,b \to s;\ F_5 : a,s \to b;\ F_6 : s,b \to a \}$;

3) $p = a+b+c \Rightarrow \{ F_7 : a,b,c \to p;\ F_8 : a,b,p \to c;\ F_9 : a,p,c \to b;\ F_{10} : p,b,c \to a \}$.

Проводимо нормалізацію ФМ:

$F_1 : a,b \to c \wedge\ F_7 : a,b,c \to p \Rightarrow F_7 : a,b \to p$ (функція F_1 інтегрується в функції F_7);

$F_1 : a,b \to c \wedge\ F_8 : a,b,p \to c \Rightarrow F_8 = \varnothing$ (функція F_8 надлишкова, видаляється);

$F_2 : a,c \to b \wedge\ F_9 : a,p,c \to b \Rightarrow F_9 = \varnothing$ (функція F_9 надлишкова, видаляється);

$F_3 : b,c \to a \wedge\ F_{10} : p,b,c \to a \Rightarrow F_{10} = \varnothing$ (функція F_{10} надлишкова, видаляється).

Остаточний набір функцій має такий склад: $\langle RTriangle, F_1, F_2, F_3, F_4, F_5, F_6, F_7 \rangle$.

Функціональна модель призми має наступний вигляд (обрано обмежений набір функцій):

1) $v = T.s \cdot h \Rightarrow \{ F_1 : T.s, h \to v ;\ F_2 : v, T.s \to h ;\ F_3 : vh \to T.s \}$;

2) $Sp = T.p \cdot h + 2 \cdot T.s \Rightarrow \{ F_4 : T.p, T.s, h \to Sp \}$.

Остаточний набір функцій призами має такий склад: $\langle PRism, F_1, F_2, F_3, F_4 \rangle$.

Проєктування логічної структури класів призводить до наступної структури (рис.1):

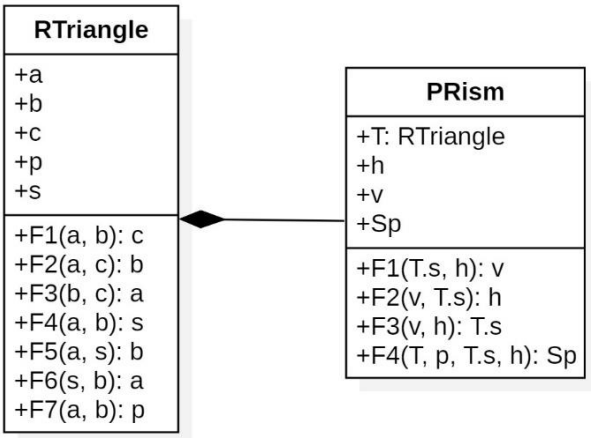

Рисунок 1 – Структура класів логічної моделі

$$Classes = \{\langle RTriangle; a,b,c,p,s; F_1, F_2, F_3, F_4, F_5, F_6, F_7 \rangle, \langle PRism; RTriangle::T, h, v, Sp; F_1, F_2, F_3, F_4 \rangle\}.$$

Аналіз логічної моделі. Можливі ключі:

$Key(RTriangle) = \{(a,b),(a,c),(b,c),(a,s),(b,s)\}$;

$Key(PRizm) = \{(p,h,s),(p,v,s),(p,v,h)\}$;

$Key(Geometry) = \{(a,b,h),(a,c,h),(b,c,h),(a,b,v)\}$ – можливі ключі всієї моделі.

Входи класів:

$Input(RTriangle) = \varnothing$; $Input(PRism) = \{p\}$.

Виходи класів:

$Output(RTriangle) = \{p\}$; $Output(PRism) = \{Sp\}$.

Фізичне проєктування АДС-класів. Створимо АДС-класи для логічного класу *RTriangle*.

1) клас $Hypo : x, y \to z$ реалізуватиме функцію F_1 за алгоритмом $AF_1 : z = \sqrt{x^2 + y^2}$.

2) клас $Leg : x, y \to z$ реалізуватиме функції F_2 та F_3 за алгоритмом $AF_2 : z = \sqrt{x^2 - y^2}$.

3) клас $Area : x, y \to z$ реалізуватиме функцію F_4 за алгоритмом $AF_3 : z = \frac{xy}{2}$.

4) клас $LegArea : x, y \to z$ реалізуватиме функції F_5 та F_6 за алгоритмом $AF_4 : z = \frac{2x}{y}$.

5) інтегральний клас $Per^1 : x, y \to z$ реалізуватиме функцію F_7 за алгоритмом

$$AF_5 : z = x + y + Hypo(x, y).$$

Створимо АДС-класи для логічного класу *PRism*.

1) клас $Volume : x, y \to z$ реалізуватиме функцію F_1 за алгоритмом $AF_6 : z = xy$.

2) клас $HSvolume : x, y \to z$ реалізуватиме функції F_2 та F_3 за алгоритмом $AF_7 : z = \frac{x}{y}$.

3) клас $Sp : x, y, w \to z$ реалізуватиме функцію F_4 за алгоритмом $AF_8 : z = xy + 2w$.

Таким чином, отримуємо наступну структуру фізичних класів *AClasses*:

$$\left\{\begin{array}{l}\langle Hypo; x, y; AF_1; z\rangle, \langle Leg; x, y; AF_2; z\rangle, \langle Area; x, y; AF_3; z\rangle, \langle LegArea; x, y; AF_4; z\rangle, \\ \langle Per; x, y; AF_5; z\rangle, \langle Volume; x, y; AF_6; z\rangle, \langle HSvolume; x, y; AF_7; z\rangle, \langle Sp; x, y; AF_8; z\rangle\end{array}\right\}.$$

Відображення логічної моделі класів на фізичну модель. Покажемо загальну таблицю відображення для всіх класів логічної моделі (табл. 2).

Таблиця 2 – Відображення логічної моделі на фізичну

E	*F*	*AClasses*	$A_{RTriangle}$						A_{Prism}	
			a	*b*	*c*	*p*	*s*	*h*	*v*	*Sp*
RTriangle	F_1	*Hypo*	*x*	*y*	*z*					
	F_2	*Leg*	*y*	*z*	*x*					
	F_3	*Leg*	*z*	*y*	*x*					
	F_4	*Area*	*x*	*y*			*z*			
	F_5	*LegArea*	*y*	*z*			*x*			
	F_6	*LegArea*	*z*	*y*			*x*			
	F_7	*Per*	*x*	*y*		*z*				
Prism	F_1	*Volume*					*x*	*y*	*z*	
	F_2	*HSvolume*					*y*	*z*	*x*	
	F_3	*HSvolume*					*z*	*y*	*x*	
	F_4	*Sp*				*x*	*w*	*y*		*z*

Таким чином, з 10 початкових функцій ФМ після нормалізації залишилось 7, які реалізуються 5 фізичними АДС-класами.

Представлення задач та генерація коду. Одним із способів представлення задач, для рішення яких достатньо створити комбінаційну схему сполуки, може бути SQL-подібна мова запитів, якою зручно формулювати задачі для розробленої моделі на різних рівнях та генерувати програмний код з відображенням логічного запиту на фізичну модель.

Розглянемо приклад задачі: по заданих катетах прямокутного трикутника та висоті призми знайти її об’єм та площу поверхні. Цю задачу можна сформулювати наступним чином.

Лістинг 1 – Задача у формі запиту до логічної моделі

```
select P.v as “Призма::P.О’бєм_v”,  P.Sp as “Призма::P.Площа_поверхні_Sp”
from Geometry.Prism::P
where (P.h = 10) and (P.T.a = 3) and (P.T.b =4)
```

Лістинг 2 – Задача як відображення логічного запиту до фізичної моделі (рис. 2)

```
using  Geometri.Prism.Volume, Geometri.Prism.Sp, Geometry.RTriangle.Area,
       Geometry.RTriangle.Per, Data
select  P_v.z as “Призма::P.О’бєм_v”,  P_Sp.z as “Призма::P.Площа поверхні_Sp”
from   Volume<float>::P_v, Sp<float>::P_Sp, Per<float>::T_p, Area<float>::T_s,
       Data<float>::h, Data<float>::a, Data<float>::b
join    T_s.x = a.X, T_s.y = b.X, T_p.x = a.X, T_p.y = b.X, P_v.x = T_s.z, P_v.y = h.X,
       P_Sp.x = T_p.z, P_Sp.y = h.X, P_Sp.w = T_s.z
where  (h.x as “Призма::P.Висота_h”= 10) and
       (a.x as “Трикутник::T.Катет_a”= 3) and
       (b.x as “Трикутник::T.Катет_b”= 4)
```

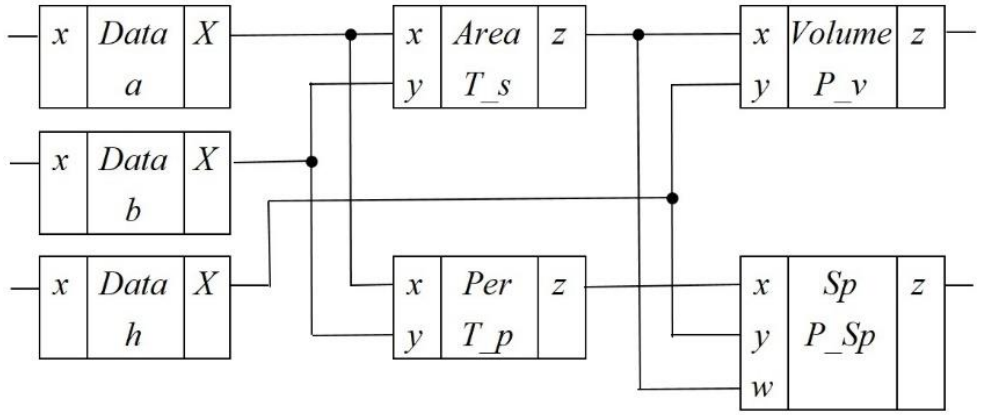

Рисунок 2 – Відповідна схема сполуки об’єктів АДС-класів розв’язку задачі

Лістинг 3 – Згенерований програмний код мовою Сі++

```
#include ”Connectors.h” // реалізує АІО
#include ”Data.h”          // включення класу даних Data
#include ”\\Geometry\\RTriangle\\Area.h”   // включення класу LegArea
#include ”\\Geometry\\RTriangle\\Per.h”     // включення класу Per
#include ”\\Geometry\\Prism\\Volume.h”     // включення класу Volume
#include ”\\Geometry\\Prism\\Sp.h”          // включення класу Sp
#include <iostream>    // стандартні потоки введення/виведення
int main()
{  // створення об’єктів
    Data<float> *h = new Data<float>;
    Data<float> *a = new Data<float>;
    Data<float> *b = new Data<float>;
    Area <float> *T_s  = new Area <float>;
    Per <float> *T_p  = new Per <float>;
    Volume <float> *P_v  = new Volume <float>;
    Sp <float> *P_Sp  = new Sp <float>;
  // створення зв’язків
    Connection(P_v->x, T_s->z);
    Connection(P_v->y, h->X);
    Connection(P_Sp->x, T_p->z);
    Connection(P_Sp->y, h->X);
    Connection(P_Sp->w, T_s->z);
    Connection(T_p->x, a->X);
    Connection(T_p->y, b->X);
    Connection(T_s->x, a->X);
    Connection(T_s->y, b->X);
  // введення даних
    h->x.set_value(10); a->x.set_value(3); b->x.set_value(4);
  // виведення результату
   if(P_v->z.get_value()==NULL)  cout<<”Error”; else
   cout<<"Призма.Об’єм_v = "<<*P_v->z.get_value();
   if(P_Sp->z.get_value()==NULL)  cout<<”Error”; else
   cout<<"Призма.Площа поверхні_Sp = "<<*P_Sp->z.get_value();
  // знищення об’єктів
   delete a; delete b; delete h; delete T_p; delete T_s; delete P_v; delete P_Sp;
 return 0;}
```

Висновки. Розроблений метод проєктування системи класів на основі AIO дозволяє отримати такий позитивний ефект:

- зменшення кількості класів, що підлягають реалізації, до функціонально-необхідного мінімуму за рахунок нормалізації логічної моделі та відображення на фізичну модель з можливістю повторного використання класів;
- зменшення кількості аргументів методів класів за рахунок забезпечення повної функціональної залежності результатів від аргументів за рахунок нормалізації;
- зменшення витрат пам'яті під час виконання програм за рахунок використання тільки потрібних властивостей та методів класів для розв'язку задач за рахунок віртуалізації класів;
- можливість формулювання задач та отримання результатів в термінах предметної області та автоматичної генерації програмного коду для декларативних постановок задач.

У перспективах подальших досліджень планується врахування в методі проєктування моделі задач, які має розв'язувати програмна система для можливості верифікації коректності проєкту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] V. Sokolov, "Architecture of software based on integrated objects", *Information Technology and Security*, vol. 5, no. 2, pp. 51-59, July-December 2017, doi: https://doi.org/10.20535/2411-1031.2017.5.1.120559.

[2] V. Sokolov, "Application of functional and relational models in object-oriented programming", *Information Technology and Security*, vol. 5, no. 1, pp. 54-63, January-June 2017, doi: https://doi.org/10.20535/2411-1031.2017.5.1.120559.

[3] F. Ciccozzi, I. Malavolta, and B. Selic, "Execution of UML models: a systematic review of research and practice", *Software & Systems Modeling*, vol. 18, no. 18, pp. 2313-2360, 2019, doi: https://doi.org/10.1007/s10270-018-0675-4.

[4] T. Buchmann, and A. Rimer, "Unifying Modeling and Programming with ALF", *in Proc. 2nd International Conference on Advances and Trends in Software Engineering (SOFTENG 2016)*, Lisbon, Portugal, pp. 10-15, 2016.

[5] Zhang, Weixin, and Bruno Oliveira. "Shallow EDSLs and Object-Oriented Programming: Beyond Simple Compositionality", *The Art, Science, and Engineering of Programming*, vol. 3, no. 3, article 10, 2019, doi: https://doi.org/10.22152/programming-journal.org/2019/3/10/.

[6] Sabah Al-Fedagh, "Classes in Object-Oriented Modeling (UML): Further Understanding and Abstraction", *IJCSNS International Journal of Computer Science and Network Security*, vol. 21, no. 5, pp. 139-150, May 2021, doi: https://doi.org/10.48550/arXiv.2106.00267.

[7] Du, et al., "COSPA: Identifying Key Classes in Object-Oriented Software Using Preference Aggregation", *IEEE Access*, vol. 9, pp. 114767-114780, 2021, doi: https://doi.org/10.1109/ACCESS.2021.3105475.

[8] T. Kovaliuk, and N. Kobets, "The Object Model of the Subject Domain with the Use of Semantic Networks", *in Proc. 3rd International Conference on Computational Linguistics and Intelligent Systems (COLINS 2019)*, Kharkiv, Ukraine, vol. 2362, 2019, pp. 228-242.

[9] Le, Duc Minh, Duc-Hanh Dang, and Viet-Ha Nguyen, "Generative software module development for domain-driven design with annotation-based domain specific language", *Information and Software Technology*, vol. 120, 2020, doi: https://doi.org/10.1016/j.infsof.2019.106239.

[10] Le, Duc Minh, Duc-Hanh Dang, and Viet-Ha Nguyen, "On domain driven design using annotation-based domain specific language", *Computer Languages, Systems & Structures*, vol. 54, pp. 199-235, 2018, doi: https://doi.org/10.1016/j.cl.2018.05.001.

Стаття надійшла до редакції 12.01.2022.

REFERENCE

[1] V. Sokolov, "Architecture of software based on integrated objects", *Information Technology and Security*, vol. 5, no. 2, pp. 51-59, July-December 2017, doi: https://doi.org/10.20535/2411-1031.2017.5.1. 120559.

[2] V. Sokolov, "Application of functional and relational models in object-oriented programming", *Information Technology and Security*, vol. 5, no. 1, pp. 54-63, January-June 2017, doi: https://doi.org/10.20535/2411-1031.2017.5.1.120559.

[3] F. Ciccozzi, I. Malavolta, and B. Selic, "Execution of UML models: a systematic review of research and practice", *Software & Systems Modeling*, vol. 18, no. 18, pp. 2313-2360, 2019, doi: https://doi.org/10.1007/s10270-018-0675-4.

[4] T. Buchmann, and A. Rimer, "Unifying Modeling and Programming with ALF", *in Proc. 2nd International Conference on Advances and Trends in Software Engineering (SOFTENG 2016)*, Lisbon, Portugal, 2016, pp. 10-15.

[5] Zhang, Weixin, and Bruno Oliveira. "Shallow EDSLs and Object-Oriented Programming: Beyond Simple Compositionality", *The Art, Science, and Engineering of Programming*, vol. 3, no. 3, article 10, 2019, doi: https://doi.org/10.22152/programming-journal.org/2019/3/10/.

[6] Sabah Al-Fedagh, "Classes in Object-Oriented Modeling (UML): Further Understanding and Abstraction", *IJCSNS International Journal of Computer Science and Network Security*, vol. 21, no. 5, pp. 139-150, May 2021, doi: https://doi.org/10.48550/arXiv.2106.00267.

[7] Du, et al., "COSPA: Identifying Key Classes in Object-Oriented Software Using Preference Aggregation", *IEEE Access*, vol. 9, pp. 114767-114780, 2021, doi: https://doi.org/10.1109/ ACCESS.2021.3105475.

[8] T. Kovaliuk, and N. Kobets, "The Object Model of the Subject Domain with the Use of Semantic Networks", *in Proc. 3rd International Conference on Computational Linguistics and Intelligent Systems (COLINS 2019)*, Kharkiv, Ukraine, vol. 2362, 2019, pp. 228-242.

[9] Le, Duc Minh, Duc-Hanh Dang, and Viet-Ha Nguyen, "Generative software module development for domain-driven design with annotation-based domain specific language", *Information and Software Technology*, vol. 120, 2020, doi: https://doi.org/10.1016/j.infsof. 2019.106239.

[10] Le, Duc Minh, Duc-Hanh Dang, and Viet-Ha Nguyen, "On domain driven design using annotation-based domain specific language", *Computer Languages, Systems & Structures*, vol. 54, pp. 199-235, 2018, doi: https://doi.org/10.1016/j.cl.2018.05.001.

VOLODYMYR SOKOLOV,
DMYTRO SHARADKIN,
OLEKSII TSARENOK

METHOD OF DESIGNING THE SYSTEM OF CLASSES OF THE SUBJECT AREA MODEL ON THE BASIS OF THE ARCHITECTURE OF INTEGRATED OBJECTS

The article presents a method of designing the system of software classes, which includes the stages of conceptual, logical and physical design of classes with the implementation of a software system in the architecture of integrated objects. The input data for design is the model of the subject area, which is obtained as the result of object-oriented analysis, which is represented by conceptual, mathematical and semantic models. The conceptual model defines the entities and attributes of entities and is considered as the basis for designing the properties of classes. Mathematical model, as a dynamic model of entity behavior, is formally or informally defined by mathematical formulas, algorithms, diagrams or descriptions of the implementation, which is the basis for designing class methods. The semantic model defines the concepts of subject area in relation to the elements of its model and includes identifiers, meaningful names and descriptions used for the design, programming and implementation of system functionality in terms of subject area. The proposed design method is

based on the following principles: normalization of classes (reduction of data to atomic values, reduction of class methods arguments and removal of redundant class methods), replacement of inheritance by logical composition and physical integration, decapsulation (separation of physical functional model from logical data model), virtualization of classes (logical model of classes are considered as a set of relational data model and physical functional model of class methods), interrelation of models (end-to-end mapping of conceptual model on logical and physical model, and class implementation). The essence of the method is to perform successive stages of design: normalization of conceptual model, logical design of classes (transformation of mathematical model into normalized functional model, design and analysis of class characteristics), physical design of classes in architecture of integrated objects (creation of classes, mapping of logical classes on physical classes, implementation of classes in programming language). Actually, the developed design method leads to the creation of a model focused on solving problems, when instances of entities are not created entirely and only the properties and methods that are needed to calculate the desired results for given input values are used. An example of application of the method for a certain subject area with demonstration of design results is considered, and also the approach on the basis of query language of formulation of problems in terms of subject area with generation of source code in programming language is shown. The application of the method allows to increase the efficiency of creating software systems due to the non-redundancy of the system of classes and parameters of class methods, reuse of code and selection of the minimum required set of properties and class methods when solving specific problems.

Keywords: design of classes, decapsulation, subject area model.

Соколов Володимир Володимирович, кандидат технічних наук, доцент, доцент кафедри кібербезпеки і застасування інформаційних систем і технологій, Інститут спеціального зв'язку та захисту інформації Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна, ORCID 0000-0002-5779-7167, vsokolov@i.ua.

Шарадкін Дмитро Михайлович, кандидат технічних наук, доцент, доцент кафедри кібербезпеки і застасування інформаційних систем і технологій, Інститут спеціального зв'язку та захисту інформації Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна, ORCID 0000-0001-6407-8040, dmsh@ukr.net.

Царенок Олексій Олексійович, заступник завідувача кафедри військової підготовки з навчальної роботи, Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна, ORCID 0000-0002-0491-8228, o.tsarenok@ukr.net.

Sokolov Volodymyr, candidate of technical sciences, associate professor, associate professor at the cybersecurity and application of information systems and technologies academic department, Institute of special communication and information protection of National technical university of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.

Sharadkin Dmytro, candidate of technical sciences, associate professor, associate professor at the cybersecurity and application of information systems and technologies academic department, Institute of special communication and information protection of National technical university of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.

Tsarenok Oleksii, deputy head of the department of military training for educational work, National technical university of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.