УДК 681.03

В.М. Грищенко

МЕТОД ОБ'ЄКТНО-КОМПОНЕНТНОГО ПРОЕКТУВАННЯ ПРОГРАМНИХ СИСТЕМ

Запропоновано комплексний метод проектування компонентних програмних систем, який заснований на концепції двохфазового аналізу та проектування. Завдання першої фази полягає в аналізі предметної області з метою побудови моделі, яка адекватно відображає структуру ПрО та її властивості. До складу задач другої фази входить проектування конкретних компонентів та компонентних систем на основі результатів аналізу ПрО. Наведено необхідну термінологію, формальні моделі, математичний апарат, які визначають концепції та зміст запропонованого методу.

Вступ

Одним з найбільш поширених сучасних методів програмування є компонентний підхід, суть якого полягає у побудові програмних систем шляхом інтеграції окремих самостійних програмних елементів – програмних компонентів [1]. Поняття самостійності компонента означає, що він у загальному випадку створюється без орієнтації на конкретне застосування, тобто передбачається, що компонент буде застосовуватись у різних компонентних системах за умови виконання вимог щодо його функціональних можливостей та властивостей у компонентному середовищі.

Забезпечення умов широкого застосування компонента є однією з головних цілей на етапі його розробки. Крім уніфікації та стандартизації архітектурних, структурних, технологічних характеристик компонента, важливу роль відіграє вибір його функціональності та методів доступу до неї. Це, у свою чергу, потребує більш детального підходу до визначення та специфікації компонента щодо певної предметної області (ПрО), задач, які вирішуються цією ПрО, конкретних функцій обробки даних тощо. Складність цього підходу полягає у тому, що для компонента не існує наперед визначених функціональних вимог, як у випадку специфікації компонентів для конкретної цільової системи. Тому необхідною умовою розробки компонентів багаторазового застосування є умова існування певних об'єктивних концепцій, принципів, критеріїв щодо вибору та специфікації їх функціональних властивостей.

Одним з найбільш обгрунтованих підходів щодо визначення таких концепій та принципів є застосування процесів концептуального моделювання ПрО з відповідним визначенням понять та сутностей предметної області, їх взаємовідношень, атрибутів, поведінки тощо. Умова об'єктивності цього підходу полягає у адекватності подання елементів ПрО щодо понять та об'єктів дійсної реальності, які входять до складу відповідної предметної області. Одним з основних результатів концептуального моделювання є об'єктна модель ПрО, на основі окремих об'єктів чи фрагментів якої визначаються функціональні властивості компонентів, що будуть розроблені у рамках цієї ПрО та застосовані при побудові різних компонентних систем з відповідними функціональними вимогами.

У роботі пропонується метод об'єктно-компонентного проектування програмних систем, який узагальнює та формалізує підхід щодо створення компонентних систем із застосуванням концептуального моделювання ПрО. Сутність метода полягає у формалізації процесу проектування програмної системи, який складається з двох фаз. На першій фазі застосовуються формальний апарат та методи об'єктно-орієнтованого аналізу для концептуального моделювання ПрО та побудови об'єктної моделі. З другою фазою пов'язане безпосереднє проектування окремих компонентів та компонентних програм їз застосуванням компонентно-орієнтованих моделей. При цьому між обома типами моделей встановлюються

еквівалентні відображення, що забезпечують формальне перетворення об'єктного подання в компонентне.

Мета, принципи та сутність фази об'єктного аналізу

Для розробки формалізмів першої фази методу об'єктно-компонентного проектування застосовуються загальні концепції щодо підходів та методів об'єктно-орієнтованого аналізу та проектування [2, 3]. Проте їх сутність та зміст підпорядковані побудові подання об'єктної моделі, яке забезпечує його подальше відображення у компонентну модель.

Головна мета об'єктного аналізу – подати предметну область як множину об'єктів з властивостями та характеристиками, які необхідні й достатні для визначення та ідентифікації окремих об'єктів, а також опису їх поведінки у рамках вибраної системи понять та абстракцій**.**

Визначення поняття об'єкту. Виходячи з умови максимальної адекватності об'єктної моделі дійсній реальності, у методі об'єктно-компонентного проектування визначається поняття об'єктів на основі концепцій теорії Фреге [4]. Кожен з об'єктів подається як трикутник Фреге з наступними складовими (рис.1):

- знак – ідентифікатор, який має властивість позначати певну сутність з дійсної реальності, що може бути предметом, подією, поняттям тощо з певним змістом;
- денотат – сутність з дійсної реальності, яку позначає знак;
- концепт – сутність (семантика) денотату.

Згідно з такою концепцією визначається поняття об'єкта. **Об'єкт** – іменована частина дійсної реальності з певним рівнем абстракції щодо вибраної предметної області та з цілком обумовленою поведінкою. Поданням об'єкта є понятійна структура у вигляді трикутника Фреге.

Процес об'єктного аналізу ПрО виконується згідно з наступними принципами.

Принцип загальності об'єктного визначення. На довільному кроці об'єктного аналізу всі сутності – суть об'єкти.

Наслідок 1. Предметна область, що моделюється, сама є об'єктом.

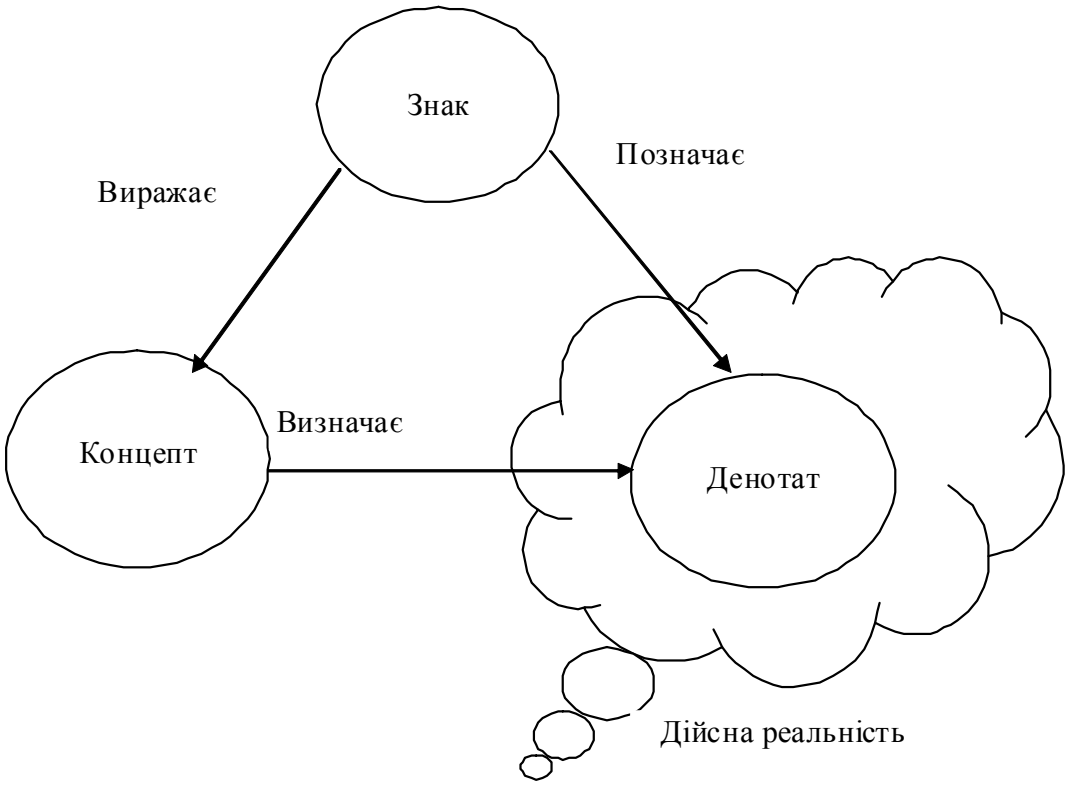

Рис.1. Подання об'єкту як трикутника Фреге

Наслідок 2. Предметна область, що моделюється, може бути окремим об'єктом у складі іншої предметної області (визначається ієрархія предметних областей).

Принцип суттєвості об'єктних відмінностей. На довільному кроці об'єктного аналізу кожний об'єкт є унікальним елементом.

Наслідок. Кожний об'єкт має принаймі одну властивість чи характеристику, яка дозволяє його унікальну ідентифікацію у множині об'єктів.

Принцип об'єктної впорядкованості. На довільному кроці об'єктного аналізу всі об'єкти впорядковані відповідно до відношень між об'єктами.

Наслідок. Кожний об'єкт має принаймі одне відношення з іншим об'єктом, яке забезпечує його впорядкованість в рамках цієї пари об'єктів.

Принцип цілісності об'єктної моделі. На довільному кроці об'єктного аналізу сукупність об'єктів та відношень між ними однозначно визначає об'єктну модель (ОМ) вибраної предметної області для певного рівня абстракції опису цієї ПрО.

Наслідок. На довільному кроці об'єктного аналізу об'єктну модель можна подати у вигляді орієнтованого зв'язного графу, вершинами якого є об'єкти, а дугам відповідають відношення між об'єктами.

Основні формальні рівні абстракції подання об'єктів. Вищенаведені принципи та загальне визначення об'єкта є основою формування послідовності рівнів подання об'єктів та об'єктної системи в цілому. На кожному з цих рівнів застосовується певний математичний апарат [5].

Узагальнюючий рівень. Об'єкт – це клас (клас у цьому випадку розглядається як відповідне математичне поняття згідно з аксіоматичної теорії множин Геделя–Бернайса). Нехай $E=(E_0, E_1, \dots E_n)$ – сукупність об'єктів для предметної області, де E_0 відповідає самій предметній області (принцип загальності об'єктного визначення). Тоді

$$\forall i \exists j[(i>0)\&(\ j\geq 0)\&\ (i \neq j)\&\ (E_i \in E_j)]. \quad (1)$$

Між певними елементами множини E існують відношення належності.

Структурно-впорядковуючий рівень. Об'єкти вже визначені на узагальнюючому рівні і кожен з них подається як множина або елемент певної множини. Виключаючи з E елемент E_0, який не належить іншим елементам, отримуємо множину $E'=(E_1, E_2, \dots E_n)$. Вираз (1) трансформується у такий:

$$\forall i \exists j[(i>0)\&(\ j>0)\&\ (i \neq j)\&\ (E_i \in E_j)]. \quad (2)$$

У виразі (2) кожен з об'єктів є множиною або елементом певної множини (згідно з визначенням множини в теорії Геделя–Бернайса) і до них застосовуються операції теоретико-множинної алгебри. Також цей вираз визначає об'єктні відношення "частка-ціле", екземпляризація, агрегація тощо. Нехай Ω – сукупність теоретико-множинних операцій. Тоді $\Sigma = (E', \Omega)$ визначає алгебраїчну систему для структурно-впорядковуючого рівня.

Характеристичний рівень. Об'єкти вже визначені на структурно-впорядковуючому рівні і для кожного з них формується його концепт. Якщо $E'=(E_1, E_2, \dots E_n)$ – сукупність об'єктів ПрО, а $P'=(P_1, P_2, \dots P_r)$ – множина унарних предикатів, які пов'язані із властивостями об'єктів ПрО, то концепт об'єкта E_i є множиною тверджень, які побудовані на основі предикатів з P', що приймають значення істини для відповідного об'єкта. Концепт $Con_i = \{P_{ik}\}$ за умови, що $P_k(\ E_i) = true$, де P_{ik} є твердженням для об'єкта E_i згідно з предикатом P_k. Цим для об'єктів визначаються їх властивості і характеристики в рамках певної системи абстракцій для конкретної ПрО. Вираз $\Lambda = (E', P')$ визначає алгебраїчну систему (алгебраїчну модель) для характеристичного рівня. Згідно зі структурами концептів між об'єктами визначаються відношення типу "рід–вид" та похідні від нього.

Поведінковий рівень. Об'єкти вже визначені на характеристичному рівні. У відповідності з концепціями формування поведінки [6] на основі сукупності атрибутів об'єктів та їх значень визначаються послідовності станів об'єктів та будуються їх життєві цикли, що відображається у діаграмах переходів станів. Взаємозв'язки між

об'єктами формуються на основі бінарних предикатів, які пов'язані із властивостями об'єктів ПрО, і деталізуються до рівня взаємозв'язків між станами об'єктів. Залежність від параметра часу досягається внесенням у об'єктну модель спеціального об'єкта *Timer*, основне призначення якого полягає у розсилці спеціальних повідомлень поточного часу з певним рівнем дискретності. Кожен об'єкт має відповідний метод, який аналізує отримане значення і виконує перехід до іншого стану або залишає його без змін.

Основні функції об'єктного аналізу. Відповідно до розробленого формального апарату визначено множину базових функцій об'єктного аналізу, які пов'язані з декомпозиційними та композиційними змінами денотатів та концептів об'єктів під час концептуального моделювання ПрО. Множина складається з 10 функцій, які охоплюють трансформації денотатів та концептів у процесі об'єктного аналізу і до складу яких входять зміни, що пов'язані із збільшенням або зменшенням кількості об'єктів (деталізація, екземпляризація, агрегація та ін.) та розширенням або звужуванням концептів об'єктів. Проте ці зміни підпорядковуються певним правилам та умовам, які забезпечують коректність виконання функцій, а також відбуваються у відповідності до наведеного багаторівневого підходу. Визначені наступні функції об'єктного аналізу.

***Декомпозиційні зміни денотату*:**

– відповідна сутність дійсної реальності, яка відповідає певному об'єкту, подається як сукупність однорідних предметів;

– відповідна сутність дійсної реальності, яка відповідає певному об'єкту, подається як сукупність неоднорідних предметів.

***Композиційні зміни денотату*:**

– денотати кількох однорідних предметів подаються як складові певної сутності з вибраної предметної області;

– денотати кількох неоднорідних предметів подаються як складові певної сутності з вибраної предметної області.

***Зміни концептів, які відповідають декомпозиційним змінам денотатів*:**

– концепти нових деталізованих об'єктів формуються на основі концепту початкового об'єкта;

– концепти нових деталізованих об'єктів формуються без врахування концепту початкового об'єкта.

***Зміни концептів, які відповідають композиційним змінам денотатів*:**

– концепт нового композиційного об'єкта формується на основі однакових концептів початкових об'єктів;

– концепт нового композиційного об'єкта формується на основі різних концептів початкових об'єктів.

***Зміни рівня деталізації (абстракції) концептів*:**

– з концепту об'єкта виключається одна чи кілька властивостей або характеристик при формуванні концепту нового об'єкта з однаковим денотатом;

– в концепт об'єкта включається одна чи кілька властивостей або характеристик при формуванні концепту нового об'єкта з однаковим денотатом.

Алгебра об'єктного аналізу. Нехай $E=(E_1, E_2, \dots E_n)$ – множина об'єктів на певному кроці об'єктного аналізу і $E_i = E_i(Name_i, Den_i, Con_i)$, де $Name_i$, Den_i, Con_i – знак (ім'я), денотат та концепт відповідно. $P=(P_1, P_2, \dots P_r)$ – множина предикатів, на основі яких визначаються концепти об'єктів $Con_i = (P_{i1}, P_{i2}, \dots P_{is})$. Введемо наступні базові операції:

1. Декомпозиційна зміна денотату для формування нових однорідних об'єктів:

$$decds(E_i): E_i \rightarrow (E_{i1}, \dots E_{ik}),$$

де $E_{ij} = E_{ij}(Name_{ij}, Den_{ij}, Con_{ij})$; $\forall j\ Con_{ij} = Con_i$; $Den_i = Den_{i1} \cup \dots \cup Den_{ik}$.

2. Декомпозиційна зміна денотату для формування нових неоднорідних об'єктів:

$$decdn(E_i): E_i \rightarrow (E_{i1}, \dots E_{ik}),$$

де $E_{ij} = E_{ij}(Name_{ij}, Den_{ij}, Con_{ij})$; $\forall j\ Con_{ij} = \varnothing$; $Den_i = Den_{i1} \cup \dots \cup Den_{ik}$.

3. Композиційна зміна денотатів однорідних об'єктів для формування нового об'єкта:

$$comds(E_{i1}, \dots E_{ik}): (E_{i1}, \dots E_{ik}) \rightarrow E_i,$$

де $E_i = E_i(Name_i, Den_i, Con_i)$; $\forall j\ Con_i = Con_{ij}$; $Den_{i1} \cup \dots \cup Den_{ik} = Den_i$.

4. Композиційна зміна денотатів неоднорідних об’єктів для формування нового об’єкту:

$comdn(E_{i1}, \dots E_{ik}): (E_{i1}, \dots E_{ik}) \rightarrow E_i$,

де $E_i = E_i(Name_i, Den_i, Con_i)$; $Con_i = \varnothing$; $Den_{i1} \cup \dots \cup Den_{ik} = Den_i$.

5. Розширення концепту існуючого об’єкта. Якщо предикат $P_t \in P$, $P_t \notin Con_i$ і $P_t(E_i)$ приймає значення істини, то

$conexp(E_i, P_t): E_i \rightarrow E_i`$,

де $E_i` = E_i`(Name_i, Den_i, Con_i`)$; $Con_i \cup \cup \{P_t\} = Con_i`$.

6. Звуження концепту існуючого об’єкта. Якщо предикат $P_t \in Con_i$, то

$connar(E_i, P_t): E_i \rightarrow E_i`$,

де $E_i` = E_i`(Name_i, Den_i, Con_i`)$; $Con_i`= = Con_i \setminus P_t$.

Цим для множини функцій об’єктного аналізу побудована алгебра об’єктного аналізу $\Sigma = (E', \Psi)$, де $E'=(E_1, E_2, \dots E_n)$ – множина об’єктів, а $\Psi = \{decds, decdn, comds, comdn, conexp, connar\}$ – множина операцій над елементами E'. Кожна з операцій має певний пріоритет та арність, а також пов’язана з відповідними допустимими змінами денотатів та концептів.

Для обгрунтування переходу від функцій об’єктного аналізу до операцій алгебри об’єктного аналізу сформульовано та доведено наступну теорему.

Теорема 1. Множина операцій Ψ алгебри Σ – повна система операцій щодо функцій об’єктного аналізу.

Правила об’єктного аналізу. Концептуальне моделювання певної ПрО має ітеративний характер і починається з визначення самої ПрО як початкового об’єкта ОМ. На кожній ітерації у відповідності до потреб моделювання застосовуються функції об’єктного аналізу, які наближають структуру та властивість ОМ до кінцевих цілей. Кожна з функцій розглядається як послідовність виконання операцій алгебри об’єктного аналізу, чим підтримується цілісність подання ОМ. Процес завершується формалізованим описом сутностей та моделі ПрО з урахуванням кожного аспекту абстрагування та застосування відповідного математичного апарату. Сформовані наступні правили об’єктного аналізу:

– об’єктний аналіз виконується за умови мінімізації втрати інформації щодо опису дійсної реальності для вибраної предметної області;

– всі зміни, які відбуваються з об’єктною моделлю, відповідають процесам деталізації опису предметної області та визначаються у рамках подання множини об’єктів як сукупності трикутників Фреге;

– кожний крок об’єктного аналізу визначається змінами денотату чи концепту одного або кількох об’єктів з об’єктної моделі;

– нові об’єкти на певному кроці об’єктного аналізу визначаються відповідними змінами у денотатах існуючих об’єктів;

– всі зміни, які відбуваються з об’єктною моделлю, відповідають умовам існування та визначення формальних рівней абстракції подання об’єктів;

– функції об’єктного аналізу визначаються перетвореннями у відповідності з допустимими змінами об’єктної моделі та її окремих елементів;

– на кожному кроці об’єктного аналізу забезпечуються умови цілісності об’єктної моделі.

На основі наведених формалізмів запропонована концепція визначення та впорядкування базової термінології для об’єктно-орієнтованого програмування, суть якої полягає у послідовному визначенні термінів згідно з побудовою відношенн між поняттями. Так, з узагальнюючим рівнем пов’язане визначення лише одного основного терміну – об’єкт. Структурно-впорядковуючий рівень передбачає визначення таких понять, як клас, екземпляр класу, абстрактний клас та ін. З характеристичним рівнем пов’язане визначення наступних термінів: властивість об'єкта, відношення між об'єктами, агрегація; деталізація, класифікація, екземпляризація, асоціація, характеристика об'єкта тощо. На поведінковому рівні визначаються такі терміни: атрибут стану, статичний атрибут стану, динамічний атрибут стану, стан,

простір станів, життєвий цикл об'єкта, метод та ін. [5, 6].

Теоретичні аспекти компонентного проектування

Компонентне програмування є різновидом композиційного програмування, де роль елемента композиції відіграє програмний компонент. Сутність такого програмування визначається як процес створення програмних систем з базових об'єктів, цільових програмних компонентів та компонентів повторного використання (ПВК). Згідно з життєвим циклом побудови таких систем та комплексному методу проектування сама система подається як сукупність взаємодіючих об'єктів, компонентів та ПВК у рамках певного компонентного середовища.

Відповідно до цього фаза компонентного проектування пов'язана з побудовою компонентного подання програми, що створюється у рамках певної ПрО. Для забезпечення переходу від ОМ до компонентного подання встановлено зв'язок між об'єктно-орієнтованим аналізом та компонентним проектуванням програмних систем. Для формалізації процесу компонентного проектування у роботі пропонується відповідний формальний апарат у складі базової термінології (компонент, каркас, компонентна модель, компонентне середовище тощо), моделей основних елементів компонентного програмування, а також зовнішньої та внутрішньої компонентних алгебр.

Для встановлення зв'язків між об'єктно-орієнтованим і компонентним проектуванням розглядаються об'єктне та компонентне подання програми [7], які формуються як об'єктна та інтерфейсна моделі. Об'єктна модель має наступний вираз:

$$OSyst = (OClass, G),$$

де $OClass = \{OClass^i\}$ – множина класів, G – об'єктний граф, у якому вершинами є класи, а дуги визначають об'єктні відношення. Кожен клас з OClass подається як модель

$$OClass^i = (ClassName^i, Method^i, Field^i\},$$

де $ClassName^i$ – ім'я класу; $Method^i = \{Method_j^i\}$ – множина методів класу; $Field^i = \{Field_n^i\}$ – множина змінних, які визначають стан екземплярів класу. Нехай $PField^i \subset Field^i$ – множина зовнішніх змінних (public) класу. Кожному $PField_n^i \in PField^i$ поставимо у відповідність методи $get<PField_n^i>$ и $set<PField_n^i>$ для доступу та модифікації значень змінних, тобто ці змінні подаються як атрибути. Сформуємо нову множину методів

$$IMethod^i = Method^i \cup \{get<PField_n^i>\} \cup \cup \{set<PField_n^i>\}.$$

Кожній множині $IMethod^i$ ставиться у відповідність певний інтерфейс $IFunc^i$, який складається з прототипів методів з $IMethod^i$ та реалізація якого забезпечується функціональністю методів класу та його атрибутів. Розглянемо наступну інтерфейсну модель

$$ISyst = (IFunc, IG),$$

де $IFunc = \{IFunc^i\}$ – множина інтерфейсів, які побудовані для класів з OClass; IG – інтерфейсний граф, що еквівалентний графу G. Тобто IG – граф, у якому вершинами є інтерфейси, а дуги визначають відношення між компонентами відповідно до відношень між їх інтерфейсами.

Таким чином, між графами G та IG існує ізоморфне відображення, а функціональність реалізацій для інтерфейсів $IFunc^i$ еквівалентна функціональності класу $OClass^i$. Для класів з OClass визначаються умови, коли вони дозволяють сформувати їх подання як елементів множини інтерфейсів в інтерфейсному графі. Введемо наступне визначення.

Визначення 1. Декларована в класі змінна називається керованою щодо доступу до її значення з боку інших класів, якщо вона є public-змінна або для неї реалізований доступ за допомогою public-методів класу.

Якщо зовнішня взаємодія з класом відбувається лише за допомогою public-методів та керованих змінних, то для нього реалізується інтерфейсний принцип доступу, тотбо не існує інших можливостей зовнішнього доступу до функціональності класу або його змінних.

Цим розглядом доводиться наступна теорема.

Теорема 2. Для кожної об'єктної моделі OSyst, зовнішня взаємодія з класами якої відбувається на основі public-методів та керованих змінних, існує єдине інтерфейсне подання ISyst з еквівалентною функціональністю.

Необхідно зауважити, що ця теорема визначає умови існування єдиного еквівалентного відображення між об'єктним та інтерфейсним поданнями програми. Між об'єктним та компонентним поданнями відображення існує, але воно не єдине. Ця неоднозначність породжується тим, що певний компонент може мати реалізації для кількох інтерфейсів з ISyst. В частковому випадку, коли кожен з інтерфейсів реалізується у окремому компоненті, буде існувати єдине еквівалентне відображення між об'єктним та компонентним поданнями.

На рис. 2 показано приклад об'єктного та компонентного подання програми. Об'єктна модель OSyst складається з чотирьох класів: E1, E2, E3, E4. Для класів E2, E3, E4 сукупності public-методів та керованих змінних означені як вхідні об'єктні інтерфейси IE2, IE3, IE4 відповідно. У інтерфейсному поданні ISyst їм відповідають компонентні інтерфейси IC2, IC3, IC4 (пунктирні лінії). OC11, OC12, OC2, OC3 – вихідні інтерфейси в компонентній моделі. Між об'єктними та компонентними інтерфейсами встановлено однозначне відображення, але для об'єктної та компонентної моделі такого відображення не існує, бо компонент Comp3 має два вхідних інтерфейси, тобто функціональність класів E3 та E4 реалізована в одному компоненті.

Наведемо основні визначення та моделі для фази компонентного проектування об'єктно-компонентного методу.

Визначення 2. *Програмний компонент* чи просто компонент – це незалежний від мови програмування, самостійно реалізований програмний об'єкт, який забезпечує виконання певної сукупності прикладних сервісів, доступ до яких можливий тільки за допомогою інтерфейсів, що визначають функціональні можливості компонента і порядок звертання до його операцій.

Компонент є відображенням типового рішення щодо певного фрагменту ПрО, має типову архітектуру, структуру, характеристики і атрибути в його інтерфейсній частині для обміну даними в компонентному середовищі. Тобто компонент, поданий таким чином, стає неподільним та інкапсульованим програмним елементом, який задовольняє функціональним вимогам, а також вимогам щодо архітектури системи і компонентного середовища.

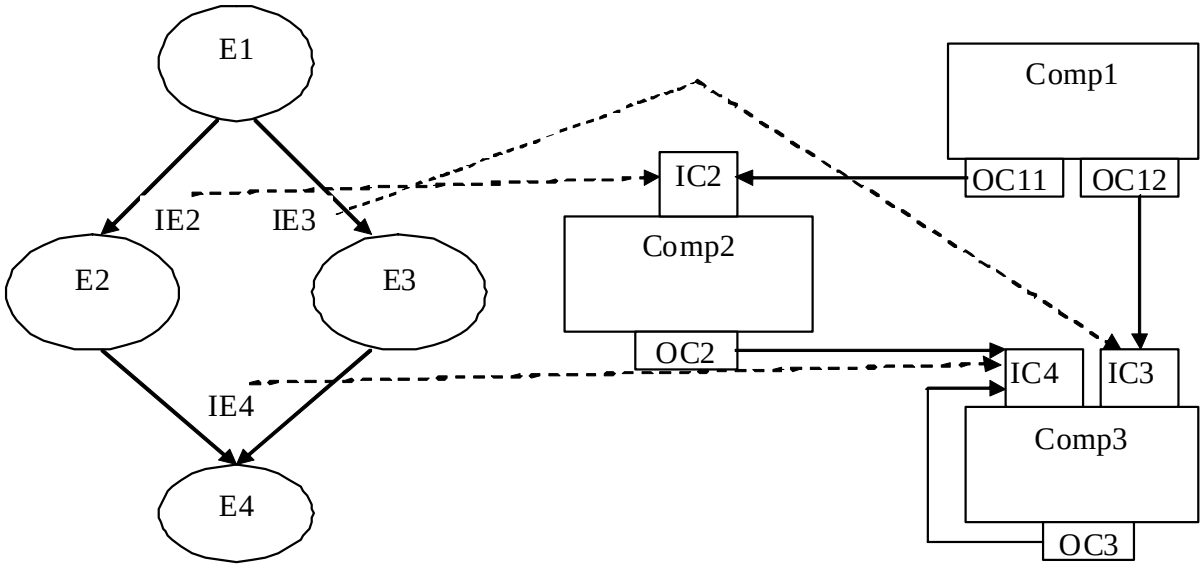

Рис. 2. Об'єктне та компонентне подання програми

Визначення 3. *Компонентна програма* – це сукупність компонентів, що необхідні для забезпечення функціональних та нефункціональних вимог, і яка побудована та функціонує у відповідності до правил створення компонентних конфігурацій і компонентної взаємодії у рамках об'єктних і компонентних моделей.

Модель компонента є наслідком узагальнених типових рішень щодо сутності об'єкта, їх архітектури, структури, властивостей, характеристик, що поступово переходять або відображаються у компонент. Формально модель компонента має вигляд

$$\text{Comp} = (\text{CName}, \text{CInt}, \text{CFact}, \text{CImp}, \text{CServ}), \quad (3)$$

де CName – унікальне ім'я компонента; $\text{CInt} = \{\text{CInt}^i\}$ – множина інтерфейсів, зв'язаних з компонентом; CFact – інтерфейс керування екземплярами компонента; $\text{CImp} = \{\text{CImp}^j\}$ – множина реалізацій компонента; $\text{CServ} = \{\text{CServ}^r\}$ – множина системних сервісів.

Множина $\text{CInt} = \text{CIntI} \cup \text{CIntO}$ складається з вхідних CIntI та вихідних CIntO інтерфейсів. Відмінність між ними полягає у тому, що для вхідних інтерфейсів компонент має власні реалізації, а для вихідних інтерфейсів реалізації знаходяться в інших компонентах. Інтерфейс CFact визначає методи, які необхідні для керування екземплярами компонента (пошук, створення, знищення, тощо).

Модель інтерфейсу має вигляд

$$\text{CInt}^i = (\text{IntName}^i, \text{IntFunc}^i, \text{IntSpec}^i),$$

де IntName^i – ім'я інтерфейсу; IntFunc^i – функціональність (сукупність методів); IntSpec^i – специфікація інтерфейсу (опису типів, констант, інших елементів даних, сигнатур методів тощо).

Необхідною вимогою існування компонента є умова його цілісності:

$$\forall \text{CInt}^i \in \text{CIntI}\ \exists \text{CImp}^j \in \in \text{CImp}\ [\text{Provide}(\text{CInt}^i) \subseteq \text{CImp}^j],$$

де $\text{Provide}(\text{CInt}^i)$ означає функціональність, що забезпечує реалізацію методів інтерфейсу CInt^i. Для взаємодії двох компонентів Comp_1 та Comp_2 існує наступна необхідна умова: якщо $\text{CInt}^i{}_1 \in \text{CIntO}_1$, то повинен існувати $\text{CInt}^k{}_2 \in \text{CIntI}_2$ такий, що:

$$\text{Sign}(\text{CInt}^i{}_1) = \text{Sign}(\text{CInt}^k{}_2) \&$$
$$\& \ \text{Provide}(\text{CInt}^i{}_1) \subseteq \text{CImp}^j{}_2,$$

де Sign(…) означає сигнатуру відповідного інтерфейсу.

Для компонентної моделі певного типу (CFact та CServ є фіксованими) введемо наступні визначення.

Визначення 4. Два компоненти Comp_1 та Comp_2 є тотожніми (рівними), якщо тотожніми є їх відповідні складові. Як наслідок, заміна Comp_1 на Comp_2 не впливає на компонентну програму, до якої належить Comp_1.

Визначення 5. Два компоненти Comp_1 та Comp_2 є еквівалентними, якщо тотожніми є їх множини інтерфейсів та реалізацій. Заміна Comp_1 на Comp_2 не міняє функціональності компонентної програми за умови встановлення відповідності між іменами існуючих та нових компонентів.

Визначення 6. Два компоненти Comp_1 та Comp_2 є подібними, якщо тотожніми є їх множини інтерфейсів. Заміна Comp_1 на Comp_2 зберігає взаємозв'язки компонентів, але функціональність компонентної програми може змінитись.

Основні типи відношень між компонентами.

Відношення успадковування двох компонентів визначається встановленням відношення успадковування для їх вхідних інтерфейсів (аналогічно відношенню успадковування в об'єктно-орієнтованому програмуванні).

Відношення екземпляризації. Екземпляти компонента створюються у відповідності до його певного вхідного інтерфейса CIntI^i. Вираз $\text{CIns}_k{}^{ij} = (\text{IIns}_k{}^{ij}, \text{IntFunc}^i, \text{ImpFunc}^j)$ описує екземпляр компонента Comp, де: $\text{IIns}_k{}^{ij}$ – унікальний ідентифікатор екземпляра, IntFunc^i – функціональність інтерфейса $\text{CIntI}^i \in \text{CInt}$, ImpFunc^j – програмний елемент, що забезпечує виконання реалізації $\text{CImp}^i \in \text{CImp}$.

Відношення контракту. Відношення контракту між компонентами Comp_1 і Comp_2 описується виразом:

$$\text{Cont}_{12}{}^{im} = (\text{CIntO}_1{}^i, \text{CIntI}_2{}^m, \text{IMap}_{12}{}^{im}),$$

де $CIntO_1^i \in CInt_1$ – вихідний інтерфейс першого компонента, $CIntI_2^m \in CInt_2$ – вхідний інтерфейс другого компонента; $IMap_{12}^{im}$ – відображення відповідності між методами, які входять до складу обох інтерфейсів з урахуванням сигнатур та типів даних, що передаються. Відношення контракту існує, якщо компонент $Comp_2$ має реалізацію для інтерфейса $CIntI_2^m$, яка забезпечує виконання функціональності $IntFunc_1^i$ інтерфейса $CIntO_1^i$.

Відношення зв'язування. Якщо між компонентами $Comp_1$ і $Comp_2$ існує відношення контракту $Cont_{12}^{im}$, то між їх екземплярами $CIns_{1k}^{ij}$ = ($IIns_{1k}^{ij}$, $IntFunc^i$, $ImpFunc^j$) та $CIns_{2p}^{mq}$ = ($IIns_{2p}^{mq}$, $IntFunc^m$, $ImpFunc^q$) існує відношення зв'язування щодо контракту $Cont_{12}^{im}$, яке описується виразом $Bind(IIns_{1k}^{ij}, IIns_{2p}^{mq}, Cont_{12}^{im})$.

Модель компонентного середовища має вигляд

$$CE = (NameSpace, IntRep, ImpRep, CServ, CServImp), \quad (4)$$

де $NameSpace = \{CName^m\}$ – множина імен компонентів середовища; $IntRep = \{IntRep^i\}$ – репозитарій інтерфейсів компонентів середовища; $ImpRep = \{ImpRep^j\}$ – репозитарій реалізацій; $CServ = \{CServ^r\}$ – інтерфейс множини системних сервісів; $CServImp = \{CServImp^r\}$ – множина реалізацій для системних сервісів.

Компонентне середовище на рівні моделі розглядається як множина серверів застосувань, де розгортаються компоненти-контейнери, екземпляри яких забезпечують реалізацію функціональності компонента. Взаємозв'язок контейнера з сервером забезпечується через стандартизовані інтерфейси (CFact). Зв'язок між компонентами, які розгорнуті у різних серверах забезпечується реалізаціями інтерфесу CServ.

Визначення 7. Каркасом компонентного середовища називається середовище, для якого сукупності імен компонентів, інтерфейсів та реалізацій – суть порожні множини, тобто

$$FW = (\varnothing, \varnothing, \varnothing, CServ, CServImp).$$

Нехай $FW_1 = (\varnothing, \varnothing, \varnothing, CServ_1, CServImp_1)$ і $FW_2 = (\varnothing, \varnothing, \varnothing, CServ_2, CServImp_2)$ – два каркаси.

Визначення 8. Каркас FW_1 сумісний з каркасом FW_2, якщо існує відображення SMap: $CServ_1 \rightarrow CServ_2$ таке, що $SMap(CServ_1) \subseteq CServ_2$.

Зовнішня компонентна алгебра. Моделі подання компонентів та компонентних середовищ є основою формування зовнішньої компонентної алгебри, яка визначає множину операцій над відповідними елементами і має такий вираз:

$$\Psi = \{ CSet, CESet, \Omega \}, \quad (5)$$

де CSet – множина компонентів, кожен з яких має модель (3); CESet – множина компонентних середовищ, кожне з яких описується виразом (4); Ω – множина операцій. До складу множини Ω входять такі операції (Comp – компонент, CE_1, CE_2, CE_3 – компонентні середовища):

– інсталяція (розгортання компонента) $CE_2 = Comp \oplus CE_1$;

– об'єднання компонентних середовищ $CE_3 = CE_1 \cup CE_2$;

– видалення компонента з компонентного середовища $CE_2 = CE_1 \setminus Comp$.

Для операцій зовнішньої компонентної алгебри сформовані та доведені такі теореми.

Теорема 3. Кожне компонентне середовище CE є результатом виконання послідовності операцій розгортання компонентів, які входять до його складу, в компонентному каркасі:

$$CE = Comp_1 \oplus Comp_2 \oplus \ldots \oplus \oplus Comp_n \oplus FW.$$

Теорема 4. Побудова компонентного середовища не залежить від порядка інсталяції компонентів, які входять до складу цього середовища, тобто:

$$Comp_1 \oplus (Comp_2 \oplus CE) = Comp_2 \oplus (Comp_1 \oplus CE).$$

Теорема 5. Операція об'єднання компонентних середовищ асоціативна:

$$(CE_1 \cup CE_2) \cup CE_3 = CE_1 \cup (CE_2 \cup CE_3).$$

Теорема 6. Операція об'єднання компонентних середовищ комутативна:

$$CE_1 \cup CE_2 = CE_2 \cup CE_1.$$

Теорема 7. Для будь-якого компонентного середовища $CE \cup FW = FW \cup CE = CE$.

Теорема 8. Для довільних компонентних середовищ CE_1 і CE_2 та компонента Comp завжди виконується:

$$Comp \oplus (CE_1 \cup CE_2) = (Comp \oplus CE_1) \cup \cup CE_2 = (Comp \oplus CE_2) \cup CE_1.$$

Теорема 9. Для будь-якого компонента Comp та компонентного середовища CE завжди виконується: $(Comp \oplus CE) \setminus Comp = CE$.

Модель компонентної програми для певного компонентного середовища описується виразом:

$$CP = (CE, Cont, CompO),$$

де CE – компонентне середовище; Cont – множина контрактів для компонентів, що входять до складу CE; CompO – підмножина компонентів з CE, що включають реалізації, для яких відсутні вхідні інтерфейси і які звертаються до інших компонентів за допомогою своїх вихідних інтерфейсів. Компоненти з CompO є вхідними, тобто з них починається виконання програми.

Умова цілісності компонентної програми полягає в існуванні для кожного компонента $Comp_1$ з CE, що має вихідний інтерфейс $CIntO_1^i$, компонента $Comp_2$ з відповідним вхідним інтерфейсом $CIntI_2^m$ і контракт $Cont_{12}^{im} = (CIntO_1^i, CIntI_2^m, IMap_{12}^{im})$ входить до складу множини Cont.

Процес побудови компонентної програми включає: розгортання компонентів і створення компонентного середовища, визначення початкових компонентів та формування множини контрактів згідно з функціональними вимогами до програми.

Внутрішня алгебра компонентного програмування складається з операцій перебудови (трансформації) компонентів середовищі за допомогою функцій рефакторінгу з виконанням умови незмінності інтерфейсів і компонентної моделі, функцій реінжинірингу з незмінністю компонентної моделі та функцій реверсної інженерії (дозволяються довільні трансформації) [8].

На основі семантики, умов та вимог виконання функцій рефакторінгу побудована алгебра рефакторінгу компонентів [9]:

$$\Sigma^{rf} = (CSet, Refac),$$

де $CSet = \{Comp_n\}$ - множина компонентів; Refac = {AddOImp, AddNImp, ReplImp, AddInt} – сукупність операцій рефакторінгу компонентів.

Розглянемо операції рефакторінгу.

Операція AddOImp – додавання нової реалізації для існуючого інтерфейсу:

$$NewComp = AddOImp(OldComp, NewCImp^s, NewCIntO^s)$$

і має семантику:

- $NewCInt = OldCInt \cup NewCIntO^s$ (додаються нові вихідні інтерфейси для нової реалізації);
- $NewCImp = OldCImp \cup \cup \{NewCImp^s\}$ (додається нова реалізація);
- $\exists OldCInt^t \in OldCIntI [Provide(OldCInt^t) \subseteq NewCImp^s]$ (для реалізації, що додається, існує вхідний інтерфейс з відповідною функціональністю).

Операція є асоціативною та комутативною. Умова цілісності компонента зберігається.

Операція AddNImp – додавання нової реалізації, вхідний інтерфейс для якої не входить до складу множини інтерфейсів компонента:

$$NewComp = AddNImp(OldComp, NewCImp^s, NewCIntO^s)$$

і має семантику:

- $NewCInt = OldCInt \cup NewCIntO^s$ (додаються нові вихідні інтерфейси для нової реалізації);
- $NewCImp = OldCImp \cup \{NewCImp^s\}$ (додається нова реалізація).

Операція є асоціативною та комутативною. Умова цілісності компонента зберігається.

Операція ReplImp – заміщення існуючої реалізації новою без зміни вхідного інтерфейсу:

$$NewComp=ReplImp(OldComp,NewCImp^s, NewCIntO^s,OldCImp^r,OldCIntO^r)$$

і має семантику:

- $NewCInt=OldCInt \cup \{NewCIntO^s\} \setminus \{OldCIntO^r\}$ (додаються вихідні інтерфейси для нової реалізації і видаляються вихідні інтерфейси старої реалізації);

- $NewImp = OldImpl \cup \{NewCImp^s\} \setminus \{OldCImp^r\}$ (додається нова реалізація і видаляється стара).

Умовою виконання операції є незмінність функціональності усіх інтерфейсів, що пов'язані зі старою реалізацією, після заміщення її новою:

$$\forall OldCInt^t \in OldCIntI$$
$$[(Provide(OldCInt^t) \subseteq OldCImp^r) \Rightarrow$$
$$(Provide(OldCInt^t) \subseteq NewCImp^s)] \vee$$
$$\exists OldCImp^j \in (OldCImp \setminus \{OldCImp^r\})$$
$$[Provide(OldCInt^t) \subseteq OldCImp^j].$$

Лема 1. Операція заміщення існуючої реалізації новою з вищевизначеними умовою та семантикою зберігає цілісність компонента.

Операція AddInt – додавання нового вхідного інтерфейсу для існуючої реалізації:

$$NewComp = AddInt(OldComp, NewCIntI^q)$$

і має семантику:

- $NewCInt = OldCInt \cup \{NewCIntI^q\}$ (додається новий вхідний інтерфейс);

- $\exists OldCImp^s \in OldCImp$ $[Provide(NewCIntI^q) \subseteq OldCImp^s]$ (для нового інтерфейса існує реалізація з відповідною функціональністю).

Лема 2. Операція додаванням нового вхідного інтерфейсу для існуючої реалізації з вище визначеними умовою та семантикою зберігає цілісність компонента.

Наведені операції є базовими для більш складних. Наприклад, додавання реалізації разом з новим вхідним інтерфейсом визначаються наступною суперпозицією операцій рефакторингу:

$$NewComp =$$
$$= AddInt(AddNImp(OldComp,NewCImp^s, NewCIntO^s), NewCIntI^q).$$

Розгляд семантики операцій та умов виконання доводить наступну теорему.

Теорема 10. Алгебра рефакторингу компонентів $\sum^{rf}$ = (CSet, Refac) є повною та незаперечною.

Зв'язок зовнішньої та внутрішньої компонентних алгебр. Згідно з теоремою 10 результатом операцій рефакторингу або суперпозицій операцій є певний компонент Це свідчить про можливість зв'язку зовнішньої компонентної алгебри з алгеброю рефакторингу. Множина CSet складається з компонентів репозитарію і різних модифікацій компонентів як результатів виконання операцій рефакторингу, тобто у операціях зовнішньої компонентної алгебри замість компонентів можуть застосовуватись їх модифікації. Наприклад, для певної компонентної моделі компонентна конфігурація, яка складається з двох компонентів і для другого компонента додаються реалізація та вхідний інтерфейс, описується таким виразом:

$$CE = Comp_1 \oplus$$
$$\oplus AddInt(AddNImp(Comp_2, NewCImp^s, NewCIntO^s), NewCIntI^q) \oplus FW.$$

Моделі реінжинірингу та реверсної інженерії. Внутрішня компонентна алгебра реінжинірингу компонентів $\sum^{re}$ = (CSet, Reeng) може бути побудована лише за умови порушення цілісності подання компонентів. Крім операцій рефакторингу (Refac) до множини Reeng входять операції, які видаляють з компонента існуючий інтерфейс або змінюють його сигнатуру. Це порушує умову цілісності, бо інші компоненти, які звертаються до нього, не зможуть мати доступу до необхідної функціональності. Виходячи з таких умов, доцільно замість алгебри реінжинірингу до складу формальних методів компонентного програмування включити модель реінжинірингу.

Модель реінжинірингу компонентів має вигляд M_{Reeng} = (CSet, Reeng). Семантика операцій з Reeng може полягати у одночасній трансформації не тільки цільового компонента, а й інших. Наприклад,

при зміні сигнатури вхідного інтерфейсу необхідна одночасна зміна інших компонентів, в яких існують звернення до методів цього інтерфейсу.

Аналогічним чином, формується відповідна модель реверсної інженерії M_{Revers} = (CSet, Revers), при цьому Reeng $\subset$ Revers. Особливість множини Revers полягає у тому, що вона не визначена повністю (кількість операцій є необмеженою), а дозволяє лише класифікацію операцій. Наприклад, зміни якісних характеристик компонентів можуть виконуватись довільним способом і відповідні операції складають сукупність операцій для зміни певного показника якості (сам показник є класифікаційною ознакою у системі класифікації множини Revers).

Таким чином, алгебра Σ^{rf} та моделі M_{Reeng} і M_{Revers} складають формальний апарат аналізу внутрішніх методів еволюції компонентів.

Висновок

У роботі запропоновано комплексний метод аналізу і проектування компонентних програмних систем. Цей метод заснований на концепції двохфазового аналізу та проектування відповідно до головних завдань, що виникають на початкових етапах життєвого циклу компонентних систем. Завдання першої фази полягає в аналізі предметної області з метою побудови моделі, яка адекватно відображає структуру ПрО, її об'єкти та відношення між ними тощо. Головне завдання другої – проектування конкретних компонентів та компонентних систем за результатами аналізу ПрО. Метод узагальнює поняття об'єктів, як елементів дійсної реальності шляхом концептуального моделювання та об'єктно-орієнтованого аналізу предметної області із застосуванням математичних формалізмів на різних рівнях подання об'єктів та об'єктної моделі в цілому, а також послідовно визначає кроки у проектуванні об'єктів, починаючи з їх розгляду як окремих сутностей ПрО і закінчуючи повним поданням з урахуванням характеристик і поведінки. На основі результатів аналізу предметної області будується відображення ОМ у компонентні моделі шляхом формування функціональних інтерфейсів та розподілом їх між конкретними компонентами. Наведені визначення понять об'єкта та компонента, сформовані компонентні моделі, побудована алгебра об'єктного аналізу, а також зовнішня і внутрішня компонентні алгебри складають теорію методу об'єктно-компонентного аналізу та проектування програмних систем.

1. *Грищенко В.Н., Лаврищева Е.М.* Методы и средства компонентного программирования // Кибернетика и системный анализ.– 2003.– № 1.– С. 39–55.
2. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. - М.: "Издательство Бином", СПб.: "Невский диалект", 2001. – 560 с.
3. *Мейер Б.* Объектно-ориентированное конструирование программных систем. – "Русская Редакция", 2005. – 1232 с.
4. *Фреге Г.* Логика и логическая семантика. – М.: Аспект-пресс, 2000. – 512 с.
5. *Грищенко В.Н.* Подход к формализации объектно-ориентированной методологии // Проблемы программирования. – 1997. – № 1.– С. 33–39.
6. *Грищенко В.М.* Підхід до аналізу поведінки об`єктних систем при моделюванні предметних областей // Проблеми програмування. – 1998. – № 4. – С. 67–75.
7. *Грищенко В.Н.* Формальные модели компонентного программирования // Проблемы программирования. – 2003. – № 2.– С. 42–57.
8. *Грищенко В.Н.* Методи еволюції програмних компонентів для їх повторного застосування// Проблеми програмування. – 2004. – № 2–3. – С. 215–222.
9. *Грищенко В.Н.* Алгебраїчна модель рефакторінгу компонентів // Проблеми програмування. – 2003. – № 4.– С. 43–53.

Отримано 19.04.2007

Про автора:

Грищенко Володимир Миколайович,
кандидат фізико-математичних наук.

Місце роботи автора:

Інститут програмних систем НАН України, 03187, Київ-187,
проспект Академіка Глушкова, 40.
Тел. (044) 522 4656.