Методичні вказівки
до самостійної роботи студентів
з проектування UML діаграм
в ході виконання курсових робіт з дисципліни
«Об’єктно-орієнтоване програмування»
для студентів спеціальності 121 –
«Інженерія програмного забезпечення»

Міністерство освіти і науки України
Вінницький національний технічний університет

Методичні вказівки до самостійної роботи студентів з проектування UML діаграм в ході виконання курсових робіт з дисципліни «Об'єктно-орієнтоване програмування» для студентів спеціальності 121 − «Інженерія програмного забезпечення»

Електронне видання комбінованого
(локального та мережного) використання

Вінниця
ВНТУ
2021

Рекомендовано до видання Методичною Радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 4 від 23.12.2020 р.)

Рецензенти:
В. О. Денисюк, кандидат технічних наук, доцент
І. Р. Арсенюк, кандидат технічних наук, доцент

Методичні вказівки до самостійної роботи студентів з проектування UML діаграм в ході виконання курсових робіт з дисципліни «Об'єктно-орієнтоване програмування» для студентів спеціальності 121 − «Інженерія програмного забезпечення» [Електронний ресурс] / Уклад. Д. І. Катєльніков, О. О. Дудник, А. В. Денисюк – Вінниця : ВНТУ, 2021. – 28 с.

У методичних вказівках наведено основні теоретичні дані до виконання самостійної роботи з проектування UML діаграм, які зустрічаються в ході виконання курсових робіт з дисципліни «Об'єктно-орієнтоване програмування».

Методичні вказівки розроблено відповідно до навчальної програми дисципліни «Об'єктно-орієнтоване програмування».

ЗМІСТ

ВСТУП

UML (англ. Unified Modeling Language) − уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розроблення програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, яка називається UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

Перша версія (1.0) UML вийшла 13 січня 1997 р., вона була створена консорціумом UML Partners за запитом Object Management Group (OMG) − організації, відповідальної за прийняття стандартів в галузі об'єктних технологій і баз даних. Після обговорення, у вересні 1997 року, версія 1.1 UML була представлена на голосування в OMG. Розробку UML підтримали і вже тоді використовували як стандарт такі гранди ринку інформаційних технологій, як Microsoft, IBM, Hewlett-Packard, Oracle, DEC, Sybase, Logic Works й інші.

В UML використовується 14 видів діаграм (для уникнення непорозумінь, також наведено англомовні назви):

Structure Diagrams:

- Class diagram
- Component diagram
- Composite structure diagram
  - Collaboration (UML2.0)
- Deployment diagram
- Object diagram
- Package diagram

Behavior Diagrams:

- Activity diagram
- State Machine diagram
- Use case diagram

Interaction Diagrams:

- Collaboration (UML1.x) / Communication diagram (UML2.0)
- Interaction overview diagram (UML2.0)
- Sequence diagram
- UML Timing Diagram (UML2.0)

Структурні діаграми:

- Класів
- Компонент
- Композитної/складеної структури
  - Кооперації (UML2.0)
- Розгортування
- Об'єктів
- Пакетів

Діаграми поведінки:

- Діяльності
- Станів
- Прецедентів

Діаграми взаємодії:

- Кооперації (UML1.x) / Комунікації (UML2.0)
- Огляду взаємодії (UML2.0)
- Послідовності
- Синхронізації (UML2.0)

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

Діаграми дають можливість подати систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код.

Основною причиною використання мови UML є спілкування розробників між собою [1].

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML чудово зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки.

Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і спрямувати зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розроблення документації.

UML необхідний:

- керівникам проектів, які керують розподілом завдань і контролем за проектом
- проектувальникам інформаційних систем які розробляють технічні завдання для програмістів;
- бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії;
- програмістам які реалізують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

1 ДІАГРАМИ СТРУКТУРНОГО МОДЕЛЮВАННЯ

Класи (Class) є основними будівельними блоками об'єктно-орієнтованої системи при проектуванні програмної системи і описують множини об'єктів з однаковими властивостями і поведінкою[1, 2]. Класи характеризуються спільними атрибутами, операціями, зв'язками і семантикою. Клас є **концептуальним виглядом** цілої множини об'єктів [3, 4, 5].

Вимоги до проектування класів. Класи використовуються для побудови словника системи і включають абстракції, що є частиною проблемної області та інші абстракції, на яких ґрунтується реалізація ПЗ. Класи можна застосовувати для описування концептуальних, програмних та апаратних сутностей. Добре структуровані класи мають чітко окреслені границі і формують збалансований розподіл обов'язків у системі. Проектування системи охоплює ідентифікацію сутностей, важливих для конкретного вигляду її реалізації, що формують **словник системи**. *При проектуванні автомобіля важливо знати, якими будуть двері, вікна, сидіння, освітлення тощо. Кожна із цих сутностей відрізняється від іншої і має свої властивості. Двері мають певну висоту, ширину, форму, кріплення, отвір для вікна, ручку. Вікна характеризуються матеріалом, кріпленням, свободою руху та ізоляцією, їх основна функціональність – пропускати сонячне світло. Сидіння характеризуються формою та можливістю її зміни, матеріалом оббивки, кріпленням ременів безпеки тощо. Окремі двері, вікна, крісла не існують самі по собі – це конкретні екземпляри цих сутностей, з'єднаних між собою. Ці сутності і зв'язки залежать від використання дверей, вікон, дизайнерського вирішення тощо. Для машинобудівника важливим є розташування корпуса, дверей, вікон, двигуна, паливного бака, труб, електричних кабелів тощо. Користувач авто не обов'язково зверне увагу на такі речі (за винятком видимих порушень: не можна вмістити ноги, голова впирається в стелю).*

В UML сутності моделюються класами, що є їх абстракціями і частиною словника системи. Клас концептуально описує цілу множину об'єктів (а не індивідуальний об'єкт: «двері» є клас об'єктів зі спільними властивостями і поведінкою: висота, довжина, форма, кріплення тощо. Конкретні двері у авто є реальним об'єктом). Створювані UML-абстракції безпосередньо можна виражати мовою ООП. Графічна UML-нотація класу (рис. 1) дозволяє візуалізувати абстракцію незалежно від мови ООП, підкресливши її найважливіші частини: ім'я, атрибути і операції класу (опису множини об'єктів з однаковими атрибутами, операціями, зв'язками і семантикою). Ім'я (name) є унікальною назвою кожного класу у вигляді текстового рядка, що відрізняє його від інших і відображає його сутність. Окреме ім'я класу є простим, а ім'я класу з префіксом – з іменем пакета, в який включено клас, є кваліфікованим.

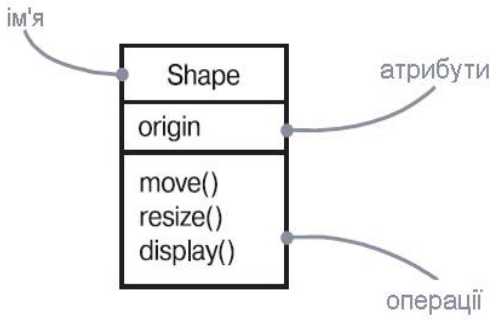

Рисунок 1 − Загальна нотація класу

Атрибут (attribute) – іменована властивість класу, що описує діапазон значень, які може набувати екземпляр атрибута [2]. Клас може мати будь-яке число атрибутів або не мати жодного. Атрибут є певною властивістю модельованої сутності, спільної для усіх об'єктів класу (у кожного співробітника є ім'я, адреса, номер телефону, дата народження). Атрибут є абстракцією виду даних або стану, яким може бути наділений об'єкт класу. У кожен певний момент об'єкт класу характеризується конкретними значеннями (екземплярами) атрибутів. Атрибути перераховані в розділі нотації безпосередньо під іменем класу. Специфікацію атрибута можна уточнити, вказавши його тип і початкове значення (рис. 2).

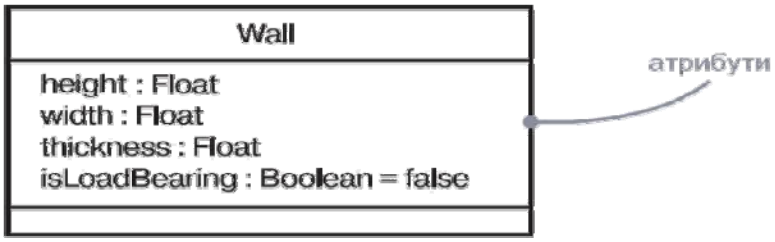

Рисунок 2 − Атрибути з їхніми типами

Операція (operation) – це реалізація послуги об'єкта класу, на яку може бути запит з боку іншого об'єкта чи актора, щоб викликати певну поведінку об'єкта класу [2, 3, 6]. Операція є абстракцією того, що можна зробити з конкретним об'єктом або з усіма об'єктами класу. Клас може мати будь-яке число операцій або не мати жодної. Виклик операції об'єкта може змінювати його дані (стан). Список операції класу вказуються в нотації безпосередньо під списком атрибутів (рис. 3). Для імені операції використовується коротке дієслово або дієслівний зворот, що відповідає певній поведінці классу [6]. Кожне слово в операції потрібно писати з великої букви.

Операцію можна специфікувати повною сигнатурою, що включає ім'я, тип, значення за замовчуванням усіх параметрів, а також тип (клас) значення, що повертається операцією.

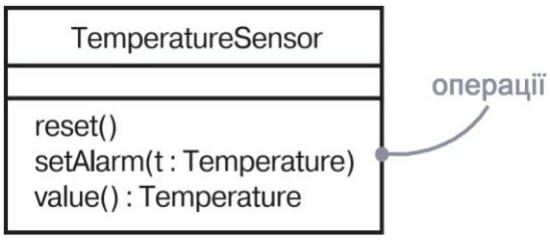

Рисунок 3 − Операції, їх сигнатури

Керування реалізацією класа здійснюється через використання засобів видимості класа, приховуючи певні деталі реалізації і роблячи видимими тільки ті властивості, які необхідні йому для виконання своїх обов'язків. Це основа інкапсуляції (приховання інформації), що забезпечує побудову стійкої системи. Якщо властивість класа не визначена явно символом видимості, то за замовчуванням береться public.
Область дії екземпляра й статична область дії. Область дії (scope) властивості (атрибута, операції) класа вказує на те, чи кожен екземпляр класа має власне значення цієї властивості, чи повинно бути тільки одне її значення для всіх екземплярів класа (статичні члени класу в C++). В UML є два типи області дії: instance – область дії екземпляра (exemplar scope): кожен екземпляр класа має власне значення цієї властивості. Це є варіант за замовчуванням (не вимагає додаткової нотації); static – статична область дії – область дії класу (class scope): є тільки одне значення властивості для всіх екземплярів класа. Атрибут зі статичною областю дії (class scope) позначається підкресленням (рис. 4). Для відображення області дії екземпляра не використовуються ніякі доповнення. Більшість властивостей (атрибутів і операцій) класів, що моделюються, мають область дії екземпляра. У статичній області дії перебувають закриті атрибути, однакові для всіх екземплярів класу. Нестатична операція екземпляра має неявний параметр, який вказує на об'єкт, що її викликає. Статичні операції прив'язані до класу, а не до екземплярів і використовуються для роботи зі статичними атрибутами.

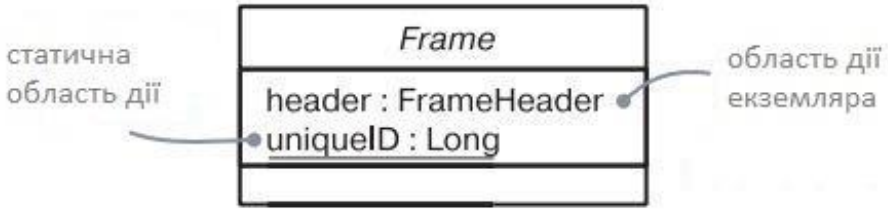

Рисунок 4 − Область дії атрибута, операції

Абстрактні, листові й поліморфні елементи (класи). Зв'язки узагальнення використовуються для моделювання структури класів із загальнішими абстракціями, розташованими на вершині ієрархії і

детальнішими – внизу. У межах такої ієрархії деякі класи часто визначаються як абстрактні – вони не можуть мати прямих (безпосередніх) екземплярів (хоча екземпляри його нащадків можуть існувати в будь-якій кількості). Класи Icon (Піктограма), Rectangularlcon (Прямокутна Піктограма) і Arbitrarylcon (Піктограма Довільної Форми) (рис. 5) – абстрактні класи. На відміну від них, конкретні класи Button (Кнопка) і Okbutton (Кнопка ОК) можуть супроводжуватися екземплярами.

Новий клас створюється з метою успадкування його властивостей від інших, більш загальних класів, а також щоб він мав нащадків – більш спеціалізовані класи, що успадковують його властивості (це нормальна семантика класів у UML). Однак можуть бути випадки, що клас не повинен мати нащадків. Такий клас називається листовим (якого заборонено успадковувати) і позначається в UML властивістю leaf, записаним під іменем класу. Клас Okbutton є листовим класом (рис. 5).

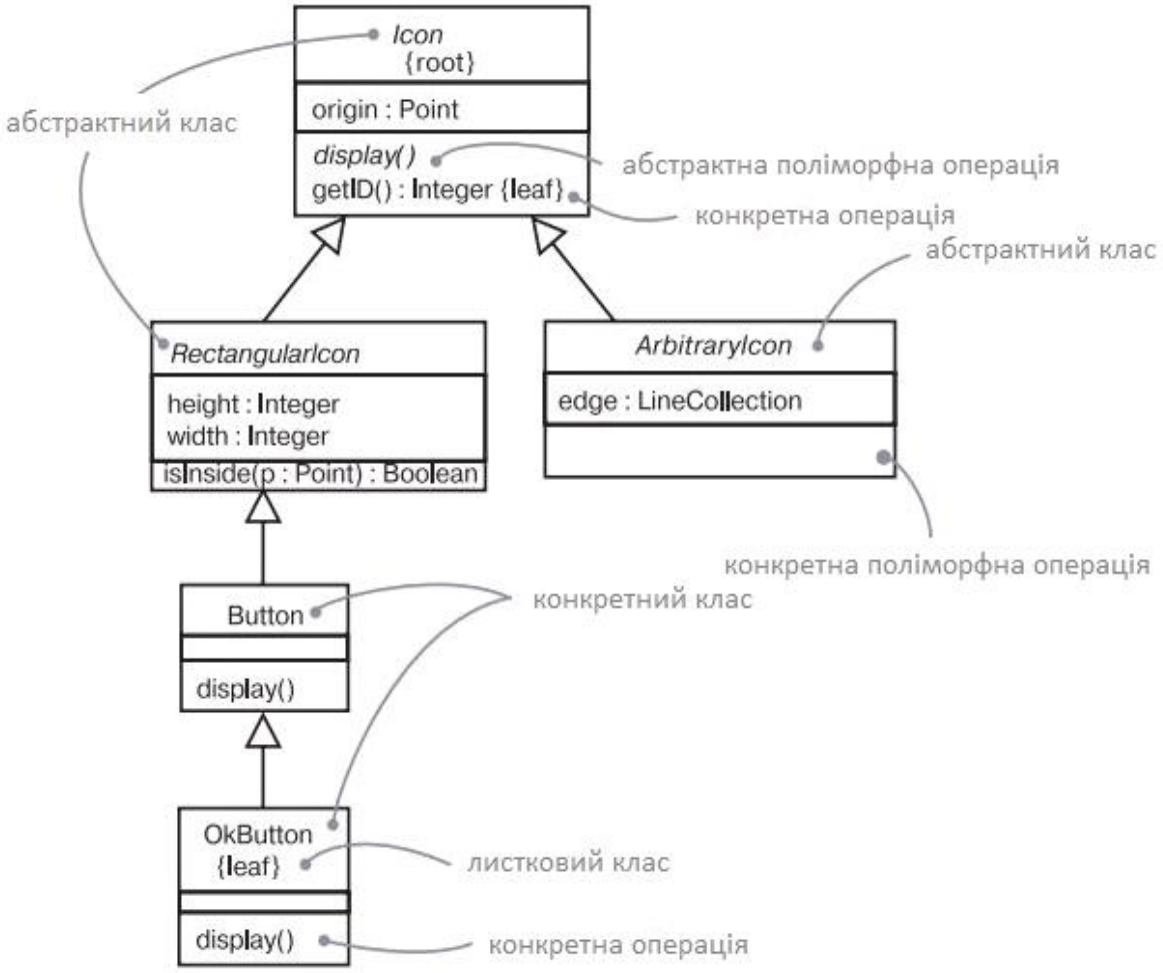

Рисунок 5 − Абстрактні й конкретні класи та операції

Абстрактні, листові й поліморфні операції. Операції мають аналогічні властивості. Як правило, операція є поліморфною, тобто її можна специфікувати операцію з тією же сигнатурою в різних місцях ієрархії класів. Операція в дочірньому класі скасовує поведінку такої ж операції батьківського класу (пріоритетна перед нею). Коли повідомлення посилається під час виконання, конкретна операція, що викликається, вибирається з ієрархії поліморфно: визначається під час виконання відповідно до класу об'єкта. Операції display (відобразити) і IsInside (всередині) поліморфні (*див*. рис. 5). Операція Icon::display() є абстрактна операція, що нереалізована в класі Icon і «вимагає» від нащадків надання її власних реалізацій. Імена абстрактних операцій в UML за аналогією з абстрактними класами виділяються курсивом. Операція Icon::getid() є листовою операцією (забороненою для успадкування), на що вказує ключове слово leaf. Вона не є поліморфною і не може бути перевизначена в класах-нащадках. В Java такі операції називаються final (кінцевими).

Моделювання структурних зв'язків: ІС університету. Щоб змоделювати структурні зв'язки, потрібно встановити асоціацію між кожною парою класів, вказавши навігацію між об'єктами цих класів і проаналізувати як об'єкти одного з них, що потребують взаємодії з об'єктами іншого (але не у вигляді параметрів операції). Для кожної із встановлених асоціацій специфікувати множинність та імена ролей. Якщо клас на одному кінці асоціації структурно або організаційно є ціле, а клас на іншому кінці є його частиною, позначити такий зв'язок як включення або композиція.

На основі аналізу варіантів використання (ВВ) системи можна встановити усі структурні й поведінкові сценарії. Для будь-яких класів, що взаємодіють між собою, встановлюються зв'язки асоціації.

На рис. 6 показано набір класів інформаційної системи (ІС) навчального закладу. Це класи Student (Студент), Course (Курс або Дисципліна) і Professor (Викладач). Існує асоціація між Student і Course, що

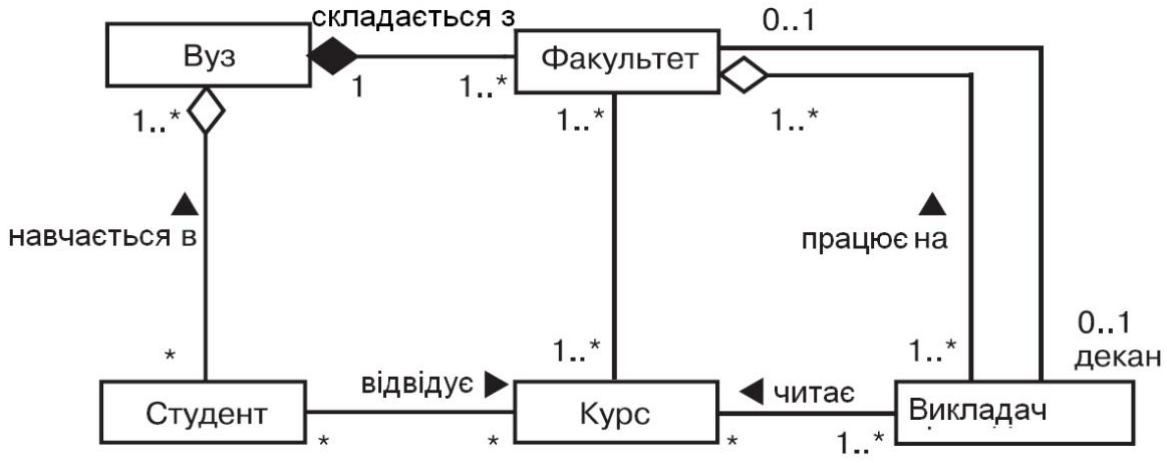

Рисунок 6 − Діаграма класів, що відображає структурні зв'язки

свідчить про те, що студенти відвідують курси. Кожен студент може відвідувати багато дисциплін і кожна дисципліна може бути прочитана для багатьох студентів. Показана асоціація між Course і Professor, яка визначає, що викладачі читають дисципліни. Для кожної дисципліни призначений як мінімум один викладач, причому кожен може читати одну або кілька дисциплін або не читати жодної. Кожна дисципліна закріплена за певним факультетом (кафедрою).

Діаграми об’єктів (object diagrams) дозволяють моделювати екземпляри сутностей, які містяться в діаграмах класів. На діаграмі об’єктів показано множини об’єктів і зв’язків між ними в певний момент часу. Діаграми об’єктів застосовують при моделюванні статичних виглядів системи з погляду проектування й процесів. При цьому моделюється «знімок» системи в конкретний момент часу і відображається множина об’єктів, їх станів і зв’язків між ними [4, 7]. Діаграми об’єктів важливі для візуалізації, специфікації, конструювання й документування структурних моделей і для конструювання статичних аспектів системи за допомогою прямого і зворотного проектування.

Для людини, не знайомої з правилами гри, футбол може здатися надзвичайно простим видом спорту: юрба народу хаотично носиться полем, ганяючи м’яч. Недосвідчений глядач навряд чи побачить у цьому русі який-небудь порядок або оцінить красу гри. Якщо перервати матч і показати глядачеві, які ролі виконують окремі гравці, перед ним постане зовсім інша картина. У загальній масі він розрізнить нападників, захисників і півзахисників, а поспостерігавши за ними, зрозуміє, як вони взаємодіють за певною стратегією, спрямованою на те, щоб забити гол у ворота супротивника: ведуть м’яч по полю, відбирають його один в одного й атакують. У досвідченій команді ніколи не знайдеться гравців, що безладно і безцільно переміщуються по полю. Навпаки, у будь-який момент часу розташування гравців і їх взаємодії точно розраховані.

Спостерігаючи за потоком керування в працюючій системі ПЗ, швидко втрачається загальна уява про організацію її складових, особливо якщо є кілька потоків. Вивчення стану одного об’єкта в конкретний момент часу так само не допоможе зрозуміти таку складну структуру. Потрібно розглянути не тільки сам об’єкт, але і його найближчих сусідів та зв’язки між ними. Об’єкти не існують автономно, а певним чином зв’язані з множиною інших. Більше того, неполадки в таких системах найчастіше пояснюються не логічними помилками, а саме порушеннями взаємозв’язків об’єктів або непередбаченими змінами їх стану.

Статичні аспекти UML – будівельних блоків системи – візуалізують за допомогою діаграм класів. Діаграми взаємодії дозволяють побачити динамічні аспекти системи, включаючи екземпляри цих будівельних блоків і повідомлення, якими вони обмінюються. Діаграма об’єктів містить множини екземплярів сутностей, наведених на діаграмі класів.

Діаграми об’єктів зображують статичну складову взаємодії й складаються із взаємодіючих об’єктів, однак повідомлення на них не показані. Їх подають у вигляді графа, що складається з вершин і ребер. Діаграма об’єктів відображає стан системи у фіксований момент часу (рис. 7).

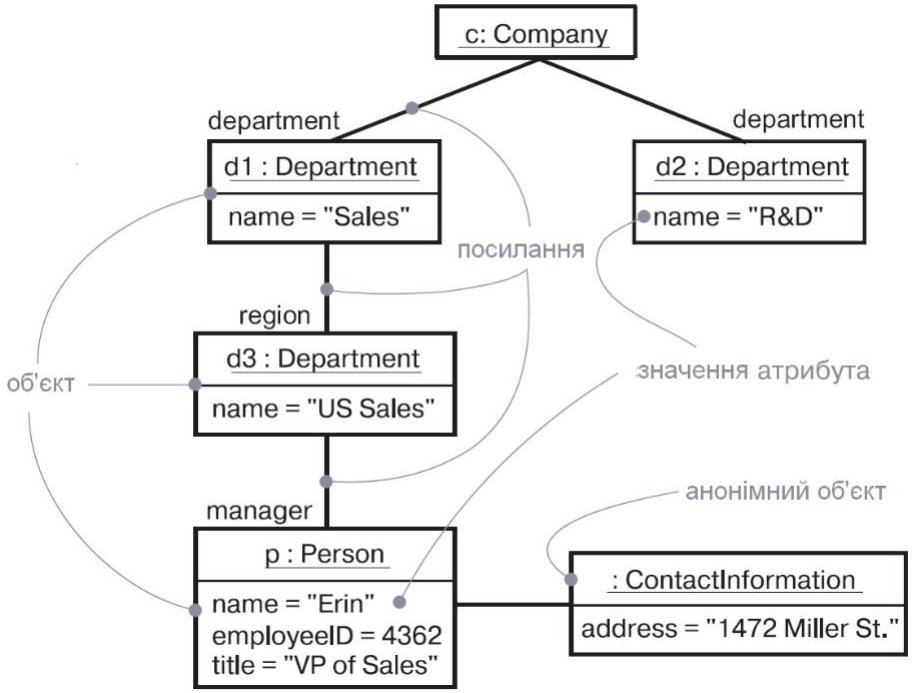

Рисунок 7 − Діаграма об’єктів

На діаграмі об’єктів показані об’єкти і зв’язок між ними в певний момент часу. Загальні властивості. У діаграми об’єктів, як і в будь-якої діаграми, є ім’я і графічний зміст, що є проекцією моделі. Від інших діаграм відрізняється своїм конкретним наповненням. Зміст діаграми об’єктів, як правило, містить об’єкти й посилання та може містити в собі примітки та обмеження. Іноді в них вміщують і класи, особливо якщо треба візуалізувати класи, що визначають екземпляри.

Статичний кадр у динамічному сценарії. За допомогою діаграм об’єктів, як і за допомогою діаграм класів, моделюють статичний вигляд системи з погляду проектування або процесів, враховуючи реальні екземпляри чи прототипи. Цей вигляд відображає в основному функціональні вимоги до системи, тобто послуги, які вона має надавати кінцевим користувачам. Діаграми об’єктів дозволяють моделювати статичні структури даних.

2 ДІАГРАМИ МОДЕЛЮВАННЯ ПОВЕДІНКИ

Об’єктно-орієнтована система не існує в ізоляції, а взаємодіє з дійовими особами (актантами: людьми, системами), які використовують її для досягнення деякої мети, очікуючи від неї певної поведінки. Варіанти використання (ВВ) специфікують цю очікувану поведінку системи, описують послідовності дій, які вона здійснює для досягнення дійовою особою певного результату. ВВ застосовуються для вираження необхідної поведінки розроблювальної системи, без описування реалізації цієї поведінки. Вони дозволяють розробникам, кінцевим користувачам і експертам у предметній області досягти консенсусу, допомагають впевнитися в правильності архітектурних рішень і перевірити систему в ході її розроблення. У процесі створення системи ВВ реалізуються за допомогою кооперацій, елементи яких працюють спільно для досягнення цілей кожного з них. Добре структуровані ВВ описують тільки істотні аспекти поведінки і не є ні надто узагальненими, ні надто деталізованими.

Правильно спроектований будинок − це більше, ніж стіни, що підпирають дах, який захищає мешканців від дощу. Працюючи разом з архітектором над проектом, продумують план використання приміщень: для вітальні, щоб приймати гостей і зручно спілкуватися; кухні – для приготування їжі, розміщення меблів, побутової техніки. Навіть маршрут транспортування продуктів з машини на кухню вплине на розташування кімнат. Є потреба у визначенні оптимальної кількості ванних кімнат і їх розміщення, що дозволило б уникнути «черг», коли всі одночасно збираються на роботу, до школи, університету.

Це приклад аналізу ВВ. Потрібно розглядати різні ВВ будинку, які в підсумку зумовлюють його архітектуру. Для багатьох родин ВВ є подібними: в усіх будинках їдять, сплять, виховують дітей, обговорюють витвори провінційних брендів за бокалом шамбертен. Але в кожному разі висуваються індивідуальні вимоги до житла. Потреби великої родини відрізняються від вимог самотнього холостяка, що вплине на майбутній вигляд будинку.

Варіант використання (ВВ) (use case) – це опис багатьох послідовних дій (включаючи варіації), які виконуються системою з метою отримання результату, важливого для деякої дійової особи [7].

Дійова особа являє собою логічно зв’язану множину ролей, які виконують користувачі системи під час взаємодії з нею. Дійовими особами можуть бути як люди, так і автоматизовані системи. Процес опрацювання позик містить взаємодію клієнта зі співробітником кредитного відділу.

UML-нотація ВВ зображується у вигляді еліпса, а дійових осіб – «чоловічками», що дозволяє візуалізувати певний ВВ у контексті інших і окремо від його реалізації (рис. 8).

Рисунок 8 − Дійові особи та варіанти використання

В UML діаграми ВВ застосовуються, щоб візуалізувати поведінку системи, аби користувач міг зрозуміти, як застосовувати цей елемент, а розробник – як реалізувати його. Діаграма ВВ допомагає змоделювати поведінку того самого «загадкового» пристрою із кнопками і дисплеєм, у якому більшість людей без проблем впізнає мобільний телефон (рис. 9).

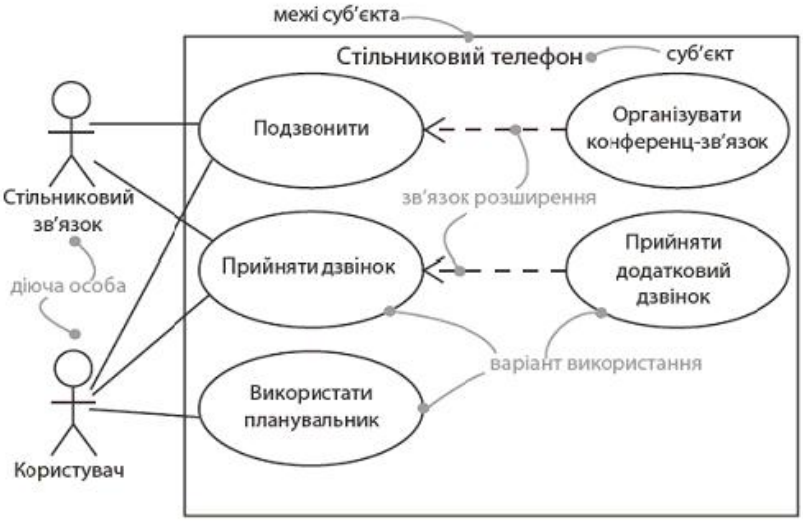

Рисунок 9 − Діаграма варіантів використання

Діаграми взаємодій використовуються для моделювання **потоків керування** (*в межах операції, класу, компоненти, ВВ чи системи в цілому*), визначивши, як повідомлення передаються в часі, якими є структурні зв'язки між об'єктами у взаємодії, як повідомлення передаються в контексті структури [2,3,4].

Повідомлення (message) є специфікацією взаємодії об'єктів, що передає інформацію й очікує подальших дій [2]. Повідомлення можуть супроводжуватися викликом операції або передавання сигналу та можуть супроводжуватися створенням і знищенням інших об'єктів. На концептуальному рівні повідомлення специфікуються від взаємодії між класами об'єктів. Конкретна взаємодія між двома об'єктами визначатиметься конкретним екземпляром повідомлення. Графічна нотація повідомлень в UML є лінією зі стрілкою і специфікує його найважливіші

властивості: номер послідовності, ім’я його операції, параметри, якщо є (рис. 10).

Контекст взаємодії. Взаємодія має місце, коли об’єкти з’єднані один з одним. Ввзаємодії проявляються в кооперaціях об’єктів, що існують у контексті системи (підсистем, класів із реалізації ВВ та окремих операцій). У контексті взаємодії можна виявити екземпляри класів, компонентів, вузлів і ВВ [2, 3, 7].

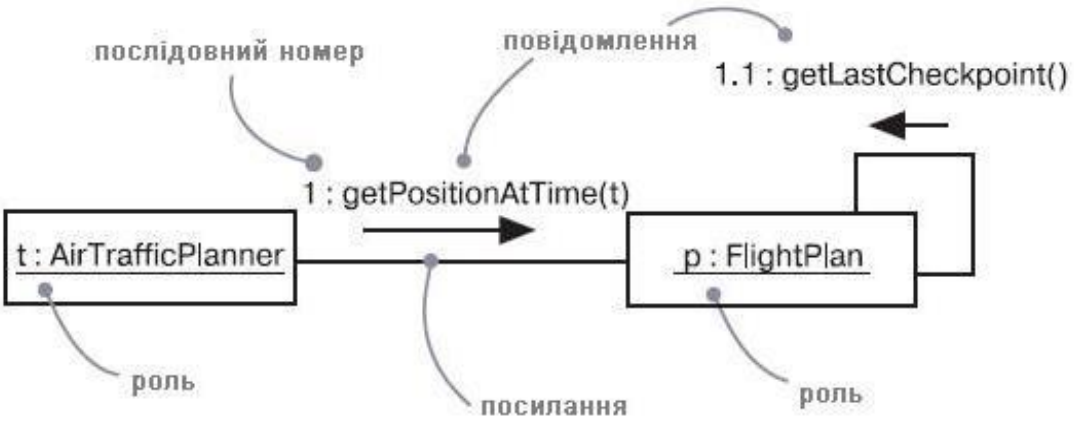

Рисунок 10 − Повідомлення, посилання і послідовність

Прототипні об’єкти (ролі) та **прототипні зв’язки або посилання** (коннектори). При розробленні моделей більше використовуються прототипні об’єкти та їх прототипні посилання ніж конкретні індивідуальні об’єкти з їхніми посиланнями. Множинність ролей (прототипних об’єктів) і коннекторів (прототипних посилань) визначена щодо контексту, що їх включає (контекстом тут є кооперація або внутрішня структура класифікатора) [2, 3]. Якщо множинність ролі дорівнює 1, то на один об’єкт, що представляє контекст, припадає один об’єкт, що представляє роль. Неідентифікована роль візуалізується як анонімний чи прототипний об’єкт. Ролі (прототипні об’єкти) зв’язані коннекторами (прототипними посиланнями /зв’язками/) із вказаними повідомленнями (концептуальними), а об’єкти – посиланнями (зв’язками як екземплярами асоціацій), позначеними реальними екземплярами повідомлень.

У верхній частині рис. 11 показана діаграма класів, яка оголошує класи Person (Особа) і Company (Компанія) та існуючу між ними асоціацію «багато до багатьох» employee-employer (наймач-працівник). Середня представляє зміст кооперації WorkAssignment (Призначення на посаду), що містить у собі дві ролі з коннектором між ними. Внизу поданий екземпляр кооперації, у якому є об’єкти і посилання, що представляють ролі й коннектори. Конкретне повідомлення є прототипним оголошенням повідомлення в даній кооперації.

Типи та екземпляри повідомлень. Діаграма об’єктів, що включає набір об’єктів і посилань, які з’єднують ці об’єкти, є статичною моделлю,

що описує стан співтовариства об’єктів у заданий момент часу. При моделюванні змінного стану набору об’єктів за деякий період часу потрібно наочно продемонструвати їх переміщення в наступні моменти часу, включаючи передавання повідомлень між об’єктами, події та виклики операцій. У кожному такому «кадрі» можна явно візуалізувати поточний стан і роль індивідуальних екземплярів [3].

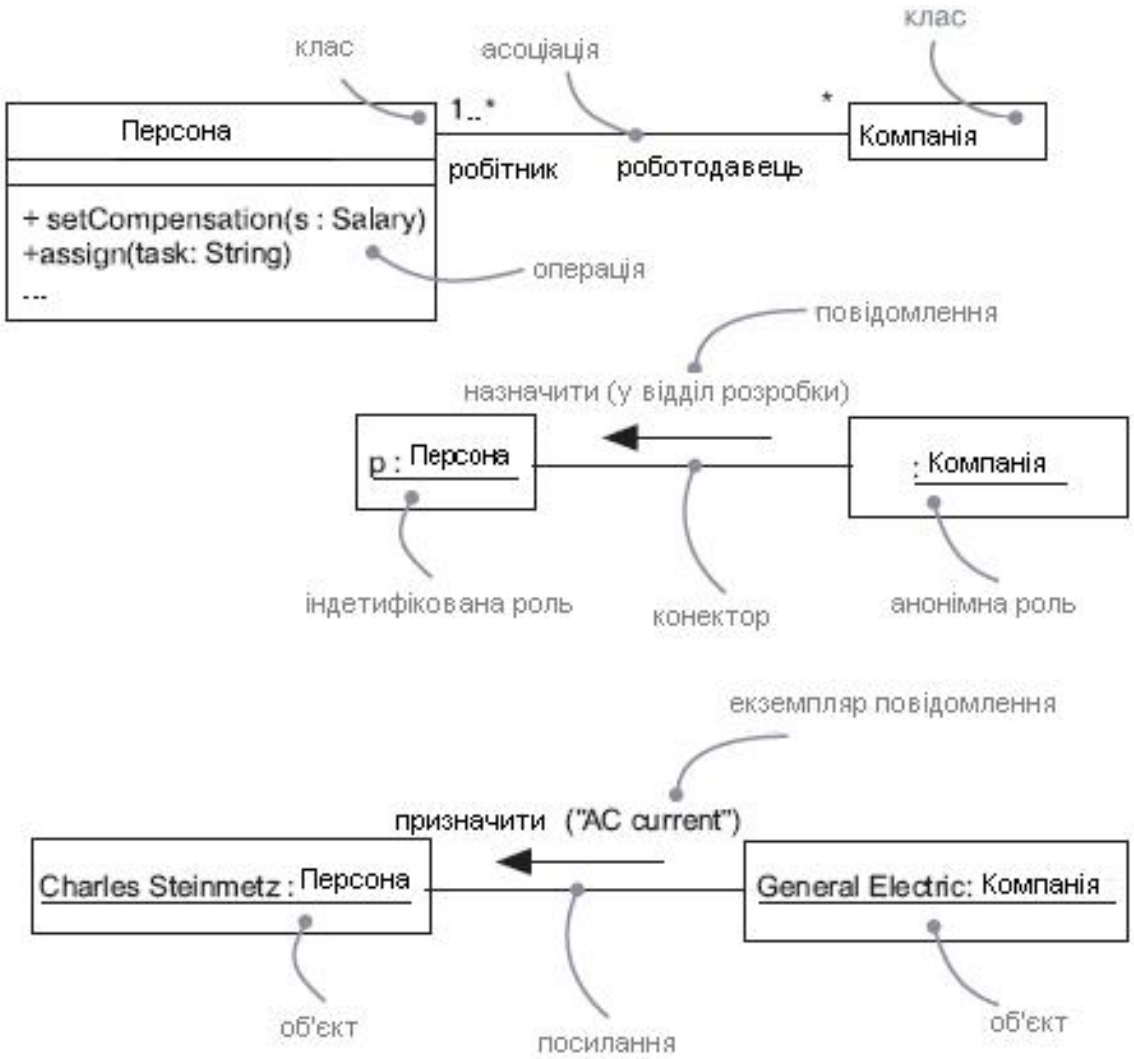

Рисунок 11 − Асоціації, посилання (екземпляри асоціацій), коннектори (прототипні посилання)

Типи повідомлень, моделювання яких забезпечує UML:

- call (викликати) – об’єкт викликає операцію іншого об’єкта або свою власну, посилаючи повідомлення самому собі (локальний виклик операції);
- return (повернути) – повертає значення викликаючому об’єкту;
- send (послати) – посилає сигнал об’єкту;
- create (створити) – створює об’єкт;
- destroy (знищити) – знищує об’єкт. Об’єкт може знищити сам себе.

В UML ці типи повідомлень візуально відрізняються (рис. 12). Об’єкт не може викликати будь-яку довільну операцію. Якщо об’єкт (рис. 12) викликає, операцію setItinerary (вказати напрямок) на екземплярі класу TicketAgent (Агент продажі білетів), то вона має бути оголошена в класі TicketAgent або в одному з його батьків і бути видима для викликаючого об’єкта С.

Коли об’єкт викликає операцію чи посилає сиґнал іншому об’єкту, то повідомлення може бути оснащене конкретним параметром. Аналогічно, коли один об’єкт повертає керування іншому, при цьому також можна змоделювати значення, що повертається.

Посилання сигналів. Повідомлення можуть мати відношення й до посилання сиґналів. **Сигнал** (signal) – це значення об’єкта, передане цільовому об’єкту асинхронно. Після відправлення сигналу об’єкт, який його відправив, продовжує свою діяльність. Отримуючи повідомлення сигналу, цільовий об’єкт незалежно приймає рішення стосовно того, що з ним потрібно робити [2].

Сигнали ініціюють перехід автомата цільового об’єкта в інший стан. Ініціалізація такого переходу змушує цільовий об’єкт виконати якісь дії й змінити свій стан. У системах з асинхронним передаванням подій взаємодіючі об’єкти виконуються паралельно й незалежно. Вони розділяють інформацію тільки за допомогою передавання повідомлень, тому не виникає небезпеки конфлікту за використовувані ресурси.

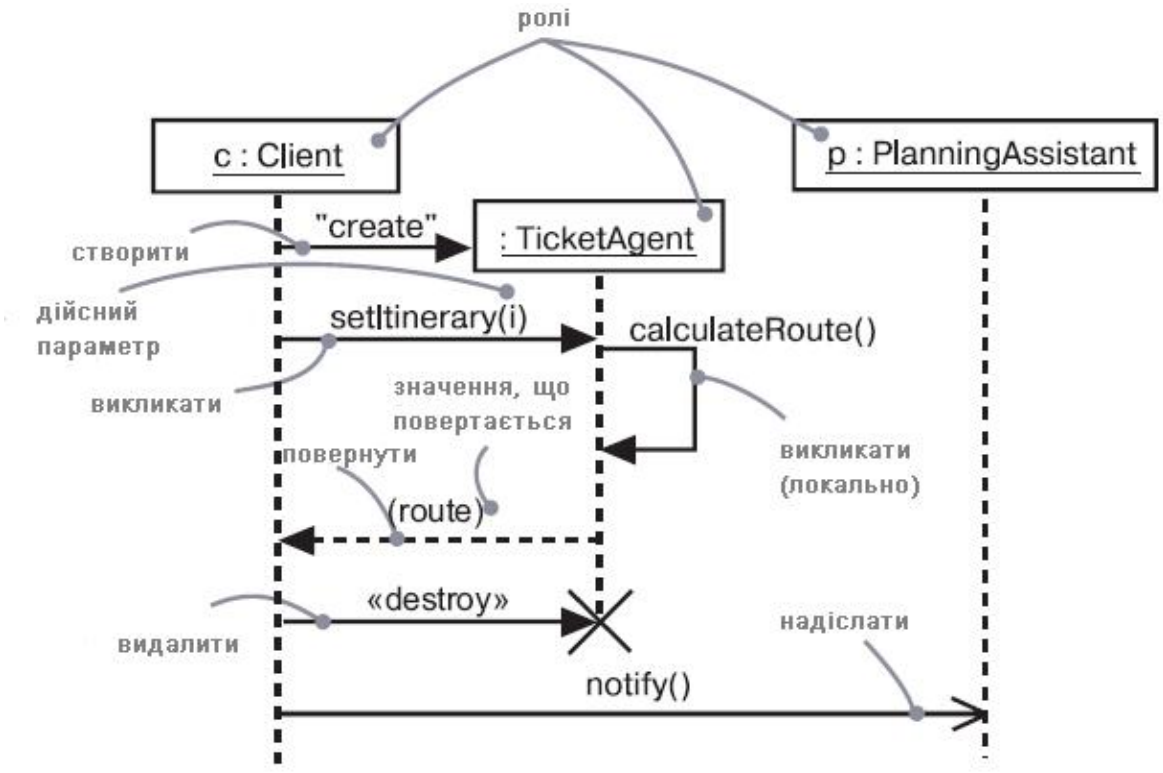

Рисунок 12 − Типи повідомлень на діаґрамі кооперації

Потоки та послідовності (sequencing) **повідомлень**. Коли один об’єкт передає повідомлення іншому (делегуючи йому деяку дію), то об’єкт, що приймає, може, в свою чергу, послати повідомлення ще одному об’єкту, а той – наступному й т.д у вигляді потоку повідомлень, що формує послідовність [8, 9] (див. рис. 12). Усяка послідовність повідомлень повинна мати початок. Її запуск виконується певним процесом або потоком керування. Послідовність триває доти, поки існує процес або потік, що володіє нею. Системи «нон-стоп», що використовуються для моделювання СРЧ, працюють доти, поки працює вузол, на якому вони запущені.

Потоки керування або **процеси** визначають упорядковані в часі (динамічно) послідовності повідомлень у системі й характеризують її поведінку в цілому. Потік керування (процес) запускає (ініціалізує) послідовність повідомлень (рис. 13). Щоб краще візуалізувати послідовність повідомлень, можна явно змоделювати їхній порядок відносно початку послідовності, надавши повідомленню префікс, що вказує його номер з двокрапкою як роздільника.

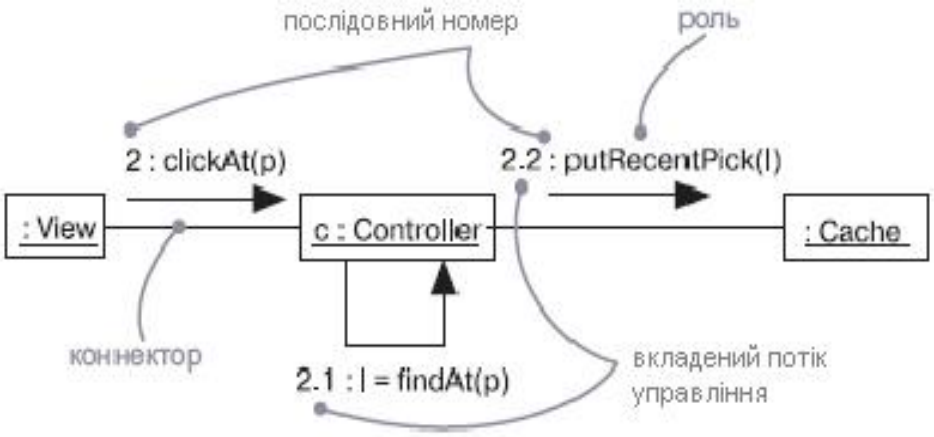

Рисунок 13 − Процедурний (вкладений) потік керування

Моделювання потоку керування. Найчастіше взаємодії використовуються для моделювання потоку керування, що характеризує поведінку системи в цілому (зокрема ВВ, зразки, механізми і каркаси) або поведінку класу чи окремої операції. Моделюючи взаємодію створюється сценарій дій, що реалізовується кооперацією об’єктів певних класів (рис. 14).

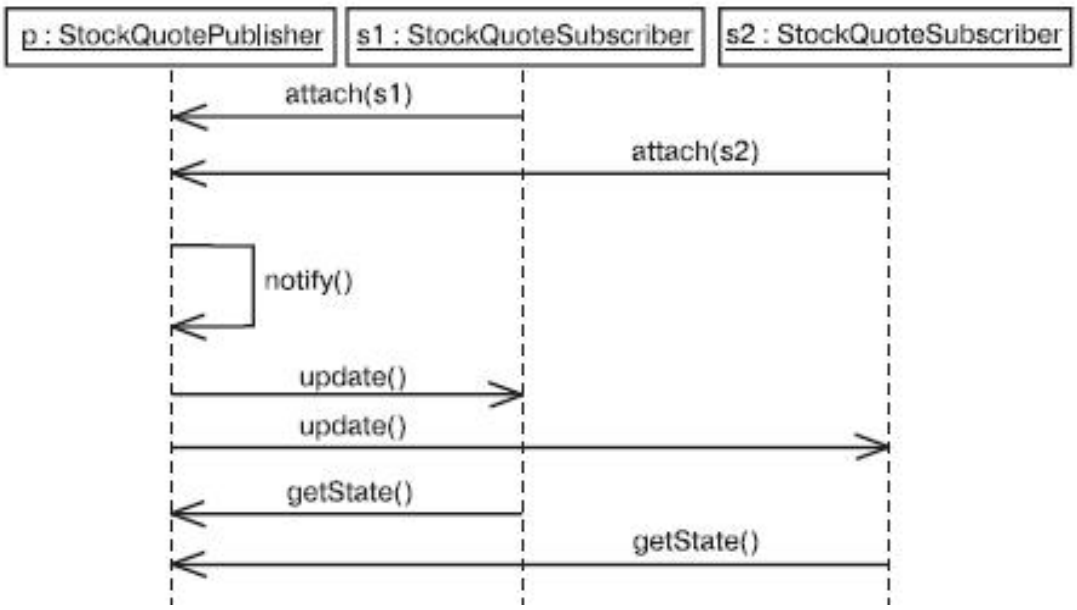

Рисунок 14 − Потік керування в часі (діаграма послідовності)

Для моделювання потоку керування необхідно: встановити контекст взаємодії (чи це система, клас, операція); визначити фазу взаємодії, вияснивши, які об’єкти відіграють ті чи інші ролі; встановити початкові характеристики об’єктів (значення атрибутів, стан і роль та дати ролям імена). Якщо модель описує структурну організацію об’єктів, ідентифікувати з’єднуючі їх посилання, важливі для здійснення комунікацій; специфікувати їх природу, використовуючи необхідні засоби UML. Відповідно до послідовності потрібно вказати повідомлення, передані від об’єкта до об’єкта; виділити різні їх види; описати параметри і значення, що повертаються. Для деталізації взаємодії необхідно наділити кожен об’єкт, поданий у різні моменти часу, інформацією про його стан і ролі.

3 ДІАГРАМИ МОДЕЛЮВАННЯ ПОДІЙ

Реальний світ насичений подіями, які дуже часто настають раптово та одночасно. В UML будь-яке явище, яке може мати місце в дійсності, моделюється як подія.

Усі системи, пов’язані з реальністю, тією чи іншою мірою є динамічними, причому динаміку зумовлюють саме події, що відбуваються всередині системи або за її межами. Роботу банкомата ініціює користувач, який натискає кнопку для здійснення банківської транзакції. Автономний робот починає діяти, коли зустрічається з деяким предметом. Мережевий маршрутизатор реагує на виявлення переповнення буферів повідомлень. На хімічному заводі сигналом до дії стає закінчення періоду, необхідного для завершення реакції.

Подія (event) – це опис істотного факту, що відбувся в певному часі й просторі. Отримання сигналу, закінчення проміжку часу, зміна стану – це приклади асинхронних подій, які можуть відбутися в будь-який момент [2]. У контексті кінцевих автоматів події використовуються для моделювання певного впливу, який може викликати перехід з одного стану в інший. До подій належать сигнали, виклики, закінчення певного проміжку часу або зміна стану. Події можуть бути синхронними й асинхронними; їхнє моделювання – одна зі складових моделювання процесів і потоків.

У сенсі автоматів подія означає вплив, який може викликати перехід з одного стану в інший. Настання події – зміна станів. Графічна UML нотація подій дозволяє реалізовувати оголошення подій у вигляді сиґналів (сигнал OffHook (Трубка повішена) і показати, як настання події призводить до переходу між станами: сигнал OffHook викликає перехід телефону зі стану Active (Активний) у стан Idle (Очікування) й виконання дії drop connection (розірвати зв’язок) (рис. 15).

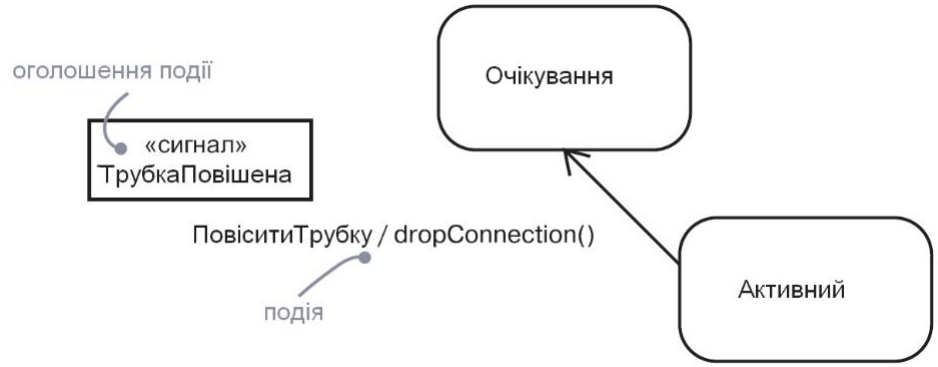

Рисунок 15 − Події

Сигнал (signal) є різновидом події, при використанні якого повідомлення передається асинхронно від одного екземпляра до іншого.

Події можуть бути внутрішніми або зовнішніми. Зовнішні події передаються між системою і актантами (натискання кнопки або переривання від сенсора запобігання зіткнень), а внутрішні – між об’єктами, що існують у самій системі (винятки, що генерується при переповненні).

Види подій. В UML можна моделювати всі чотири види подій: сигнали, виклики, закінчення проміжку часу і зміна стану.

Повідомлення є іменований об'єкт, який асинхронно посилається одним об'єктом і приймається іншим. Сигнал є класифікатором для повідомлень і сам є типом повідомлення. У сиґналів є багато загального з класами. Можна говорити про екземпляри сигналів. Сигнали можуть брати участь у зв'язках узагальнення, що дозволяє моделювати ієрархії подій, де одні є загальними (сигнал NetworkFailure (Збій мережі), а інші – їх спеціалізованими версіями (WarehouSeserverFailure (Відмова складського сервера) є спеціалізаціяація події Networkfailure). Як і класи, сигнали можуть мати атрибути й операції.

Моделювання сигналів класами. Сигнал може відправлятися в результаті виконання якоїсь дії в процесі переходу (зміни стану) в автоматі. Сигнал можна моделювати як повідомлення, передане між двома ролями при деякій їхній взаємодії. При виконанні методу також можуть передаватися сиґнали. При моделюванні поведінки класів та інтерфейсів важливою частиною специфікації поведінки є вказівки сигналів, які операції можуть їх посилати [3]. Сиґнали в UML моделюються класами зі стереотипами (рис. 16). Для моделювання вказівки, що деяка операція (moveTo()) посилає сиґнал, використовується залежність зі стереотипом «send» (вказівка).

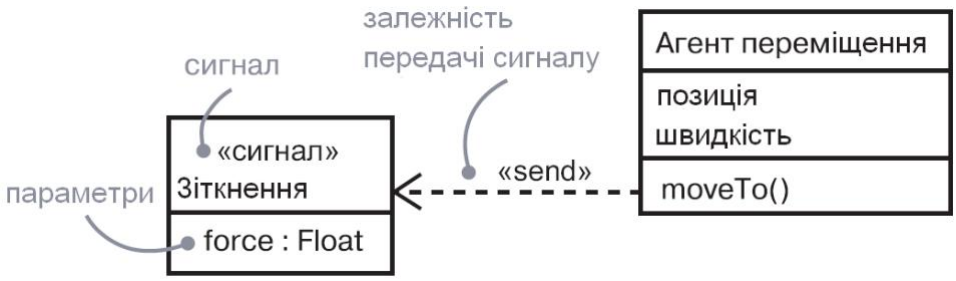

Рисунок 16 − Моделювання сигналів класами

Поліморфні події. Моделюючи подібним чином ієрархії сигналів, можна специфікувати поліморфні події. Нехай автомат, у якому деякий перехід спрацьовує тільки при отриманні сиґналу Motorstall. Оскільки цей сигнал є в ієрархії листовим, то перехід ініціюється тільки ним, так що поліморфізм відсутній. Для автомата, у якому перехід спрацьовує при отриманні сигналу Hardwarefault, перехід буде поліморфний: його викликає як сам сигнал Hardwarefault, так і будь-яка його спеціалізація (різновид), включаючи Batteryfault, Movement Fault і Motorstall.

Для моделювання множини сигналів необхідно: розглянути всі різновиди сигналів, на які може відповідати така множина активних об'єктів; виявити схожі види сиґналів і вмістити їх в ієрархію типу «узагальнення/спеціалізація». На рис. 17 наведена множина сигналів, які б могли оброблятися автономним роботом. Кореневий сигнал Robotsignal (Сигнал робота) є абстрактним, тобто безпосереднє створення його екземплярів неможливе. У цього сигнала є дві конкретні спеціалізації (Collision і Hardwarefault), одна з яких (Hardwarefault) спеціалізується й далі. В сигнала Collision є один параметр.

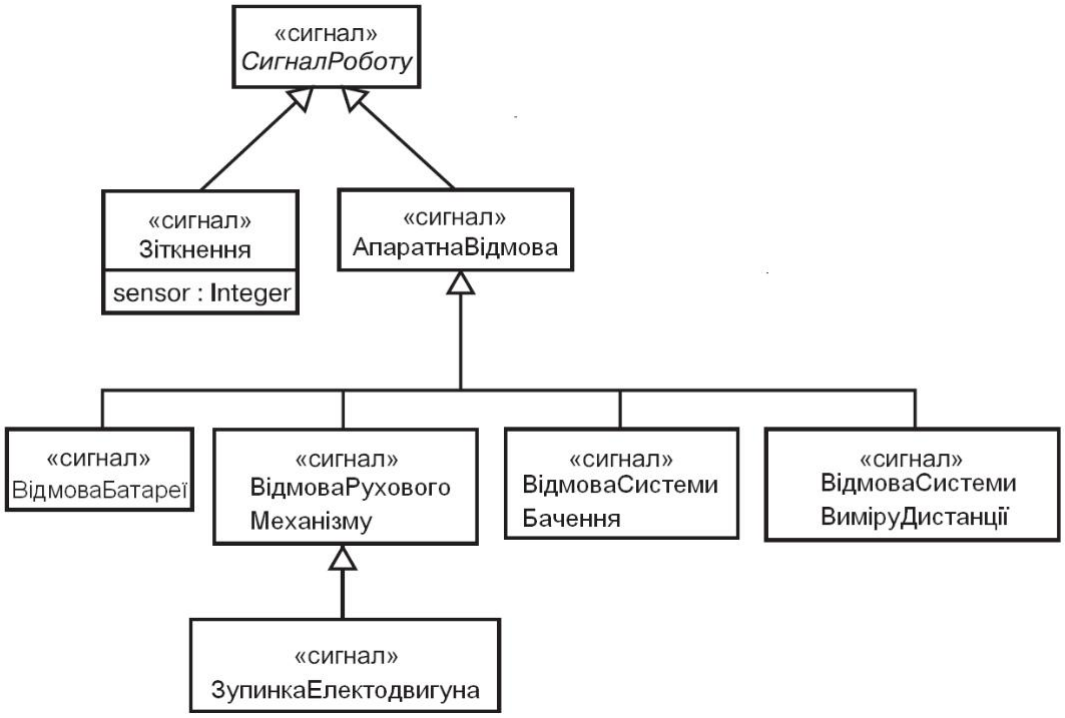

Рисунок 17 − Моделювання сімї сиґналів

Динамічні аспекти системи в UML моделюються кінцевими автоматами (state machines). Автомат – це поведінка, яка специфікує послідовність станів об'єкта, через які він проходить протягом свого життєвого циклу (ЖЦ) у відповідь на події, а також реакції на ці події [2].

У той час, як взаємодія моделює співтовариство об'єктів, що спільно працюють для виконання деяких дій, автомат моделює ЖЦ окремого об’єкта – екземпляр класу, ВВ або навіть усієї системи. За час ЖЦ в об’єкті може відбуватися безліч різноманітних подій, таких, як передавання і приймання сигналів, виклики операцій, створення та знищення об’єкта, закінчення періоду часу, відведеного на якусь дію, або зміна якихось умов. У відповідь на ці події об’єкт виконує певну дію, що являє собою обчислення, а потім змінює свій стан на інше. Тому поведінка такого об’єкта залежить від минулого, принаймні в тому ступені, у якому воно впливає на його поточний стан. Об’єкт може прийняти подію, відповісти

на нього дією, потім змінити свій стан. Коли ж він приймає іншу подію, його реакція може бути іншою – залежно від поточного стану, який є результатом попередньої події.

Події, стани, переходи й ефекти в ЖЦ об’єктів. Автомати використовуються для моделювання поведінки будь-якого елемента, найчастіше класу, ВВ або всієї системи. Динаміку виконання деякого процесу можна візуалізувати двома способами: визначаючи потік керування від однієї діяльності до іншої (діаграми діяльності) або виділяючи потенційні стани об’єкта і переходи між ними (діаграми станів). Автомати можуть бути добре візуалізовані на діаграмах станів. Можна зосередитися на поведінці об’єкта, що залежить від послідовності подій, що особливо зручно для моделювання реактивних систем. Добре структуровані автомати подібні добре структурованим алґоритмам: вони ефективні, прості, адаптовані й зрозумілі.

Графічна нотація станів, переходів, подій і ефектів в UML дозволяє показати поведінку об’єкта таким чином, щоб виділити важливі елементи його ЖЦ (рис. 18).

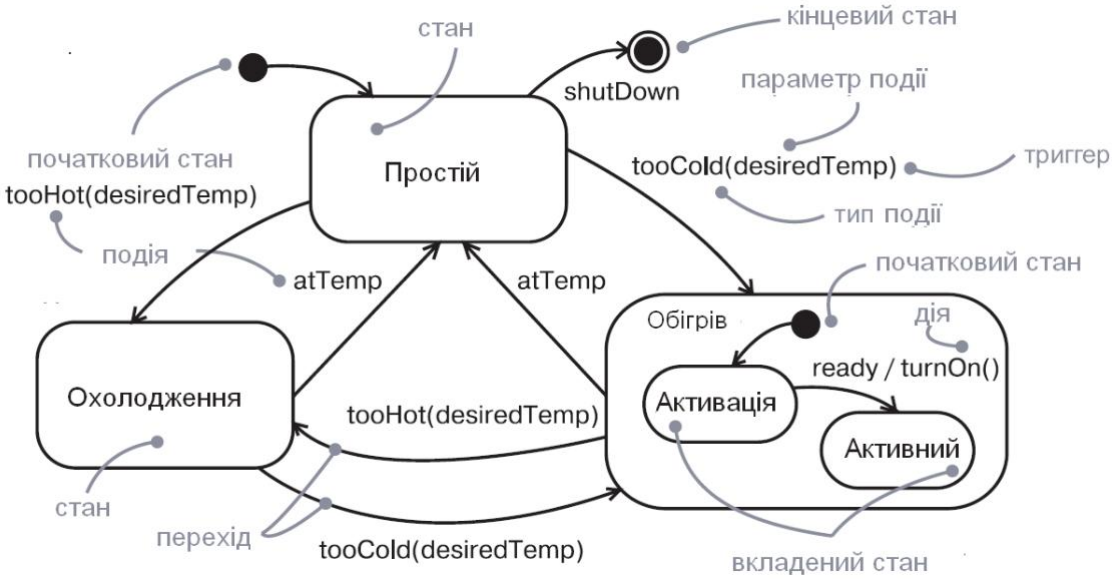

Рисунок 18 − Автомати

Стан (state) – це ситуація в ЖЦ об’єкта, у якій він задовольняє задані умови, здійснює якусь діяльність або очікує певної події. Об’єкт перебуває в тому або іншому стані обмежений період часу [6]. Обігрівач у будинку може бути в кожному із чотирьох станів: простий (очікування команди на ввімкнення обігріву), активація (газ поданий, але очікується досягнення певної температури), функціонування (газ і казан ввімкнені), вимикання (подача газу припинилася, але казан увімкнений, залишкове тепло скидається із системи).

4 ДІАГРАМИ МОДЕЛЮВАННЯ АРХІТЕКТУРИ

Системна архітектура є найважливішим артефактом, який може бути використаний для керування всіма цими різноманітними точками зору і тим самим управляє ітеративним і покроковим процесами розроблення системи протягом усього її ЖЦ. Архітектура – це набір суттєвих рішень відносно:

- організації програмної системи;
- вибору структурних елементів, що складають систему, їх інтерфейсів;
- поведінки цих елементів, специфікованої в їхніх кооперaціях;
- об'єднання цих структурних елементів та елементів поведінки у більші підсистеми;
- архітектурного стилю, що визначає організацію системи: статичні й динамічні елементи, їх інтерфейси, кооперацію і композицію.

Аналіз і планування вимог (Requirements) – перша фаза процесу (рис. 19), впродовж якої початкова ідея набуває достатнього обґрунтування (принаймні, внутрішнього) для забезпечення переходу до фази розробки. На цій фазі добре розроблена ідея перетворюється в концепцію готового продукту, і створюється бізнес-план розролення цього продукту. На цій фазі мають бути отримані відповіді на питання:

- Що має робити система, насамперед, для її основних користувачів?
- Який вигляд повинна мати архітектура системи?
- Який план і яка вартість розроблення продукту?

У фазі проектування (Design) детально описуються більшість ВВ і розробляється архітектура системи. Між архітектурою системи і системою є прямий і нерозривний зв'язок. Архітектурний прототип набуває форми, у якій він може бути представлений користувачам. На цьому етапі вимоги до системи, особливо критерії оцінювання, постійно переглядаються відповідно до мінливих потреб. Також виділяються необхідні ресурси для зменшення ризиків.

У фазі побудови (Construction) відбувається створення продукту – до архітектури (скелету) додаються закінчені програми (м'язи). На цій фазі базовий рівень архітектури розростається до повної розвиненої системи. Концепції розвиваються до продукту, готового до передавання користувачам. У фазі побудови продукт до рівня, що містить в собі всі ВВ, які керівництво і замовник домовилися включити в поточний випуск. Архітектура системи стабільна, однак, оскільки розробники можуть знайти найкращі способи структурування системи, від них можуть виходити пропозиції про внесення в архітектуру системи незначних змін.

Упровадження (Transition) – четверта фаза процесу, у ході якої ПЗ передається користувачам. Але процес розроблення не часто завершується на цьому, тому що навіть на такій фазі система безупинно удосконалюється: усуваються помилки і додаються нові функціональні можливості, що не ввійшли в більш ранні версії.

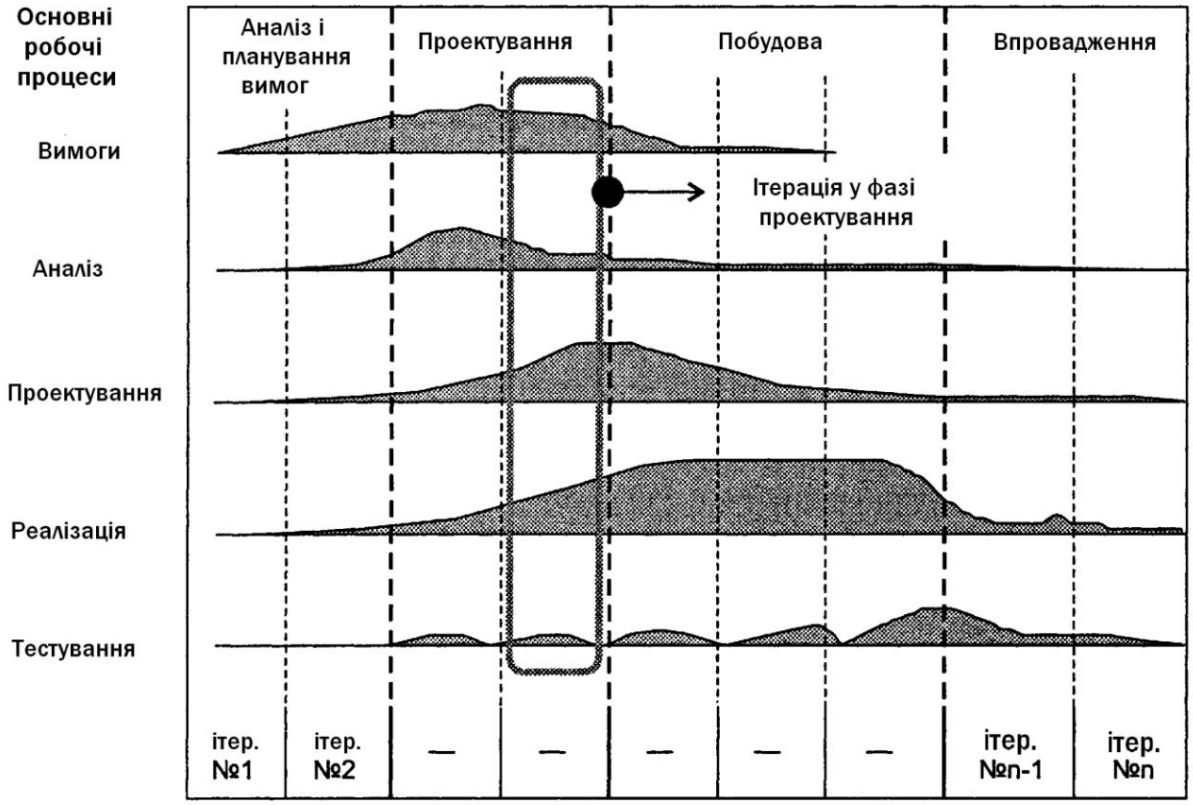

Рисунок 19 − Життєвий цикл розроблення програмного забезпечення

Механізми (як зразки проектування співтовариства класів). Існує низка типових проблем проектування, зокрема для програмістів, що працюють на одній із мов ООП (Java): як змінити клас, налаштований реагувати на деяку множину подій, щоб він також реагував на інші події, не зачіпаючи вихідного коду класу. Типове вирішення проблеми – застосування зразка адаптера (adaptor pattern), структурного зразка проектування, який конвертує один інтерфейс в інший. Цей зразок (як багато подібних) є настільки загальним, що доцільно дати йому певну назву, а потім використовувати його в різних моделях при виникненні подібної проблеми.

Шаблони й параметризовані кооперації. При моделюванні механізми проявляють себе двояко. По-перше, механізм просто іменує набір абстракцій, що працюють разом для реалізації певної типової поведінки системи (рис. 20). Такі механізми моделюються як прості кооперації, оскільки вони є всього лише іменами для співтовариства класів. Розкривши таку кооперацію, можна побачити її структурні аспекти

(зображувані на діаграмі класів), а також поведінкові аспекти (зображувані на діаграмах взаємодії). Кооперації подібного типу охоплюють різні рівні абстракції системи, де певний конкретний клас може брати участь у кількох кооперація.

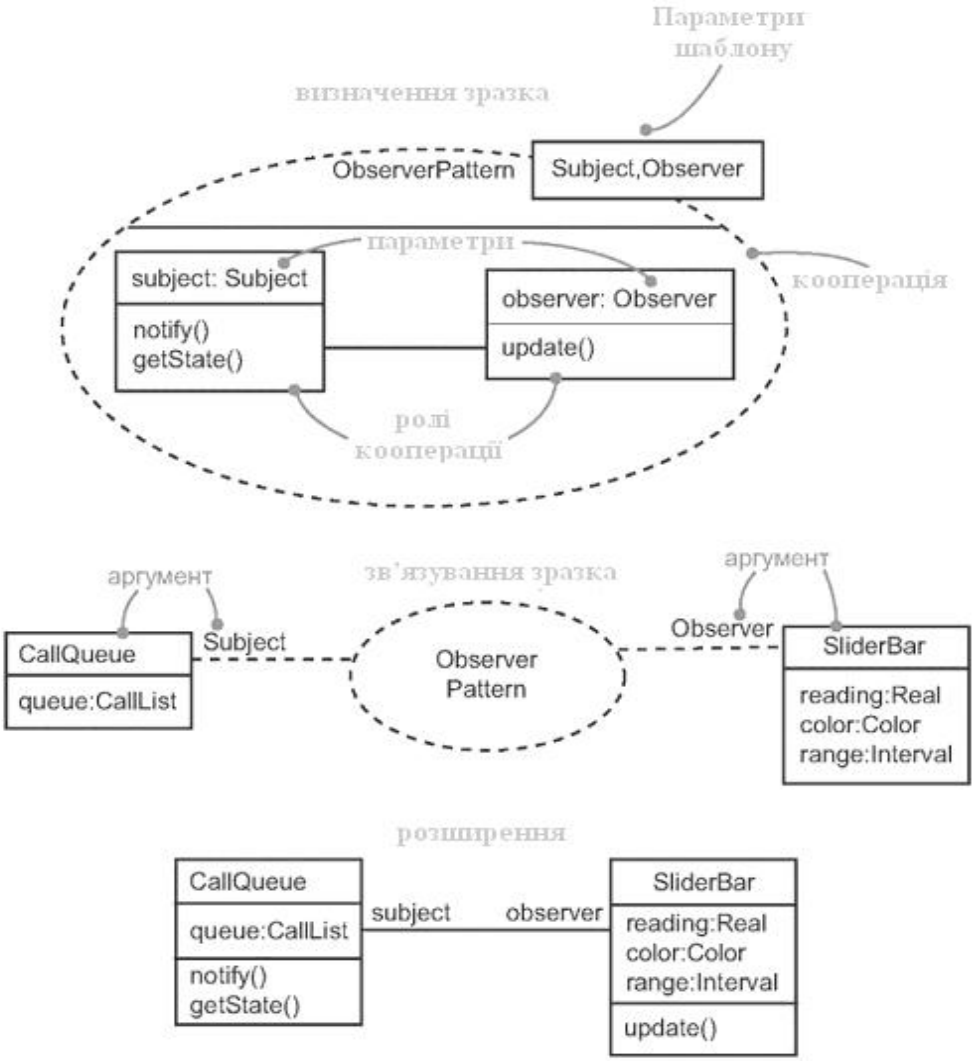

Рисунок 20 − Механізми

Як показано на рис. 20, механізм іменує шаблон для набору абстракцій, що працюють спільно для забезпечення деякої типової поведінки. Такі механізми моделюються у вигляді параметризованих кооперацій, що зображуються в UML подібно до шаблонних класів. Розкривши таку кооперацію, можна побачити її структурні й поведінкові аспекти. Якщо згорнути її, то можна побачити, як зразок застосовується до системи, зв'язуючи шаблонні частини кооперації з існуючими абстракціями.

ПЕРЕЛІК ПОСИЛАНЬ

1. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд. / Пер. с англ. М. : Вильямс, 2020. 720 с.

2. Буч Г., Рамбо Дж., Якобсон И. Язык UML. Руководство пользователя. 2-е изд. / Пер. с англ. М. : ДМК Пресс, 2006. 496 с.

3. Буч Г., Рамбо Дж., Якобсон И. Введение в UML от создателей языка. 2-е изд. / Пер. с англ. М. : ДМК Пресс, 2011. 496 с.

4. Рамбо Дж. Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка. 2-е изд. / Пер. с англ. СПб. : Питер, 2007. 544 с.

5. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения / Пер. с англ. В. Горбунков. СПб. : Питер, 2002. 496 с.

6. Иванов Д., Новиков Ф. Моделирование на UML. Учебно-методическое пособие. СПб. : СПбГУ ИТМО, 2010. 200 с.

7. Хассан Гома. UML. Проектирование систем реального времени, распределенных и параллельных приложений. 2-е изд. / Пер. с англ. М. : ДМК Пресс, 2011. 700 с.

8. Макконелл С. Совершенный код. Мастер-класс / Пер. з англ. СПб. : Питер, 2020. 896 с.

9. Фаулер М., Парсонс Р. Предметно-ориентированные языки программирования / Пер. с англ. М. : Вильямс, 2011. 576 с.

Електронне навчальне видання
комбінованого використання.
Можна використовувати в локальному та мережному режимах

Методичні вказівки
до самостійної роботи студентів
з проектування UML діаграм
в ході виконання курсових робіт з дисципліни
«Об'єктно-орієнтоване програмування»
для студентів спеціальності 121 –
«Інженерія програмного забезпечення»

Укладачі: *Денис Іванович Катєльніков,*
Олександр Олександрович Дудник,
Алла Василівна Денисюк

Рукопис оформив *Д. Катєльніков*

Редактор *О. Ткачук*

Оригінал-макет виготовив *Г. Багдасар'ян*

Підписано до видання 23.11.2021.
Гарнітура Times New Roman.
Зам. № Р2021-043.

Видавець та виготовлювач
інформаційний редакційно-видавничий центр.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
Тел. (0432) 65-18-06.
press.vntu.edu.ua;
Email: irvc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.