МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ

імені адмірала Макарова

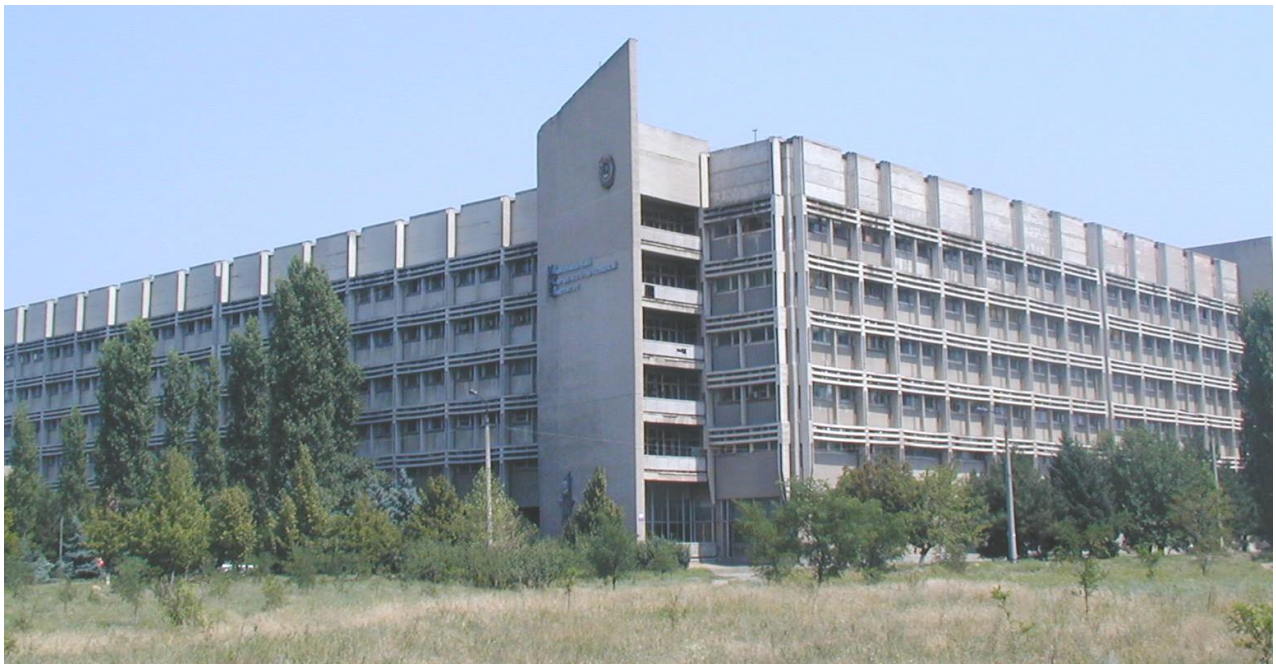
МЕТОДИЧНІ ВКАЗІВКИ

О.М. Дудченко,

С.О. Карпова

“CASE-ЗАСОБИ РОЗРОБКИ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ”

Методичні вказівки до виконання лабораторних робіт



Миколаїв 2018

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ
імені адмірала Макарова

О.М. ДУДЧЕНКО,
С.О. КАРПОВА

“CASE-ЗАСОБИ РОЗРОБКИ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ”

Рекомендовано Методичною радою НУК
як методичні вказівки до виконання лабораторних робіт

Миколаїв • НУК • 2018

УДК 004.4

Дудченко О.М., Карпова С.О. Case-засоби розробки та тестування програмного забезпечення. Методичні вказівки до виконання лабораторних робіт – Миколаїв: НУК, 2018,- 86 с.

Кафедра інформаційних технологій та фізико-математичних дисциплін.

Методичні вказівки призначені для студентів V курсу зі спеціальностей "Інженерія програмного забезпечення" та "Інформаційні управляючі системи та технології" Херсонської філії НУК. Метою циклу лабораторних робіт є практичне ознайомлення студентів з CASE-засобами розробки та тестування програмного забезпечення та надбання практичних навичок їх використання.

Кожна лабораторна робота складається з методичних вказівок, зразка її виконання та набору варіантів завдань.

Виконавши лабораторні роботи, студент має дізнатися, що таке CASE-засоби, які види моделей використовуються для опису бізнес-процесів. В результаті студент повинен навчитися будувати діаграми, що описують бізнес-процес в цих нотаціях, тестувати програмне забезпечення.

Рецензент: П.С. Носов, к.т.н., доцент

ВСТУП

Більшість існуючих методів об'єктно-орієнтованого аналізу та проектування (ООАП) включають мову моделювання й опис процесу моделювання. Мова моделювання - це нотація (в основному графічна), яка використовується методом для опису проектів.

Моделювання предметної галузі є одним з найбільш важливих етапів робіт при проектуванні програмних систем в масштабі підприємства. На даний час, для виконання задач моделювання предметної галузі, на ринку програмних продуктів представлено широкий спектр Computer Aided System / Software Engineering (CASE)-засобів.

CASE-засоби - це спеціальні програми, які підтримують одну або кілька методологій аналізу та проектування інформаційних систем.

CASE-технологія, включає в себе методи, за допомогою яких, на основі нотацій будуються діаграми, підтримувані конкретним CASE-засобом. CASE-технології не можуть вважатися самостійними, вони лише забезпечують високу ефективність їх застосування.

Стандартною нотацією для моделювання великих інформаційних систем (ІС) на основі об'єктно-орієнтованої методології служить уніфікована мова моделювання Unified Modeling Language (UML), яка підтримується низкою CASE-продуктів, одним з найбільш поширених є Rational Rose. Важливим аспектом успішного створення складної ІС є використання методології розробки проекту, в рамках якої, вводяться етапи роботи, ставляться завдання аналітикам, проектувальникам, програмістам, тестувальникам, системним інтеграторам в рамках методології Rational Unified Process (RUP). Концепція RUP та основні її елементи розроблені А. Джекобсоном в ході його роботи над мовою UML. В даний час продукт підтримується Rational Software.

Суть концепції RUP полягає в послідовній декомпозиції або розбитті процесу ООАП на окремі етапи, на кожному з яких здійснюється розробка відповідних типів канонічних діаграм моделі системи. При цьому на початкових етапах RUP будуються логічні уявлення статичної моделі структури системи, потім - логічні представлення моделі поведінки, та лише після цього - фізичні представлення моделі системи.

Одним з найважливіших етапів розробки, згідно RUP, є етап **визначення вимог**, який полягає в зборі всіх можливих побажань до роботи системи, які тільки можуть прийти в голову користувачам та аналітикам. Пізніше ці дані мають бути систематизовані та структуровані, але на даному етапі, в процесі інтерв'ю з користувачами та вивчення документів, аналітики повинні зібрати якомога більше вимог до майбутньої системи. Користувачі часто самі не уявляють, що вони повинні отримати в кінцевому підсумку. Для полегшення цього процесу аналітики використовують діаграму варіантів використання (use case).

Для деталізації конкретного варіанту використання використовується діаграма діяльності.

Для того щоб вірно визначити вимоги, розробники повинні розуміти контекст (частина предметної галузі) в якому працюватиме майбутня система. Для цього створюються модель предметної галузі та бізнес-модель, що є різними підходами до одного та того ж питання. Часто створюється щось одне: модель предметної галузі або бізнес-модель.

Відмінності цих моделей в тому, що модель предметної галузі описує важливі поняття, з якими буде працювати система та зв'язок їх між собою. Тоді як бізнес-модель описує бізнес-процеси (існуючі або майбутні), які повинна підтримувати система. Тому окрім визначення бізнес-об'єктів, залучених до процесу, ця модель визначає працівників, їх обов'язки та дії, які вони повинні виконувати.

Для створення моделі предметної галузі використовується звичайна діаграма класів, проте для створення бізнес-моделі її вже явно недостатньо. У

цьому випадку застосовується діаграма варіантів використання з використанням додаткових значків, які відображають сутність бізнес-процесів - це бізнес-актор, бізнес-варіант використання, бізнес-сутність і бізнес-управління.

Після визначення вимог і контексту, в якому працюватиме система, настає черга **аналізу** отриманих даних. У процесі аналізу створюється аналітична модель, яка підводить розробників до архітектури майбутньої системи. Аналітична модель - це погляд на систему зсередини, на відміну від моделі прецедентів, яка показує, як система буде виглядати зовні.

Ця модель дозволяє зрозуміти, як система повинна бути спроектована, які в ній мають бути класи й як вони мають взаємодіяти між собою. Основне її призначення - визначити напрям реалізації функціональності, виявленої на етапі збору вимог і зробити начерк архітектури системи.

На відміну від створюваної надалі моделі проектування, модель аналізу є переважно концептуальною моделлю та тільки наближає розробників до класів реалізації. Ця модель не повинна мати можливих протиріч, які можуть зустрітися в моделі прецедентів.

Для відображення моделі аналізу використовується діаграма класів зі стереотипами (зразками поведінки) «граничний клас», «сутність», «управління», а для деталізації використовуються діаграми кооперації. Якщо акцентувати увагу на порядок взаємодії, то іншим його поданням буде діаграма послідовності. Ця діаграма дозволяє поглянути на обмін повідомленнями в часі, наочно відобразити послідовність процесу.

Наступним етапом, у процесі створення системи, буде **проектування**, в ході якого на підставі моделей, створених раніше, створюється модель проектування. Ця модель відображає фізичну реалізацію системи й описує створюваний продукт на рівні класів і компонентів. На відміну від моделі аналізу, модель проектування має явно виражену залежність від умов реалізації, мов програмування та компонентів що застосовуються. Для максимально точного розуміння архітектури системи ця модель має бути

максимально формалізована та підтримуватися в актуальному стані протягом всього життєвого циклу розробки системи.

Для створення моделі проектування використовуються цілий набір діаграм: діаграми класів; діаграми кооперації; діаграми станів; діаграми діяльності.

Додатково в цьому робочому процесі може створюватися модель розгортання, яка реалізується на основі діаграми розгортання. Це найпростіший тип діаграм, призначений для моделювання розподілу пристроїв у мережі. Для відображення використовується всього два варіанти значків - процесор і пристрій разом зі зв'язками між ними.

Основне завдання процесу **реалізації** - створення системи у вигляді компонентів - вихідних текстів програм, сценаріїв, двоїчних файлів, виконуваних модулів тощо. На цьому етапі створюється модель реалізації, яка описує те, як реалізуються елементи моделі проектування, які класи будуть включені в конкретні компоненти. Дана модель описує спосіб організації цих компонентів відповідно до механізмів структурування та розбиття на модулі, прийнятими в обраному середовищі програмування та представляється діаграмою компонентів.

У процесі **тестування** перевіряються результати реалізації. Для даного процесу створюється модель тестування, яка складається з тестових прикладів, процедур тестування, тестових компонентів.

Лабораторна робота №1

РОБОТА У СЕРЕДОВИЩІ CASE-ЗАСОБУ VISUAL UML. РОЗРОБКА ДІАГРАМ USE CASE У СЕРЕДОВИЩІ CASE-ЗАСОБУ VISUAL PARADIGM FOR UML

Мета роботи: Ознайомлення з основними компонентами мови Visual Paradigm for UML. Вивчення об'єктно-орієнтованого моделювання та дослідження процесу побудови при створення моделі діаграми Use Case в середовищі Visual Paradigm for UML.

Короткі теоретичні відомості

Мова UML являє собою загальноцільову мову візуального моделювання, яка розроблена для специфікації, візуалізації, проектування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Мова UML одночасно є простим і потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних і графічних моделей складних систем різноманітного цільового призначення.

Конструктивне використання мови UML ґрунтується на розумінні загальних принципів моделювання складних систем та особливостей процесу об'єктно-орієнтованого проектування (ООП) зокрема. Вибір виразних засобів для побудови моделей складних систем зумовлює ті завдання, які можуть бути вирішені з використанням даних моделей. При цьому, одним з основних принципів побудови моделей складних систем є принцип абстрагування, який наказує включати в модель тільки ті аспекти проектованої системи, які мають безпосереднє відношення до виконання системою своїх функцій або

свого цільового призначення. Всі другорядні деталі опускаються, щоб надмірно не ускладнювати процес аналізу та дослідження отриманої моделі.

Іншим принципом побудови моделей складних систем - є принцип багатомодельності. Цей принцип базується на твердженні про те, що ніяка єдина модель не може з достатнім ступенем адекватності описувати різні аспекти складної системи.

Ще одним принципом прикладного системного аналізу є принцип ієрархічної побудови моделей складних систем. Цей принцип наказує розглядати процес побудови моделі на різних рівнях абстрагування або деталізації в рамках фіксованих представлень. При цьому вихідна або первісна модель складної системи має найбільш загальне уявлення (мета уявлення). Така модель будується на початковому етапі проектування та може не містити багатьох деталей й аспектів модельованої системи.

Мова UML прийнята на озброєння практично усіма найбільшими компаніями - виробниками ПЗ (Microsoft, IBM, Hewlett-Packard, Oracle, Sybase, ін.). Майже всі світові виробники CASE-засобів, крім Rational Software (Rational Rose), підтримують UML у своїх продуктах (Paradigm Plus 3.6, System Architect, Microsoft Visual Modeler for Visual Basic, Delphi, PowerBuilder та ін.).

Основні компоненти мови UML

Словник UML утворює три різновиди будівельних блоків: предмети, відносини, діаграми.

Предмети - це абстракції які є основними елементами в моделі, відносини пов'язують ці предмети, діаграми групують колекції предметів.

В UML є чотири різновиди предметів:

- структурні предмети;
- предмети поведінки;

- групируючі предмети;

- пояснюючі предмети.

Структурні предмети є іменниками в UML-моделях, це:

1) Клас (class) - опис безлічі об'єктів, які поділяють однакові властивості, операції, відносини та семантику (сенс). Клас реалізує один або декілька інтерфейсів (рис. 1.1). Графічно клас відображається у вигляді прямокутника, зазвичай включає секції з ім'ям, властивостями (атрибутами) й операціями.

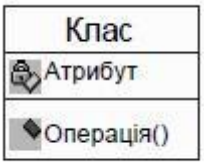


Рис. 1.1. Клас

2) Інтерфейс (interface) - набір операцій, які визначають послуги класу або компонента. Інтерфейс описує поведінку елемента, що видима ззовні. Графічно інтерфейс зображується у вигляді кружка з ім'ям (рис. 1.2). Ім'я інтерфейсу зазвичай починається з букви «I». Інтерфейс рідко показують самостійно. Зазвичай його приєднують до класу або компоненту, який реалізує інтерфейс.

IНавчання

Рис. 1.2. Інтерфейс

3) Кооперація (collaboration) визначає взаємодію й є сукупністю ролей й інших елементів, які працюють разом для забезпечення колективної поведінки більш складного процесу, ніж проста сума всіх його елементів.

Графічно кооперація зображується як пунктирний еліпс, в який вписується її ім'я (рис. 1.3).

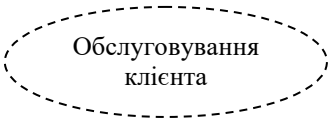
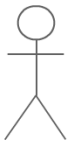


Рис. 1.3. Кооперація

4) Актор (actor) - набір узгоджених ролей, які можуть грати користувачі при взаємодії з системою (її елементами Use Case). Кожна роль вимагає від системи певної поведінки. Актор зображується як дротяна людинка з ім'ям (рис. 1.4).



Замовник

Рис. 1.4. Актор

5) Елемент Use Case (прецедент або варіант використання) - опис послідовності дій (або декількох послідовностей), що виконуються системою в інтересах окремого актора та здійснюють видимий для актора результат. Він зображується як еліпс, в який вписується ім'я (рис. 1.5).

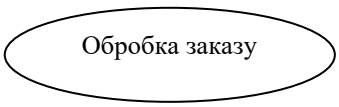


Рис. 1.5. Варіант використання

6) Активний клас (active class)- це клас, об'єкти якого мають один або декілька процесів (або потоків) і тому можуть ініціювати керуючу діяльність. Активний клас схожий на звичайний клас за винятком того, що його об'єкти діють одночасно з об'єктами інших класів. Активний клас зображується як

потовщений прямокутник, що зазвичай включає ім'я, властивості (атрибути) й операції (рис. 1.6) [8].

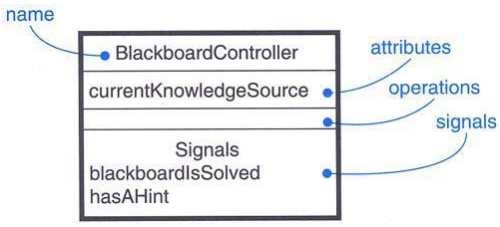


Рис. 1.6. Активний клас

7) Компонент (component) - фізична та замінна частина системи, яка відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів. Компонент зображується як прямокутник з вкладками, що включає ім'я (рис. 1.7).

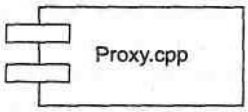


Рис. 1.7. Компонент

8) Вузол (node) - фізичний елемент, який існує в період роботи системи й є ресурсом, що має пам'ять і можливості обробки. Вузол зображується як куб з ім'ям (рис. 1.8).

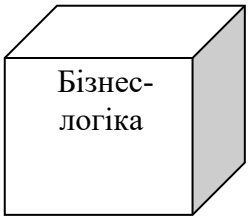


Рис. 1.8. Вузол

Предмети поведінки - динамічні частини UML-моделей. Вони є дієсловами моделей, поданням поведінки в часі та просторі. Існують дві основні різновиди предметів поведінки:

- Взаємодія - поведінка, містить в собі набір повідомлень (message), якими обмінюється набір об'єктів в конкретному контексті для досягнення певної мети. Елементами взаємодії є повідомлення, послідовність дій

(поведінка, що викликається повідомленням) та зв'язку (з'єднання між об'єктами). Повідомлення зображується у вигляді спрямованої лінії з ім'ям її операції (рис. 1.9).

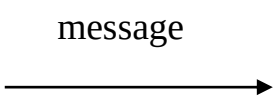


Рис. 1.9. Повідомлення

- Кінцевий автомат - поведінка, яка визначає послідовність станів об'єкта або взаємодії, що виконуються в процесі його існування у відповідь на події (з урахуванням обов'язків по цим подіям). Елементами кінцевого автомата є стани (state), переходи (від стану до стану), події (предмети, що викликають переходи) та дії (реакції на перехід). Стан (рис. 1.10) зображується як закруглений прямокутник, що зазвичай включає його ім'я та підстан (якщо він є).

Рис. 1.10. Стан

Ці два елементи - взаємодія та кінцевий автомат - є базисними предметами поведінки, які можуть включатися в UML-моделі.

Групируючі предмети - організаційні частини UML-моделей. Це ящики, за якими може бути розкладена модель. Передбачено один різновид згрупованого предмета - пакет.

Пакет (package) - загальний механізм для розподілу елементів по групах. У пакет можуть поміщатися структурні предмети, предмети поведінки та навіть інші угруповування предметів. Пакет зображується як папка із закладкою, на якій позначено його ім'я, а іноді й його зміст (рис. 1.11).

Рис. 1.11. Пакет

Пояснюючі предмети - роз'яснюють частини UML-моделей. Вони є зауваженнями, які можна застосувати для опису, пояснення та коментування будь-якого елементу моделі. Передбачено один різновид пояснюючого предмету - примітка (notes).

Примітка - символ для зображення обмежень і зауважень, що приєднуються до елементу або сукупності елементів. Примітка зображується у вигляді прямокутника із загнутим кутом, в якій вписується текстовий або графічний коментар (рис. 1.12).

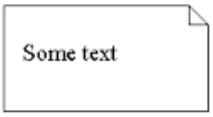


Рис. 1.12. Примітка

Відношення в UML. В UML є чотири різновиди відносин:

- Залежність (dependency relationship). Семантичне відношення між двома предметами, в якому зміна в одному предметі (незалежному предметі) може впливати на семантику іншого предмета (залежного предмета). Залежність зображується у вигляді пунктирної лінії, можливо спрямованої на незалежний предмет та іноді має мітку (рис. 1.13)

Рис. 1.13. Залежність

- Асоціація (association relationship). Структурне відношення, яке описує набір зв'язків, що є з'єднанням між об'єктами. Агрегація - це спеціальний різновид асоціації, який відображає структурне відношення між цілим та його частинами. Асоціація зображується у вигляді суцільної лінії,

можливо спрямованої, іноді має мітку та часто включає інші «прикраси», такі як потужність та імена ролей (рис. 1.14). Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано за допомогою одного примірника асоціації. Це число називається кратністю (multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Рис. 1.14. Асоціація

- Узагальнення (generalization relationship). Відношення спеціалізації/ узагальнення, в якому об'єкти спеціалізованого елемента (нащадка, дитини) можуть заміняти об'єкти узагальненого елемента (предка, батька). Інакше кажучи, нащадок розділяє структуру та поведінку батьків. Узагальнення зображується у вигляді суцільної стрілки з порожнистим наконечником, що вказує на батька (рис. 1.15).

Рис. 1.15. Узагальнення

- Реалізація **(**realization relationship**)** - семантичне відношення між класифікаторами, де один класифікатор визначає контракт, який інший класифікатор зобов'язується виконувати (до класифікаторів відносять класи, інтерфейси, компоненти, елементи Use Case, кооперації). Відносини реалізації застосовують у двох випадках: між інтерфейсами та класами (або компонентами), що реалізують їх; між елементами Use Case та кооперації, які реалізують їх. Реалізація зображується як щось середнє між узагальненням і залежністю (рис. 1.16).

Рис. 1.16. Реалізація

Діаграма в UML - це графічне представлення набору елементів, що зображується найчастіше у вигляді пов'язаного графа з вершинами (сутностями) та ребрами (відносинами). Діаграми будують для візуалізації системи з різних точок зору. Діаграма - в деякому розумінні одна з проекцій системи. Один і той же елемент може бути присутнім у всіх діаграмах, або тільки в декількох (найпоширеніший варіант), або не бути присутнім в жодній (дуже рідко).

В UML виділяють такі типи діаграм:

- діаграми варіантів використання (use case diagrams) - для моделювання бізнес-процесів організації (вимог до системи);

- діаграми класів (class diagrams) - для моделювання статичної структури класів системи та зв'язків між ними. На таких діаграмах показують класи, інтерфейси, об'єкти та кооперації, а також їх відносини. При моделюванні об'єктно-орієнтованих систем цей тип діаграм використовують найчастіше;

- діаграми станів (statechart diagrams) - для моделювання поведінки об'єктів системи при переході з одного стану в інший. На них представлений автомат, що включає в себе стани, переходи, події та види дій. Діаграми станів відносяться до динамічного виду системи; особливо вони важливі при моделюванні поведінки інтерфейсу, класу або кооперації. Також акцентують увагу на поведінці об'єкта, що залежить від послідовності подій, що дуже корисно для моделювання реактивних систем;

- діаграми взаємодії (interaction diagrams) - для моделювання процесу обміну повідомленнями між об'єктами. Існують два види діаграм взаємодії: діаграми послідовності (sequence diagrams) та діаграми кооперації (collaboration diagrams). На діаграмах взаємодії відображаються зв'язки між

об'єктами; показані, зокрема, повідомлення, якими об'єкти можуть обмінюватися. Діаграми взаємодії відносяться до динамічного виду системи. При цьому діаграми послідовності відображають тимчасову упорядкованість повідомлень, а діаграми кооперації - структурну організацію, обмінюються повідомленнями об'єктів. Ці діаграми є ізоморфними, тобто можуть бути перетворені одна в одну;

- діаграми реалізації (implementation diagrams): діаграми компонентів (component diagrams) - для моделювання ієрархії компонентів (підсистем) системи; діаграми розгортання (deployment diagrams) - для моделювання фізичної архітектури системи. На діаграмі компонентів представлена організація сукупності компонентів та існуючі між ними залежності. Вони можуть бути співвіднесені з діаграмами класів, так як компонент зазвичай відображається на один або декілька класів, інтерфейсів або кооперацій;

- діаграми діяльностей (activity diagrams) - для моделювання поведінки системи в рамках різних варіантів використання або моделювання діяльностей. Це окремий випадок діаграми станів, на ній представлені переходи потоку управління від однієї діяльності до іншої всередині системи. Діаграми діяльності відносяться до динамічного виду системи; вони найбільш важливі при моделюванні її функціонування та відображають потік управління між об'єктами.

Діаграма Use Case

Візуальне моделювання в UML можна представити як певний процес порівневого спуску від найбільш загальної й абстрактної концептуальної моделі вихідної системи до логічної, а потім і до фізичної моделі відповідної програмної системи. Для досягнення цих задач, спочатку будується модель у формі так званої діаграми варіантів використання (use case diagram), яка описує функціональне призначення системи або, іншими словами, те, що система буде виконувати в процесі свого функціонування.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей або акторів, що взаємодіють з системою за допомогою так званих варіантів використання. При цьому актором (actor) або дійовою особою називається будь-яка сутність, яка взаємодіє з системою ззовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, що може служити джерелом впливу на модельовану систему так, як визначить сам розробник. У свою чергу, варіант використання (use case) служить для опису сервісів, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконується системою при діалозі з актором.

Діаграма варіантів використання - граф спеціального виду, який є графічною нотацією для представлення конкретних варіантів використання, акторів, можливо, деяких інтерфейсів і відносин між цими елементами.

Елементами діаграми Use Case є:

- Варіант використання (use case). Позначається на діаграмі еліпсом, всередині якого міститься його коротка назва або ім'я у формі дієслова з пояснювальними словами. Мета варіанту використання полягає в тому, щоб визначити закінчений аспект або фрагмент поведінки деякої сутності без розкриття її внутрішньої структури. В якості такої сутності може виступати вихідна система або будь-який інший елемент моделі, який володіє власною поведінкою, подібно підсистемі або класу в моделі системи.

- Актор (actor) являє собою будь-яку зовнішню по відношенню до системи, що моделюється, сутність, яка взаємодіє з системою та використовує її функціональні можливості для досягнення певної мети або вирішення приватних завдань. При цьому актори служать для позначення узгодженого безлічі ролей, які можуть грати користувачі в процесі взаємодії з проектованою системою. Кожен актор може розглядатися як якась окрема роль щодо конкретного варіанту використання. Стандартним графічним позначенням актора на діаграмах є фігурка "людинки", під якою записується конкретне ім'я актора.

- Інтерфейс (interface) служить для специфікації параметрів моделі, які видимі ззовні без зазначення їх внутрішньої структури. На діаграмі варіантів використання інтерфейс зображається у вигляді маленького кола, поряд з яким записується його ім'я. Якщо ім'я записується англійською, то воно повинно починатися з великої літери I.

- Примітки (notes) в мові UML призначені для включення в модель довільної текстової інформації, що має безпосереднє відношення до контексту розроблюваного проекту. Графічно примітки позначаються прямокутником із "загнутим" верхнім правим куточком. Усередині прямокутника міститься текст примітки. Примітка може відноситися до будь-якого елементу діаграми, в цьому випадку їх з'єднує пунктирна лінія. Якщо примітка відноситься до кількох елементів, то від неї проводяться, відповідно, декілька ліній. Зрозуміло, примітки можуть бути присутні не тільки на діаграмі варіантів використання, а й на інших канонічних діаграмах.

- Відносини (association). В UML є декілька стандартних видів відносин між акторами та варіантами використання [9]:

Відношення асоціації (association relationship) служить для позначення специфічної ролі актора в окремому варіанті використання. Це відношення встановлює, яку конкретну роль грає актор при взаємодії з екземпляром варіанту використання. На діаграмі варіантів використання, відношення асоціації позначається суцільною лінією між актором і варіантом використання. Ця лінія може мати додаткові умовні позначення, такі, наприклад, як ім'я та кратність (рис. 1.17).

Рис. 1.17. Приклад графічного представлення відношення асоціації між актором і варіантом використання

Відношення розширення (extend relationship) визначає взаємозв'язок примірників окремого варіанту використання з більш загальним варіантом, властивості якого визначаються на основі способу спільного об'єднання даних екземплярів. Відношення розширення між варіантами використання позначається пунктирною лінією зі стрілкою, спрямованої від того варіанту використання, який є розширенням для початкового варіанту використання. Дана лінія зі стрілкою позначається ключовим словом "extend" ("розширює"), як показано на рисунку 1.18.

Рис. 1.18. Приклад графічного зображення відношення розширення між варіантами використання

Відношення узагальнення (generalization relationship) служить для вказівки того факту, що деякий варіант використання А може бути узагальнений до варіанту використання В. В цьому випадку варіант А буде спеціалізацією варіанту В. При цьому В називається предком або батьком по відношенню А, а варіант А - нащадком по відношенню до варіанту використання В. Слід підкреслити, що нащадок успадковує всі властивості та поведінку свого батька, а також може бути доповнений новими властивостями й особливостями поведінки. Графічно дане відношення позначається суцільною лінією зі стрілкою у формі незафарбованого трикутника, яка вказує на батьківський варіант використання (рис. 1.19). Ця лінія зі стрілкою має спеціальну назву - стрілка "узагальнення".

Рис. 1.19. Приклад графічного зображення відношення узагальнення між варіантами використання

Відношення включення (include relationship) між двома варіантами використання вказує, що деяка задана поведінка для одного варіанта використання включається в якості складового компонента в послідовність поведінки іншого варіанту використання. Графічно дане відношення позначається пунктирною лінією зі стрілкою (варіант відносини залежності), спрямованої від базового варіанту використання до такого, що включається. Дана лінія (рис. 1.20.) зі стрілкою позначається ключовим словом "include" ("включає").

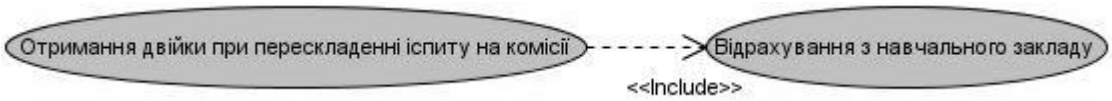


Рис. 1.20. Приклад графічного зображення відношення включення між варіантами використання

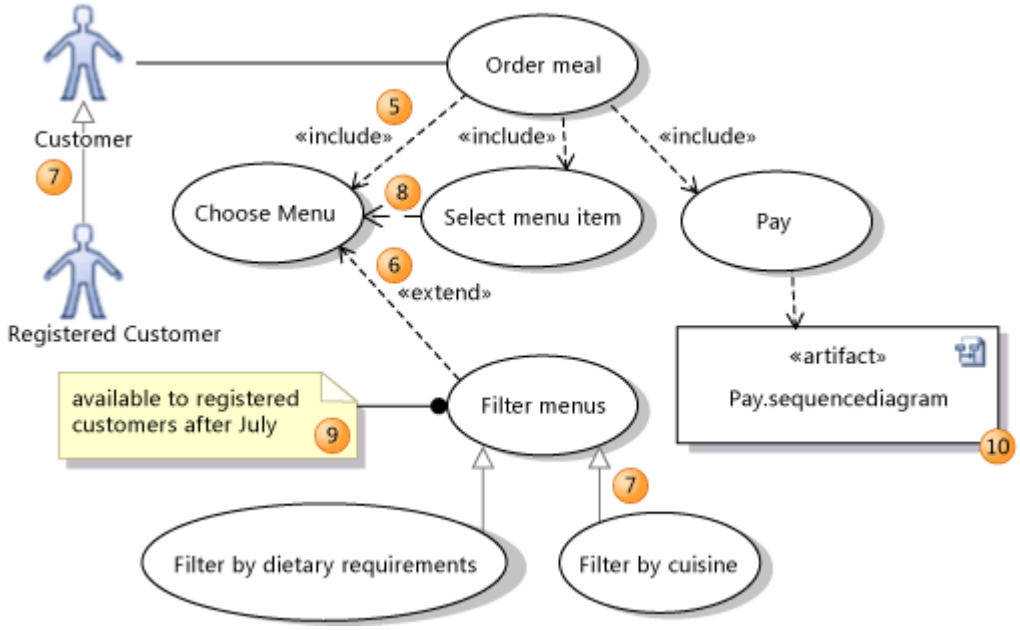


Рис. 1.21. Приклад графічного зображення відношень між варіантами використання [10]

Завдання

Побудувати за допомогою CASE-засобу Visual Paradigm for UML діаграму варіантів використання для заданого процесу (див. табл. 1). Номер варіанта завдання взяти за списком журналу відвідування занять.

Таблиця 1. Варіанти завдання

№ варіанту	**Назва процесу**
1	Книжковий магазин
2	Магазин промислових товарів
3	Написання програми
4	Центр замовлень
5	Поштова служба, включаючи службу доставки
6	Поштове відділення
7	Управління персоналом в компанії (відділ кадрів)
8	Управління компанією (заводом)
9	Центр міжміських (міжнародних) переговорів
10	Залізничний вокзал
11	Автовокзал
12	Автотранспортні послуги
13	ВНЗ або факультет ВНЗ
14	Школа
15	Спортивний клуб (організація й управління)
16	Бібліотека
17	Банк
18	Процес навчання деякому предмету (організація й управління)
19	Аеропорт
20	Склад

Лабораторна робота №2

РОЗРОБКА ДІАГРАМ ВЗАЄМОДІЇ У СЕРЕДОВИЩІ CASE-ЗАСОБУ VISUAL PARADIGM FOR UML

Мета роботи: Вивчення об'єктно-орієнтованого моделювання та дослідження процесу побудови та створення моделей діаграм послідовності та кооперації в середовищі Visual Paradigm for UML за допомогою її елементів.

Короткі теоретичні відомості

Для моделювання взаємодії об'єктів в мові UML використовуються відповідні діаграми взаємодії (interaction diagrams). Взаємодії об'єктів можна розглядати в часі, тоді для представлення тимчасових особливостей передачі та прийому повідомлень між об'єктами використовується діаграма послідовності. Часовий аспект поведінки може мати суттєве значення при моделюванні синхронних процесів, що описують взаємодії об'єктів. Саме для цієї мети в мові UML використовуються діаграми послідовності (sequence diagrams) див. рис. 2.1. Елементами даної діаграми є об'єкти та повідомлення.

На діаграмі послідовності зображуються виключно ті об'єкти, які безпосередньо беруть участь у взаємодії та не показуються можливі статичні асоціації з іншими об'єктами. Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів в часі. При цьому діаграма послідовності має як би два виміри. Один - зліва направо у вигляді вертикальних ліній, кожна з яких зображує лінію життя окремого об'єкта, який бере участь у взаємодії. Графічно кожен об'єкт зображується прямокутником і розташовується у верхній частині своєї лінії життя. Усередині прямокутника записуються ім'я об'єкту й ім'я класу, розділені

двокрапкою. При цьому весь запис підкреслюється, що є ознакою об'єкта, який, як відомо, являє собою екземпляр класу. Всі об'єкти на діаграмі послідовності утворюють деякий порядок, який визначається ступенем активності цих об'єктів при взаємодії один з одним.

Другий вимір діаграми послідовності - вертикальна тимчасова вісь, спрямована зверху вниз. Початковому моменту часу відповідає сама верхня частина діаграми. При цьому взаємодії об'єктів реалізуються за допомогою повідомлень, які посилаються одними об'єктами іншим. Повідомлення зображуються у вигляді горизонтальних стрілок з ім'ям повідомлення та також утворюють порядок за часом свого виникнення.

Лінія життя об'єкту (object lifeline) зображується пунктирною вертикальною лінією, асоційованою з єдиним об'єктом на діаграмі послідовності. Лінія життя служить для позначення періоду часу, протягом якого об'єкт існує в системі, отже, може потенційно брати участь у всіх її взаємодіях. Якщо об'єкт існує в системі постійно, то його лінія життя повинна тривати по всій площині діаграми послідовності від самої верхньої її частини до найнижчої.

У процесі функціонування об'єктно-орієнтованих систем одні об'єкти можуть перебувати в активному стані, безпосередньо виконуючи певні дії або в стані пасивного очікування повідомлень від інших об'єктів. Щоб явно виділити подібну активність об'єктів, в мові UML застосовується спеціальне поняття, яке отримало назву фокуса управління (focus of control). Фокус управління зображується у формі витягнутого вузького прямокутника, верхня сторона якого позначає початок отримання фокусу управління об'єкта (початок активності), а його нижня сторона - закінчення фокусу управління (закінчення активності). Цей прямокутник розташовується нижче позначення відповідного об'єкта та може замінювати його лінію життя, якщо на всій її довжині він є активним.

Кожна взаємодія описується сукупністю повідомлень, в яких об'єкти, що в них використовуються, обмінюються між собою. У цьому сенсі

повідомлення (message) виглядає як закінчений фрагмент інформації, який відправляється одним об'єктом до іншого. При цьому прийом повідомлення ініціює виконання певних дій, спрямованих на вирішення окремого завдання тим об'єктом, якому це повідомлення відправлено.

У мові UML можуть зустрічатися декілька різновидів повідомлень, кожне з яких має своє графічне зображення. Зазвичай повідомлення зображуються горизонтальними стрілками, що з'єднують лінії життя або фокуси управління двох об'єктів на діаграмі послідовності.

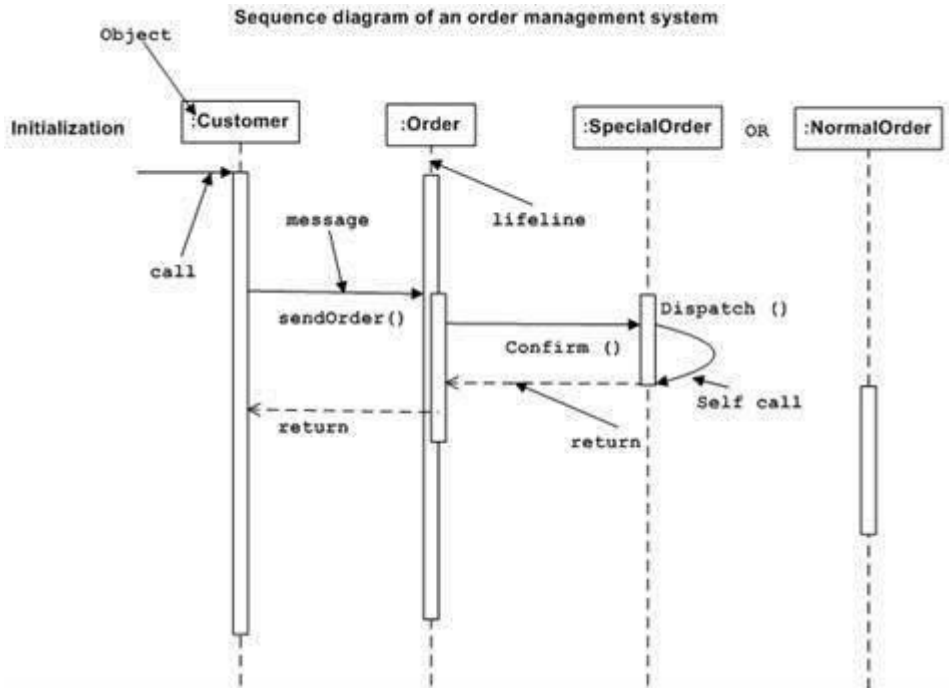


Рис. 2.1. Приклад діаграми послідовності [21]

Для подання структурних особливостей передачі та прийому повідомлень між об'єктами використовується діаграма кооперації (collaboration diagrams) див. рис. 2.2. Головна особливість діаграми кооперації полягає в можливості графічно представити не тільки послідовності взаємодії, але всі структурні відносини між об'єктами, які беруть участь в цій взаємодії. На діаграмі кооперації у вигляді прямокутників

зображуються об’єкти, які використовують у взаємодії об'єкти, що містять ім'я об'єкту, його клас, значення атрибутів.

На відміну від діаграми послідовності, на діаграмі кооперації зображуються тільки відносини між об'єктами, що грають певні ролі у взаємодії. З іншого боку, на цій діаграмі не вказується час у вигляді окремого виміру. Тому послідовність взаємодій та паралельних потоків може бути визначена за допомогою порядкових номерів. Отже, якщо необхідно явно специфікувати взаємозв'язки між об'єктами в реальному часі, краще це робити на діаграмі послідовності.

Поняття кооперації (collaboration) є одним з фундаментальних понять в мові UML. Воно служить для позначення безлічі взаємодіючих з певною метою об'єктів в загальному контексті, що моделюється. Мета самої кооперації полягає в тому, щоб специфікувати особливості реалізації окремих найбільш значущих операцій в системі.

Кооперація може бути представлена на двох рівнях:

1) на рівні специфікації - показує ролі класифікаторів і ролі асоціацій в розглянутому взаємодії.

2) на рівні прикладів - вказує екземпляри та зв'язки, що утворюють окремі ролі в кооперації.

Елементами діаграми кооперації є об'єкти, зв'язки, повідомлення.

Об'єкт (object) є окремим екземпляром класу, який створюється на етапі виконання програми. Об'єкти діляться на:

- мультіоб'єкти (multiobject) представляють собою безліч об'єктів на одному з кінців асоціації. На діаграмі кооперації мультіоб'єкт використовується для того, щоб показати операції та сигнали, які адресовані безлічі об'єктів, а не тільки одному;

- пасивні й активні об'єкти. Пасивний об'єкт оперує тільки даними та не може ініціювати діяльність з управління іншими об'єктами. Однак пасивні об'єкти можуть посилати сигнали в процесі виконання запитів, які вони

отримують. Активний об'єкт (active object) має свою власну нитку (thread) управління та може ініціювати діяльність з управління іншими об'єктами. При цьому під ниткою розуміється деякий полегшений потік управління, який може виконуватися паралельно з іншими обчислювальними нитками або нитками управління в межах одного обчислювального процесу або процесу управління.

- Складений об'єкт (composite object) або об'єкт-контейнер призначений для представлення об'єкту, що має власну структуру та внутрішні потоки (нитки) управління.

Зв'язок (link) є екземпляром або прикладом довільної асоціації. Зв'язок як елемент мови UML може мати місце між двома та більше об'єктами. Бінарний зв'язок на діаграмі кооперації зображується відрізком прямої лінії, що з'єднує два прямокутника об'єктів. На кожному з кінців цієї лінії можуть бути явно вказані імена ролей даної асоціації.

Повідомлення (massager), специфікує комунікацію між двома об'єктами, один з яких передає іншому певну інформацію. При цьому перший об'єкт очікує, що після отримання повідомлення другим об'єктом почнеться виконання певної дії. Таким чином, саме повідомлення є причиною або стимулом для початку виконання операцій, відправки сигналів, створення та знищення окремих об'єктів.

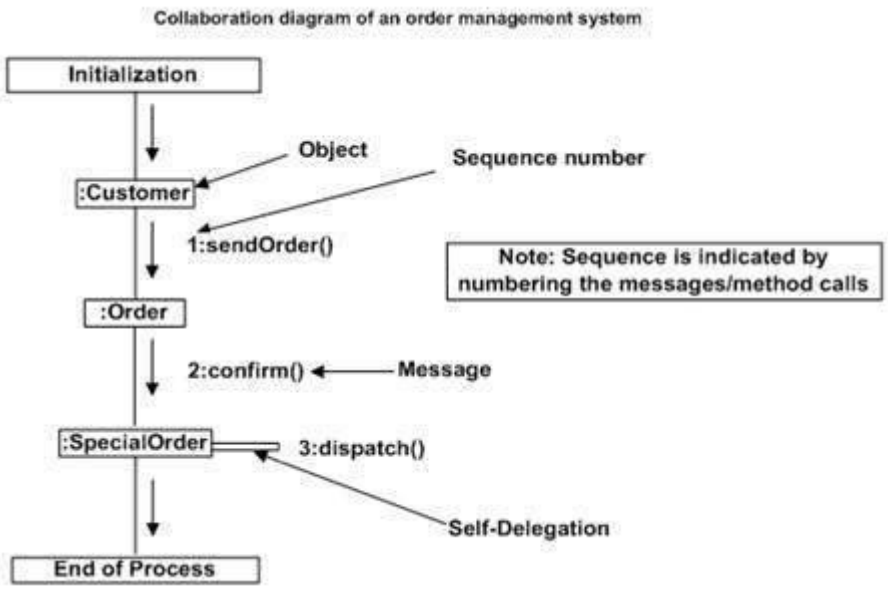


Рис. 2.2. Приклад діаграми кооперації [21]

Завдання

Побудувати за допомогою CASE-засобу Visual Paradigm for UML діаграму послідовності та кооперації для заданого процесу (див. табл. 1). Номер варіанта завдання взяти за списком журналу відвідування занять.

Лабораторна робота №3

РОЗРОБКА ДІАГРАМ КЛАСІВ У СЕРЕДОВИЩІ CASE-ЗАСОБУ VISUAL PARADIGM FOR UML

Мета роботи: Вивчення об'єктно-орієнтованого моделювання та дослідження процесу побудови та створення моделі діаграми класів в середовищі Visual Paradigm for UML за допомогою її елементів.

Короткі теоретичні відомості

Діаграма класів (class diagram) служить для подання статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відбивати, зокрема, різні взаємозв'язки між окремими сутностями предметної галузі, такими як об'єкти та підсистеми, а також описує їх внутрішню структуру та типи відносин. На даній діаграмі не вказується інформація про тимчасові аспекти функціонування системи. З цієї точки зору діаграма класів є подальшим розвитком концептуальної моделі проектованої системи.

Діаграма класів - це певний граф, вершинами якого є елементи типу "класифікатор", які пов'язані різними типами структурних відносин. Слід зауважити, що діаграма класів може також містити інтерфейси, пакети, відносини та навіть окремі екземпляри, такі як об'єкти та зв'язки. Коли говорять про дану діаграму, мають на увазі статичну структурну модель проектованої системи. Тому діаграму класів прийнято вважати графічним представленням таких структурних взаємозв'язків логічної моделі системи, що не залежать або інваріантні від часу. Елементами діаграми класів є:

Клас (class) в мові UML служить для позначення безлічі об'єктів, які мають однакову структуру, поведінку та відносини з об'єктами з інших

класів. Графічно клас зображується у вигляді прямокутника, який додатково може бути розділений горизонтальними лініями на розділи або секції. У цих розділах можуть зазначатися ім'я класу, атрибути (змінні) й операції (методи).

Ім'я класу має бути унікальним в межах пакету, який описується деякою сукупністю діаграм класів. Рекомендується в якості імен класів використовувати іменники. Ім'я класу записується по центру секції напівжирним шрифтом і має починатися з великої літери. Клас може не мати екземплярів або об'єктів. У цьому випадку він називається абстрактним класом, а для позначення його імені використовується курсив.

Ім'я атрибута є рядок тексту, який використовується в якості ідентифікатора відповідного атрибута та тому має бути унікальним в межах даного класу. Ім'я атрибута є єдиним обов'язковим елементом синтаксичного позначення атрибута.

Операція (operation) являє собою деякий сервіс, що надає кожен екземпляр класу за певною вимогою. Сукупність операцій характеризує функціональний аспект поведінки класу. Ім'я операції являє собою рядок тексту, який використовується в якості ідентифікатора відповідної операції, тому повинне бути унікальним в межах даного класу.

Базовими відносинами або зв'язками в мові UML є:

- ставлення залежності (dependency relationship);
- ставлення асоціації (association relationship);
- ставлення узагальнення (generalization relationship);
- ставлення реалізації (realization relationship).

Інтерфейси є елементами діаграми варіантів використання, вони розглядалися раніше. Однак, при побудові діаграми класів, окремі інтерфейси можуть уточнюватися та в цьому випадку для їх зображення

використовується спеціальний графічний символ - прямокутник класу з ключовим словом або стереотипом "interface".

Об'єкт (object) - це окремий екземпляр класу, який створюється на етапі виконання програми. Він має своє власне ім'я та конкретні значення атрибутів. Для графічного зображення об'єктів використовується такий же символ прямокутника, що й для класів.

Шаблон (template) або параметризований клас (parameterized class) призначений для позначення такого класу, який має один (або більше) нефіксований формальний параметр. Він визначає цілу групу або безліч класів, кожен з яких може бути отриманий зв'язуванням цих параметрів з дійсними значеннями. Зазвичай параметрами шаблонів служать типи атрибутів класів (цілі числа, перерахування, масив рядків та ін.).

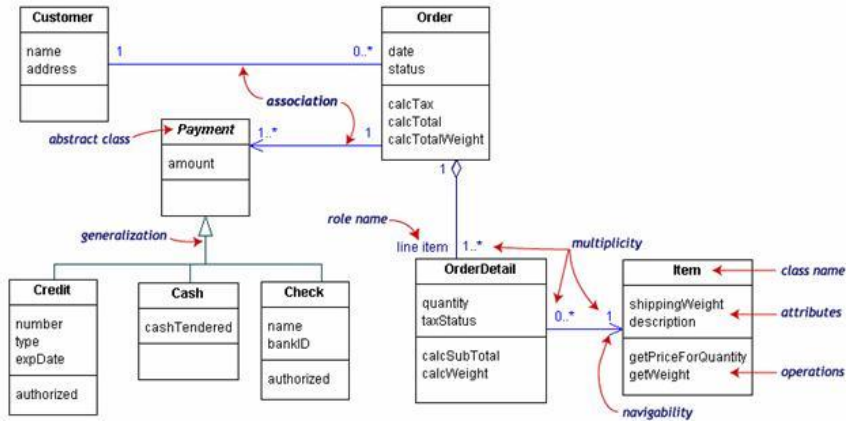


Рис. 3.1. Приклад діаграми класів [15]

Завдання

Побудувати за допомогою CASE-засобу Visual Paradigm for UML діаграму класів для заданого процесу (див. табл. 1). Номер варіанта завдання взяти за списком журналу відвідування занять.

Лаботаторна робота №4

РОЗРОБКА ДІАГРАМ СТАНІВ І ДІЯЛЬНОСТІ У СЕРЕДОВИЩІ CASE-ЗАСОБУ VISUAL PARADIGM FOR UML

Мета роботи: Вивчення об'єктно-орієнтованого моделювання та дослідження процесу побудови та створення моделі діаграми станів і діяльності в середовищі Visual Paradigm for UML за допомогою її елементів.

Короткі теоретичні відомості

Для моделювання поведінки на логічному рівні в мові UML можуть використовуватися відразу кілька канонічних діаграм: станів; діяльності; послідовності та кооперації, кожна з яких фіксує увагу на окремому аспекті функціонування системи. На відміну від інших діаграм, діаграма станів описує процес зміни станів тільки одного класу, а точніше - одного примірника певного класу, тобто моделює всі можливі зміни в стані конкретного об'єкта. При цьому зміна стану об'єкта може бути викликана зовнішніми впливами з боку інших об'єктів або ззовні. Саме для опису реакції об'єкту на подібні зовнішні впливи використовуються діаграми станів (statechart diagram).

Головне призначення цієї діаграми - описати можливі послідовності станів і переходів, які в сукупності характеризують поведінку елемента моделі протягом його життєвого циклу. Діаграма станів представляє динамічну поведінку сутностей, на основі специфікації їх реакції на сприйняття деяких конкретних подій. Системи, які реагують на зовнішні дії від інших систем або від користувачів, іноді називають реактивними. Якщо такі дії ініціюються в довільні випадкові моменти часу, то говорять про асинхронну поведінку моделі.

Діаграма станів по суті є графом спеціального виду, який представляє певний автомат. Поняття автомата в контексті UML має досить специфічну семантику, засновану на теорії автоматів. Вершинами цього графа є стан і деякі інші типи елементів автомата (псевдостан), які зображуються відповідними графічними символами. Дуги графа служать для позначення переходів зі стану в стан. Діаграми станів можуть бути вкладені одна в одну, утворюючи вкладені діаграми більш детального уявлення окремих елементів моделі. Для розуміння семантики конкретної діаграми станів необхідно представляти не тільки особливості поведінки модельованої сутності, а й знати загальні відомості з теорії автоматів. Елементами діаграми станів є:

- Автомат (state machine) в мові UML є певним формалізмом для моделювання поведінки елементів моделі та системи в цілому. В метамоделі UML автомат є пакетом, в якому визначено безліч понять, необхідних для подання поведінки модельованої сутності у вигляді дискретного простору з кінцевим числом станів і переходів. З іншого боку, автомат описує поведінку окремого об'єкта в формі послідовності станів, які охоплюють всі етапи його життєвого циклу, починаючи від створення об'єкта та закінчуючи його знищенням. Кожна діаграма станів представляє деякий автомат.

- Стан (state). У мові UML під станом розуміється абстрактний метаклас, що використовується для моделювання окремої ситуації, протягом якої має місце виконання деякої умови. Стан може бути задано у вигляді набору конкретних значень атрибутів класу або об'єкта, при цьому зміна їх окремих значень буде відображати зміну стану модельованого класу або об'єкта. Стан на діаграмі зображується прямокутником із закругленими вершинами.

Стан ділитися на:

а) початковий стан (start state), являє собою окремий випадок стану, який не містить ніяких внутрішніх дій (псевдостан).

б) Кінцевий (фінальний) (final state) стан являє собою окремий випадок стану, який також не містить ніяких внутрішніх дій (псевдостан).

Рис. 5.1. Графічне зображення початкового та кінцевого станів:

а - початковий стан;

б- кінцевий стан

- Перехід (transition). Простий перехід (simple transition) є відношення між двома послідовними станами, яке вказує на факт зміни одного стану іншим. На діаграмі станів перехід зображується суцільною лінією зі стрілкою, яка направлена в цільовий стан.

- Термін подія (event) вимагає окремого пояснення, оскільки є самостійним елементом мови UML. Формально, подія є специфікацію деякого факту, що має місце в просторі та в часі. Про події говорять, що вони "відбуваються", при цьому окремі події повинні бути впорядковані в часі. Після настання деякої події не можна вже повернутися до попередніх подій, якщо така можливість не передбачена явно в моделі. У мові UML події грають роль стимулів, які ініціюють переходи з одних станів в інші.

- Складений стан і підстан. Складений стан (composite state) – це стан, який складається з інших вкладених в нього станів. Останні будуть виступати по відношенню до першого як підстани (substate). Хоча між ними має місце відношення композиції. Графічно всі вершини діаграми, які відповідають вкладеним станам, зображуються всередині символу складеного стану. У цьому випадку розміри графічного символу складеного стану збільшуються, так щоб вмістити в себе всі підстани.

- Історичний стан (history state) застосовується в контексті складеного стану. Він використовується для запам'ятовування тих з послідовних підстанів, які були поточними в момент виходу з складеного стану. При цьому існує два різновиди історичного стану: недавнє та давнє. Історичний стан втрачає свою історію в той момент, коли підавтомат доходить до свого кінцевого стану. При цьому недавній історичний стан запам'ятовує історію

тільки того підавтомату, до якого він належить. Іншими словами, цей тип стану здатний запам'ятати історію тільки одного з ним рівня вкладеності.

- Складні переходи. В окремих випадках перехід може мати кілька станів-джерел і кілька цільових станів. Такий перехід отримав спеціальну назву - паралельний перехід (concurrent substates). Введення в розгляд паралельного переходу зумовлено необхідністю синхронізувати та/або розділити окремі підпроцеси на паралельні нитки без специфікації додаткової синхронізації в паралельних підавтоматах.

Перехід, стрілка якого сполучена з кордоном деякого складеного стану, позначає перехід до складеного стану. Він еквівалентний переходу в початковий стан кожного з підавтоматів (можливо, єдиному), що входять до складу даного суперстану.

Синхронізуючий стан (synch state) позначається невеликою окружністю, всередині якої поміщений символ зірочки "*". Він використовується спільно з переходом-з'єднанням або переходом-розгалуженням для того, щоб явно вказати події в інших підавтоматах, які безпосередньо впливають на поведінку даного підавтомату.

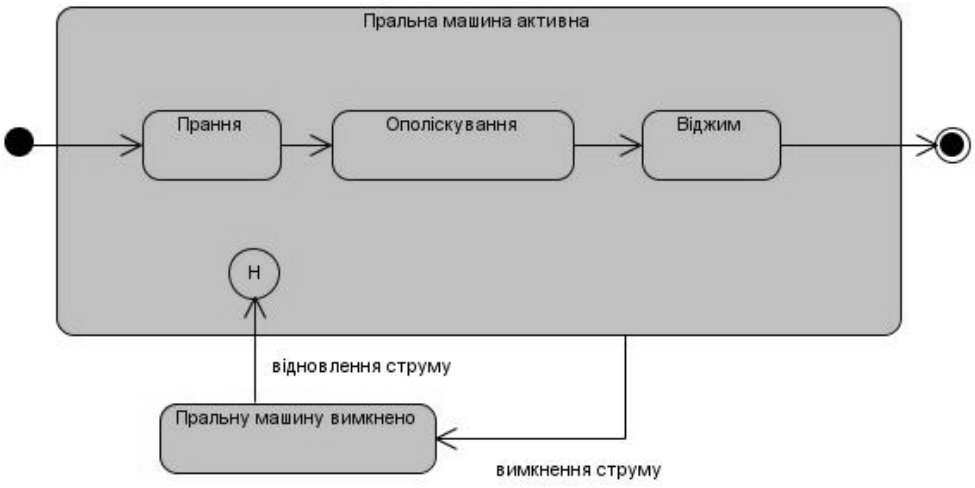


Рис. 5.1. Графічне зображення діаграми станів функціонування пральної машини [9]

Для моделювання процесу виконання операцій в мові UML використовуються так звані діаграми діяльності (activity diagrams) рис. 5.2. Застосовувана в них графічна нотація багато в чому схожа на нотацію діаграми станів, оскільки на діаграмах діяльності також присутні позначення станів і переходів. Відмінність полягає в семантиці станів, які використовуються для уявлення не діяльності, а дій, у відсутності на переходах сигнатури подій. Кожний стан на діаграмі діяльності відповідає виконанню деякої елементарної операції, а перехід в наступний стан спрацьовує тільки при завершенні цієї операції в попередньому стані. Графічно діаграма діяльності зображується у формі графа діяльності, вершинами якого є стани дії, а дуги - переходи від одного стану дії до іншого.

Діаграми діяльності можна вважати окремим випадком діаграм станів. Саме вони дозволяють реалізувати в мові UML особливості процедурного та синхронного управління, обумовленого завершенням внутрішніх діяльностей та дій.

У контексті мови UML діяльність (activity) являє собою деяку сукупність окремих обчислень, виконуваних автоматом. При цьому окремі елементарні обчислення можуть призводити до деякого результату або дії (action). На діаграмі діяльності відображається логіка або послідовність переходу від однієї діяльності до іншої, при цьому увага фіксується на результаті діяльності. Сам же результат може привести до зміни стану системи або повернення деякого значення.

Елементи діаграми діяльності:

- Стан дії (action state) є спеціальним випадком стану з деякою вхідною дією та принаймні одним вихідним зі стану переходом. Цей перехід неявно передбачає, що вхідна дія вже завершилася. Стан дії не може мати внутрішніх переходів, оскільки є елементарним. Звичайне використання стану дії полягає в моделюванні одного кроку виконання алгоритму

(процедури) або потоку управління. Графічно стан дії зображується фігурою, що нагадує прямокутник, бічні сторони якого замінені опуклими дугами. Усередині цієї фігури записується вираз дії (action-expression), який повинен бути унікальним в межах однієї діаграми діяльності.

- Переходи (transition). При побудові діаграми діяльності використовуються тільки нетригерні переходи, це такі, які спрацьовують одразу після завершення діяльності або виконання відповідної дії. Цей перехід переводить діяльність в подальший стан відразу, як тільки закінчиться дія в попередньому стані. На діаграмі такий перехід зображується суцільною лінією зі стрілкою.

- Доріжки (swimlanes). Для моделювання в мові UML використовується спеціальна конструкція, яка отримала назву доріжки (swimlanes). Мається на увазі візуальна аналогія з плавальними доріжками в басейні, якщо дивитися на відповідну діаграму. При цьому їхні дії на діаграмі діяльності діляться на окремі групи, які відділяються одна від одної вертикальними лініями. Дві сусідні лінії й утворюють доріжку, а група станів між цими лініями виконується окремим підрозділом (відділом, групою, відділенням, філією).

- Об'єкти (object). У загальному випадку дії на діаграмі діяльності виконуються над тими чи іншими об'єктами. Ці об'єкти або ініціюють виконання дій, або визначають деякий результат цих дій. При цьому дії специфікують виклики, які передаються від одного об'єкта графа діяльності до іншого.

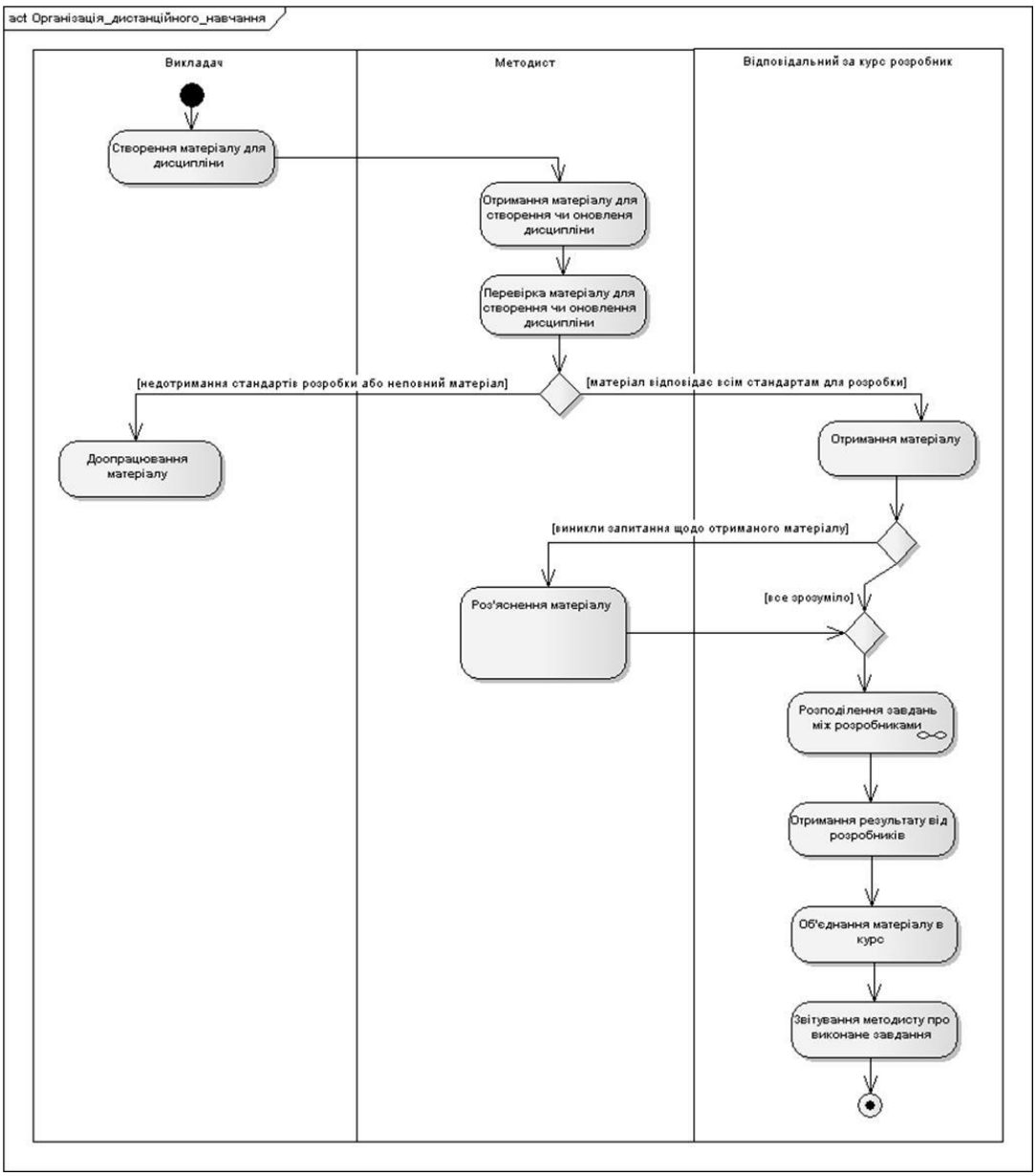


Рис. 5.2. Приклад діаграми діяльності процесу
«Організація дистанційного навчання» [12]

Завдання

Побудувати за допомогою CASE-засобу Visual Paradigm for UML діаграму станів і діяльності для заданого процесу (див. табл. 1). Номер варіанта завдання взяти за списком журналу відвідування занять.

Лабораторна робота №5

РОЗРОБКА ДІАГРАМ РЕАЛІЗАЦІЇ У СЕРЕДОВИЩІ CASE-ЗАСОБУ VISUAL PARADIGM FOR UML

Мета роботи: Вивчення об'єктно-орієнтованого моделювання та дослідження процесу побудови та створення моделей діаграм компонентів і розгортання в середовищі Visual Paradigm for UML за допомогою їх елементів.

Короткі теоретичні відомості

У мові UML для фізичного представлення моделей систем використовуються так звані діаграми реалізації (implementation diagrams), які включають в себе дві окремі канонічні діаграми: діаграму компонентів (component diagram) і діаграму розгортання (deployment diagram).

Діаграма компонентів дозволяє визначити архітектуру розроблюваної системи, встановивши залежності між програмними компонентами, в ролі яких може виступати вихідний, бінарний та виконуваний код. У багатьох середовищах розробки модуль або компонент відповідає файлу. Пунктирні стрілки, що з'єднують модулі, показують відношення взаємозалежності, аналогічно тим, які мають місце при компіляції початкового програмного коду. Основними графічними елементами діаграми компонентів є:

- *Компонент (component)* реалізує деякий набір інтерфейсів і служить для загального позначення елементів фізичного представлення моделі. Для графічного представлення компонента може використовуватися спеціальний символ - прямокутник зі вставленими зліва двома дрібнішими прямокутниками (рис. 5.1).

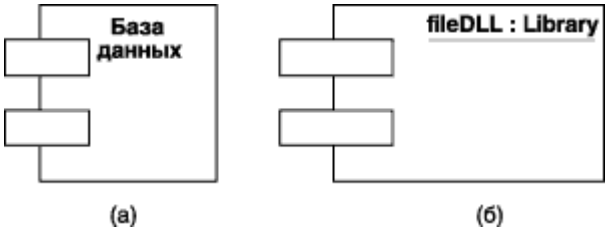


Рис. 5.1. Графічне зображення компонента в мові UML

Так, в першому випадку (рис. 5.1, а) з компонентом рівня екземпляра зв'язується тільки його ім'я, а в другому (рис. 5.1, б) - додатково ім'я пакету та помічене значення.

У мові UML виділяють три види компонентів:

1) компоненти розгортання, які забезпечують безпосереднє виконання системою своїх функцій. Такими компонентами можуть бути спільні бібліотеки з розширенням .dll (рис. 5.2, а), Web-сторінки на мові розмітки гіпертексту з розширенням .html (рис. 5.2, б) і файли довідки з розширенням .hlp (рис. 5.2, в).

2) компоненти-робочі продукти. Це файли з вихідними текстами програм, наприклад, з розширеннями .h або .cpp для мови C ++ (рис. 5.2, г).

3) компоненти виконання, що представляють виконані модулі - файли з розширенням exe. Вони позначаються звичайним чином.

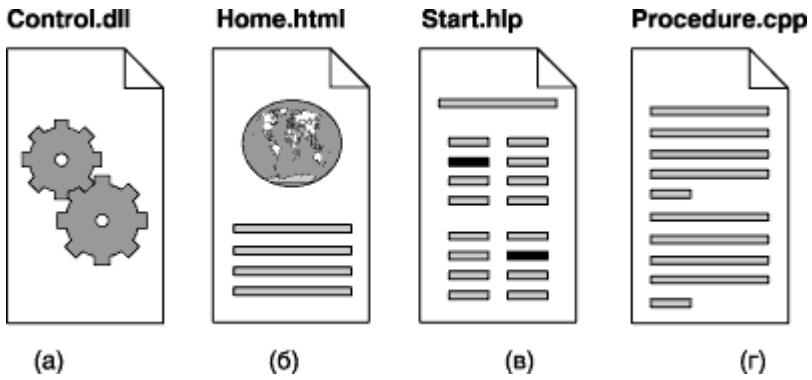


Рис. 5.2. Варіанти графічного зображення компонентів
на діаграмі компонентів

- Інтерфейс (interface). Інтерфейс графічно зображується колом, яке з'єднується з компонентом відрізком лінії без стрілок (рис. 5.3, а). При цьому ім'я інтерфейсу, яке обов'язково має починатися з великої літери "I", записується поряд з колом. Семантично лінія означає реалізацію інтерфейсу, а наявність інтерфейсів у компонента означає, що даний компонент реалізує відповідний набір інтерфейсів.

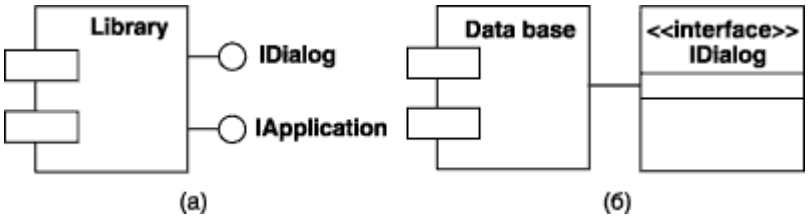


Рис. 5.3. Графічне зображення інтерфейсів
на діаграмі компонентів

Іншим способом представлення інтерфейсу на діаграмі компонентів є його зображення у вигляді прямокутника класу зі стереотипом "інтерфейс" і можливими секціями атрибутів та операцій (рис. 5.3, б). Як правило, цей варіант позначення використовується для представлення внутрішньої структури інтерфейсу, яка може бути важлива для реалізації.

- Залежності. Залежність служить для подання лише факту наявності такого зв'язку, коли зміна одного елемента моделі впливає або приводить до зміни іншого елемента моделі. Ставлення залежності на діаграмі компонентів зображається пунктирною лінією зі стрілкою, спрямованої від клієнта (залежного елемента) до джерела (незалежному елементові). Залежності можуть відображати зв'язки модулів програми на етапі компіляції та генерації об'єктного коду.

Діаграма розгортання застосовується для уявлення загальної конфігурації та топології розподіленої програмної системи та містить розподіл компонентів по окремих вузлах системи. Крім того, діаграма

розгортання показує наявність фізичних з'єднань - маршрутів передачі інформації між апаратними пристроями, задіяними в реалізації системи.

Діаграма розгортання призначена для візуалізації елементів і компонентів програми, існуючих лише на етапі її виконання (runtime). При цьому подаються тільки компоненти-екземпляри програми, які є здійснимими файлами або динамічними бібліотеками. Ті компоненти, які не використовуються на етапі виконання, на діаграмі розгортання не показуються. Так, компоненти з вихідними текстами програм можуть бути присутніми тільки на діаграмі компонентів. На діаграмі розгортання вони не вказуються.

Діаграма розгортання містить графічні зображення процесорів, пристроїв, процесів і зв'язків між ними. На відміну від діаграм логічного представлення, діаграма розгортання є єдиною для системи в цілому, оскільки повинна цілком відображати особливості її реалізації. Ця діаграма, по суті, завершує процес ООАП для конкретної програмної системи, а її розробка, як правило, є останнім етапом специфікації моделі.

Елементами діаграми розгортання є:

- Вузол (node) - це деякий фізично існуючий елемент системи, що володіє деяким обчислювальним ресурсом. В якості обчислювального ресурсу вузла може розглядатися наявність щонайменше деякого об'єму електронної або магнітооптичної пам'яті та/або процесора. Графічно на діаграмі розгортання вузол зображається у формі тривимірного куба (строго кажучи, псевдо тривимірного прямокутного паралелепіпеда). Вузол має власне ім'я, яке вказується всередині цього графічного символу.

- З'єднання. Є різновидом асоціації та зображуються відрізками ліній без стрілок. Наявність такої лінії вказує на необхідність організації фізичного каналу для обміну інформацією між відповідними вузлами. Характер з'єднання може бути додатково специфікований приміткою, поміченим значенням або обмеженням.

Завдання

Побудувати за допомогою CASE-засобу Visual Paradigm for UML діаграми компонентів і розгортання для заданого процесу (див. табл.1). Номер варіанта завдання взяти за списком журналу відвідування занять.

Лабораторна робота №6

РОБОТА У СЕРЕДОВИЩІ CASE- АСОБУ RATIONAL ROSE. ГЕНЕРАЦІЯ ПРОГРАМНОГО КОДУ C++, JAVA IDL

Мета роботи: Ознайомлення з CASE-засобом Rational Rose та його основними компонентами. За допомогою побудованих раніше видів діаграм (класів, варіантів використання, розгортання, компонентів, послідовності) в середовищі Rational Rose виконати перевірку моделі на відсутність помилок і вивчити процес кодогенерації.

Короткі теоретичні відомості

Rational Software - компанія-розробник програмного забезпечення. До 2003 року Rational була незалежною компанією, в 2003 році її поглинула корпорація IBM. Більшість продуктів компанії призначені для моделювання, а також для розробки та підтримки програмного забезпечення. У компанії розроблена методологія розробки програмного забезпечення Rational Unified Process (RUP). У методології надаються рекомендації по всіх етапах розробки: від моделювання бізнесу до тестування та здачі в експлуатацію готової програми. Особливе місце в RUP займають проектування та конфігураційне управління. Особливо виділяються вони тому, що ці два інструменти, підтримуються на даних етапах (Rational Rose та Rational ClearCase), використовуються протягом всього життєвого циклу розробки програмного забезпечення.

Продукти компанії Rational Software:

- Rational Rose - засіб моделювання;

- Rational Software Architect - засіб моделювання, подальший розвиток Rational Rose (на платформі Eclipse);

- Rational PurifyPlus - набір програм для виловлювання витоків пам'яті, аналізу області перекриття коду та продуктивності коду;

- Rational ClearCase - система управління версіями;

- Rational RequisitePro - система управління вимогами;

- Rational ClearQuest - система управління змінами;

- SoDA - система автоматизованого документування та звітності;

- Rational Robot та Rational Functional Tester - засіб автоматизованого тестування;

- Rational Performance Tester - засіб автоматизованого тестування навантаження;

- Rational Process Advisor - інструмент інтеграції процесу розробки ПЗ за допомогою інструментів розробки та тестування;

- Rational Application Developer - середовище розробки ПЗ (надбудова над Eclipse).

Rational Rose, будучи об'єктно-орієнтованим засобом проектування, здатна моделювати ситуації будь-якої складності: від проектування банківської системи до розробки коду на C ++.

Інструментарій програми допускає, як високорівневе (абстрактне) уявлення (наприклад, схема автоматизації підприємства), так і низькорівневе проектування (інтерфейс програми, схема бази даних, частковий опис класів). Вся міць програми базується всього на 7 діаграмах, які в залежності від ситуації, здатні описувати різні дії:

- Activity diagram (діаграми діяльності);

- Use case diagram (діаграми варіантів використання);

- Class diagram (діаграми класів);

- State diagram (діаграми станів);

- Sequence diagram (діаграми послідовності);

- Collaboration diagram (діаграми кооперації);

- Component diagram (діаграми компонентів);

- Deployment diagram (діаграма розгортання).

Rational Rose є інструментом, який дозволяє будувати зазначені діаграми при проектуванні програмних систем.

CASE-засіб Rational Rose з часу своєї появи зазнав серйозну еволюцію та перетворився на сучасний та потужний засіб аналізу, моделювання та розробки програмних систем. Саме в Rational Rose мова UML стала базовою технологією візуалізації та розробки програм, що визначило популярність і стратегічну перспективність цього інструментарію.

В рамках Rational Rose існують різні програмні інструментарії, що відрізняються між собою діапазоном реалізованих можливостей.

Базові засоби Rational Rose:

- Rational Rose Enterprise;
- IBM Rational Rose Developer for Visual Studio software;
- IBM Rational Rose Data Modeler;
- IBM Rational Rose Technical Developer software;
- IBM Rational Rose Developer for Linux/ UNIX software;
- IBM Rational Rose Developer for Java™ software;
- IBM Rational Rose Modeler.

Одним з найбільш важливих властивостей програми IBM Rational Rose є можливість генерації програмного коду на декількох мовах програмування, яка може бути використана розробником після побудови моделі. Для цієї мети в середовищі IBM Rational Rose є великий вибір мов програмування та схем баз даних. Однак можливість генерації тексту програми на тій чи іншій мові програмування залежить від встановленої версії IBM Rational Rose.

Загальна послідовність дій, які необхідно виконати для генерації програмного коду в середовищі IBM Rational Rose, складається з наступних етапів:

1. Перевірка моделі на відсутність помилок.
2. Створення компонентів для реалізації класів.
3. Відображення класів на компоненти.
4. Вибір мови програмування для генерації тексту програмного коду.
5. Встановлення властивостей генерації програмного коду.
6. Вибір класу, компонента або пакета.
7. Генерація програмного коду.

Завдання

Ознайомитися із середовищем IBM Rational Rose, його компонентами. Виконати генерацію кода по одному із видів діаграм побудованих в середовищі Rational Rose (діаграма класів, діаграма варіантів використання, діаграма розгортання, діаграма компонентів, діаграма послідовності).

Лабораторна робота №7

ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета роботи: Ознайомитися з методами та видами тестування ПЗ і провести тестування програмного продукта.

Короткі теоретичні відомості

Тестування програмного забезпечення (software testing) - процес дослідження, випробування програмного продукту, що має дві різні цілі:

- продемонструвати розробникам і замовникам, що програма відповідає вимогам;

- виявити ситуації, в яких поведінка програми є неправильною, небажаною або не відповідає специфікації.

Верифікація (verification) - це процес оцінки системи або її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються наші цілі, терміни, завдання по розробці проекту, визначені на початку поточної фази.

Валідація (validation) - це визначення відповідності розробляємого ПЗ очікуванням і потребам користувача, вимогам до системи.

План тестування (test plan) - це документ, що описує весь обсяг робіт з тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку та закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх вирішення.

Тест дизайн (test design) - це етап процесу тестування ПО, на якому проектуються та створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріїв якості та цілей тестування.

Тестовий випадок (test case) - це артефакт, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестуємой функції або її частини.

Баг / дефект репорт (bug report) - це документ, що описує ситуацію або послідовність дій, яка призвела до некоректної роботи об'єкта тестування, із зазначенням причин та очікуваного результату.

Тестове покриття (test coverage) - це одна з метрик оцінки якості тестування, що вдає із себе щільність покриття тестами вимог або виконуваного коду.

Деталізація тест кейсів (test case specification) - це рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття.

Час проходження тест кейса (test case pass time) - це час від початку проходження кроків тест кейса до отримання результату тесту [24].

Види тестування програмного забезпечення

Всі види тестування програмного забезпечення, в залежності від переслідуваних цілей, можна умовно розділити на наступні групи:

- Функціональні;
- Нефункціональні;
- Пов'язані зі змінами.

Говорячи про якість комп'ютерних програм, дотримуються трактування, якє викладено в стандарті ISO 9126. Цей стандарт виділяє 6 аспектів якості, у кожного з яких ще виділяється певна кількість підхарактеристик:

- Функціональність. Основний аспект якості. Найбільш важливою його підхарактеристикою є придатність до використання (suitability). Програма повинна робити це правильно (accuracy). Тобто обчислювати з певною точністю, правильно конвертувати дані. Програма повинна мати

здатність до взаємодії з іншими програмами (interoperability), з операційною системою, з іншими версіями тієї ж самої програми, зокрема, вона повинна підтримувати якісь стандарти (compliance), відповідати певним правилам.

Так сама програма не повинна руйнувати дані, не повинна заважати працювати іншим програмам, не повинна надавати доступ до даних тим, кому цей доступ не дозволений.

- Надійність. До нього відносяться такі підхарактеристики, як зрілість (maturity), яка є зворотною величиною до частоти відмов, і стійкість до відмов (fault tolerance), тобто здатність системи не реагувати на якісь внутрішні проблеми. У тому числі сюди відноситься транзакційна цілісність і здатність до відновлення працездатності при відмовах (recoverability).

- Практичність. Це зрозумілість програми (understandability), тобто користувач повинен зрозуміти, як скористатися нею для досягнення своїх цілей. Програма повинна бути зручною для навчання (learnability) або вивчення. Програма повинна бути працездатною (operability) або керованої, тобто користувач повинен мати можливість керувати поведінкою програми, вона повинна реагувати на його дії. Також програма повинна бути привабливою (attractiveness).

- Ефективність (продуктивність). Сюди відносяться тимчасові характеристики (time behaviour), час відгуку, швидкість роботи програми, швидкість обробки певних даних і так далі. І використання ресурсів (resource utilisation), використання дискових ресурсів, використання ресурсів процесора, використання оперативної пам'яті, використання мережевих ресурсів.

- Супроводження. Цей аспект якості більшою мірою не зовнішній, а внутрішній, він важливий самим розробникам і тестувальникам. Для контролю якості застосовуються такі засоби, як, статичний аналіз коду, код рев'ю, аналіз документації. Має на увазі аналізований код (analyzability), змінність (changeability), тобто зручність внесення змін до програмного коду, ризик виникнення несподіваних ефектів після того, як ці зміни внесли

(stability), а також контрольованість (testability), або ж - зручність тестування програми.

- Переносимість (або мобільність). Програма повинна вміти працювати в різних середовищах (adaptibility), повинна бути досить проста в установці (installability), повинна працювати одночасно з іншими програмами та не заважати їм (coexistence), а вони повинні не заважати їй. Хороша програма повинна надавати можливість користувачеві перейти з якогось іншого аналогічного програмного забезпечення на використання цієї програми - заміна іншого ПО даними (replaceability). Тобто надати якісь можливості з міграції, найчастіше з міграції даних.

Відповідно, виходячи з цієї класифікації, для кожного аспекту якості виділяють відповідний вид тестування:

- тестування функціональності;
- тестування надійності;
- тестування ефективності;
- тестування практичності;
- тестування супроводжуваємості;
- тестування переносимості.

Функціональні види (functional testing) тестування

Функціональні тести базуються на функціях та особливостях, а також взаємодії з іншими системами, та можуть бути представлені на всіх рівнях тестування: компонентному або модульному (component / unit testing), інтеграційному (integration testing), системному (system testing) та приймальному (acceptance testing). Функціональні види тестування розглядають зовнішню поведінку системи.

Види функціональних тестів:

- Функціональне тестування (functional testing);
- Тестування безпеки (security and access control testing);
- Тестування взаємодії (interoperability testing).

Нефункціональні види тестування

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, "Як" система працює. Основні види нефункціональних тестів:

1) Усі види тестування продуктивності:

- тестування навантаження (performance and load testing). Завданням тестування продуктивності є визначення масштабованості додатка під навантаженням, при цьому відбувається:

а) вимір часу виконання обраних операцій при певних інтенсивностях виконання цих операцій;

б) визначення кількості користувачів, що одночасно працюють з додатком;

в) визначення меж прийнятної продуктивності при збільшенні навантаження;

г) дослідження продуктивності на високих, граничних, стресових навантаженнях.

- стресове тестування (stress testing). Дозволяє перевірити наскільки додаток і система в цілому працездатні в умовах стресу й оцінити здатність системи до регенерації, тобто до повернення в нормальний стан після припинення впливу стресу.

- тестування стабільності або надійності (stability / reliability testing). Завданням тестування стабільності (надійності) є перевірка працездатності програми при тривалому (багатогодинному) тестуванні із середнім рівнем навантаження.

- об'ємне тестування (volume testing). Завданням об'ємного тестування є отримання оцінки продуктивності при збільшенні обсягів даних в базі даних програми, при цьому відбувається:

а) вимір часу виконання обраних операцій при певних интенсивностях виконання цих операцій;

б) може проводитися визначення кількості користувачів, що одночасно працюють з додатком.

2) Тестування установки (installation testing) направлено на перевірку успішної інсталяції та настройки, а також поновлення або видалення програмного забезпечення.

3) Тестування зручності користування (usability testing). Перевірка ергономічності - дослідження, яке виконується з метою визначення, зручний чи ні деякий штучний об'єкт (такий як веб-сторінка, призначений для користувача інтерфейс або пристрій) для його передбачуваного застосування. Таким чином, перевірка ергономічності вимірює ергономічність об'єкта або системи. Перевірка ергономічності зосереджена на певному об'єкті або невеликому наборі об'єктів, в той час як дослідження взаємодії людина-комп'ютер в цілому - формулюють універсальні принципи. Перевірка ергономічності - метод оцінки зручності продукту у використанні, заснований на залученні користувачів в якості тестувальників, випробувачів і підсумовуванних отриманих від них висновків.

4) Тестування на відмову та відновлення (failover and recovery testing) перевіряє тестований продукт з точки зору здатності протистояти й успішно відновлюватися після можливих збоїв, що виникли в зв'язку з помилками програмного забезпечення, відмовами обладнання або проблемами зв'язку (наприклад, відмова мережі). Метою даного виду тестування є перевірка систем відновлення, які, в разі виникнення збоїв, забезпечать збереження та цілісність даних тестованого продукту.

5) Конфігураційне тестування (configuration testing) - спеціальний вид тестування, спрямований на перевірку роботи програмного забезпечення при різних конфігураціях системи (заявлених платформах, підтримуваних драйверах, при різних конфігураціях комп'ютерів і т.і.).

Пов'язані зі змінами види тестування

Після проведення необхідних змін, таких як виправлення бага / дефекту, програмне забезпечення повинно бути перетестованне для підтвердження того факту, що проблема була дійсно вирішена. Види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності додатки або правильності здійсненого виправлення дефекту:

- Димове тестування (smoke testing) розглядається як короткий цикл тестів, що виконується для підтвердження того, що після складання коду (нового або відредагованого) додаток, що встановлюється, стартує та виконує основні функції.

Висновок про працездатність основних функцій робиться на підставі результатів поверхневого тестування найбільш важливих модулів програми на предмет можливості виконання необхідних завдань і наявності швидкознаходженних критичних і блокуючих дефектів. У разі відсутності таких дефектів димове тестування оголошується пройденим, і додаток передається для проведення повного циклу тестування, в іншому випадку, димове тестування оголошується проваленим, і додаток йде на доопрацювання.

- Регресійне тестування (regression testing). Це вид тестування спрямований на перевірку змін, зроблених в додатку або навколишньому середовищі (лагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб сервер або сервер додатків). Для підтвердження того факту, що існуюча раніше функціональність працює як і раніше. Регресійними можуть бути як функціональні, так і нефункціональні тести.

Як правило, для регресійного тестування використовуються тест кейси, написані на ранніх стадіях розробки та тестування. Це дає гарантію того, що зміні в новій версії програми не пошкодили вже існуючу функціональність. Рекомендується робити автоматизацію регресійних тестів,

для прискорення подальшого процесу тестування та виявлення дефектів на ранніх стадіях розробки програмного забезпечення.

Сем Канер, описує три основних типи регресивного тестування:

1) Регресія багів (bug regression) - спроба довести, що виправлена помилка насправді не виправлена.

2) Регресія старих багів (old bugs regression) - спроба довести, що нещодавня зміна коду або даних зламала виправлення старих помилок, тобто старі баги стали знову відтворюватися.

3) Регресія побічного ефекту (side effect regression) - спроба довести, що нещодавня зміна коду або даних зламала інші частини розробляємого додатка.

- Тестування збірки (build verification test). Це тестування спрямоване на визначення відповідності, випущеної версії, критерієм якості для початку тестування. За своїми цілями є аналогом димового тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію. Вглиб воно може проникати далі, в залежності від вимог до якості випущеної версії.

- Санітарне тестування або перевірка узгодженості / справності (sanity testing). Санітарне тестування - це вузьконаправлене тестування достатнє для доказу того, що конкретна функція працює згідно із заявленими в специфікації вимогам. Є підмножиною регресійного тестування. Використовується для визначення працездатності певної частини програми після змін вироблених в ній або в навколишньому середовищу. Зазвичай виконується вручну.

На відміну від димового, санітарне тестування направлено вглиб перевіряємой функції, в той час як димове направлено вшир, для покриття тестами якомога більшого функціоналу в найкоротші терміни.

Інщі підвиди тестування

- Тестування локалізації (localization testing) - це процес тестування локалізованої версії програмного продукту або сайту. Локалізація

програмного забезпечення - процес адаптації до культурних особливостей тієї чи іншої країни: переклад документації, елементів призначеного для користувача інтерфейсу, допоміжних матеріалів з однієї мови на іншу.

- Системне тестування (system testing). Основним завданням системного тестування є перевірка як функціональних, так і не функціональних вимог в системі в цілому. При цьому виявляються дефекти, такі як неправильне використання ресурсів системи, непередбачені комбінації даних рівня користувача, несумісність з оточенням, непередбачені сценарії використання, відсутня або неправильна функціональність, незручність використання. Для мінімізації ризиків, пов'язаних з особливостями поведінки системи в тому чи іншому середовищі, під час тестування рекомендується використовувати оточення максимально наближене до того, на яке буде встановлено продукт після видачі.

- Компонентне або модульне тестування (component or unit testing) перевіряє функціональність і шукає дефекти в частинах додатка, які доступні та можуть бути протестовані по-окремо (модулі програм, об'єкти, класи, функції). Зазвичай компонентне (модульне) тестування проводиться викликаючи код, який необхідно перевірити та при підтримці середовищ розробки, таких як фреймворки (frameworks - каркаси) для модульного тестування або інструменти для налагодження. Всі знайдені дефекти, як правило виправляються в коді без формального їх опису в системі менеджменту багів (bug tracking system).

Один з найбільш ефективних підходів до компонентного (модульному) тестування - це підготовка автоматизованих тестів до початку основного кодування (розробки) програмного забезпечення. Це називається розробка від тестування (test-driven development) або підхід тестування спочатку (test first approach). При цьому підході створюються й інтегруються невеликі шматки коду, навпроти яких запускаються тести, написані до початку кодування. Розробка ведеться до тих пір, поки всі тести не будуть успішно пройдені.

Різниця між компонентним і модульним тестуванням лише в тому, що в компонентному тестуванні в якості параметрів функцій використовують реальні об'єкти та драйвери, а в модульному тестуванні - конкретні значення.

- Тестування безпеки (security and access control testing) - це стратегія тестування, яка використовується для перевірки безпеки системи, а також для аналізу ризиків, пов'язаних із забезпеченням цілісного підходу до захисту додатків, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних.

Загальна стратегія безпеки грунтується на трьох основних принципах:

а) Конфіденційність - це приховування певних ресурсів або інформації. Під конфіденційністю можна розуміти обмеження доступу до ресурсу деякої категорії користувачів, або іншими словами, за яких умов авторизований користувач отримає доступ до цього ресурсу.

б) Цілісність. Існує два основних критерії при визначенні поняття цілісності:

- Довіра. Очікується, що ресурс буде змінений тільки відповідним способом певною групою користувачів.

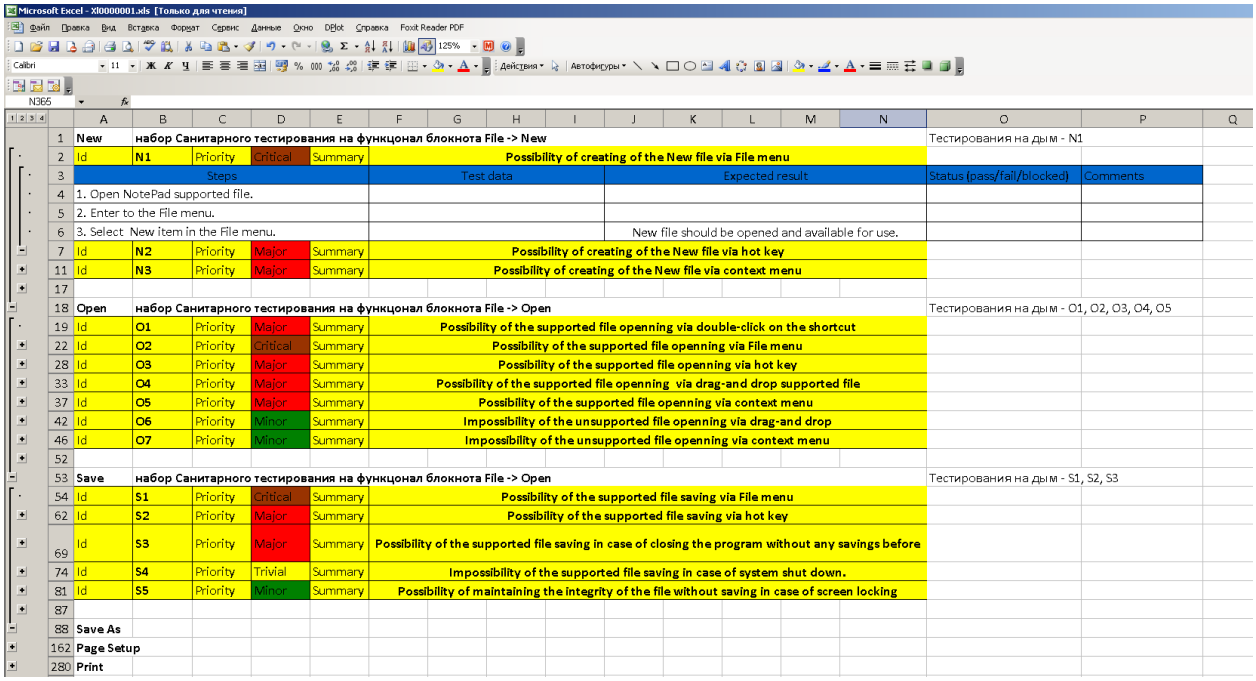
- Пошкодження та відновлення. У разі коли дані пошкоджуються або неправильно змінюються авторизованим або не авторизованим користувачем, треба визначити на скільки важливою є процедура відновлення даних.

в) Доступність є вимога про те, що ресурси повинні бути доступні авторизованому користувачеві, внутрішньому об'єкту або пристрою. Як правило, чим більш критичний ресурс тим вище рівень доступності повинен бути.

Завдання

1) Протестувати додаток віртуального банкомату https://mobiasbanca.md/ru/simulator (пинкод 1234), використовуючи види функціонального та нефункціонального тестування.

2) Додаток згрупувати по видам тестування, прив'язанного к змінам - димовє, санітарнє, регресіяя. Варіанти завдання наведені в таблиці 2.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	New	набор Санитарного тестирования на функцонал блокнота File -> New													Тестирования на дым - N1		
2	Id	N1	Priority	Critical	Summary	Possibility of creating of the New file via File menu											
3	Steps					Test data				Expected result					Status (pass/fail/blocked)	Comments	
4	1. Open NotePad supported file.																
5	2. Enter to the File menu.																
6	3. Select New item in the File menu.									New file should be opened and available for use.							
7	Id	N2	Priority	Major	Summary	Possibility of creating of the New file via hot key											
11	Id	N3	Priority	Major	Summary	Possibility of creating of the New file via context menu											
17																	
18	Open	набор Санитарного тестирования на функцонал блокнота File -> Open													Тестирования на дым - O1, O2, O3, O4, O5		
19	Id	O1	Priority	Major	Summary	Possibility of the supported file openning via double-click on the shortcut											
22	Id	O2	Priority	Critical	Summary	Possibility of the supported file openning via File menu											
28	Id	O3	Priority	Major	Summary	Possibility of the supported file openning via hot key											
33	Id	O4	Priority	Major	Summary	Possibility of the supported file openning via drag-and drop supported file											
37	Id	O5	Priority	Major	Summary	Possibility of the supported file openning via context menu											
42	Id	O6	Priority	Minor	Summary	Impossibility of the unsupported file openning via drag-and drop											
46	Id	O7	Priority	Minor	Summary	Impossibility of the unsupported file openning via context menu											
52																	
53	Save	набор Санитарного тестирования на функцонал блокнота File -> Open													Тестирования на дым - S1, S2, S3		
54	Id	S1	Priority	Critical	Summary	Possibility of the supported file saving via File menu											
62	Id	S2	Priority	Major	Summary	Possibility of the supported file saving via hot key											
69	Id	S3	Priority	Major	Summary	Possibility of the supported file saving in case of closing the program without any savings before											
74	Id	S4	Priority	Trivial	Summary	Impossibility of the supported file saving in case of system shut down.											
81	Id	S5	Priority	Minor	Summary	Possibility of maintaining the integrity of the file without saving in case of screen locking											
87																	
88	Save As																
162	Page Setup																
280	Print																

Рис. 7.1 – Приклад тестувння «Блокнота»

Таблиця 2. Варіанти завдання

№ варианту	**Додатки Windows**
1	Microsoft Paint
2	Internet Explorer
3	Microsoft Messaging
4	WordPad
5	Дефрагментация диска
6	Диспетчер задач
7	Диспетчер программ
8	Калькулятор
9	Проводник
10	Outlook Mail
11	Windows Movie Maker
12	Диспечер файлов
13	Игры (пасьянс)
14	SmartDrive
15	Фотоальбом Windows
16	Сервис проверки подлинности локальной системы безопасности
17	Boot.ini
18	NetMeeting
19	Сапёр (игра)
20	Блокнот

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Буч Г., Рамбо Дж. и др. Язык UML. Руководство пользователя: Пер. с англ. – М: ДМК, 2000. – 432 с., ил.
2. Иванов Д. Ю. , Новиков Ф. А., Моделирование на UML. Теория, практика, видеокурс: Наука и Техника, 2010 г., 640 с.
3. Калянов Г.Н. CASE. Структурный системный анализ М.: “Лори”, 1996. - 241 с. ил.
4. Орлов С.А. Технологии разработки программного обеспечения: Учебник/ СПб: Питер, 2002. - 464 с. ил.
5. Хассан Гома, UML Проектирование систем реального времени, распределенных и реальных приложений (Designing Concurrent, Distributed, and Real-Time Applications with UML): ДМК Пресс, 2011. -704с.
6. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. - СПб: Питер, 2002. 496 с. ил.
7. Miro Samek, Practical UML Statecharts in C/C++, Second Edition: Event-Driven Programming for Embedded Systems, Published May 1st 2008 by Focal Press, 728 pages
8. https://praveenthomasln.wordpress.com/2012/02/25/class-diagrams/
9. http://moodle.ipo.kpi.ua/moodle/mod/resource/index.php?id=513
10. http://stackoverflow.com/questions/2167688/uml-relation-between-usecases-extend-include
11. http://posibniki.com.ua/catalog-avtomatizovane-proektuvannya-informaciynih-sistem---denisova-o-o
12. http://vunivere.ru/work23987/page4
13. http://www.intuit.ru/studies/courses/32/32/lecture/1026
14. http://www.javatpoint.com/facade-pattern
15. http://sites.znu.edu.ua/webprog/lect/1238.ukr.html
16. http://www.e-reading.club/book.php?book=33640
17. http://edn.embarcadero.com/article/31863

18. http://staruml.io/
19. http://www.agilemodeling.com/essays/umlDiagrams.htm
20. https://www.visual-paradigm.com/features/uml-and-sysml-modeling/
21. http://www.uk.w3eacademy.com/uml/uml_interaction_diagram.htm
22. http://www.studfiles.ru/preview/5249267/page:8/
23. https://www.youtube.com/watch?v=YO0WYZtRY8Q
24. http://www.protesting.ru/testing/

ДОДАТОК.

Приклад оформлення лабораторних робіт

					ЛР 01-07.121.5157.01		
Змін.	Арк.	№ докум.	Підпись	Дата			
Розробив		Ритчи Д.М			Лабораторні роботи з дисципліни “ CASE-засоби розробки програмного забезпечення”	Літ.	Арк. 1 / Аркушів
Перевірив		Карпова С.О.					
							ХФ НУК

Лабораторна робота №1

Тема: Робота у середовищі CASE - засобу Visual Paradigm for UML.
Розробка діаграм Use Case у середовищі
CASE - засобу Visual Paradigm for UML.

Мета роботи: Ознайомлення з основними компонентами мови UML. Вивчення об'єктно-орієнтованого моделювання та дослідження процесу побудови при створення моделі діаграми Use Case в середовищі Visual Paradigm for UML.

Короткі теоретичні відомості

Мова UML - це загальноприйнята мову візуального моделювання, яка розроблена для специфікації, візуалізації, проектування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Мова UML одночасно є простим і потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних і графічних моделей складних систем самого різного спрямування та призначення.

Конструктивне використання мови UML ґрунтується на розумінні загальних принципів моделювання складних систем та особливостей процесу об'єктно-орієнтованого проектування (ООП) зокрема. Вибір виразних засобів для побудови моделей складних систем зумовлює ті завдання, які можуть бути вирішені з використанням даних моделей. При цьому, одним з основних принципів побудови моделей складних систем, є принцип абстрагування, який наказує включати в модель тільки ті аспекти проектованої системи, які мають безпосереднє відношення до виконання системою своїх функцій або свого цільового призначення. В цьому випадку, всі другорядні деталі

опускаються, щоб надмірно не ускладнювати процес аналізу та дослідження отриманої моделі.

Словник UML утворює три різновиди будівельних блоків: предмети, відносини, діаграми.

Предмети - це абстракції які є основними елементами в моделі, відносини пов'язують ці предмети, діаграми групують колекції предметів.

В UML є чотири різновиди предметів:

- стуктурні предмети;
- предмети поведінки;
- згруповуючі предмети;
- пояснюючі предмети.

В UML є чотири різновиди відношення:

- асоціація;
- узагальнення;
- залежність;
- реалізація.

Діаграма в UML - це графічне представлення набору елементів, що зображується найчастіше у вигляді пов'язаного графа з вершинами (сутностями) та ребрами (відносинами).

В UML виділяють такі типи діаграм:

- діаграми варіантів використання (use case diagrams);
- діаграми класів (class diagrams);
- діаграми поведінки системи (behavior diagrams);
- діаграми взаємодії (interaction diagrams);
- діаграми послідовності (sequence diagrams);
- діаграми кооперації (collaboration diagrams);
- діаграми станів (statechart diagrams);
- діаграми діяльностей (activity diagrams);
- діаграми реалізації (implementation diagrams);

- діаграми компонентів (component diagrams);
- діаграми розгортання (deployment diagrams).

Діаграма варіантів використання в UML на якій зображено відношення між акторами та варіантами використання в системі.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (англ. use case) застосовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

Елементи діаграми Use Case є варіант використання (Use Case), актор (actor), інтерфейс (interface), примітки (notes), відношення (relationship).

Відношення бувають:

- відношення асоціації(association relationship);
- відношення розширення (extend relationship);
- відношення узагальнення (generalization relationship);
- відношення включення (include relationship).

Завдання: Побудувати за допомогою CASE-засобу Visual Paradigm for UML діаграму Use Case процесу взаємозв'язку агентства нерухомості й його клієнтів.

Опис процесу. За допомогою CASE-засобу Visual Paradigm будуємо діаграму Use Case (рис. 1.1-1.2) процесу взаємозв'язку агентства нерухомості та клієнта (рис. 1.3). В якості акторів даної системи виступають: агентство та клієнти. Система роботи агентства демонструє, що клієнт, звернувшись із запитом або пропозицією до агентства, може отримати повну необхідну йому інформацію про ринок нерухомості, узгодити терміни й умови послуг. В результаті чого оформити замовлення на купівлю, продаж або обмін

нерухомості. Агентство може брати участь в оформленні декількох замовлень.

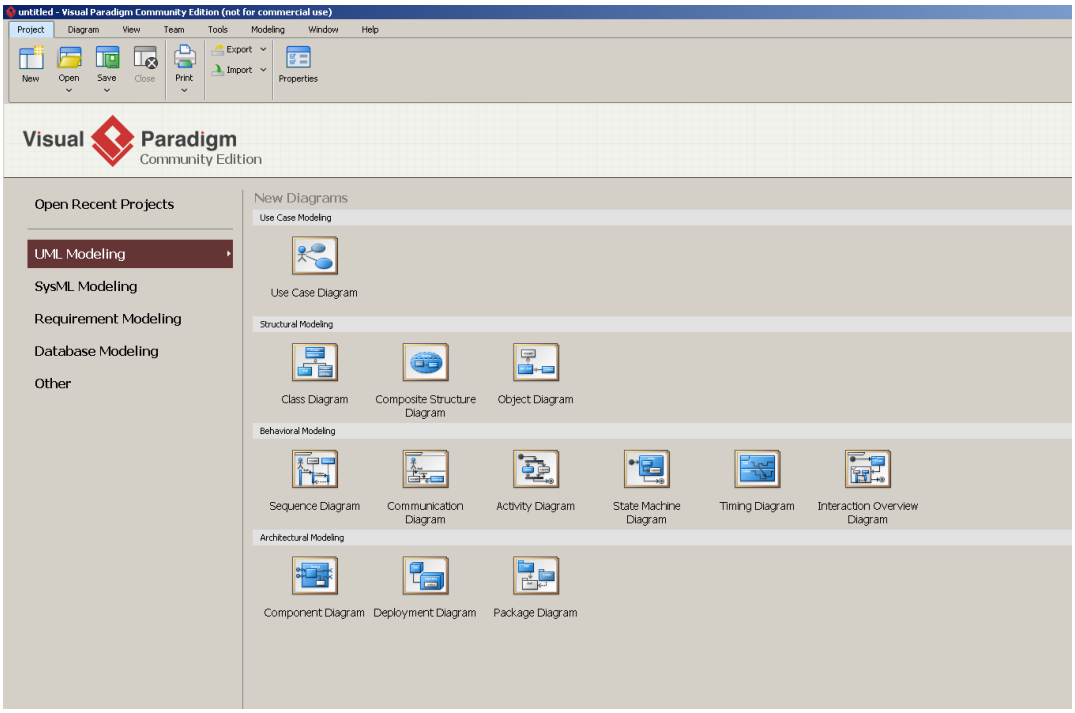


Рисунок 1.1 – Вікно Visual Paradigm. Вибір діаграм

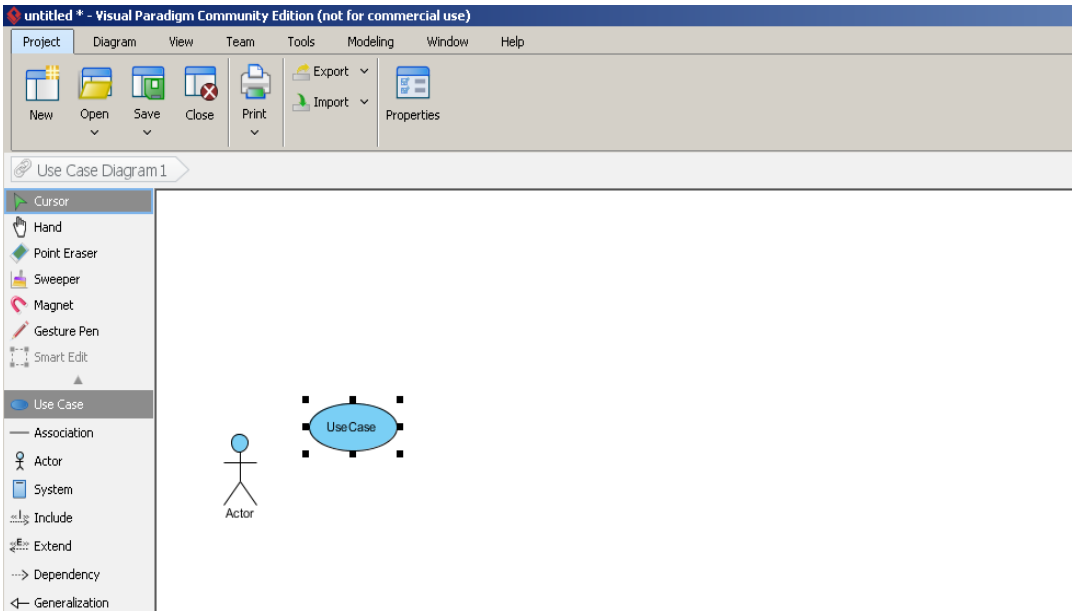


Рисунок 1.2 – Вікно з елементами діаграми варіантів використання (use case diagrams)

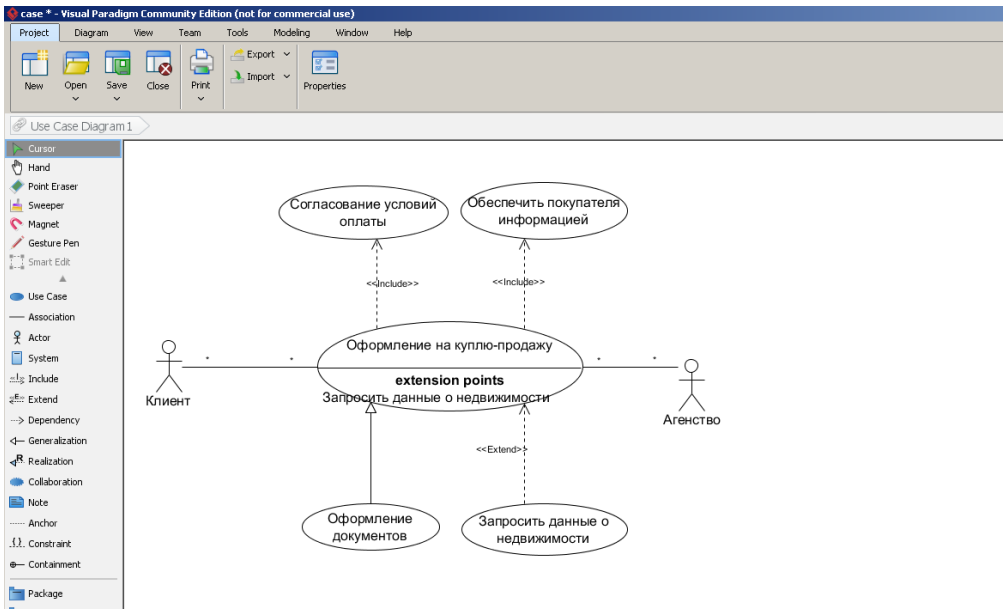


Рисунок 1.3 – Фрагмент діаграми Use Case процесу взаємозв'язку агентства з купівлі-продажи нерухомості та клієнтів

Висновок

Діаграми варіантів використання застосовуються для моделювання виду системи з точки зору варіантів використання. Найчастіше це передбачає моделювання контексту системи, підсистеми або класу чи моделювання вимог, що висуваються до поведінки зазначених елементів.

Ці діаграми мають велике значення для візуалізації, специфікації та документування поведінки елемента. Вони полегшують розуміння систем, підсистем або класів, представляючи погляд ззовні на те, як дані-елементи можуть бути використані у відповідному контексті. Крім того, такі діаграми важливі для тестування виконуваних систем в процесі прямого проектування та для розуміння їх внутрішнього устрою при зворотному проектуванні.

Лабораторна робота №2

Тема: Розробка діаграм взаємодії у середовищі
CASE - засобу Visual Paradigm for UML

Мета роботи: Вивчення об'єктно-орієнтованого моделювання та дослідження процесу побудови та створення моделей діаграм послідовності та кооперації в середовищі Visual UML за допомогою її елементів.

Короткі теоретичні відомості

Діаграма взаємодії - одна з моделей опису поведінки взаємодіючих груп об'єктів в UML. Як правило, кожна окрема діаграма взаємодії описує поведінку тільки в межах одного варіанта використання. На такій діаграмі прийнято відображати екземпляри об'єктів і повідомлення, якими ці об'єкти обмінюються один з одним в рамках даного варіанта використання.

Діаграма послідовності - відображає взаємодії об'єктів впорядкованих за часом. На діаграмі кооперації зображуються тільки відносини між об'єктами, що грають певні ролі у взаємодії. З іншого боку, на цій діаграмі не вказується час у вигляді окремого вимірювання.

Завдання: Побудувати за допомогою Visual Paradigm for UML діаграму послідовності та кооперації процесу взаємозв'язку агентства нерухомості й його клієнтів.

Опис процесу. За допомогою Visual Paradigm побудуємо діаграми послідовності (рис. 2.1) та кооперації (рис. 2.2) процесу взаємозв'язку агентства нерухомості та клієнтів, для моделювання процесу обміну квартири одного клієнта на будинок іншого клієнта.

Об'єктами в діаграмі послідовності (рис. 2.1) є два клієнта: кліент1 і кліент2, два види нерухомості: квартира та будинок, агентство та договір обміну житла як об'єкт моделювання.

Клієнтів розглядаємо як акторів, причому перший з них - кліент1 - грає активну роль, а другий - кліент2 - пасивну роль. Тому перший отримує фокус управління відразу після своєї появи в системі. Агентство має постійну активність.

Договір обміну житла в якості об'єкту з'являється лише після погодження обома сторонами та знищується після його завершення.

На рисунку 2.2 зображена діаграма кооперації для моделювання процесу обміну нерухомості. Така діаграма створюється вже на більш пізній стадії розвитку проекту, тому на ній показані взаємодії внутрішніх об'єктів.

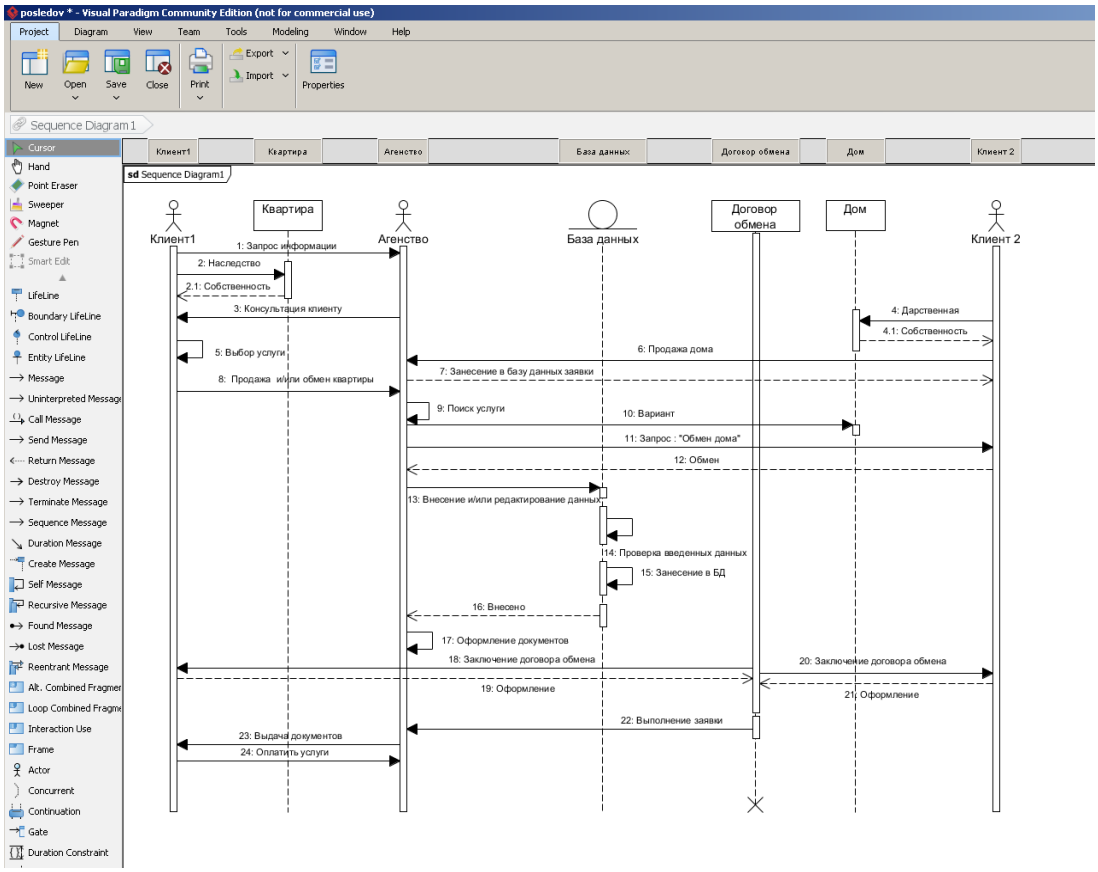


Рисунок 2.1 - Діаграма послідовності процесу взаємозв'язку агентства нерухомості та клієнтів

Для того, щоб створити діаграму кооперації потрібно натиснути праву клавішу миші та вибрати “Synchronize to Communication Diagram” в спливаючим меню.

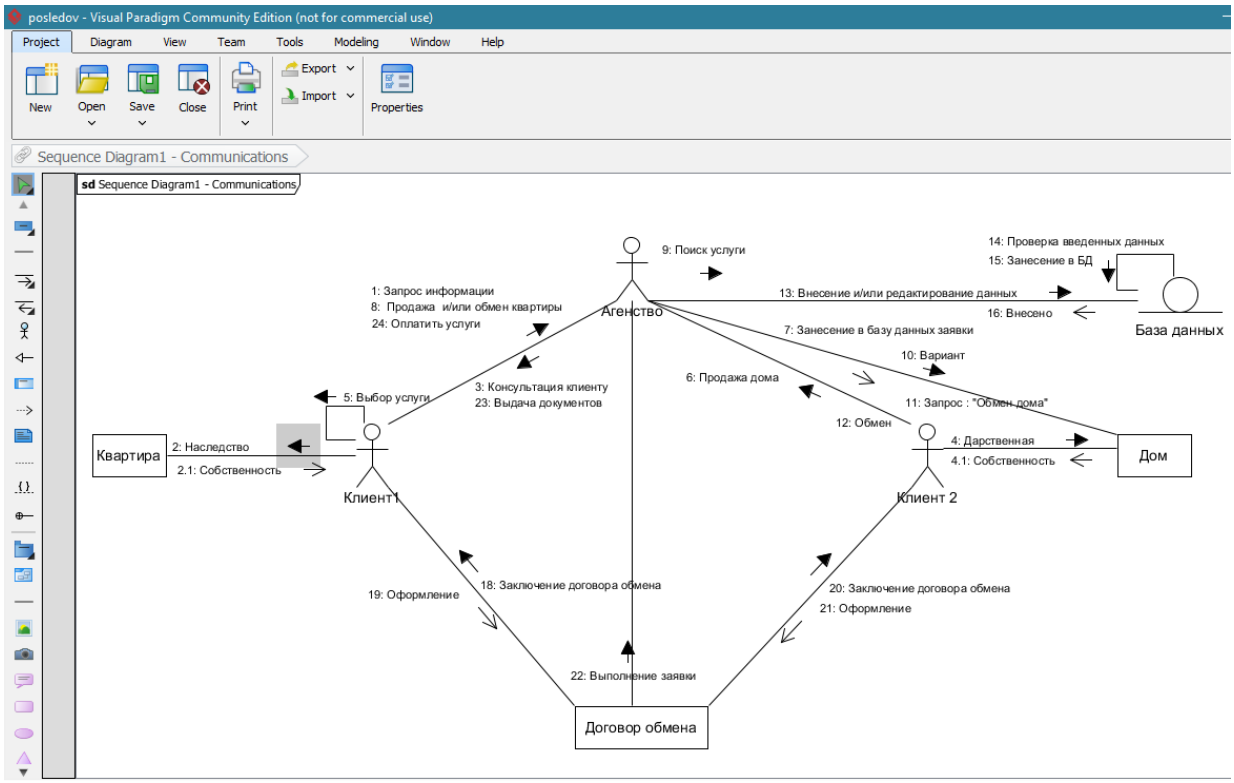


Рисунок 2.2 – Діаграма кооперації процесу взаємозв'язку агентства нерухомості та клієнтів

Висновок

Діаграми послідовностей та кооперації відносяться до числа п'яти видів діаграм, що застосовуються в UML для моделювання динамічних аспектів системи. На діаграмах взаємодії показується зв'язок, що включає безліч об'єктів і відносин між ними, в тому числі повідомлення, якими об'єкти обмінюються. При цьому діаграма послідовностей акцентує увагу на тимчасовій впорядкованості повідомлень, а діаграма кооперації - на структурній організації посилань і приймання повідомлення об'єктів.

Діаграми взаємодії використовуються для моделювання динамічних аспектів системи. Сюди входить моделювання конкретних і прототипічних примірників класів, інтерфейсів, компонентів і вузлів, а також повідомлень, якими вони обмінюються, - та все це в контексті сценарію, що ілюструє дану поведінку. Діаграми взаємодії можуть служити для візуалізації, специфіцирування, конструювання та документування динаміки конкретної спільноти об'єктів, а також використовуватися для моделювання окремого потоку управління в складі варіанта використання.

Діаграми взаємодії важливі не тільки для моделювання динамічних аспектів системи, але й для створення виконуваних систем за допомогою прямого та зворотного проектування.

Лабораторна робота №3

Тема: Розробка діаграм класів у середовищі CASE-засобу Visual Paradigm for UML

Мета роботи: Вивчення об'єктно-орієнтованого моделювання та дослідження процесу побудови та створення моделі діаграми класів в середовищі Visual Paradigm for UML за допомогою її елементів.

Короткі теоретичні відомості

Діаграма класів (class diagram) - включає класи, типи даних, їх зміст і відношення. Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти та кооперації, а також їхні відносини.

Завдання: Побудувати за допомогою Visual Paradigm for UML діаграму класів процесу взаємозв'язку агентства з купівлі-продажу нерухомості та клієнтів цього агентства.

Опис процесу. За допомогою Visual Paradigm for UML побудували діаграму класів процесу взаємозв'язку агентства нерухомості й його клієнтів (рис. 3.2), яка містить два класи Клієнт та Агентство.

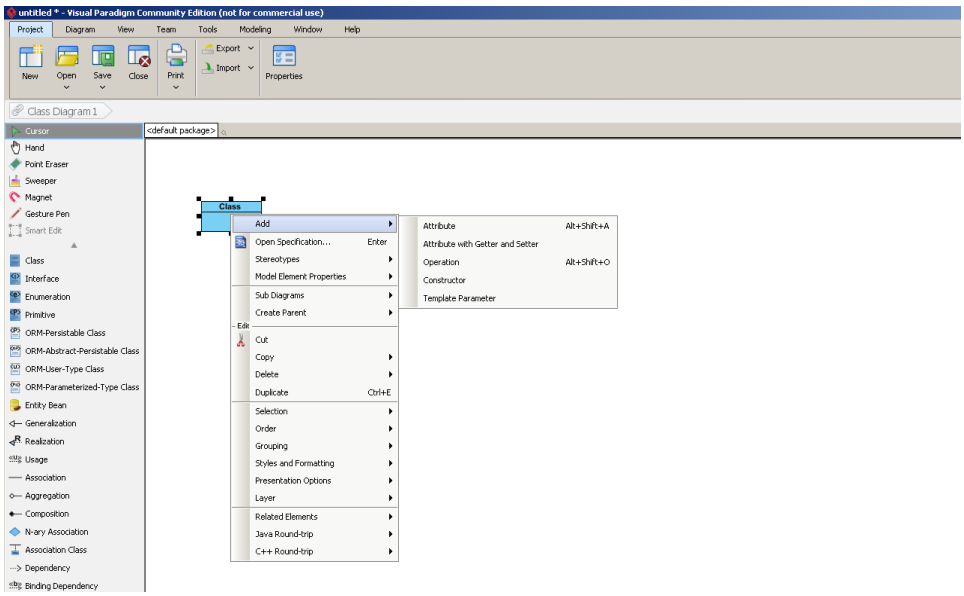


Рисунок 3.1 – Побудова діаграми класів.

Створеня класу, додавання його атрибутів та операцій

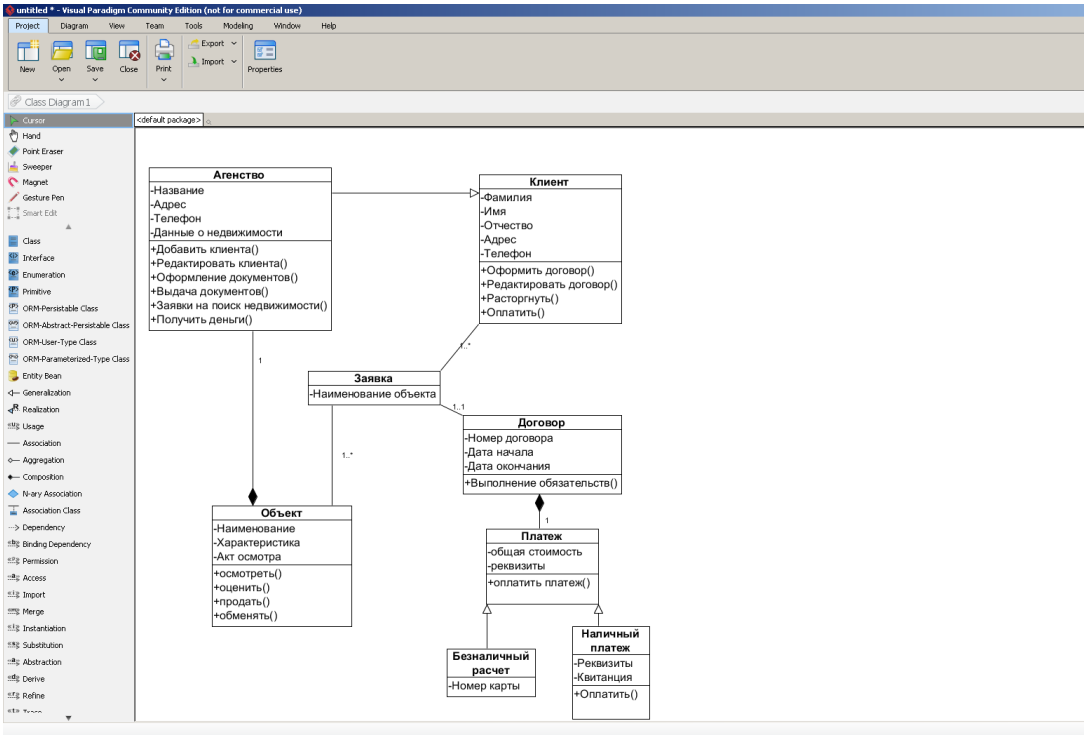


Рисунок 3.2 – Діаграма класів процесу взаємозв'язку агентства нерухомості та клієнтів

Висновок

Діаграми класів при моделюванні об'єктно-орієнтованих систем зустрічаються частіше інших. На таких діаграмах показується безліч класів, інтерфейсів, кооперацій та відносин між ними.

Діаграма класів використовується для моделювання статичного виду системи з точки зору проектування. Сюди здебільшого відноситься моделювання словника системи, кооперацій та схем. Крім того, діаграми класів складають основу ще двох діаграм - компонентів і розгортання.

Діаграми класів важливі не тільки для візуалізації, специфікування та документування структурних моделей, але також для прямого та зворотного проектування виконуваних систем.

Лабораторна робота №4

Тема: Розробка діаграм станів і діяльності у середовищі CASE-засобу Visual Paradigm for UML

Мета роботи: Вивчення об'єктно-орієнтованого моделювання та дослідження процесу побудови та створення моделі діаграми станів і діяльності в середовищі Visual Paradigm for UML за допомогою її елементів.

Короткі теоретичні відомості

Діаграма станів - спрямований граф, вершинам якого відповідають стани автомата, а дугам - вхідні сигнали.

Елементами діаграми станів є:

- автомат (state machine);
- стан (state);
- перехід (transition);
- складений стан (composite state) і підстан (substate);
- історичний стан (history state);
- складні переходи.

Діаграма діяльності в UML - це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.

Елементами діаграми діяльності є:

- стан дії;
- переходи;
- об'єкти.

Завдання: Побудувати за допомогою Visual Paradigm for UML діаграму станів і діяльності, яка представляє систему процесу укладення контракту агентства нерухомості з клієнтами.

Опис процесу. За допомогою Visual Paradigm for UML в середовищі CASE побудовані діаграми станів (рис. 4.1) і діяльності (рис. 4.2), які представляють систему процесу укладення контракту агентства купівлі-продажи нерухомості з клієнтами.

Початковий стан системи (чорна крапка на рисунку 4.1) визначає початковий процес зміни станів, який переходить в стан «Регистрация клиента». Даний стан виконує функцію вітання, що пропонує відвідувачам стати клієнтом фірми, для переходу стану в наступний стан «Заполнение анкеты» необхідно передати події: особисті дані та список параметрів (прізвище, ім'я, по-батькові, номер телефону, інше). У цьому стані дані заносяться, редагуються та видаляються. Наступний етап: узгодження умов, дозволяє ознайомитися з умовами надання послуг, їх вартості та ціною; переводить в стан «Заключение договора». Після позитивної відповіді обох сторін переходимо в стан «Регистрация нотариусом». У цьому стані повторюються подібні події реквізити зі списком параметрів: паспортні дані, податковий код та інше. Після цього переходимо до стану «Выписка». Після того, як спрацьовує сторожова умова контракта, робота кінцевого автомата закінчується та переходить в кінцевий стан (чорна крапка в обрамленні на рисунку 4.1).

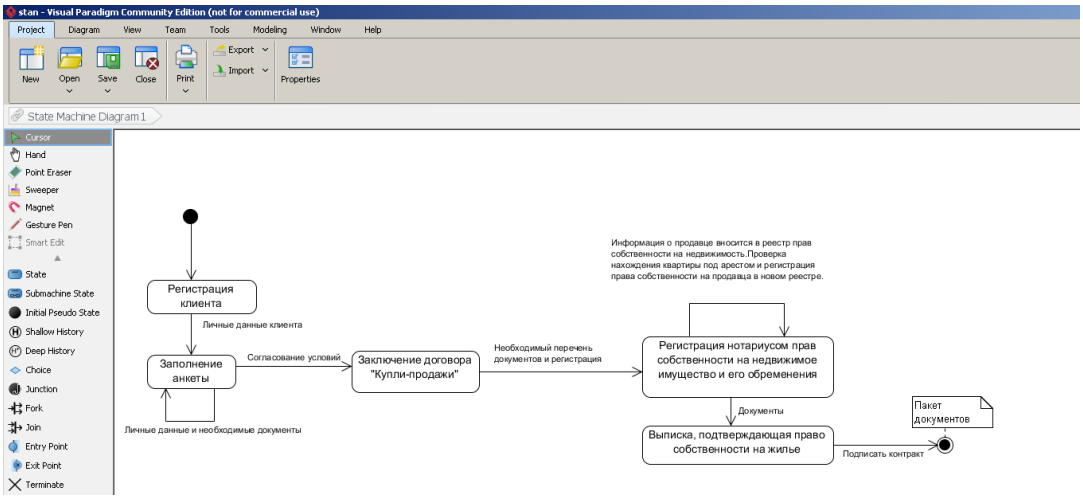


Рисунок 4.1 – Діаграма станів процесу укладення контракту агентства з купівлі-продажу нерухомості з її клієнтами

Діаграма діяльності містить стани дії та розгалуження; має єдиний початковий й єдиний кінцевий стан. Стрілками на діаграмі показані послідовні переходи. Жирною рискою відзначені місця з'єднання потоків управління. Наявність паралельності проявляється в тому, що є можливість зайнятися пошуками нотаріуса під час заповнення анкети клієнта, а також ознайомитися з умовами надання послуг. Після позитивної відповіді обох сторін переходимо в стан «Заполнить поля договора», що дозволить перейти в наступний стан дії «Регистрация нотариусом». У цьому стані перевіряються реквізити зі списком параметрів: паспортні дані, податковий код та інше. Після закінчення цієї роботи ініціюється дія «Оформить договор». Якщо умови виконуються, договір після сплати усіх послуг передається клієнту.

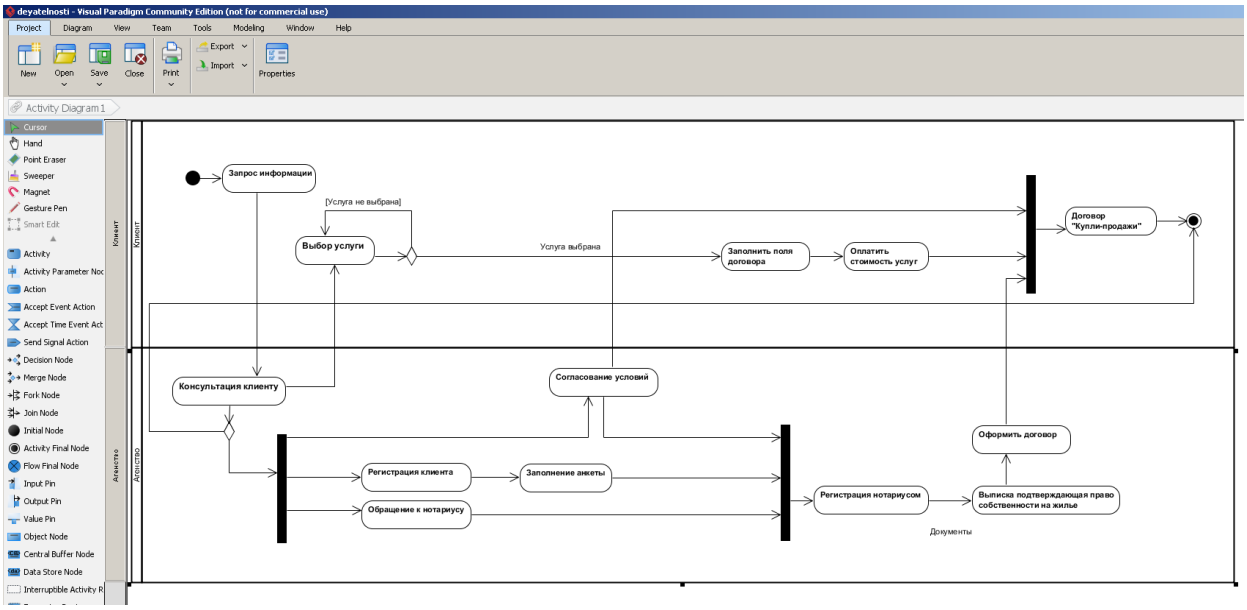


Рисунок 4.4 – Діаграма діяльності процесу взаємозв'язку агентства з купівлі-продажу нерухомості з її клієнтами

Висновок

Діаграми станів - це один з п'яти видів діаграм в мові UML, які використовуються для моделювання динамічних аспектів системи.

Діаграми станів використовуються для моделювання динамічних аспектів системи. Здебільшого під цим мається на увазі моделювання поведінки реактивних об'єктів. Реактивним називається об'єкт, поведінка якого найкраще характеризується його реакцією на події, що відбулися поза його власним контекстом. У реактивного об'єкта є чітко виражений життєвий цикл, коли поточна поведінка обумовлено минулим. Діаграми станів можна приєднувати до класів, прецедентів або системи в цілому для візуалізації, специфіцирування, конструювання та документування динаміки окремого об'єкта. Діаграми станів важливі не тільки для моделювання динамічних аспектів системи, але для конструювання виконуваних систем шляхом прямого та зворотного проектування.

Діаграма діяльності - це, по суті, блок-схема, яка показує, як потік управління переходить від однієї діяльності до іншої. Діаграми діяльності можна використовувати для моделювання динамічних аспектів поведінки системи. Як правило, вони застосовуються, щоб промоделювати послідовні (а іноді паралельні) кроки обчислювального процесу. За допомогою діаграм діяльності можна також моделювати життя об'єкта, коли він переходить з одного стану в інший в різних точках потоку управління. Діаграми діяльності можуть використовуватися самостійно для візуалізації, специфицирування, конструювання та документування динаміки сукупності об'єктів, але вони придатні також і для моделювання потоку управління при виконанні деякої операції. Якщо в діаграмах взаємодій акцент робиться на переходах потоку управління від об'єкта до об'єкта, то діаграми діяльності описують переходи від однієї діяльності до іншої. Діяльність (activity) - це деякий відносно тривалий етап виконання в автоматі. Діаграми діяльності важливі не тільки для моделювання динамічних аспектів поведінки системи, а також для побудови виконуваних систем за допомогою прямого та зворотного проектування.

Лабораторна робота №5

Тема: Розробка діаграм компонентів і розгортання у середовищі CASE - засобу Visual Paradigm for UML

Мета роботи: Вивчення об'єктно-орієнтованого моделювання, дослідження процесу побудови та створення моделей діаграм компонентів і розгортання в середовищі Visual Paradigm for UML за допомогою їх елементів.

Короткі теоретичні відомості

Діаграма компонентів - це діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Основними графічними елементами діаграми компонентів є: компонент (component), інтерфейс (interface), залежності.

Діаграма компонентів відображає лише структурні характеристики, для відображення окремих екземплярів компонентів слід використовувати діаграму розгортання.

Елементами діаграми розгортання є: вузол (node), з'єднання.

Завдання: Побудувати за допомогою Visual Paradigm for UML діаграму компонентів і розгортання яка представляє програму «Агентство_недвижимости» .

Опис процесу. Для представлення програми «Агентство_недвижимости» будуємо діаграму компонентів і розгортання (рис.5.1, рис. 5.2). На рисунку 5.1 зображена діаграма програми

«Агентство_недвижимости», що включає шість компонентів. Основним компонентом є виконання *Agenstvo_Nedvigimosti.exe*, що представляє виконуваний модуль. Робочим продуктом є компонент *program_AN.cpp*, файл з вихідним текстом програми. Компоненти розгортання *help.chc*, *Agenstvo_Nedvigimosti.html*, *AN.asp* і *db_Server.dbp*, спільні бібліотеки, які забезпечують безпосереднє виконання системою своїх функцій. Пунктирними стрілками зображено залежність між компонентами, яка вказує, як зміна одного компонента системи може вплинути на зміну іншого компонента.

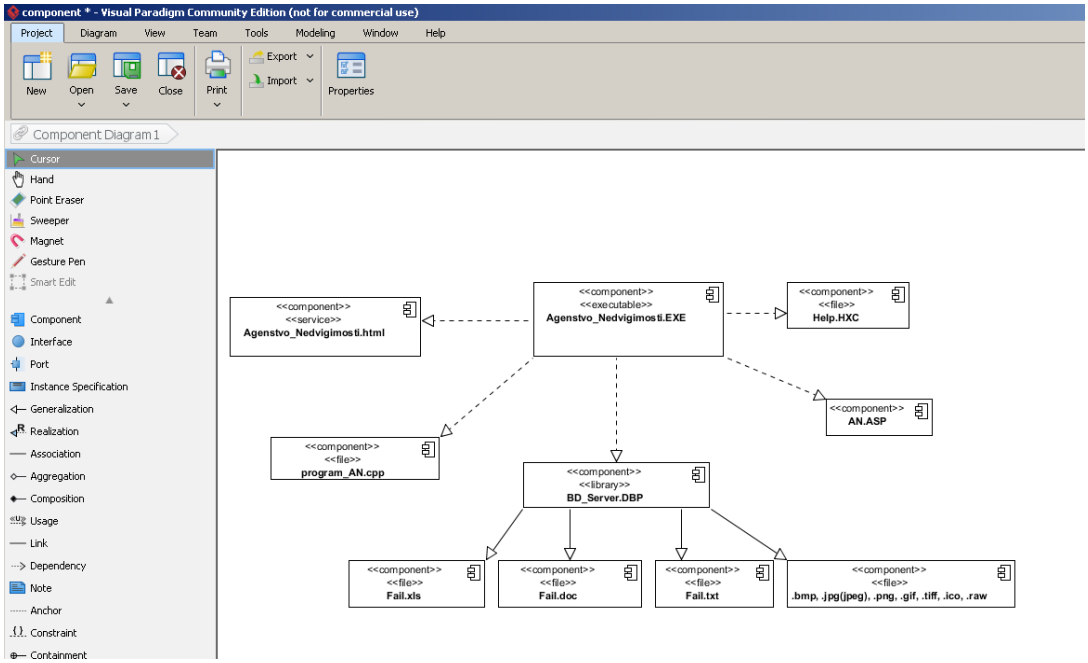


Рисунок 5.1 – Діаграма компонентів процесу «Програма агенства нерухомості»

На рисунку 5.2 показана описова діаграма розгортання «Програма агенства нерухомості». Діаграма розгортання містить графічне зображення процесорів і зв'язків між усіма вузлами реалізованої системи. Вузол зображується у формі тривимірного куба. Мережа виступає в якості типів моделі системи, а Сервер і Клієнт у вигляді примірників.

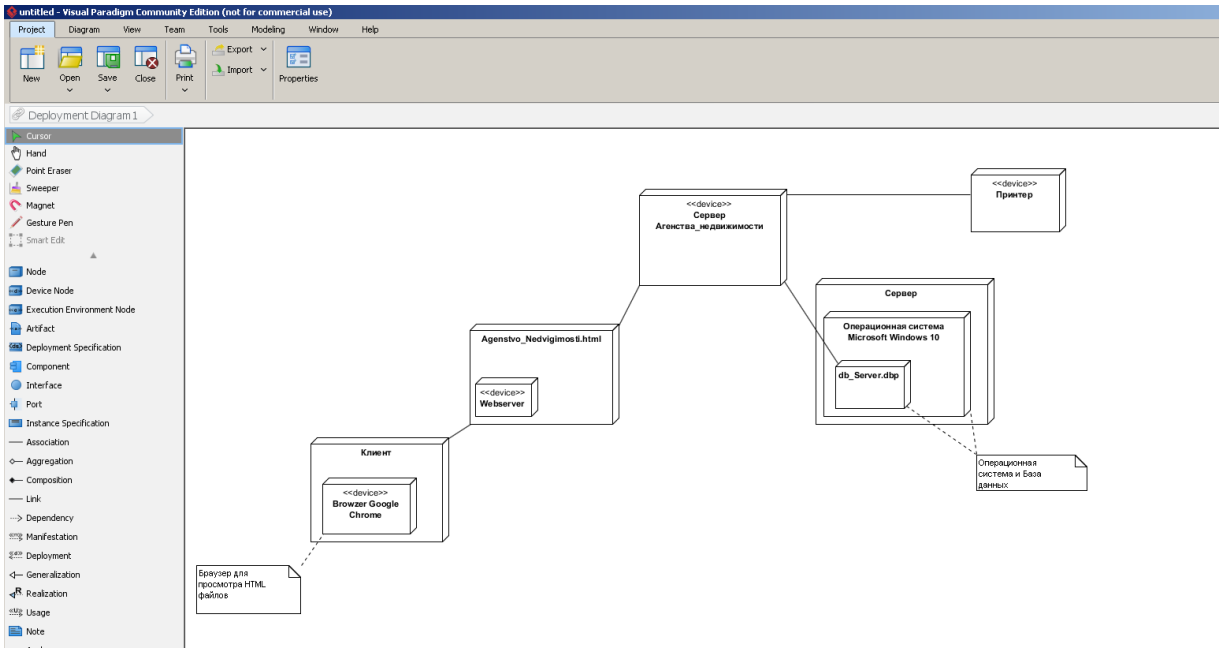


Рисунок 5.2 – Діаграма розгортання процесу
«Агенство нерухомості й його клієнти»

Висновок

Діаграми реалізації - це діаграми, що застосовуються при моделюванні фізичних аспектів об'єктно-орієнтованої системи. Діаграми компонентів застосовуються для моделювання статичного виду системи з точки зору реалізації. Так само як моделювання фізичних сутностей, розгорнутих в вузлі, наприклад виконуваних програм, бібліотек, таблиць, файлів і документів. По суті, діаграми компонентів - це не що інше, як діаграми класів, сфокусовані на системних компонентах. Діаграми розгортання, або застосування показують конфігурацію вузлів, де проводиться обробка інформації, й які компоненти розміщені на кожному вузлі. Діаграми розгортання використовуються для моделювання статичного виду системи з точки зору розгортання. В основному під цим розуміється моделювання топології апаратних засобів, на яких виконується система. По суті, діаграма розгортання - це просто діаграми класів, зосереджені на системних вузлах.

ЗМІСТ

Навчальне видання

ДУДЧЕНКО Олег Миколайович
КАРПОВА Світлана Олегівна

“CASE-ЗАСОБИ РОЗРОБКИ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ”

Методичні вказівки до виконання лабораторних робіт

(українською мовою)

Комп’ютерне складання та верстання
Коректор

Формат 60Х84/16. Ум. друк. арк. 6,3. Тираж 30 прим. Вид. № . Зам. №

Для нотаток