Міністерство освіти і науки України
Університет митної справи та фінансів

ЯКОВЕНКО В. О., УЛЬЯНОВСЬКА Ю. В., КОСТЕНКО В. В., КОСТЕНКО Д. Є., ЛАВРЕНЮК І. В., МОЛОТКОВ О. Н.

ОСНОВИ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ СКЛАДНИХ ОБ'ЄКТІВ І СИСТЕМ

Навчальний посібник

Дніпро
2018

УДК 004.41(075.8)

Рекомендовано до друку
вченою радою Університету митної справи та фінансів
як навчальний посібник (протокол № 4 від 29.10.2018 р.)

Рецензенти:

Є. В. Кочура – *д.е.н., професор, завідувач кафедри електронної економіки та економічної кібернетики Національного технічного університету "Дніпропетровська політехніка";*

В. Б. Говоруха – *д.ф-м.н., професор, завідувач кафедри вищої математики та фізики Дніпровського державного аграрно-економічного університету*

Основи автоматизованого проектування складних об'єктів та систем : навчальний посібник / Яковенко В. О., Ульяновська Ю. В., Костенко В. В., Костенко Д. Є., Лавренюк І. В., Молотков О. Н. – Дніпро : Університет митної справи та фінансів, 2018. – 114 с.

ISBN 978-966-328-142-1

Знання основ автоматизації проектування та вміння працювати із CASE-засобами потрібні будь-якому розробнику програмного забезпечення.

Навчальний посібник містить теоретичний матеріал, завдання та приклади для проведення практичних занять, які допоможуть глибше вивчити ті або інші аспекти автоматизованого проектування складних об'єктів та систем, що інтенсивно розвивається, охоплюючи все нові галузі.

Для студентів спеціальностей 121 "Інженерія програмного забезпечення" та 122 "Комп'ютерні науки", викладачів та фахівців.

ISBN 978-966-328-142-1

ЗМІСТ

ПЕРЕДМОВА

Однією з проблем сучасної науки є розробка і впровадження в практику методів дослідження функціонування складних систем. До класу складних систем належать технологічні, виробничі, енергетичні комплекси, системи автоматизації керування й інших об'єктів. Автоматизація проектування – один із найпотужніших сучасних засобів дослідження подібних систем.

Автоматизація проектування – це синтетична дисципліна, до її складу входять багато інших сучасних інформаційних дисциплін. Технічне забезпечення систем автоматизованого проектування (САПР) грунтується на використанні обчислювальних мереж і телекомунікаційних технологій, у САПР використовуються персональні комп'ютери й робочі станції. Математичне забезпечення САПР вирізняється багатством і розмаїтістю використовуваних методів обчислювальної математики, статистики, математичного програмування, дискретної математики, штучного інтелекту. Програмні комплекси САПР входять до найскладніших сучасних програмних систем, що ґрунтуються на операційних системах Unix, Windows-NT, мовах програмування C, C++, Java та інших сучасних CASE-технологіях, реляційних і об'єктно-орієнтованих системах керування базами даних (СКБД), стандартах відкритих систем і обміну даними в комп'ютерних середовищах.

Знання основ автоматизації проектування та вміння працювати із засобами САПР потрібні будь-якому інженерові-розробнику.

До нинішнього часу створено велику кількість програмно-методичних комплексів для САПР із різним ступенем спеціалізації та прикладною орієнтацією. У результаті автоматизація проектування стала необхідною складовою підготовки інженерів різних спеціальностей.

Підготовка інженерів різних спеціальностей у сфері САПР включає базову й спеціальну компоненти. Найбільш загальні положення, моделі й методики автоматизованого проектування входять у програму курсу, присвяченого основам САПР, детальне вивчення тих методів і програм, які специфічні для конкретних спеціальностей, передбачається в профільних дисциплінах: "Технологія комп'ютерного проектування", "Проектування інформаційних систем".

Розділ 1 містить початкові відомості про процес проектування складних об'єктів та систем, тут викладено основні поняття про системи автоматизованого проектування, пояснено структуру САПР.

У розділі 2 подано відомості про основні методології автоматизованого проектування складних систем.

Навчальний посібник містить завдання та приклади для проведення практичних занять, які допоможуть глибше вивчити ті або інші аспекти об'єктно-орієнтованого проектування, оскільки це найперспективніший сучасний напрям комп'ютерного моделювання, що інтенсивно розвивається, охоплюючи все нові галузі.

Посібник орієнтований на базову підготовку студентів спеціальності "Інформаційні управляючі системи та технології" напряму "Комп'ютерні науки" в галузі автоматизованого проектування.

РОЗДІЛ 1. ОСНОВИ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ

1.1. Загальне визначення проектування

Візуальне моделювання передбачає графічну форму опису моделі і надання результатів дослідження. Основним елементом графічного зображення системи, що моделюється в сучасних середовищах, є структурна схема, побудована з образів окремих компонентів, з'єднаних функціональними зв'язками. Для надання результатів використовується математична графіка, а також дво- і тривимірна анімація.

Ні для кого не секрет, що ідеалізована схема проектування, яка обіцяє автоматичний синтез системи із заданими властивостями, досить далека від реальності. Робота проектувальника багато в чому спирається на інтуїцію, добирання і порівняння багатьох різних варіантів проектованої системи в умовах досить твердих часових обмежень. Тому простота і швидкість реалізації окремого варіанта системи та його оцінок прямо впливає на якість проектування. Сучасні інструменти візуального моделювання дають проектувальникові новий засіб проектування – віртуальний випробувальний стенд, на якому він може проводити експерименти, використовуючи ту ж технологію, що й на звичайному випробувальному стенді, але значно швидше.

Характерна риса сучасних технічних об'єктів – різке підвищення їхньої логічної складності, пов'язане з широким упровадженням убудованих мікропроцесорних систем. Для моделювання об'єктів з такими властивостями, що поєднують у собі безперервні й дискретні аспекти поведінки, використовується математична модель, названа гібридною системою. У цього терміна є синоніми: безперервно-дискретна система або система зі змінною структурою. *Проектування* технічного об'єкта – створення, перетворення й подання в прийнятній формі образу цього ще не існуючого об'єкта. Образ об'єкта або його складових частин може створюватися в уяві людини в результаті творчого процесу або генеруватися відповідно до деяких алгоритмів у процесі взаємодії людини й ЕОМ. У кожному разі інженерне проектування починається за наявності вираженої потреби суспільства в деяких технічних об'єктах, це можуть бути об'єкти будівництва, промислові вироби або процеси. Проектування включає розробку технічної пропозиції й (або) технічного завдання (ТЗ), що відбивають ці потреби, і реалізацію ТЗ у вигляді проектної документації.

ТЗ представляють у вигляді деяких документів, і воно є вихідним (первинним) описом об'єкта. Результатом проектування, як правило, служить повний комплект документації, що містить достатні відомості для виготовлення об'єкта в заданих умовах. Ця документація і є *проектом,* точніше остаточним описом об'єкта. Отже, проектування – процес, що полягає в одержанні й перетворенні вихідного опису об'єкта в остаточний опис на основі виконання комплексу робіт дослідницького, розрахункового й конструкторського характеру.

Перетворення вихідного опису на остаточний породжує ряд проміжних описів, що підсумовують виконання деяких завдань і використовуються під час обговорення та прийняття проектних рішень щодо закінчення або продовження проектування.

Проектування, за якого всі проектні рішення або їх частину одержують шляхом взаємодії людини й ЕОМ, називають *автоматизованим,* на відміну від ручного (без використання ЕОМ) або автоматичного (без участі людини на проміжних етапах). Автоматичне проектування можливе тільки в окремих випадках, для порівняно нескладних об'єктів. Нині переважає автоматизоване проектування. Система, що реалізує автоматизоване проектування, являє собою *систему автоматизованого проектування* – САПР (в англомовному написанні *CAD System – Computer Aided Design System).*

Проектування складних об'єктів грунтується на застосуванні ідей і принципів, викладених у ряді теорій і підходів. Найпоширеніший системний підхід, на якому базуються різні методики й технології проектування складних систем.

1.2. Системний підхід до проектування. Принципи системного підходу

Основні ідеї та принципи проектування складних систем виражені в системному підході. Загальний принцип системного підходу полягає в розгляді частин явища або складної системи з урахуванням їх взаємодії. *Системний підхід* передбачає виявлення структури системи, типізацію зв'язків, визначення атрибутів, аналіз впливу зовнішнього середовища. Мета досягається в багатокрокових процесах прийняття рішень.

Проектування й супровід будь-яких сучасних систем неможливі без системного підходу.

Інтерпретація й конкретизація системного підходу мають місце в ряді відомих підходів з іншими назвами, маються на увазі структурний, блочно-ієрархічний, об'єктно-орієнтований підходи.

Структурний підхід як різновид системного передбачає синтез варіантів системи з компонентів (блоків) і оцінювання варіантів з попереднім прогнозуванням характеристик компонентів.

Блочно-ієрархічний підхід до проектування використовує ідеї декомпозиції складних описів об'єктів і, відповідно, засобів їх поділу на ієрархічні рівні й аспекти, вводить поняття стилю проектування (висхідне й спадне), встановлює зв'язок між параметрами сусідніх ієрархічних рівнів.

Ряд важливих структурних принципів, використовуваних у розробці інформаційних систем, насамперед їхнього програмного забезпечення (ПЗ), виражений в *об'єктно-орієнтованому підході* до проектування. Такий підхід має певні переваги у розв'язанні проблем керування складністю та інтеграції ПЗ:

1) вносить у моделі додатків більшу структурну визначеність, розподіляючи дані в додатку й процедури між класами об'єктів;

2) скорочує обсяг специфікацій завдяки введенню в описи ієрархії об’єктів і відносин спадкування між властивостями об’єктів різних рівнів ієрархії;

3) зменшує ймовірність перекручування даних внаслідок помилкових дій за рахунок обмеження доступу до певних категорій даних в об’єктах. Опис у кожному класі об’єктів припустимих звертань до них і прийнятих форматів повідомлень полегшує узгодження та інтеграцію ПЗ.

Для всіх підходів до проектування складних систем характерні також деякі особливості.

1. Структуризація процесу проектування, що виражається декомпозицією проектних завдань і документації, виділенням стадій, етапів, проектних процедур. Ця структуризація є сутністю блочно-ієрархічного підходу до проектування.

2. Ітераційний характер проектування.

3. Типізація та уніфікація проектних рішень і засобів проектування.

1.3. Стадії проектування за системного підходу

Стадії проектування – найбільші частини проектування як процесу, що розвивається в часі. У загальному випадку виділяють стадії науково-дослідних робіт (НДР), ескізного проекту або дослідно-конструкторських робіт, технічного, робочого проектів, випробувань дослідних зразків або партій. Стадію НДР іноді називають передпроектними дослідженнями або стадією технічної пропозиції. Очевидно, що в міру переходу від стадії до стадії ступінь деталізації та ретельність опрацювання проекту зростають, і робочий проект має бути цілком достатнім для виготовлення дослідних або серійних зразків. Близьким до визначення стадії, але менш чітко застереженим є поняття етапу проектування.

Стадії (етапи) проектування поділяють на складові частини, які називаються *проектними процедурами.* Прикладами проектних процедур можуть служити підготовка деталіровочних креслень, аналіз кінематики, моделювання перехідного процесу, оптимізація параметрів та інші проектні завдання. У свою чергу, проектні процедури можна розчленувати на дрібніші компоненти, які називаються *проектними операціями.* Наприклад, під час аналізу міцності деталі сітковими методами операціями може бути побудова сітки, вибір або розрахунок зовнішніх впливів, моделювання полів напруг і деформацій, подання результатів моделювання в графічній і текстовій формах. Проектування зводиться до виконання деяких послідовностей проектних процедур – *маршрутів проектування.*

Іноді розробку ТЗ на проектування називають *зовнішнім* проектуванням, а реалізацію ТЗ – *внутрішнім* проектуванням.

1.4. Оформлення документації. Зміст технічного завдання на проектування

У ТЗ на проектування об’єкта зазначаються вихідні дані.

1. Призначення об’єкта.

2. Умови експлуатації. Поряд з якісними характеристиками (у вербальній формі) є числові параметри, що називаються *зовнішніми,* для них зазначаються області припустимих значень. Приклади зовнішніх параметрів: температура навколишнього середовища, зовнішні сили, електричні напруги, навантаження тощо.

3. Вимоги до *вихідних* параметрів, тобто до величин, що характеризують властивості об'єкта і цікавлять споживача. Ці вимоги виражені у вигляді *умов працездатності*

$$y_i R T_i ,$$

де y_i – i-й вихідний параметр;

$R \in \{=, <, >, \geq, \leq\}$ – вид відношення;

T_i – норма i-го вихідного параметра. Якщо R = (дорівнює), потрібно задати необхідну точність виконання рівності.

Приклади умов працездатності: витрата палива на 100 км пробігу автомобіля <8 л; коефіцієнт підсилення підсилювача на середніх частотах >300; швидкодія процесора >40 Мфлопс.

1.5. Визначення і класифікація моделей, що використовуються для автоматизованого проектування. Процес та особливості побудови і дослідження комп'ютерних моделей

Система (у реальному світі або програмному забезпеченні), як правило, надзвичайно складна. Отже, щоб мати можливість аналізувати її та керувати нею, слід виконати декомпозицію системи на менші й доступніші для розуміння складові частини, що її можна представити як *моделі* (model), в котрих містяться і формально описуються важливі аспекти системи.

Таким чином, для проектування програмної системи дуже корисно спочатку створити моделі, що відображають важливі деталі проблеми з реального життя, для розв'язання якої і конструюється система. Такі моделі повинні містити чітко пов'язані елементи. Моделі можна описати, виділивши *статичну* (static) або *динамічну* (dynamic) інформацію про систему. *Статична модель* (static model) описує структурні властивості, тоді як *динамічна модель* (dynamic model) описує поведінкові властивості системи.

Моделлю об'єкта називається будь-який інший об'єкт, окремі властивості якого цілком або частково збігаються з властивостями вихідного.

Слід чітко розуміти, що вичерпно повної модель бути не може. Вона завжди обмежена і повинна лише відповідати цілям моделювання, відбиваючи рівно стільки властивостей вихідного об'єкта і в такій повноті, скільки необхідно для конкретного дослідження. Вихідний об'єкт може бути реальним уявним. З уявлюваними об'єктами в інженерній практиці маємо справу на ранніх етапах проектування технічних систем.

Модель створюється для досліджень, які на реальному об'єкті проводити або неможливо, або дорого, або просто незручно. Можна виділити кілька цілей, заради яких створюються моделі і ряд основних типів досліджень.

- Модель як засіб осмислення допомагає виявити взаємозалежності змінних, характер їх зміни в часі, знайти існуючі закономірності. Під час складання моделі стає більш зрозумілою структура досліджуваного об'єкта, розкриваються важливі причинно-наслідкові зв'язки. У процесі моделювання поступово відбувається поділ властивостей вихідного об'єкта на суттєві й другорядні з погляду сформульованих вимог до моделі. Ми намагаємося знайти у вихідному об'єкті тільки ті риси, що безпосередньо стосуються однієї зі сторін функціонування, що зацікавила нас. У певному сенсі вся наукова діяльність зводиться до побудови і дослідження моделей природних явищ.
- Модель як засіб прогнозування дає можливість навчитися прогнозувати поведінку об'єкта і випробовувати різні варіанти керування ним. Експериментувати з реальним об'єктом часто буває незручно, а іноді й просто небезпечно або взагалі неможливо через ряд причин: велика тривалість експерименту, ризик пошкодити об'єкт або знищити його, відсутність реального об'єкта.
- Побудовані моделі можуть використовуватися для пошуку оптимальних співвідношень параметрів, дослідження особливих (критичних) режимів роботи.
- Модель також може в деяких випадках замінювати вихідний об'єкт для навчання, наприклад використовуватися як тренажер у підготовці персоналу до наступної роботи в реальній обстановці або бути досліджуваним об'єктом у віртуальній лабораторії. Моделі, реалізовані у вигляді модулів, застосовуються і як імітатори об'єктів керування в ході стендових іспитів систем керування, і на ранніх стадіях проектування замінюють майбутні апаратно реалізовані системи керування.

Моделі можна умовно поділити на дві групи: матеріальні та ідеальні. Відповідно, слід розрізнювати предметне й абстрактне моделювання. Основні різновиди предметного моделювання – фізичне й аналогове.

Фізичним називають таке моделювання (макетування), коли об'єктові ставиться у відповідність його збільшена або зменшена копія, що створюється на основі теорії подібності, що й дозволяє стверджувати, що в моделі збереглися необхідні властивості. У фізичних моделях, крім геометричних пропорцій, може бути збережено, наприклад, матеріал або колірну гаму вихідного об'єкта, а також інші властивості, необхідні для конкретного дослідження.

Аналогове моделювання грунтується на заміні вихідного об'єкта об'єктом іншої фізичної природи, що має аналогічні характеристики. Коливання й резонанс можна вивчати як за допомогою механічних систем так, і за допомогою електричних ланцюгів. Під час аналового моделювання

важливо побачити в об'єкті-замінніку потрібні риси і правильно їх інтерпретувати. Зазвичай для об'єктів з аналогічними властивостями можна використовувати таку саму математичну модель.

І фізичне, і аналогове моделювання як основний спосіб дослідження передбачає проведення натурного експерименту з моделлю, але цей експеримент виявляється більш привабливим, ніж експеримент із вихідним об'єктом. Наприклад, свого часу дуже широко використовувалися аналогові обчислювальні машини. Моделювання з їх допомогою грунтується на тому, що електричні явища подібні до багатьох явищ іншої фізичної природи.

Ідеальні моделі – це абстрактні образи реальних або уявних об'єктів. Розрізнюють два типи ідеального моделювання: інтуїтивне і знакове.

Про інтуїтивне моделювання говорять, коли не можуть навіть описати використовувану модель, хоча вона й існує, але беруться з її допомогою передбачати або пояснювати навколишній світ. Ми знаємо, що живі істоти можуть пояснювати і передбачати явища без видимої присутності фізичної або абстрактної моделі. Наприклад, у цьому сенсі, життєвий досвід кожної людини може вважатися його інтуїтивною моделлю навколишнього світу.

Знаковим називається моделювання, що використовує як моделі знаки або символи: схеми, графіки, креслення, тексти на різних мовах, включаючи формальні, математичні формули і теорії. Обов'язковим учасником знакового моделювання є інтерпретатор знакової моделі, найчастіше людина, але з інтерпретацією може впоратися і комп'ютер. Креслення, тексти, формули самі по собі не мають ніякого сенсу без того, хто розуміє їх і використовує у своїй повсякденній діяльності.

Найважливішим різновидом знакового моделювання є *математичне моделювання.* Абстрагуючись від фізичної природи об'єктів, математика вивчає ідеальні об'єкти.

Модель може бути фізичним об'єктом (макет, стенд) або специфікацією. Серед моделей-специфікацій розрізняють згадані вище функціональні, поведінкові, інформаційні, структурні моделі (опису). Ці моделі називають *математичними,* якщо вони формалізовані засобами апарату й мови математики.

Своєю чергою, математичні моделі можуть бути геометричними, топологічними, динамічними, логічними тощо, коли вони відбивають відповідні властивості об'єктів. Поряд з математичними моделями для проектування використовують розглянуті нижче функціональні IDEF 0-моделі, інформаційні моделі у вигляді діаграм сутність – відношення, геометричні моделі-креслення. Надалі, якщо немає спеціального застереження, під словом "модель" матимемо на увазі математичну модель (ММ).

Математична функціональна модель у загальному випадку являє собою алгоритм обчислення вектора вихідних параметрів *Y* із заданими векторами параметрів елементів *X* і зовнішніх параметрів *Q*.

Математичні моделі можуть бути символічними й числовими. У разі використання *символічних* моделей оперують не значеннями величин, а їх-

німи символічними позначеннями (ідентифікаторами). *Числові* моделі можуть бути *аналітичними,* тобто їх можна подати у вигляді явно виражених залежностей вихідних параметрів *Y* від *параметрів внутрішніх X* і зовнішніх *Q*, або *алгоритмічними,* у яких зв'язок *Y, X* і *Q* заданий неявно у вигляді алгоритму моделювання. Найважливіший окремий випадок алгоритмічних моделей – *імітаційні,* вони відображають процеси в системі за наявності зовнішніх впливів на систему. Інакше кажучи, імітаційна модель – це алгоритмічна поведінкова модель.

Класифікацію математичних моделей виконують також за іншими ознаками. Так, залежно від належності до того або іншого ієрархічного рівня виділяють моделі рівнів системного, функціонально-логічного, макрорівня (зосередженого) і мікрорівня (розподіленого).

За характером використовуваного для опису математичного апарату розрізняють моделі лінгвістичні, теоретико-множинні, абстрактно-алгебраїчні, нечіткі, автоматні тощо.

Наприклад, на системному рівні переважно застосовують моделі систем масового обслуговування й мережі Петрі, на функціонально-логічному рівні – автоматні моделі на основі апарата передатних функцій або кінцевих автоматів, на макрорівні – системи алгебраїчних і диференціальних рівнянь, на мікрорівні – диференціальні рівняння в частинних похідних. Особливе місце займають геометричні моделі, використовувані в системах конструювання.

Крім того, введено поняття повних моделей і макромоделей, моделей статичних і динамічних, детермінованих і стохастичних, аналогових і дискретних, символічних і числових.

Повна модель об'єкта, на відміну від *макромоделі,* описує не тільки процеси на зовнішніх зв'язках об'єкта, що моделюється, але й внутрішні для об'єкта процеси.

Статичні моделі описують статичні стани, в них немає такої не залежної змінної, як час. *Динамічні* моделі відображають поведінку системи, тобто в них обов'язково використовується час.

Стохастичні й детерміновані моделі розрізнюють залежно від урахування або неврахування випадкових факторів.

В *аналогових* моделях фазові змінні – безперервні величини, у *дискретних* – дискретні, в окремому випадку дискретні моделі є *логічними (булевими),* у них стан системи та її елементів описується булевими величинами. У ряді випадків корисно застосовувати *змішані* моделі, в яких одна частина підсистем характеризується аналоговими моделями, інша – логічними.

Інформаційні моделі належать до інформаційної страти автоматизованих систем, їх використовують насамперед в інфологічному проектуванні баз даних для опису зв'язків між одиницями інформації.

Найбільші труднощі виникають у разі створення моделей слабоструктурованих систем, що характерно насамперед для системного рівня проек-

тування. Тут значна увага приділяється експертним методам. У теорії систем сформульовано загальні рекомендації щодо добору експертів для розробки моделі, організації експертизи, обробки отриманих результатів. Досить загальний підхід до побудови моделей складних слабоструктурованих систем виражений у методиках IDEF.

Комп'ютерна модель – це програмна реалізація математичної моделі, доповнена різними службовими програмами (що малюють і змінюють графічні образи в часі). Комп'ютерна модель має дві складові – програмну й апаратну. Програмна складова так само є абстрактною знаковою моделлю. Це лише інша форма абстрактної моделі, що, однак, може інтерпретуватися вже не тільки математиками і програмістами, але й технічним пристроєм – процесором комп'ютера.

Комп'ютерна модель виявляє властивості фізичної моделі, коли вона, а точніше її абстрактні складові програми, інтерпретується фізичним пристроєм, комп'ютером. Комп'ютерна модель як фізичний пристрій може входити до складу випробувальних стендів, тренажерів і віртуальних лабораторій. Цей спеціальний різновид моделей, що поєднують у собі абстрактні й фізичні риси, має унікальний набір корисних властивостей. Головна з них – простота створення і модифікації моделі. Заново пишеться і змінюється тільки програма ("м'яка" складова моделі), водночас апаратура комп'ютера ("тверда" складова) залишається незмінно. Комп'ютерна модель в умовах висловлених про неї припущень нічим не відрізняється від реального об'єкта. Її можна "підключати" до реальних об'єктів так само, як її фізичні прототипи. Якщо, наприклад, фізичні осцилятори використовуються як джерела періодичного сигналу в реальній апаратурі, то й комп'ютерна модель за наявності спеціального пристрою, цифро-аналогового перетворювача (ЦАП), може успішно служити джерелом такого сигналу.

Процес побудови і дослідження комп'ютерних моделей прийнято також називати обчислювальним експериментом. Його можна подати як послідовність таких основних кроків:

1. Виділення істотних для даного дослідження властивостей вихідного об'єкта і побудова математичної моделі.

2. Проектування і налагодження комп'ютерної моделі.

3. Оцінка адекватності побудованої комп'ютерної моделі. Як правило, оцінка адекватності призводить до перегляду вимог до моделі і повернення на етап 1: доводиться уточнювати або заново будувати математичну модель.

4. Дослідження моделі.

5. Аналіз отриманих результатів. Можна виявити, що запланованих експериментів недостатньо для завершення робіт або що потрібно знову уточнити математичну модель.

Сучасні інструменти комп'ютерного моделювання дають можливість значною мірою автоматизувати обчислювальний експеримент.

З вищезазначеного випливає, що моделювання – процес циклічний, у ньому ті самі операції повторюються. Циклічність обумовлена двома обставинами: технологічними, пов'язаними з "прикрими" помилками, допущеними на кожному з розглянутих етапів моделювання, та "ідеологічними", пов'язаними з уточненням моделі, та навіть з відмовою від неї, і переходом до іншої. Ще один додатковий "зовнішній" цикл може з'явитися, коли треба розширити область застосування моделі і змінити вихідні дані, які вона мала правильно враховувати, або допущення, за яких вона повинна бути правдивою.

Після побудови математичної моделі слід скласти програму, яка її реалізує. Для створення програмного додатка потрібно описати проблему і вимоги до системи. Етап *аналізу* (analysis) полягає в дослідженні проблеми, а не в пошуках шляхів її розв'язання. Наприклад, під час розробки нової інформаційної системи для комп'ютерної бібліотеки треба описати економічні процеси, позв'язані з її використанням. Розробляючи додаток, необхідно також забезпечити високий рівень і докладний опис логіки розв'язання, що відповідає вимогам до системи й обмеження, що накладаються. У процесі *проектування* (design) основна увага приділяється логічному розв'язанню, що забезпечує виконання основних вимог.

На історично ранніх етапах комп'ютерного моделювання програми створювалися або на мовах програмування "високого рівня" (Фортрані, Алголі), або навіть мовою Асемблера.

Зараз розроблено велику кількість систем або пакетів моделювання. У системі автоматизації моделювання (далі – система моделювання) математична модель записується на якійсь формальній вхідній мові моделювання, а потім автоматично, за допомогою відповідного транслятора, перекладається на мову, зрозумілу комп'ютерові (рис. 1.1). Часто за проміжну використовують яку-небудь універсальну мову програмування (Fortran, Pascal, Java, C тощо). У цьому випадку трансляція відбувається у два етапи: на першому опис мовою моделювання транслюється в проміжний текст мовою програмування, а на другому цей текст компілюється якимось компілятором мови програмування, написаним для конкретної операційної системи і комп'ютера. У моделюючу програму, крім згенерованого коду, необхідного для реалізації конкретної моделі, автоматично включаються вже готові модулі підтримки періоду виконання, надані системою моделювання (механізм просування модельного часу, обчислювальні бібліотеки, графічні функції). Система моделювання повинна також автоматично переводити вхідні дані моделі, записані у зручній для аналізу формі, в машинну форму подання й аналогічну операцію виконувати з вхідними даними в машинній формі. Іншими словами, інтерпретувати вхідні і вихідні дані, щоб полегшити роботу проектувальникові.

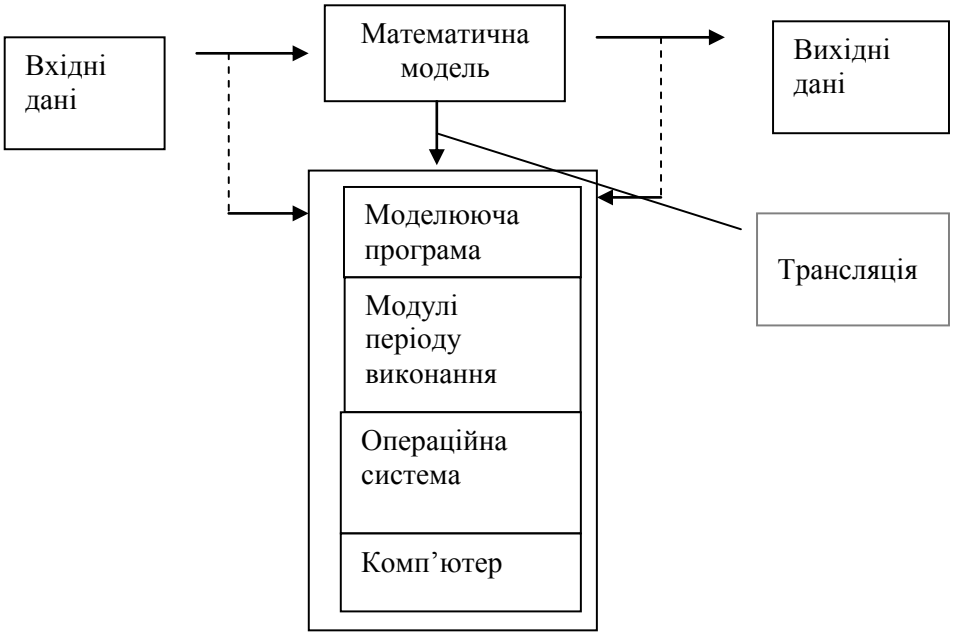


Рис. 1.1. Будова комп'ютерної моделі

З погляду користувача сучасних пакетів побудова ком'ютерної моделі зводиться в основному до перекладу опису моделі з мови математики на вхідну мову використовуваної системи і до вибору числового методу з наявних для одержання необхідного рішення.

1.6. Типові технологічні проектні операції

Створити проект об'єкта (виробу або процесу) означає вибрати структуру об'єкта, знайти значення всіх його параметрів і подати результати у встановленій формі. Результати (проектна документація) можуть бути виражені у вигляді креслень, схем, пояснювальних записок, програм для програмно-керованого технологічного устаткування й інших документів на папері або на машинних носіях інформації.

Розробка (або вибір) структури об'єкта – це проектна процедура, яку називають *структурним синтезом,* а розрахунок (або вибір) значень параметрів елементів X − процедура *параметричного синтезу.*

Завдання структурного синтезу формулюється в системотехніці як *завдання прийняття рішень (ЗПР).* Її суть полягає у визначенні мети, множини можливих рішень та обмежувальних умов.

Класифікацію ЗПР здійснюють за рядом ознак. Розрізнюють завдання одно- і багатокритеріальні. За ступенем невизначеності бувають ЗПР детерміновані, ЗПР в умовах ризику (за наявності у формулюванні завдання випадкових параметрів), ЗПР в умовах невизначеності, тобто коли вихідна інформація неповна або невірогідна.

Реальні завдання проектування, як правило, багатокритеріальні. Одна з основних проблем постановки багатокритеріальних завдань – установлення правил переваги варіантів. Способи розв'язання типових задач вивчаються в дисциплінах, присвячених методам оптимізації та математичному програмуванню.

Наявність випадкових факторів ускладнює виконання ЗПР. Основні підходи до ЗПР в умовах ризику полягають або у варіанті "для найгіршого випадку", або в урахуванні в цільовій функції математичного сподівання й дисперсії вихідних параметрів. У першому випадку завдання виконують як детерміноване із завищеними вимогами до якості рішення, що є головним недоліком підходу. У другому випадку вірогідність результатів набагато вища, але виникають труднощі з оцінкою цільової функції. Застосування методу Монте-Карло у випадку алгоритмічних моделей стає єдиною альтернативою, а отже, для виконання потрібні значні обчислювальні ресурси.

Існують дві групи ЗПР в умовах невизначеності. Одна з них виконується за наявності протидії розумного супротивника. Такі завдання вивчаються в теорії ігор, для проектування в техніці вони не характерні. У другій групі досягненню мети протидіють сили природи. Для їх виконання корисно застовувати теорію й методи нечітких множин.

Наприклад, під час синтезу структури автоматизованої системи постановка завдання повинна включати такі вихідні дані:

• множина виконуваних системою функцій (інакше кажучи, множина робіт, кожна з яких може складатися з однієї або більше операцій); можливо, що в цій множині є часткова впорядкованість робіт, що може бути подана у вигляді орієнтованого графа, де вершини відповідають роботам, а дуги – відношенням порядку;

• типи припустимих для використання серверів (машин), що виконують функції системи;

• множина зовнішніх джерел і споживачів інформації;

• у багатьох випадках задається також деяка вихідна структура системи у вигляді взаємозалежної сукупності серверів певних типів; ця структура може розглядатись як узагальнена надлишкова або як варіант першого наближення;

• різного роду обмеження, зокрема на витрати матеріальних ресурсів і (або) на час виконання функцій системи.

Завдання полягає в синтезі (або корекції) структури, визначенні типів серверів (програмно-апаратних засобів), розподілі функцій за серверами таким чином, щоб досягався екстремум цільової функції під час дотримання заданих обмежень.

Конструювання, розробка технологічних процесів, оформлення проектної документації – окремі випадки структурного синтезу.

Завдання параметричного синтезу називають *параметричною оптимізацією* (або оптимізацією), якщо її виконують як завдання математичного програмування, тобто

$$extr\ F(X), X \in D_x,$$

де $F(X)$ – цільова функція;

X – вектор керованих (їх називають також проектними або варійованими) параметрів;

$D_x = \{X \mid \varphi(X) < 0, \psi(X) = 0\}$ – припустима область;

$\varphi(X)$ і $\psi(X)$ – функції-обмеження.

Наступна після синтезу група проектних процедур – процедури аналізу. Мета *аналізу* – одержання інформації про характер функціонування та значення вихідних параметрів *Y* із заданими структурою об'єкта, відомостями про зовнішні параметри *Q* і параметрами елементів *X*. Якщо задано фіксовані значення параметрів *X* і *Q*, то має місце процедура *одноваріантного аналізу,* що зводиться до розв'язання рівнянь математичної моделі та обчислення вектора вихідних параметрів *Y*. Якщо задано статистичні відомості про параметри *X* і потрібно одержати оцінки числових характеристик розподілів вихідних параметрів (наприклад, оцінки математичних сподівань і дисперсій), то це процедура *статистичного аналізу.* Якщо потрібно розрахувати матриці абсолютної *A* та (або) відносної *B* чутливості, то має місце завдання *аналізу чутливості.*

У процедурах різноманітного аналізу визначається вплив зовнішніх параметрів, розкиду й нестабільності параметрів елементів на вихідні параметри. Процедури статистичного аналізу й аналізу чутливості – характерні приклади процедур різноманітного аналізу.

1.7. Інструментальні засоби проектування: основні типи, структура та різновиди

Автоматизація проектування здійснюється САПР. Прийнято виділяти в САПР системи функціонального, конструкторського й технологічного проектування. Перші з них називають системами розрахунків та інженерного аналізу або системами *CAE (Computer Aided Engineering).* Системи конструкторського проектування називають системами *CAD (Computer Aided Design).* Проектування технологічних процесів становить частину технологічної підготовки виробництва і виконується в системах *CAM (Computer Aided Manufacturing).* Функції координації роботи систем CAE/CAD/CAM, керування проектними даними й проектуванням покладено на систему керування проектними даними *PDM (Product Data Management).*

Уже на стадії проектування потрібні послуги системи *керування ланцюжками поставок (SCM — Supply Chain Management),* яку іноді називають системою *Component Supplier Management (CSM).* На етапі виробництва ця система управляє поставками необхідних матеріалів і комплектуючих.

Інформаційна підтримка етапу виробництва продукції здійснюється *автоматизованими системами керування підприємством (АСКП)* та *автоматизованими системами керування технологічними процесами (АСКТП)*. До АСКП належать системи планування й керування підприємством *ERP (Enterprise Resource Planning),* планування виробництва й вимог до матеріалів MRP-*2 (Manufacturing Requirement Planning),* виробнича виконавча система *MES (Manufacturing Execution Systems),* а також SCM і система керування відносинами із замовниками *CRM (Customer Requirement Management).*

Найбільш розвинені системи ERP виконують різні бізнес-функції, пов'язані з плануванням виробництва, закупівлями, збутом продукції, аналізом перспектив маркетингу, керуванням фінансами, персоналом, складським господарством, обліком основних фондів тощо. Системи MRP-2 орієнтовані переважно на бізнес-функції, безпосередньо пов'язані з виробництвом, а системи MES – на оперативне керування проектуванням, виробництвом і маркетингом.

На етапі реалізації продукції виконуються функції керування відносинами із замовниками й покупцями, проводиться аналіз ринкової ситуації, визначаються перспективи попиту на плановані вироби. Ці функції здійснює система CRM. Маркетингові завдання іноді покладають на систему *S&SM (Sales and Service Management),* що, крім того, використовується для розв'язання проблем обслуговування виробів. На етапі експлуатації застосовують також спеціалізовані комп'ютерні системи, зайняті питаннями ремонту, контролю, діагностики експлуатованих систем.

Автоматизовані системи керування технологічними процесами контролюють і використовують дані про стан технологічного устаткування і перебіг технологічних процесів. Саме їх найчастіше називають системами промислової автоматизації.

Для виконання диспетчерських функцій (збирання й обробка даних про стан устаткування й технологічних процесів) і розробки ПЗ для вбудованого устаткування до складу АСКТП уводять систему *SCADA (Supervisory Control and Data Acquisition).* Безпосереднє програмне керування технологічним устаткуванням здійснюють за допомогою системи *CNC (Computer Numerical Control)* на базі контролерів (спеціалізованих комп'ютерів, що називаються промисловими), які вбудовані в технологічне устаткування.

Останнім часом зусилля багатьох компаній, що виробляють програмно-апаратні засоби АС, спрямовані на створення *систем електронного бізнесу (E-Commerce).* Завдання, виконувані системами E-Commerce, зводяться не тільки до організації на сайтах Internet вітрин товарів і послуг. Вони поєднують в інформаційному просторі запити замовників і дані про можливості множини організацій, що спеціалізуються на наданні різних послуг і виконанні тих або інших процедур і операцій з проектування, виготовлення, поставок замовлених виробів. Такі системи E-Commerce називають

системами керування даними в інтегрованому інформаційному просторі *CPC (Collaborative Product Commerce)* або *PLM (Product Lifecycle Management).* Проектування безпосередньо під замовлення дозволяє домогтися найкращих параметрів створюваної продукції, а оптимальний вибір виконавців і ланцюжків поставок веде до мінімізації часу й вартості виконання замовлення. Характерна риса *CPC* – забезпечення взаємодії багатьох підприємств, тобто технологія *CPC* є основою, що інтегрує інформаційний простір, у якому функціонують САПР, ERP, PDM, SCM, CRM та інші АС різних підприємств.

Як і будь-яка складна система, САПР складається з підсистем. Розрізняють підсистеми проектування та обслуговування.

Підсистеми проектування безпосередньо виконують проектні процедури. Прикладами можуть служити підсистеми геометричного тривимірного моделювання механічних об'єктів, виготовлення конструкторської документації, схемотехнічного аналізу, трасування з'єднань у друкованих платах.

Обслуговуючі підсистеми забезпечують функціонування проектувальних підсистем, їхню сукупність часто називають системним середовищем (або оболонкою) САПР. Типовими обслуговуючими підсистемами є підсистеми керування проектними даними, підсистеми розробки й супроводу програмного забезпечення *CASE (Computer Aided Software Engineering),* що навчають підсистеми для освоєння користувачами технологій, реалізованих у САПР.

Структурування САПР за різними аспектами обумовлює поява *видів забезпечення* САПР. Прийнято виділяти сім видів забезпечення САПР:

• *технічне,* що включає різні апаратні засоби (ЕОМ, периферійні пристрої, мережний комутаційне устаткування, лінії зв'язку, вимірювальні засоби);

• *математичне,* що поєднує математичні методи, моделі й алгоритми для виконання проектування;

• *програмне,* що представляється комп'ютерними програмами САПР;

• *інформаційне,* що складається з бази даних, СКБД, а також включає інші дані, використовувані під час проектування; зазначимо, що вся сукупність використовуваних для проектування даних називається інформаційним фондом САПР, а база даних разом із СКБД зветься банком даних;

• *лінгвістичне,* що виражається мовами спілкування між проектувальниками й ЕОМ, мовами програмування й мовами обміну даними між технічними засобами САПР;

• *методичне,* що включає різні методики проектування, іноді до нього зараховують також математичне забезпечення;

• *організаційне,* що являє собою штатні розклади, посадові інструкції та інші документи, які регламентують роботу проектного підприємства.

Класифікацію САПР здійснюють за рядом ознак, наприклад за додатком, цільовим призначенням, масштабами (комплексність завдань), характером базової підсистеми – ядра САПР.

За додатками найбільш представницькими й широко використовуваними є такі групи САПР.

1. САПР для застосування в галузях загального машинобудування. Їх часто називають машинобудівними САПР або системами *MCAD (Mechanical CAD).*

2. САПР для радіоелектроніки: системи *ECAD (Electronic CAD)* або *EDA (Electronic Design Automation).*

3. САПР у галузі архітектури й будівництва.

Крім того, існує велика кількість спеціалізованих САПР, що виділяються в зазначених групах або формують самостійну галузь у класифікації. Прикладами таких систем є САПР більших інтегральних схем (БІС); САПР літальних апаратів; САПР електричних машин тощо.

За цільовим призначенням розрізнюють САПР або підсистеми САПР, що забезпечують різні аспекти (страти) проектування. Так, у складі MCAD з'являються розглянуті вище CAE/CAD/CAM-системи.

Відповідно до масштабів розрізнюють окремі програмно-методичні комплекси (ПМК) САПР, наприклад: комплекс аналізу міцності механічних виробів за методом кінцевих елементів (МКЕ) або комплекс аналізу електронних схем; системи ПМК; системи з унікальними архітектурами не тільки програмного (software), але й технічного (hardware) забезпечень.

За характером базової підсистеми існують такі різновиди САПР.

1. САПР на базі підсистеми машинної графіки й геометричного моделювання. Ці САПР орієнтовані на додатки, де основною процедурою проектування є конструювання, тобто визначення просторових форм і взаємного розташування об'єктів. До цієї групи систем належить більшість САПР у галузі машинобудування, створених на базі графічних ядер. Нині широко використовують уніфіковані графічні ядра, застосовувані більш ніж в одній САПР (ядра Parasolid фірми EDS Unigraphics і ACIS фірми Intergraph).

2. САПР на базі СКБД. Вони орієнтовані на додатки, в яких за допомогою порівняно нескладних математичних розрахунків опрацьовується великий обсяг даних. Такі САПР переважно зустрічаються в техніко-економічних додатках, наприклад під час проектування бізнес-планів, але також і під час проектування об'єктів, подібних до щитів керування в системах автоматики.

3. САПР на базі конкретного прикладного пакета. Фактично це автономно використовувані ПМК, наприклад імітаційного моделювання виробничих процесів, розрахунку міцності за МКЕ, синтезу й аналізу систем автоматичного керування тощо. Часто такі САПР належать до систем CAE. Прикладами можуть служити програми логічного проектування на базі мови VHDL, математичні пакети типу MathCAD.

4. Комплексні (інтегровані) САПР, що складаються із сукупності підсистем попередніх видів. Характерними прикладами комплексних САПР є CAE/CAD/CAM-системи в машинобудуванні або САПР БІС. Так, САПР

БІС містить у собі СКБД і підсистеми проектування компонентів, принципових, логічних і функціональних схем, топології кристалів, тестів для перевірки придатності виробів. Для керування такими складними системами застосовують спеціалізовані *системні середовища.*

1.8. Особливості проектування автоматизованих інформаційних систем

1.8.1. Етапи проектування

Проектування АС безпосередньо стосуються два напрямки діяльності: 1) проектування АС конкретних підприємств (галузей) на базі готових програмних і апаратних компонентів за допомогою спеціальних інструментальних засобів розробки; 2) проектування згаданих компонентів АС та інструментальних засобів, орієнтованих на багаторазове застосування під час розробки багатьох конкретних автоматизованих систем.

Сутність першого напряму можна охарактеризувати словами *системна інтеграція* (інше близьке поняття має назву *консалтинг).* Розробник АС повинен бути фахівцем у сфері системотехніки, добре знати відповідні міжнародні стандарти, стан і тенденції розвитку інформаційних технологій і програмних продуктів, володіти інструментальними засобами розробки додатків (CASE-Засобами) і бути готовим до сприйняття й аналізу автоматизованих процесів у співробітництві зі спеціалістами-прикладниками. Існує ряд фірм, що спеціалізуються на розробці проектів АС (наприклад, Price Waterhouse, Jet Info, Consistent Software, Interface тощо).

Другий напрямок більшою мірою належить до розробки ММ й ПЗ для реалізації функцій АС-моделей, методів, алгоритмів, програм на базі знання системотехніки, методів аналізу й синтезу проектних рішень, технологій програмування, операційних систем та ін. Існує ряд загальновідомих технологій (методик) проектування ПЗ АС, серед яких насамперед слід назвати компонентно-компонентно-орієнтовану розробку – технологію індустріальної розробки програмних систем.

Для кожного класу АС (САПР, ERP, геоінформаційні системи й тощо) можна назвати фірми, що спеціалізуються на розробці програмних (а іноді й програмно-апаратних) систем. Багато на основі однієї з базових технологій реалізують свій підхід до створення АС і дотримуються стратегії або тотального постачальника, або відкритості й розширення системи додатками й доповненнями третіх фірм.

Існують міжнародні стандарти стадій життєвого циклу програмної продукції (ISO 12207:1995). Як власне АС, так і компоненти АС – це складні системи, і для них потрібно використовувати один зі стилів проектування:

• *спадне (Top-of-Design);* чітка реалізація спадного проектування утворює *спіральну модель* розробки ПЗ, на кожному витку спіралі блоки попереднього рівня деталізуються, використовуються зворотні зв’язки (альтернативою є так звана *каскадна модель,* що належить до почергової реалізації частин системи);

• *висхідне (Bottom-of-Design);*

• *еволюційне (Middle-of-Design).*

Найчастіше застосовують спадний стиль блочно-ієрархічного проектування. Розглянемо його етапи.

Верхній рівень проектування АС часто називають *концептуальним*. Концептуальне проектування виконують у процесі передпроектних досліджень, формулювання ТЗ, розробки ескізного проекту й прототипування (відповідно до держстандарту 34.601-90, ці стадії називають формуванням вимог до АС, розробкою концепції АС і ескізний проект).

Передпроектні дослідження проводять шляхом аналізу (обстеження) діяльності підприємства (компанії, установи, офісу), на якому створюється або модернізується АС. При цьому потрібно одержати відповіді на питання: що не влаштовує в існуючій технології? що можна поліпшити? кому це потрібно й, отже, який буде ефект? Перед обстеженням формуються і в процесі його проведення уточнюється мета обстеження: визначення можливостей і ресурсів для підвищення ефективності функціонування підприємства на основі автоматизації процесів керування, проектування, документообігу й ін. Зміст обстеження – виявлення структури підприємства, виконуваних функцій, інформаційних потоків, наявного досвіду й засобів автоматизації. Обстеження проводять системні аналітики (системні інтегратори) разом із представниками організації-замовника.

На основі аналізу результатів обстеження будують модель, що відбиває діяльність підприємства на даний момент (до реорганізації). Таку модель називають *As Is (як є).* Далі розробляють вихідну концепцію АС. Ця концепція містить у собі пропозиції щодо зміни структури підприємства, взаємодії підрозділів, інформаційних потоків, що виражається в моделі *To Be (як має бути)*.

Результати аналізу конкретизуються в ТЗ на створення АС. У ТЗ зазначають потоки вхідної інформації, типи вихідних документів і надаваних послуг, рівень захисту інформації, вимоги до продуктивності (пропускної здатності) тощо. ТЗ направляють замовникові для обговорення й остаточного узгодження.

Ескізний проект (технічну пропозицію) подають у вигляді проектної документації, що описує архітектуру системи, структуру її підсистем, склад модулів. Тут же містяться пропозиції щодо вибору базових програмно-апаратних засобів, які мають ураховувати прогноз розвитку підприємства. Стосовно апаратних засобів, і особливо ПЗ, такий вибір найчастіше являє собою вибір фірми-постачальника необхідних засобів (або, принаймні, базового ПЗ), тому що правильна спільна робота програм різних фірм досягається на превелику силу. У проекті може бути запропоновано кілька варіантів вибору. Під час аналізу з'ясовуються можливості покриття автоматизованих функцій наявними програмними продуктами й, отже, обсяги робіт на створення оригінального ПЗ. Такий аналіз необхідний для попередньої оцінки часових і матеріальних витрат на автоматизацію. Урахування ресурс-

них обмежень дозволяє уточнити досяжні масштаби автоматизації, поділити проектування АС на роботи першої, другої черги тощо.

Після прийняття ескізного проекту розробляють *прототип* АС, що являє собою набір програм, які емулюють роботу готової системи. Завдяки прототипізації не тільки розробники, але й майбутні користувачі АС можуть побачити контури й особливості системи, а отже, завчасно внести корективи в проект.

Як на етапі передпроектних досліджень, так і на наступних етапах доцільно дотримуватися певної дисципліни фіксації й подання одержуваних результатів, що базується на тій чи іншій методиці формалізації специфікацій. Формалізація потрібна для однозначного розуміння виконавцями й замовником вимог, обмежень і прийнятих рішень.

Концептуальне проектування передбачає ряд специфікацій, серед яких центральне місце займають моделі перетворення, зберігання й передачі інформації. Моделі, отримані в процесі обстеження підприємства, є моделями його функціонування. У процесі розробки АС моделі, як правило, зазнають істотних змін (перехід від “As Is” до “To Be”) і в остаточному виді модель “To Be” розглядають як модель проектованої АС.

Розрізнюють функціональні, інформаційні, поведінкові й структурні моделі. *Функціональна* модель системи описує сукупність виконуваних системою функцій. *Інформаційна* модель відбиває структури даних – їх склад і взаємозв’язки. *Поведінкова* модель описує інформаційні процеси (динаміку функціонування), у ній фігурують такі категорії, як стан системи, подія, перехід з одного стану в інший, умови переходу, послідовність подій, здійснюється прив’язка в часі. *Структурна* модель характеризує морфологію системи (її побудову) – склад підсистем, їхній взаємозв’язок.

Змістом наступних етапів спадного проектування (відповідно до ГОСТ 34.601-90 це стадії розробки технічного проекту, робочої документації, запровадження в дію) є уточнення переліків устаткування, що здобувається, і готових програмних продуктів, побудова системного середовища, детальне проектування баз даних і їх первісне наповнення, розробка власного оригінального ПЗ, що, своєю чергою, ділиться на ряд етапів спадного проектування. Ці роботи становлять зміст *робочого проектування.* Після цього йде закупівля й інсталяція програмно-апаратних засобів, впровадження й експериментальна експлуатація системи.

Особливе місце в ряді проектних завдань займає розробка проекту корпоративної обчислювальної мережі, оскільки ТЕ АС має мережну структуру. Якщо територіально АС розташовується в одному будинку або в декількох близько розташованих будинках, то корпоративна мережа може бути виконана у вигляді сукупності декількох локальних підмереж, пов’язаних опорною локальною мережею. Крім вибору типів підмереж, зв’язних протоколів і комутаційного устаткування, доводиться виконувати розподіл вузлів по підмережах, виділення серверів, вибір мережного ПЗ,

визначення способу керування даними в обраній схемі розподілених обчислень тощо.

У випадку якщо АС розташовується у віддалених один від одного пунктах, зокрема розташованих у різних містах, то вирішується питання про оренду каналів зв'язку для корпоративної мережі, оскільки альтернативний варіант використання виділеного каналу в більшості випадків виявляється неприйнятним унаслідок високої ціни. Природно, що при цьому насамперед розглядається можливість використання послуг Internet виконують. При цьому виникають проблеми, пов'язані із забезпеченням інформаційної безпеки та надійності доставки повідомлень.

1.8.2. Відкритість систем

Однією з головних тенденцій сучасної індустрії інформатики є створення *відкритих систем.* Властивість відкритості означає, по-перше, можливість переносити (мобільність) ПЗ на різні апаратні платформи, по-друге, пристосованість системи до її модифікацій (модифікованість або власне відкритість) і комплексуванню з іншими системами з метою розширення її функціональних можливостей і (або) додання системі нових якостей (інтегрованість).

Перехід до відкритих інформаційних систем дозволяє істотно прискорити науково-технічний прогрес у результаті заміни тривалої й дорогої розробки нових систем їхнім компонуванням з раніше спроектованих підсистем або швидкою модернізацією вже існуючих систем (реінжиніринг).

Відкритість має на увазі виділення в системі інтерфейсної частини (входів і виходів), що забезпечує сполучення з іншими системами або підсистемами, причому для комплексування досить мати відомості тільки про інтерфейсні частини об'єктів, що сполучаються. Якщо ж інтерфейсні частини виконані відповідно до заздалегідь застережених правил і угод, яких повинні дотримуватися всі творці відкритих систем певного додатка, то проблема створення нових складних систем істотно спрощується. Із цього випливає, що основою створення відкритих систем є стандартизація й уніфікація в галузі інформаційних технологій.

Концепція відкритості розвинулася в напрямі побудови обчислювальних мереж, що відобразилося в еталонній моделі взаємозв'язку відкритих систем, підтримуваної низкою міжнародних стандартів. Ідеї відкритості широко використовуються для побудови програмного, інформаційного та лінгвістичного забезпечення АС; у результаті підвищується ступінь універсальності програм і розширюються можливості їхньої адаптації до конкретних умов.

Аспекти відкритості закріплено в стандартизації:

• *API (Application Program Interface)* – інтерфейсів прикладних програм з операційним оточенням, у тому числі системних викликів і утиліт операційної системи (ОС), тобто зв'язків з ОС;

• міжпрограмного інтерфейсу, включаючи мови програмування;

• мережної взаємодії;

• користувацького інтерфейсу, в тому числі засобів графічної взаємодії користувача з ЕОМ;

• засобів захисту інформації.

Стандарти, що забезпечують відкритість ПЗ, у цей час розробляються такими організаціями, як ISO (International Standard Organization), IEEE (Institute of Electrical and Electronics Engineers), EIA (Electronics Industries Association) тощо. Стандарти POSIX (Portable Operating System Interface) призначені для API і становлять групу стандартів IEEE 1003. У цих стандартах містяться перелік і правила виклику інтерфейсних функцій, визначаються способи взаємодії прикладних програм з ядром ОС мовою С (що означає переважну орієнтацію на ОС Unix), дані розширення для взаємодії із програмами на інших мовах, способи тестування інтерфейсів на відповідність стандартам POSIX, правила адміністративного керування програмами й даними тощо. Ряд стандартів ISO присвячений мовам програмування. Є стандарти на мові С (ISO 9899), Fortran (ISO 1539), Pascal (ISO 7185) і ін. Серед інших стандартів, що сприяють відкритості ПЗ АС, слід зазначити стандарти графічного користувацького інтерфейсу, зберігання й передачі графічних даних, побудови баз даних і файлових систем, супроводу й керування конфігурацією програмних систем та ін.

Важливе значення для створення відкритих систем мають уніфікація й стандартизація засобів міжпрограмного інтерфейсу, або, інакше кажучи, необхідна наявність профілів АС для інформаційної взаємодії програм, що входять в АС. *Профілем* відкритої системи називають сукупність стандартів та інших нормативних документів, що забезпечують виконання системою заданих функцій.

Так, у профілях АС можуть фігурувати мова Express стандарту STEP, специфікація графічного користувальницького інтерфейсу Motif, уніфікована мова SQL обміну даними між різними СКБД, стандарти мережної взаємодії, у профілі MCAD може входити формат IGES і у випадку ECAD – формат EDIFi та ін.

Запитання і завдання для самоконтролю

1. Дайте визначення поняття “проектування”.
2. Назвіть ознаки, властиві складній системі.
3. Які особливості характерні для всіх підходів до проектування складних систем?
4. Наведіть приклади умов працездатності.
5. Чому проектування зазвичай має ітераційний характер?
6. Назвіть основні стадії проектування технічних систем.
7. Назвіть основні типи АС і види їхнього забезпечення.
8. Що розуміють під комплексною АС?
9. Дайте визначення профілю відкритої системи.
10. Чим забезпечується відкритість систем?

РОЗДІЛ 2. СТРУКТУРНА (SADT) ТА ОБ'ЄКТНО-ОРІЄНТОВАНА ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ

2.1. Структурний аналіз предметної області (технологія IDEF0)

Опис системи за допомогою IDEF0 називається функціональною моделлю. Функціональна модель призначена для опису існуючих бізнес-процесів, в якому використовують як природну, так і графічну мови. Для передачі інформації про конкретну систему джерелом графічної мови є сама методологія IDEF0.

Методологія IDEF0 пропонує побудову ієрархічної системи діаграм – одиничних описів фрагментів системи. Спочатку проводиться опис системи в цілому і її взаємодії з навколишнім світом (контекстна діаграма), після чого виконується функціональна декомпозиція – система розбивається на підсистеми й кожна підсистема описується окремо (діаграми декомпозиції). Потім кожна підсистема розбивається на дрібніші до досягнення потрібного ступеня деталізації.

Кожна IDEF0-діаграма містить блоки й дуги. Блоки зображують функції системи, що моделюється. Дуги зв'язують блоки разом і відображають взаємодії та взаємозв'язки між ними.

Функціональні блоки (роботи) на діаграмах зображуються прямокутниками, що означають пойменовані процеси, функції або завдання, які відбуваються протягом певного часу і мають розпізнавані результати. Ім'я роботи потрібно позначати віддієслівним іменником, що називає дію.

IDEF0 вимагає, щоб у діаграмі було не менш трьох і не більше шести блоків. Ці обмеження підтримують складність діаграм і моделі на рівні, доступному для читання, розуміння й використання.

Кожна сторона блоку має особливе, цілком певне призначення. Ліва сторона блоку призначена для входів, верхня – для керування, права – для виходів, нижня – для механізмів. Таке позначення відбиває певні системні принципи: входи перетворюються на виходи, керування обмежує або пропонує умови виконання перетворень, механізми показують, що і як виконує функція.

Блоки в IDEF0 розміщуються за ступенем важливості, як і розуміє автор діаграми. Цей відносний порядок називається домінуванням. Домінування розуміється як вплив одного блоку на інші блоки діаграми. Наприклад, головним домінантним блоком діаграми може бути або перший з необхідної послідовності функцій, або планувальна чи контрольна функція, що впливає на всі інші. Головний домінантний блок зазвичай розміщується у верхньому лівому куті діаграми, а найменш домінантний – у правому куті. Розташування блоків на сторінці відбиває авторське визначення домінування. Таким чином, топологія діаграми показує, які функції впливають на інші. Щоб підкреслити це, аналітик може перенумерувати блоки відповідно до порядку їхнього домінування. Порядок домінування може позначатися цифрою, розміщеною в правому нижньому куті кожного пря-

мокутника: цифра 1 означатиме найбільше домінування, 2 – наступне у напрямі зменшення і т. д.

Зв'язки робіт із зовнішнім світом і між собою описуються у вигляді одинарних ліній зі стрілками на кінцях. Стрілки являють собою якусь інформацію й називаються (підписуються) іменниками.

В IDEF0 розрізнюють п'ять типів стрілок.

Вхід – об'єкти, використовувані й перестворені роботою для одержання результату (виходу). Допускається, що робота може не мати жодної стрілки входу. Стрілка входу зображується як вхідна в ліву грань роботи.

Керування – інформація, що управляє діями роботи. Зазвичай керівні стрілки несуть інформацію, що повинна виконувати робота. Кожна робота повинна мати хоча б одну стрілку керування, що зображується як вхідна у верхню грань роботи.

Вихід – об'єкти, в які перетворяться входи. Кожна робота повинна мати хоча б одну стрілку виходу із правої грані роботи.

Механізм – ресурси, що виконують роботу. Стрілка механізму входить у нижню грань роботи. На розсуд аналітика стрілки, механізму можуть не зображуватися на моделі.

Виклик – спеціальна стрілка, що вказує на іншу модель роботи. Стрілка виклику виходить з нижньої частини роботи і показує, що певна робота виконується за межами системи, яку моделюють.

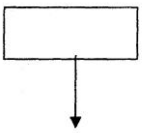


Рис. 2.1. Стрілка виклику

У методології IDEF0 використовуються тільки п'ять типів взаємодій між блоками для опису їхніх відношень: керування, вхід, зворотний зв'язок щодо керування, зворотний зв'язок щодо входу, вихід-механізм. Зв'язки керування і входу найпростіші, оскільки вони відбивають прямі впливи, які інтуїтивно зрозумілі й дуже прості.

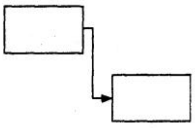


Рис. 2.2. Зв'язок виходу

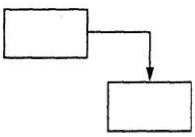


Рис. 2.3. Зв'язок керування

Відношення керування виникає тоді, коли вихід одного блоку безпосередньо впливає на блок з меншим домінуванням.

Зворотні зв'язки керування і входу складніші, оскільки являють собою ітерацію або рекурсію: виходи з однієї роботи впливають на майбутнє виконання інших робіт, що згодом вплине на вихідну роботу.

Зворотний зв'язок керування виникає тоді, коли вихід деякого блоку впливає на блок з більшим домінуванням. Зв'язки "вихід-механізм" зустрічаються нечасто. Вони відбивають ситуацію, коли вихід однієї функції стає засобом досягнення мети для іншої. Зв'язки "вихід-механізм" характерні розподілу джерел ресурсів (наприклад, необхідні інструменти, навчений персонал, фізичний простір, устаткування, фінансування, матеріали).

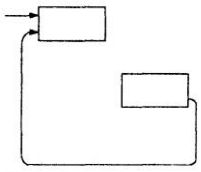


Рис. 2.4. Зворотний зв'язок входу

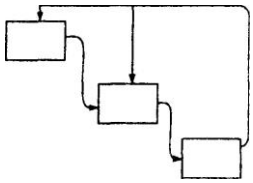


Рис. 2.5. Зворотний зв'язок керування

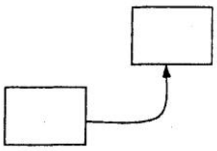


Рис. 2.6. Зв'язок "вихід-механізм"

В IDEF0 дуга рідко зображує один об'єкт, зазвичай вона символізує набір об'єктів. Оскільки дуги являють набори об'єктів, вони можуть мати множину початкових точок (джерел) і кінцевих точок (призначень). Тому дуги можуть розгалужуватися і з'єднуватися різними способами. Вся дуга або її частина може виходити з одного або декількох блоків і закінчуватися в одному або декількох блоках.

Розгалуження дуг, зображуване у вигляді розбіжних ліній, означає, що весь уміст дуг або його частина може з'явитися в кожному відгалуженні. Дуга завжди позначається до розгалуження, щоб дати назву всьому набору. Крім того, кожна гілка дуги може бути позначена або не позначена відповідно до таких правил:

• не помічені галузі містять всі об'єкти, зазначені в мітці дуги перед розгалуженням;

• відгалуження, позначені після точки розгалуження, містять усі об'єкти або їх частину, зазначені в мітці дуги перед розгалуженням.

Злиття дуг в IDEF0, зображуване як збіжні разом лінії, свідчить, що вміст кожної галузі йде на формування мітки для дуги, яка є результатом злиття вихідних дуг. Після злиття результуюча дуга завжди позначає новий набір об'єктів, що виник після об'єднання. Крім того, кожна галузь перед злиттям може позначатися або не позначатися відповідно до таких правил:

• не помічені лінії відгалуження містять усі об'єкти, зазначені в загальній мітці дуги після злиття;

• позначені перед злиттям ліній містять усі або деякі об'єкти з перерахованих у загальній мітці після злиття.

Для проведення кількісного аналізу діаграм перелічимо показники моделі:

• кількість блоків на діаграмі – *N;*

• рівень декомпозиції діаграми – *L;*

• збалансованість діаграми – *B;*

• кількість стрілок, що з'єднуються з блоком, – *A*.

Такий набір факторів стосується кожної діаграми моделі. Далі будуть перераховані рекомендації з бажаних значень факторів діаграми.

Необхідно прагнути до того, щоб кількість блоків на діаграмах нижніх рівнів була нижчою за кількість блоків на батьківських діаграмах, тобто зі збільшенням рівня декомпозиції зменшувався й коефіцієнт *N/L*. Таким чином, зменшення цього коефіцієнта свідчить про те, що в міру декомпозиції моделі функції повинні спрощуватися, отже, кількість блоків має зменшуватися.

Діаграми мають бути збалансовані. Це означає, що в рамках однієї діаграми не повинна створюватися ситуація, зображена на рис. 2.7: у роботі 1 вхідних стрілок і стрілок керування значно більше, ніж вихідних. Слід зазначити, що дана рекомендація може не виконуватися в моделях, що описують виробничі процеси. Наприклад, під час опису процедури складання деталей у блок може входити багато стрілок, що описують компоненти виробу, а виходитиме одна стрілка – готовий виріб.

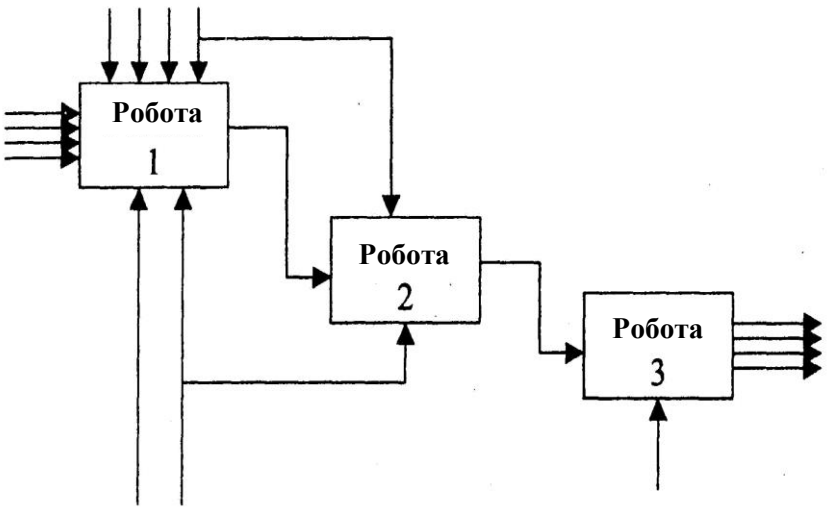


Рис. 2.7. Приклад незбалансованої діаграми

Уведемо коефіцієнт збалансованості діаграми:

$$Kb = \left| \frac{\sum_{i=1}^{N} Ai}{N} - \max_{i=1}^{N}(Ai) \right|.$$

Слід прагнути, щоб K_b був мінімальний для діаграми.

Крім аналізу графічних елементів діаграми, потрібно розглядати найменування блоків. Для оцінки імен складається словник елементарних (тривіальних) функцій системи, що моделюється. До цього словника повинні потрапити функції нижнього рівня декомпозиції діаграм. Наприклад, для моделі БД елементарними можуть бути функції "знайти запис", "додати запис у БД", водночас функція "реєстрація користувача" потребує подальшого опису.

Після формування словника і складання пакета діаграм системи потрібно розглянути нижній рівень моделі. Якщо на ньому виявляться збіги назв блоків діаграм і слів зі словника, то це свідчить, що достатній рівень декомпозиції досягнутий. Коефіцієнт, що кількісно відбиває даний критерій, можна записати як $L \times C$ – добуток рівня моделі на число збігів імен блоків зі словами зі словника. Чим нижчий рівень моделі (більше *L),* тим цінніші збіги.

Запитання і завдання для самоконтролю

1. Що являє собою модель у нотації IDEF0?
2. Що позначають роботи в IDEF0?
3. Назвіть порядок найменування робіт.
4. Яка кількість робіт повинна бути на одній діаграмі?
5. Що називається порядком домінування?
6. Як розташовуються роботи за принципом домінування?
7. Яке призначення сторін прямокутників робіт на діаграмах?
8. Перелічіть типи стрілок.
9. Назвіть види взаємозв'язків.
10. Що означають граничні стрілки?
11. Поясніть принцип іменування стрілок, що розгалужуються й зливаються.

2.2. Моделювання потоків робіт. Доповнення моделей процесів діаграмами DFD і WorkFlow (IDEF3)

2.2.1. Діаграми потоків даних (Data Flow Diagrams)

Ці діаграми являють собою мережу пов'язаних між собою робіт. Їх зручно використовувати для опису документообігу й обробки інформації.

DFD описує:

1) функції обробки інформації (роботи);

2) документи (стрілки, arrow), об’єкти, співробітників або відділи, які беруть участь в обробці інформації;

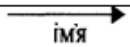
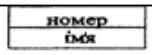
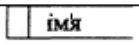
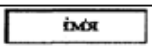
3) зовнішні посилання (external reference), які забезпечують інтерфейс із зовнішніми об’єктами, що перебувають за межами модельованої системи;

4) таблиці для зберігання документів (сховища даних, data store).

Для побудови діаграм DFD в BPWin використається нотація Гейна–Сарсона.

Таблиця 2.1

Нотація Гейна–Сарсона

Компонент	Позначення
Потік даних	
Процес	
Сховище	
Зовнішня сутність	

Потоки даних є механізмами, що використовуються для моделювання передачі інформації (або фізичних компонентів) з однієї частини системи в іншу. Потоки зображуються на діаграмі іменованими стрілками, орієнтація яких показує напрямок руху інформації. Стрілки можуть підходити до будь-якої грані прямокутника роботи й можуть бути двонаправленими для опису взаємодії типу “команда – відповідь”.

Призначення процесу полягає в продукуванні вихідних потоків із вхідних відповідно до дії, що задає ім’я процесу. Кожен процес повинен мати унікальний номер для посилань на нього всередині діаграми. Цей номер може використатися разом з номером діаграми для одержання унікального індексу процесу у всій моделі.

Сховище даних дозволяє на певних ділянках визначати дані, які зберігатимуться в пам’яті між процесами. Фактично сховище являє собою “зрізи” потоків даних у часі. Інформація зі сховища може використовуватися в будь-який час після її визначення, при цьому дані можуть вибиратися в будь-якому порядку. Ім’я сховища має ідентифікувати його вміст. У випадку, коли потік даних входить у сховище або виходить із нього і його структура відповідає структурі сховища, він повинен мати те саме ім’я, тому немає необхідності зазначати це ім’я на діаграмі.

Зовнішня сутність (рис. 2.8) являє собою сутність поза контекстом системи, що є джерелом або приймачем даних системи. Передбачається, що об’єкти, відображені такими вузлами, не повинні брати участь у жодній обробці. Зовнішні сутності зображуються у вигляді прямокутника з тінню і

зазвичай розташовуються по краях діаграми. Одна зовнішня сутність може бути використана багаторазово на одній або декількох діаграмах.

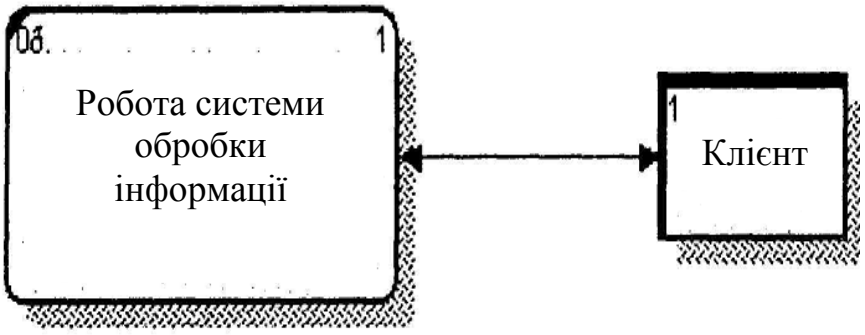


Рис. 2.8. Зовнішня сутність

2.2.2. Діаграми IDEF3

Діаграми IDEF3 також називають WorkFlow diagramming – методологією моделювання, що використовує графічний опис інформаційних потоків, відношень між процесами обробки інформації та об'єктів, що є частиною цих процесів. Діаграми WorkFlow використовуються для аналізу процедур обробки інформації.

Мета IDEF3 – дати аналітикам опис послідовності виконання процесів, а також об'єктів, що беруть участь спільно в одному процесі. IDEF3 може бути також використаний як метод створення процесів. IDEF3 доповнює IDEF0 і містить усе необхідне для побудови моделей, які можуть бути використані для імітаційного моделювання.

Діаграма є основною одиницею опису в IDEF3-моделі. Організація діаграм в IDEF3 має найбільше значення, якщо модель редагується кількома людьми. У цьому випадку розробник повинен визначати, яка інформація входитиме в ту чи іншу модель.

Одиниці роботи – Unit of Work (UOW), або просто роботи, є центральними компонентами моделі. В IDEF3 роботи зображуються прямокутниками і мають ім'я, що позначає процес дії та номер (ідентифікатор). В ім'я зазвичай включається основний результат роботи (наприклад, готування обіду).

Показують взаємозв'язки робіт. Усі зв'язки в IDEF3 односпрямовані.

Старша (Precedence) лінія − ⟶ суцільна лінія ( ), що зв'язує одиниці робіт. Рисується зліва направо або зверху вниз. Показує, що робота-джерело повинна закінчитися, перш ніж робота – мета почнеться.

Лінія відношень (Relation Link) – - - - - -→ пунктирна лінія ( ), що використовується для зображення зв'язків між одиницями робіт, а також між одиницями робіт та об'єктами посилань.

Потоки об'єктів (Object Flow) – ⟶→ стрілка з двома наконечниками ( ), застосовується для опису використання об'єкта у двох або більше одиницях роботи, наприклад коли об'єкт породжується в одній роботі, а використовується в іншій.

Перехрестя (Junction) – використовуються для відображення логіки взаємодії стрілок під час злиття і розгалуження або для відображення множини подій, які можуть або повинні бути завершені перед початком подальшої роботи. Розрізнюють перехрестя для злиття (Fan-in Junction) і розгалуження (Fanout Junction) стрілок. Перехрестя не може використовуватися одночасно для злиття й для розгалуження. Для внесення перехрестя служить позначення  .

Таблиця 2.2

Типи перехрещень

Позначення	Найменування	Зміст у випадку злиття стрілок	Зміст у випадку розгалуження стрілок
	Asynchronous AND	Всі попередні процеси повинні бути завершені	Всі наступні процеси повинні бути запущені
	Synchronous AND	Всі попередні процеси завершені одночасно	Всі наступні процеси запускаються одночасно
	Asynchronous OR	Один або кілька попередніх процесів повинні бути завершені	Один або кілька наступних процесів повинні бути запущені
	Synchronous OR	Один або кілька попередніх процесів завершені одночасно	Один або кілька наступних процесів запускаються одночасно
	XOR (Exclusive OR)	Тільки один процес завершений	Тільки один наступний процес запускається

Об’єкти-посилання є спеціальними символами, які посилаються на зовнішні частини опису процесу. Вони подаються на діаграму для того, щоб звернути увагу редактора на важливе явище, яке неможливо пов’язати зі стрілкою, роботою або перехрестям.

Для внесення об’єкта-посилання служить кнопка  . Об’єкт посилання відображається у вигляді прямокутника. Об’єкт-посилання повинні бути пов’язані з одиницями робіт або перехрестями пунктирними лініями. У разі внесення об’єктів-посилань потрібно зазначити їхній тип.

Таблиця 2.3

Типи об’єктів-посилань

Тип об’єкта-посилання	Мета опису
OBJECT	Описує участь важливого об’єкта в роботі
GOTO	Інструмент циклічного переходу (у повторюваній послідовності робіт) можливий на поточній діаграмі, але не обов’язковий. Якщо всі роботи циклу є на поточній діаграмі, цикл може також зображуватися стрілкою, що повертається на стартову роботу GOTO може посилатися на перехрестя
UOB (Unit of behavior)	Застосовується, коли необхідно підкреслити множинне використання якої-небудь роботи, але без циклу. Наприклад, робота “Контроль якості” може бути використана в процесі “Виготовлення виробу” кілька разів, після кожної одиничної операції. Зазвичай цей тип посилання не використовується для моделювання робіт, що запускаються автоматично
NOTE	Використовується для документування важливої інформації, що стосується до яких-небудь графічних об’єктів на діаграмі. NOTE є альтернативою внесенню текстового об’єкта в діаграму
ELAB (Elaboration)	Використовується для вдосконалення графіків або їх детальнішого опису. Зазвичай вживається для детального опису розгалуження і злиття стрілок на перехрестях

Запитання і завдання для самоконтролю

1. Що описує діаграма DFD?
2. Яка нотація використається в BPWin для побудови діаграм DFD?
3. Що описує діаграма IDEF3?
4. Перелічіть складові частини діаграми DFD.
5. У чому полягає призначення процесу?
6. Що називається зовнішньою сутністю?
7. Що описують сховища?
8. Поясніть механізм доповнення діаграми IDEF0 діаграмою DFD.
9. Перелічіть складові елементи діаграм IDEF3.
10. Що показують зв’язки в діаграмах IDEF3?
11. Перелічіть типи стрілок у діаграмах IDEF3.
12. Що називається перехрестям?
13. Назвіть типи перехресть.

2.3. Технологія моделювання даних IDEF1X

Методологія IDEF1X поділяється на рівні, що відповідають проектованій моделі даних системи. Кожен рівень відповідає певній фазі проекту. Такий підхід корисний для створення систем за принципом “зверху вниз”.

Верхній рівень складається з Entity Relation Diagram (діаграма сутність – зв’язок) і Key-Based model (модель даних, що грунтується на ключах). Діаграма сутність – зв’язок визначає сутності та їхні відношення. Модель даних, що ґрунтується на ключах, дає більш докладне відображення даних. Вона включає опис всіх сутностей і первинних ключів, які відповідають предметній області.

Нижній рівень складається з Transformation Model (трансформаційна модель) і Fully Attributed (повна атрибутивна модель). Трансформаційна модель містить усю інформацію для реалізації проекту, що може бути частиною загальної інформаційної системи та описувати предметну область.

Трансформаційна модель дозволяє проектувальникам та адміністраторам БД показувати, які об’єкти БД зберігаються у словнику даних, і перевірити, наскільки фізична модель даних задовольняє вимоги інформаційної системи. Із трансформаційної моделі можна автоматично можна одержати модель СУБД, що є точним відображенням системного каталога СУБД.

2.3.1. Логічні моделі

Три рівні моделей, що поєднують у собі логічні моделі, складаються з Entity Relationship Diagram (діаграма сутність – зв’язок), the Key-Based (модель даних, що грунтуються на ключах) Model і the Fully Attributed model (повна атрибутивна модель).

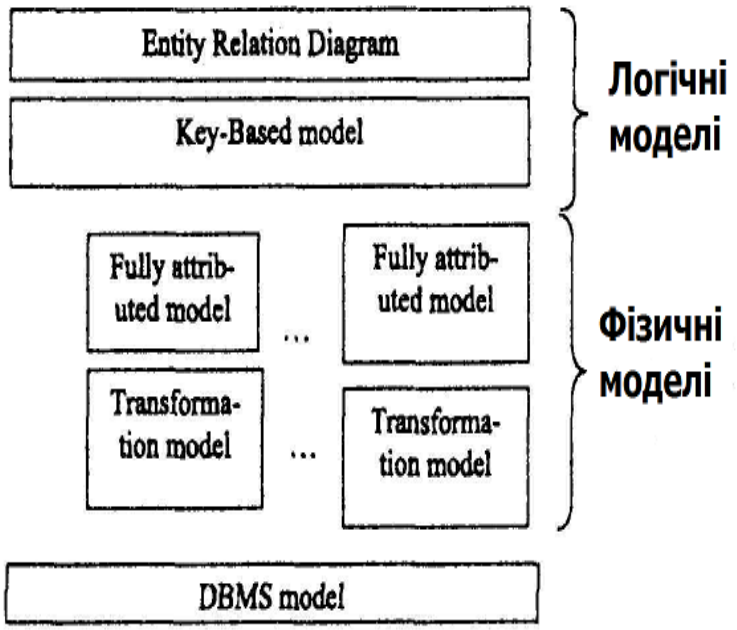


Рис. 2.9. Рівні методології IDEF1X

Діаграма сутність – зв’язок є найвищим рівнем у моделі даних і визначає набір сутностей та атрибутів проектованої системи. Метою цієї діаграми є формування загального погляду на систему для її подальшої деталізації.

Модель даних, що ґрунтується на ключах моделі описує структуру дані системи, в яку включено всі сутності й атрибути, в тому числі ключові. Метою цієї моделі є деталізація моделі сутність – зв’язок, після чого модель даних можна почати реалізовувати.

Повна атрибутивна модель містить у собі всі сутності, атрибути і є найбільш детальним відображенням структури даних. Повна атрибутивна модель подає дані в третій нормальній формі.

2.3.2. Створення логічної моделі

Першим кроком у створенні логічної моделі БД є побудова діаграми ERD (Entity Relationship Diagram). ERD-діаграми складаються з трьох частин: сутностей, атрибутів і взаємозв’язків. Сутності – це іменники, атрибути – прикметники або модифікатори, взаємозв’язки – дієслова.

ERD-діаграма дозволяє розглянути систему цілком і з’ясувати вимоги, необхідні для її розробки, що стосуються зберігання інформації. ERD-діаграми можна поділити на окремі шматки, що відповідають окремим завданням, які розв’язуватиме проектована система. Це дозволяє розглядати систему з погляду функціональних можливостей, роблячи процес проектування керованим.

Як відомо, основним компонентом реляційних БД є таблиця. Таблиця використовується для структуризації та зберігання інформації. У реляційних БД кожна клітинка таблиці містить одне значення. Крім того, всередині однієї БД існують взаємозв’язки між таблицями, кожний з яких задає спільне користування даними таблиці.

ERD-діаграма графічно зображує структуру даних проектованої інформаційної системи. Сутності відображаються за допомогою прямокутників, що містять ім’я. Імена прийнято виражати іменниками в однині, взаємозв’язки – за допомогою ліній, що з’єднують окремі сутності.

Взаємозв’язок показує, що дані однієї сутності пов’язані з даними іншої сутності або посилаються на неї.

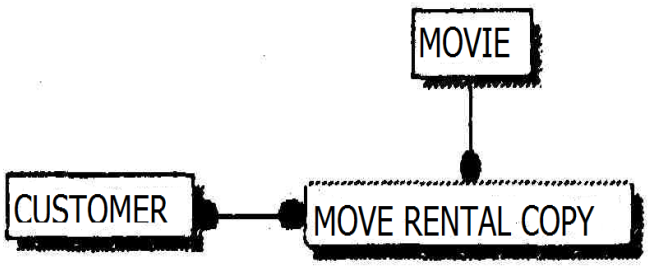


Рис. 2.10. Приклад ERD-діаграми

Сутність – це суб’єкт, місце, річ, подія або поняття, що містять інформацію. Точніше, сутність – це набір (об’єднання) об’єктів, що називаються екземплярами. У наведеному на рис. 6.1 прикладі сутність CUSTOMER (клієнт) представляє всіх можливих клієнтів. Кожен екземпляр сутності має набір характеристик. Так, кожен клієнт може мати ім’я, адресу, телефон тощо, у логічній моделі всі ці характеристики називаються атрибутами сутності.

На рис. 2.11 показано ERD-діаграму, що включає в себе атрибути сутностей.

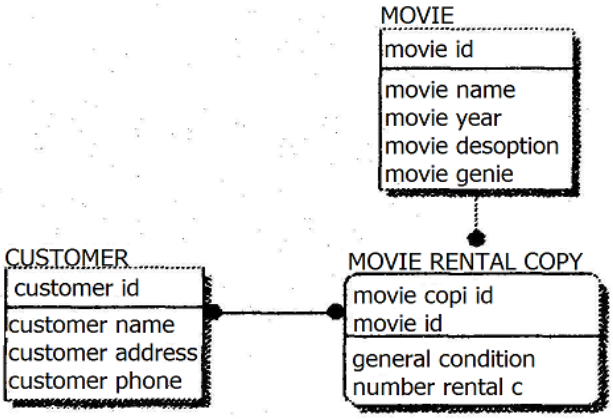


Рис. 2.11. ERD-діаграма з атрибутами

Логічні взаємозв’язки являють собою зв’язки між сутностями. Вони визначаються дієсловами, що показують, як одна сутність пов’язана з іншою.

Деякі приклади взаємозв’язків:

• команда включає багато гравців;

• літак перевозить багато пасажирів;

• продавець продає багато продуктів.

У всіх цих випадках взаємозв’язки відбивають взаємодію між двома сутностями, це взаємодія “один до багатьох”, яка означає, що один екземпляр першої сутності взаємодіє з декількома екземплярами іншої сутності. Взаємозв’язки відображаються лініями, що з’єднують дві сутності з точкою на одному кінці й дієсловом, розташованим над лінією.

Крім взаємозв’язку “один до багатьох”, існує ще один тип – “багато до багатьох”. Цей тип зв’язку описує ситуацію, коли екземпляри сутностей можуть взаємодіяти з декількома екземплярами інших сутностей. Зв’язок “багато до багатьох” використовують на початкових стадіях проектування. Цей тип взаємозв’язку відображається суцільною лінією з точками на обох кінцях. Зв’язок “багато до багатьох” може не враховувати певні обмеження системи, тому його можна замінити на “один до багатьох” під час наступного перегляду проекту.

Якщо взаємозв’язки між сутностями встановлено правильно, то можна скласти речення, що їх описують. Наприклад, за моделлю з рис. 2.12 можна скласти такі речення.

- Літак перевозить пасажирів.
- Багато пасажирів перевозяться одним літаком.

Складання таких речень дозволяє перевірити відповідність отриманої моделі вимогам та обмеженням створюваної системи.

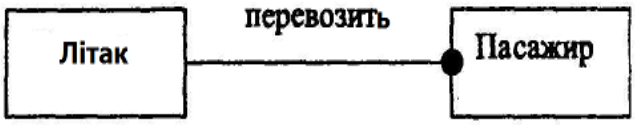


Рис. 2.12. Приклад логічної моделі із взаємозв'язком

Кожна сутність містить горизонтальну лінію, що поділяє атрибути на дві групи. Атрибути, розташовані над лінією, називаються первинним ключем. Первинний ключ призначений для унікальної ідентифікації екземпляра сутності.

Створюючи сутність, потрібно виділити групу атрибутів, які потенційно можуть стати первинним ключем (потенційні ключі), потім зробити відбір атрибутів для включення до складу первинного ключа, дотримуючись таких рекомендацій.

• Первинний ключ повинен бути підібраний таким чином, щоб за значеннями атрибутів, у нього включених, можна було точно ідентифікувати екземпляр сутності.

• Ніякий з атрибутів первинного ключа не повинен мати нульового значення.

• Значення атрибутів первинного ключа не повинні мінятися. Якщо значення змінилося, виходить, це вже інший екземпляр сутності.

Вибираючи первинний ключ, можна внести в сутність додатковий атрибут і зробити його ключем. Так, для визначення первинного ключа часто використовують унікальні номери, які можуть автоматично генеруватися системою під час додавання екземпляра сутності в БД. Застосування унікальних номерів полегшує процес індексації та пошуку в БД.

Обраний під час створення логічної моделі первинний ключ може бути невдалим для ефективного доступу до БД, його слід змінити під час проектування фізичної моделі.

Потенційний ключ, що не став первинним, називається альтернативним ключем (Alternate Key). ERWin дозволяє виділити атрибути альтернативних ключів, і надалі за замовчуванням під час генерації схеми БД за цими атрибутами генеруватиметься унікальний індекс. У разі створення альтернативного ключа на діаграмі поруч з атрибутом з'являються символи (АК).

Атрибути, що беруть участь в неунікальних індексах, називаються інверсійними входами (Inversion Entries). Інверсійні входи – це атрибут або група атрибутів, які не визначають екземпляр унікальним чином, але часто використовуються для звертання до екземплярів сутності. ERWin генерує неунікальний індекс для кожного інверсійного входу.

Під час проведення зв'язку між двома сутностями в дочірній сутності автоматично утворяться зовнішні ключі (foreign key). Зв'язок утворить по-

силання на атрибути первинного ключа в дочірній сутності, і ці атрибути утворять зовнішній ключ у дочірній сутності. Атрибути зовнішнього ключа позначаються символами (FK) після свого імені.

2.3.3. Фізичні моделі

Існує два рівні фізичних моделей: трансформаційна модель і модель СУБД. Фізичні моделі містять інформацію, потрібну системним розробникам для розуміння механізму реалізації логічної моделі в СУБД.

Метою трансформаційної моделі є надання інформації адміністраторові БД для створення ефективної структури зберігання, що включає в себе записи, які формують БД. Трансформаційна модель повинна допомогти розробникам вибрати структуру зберігання даних і реалізувати систему доступу до них.

Перед початком проектування БД слід переконатися в забезпеченні таких вимог:

• фізична модель даних повинна відповідати вимогам, пропонованим до проектованої системи;

• вибір певної фізичної моделі повинен бути аргументований;

• повинні бути визначені можливості нарощування існуючої структури зберігання, а також виявлені її обмеження.

2.3.4. Нормалізація. Створення фізичної моделі

Нормалізація – процес перевірки та реорганізації сутностей і атрибутів з метою задоволення вимог до реляційної моделі даних. Нормалізація дозволяє бути впевненим, що кожен атрибут, визначений для своєї сутності, значно скоротить обсяг пам’яті для зберігання даних.

Для розгляду видів нормальних форм уведемо поняття функціональної і повної функціональної залежності.

Функціональна залежність. Атрибут B сутності E функціонально залежить від атрибута A сутності E, якщо і тільки якщо кожне значення A в E пов’язало з ним точно одне значення B в E. Іншими словами, A однозначно визначає B.

Повна функціональна залежність. Атрибут E сутності B повністю функціонально залежить від ряду атрибутів A сутності E, якщо і тільки якщо B функціонально залежить від A і не залежить ні від якого підряду A.

Існують такі види нормальних форм.

- Перша нормальна форма (1NF). Сутність E перебуває в першій нормальній формі, якщо і тільки якщо всі атрибути містять тільки атомарні значення. Серед атрибутів не повинно зустрічатися груп, що повторюються, тобто декількох значень для кожного екземпляра.
- Друга нормальна форма. Сутність E перебуває в другій нормальній формі, якщо вона міститься в першій нормальній формі і кожен неключовий атрибут повністю залежить від первинного ключа, тобто не існує залежностей від частини ключа.

• Третя нормальна форма (3 NF). Сутність Е перебуває в третій нормальній формі, якщо вона міститься в другій нормальній формі і неключові атрибути сутності Е залежать від інших атрибутів Е.

Після третьої нормальної форми існують нормальна форма Бойсса–Кодда (четверта і п'ята нормальні форми). На практиці обмежуються зведенням до третьої нормальної форми. Часто після проведення нормалізації усі взаємозв'язки даних визначаються правильно, модель даних легше підтримується. Проте нормалізація не веде до підвищення продуктивності системи в цілому, тому у створенні фізичної моделі з метою підвищення продуктивності доводиться свідомо відходити від нормальних форм, щоб використати можливості конкретного серверу. Такий процес називається денормалізацією.

Метою створення фізичної моделі є забезпечення адміністратора відповідною інформацією для перенесення логічної моделі даних у СУБД. При цьому логічна модель трансформується у фізичну за принципом: сутності стають таблицями, атрибути стають стовпцями, а ключі стають індексами.

Таблиця 2.4

Зіставлення компонентів логічної і фізичної моделей

Логічна модель	Фізична модель
Сутність	Таблиця
Атрибут	Стовпець
Логічний тип (текст, число, дата, blob)	Фізичний тип (коректний тип, що залежить від обраної СКБД)
Первинний ключ	Первинний ключ, індекс PK
Зовнішній ключ	Зовнішній ключ, індекс FK
Альтернативний ключ	AK-індекс – унікальний, непервинний індекс
Правило бізнес-логіки	Тригер або збережена процедура
Взаємозв'язки	Взаємозв'язки, що визначаються використанням FK-атрибутів

2.3.5. Денормалізація

Після нормалізації всі взаємозв'язки даних стають визначеними, виключаючи помилки в оперуванні даними. Але нормалізація даних знижує швидкодію БД. Для більш ефективної роботи з даними, використовуючи можливості конкретного сервера БД, доводиться виконувати процес, протилежний нормалізації, – денормалізацію.

Для процесу денормалізації не існує стандартного алгоритму, тому в кожному конкретному випадку доводиться шукати своє рішення. Денормалізація зазвичай проводиться на фізичному рівні моделі.

Запитання і завдання для самоконтролю

1. Назвіть рівні методології IDEF1X.
2. З яких моделей складається логічний рівень?
3. З яких моделей складається фізичний рівень?
4. Що містить діаграма сутність – зв'язок?
5. Що містить модель даних, що грунтується на ключах?
6. Яку інформацію містить трансформаційна модель?
7. Що містить у собі повна атрибутивна модель?
8. Сформулюйте вимоги, яких потрібно дотримуватися перед початком проектування БД.
9. Що називається моделлю СУБД?
10. Назвіть основні частини ERD-діаграми.
11. Мета ERD-діаграми.
12. Що є основним компонентом реляційних БД?
13. Що називається сутністю?
14. Сформулюйте принцип іменування сутностей.
15. Що показує взаємозв'язок між сутностями?
16. Назвіть типи логічних взаємозв'язків.
17. Яким чином відображаються логічні взаємозв'язки?
18. Опишіть механізм перевірки адекватності логічної моделі.
19. Що називається первинним ключем?
20. Назвіть принципи, згідно з якими формується первинний ключ.
21. Що називається альтернативним ключем?
22. У якому випадку утворюються зовнішні ключі?
23. Що називається процесом нормалізації?
24. Що називається функціональною залежністю?
25. Що називається повною функціональною залежністю?
26. Перша нормальна форма.
27. Друга нормальна форма.
28. Третя нормальна форма.
29. Нормальна форма Бойсса–Кодда.
30. Що називається процесом денормалізації?
31. У чому сенс денормалізації?
32. Яка мета створення фізичної моделі?

2.4. Об'єктно-орієнтована технологія. Уніфікована мова моделювання UML: призначення та основні компоненти UML

2.4.1. Застосування об'єктно-орієнтованого підходу в UML

Уніфікована мова моделювання (далі UML) – універсальна мова, що дозволяє одночасно з аналізом створювати документацію для проектування складних ієрархічних систем, щоб потім втілювати її в працездатний код на кожній з мов програмування. Автори мови приклали багато зусиль, щоб UML чітко описувала процес створення програмного забезпечення.

UML вважається мовою візуального моделювання. Вона призначена для спілкування розробників у ході роботи над одним проектом і для однакового опису різних проектів. UML об'єктно-орієнтована, але водночас ніяк не пов'язана з конкретними об'єктно-орієнтованими мовами програмування. Розроблений у термінах UML проект можна легко втілити на будь-якій існуючій мові, що підтримує об'єктно-орієнтовану технологію. UML не наполягає також і на конкретній технології реалізації готового проекту і в цьому розумінні відіграє роль універсальної мови.

Використовуючи UML, можна змістовно описувати класи, об'єкти і компоненти, що належать до різних предметних сфер, які дуже відрізняються.

UML реалізує об'єктно-орієнтований підхід до розробки складних систем такими засобами:

- програмна система (далі – система) подається у вигляді множини самостійних *сутностей,* що взаємодіють одна з одною. Кожна сутність сама відповідає за збереження інформації, необхідної для її функціонування, і, крім того, має (реалізує) свою власну поведінку. З кожною сутністю пов'язане поняття класу й об'єкта;
- *клас* – це група сутностей (об'єктів), що мають подібні властивості, а саме дані й поведінку;
- кожен об'єкт захищений системою правил, що не дозволяють навколишнім об'єктам довільно змінювати його дані або впливати на його поведінку. Дані правила визначають спосіб взаємодії з оточенням (інтерфейс) і ховають деталі реалізації, іншими словами – дані й методи *інкапсульовані* в об'єкті;
- поведінка об'єкта в UML – це будь-які правила взаємодії об'єкта із зовнішнім світом і з даними самого об'єкта;
- поділ сутностей на класи і побудова загальної класифікації здійснюються за допомогою механізму спадкування і поліморфізму;
- *спадкування* – це відношення, що визначає рівень ієрархії конкретного класу в дереві класів, і говорить про те, що нащадки конкретного класу є різновидом класу-батька. Механізм спадкування реалізується за допомогою копіювання всіх атрибутів предка (спадкування) і їхнього часткового перевизначення. Перевизначати можна як дані, так і поведінку (методи);
- *поліморфізм* стосується перевизначення поведінки об'єктів. В UML для опису поліморфізму вводяться поняття *операції* і *методу.* У класів є операції, що визначають їхню поведінку. Вони успадковуються нащадками, але кожен нащадок класу може надати своп-метод реалізації будь-якої успадкованої операції, відмінний від відповідного методу предка. Підкреслимо, що з операцією пов'язаний якісний опис поведінки об'єкта, а з методом – його конкретна реалізація. Таким чином, успадковуючи операції, можна додавати їм потрібні властивості, що мають об'єкти класу-нащадка;
- для зручності ієрархічного подання великих систем класи можна поєднувати в групи (пакети) або використовувати модульний підхід під час проектування.

2.4.2. Загальні поняття

У процесі моделювання всі досліджувані сутності розглядаються з двох позицій. Насамперед ми намагаємося знайти групи близьких за своїми властивостями елементів і створити загальний для них опис. На цьому етапі ми абстрагуємося від індивідуальних властивостей елементів. Потім ми зазначаємо механізм, що дозволяє з абстрактного елемента одержати конкретний, із властивими йому індивідуальними рисами, і використовуємо елементи, що вийшли, для конструювання системи. Таким чином, сутності подаються парами тип – екземпляр. Таких пар небагато, наприклад: клас – об'єкт, асоціація – зв'язок, параметр – значення, операція – виклик процедури. Для зображення елементів цих пар на діаграмах узагальнення і конкретний екземпляр геометрично подаються однаково, так що можна зустріти й діаграми, наприклад класи і діаграми об'єктів.

Для всіх типів діаграм існує ряд спільних елементів.

- *Рядки.* Зазвичай це послідовності літералів, що можуть містити практично будь-які символи. Рядки можуть поєднуватися в параграфи. Серед рядків виділяють:

– *імена* – розташовуються в спеціально відведених місцях елементів діаграм;

– *шляхи* – локалізують відповідний елемент в ієрархії описів;

– *мітки* –дають додаткову інформацію про елемент, розташовуються поблизу об'єкта і становлять з ним єдине ціле;

– *рядки-характеристики* – мають заданий синтаксис і несуть інформацію про індивідуальні властивості об'єкта, що виражається в присвоєнні атрибутові конкретного значення.

- *Типи.* Різниця між типом і класом досить умовна (наприклад, можна зустріти мови, де ціла змінна є екземпляром класу "цілі"), але UML розрізнює ці два поняття, говорячи, що атрибути, змінні і параметри можуть належати до певного типу. Самі типи не визначаються, хоча передбачається, що вони можуть бути і визначеними, і користувацькими. Можна створити спеціальний клас "TypeExpression" і помістити туди всі потрібні визначення і правила.

2.4.3. Основні компоненти UML

2.4.3.1. Класифікація діаграм

Автори мови UML визнають, що важливе значення мають як надійна мова моделювання, так і сам процес розробки. Вони лише наводять свої рекомендації щодо організації процесу в окремих публікаціях, що не стосуються до UML, оскільки стандартизація процесу розробки програмних систем виходить за рамки мови UML.

Дотримання стандартів процесу розробки додатків не головне для створення системи. Розробникові набагато важливіше мати навички створення хороших проектів. Для цього потрібно освоїти ряд принципів і евристик, пов'язаних з ідентифікацією і виділенням основних абстрактних об'єктів, а також з розподілом обов'язків між ними.

Мова UML дозволяє стандартизувати артефакти і систему позначень, але не визначає стандарт процесу розробки. Через це:

1) стандартну систему позначень можна використовувати у створенні систем будь-якого призначення, процес розробки цих систем може відрізнятися від стандартного;

2) процес розробки системи може залежати від багатьох факторів, таких як кваліфікація фахівців, частка творчості в загальному процесі, природа проблеми, засоби її розв'язання тощо.

Основні компоненти, що утворюють UML, включають опис семантики UML, графічної нотації і додаткових понять, що дозволяють розширити зміст основних понять мови. UML – це графічна мова, вивчення якої слід починати з вивчення застосовуваних у ній графічних образів ("Notation Guide"). Для правильного трактування графічних образів слід познайомитися зі змістовною стороною ("UML Semantics"). UML − це мова моделювання, а не посібник розробника з об'єктно-орієнтованого аналізу і проектування. Природно, що методи, моделі і засоби створення ефективних програмних систем розвиватимуться й надалі. Однак лише зараз фахівці мають можливість користуватися єдиною мовою – UML.

У процесі розробки система подається у вигляді об'єднання декількох проекцій, кожна з яких описує певний аспект системи, а разом вони визначають систему в усій її повноті. Ці проекції подаються в UML за допомогою діаграм.

• *Діаграми класів* (class diagrams). Показують статичну структуру системи, тобто визначають типи об'єктів системи і різного роду статичні зв'язки й відносини між ними. Діаграми класів містять набір статичних (декларативних) елементів, таких як класи, типи та їх зв'язки, зображених у вигляді графа. Діаграми класів можуть бути логічно об'єднані в пакети.

• *Діаграми варіантів використання* (use case diagrams). Складні системи, мають багато функцій. Існує багато варіантів їх використання (сценаріїв). Та сама *дійова особа* (actor) рідко використовує усі функції системи, тому всіх дійових осіб можна умовно поділити на групи, відповідно до типових сценаріїв використання певних функцій. Для кожної групи можна скласти свій об'єднаний груповий сценарій (use case): набір усіх можливих сценаріїв застосування тієї або іншої частини системи. Список усіх групових сценаріїв визначає функціональні вимоги до системи, за допомогою яких може бути сформульовано технічне завдання. Діаграми варіантів використання – граф, за допомогою якого показано всіх типових дійових осіб та їхню взаємодію із системою. Взаємодія представлена сценаріями застосування.

• *Діаграми взаємодії* (interaction diagrams) поділяються на *діаграми послідовності та кооперативні діаграми.* На обох діаграмах зображено часову послідовність використання об'єктів у реалізації конкретного сценарію і повідомлень, якими вони при цьому обмінюються.

• *Діаграми послідовності (sequence diagrams)* показують, у якій послідовності з'являються об'єкти під час виконання тієї або іншої операції (сценарію) і який потік повідомлень при цьому виникає. Діаграми послідов-

ності мають дві осі: вертикальна відображає час, горизонтальна – різні об'єкти. Зазвичай інтерес становить лише послідовність появи об'єктів у міру виконання операції, але у випадку систем реального часу можуть знадобитися і конкретні значення часу.

• *Кооперативні діаграми (collaboration diagrams).* Мета цієї діаграми така, як і діаграми послідовності, але декому вона може здаватися зручнішою. На діаграмі у вигляді графа зображуються об'єкти, що беруть участь у виконанні операції, їхній зв'язок і послідовність появи. Як правило, це дерево, тому що для складніших випадків такі діаграми стають дуже непростими для сприйняття. Повідомлення, якими обмінюються об'єкти, зображені у вигляді стрілок. Щоб показати їх часову послідовність, кожну стрілку пронумеровано.

• *Діаграми станів (state diagrams).* Будь-який об'єкт системи може змінювати свою поведінку залежно від внутрішніх або зовнішніх подій, або, іншими словами, може реагувати на події, змінюючи свій стан. Діаграми станів показують послідовність станів, у яких може виявитися об'єкт, залежно від подій, що відбуваються, разом з їхніми реакціями на ці події. Діаграми станів описують стан тільки одного класу або об'єкта. Стани в UML трактуються так само, як у карті станів Харела, тобто, коли система перебуває в одному зі станів, фіксується певний набір значень змінних системи, що встановилися у відповідь на подію, яка відбулася.

• *Діаграми діяльності (активності).* У багатьох випадках ми спостерігаємо зміну станів об'єкта, викликану тільки внутрішніми причинами. Усередині об'єкта виконуються послідовно або паралельно певні тривалі дії. Для такого типу систем UML і пропонуємо використовувати діаграми діяльності, які призначені для того, щоб відобразити переходи, викликані внутрішніми процесами (на противагу зовнішнім подіям, з якими має справу діаграма стану). Діаграми діяльностей використовуються для опису складної поведінки класу або сценарію і відображення складних операцій у вигляді послідовності паралельних і послідовних тривалих дій.

Подані нижче діаграми ніяк прямо не стосуються моделювання, їх наведено лише для повноти картини.

• *Діаграми реалізації (implementation diagrams).* Діаграми реалізації складаються з компонентних діаграм і діаграм застосування (розгортання). Вони, на відміну від діаграм стану, взаємодії, використання і класів, що логічно відображають систему в процесі її розробки, дають фізичне уявлення про систему.

• *Компонентні діаграми (component diagrams)* показують взаємозв'язок між компонентами ПЗ, включаючи компоненти у вихідному коді, бінарні компоненти і компоненти, що виконуються. Деякі компоненти можуть існувати тільки під час виконання, зв'язування (linking) або компіляції.

• *Діаграми застосування (розгортання) (deployment diagrams)* використовуються для зображення схеми розташування процесорів і пристроїв, задіяних у реалізації системи, а також зображення з'єднань між ними маршрутів передачі інформації.

2.4.4. Кількісна оцінка діаграм

Методика кількісної оцінки і порівняння діаграм UML будується на присвоєнні елементам діаграм оцінок, залежних від їх інформаційної цінності, а також від додаткової складності, що вноситься ними в діаграму. Цінність окремих елементів змінюється залежно від типу діаграми, на якій вони містяться.

Словник мови UML включає два види будівельних блоків: сутності і відношення. Сутності – це абстракції, що є основними елементами моделі. Відношення пов'язують різні сутності.

Кількісну оцінку діаграми можна провести за такою формулою:

$$S = \frac{\sum S_{Obj} + \sum S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}},$$

де S – оцінка діаграми;
S_{Obj} – оцінки для елементів діаграми;
S_{Lnk} – оцінки для зв'язків на діаграмі;
O_{bj} – кількість об'єктів на діаграмі;
T_{obj} – кількість типів об'єктів на діаграмі;
T_{Lnk} – кількість типів зв'язків на діаграмі.

Якщо діаграма містить велику кількість зв'язків одного типу (наприклад, модель БД), то кількість і тип зв'язків можна не враховувати і формула розрахунку зводиться до вигляду

$$S = \frac{\sum S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}}.$$

Якщо на діаграмі показано атрибути й операції класів, то можна врахувати їх під час розрахунку, при цьому оцінка додається до оцінки відповідного класу:

$$S_{ds} = \frac{\sqrt{O_p} + \sqrt{Art}}{0{,}3 \times (Ob + Art)},$$

де *Scls* – оцінка операцій та атрибутів для класу;
Op – кількість операцій класу;
Art – кількість атрибутів класу.

При цьому враховуються тільки атрибути й операції, що відображені на діаграмі.

Далі в табл. 2.5 і 2.6 наводяться оцінки для різних типів елементів і зв'язків.

Таблиця 2.5

Основні елементи мови UML

Тип елемента	Оцінка для елемента
Клас (class)	5
Інтерфейс (interface)	4
Прецедент (use case)	2
Компонент (component)	4
Вузол (node)	3
Процесор (processor)	2
Взаємодія (interaction)	6
Пакет (package)	4
Стан (state)	4
Примітка (node)	2

Таблиця 2.6

Основні типи зв'язків мови UML

Тип зв'язку	Оцінка для зв'язку
Залежність (dependency)	2
Асоціація (association)	1
Агрегація (aggregation)	2
Композиція (composition)	3
Узагальнення (generalization)	3
Реалізація (realization)	2

Інші типи зв'язків потрібно розглядати як асоціації.

Недоліком діаграми вважається як занадто низька оцінка (при цьому діаграма недостатньо інформативна), так і занадто висока оцінка (діаграма зазвичай занадто складна для розуміння). У табл. 2.7 наведено діапазони оптимальних оцінок для основних типів діаграм.

Таблиця 2.7

Діапазони оцінок для діаграм UML

Тип діаграми	Діапазон оцінок
Класів (class) − з атрибутами й операціями	5–5,5
Класів (class) − без атрибутів і операцій	3–3,5
Компонентів (component)	3,5–4
Варіантів використання (use case)	2,5–3
Розгортання (deployment)	2–2,5
Послідовності (sequences)	3–3,5
Кооперативна (cooperative)	3,5–4
Пакетів (package)	3,5–4
Станів (state)	2,5–3

Далі наведено приклад оцінки простої діаграми класів за цією методикою.

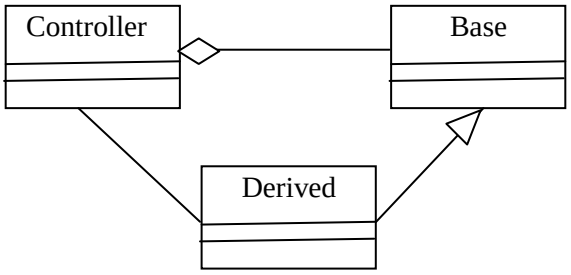


Рис. 2.13. Діаграма класів

Діаграма містить три класи без операцій і атрибутів, тому *Tobj* = 1, $\sum S_{obj} = 15$ і *Obj* = 3. У функцій зв'язку використовуються асоціація, агрегація і узагальнення; отже, $\sum S_{Lnk} = 6$ і $T_{Lnk} = 3$.

$$S = \frac{\sum S_{Obj} + \sum S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{15 + 6}{1 + 3 + 2} = 3,5.$$

Отже, числова оцінка для цієї діаграми дорівнює 3,5.

2.4.5. Діаграми варіантів використання

Діаграми варіантів використання – це графічне зображення взаємодії користувача і комп'ютерної системи, у даному випадку комп'ютерної моделі. Кожен варіант використання охоплює деяку очевидну для користувачів функцію системи і виконує певне дискретне завдання користувача. Список усіх варіантів використання функціонально визначає вимоги до системи, за допомогою яких може бути сформульоване технічне завдання, тому діаграма варіантів використання є необхідним засобом для аналізу вимог щодо проектованої системи.

Діаграма варіантів використання – граф, що описує взаємодію *дійових осіб* із системою, поданою *варіантами використання.*

Дійова особа – це користувач з певними чітко фіксованими вимогами до моделі. Варіант використання – типова взаємодія користувача і комп'ютерної системи, виконує якесь дискретне завдання користувача. Кожен варіант використання – це потенційна вимога до системи. Нотація варіанта використання не повинна містити докладні описи, досить декількома реченнями описати поставлену вимогу.

Ділову особу можна зобразити на діаграмі прямокутником (аналогічно до класу) зі стереотипом "actor". Але найчастіше дійову особу зображують піктограмою у вигляді людської фігурки, а її ім'я розташовують під фігуркою.

Варіант використання має вигляд еліпса, всередині якого або під ним розташовується його ім'я (рис. 2.14). Припустимо, що модель застосовуватимуть три типи користувачів: глядачі, що можуть тільки спостерігати за рухом конкретного автомобіля по конкретній трасі; учні, яким дозволено вибирати автомобіль і трасу; випробувачі, що мають право вибирати значення параметрів траси.

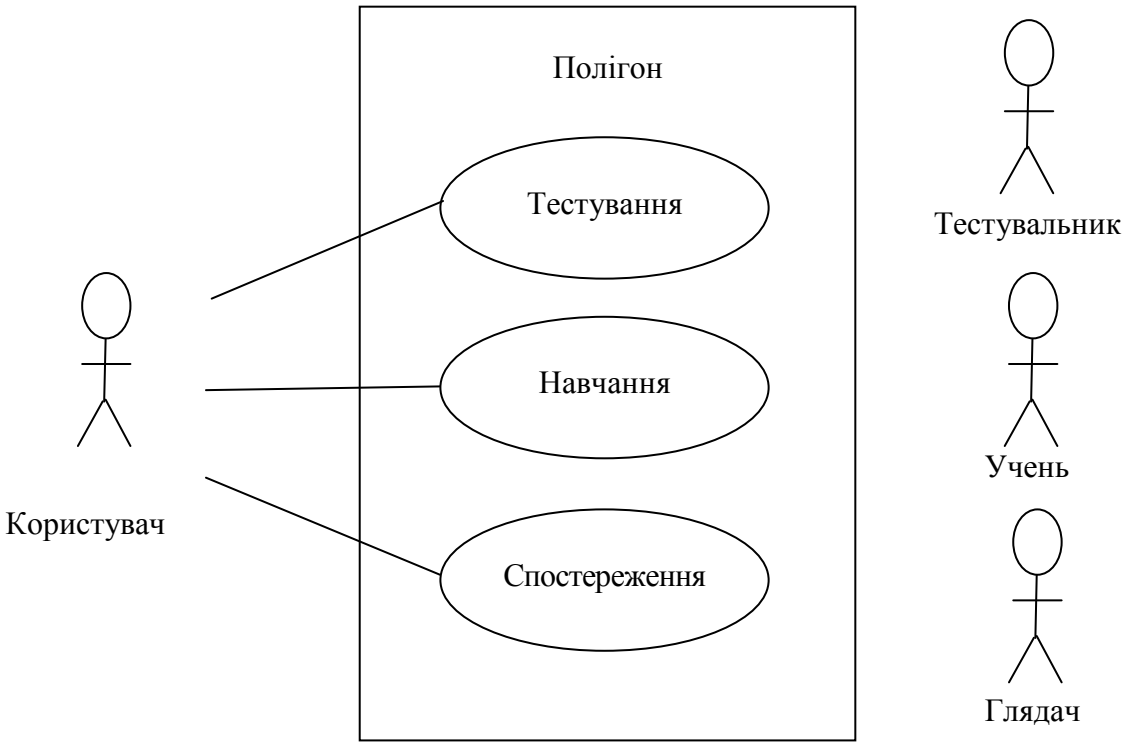


Рис. 2.14. Діаграма варіантів використання

У діаграмі варіантів використання є такі зв’язки (рис. 2.15):

- *комунікація (communicates)* – показує участь дійової особи у варіанті використання, з’єднуючи символ дійової особи і символ варіанта використання суцільною лінією. Дійова особа “спілкується” з варіантом використання цим зв’язком;
- *розширення (extends)* – лінія зі стереотипом “extends”, з незаповненою стрілкою на кінці, з’єднує базовий варіант використання з розширювальним варіантом його використання. Кінець з незаповненою стрілкою вказує на варіант використання, що є розширенням базового варіанта. Такий тип зв’язку використовується тоді, коли один варіант використання подібний до іншого, але несе додаткове навантаження. Особливо зручно використовувати такий тип зв’язку в описі обробки аварійних ситуацій, що виникають у системі, щоб не перевантажувати основний варіант використання, який описує нормальну поведінку системи, зайвою логікою;
- *використання (uses)* – лінія з написом “uses”, з незаповненою стрілкою на кінці, з’єднує один варіант використання з іншим варіантом, який він використовує. Такий тип зв’язку називається *використання* і застосовується в тих випадках, коли є певний фрагмент поведінки системи, що повторюється більш ніж в одному варіанті використання, і не хочеться копіювати його в кожному з цих варіантів. У цьому випадку даний фрагмент оформлюється як окремий варіант використання, до нього проводяться відповідні зв’язки від інших варіантів.

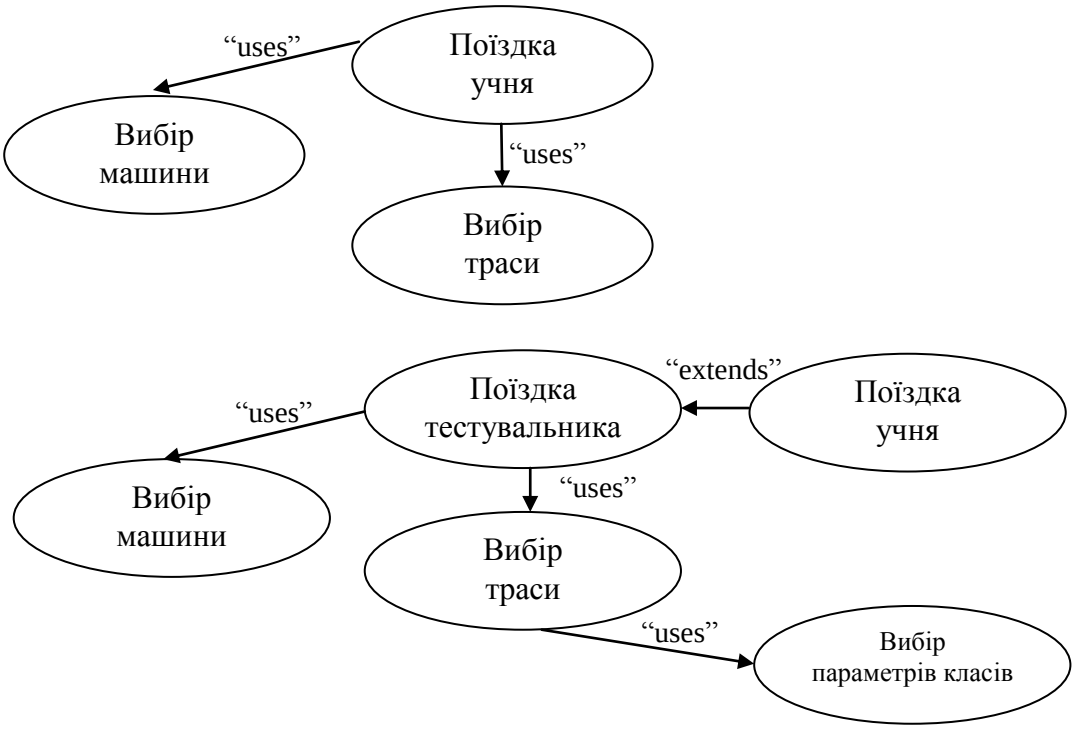


Рис. 2.15. Зв’язки на діаграмі варіантів використання

На рис. 2.15 пояснюється, чим відрізняється варіант використання випробувача від варіанта використання учня. Передбачається, що спочатку по новій трасі проїжджає випробувач і добирає параметри так, щоб за певних навичок керування по ній можна було проїхати, не відчуваючи надмірної тряски, а потім фіксує їх. Так з’являється нова траса для учня.

У ряді мов моделювання “сценарії” використання моделей заздалегідь визначено, і різні варіанти використання не потрібно “проектувати”, вони виконуються автоматично. Наприклад, якщо сценарій полягає в пошуку оптимальних параметрів, то досить звернутися до відповідної процедури. Сучасні програмні засоби моделювання зазвичай надають користувачеві можливість використовувати готові сценарії, а також писати свої.

2.4.6. Діаграми класів

Діаграми класів ілюструють зв’язки програмних елементів, а не понять із предметної сфери. Позначення класу складається з трьох частин, в яких зазначається ім’я класу, його атрибути і методи.

Діаграми класів (class diagrams) показують статичну структуру системи, тобто визначають типи об’єктів системи і різного роду статичні зв’язки та відношення між ними. Діаграми класів містять набір статичних елементів, наприклад класи, типи і їхні зв’язки, об’єднані в граф. Серед зв’язків особливо виділяють *асоціації і підтипи.* На діаграмах класів також зображуються атрибути класів, операції класів і обмеження, що накладаються на зв’язки між об’єктами. Діаграми класів можуть бути логічно об’єднані в пакети.

Клас (class) – це сутність, що описує множину об’єктів з подібною структурою, поведінкою і зв’язками з іншими об’єктами.

На діаграмах клас зображується у вигляді прямокутника, розділеного горизонтальними лініями на 3 секції (рис. 2.16).

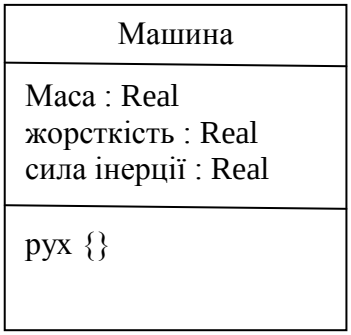


Рис. 2.16. Клас “Машина”

Верхня секція (секція імені) містить ім’я класу й інші загальні властивості (зокрема, тип класу). У середній секції наведено список атрибутів, а в нижній – список операцій. Атрибути зберігають інкапсульовані дані класу, а операції описують поведінку об’єктів класу.

Кожна з останніх двох секцій – атрибутів і операцій – може не зображуватися. Для такої секції не потрібно малювати розділову лінію й/або вказувати на наявність чи відсутність елементів у ній.

Класи можуть об’єднуватися в більші компоненти, так звані *пакети.* Область видимості класу – це пакет, у якому він описаний. Це означає, що імена класів мають бути унікальні серед імен класів того ж пакета. За замовчуванням вважається, що клас визначено в поточному пакеті. Якщо необхідно послатися на клас з іншого пакета, то чітко зазначається:

<ім’я пакета>::<ім’я класу>

Ієрархія пакетів може мати глибину вкладеності більшу, ніж 1, тоді шлях до класу може містити більше, ніж одне посилання, при цьому шлях починається від кореня ієрархії пакетів:

<ім’я пакета1>::<ім’я пакета2>::...::<ім’я пакетаN>::<ім’я класу>

У секції імені класу можуть міститися (один за одним зверху вниз):

- тип класу (і/або значок типу в правому верхньому куті) – необов’язкове поле, опускається, якщо йдеться про неспецифікований клас;
- ім’я класу (якщо клас абстрактний – курсивом);
- додаткові властивості – ім’я автора тощо (необов’язкове поле).

Середня і нижня секції прямокутника класу містять списки його атрибутів і операцій. Описуючи клас, необов’язково відразу їх заповнювати. Це можливо лише в тому випадку, коли є чітке уявлення, які операції виконуватиме даний клас і які атрибути для цього необхідні. На початковій стадії опису це не завжди ясно, тому для початку вміст цих полів може бути опускатися.

Насамперед уявимо, як трактує поняття “атрибут” UML.

Атрибут (attribute) – це елемент даних класу, тобто елемент даних, який міститься в об’єкті, що належить описуваному класові.

Атрибут повинен мати *тип (type expression),* простий або складний: Array, Real, Vector, Matrix.

Деталі, що стосуються типів атрибутів, неспецифіковані UML. Більш докладний опис типу залежить від того, яку мову програмування використовують розробники. Атрибут зображується у вигляді текстового ряд-ка, що відбиває різні його властивості:

<ознака видимості><ім’я>::<тип>=<значення за замовчуванням> {властивості}

Можуть бути такі властивості:

• *ознака видимості* має C++-семантику видимості членів класу:

• *загальний атрибут (public)* (позначається символом +) означає, що будь-яка сутність, яка має доступ до об’єкта обумовленого класу, має доступ і до цього атрибута;

• *захищений атрибут (protected)* (позначається символом #) доступний тільки для методів класу і його послідовників;

• *приватний атрибут (private)* (позначається символом -) доступний тільки для методів класу;

• *ознака ділянки видимості* може зображуватися ключовим словом “public”, “private”, “protected” або може бути опущена. Це означає, що область видимості не показується (а не те, що вона не визначена або “public” за замовчуванням);

• *ім’я* – це ідентифікатор, що подає ім’я атрибута;

• *тип* – залежно від мови реалізації опис типу атрибута;

• *значення за замовчуванням* – залежний від мови реалізації вираз, що задає початкове значення для атрибута новоствореного об’єкта. Ця частина визначення атрибута необов’язкова;

• *властивості* – ряд додаткових властивостей елемента (необов’язкова частина). Якщо властивості не зазначаються, дужки {} опускаються. Прикладом властивості може бути ім’я автора: {Author = Smith}.

За замовчуванням атрибут змінюється. Додавши в його властивостях позначку {frozen}, можна оголосити атрибут незмінним. Для атрибута можна вказувати його множинність. Якщо вона не позначена, то передбачається, що атрибут зберігає лише одне значення. Множинність може бути визначена в квадратних дужках відразу після імені атрибута:

coords[3]: integer

У системах моделювання атрибути мають надзвичайно важливі властивості, не відбиті в даному визначенні, що залежать від *виду атрибута.* Звичайно ж, їх можна було б зарахувати до групи додаткових властивостей, якби не їхня важливість. Справа в тому, що об’єкт (або екземпляр класу) у мовах моделювання, як правило, асоціюється з конкретним пристроєм або його елементом. Навіть прості технічні системи зазвичай склада-

ються з декількох взаємозалежних об’єктів, здатних сприймати і передавати інформацію.

З погляду моделювання тривалість (безперервні дискретні процеси) тих або інших операцій надзвичайно важлива на початковому етапі, якщо тривалістю деякої операції можна знехтувати, то модель може істотно спроститися.

Операція (operation) – це сутність, що визначає якусь дію, котру виконує представник класу. Операції ім’я і список аргументів.

Операція зображується текстовим рядком за такою схемою:

<ознака видимості> <ім’я> (список параметрів) : < тип виразу, що повертає значення> {властивості}, де ознака видимості, ім’я і властивості мають той же зміст, що і для атрибута;

список параметрів – список формальних параметрів, розділених комами;

тип виразу, що повертає значення – залежно від мови реалізації опис типу значення, що повертається функцією. Якщо воно не зазначено, то передбачається, що функція не повертає значення (void для C/C++).

Кожен елемент списку параметрів має такий вигляд:

<ім’я> : <тип>=<значення за замовчуванням>,

де ім’я – ім’я параметра;

тип – залежно від мови реалізації опис типу параметра;

значення за замовчуванням – значення параметра за замовчуванням.

В останній версії UML передбачено також можливість зазначення типу параметра (вхідний, вихідний або змішаний тип).

Усі операції, зазначені в класі, можна поділити на дві групи: операції класу та операції представника. Операції класу властиві не об’єктам класу, а самому класові. Звідси, зокрема, випливає, що операції класу не мають доступу до атрибутів. Типовий приклад операції класу − функція створення нового об’єкта (представника) класу. Операції класу виділяються підкресленням:

new (Машина) [маса=100, твердість=100]; операція, що не змінює стан системи, позначається в такий спосіб: у список властивостей операції додається властивість {query}.

Елементи списків атрибутів і операцій можна групувати за деякими ознаками. У цьому випадку перед групою елементів розміщується рядок у лапках, тобто рядок, що визначає властивість, причому ця властивість поширюється на всі нижче розташовані елементи до нової властивості. Ця можливість добре ілюструється прикладом:

“параметри машини”: твердість підвіски і маса машини.

Насамкінець зауважимо, що в кожної секції прямокутника класу може бути ім’я. Оскільки секція “ім’я класу” обов’язкова, то її ім’я не зазначається (рис. 2.17).

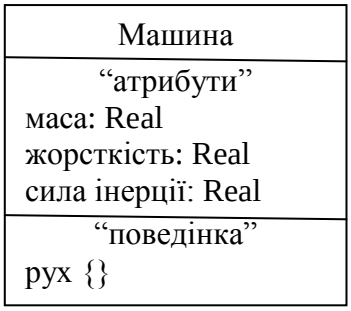


Рис. 2.17. Клас “Машина з іменами секцій”

Одне з найважливіших понять об’єктно-орієнтованого програмування – це поняття *об’єкта (object)*. Об’єкт або екземпляр класу, наділяється індивідуальними рисами. Об’єкти можуть виконувати певні ролі. Роль визначає відношення між класом і його екземплярами, виділяючи певну їх підмножину. Вважається, що всі ці об’єкти схожі за поведінкою і станами.

На діаграмі об’єкт зображується як прямокутник із двома секціями (рис. 2.18). Верхня секція містить у собі ім’я об’єкта і його класу, підкреслене суцільною лінією, і синтаксис.

За необхідностю ім’я класу може містити в собі повний шлях до даного класу. Імена пакетів мають випливати перед ім’ям класу і розділятися парами двокрапок. Наприклад:

вертикальна координата : : Підвіска : : Машина

Ім’я об’єкта може бути опущено. У цьому випадку в першій секції пишеться двокрапка й ім’я класу. Ім’я класу даного об’єкта також може бути опущене (разом із двокрапкою).

У другій секції міститься список імен атрибутів з їхніми типами і значеннями. Кожен рядок зі списку має такий синтаксис:

<ім’я атрибута>:<тип>=<значення>

Тип атрибута і його значення можуть не зазначатися. Деякі з атрибутів, що не становлять інтересу, також можуть бути опущені.

Об’єкт може набувати різних станів. Щоб позначити це, поруч з ім’ям об’єкта у квадратних дужках подається список його станів, у яких він може перебувати протягом життєвого циклу. Стани об’єкта формуються на етапі аналізу проектованої системи, тобто виділяються деякі основні фази, в яких може бути об’єкт. Далі, під час проектування системи, ці стани можна коректувати.

Рис. 2.18. Об’єкти

Складений об'єкт (composite object) − екземпляр складеного класу, тобто класу, що має відношення композиції з іншими класами. Таким чином, складений об'єкт складається з інших (можливо, також складених) об'єктів. На діаграмі так само, як і простий об'єкт, зображується ім'я об'єкта і розташовується у верхній секції прямокутника, а в нижній секції замість атрибутів об'єкта розташовуються частини складеного об'єкта (рис. 2.19). Зміст складеного об'єкта може бути опущено і повідомлення, призначені для внутрішніх складових об'єкта, можуть безпосередньо до самого об'єкта. Внутрішні повідомлення, якими обмінюються складові об'єкта, також можуть бути опущені. Повідомлення зазвичай показуються на одній діаграмі разом зі складовою об'єкта.

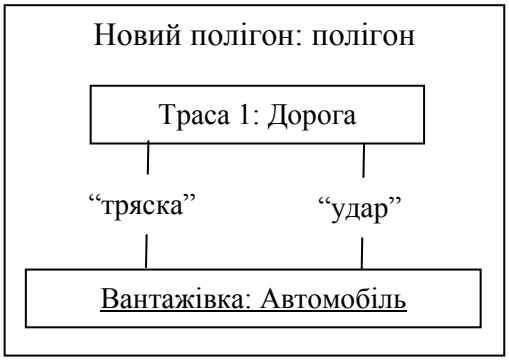


Рис. 2.19. Складений об'єкт

Активний об'єкт (active object) має можливість ініціювати дію. Пасивний об'єкт може містити в собі дані, але не може ініціювати дії. Однак пасивний об'єкт може посилати повідомлення в процесі обробки запиту, який він одержав. З погляду моделювання тут уводиться розподіл на безперервні і дискретні дії.

Активний об'єкт − має потік керування. Він зображується на діаграмі як звичайний об'єкт, обведений суцільною жирною лінією. Часто він подається як композиція зі складених частин.

2.4.7. Типи зв'язків між класами

Щоб на діаграмі можна було показати взаємодію класів або те, що вони посилають один одному повідомлення, між ними необхідно позначити зв'язок. Можливі зв'язки таких типів: асоціації, залежності, агрегації, повідомлення.

Асоціація (association) визначає логічний зв'язок між класами. Коли в системі створюються представники асоційованих класів, вони пов'язуються так, як визначає дана асоціація. В UML одна асоціація може специфікувати зв'язок між двома і більше класами. Асоціації першого типу називаються бінарними, а другого типу – N-арними.

Бінарна асоціація (binary association) характеризує логічний зв'язок між двома класами. Можливий також зв'язок класу із самим собою, що називається *рефлексивною асоціацією.*

Зображується асоціація у вигляді суцільної лінії, що з'єднує символи класів. Кожна асоціація, як і будь-який зв'язок, має напрямок, або в термінах UML – *ролі (association role).* Ролей може бути дві, щоб мати можливість підкреслити можливі розбіжності у "зв'язках" між класами, наприклад у "Начальника" можуть бути зосереджені всі права, а в "Підлеглого" – тільки обов'язки. На лінії, що зображує такі позначки:

- *ім'я асоціації* – визначає необов'язкове ім'я асоціації;
- *клас асоціації* – клас, що дозволяє визначати для асоціацій атрибути, операції та інші властивості (з'єднується з лінією асоціації пунктиром). Ця мітка використовується, якщо необхідно приєднати до асоціації якусь додаткову інформацію.

Роль (association role) – це невіддільна частина асоціації, що описує роль класу в даній асоціації.

У ролі можуть бути такі властивості:

- *Ім'я ролі* − рядок, що міститься поруч із кінцем лінії асоціації. Поле необов'язкове, але якщо ім'я задане, воно має відображатися на діаграмі.
- *Навігація.* Можливість навігації щодо ролі означає, що партнери асоціації можуть переглядати об'єкти, які відповідають цій ролі. Якщо в напрямі, що відповідає ролі, підтримується навігація, на кінці лінії може бути зображена стрілка.
- *Множинність* показує кількість конкретних об'єктів, що можуть пов'язуватися з даним партнером асоціації. У цілому множинність показує нижню і верхню границі кількості об'єктів, що можуть брати участь в асоціації.
- *Кваліфікатор* – список атрибутів класу з протилежного кінця лінії асоціації, за значеннями яких можна однозначно розбити множину об'єктів цього класу на підмножини. Використовується для зв'язку об'єкта класу – партнера асоціації з групою об'єктів іншого класу – партнера асоціації.
- *Агрегація* показує, що асоціація – це відношення типу ціле/частина.

В останній версії UML передбачено можливість зазначення змінюваності асоціації. Якщо асоціація змінювана, тобто може бути додана, вилучена і переміщена, то ніяких додаткових позначок не потрібно. В іншому разі в рядку у властивостей може бути наявною мітка {frozen}, яка показує, що асоціацію не можна додавати, видаляти і переміщати. *Множинність (multiplicity)* показує можливу кількість об'єктів, що можуть бути зв'язані відповідно до цієї асоціації. Множинність зазначається для ролей асоціації та має такий формат (рис. 2.20):

<нижня межа>..<верхня межа>

Верхня і нижня межа показують мінімальну і максимальну кількість об'єктів, що беруть участь в асоціації. Якщо верхню межу позначено символом ' * ', то це значить, що вона нескінченна.

Приклад

0. .1

10

0. .*

3..5, 10..20, 100, 200..*

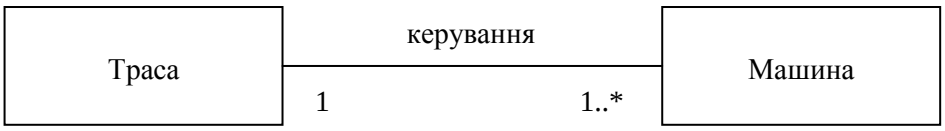


Рис. 2.20. Приклад асоціації із вказівкою множинності

У працюючій системі між об'єктами асоційованих класів установлюються зв'язки (екземпляри асоціації). Але в деяких випадках слід розбити множину об'єктів одного класу, що будуть пов'язані з об'єктом іншого класу відповідно до даної асоціації, на підмножини за значеннями деяких атрибутів цих об'єктів, і накласти обмеження на кількість об'єктів у тій або іншій підмножині.

В UML дається така можливість: в асоціації може бути атрибут за назвою *Кваліфікатор (qualifier),* що містить один або кілька атрибутів класу, прикріпленого до іншого кінцеві асоціації. Саме за значенням таких атрибутів відбувається групова вибірка об'єктів цього класу з боку об'єкта протилежного (в даній асоціації) класу. Кваліфікатор зображується у вигляді маленького прямокутника, приєднаного до початку лінії асоціації (рис. 2.21). У ньому зазначаються атрибути іншого класу – партнера асоціації.

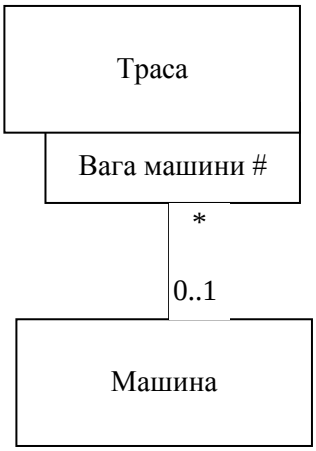


Рис. 2.21. Кваліфікатор

Кваліфікатор, зображений на рис. 2.21, можна трактувати як обмеження на використання траси машинами з певною вагою і на кількість машин, що одночасно проїжджають по цій трасі.

Якщо в ролі асоціації встановлено атрибут "aggregation", то вся асоціація є *відношенням агрегації.* Такий атрибут можна встановити тільки в одній з ролей.

Агрегація (aggregation) – це відношення між класами типу ціле/частина. Клас, що агрегується в тій або іншій формі, стає частиною агрегату. На практиці це може бути реалізовано по-різному. Наприклад, об'єкт класу-агрегату може зберігати об'єкт класу, що агрегується або зберігати посилання на нього. В UML допускається можливість агрегації одного класу багатьма, тобто один клас може бути частиною декількох цілих. Спеціальний вид агрегації, що називається *композицією (composition),* цього не допускає.

Композиція – спеціальний вид агрегації (сильна агрегація). Вона показує, що даний клас може бути частиною тільки одного класу. Зокрема, об'єкт, що агрегується, створюється тільки тоді, коли створено агрегат, а зі знищенням агрегату знищуються всі об'єкти, що агрегуються.

Агрегацію зображують на діаграмі у вигляді порожнього ромба на кінці лінії асоціації з боку класу, що агрегується (агрегату). Композицію з так само, як і агрегацію, але ромбик не порожній, а заповнений (рис. 2.22).

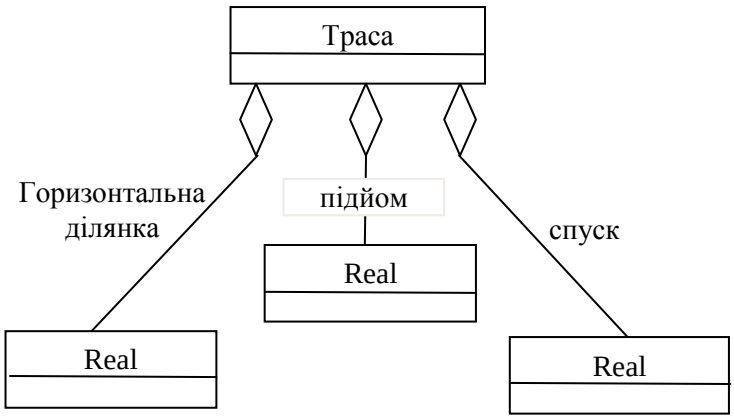


Рис. 2.22. Приклад композиції

У деяких випадках два і більше елементи моделі можуть бути семантично пов'язані. Наприклад, клас А використовує методи класу В. Тоді зі зміною класу В необхідно внести відповідні зміни в клас А. Тому в нотації UML передбачене таке відношення, як *залежність (dependency).* Для розглянутого прикладу на діаграмі класів слід зазначити, що клас А залежить від класу В. Відношення залежності універсальне, тобто за його допомогою можна пов'язувати різні типи сутностей UML.

Залежність зображується пунктирною лінією між двома елементами діаграми, вважається, що елемент, "прив'язаний" до кінця стрілки, залежить від елемента, "прив'язаного" до початку цієї стрілки. Залежність можна позначити ім'ям і специфікатором.

Існують такі види залежностей:

- *trace* показує історичний зв'язок між двома елементами, що представляли те саме поняття на різних етапах;
- *refine* – історичний зв'язок між елементами, як правило, показує, що один елемент ніби відбився від іншого;

- *uses* – ситуація, коли один елемент моделі використовує іншу;
- *bind* – установлюється між шаблоном і екземпляром шаблона;
- *Friend* – аналог ключового слова C++ friend.

Спадкування (inheritance) – це відношення типу "загальне часткове" між елементами моделі. Спадкування позначається суцільною лінією, від приватного елемента до більш загального (у термінології ООП – від нащадка до предка, від сина до батька, або від підкласу до суперкласу). З боку більш загального елемента зображується великий порожній трикутник.

Один із атрибутів відношень спадкування – *дискримінатор (discriminator)* – рядок, що задає ім'я групи нащадків. Його використання корисне, якщо в даного класу багато нащадків, і необхідно розбити їх на кілька груп. Відсутність дискримінатора означає, що дискримінатор – порожній рядок (дискримінатор за замовчуванням). Зображується дискримінатор текстовим рядком, міститься біля лінії спадкування.

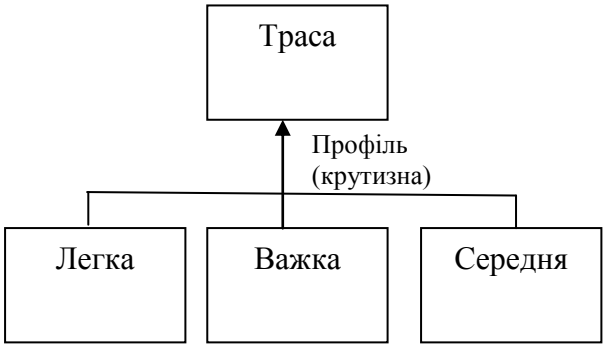


Рис. 2.23. Спадкування

На рис. 2.23 профіль – це дискримінатор, а {крутизна} – його додаткова властивість.

У UML існує кілька різновидів класу: інтерфейс, шаблон, утиліта та ін.

Інтерфейс (interface) – клас, що задає набір операцій, але не містить полів і реалізації цих операцій. Клас, що реалізує інтерфейс, сам визначає вміст цих операцій.

Шаблон (template) або *параметризований клас (parameterized class).* Шаблони UML дуже схожі на шаблони C++. Вони визначають родину класів, що відрізняються значенням деяких формальних параметрів.

Утиліта (utility) – клас, який поєднує групу загальнодоступних (глобальних) змінних і процедур.

Для визначення виду класу в UML уведено поняття *стереотипу (stereotype).* Стереотип визначає підтип будь-якого глобального типу "клас". Відповідно, класи-інтерфейси мають стереотип "interface", а класи-утиліти – "utility". Існує стандартний набір стереотипів, які за необхідності можна розширювати.

Інтерфейс (interface) у UML – це опис (без реалізації) групи функцій, які він надає для використання іншому класові. Логіка роботи цих функцій

не визначається. Існує лише можливість задати неформальний (наприклад, природною мовою) опис того, що від них потрібно.

Клас підтримує (або реалізує) інтерфейс, якщо він містить методи, що реалізують усі операції інтерфейсу. Між двома інтерфейсами можна задати відношення спадкування. Воно означатиме звичайне теоретико-множинне об'єднання списків операцій предка і нащадка.

На діаграмі класів UML інтерфейс можна зобразити двома способами: розгорнутим і скороченим. Якщо спосіб розгорнутий, інтерфейс зображується на діаграмі як клас зі стереотипом "interface" і без секції атрибутів (рис. 2.24). Припустиме також скорочене зображення інтерфейсу – невеликий кружок з ім'ям інтерфейсу біля нього.

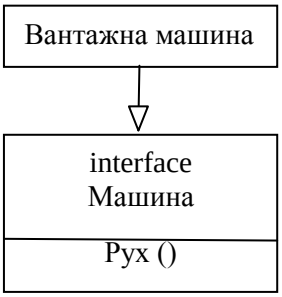


Рис. 2.24. Клас, що реалізує інтерфейс

На рис. 2.24 зображено клас *Вантажна машина,* що реалізує інтерфейс Машина. Зв'язок між ними називається *деталізація* і подається на діаграмі у вигляді пунктирної лінії з порожнім трикутником на кінці. Таким чином, клас *Вантажна машина* має надати метод, що реалізує операцію *Рух*, успадкований від інтерфейсу *Машина*.

На рис. 2.25 зображено клас Вантажна машина, що використовує інтерфейс Машина. Зв'язок між ними називається *залежністю* і зображується на діаграмі у вигляді пунктирної лінії зі стрілкою на кінці. Тобто якщо інтерфейс класу Машина змінити, то клас Вантажна машина теж може зазнати змін. Тому під час конструювання діаграм необхідно зводити число залежностей до мінімуму, щоб уникнути впливу внесених змін (докладніше про залежності буде сказано нижче).

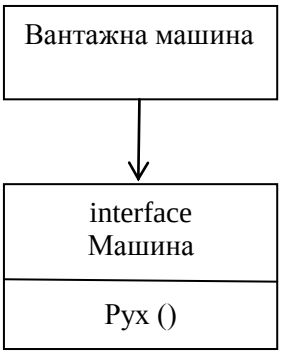


Рис. 2.25. Використання інтерфейсу класом

У деяких випадках у моделі є класи зі схожою структурою, що відрізняються деякими параметрами. Наприклад, існує опис декількох динамічних масивів для елементів різних типів, а багато операцій над їхніми елементами збігаються. Тоді доцільно визначити таку структуру даних, щоб з її допомогою було легко одержати динамічні масиви і робити це уточненням параметрів. Для цього в UML уводиться поняття *параметризованих класів (parameterized class),* ще називають *шаблонами (template).*

Параметризований клас або *шаблон* – це опис множини класів з одним або більше невизначеним формальним параметром, тобто це родина класів, де кожен клас відрізняється значеннями цих невизначених параметрів.

Таким чином, шаблон визначається в термінах формальних параметрів, його не можна використовувати як звичайний клас, тому що його параметри мають бути прив'язані до певних значень.

Шаблон не може брати участь у більшості звичайних зв'язків між класами. Існує всього два види відношень, у яких він може брати участь – зв'язки між шаблоном і класом та спрямовані асоціації. Спрямована асоціація повинна йти від шаблона (тобто навігація в напрямі від шаблона).

Слід зазначити, що сполучення нових атрибутів і операцій в екземплярі шаблона неможливе, тому операції та атрибути екземплярів шаблонів не відображаються на діаграмі. Але іноді потрібно додати нові властивості в клас, у таких випадках треба створити новий клас, чиїм предком буде екземпляр шаблона, і далі додати потрібні операції та атрибути.

Для опису класів доводиться часто користуватися деякими глобальними функціями і змінними. Тому для зручності програмування введено таке поняття, як *утиліта (utility),* клас спеціального виду, в якому збираються подібні функції і змінні. На діаграмі утиліта зображується як клас зі стереотипом "utility" і може мати як атрибути, так і операції.

Аналогічно до ключового поняття моделі класів – поняття асоціації – для об'єктів існує поняття *зв'язку (link).*

Зв'язок – екземпляр асоціації, встановленої для об'єктів даних класів. Бінарний зв'язок подається як суцільна лінія між двома об'єктами. У випадку рефлексивної асоціації зв'язок може з'єднувати об'єкт сам із собою.

Об'єкти – партнери зв'язку виконують певні ролі, імена яких зображуються на відповідних кінцях зв'язку. На відміну від асоціації, зв'язок не має власного імені, а характеризується іменами об'єктів, які він з'єднує. Оскільки зв'язки – це екземпляри асоціацій, то для них не вказується множинність. Інші властивості асоціацій, такі як агрегація, композиція, навігація, може бути показано на ролях зв'язків аналогічно. Також можна зазначити і кваліфікатор, що задає тип зв'язку (рис. 2.26).

Є такі види кваліфікаторів:

"association" – задає тип зв'язку як екземпляр асоціації, що з'єднує відповідні класи. Оскільки всі зв'язки є екземплярами асоціації, то зазначати цей кваліфікатор не має особливого значення, тому що відповідний йому тип зв'язку виставляється за замовчуванням;

“parameter” – цей кваліфікатор показує, що об’єкт є параметром операції іншого об’єкта – партнера зв’язку;

“local” – показує, що об’єкт – це локальний параметр операції або методу іншого об’єкта – партнера зв’язку;

“global” – аналогічний попередньому, тільки тут – глобальний параметр;

“self” – застосовується для позначення зв’язку об’єкта із самим собою. Використовується, наприклад, для позначення можливості посилання об’єктом повідомлень самому собі.

N-арний зв’язок зображується на діаграмі як ромб, від якого виходять з’єднання до об’єктів.

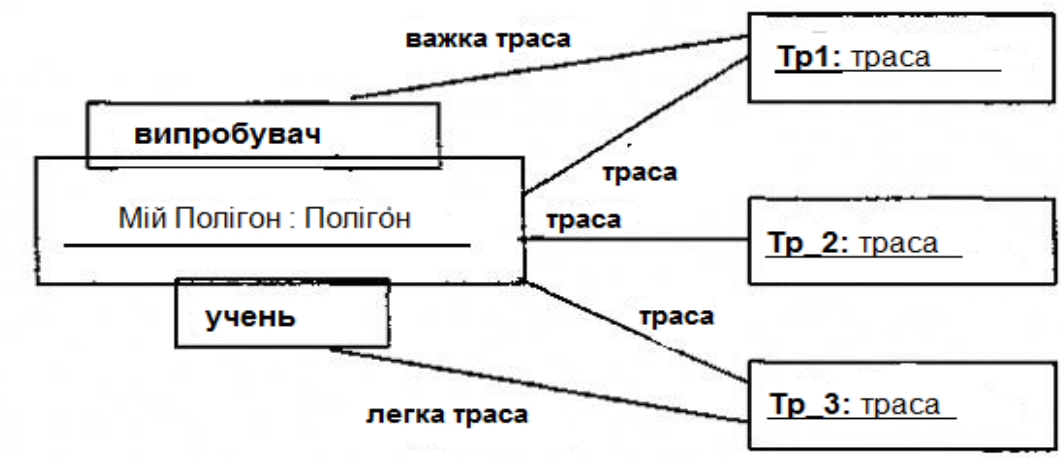


Рис. 2.26. Зв’язки

2.4.8. Використання пакетів

У контексті діаграм класів *пакет (package)* – це сховище для деякого та набору класів інших пакетів. Пакет є самостійним простором імен.

У UML немає будь-яких обмежень на правила, за якими розробники можуть або повинні групувати класи в пакети. Є деякі стандартні випадки, коли таке групування доречне, наприклад: класи, що тісно взаємодіють, або більш загальний випадок – розбиття системи на підсистеми.

Зазначимо, що пакет фізично містить сутності, визначені в ньому (тобто “сутності належать пакетові”). Тобто коли буде знищений пакет, то зникне і весь його вміст.

Описуючи класи пакета, корисно послатися на клас, визначений в іншому пакеті. Це можна зробити, імпортувавши потрібний пакет, тоді в пакеті, що імпортує, стануть доступними всі класи імпортованого пакета. При цьому простори імен не об’єднаються: для використання класу треба буде зазначити його ім’я з повним шляхом пакета, в якому він лежить.

На рис. 2.27 показано, що пакет з ім’ям “Траса” імпортує пакет з ім’ям “Підйом”.

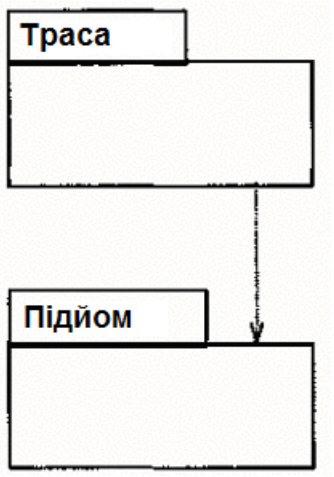


Рис. 2.27. Імпортування пакета

2.4.9. Діаграми взаємодії

Взаємодії між об'єктами в системі зображуються *діаграми взаємодії (interaction diagrams).* Діаграми взаємодії діляться на два основних типи: *діаграми послідовності (sequence diagrams)* і *кооперативні діаграми (collaboration diagrams).*

Діаграма взаємодії ілюструє процес обміну повідомленнями між екземплярами (класами) у моделі класів. Відправною точкою такої взаємодії є задоволення постумов в описах операцій.

У мові UML визначено два типи діаграм взаємодій, що можуть використовуватися для ілюстрації обміну повідомленнями.

1. Діаграми кооперації (collaboration diagram).
2. Діаграми послідовностей (sequence diagram).

На діаграмі послідовностей ілюструються події, ініційовані в системі виконавцями. Створення діаграм послідовностей – суттєва частина дослідження можливих способів побудови системи. Тому даний процес виконується в рамках створення моделі аналізу. Для ілюстрації в системі ініційованих подій можна створити діаграму послідовностей, скориставшись при цьому системою позначень, що входить до складу мови UML. Діаграми послідовностей виконуються на стадії аналізу циклу розробки. Цей процес залежить від попереднього формування прецедентів.

Діаграми послідовності мають дві розмірності: звичайно по вертикалі надано час, по горизонталі – різні об'єкти. Однак осі координат можуть мінятися місцями, так що вісь часу розташовуватиметься горизонтально, зліва направо, а список об'єктів – вертикально.

Об'єкт на діаграмі зображується у вигляді прямокутника на вершині вертикальної пунктирної лінії, що називається *лінією життя об'єкта,* це фрагмент життєвого циклу об'єкта в процесі взаємодії. Якщо об'єкт створюється або знищується на відрізку часу, зображеному на діаграмі, то його лінія життя починається і закінчується у відповідних точках, у протилежному разі лінія життя об'єкта проводиться від початку до кінця діаграми.

Символ об’єкта зображується на початку його лінії життя; якщо об’єкт створюється не на початку діаграми, то повідомлення про створення об’єкта малюється зі стрілкою, проведеної до символу об’єкта. Якщо об’єкт знищується не наприкінці діаграми, то момент його знищення позначається великим хрестиком “×”.

Лінія життя може розгалужуватися на дві (і більше) рівнобіжні лінії, показані умовно. Кожна лінія, що відгалужується, відповідає переходові в потоці повідомлень. Лінії життя можуть поєднуватися у подальшому.

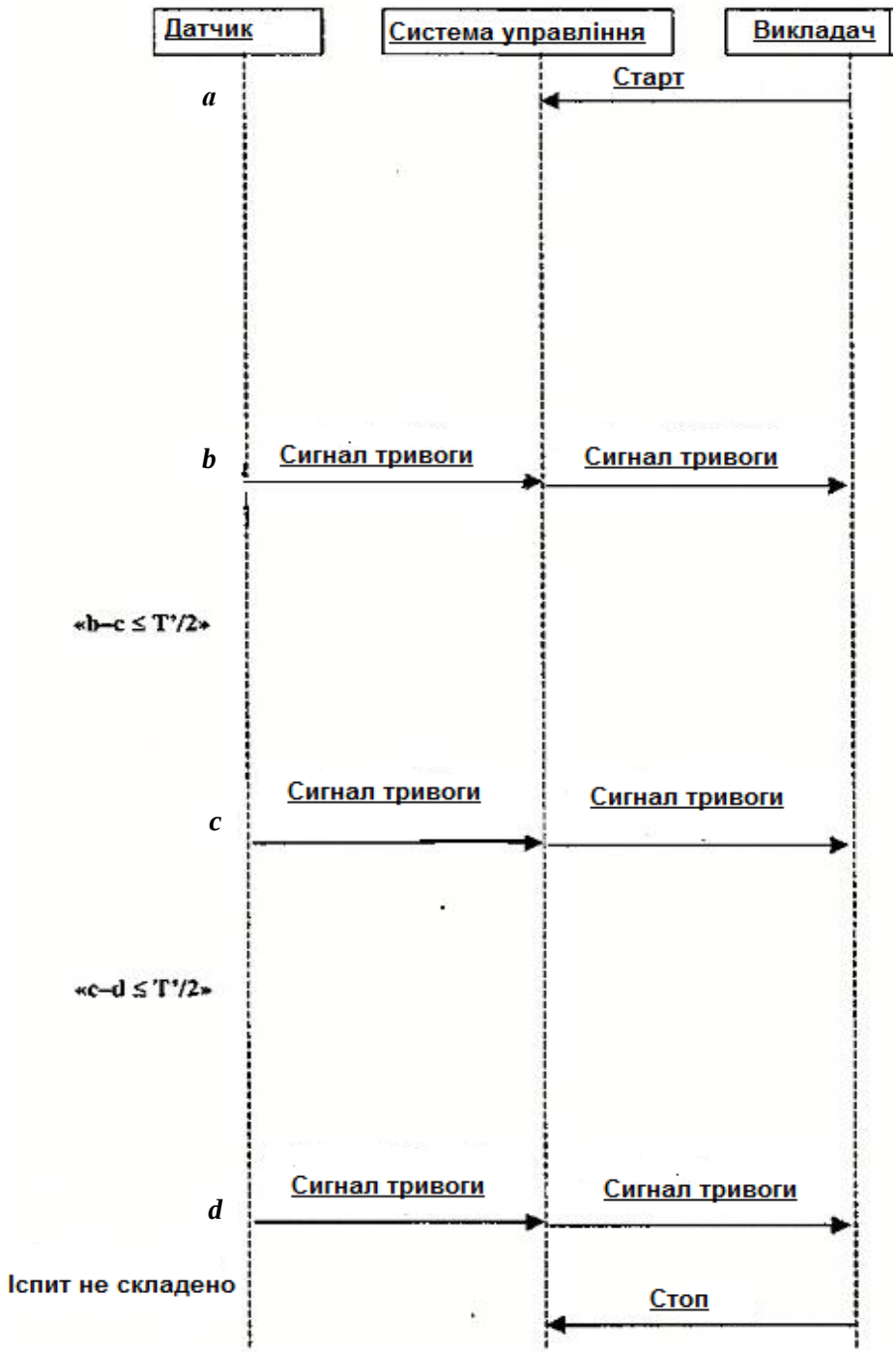


Рис. 2.28. Діаграма послідовності

Для того щоб проілюструвати даний вид діаграм, звернемося до нашого прикладу. Нехай учень складає іспит з керування автомобілем. У цьому випадку шлях учневі не підбирається, і він їде по незнайомій трасі. Іспит складено, якщо протягом відрізка часу *K* учень одержить не більше двох сигналів тривоги (не важливо, з якого приводу – перевищення рівня H_{max} чи під час тряски, що перевищує якийсь граничний рівень). Якщо ж між сигналами проходить відрізок часу більше *T*/2, то попередній сигнал анулюється. Побудовано діаграму послідовності для однієї із ситуацій, коли іспит не складено (рис. 2.28).

Повідомлення (message) пов'язують на діаграмі послідовності об'єкти між собою і передають інформацію про дію, що виконується.

Кожне повідомлення зображується на діаграмі суцільною лінією зі стрілкою на кінці, проведеною від лінії життя одного об'єкта до лінії життя іншого об'єкта. Можливе посилання повідомлення об'єктом самому собі – *самоделегування.* У цьому разі лінія може починатися і закінчуватися біля символу об'єкта. Лінія позначається ім'ям повідомлення (операція або сигнал) і значеннями його аргументів. Повідомлення можуть позначатися умовою переходу, що розташовується у квадратних дужках.

Наведемо деякі типи повідомлень:

– *асинхронні* повідомлення зображуються лінією з половинкою стрілки на кінці. Вони не блокують роботу зухвалого об'єкта, і він, таким чином, може продовжувати свій власний процес. Асинхронні повідомлення можна використовувати для створення нового об'єкта або для встановлення зв'язку вже зображується;

– *виклик процедури* малюється як заповнена стрілка. Повернення з процедури і на діаграмі звичайно не відображається. Воно позначається чітко тоді, коли це необхідно для більшої ясності, міткою (коротка поперечна лінія), розташованою біля адресата повернення.

Зазвичай стрілка з повідомленням зображується горизонтально. Це символізує, що повідомлення передається миттєво і нічого не може відбутися в момент передачі. Якщо на передачу повідомлення необхідний час, протягом коли може що-небудь статися (наприклад, надсилання повідомлення в протилежному напрямку), то лінія зі стрілкою може бути ламаною (кінець стрілки розташовується нижче її початку).

Об'єднаний набір повідомлень можна маркірувати як *ітерацію (iteration).* Маркою ітерації служить символ *. Для сценарію ітерація показує, що множина повідомлень може передаватися багаторазово. Для процедури умова продовження ітерації може зазначатися наприкінці ітерації. Можливі випадки, коли частина повідомлень є частиною ітерації, а інші повідомлення можуть бути виконані тільки одноразово.

Переходи (transition) зображуються у вигляді численних стрілок, проведених в одну точку, позначених умовою переходу. Перехід може іменуватися. Ім'я являє собою час посилки повідомлення (наприклад: A). У випадку, коли передача повідомлення відбувається не миттєво, час одер-

жання позначається ім'ям з апострофом (наприклад: A). Ім'я можна проставляти ліворуч від стрілки. Ім'я використовують для виразу, що обмежує час посилки повідомлень. Обмеження розташовуються у фігурних дужках.

Діаграми послідовності корисні для зображення паралельних процесів. Для цього в діаграмах послідовності вводяться *активації (activation),* що показують період часу, протягом якого об'єкт виконує дії безпосередньо або через залежну процедуру.

Активація зображується на діаграмі довгим тонким прямокутником, верхня частина якого вирівнюється з моментом, коли метод даного об'єкта стає активним, а нижня частина – з моментом завершення роботи даного методу. Виконувана дія позначається текстом справа від символу активації (або ліворуч, залежно від стилю), по черзі вхідні повідомлення показують дії, що виконуються в даному методі. У потоці керування процедури верхня частина символу активації попереджає про вхідне повідомлення (яке ініціює дію), а нижня є початком посилання повідомлення, що повертається.

Для паралельно працюючих об'єктів (кожний із них має власний потік керування) активація показує тривалість виконання операцій кожним об'єктом. Для процедур активація показує час, протягом якого процедура (вкладена процедура) даного об'єкта активна. У випадку рекурсивного виклику об'єкта друга активація зображується праворуч від першої, з невеликим накладанням на неї (рекурсивні виклики можуть мати довільну глибину).

Діаграми кооперації ілюструють взаємодію об'єктів у форматі графа або мережі.

Кооперативні діаграми (collaboration diagrams) дають можливість просторово розташовувати об'єкти. На відміну від діаграм послідовності, на кооперативних діаграмах екземпляри об'єктів зображуються у вигляді піктограм. На діаграмі відображаються лише об'єкти, що прямо або непрямо беруть участь у виконанні даного варіанта використання. Так само на діаграмі послідовності лінії зі стрілкою на кінці позначають повідомлення, обмін якими здійснюється в рамках даного варіанта використання. Їхня тимчасова послідовність позначається через нумерацію повідомлень.

Лінія зі стрілкою проводиться біля лінії, що з'єднує об'єкти, і показує напрям об'єкта, якому посилається повідомлення. Для позначення різних повідомлень можуть використовуватися інші типи стрілок.

- *Лінія із заповненою стрілкою.* Позначає виклик процедури. Може використовуватися також між паралельно працюючими активними об'єктами для надсилання сигналів та очікувань.
- *Лінія з половинкою стрілки.* Асинхронний потік керування. Використовується для того, щоб чітко показати асинхронний обмін повідомленнями між двома об'єктами.
- *Інші різновиди.* Можуть відображати інші різновиди керування, наприклад "balking" АБО "timeout" зазвичай сприймаються як додаткові можливості UML.

Повідомлення на кооперативній діаграмі позначаються номерами. Нумерація повідомлень, сприйняття їх послідовності важча, ніж у випадку розташування ліній на сторінці зверху вниз. Застосовують, як правило вкладену систему нумерації, тому що це дозволяє зрозуміти, яка операція що викликає, хоча при цьому може бути важче розглянути загальну послідовність.

Внутрішні повідомлення про виконання операції нумеруються, починаючи з 1. У послідовності повідомлень між об'єктами в рівнобіжних процесах нумерація повідомлень належить до одного рівня (немає вкладеності). На кооперативній діаграмі повідомлення можна позначати такою ж керівною інформацією, як і на діаграмі послідовності.

Піктограма об'єкта на кооперативній діаграмі позначається рядком імені, що має вигляд:

<Ім'я об'єкта : Ім'я класу>,

де імені об'єкта або класу може не бути. Зверніть увагу, якщо імені об'єкта немає, то перед ім'ям класу для ясності зберігається двокрапка.

Виклик взаємодії на діаграмі може бути призначено символ дійової особи.

В описаному раніше прикладі про складання учнем іспиту важлива не тільки часова послідовність подій, але й відрізки часу між ними, тому в даній задачі немає необхідності використовувати кооперативну діаграму. Тому проілюструємо даний вид діаграми на такій послідовності дій: учневі подається команда "Старт", команда передається системі керування, і починається рух. Якщо відбувається яка-небудь тривожна ситуація (удар об стелю або надлишкову тряску), то датчик сигналізує про це системі керування машини учня, а також викладачам, що приймають іспит. Система керування повинна на це відреагувати зменшенням швидкості, а викладачі – фіксацією помилки учня. Діаграма, що ілюструє даний приклад, зображена на рис. 2.29.

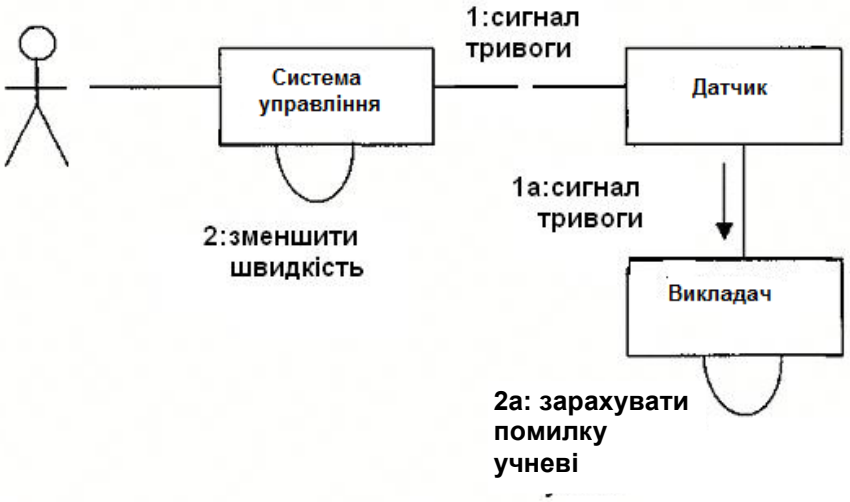


Рис. 2.29. Кооперативна діаграма

2.4.10. Діаграми станів

Діаграма станів (state diagram) визначає всі можливі стани, в яких може перебувати конкретний об'єкт, а також процес зміни станів об'єкта в результаті впливу деяких подій. Діаграми станів будуються для єдиного класу та описують поведінку єдиного об'єкта. Ця діаграма являє собою граф станів, у яких може перебувати об'єкт, і зв'язків між ними. Визначення стану і його семантика базуються на визначенні *Девіда Харела* (за винятком невеликих відмінностей).

Стан (state) являє собою відрізок часу в житті об'єкта, протягом якого є істиною деяка умова, виконуються якісь дії або очікується певна подія. Стан позначається на діаграмі як прямокутник із закругленими кутами (рис. 2.30). Він може мати одну або кілька секцій. У них розміщуються:

- *Ім'я стану.* Зазначати ім'я стану необов'язково. Два символи стану з тим самим ім'ям зображують на діаграмі той самий стан об'єкта. Використання декількох символів того самого стану використовується в діаграмі для зручності (щоб не перевантажувати один стан схожим до нього і вихідними з нього зв'язками).
- *Змінні стани.* Наводиться список змінних стану, визначених у даному стані або в його підстанах. Змінні стани мають форму атрибутів. Вираз, що описує їх початкове значення, може містити в собі атрибути даного об'єкта, змінні стани підстанів і параметри переходів у стан (якщо вони включені в усі вхідні маршрути).
- *Внутрішня поведінка.* Подається список внутрішніх дій, що виконуються, коли об'єкт перебуває в даному стані. Кожна дія описується в такий спосіб:

< ім'я події х список параметрів > V < дія >

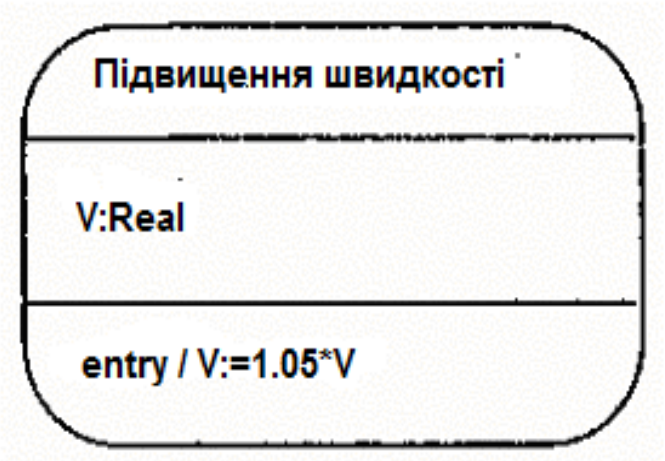


Рис. 2.30. Стан

В одному стані ім'я події може бути використано неодноразово. Однак існує три зарезервованих дії, що мають той самий формат опису, що й звичайна дія, імена яких можна використовувати тільки один раз:

- дія, виконувана під час входу в стан;
- дія, виконувана під час виходу зі стану;
- дія, виконувана під час перебування в стані.

У цих виразах можуть використовуватися змінні стани даного стану і його підстанів, атрибути даного об'єкта і параметри переходів, що входять у стан (якщо вони включені в усі вхідні маршрути). Стан має ієрархічну структуру. Кожен *підстан (substate)* – свій початковий і кінцевий псевдостан. Перехід у такий стан означає перехід у початковий псевдостан усередині нього. Перехід у кінцевий псевдостан підстану означає завершення роботи даного підстану; завершення роботи всіх підстанів означає завершення активності даного стану і вихід з нього.

Будь-який стан удосконалено введенням у нього послідовних або рівнобіжних підстанів. Його підстани так само може бути вдосконалено першим або другим способом.

Розширення стану зображують на діаграмі символом, аналогічним застосовуваному для зображення стану. У ньому, крім секцій для імені стану, змінних станів і внутрішні поведінки, є секція для вкладеної діаграми станів. Розширення стану у вигляді рівнобіжних підстанів зображується як кілька вікон, розташованих у даному стані одне під одним і розділених пунктирною лінією. Кожний підстан може мати своє ім'я і містити вкладену діаграму непересічних станів. Секції з текстовою інформацією із даним зображенням відокремлюються суцільною лінією.

Діаграми рівнобіжних підстанів використовуються в тих випадках, коли об'єкт має набір незалежних поведінок. Потрапивши в такий складний стан, що містить у собі невелику кількість паралельно функціонуючих підстанів, об'єкт починає перебувати в декількох незалежних станах одночасно. Виходячи з цих рівнобіжних станів, він опиняється в одному загальному кінцевому стані (якщо, звичайно, не залишив один з рівнобіжних підстанів раніше).

Початковий псевдостан зображується маленьким чорним кружком. Перехід з початкового псевдостану може позначатися ім'ям події – це свідчить про перехід в активний стан, викликаний даною подією. Якщо такої позначки немає, то відбувається просто перехід до активного стану. Перехід так само може мати дія, що виконується.

Кінцевий псевдостан зображується маленьким чорним кружком, обведеним суцільною лінією. Нехай у нашому прикладі з трасою під час складання іспиту передбачається ситуація з проколом шини: на трасі існують спеціальні з'їзди з траси, на які відбуксовують автомобіль із проколотою шиною. Там шину замінюють протягом часу T і автомобіль повертається на трасу. Процес заміни проколотої шини зручно подати як діаграму станів з послідовними підстанами (рис. 2.31).

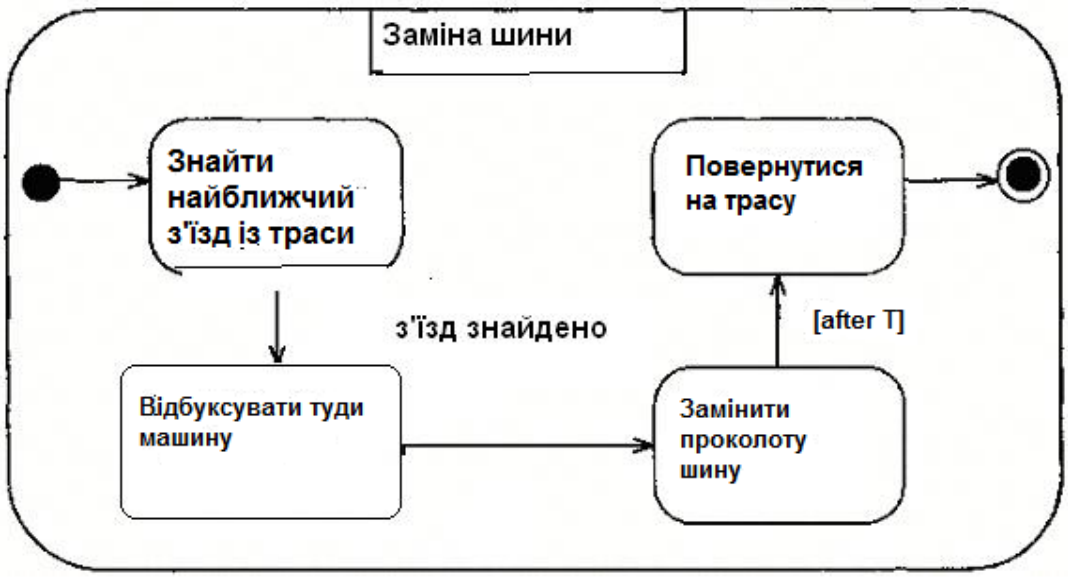


Рис. 2.31. Діаграма станів з послідовними підстанами

Припустимо, поки відбувається заміна проколотої шини, необхідно заповнити на неї квитанцію. Причому процес заповнення квитанції та її підписання відбувається одночасно із заміною шини.

Подією (event) називається подія, що заслуговує на увагу. У діаграмі станів воно може викликати перехід з одного стану в інший. Події можуть бути різних видів:

- зазначена умова, звичайно описана булевим виразом, стає істинною. Описується умовою переходу без визначення імені події;
- одержання одним об'єктом сигналу від іншого об'єкта. Описується ім'ям події, що викликає перехід;
- закінчення певного проміжку часу після означеної події. Описується часовим інтервалом, після закінчення якого викликається перехід.

Сигнал або виклик події може бути визначено в такий спосіб:

<ім'я події> (<параметр>)

Параметри мають такий формат:

<ім'я параметра> <тип>

Сигнал подано як клас зі стереотипом "signal" на діаграмі класів. Параметри будуть у цьому випадку атрибутами класу. Сигнал також визначається як підклас іншого сигналу (рис. 2.32).

Подія, пов'язана із закінченням проміжку часу, описується виразом, у якому зазначається даний відрізок модельного часу, наприклад "5 seconds". За замовчуванням, після закінчення даного відрізка часу відбувається вихід з поточного стану. У протилежному випадку подібні події описують умовним виразом, наприклад:

[date = Jan. I, 2000] або [10 seconds since exit from state A].

Події можуть зображені на діаграмі класів як клас зі стереотипом "event".

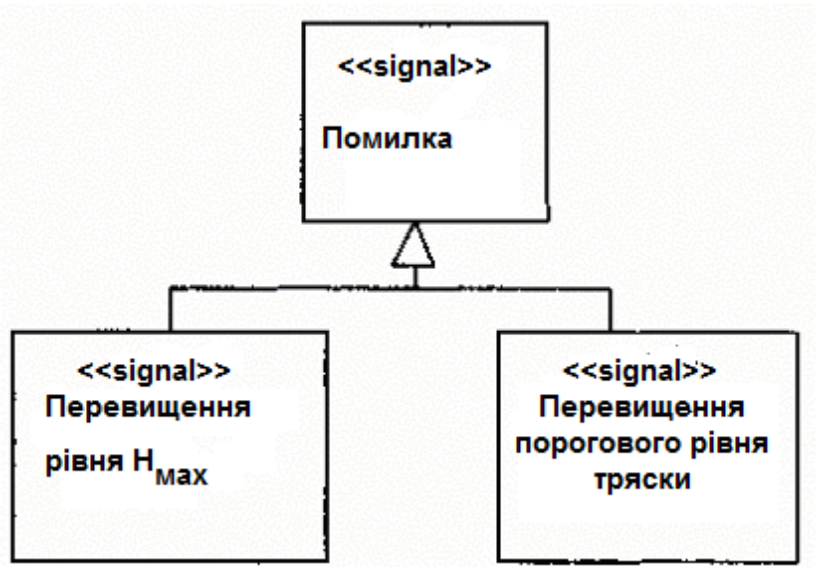


Рис. 2.32. Оголошення сигналу

Простий перехід (simple transition) – це зв'язок між двома станами об'єкта, що показує, коли об'єкт може перейти з першого стану в другий, і певна дія, яка позначає, що виконується, якщо відбулася певна подія. Подія може мати параметри, що доступні для дій, визначених на переході, або для дій, що ініціюють наступну подію, обробляються миттєво. Якщо подія не викликає ніякого переходу, то вона просто ігнорується. Якщо викликається відразу кілька переходів, то ініціюється тільки один з них; вибір може бути недетермінованим, якщо переходи не мають пріоритетів.

Перехід на діаграмі станів зображується суцільною лінією зі стрілкою на кінці, проведеною від одного стану (вихідного) до іншого (кінцевого), позначеною рядком переходу. Даний рядок має такий формат:

<опис події> [<умовний вираз>] <дія> <надсилання повідомлення>,

де:

опис події – це назва події і її аргументи:

<ім'я події> (<параметр>);

умовний вираз – це булів вираз, що описує умову, під час виконання якої відбувається дана подія;

дія є виразом, виконуваним в ініціації даного переходу. Вона може містити в собі операції, атрибути даного об'єкта і параметри викликаної події;

посилка повідомлення визначає повідомлення (або сигнал), що посилається. Він під час виникнення даної події має такий формат:

<адресат><ім'я повідомлення>(<параметр>). Перехід може містити кілька таких речень. Порядок їхнього розташування визначає порядок виконання.

Адресат – вираз, що визначає об'єкт (або множину об'єктів), якому посилається повідомлення (сигнал); ім'я повідомлення містить у собі ім'я події, про яку посилається повідомлення (сигнал);

параметри передаються разом з даним повідомленням (сигналом) об'єктові-адресатові.

Один загальний перехід може мати багато вихідних і кінцевих станів. Це відображається синхронізацією і/або поділом керування між рівнобіжними ділянками в рівнобіжних підстанах. Якщо даний об'єкт одночасно перебуває у всіх вихідних станах даного переходу, то перехід здійснюється. Якщо умова переходу стає істинною, то перехід ініціюється і виконуються дії, що стоять на даному переході. Зазвичай усі вихідні стани повинні бути активні до того моменту, коли ініціюється *складний перехід (complex transition).*

Складний перехід зображується на діаграмі станів, зафарбованим прямокутником витягнутим у вертикальному напрямку. Від прямокутника може виходити одна або кілька ліній зі стрілками на кінці, проведених до кінцевих станів. До символу складного переходу можуть бути проведені одна або кілька ліній зі стрілками на кінці (проведені від вихідних станів). Рядок, що описує умови переходу, розташовується біля даного символу. Окремі лінії, проведені до певних станів, не можуть мати своїх рядків з описом умов переходу.

Перехід у стан зі складною структурою еквівалентний переходові в його початковий псевдостан або в кожний з початкових псевдостанів його підстанів, розташованих паралельно один до одного. Перехід у стан зі складною структурою "успадковується" кожним із вкладених підстанів (на якому рівні вкладеності вони б не перебували). Перехід на діаграмі станів, уведений за складного стану, символізує перехід у складний стан. Початковий псевдостан також може бути зображений на діаграмі. Перехід може проводитися безпосередньо до кожного з підстанів системи. Перехід на діаграмі станів, що виходить через межу складного стану, символізує перехід зі складного стану в інший. Його можна провести безпосередньо від будь-якого підстану системи до стану, що міститься поза складним станом.

У разі переходу у складний стан для кожного з початкових підстанів виконуються необхідні вхідні ("entry") дії. Під час переходу зі складного стану для кожного з кінцевих підстанів виконуються необхідні вихідні ("exit") дії.

Стан може містити в собі *індикатор попереднього стану (history state indicator), що* зображується на діаграмі станів як маленький кружок з буквою "H" усередині. Індикатор попереднього стану може мати будь-яку кількість вхідних переходів, але не може мати вихідних переходів. Якщо в ньому виконується перехід, то це означає, що об'єкт повертається в той стан, у якому перебував перед тим, як залишити даний складний стан. Необхідні вхідні ("entry") дії при цьому також виконуються.

Вкладеність у складному стані може бути схована. Перехід у внутрішній стан у підстанів або вихід з підстану подається як лінія переходу, проведена до так званих *умовних псевдостанів (stubs).* Умовні псевдостани – це невеликі вертикальні лінії, розташовані в полі складного стану. Умовні

псевдостани не можуть подавати початкові або кінцеві псевдостани підстанів.

Виконувана в об’єкті дія посилає повідомлення множини об’єктів-адресатів.

Посилання повідомлення на діаграмі зображується суцільною лінією зі стрілкою на кінці, проведеною від об’єкта-відправника до об’єкта-адресата (рис. 2.33). Стрілка позначається ім’ям події і списком її аргументів.

Посилання повідомлень між діаграмами станів подається пунктирною лінією зі стрілкою на кінці, проведеною від відправника до адресата.

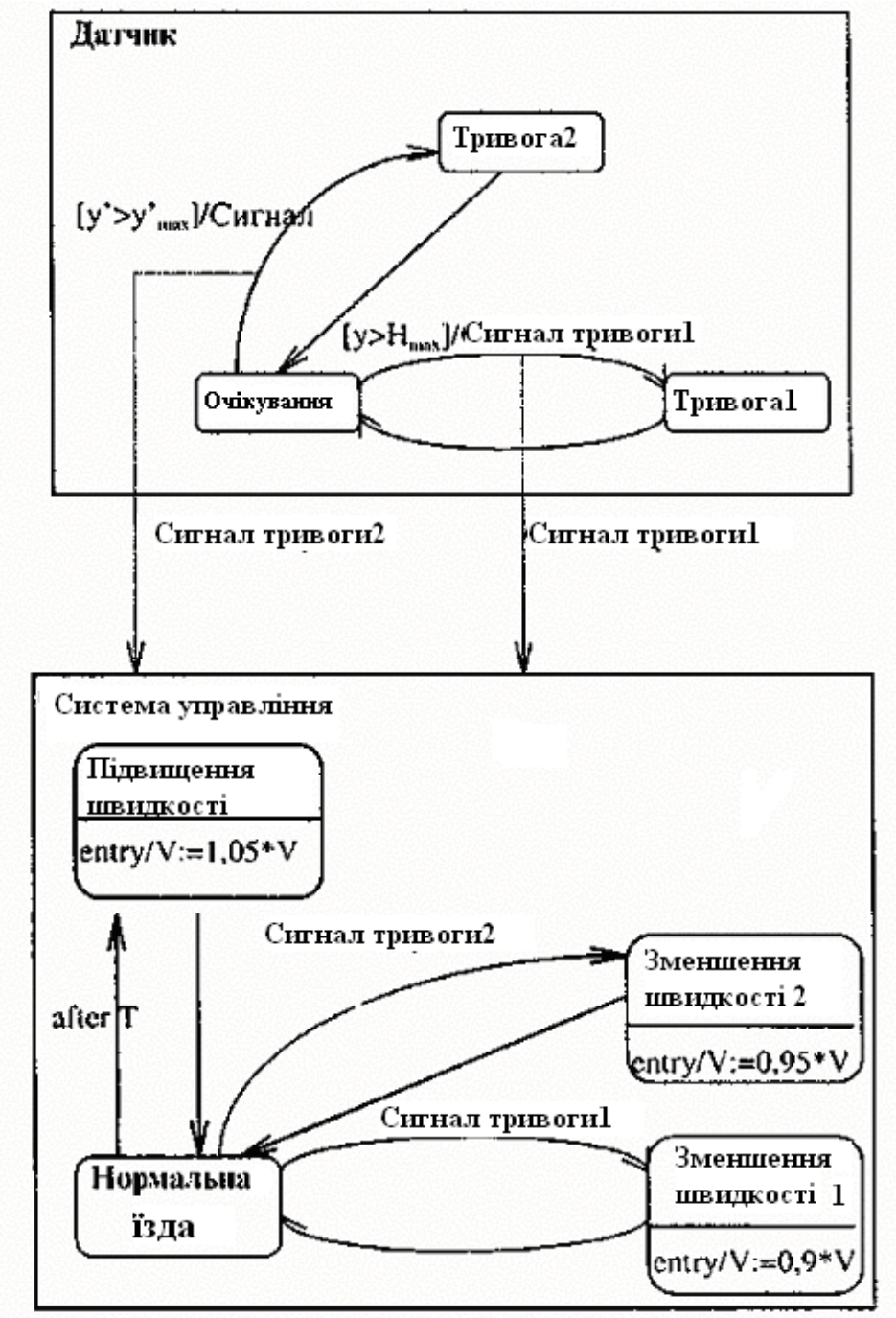


Рис. 3.33. Посилання повідомлень

2.4.11. Діаграми діяльностей

Діаграми діяльностей (activity diagrams) призначено для того, щоб відобразити виконання певного завдання, викликаного внутрішніми процесами (на противагу зовнішнім подіям). Застосовується для моделювання потоків робіт у різних варіантах для аналізу варіантів використання.

Основним елементом діаграми діяльностей є *стан дії (action state),* в якому визначена внутрішня дія, і що має хоча б один вихідний з нього перехід, який включає в себе нечітку подію завершення даної внутрішньої дії (за наявності умов переходу таких переходів може бути небагато). Стани дії не можуть мати внутрішніх або зовнішніх вихідних переходів, що ґрунтуються на явних подіях; у таких ситуаціях використовуються звичайні стани. За одним станом дії випливає інший стан, разом вони утворюють просту послідовність дій. Переходи, що виходять зі стану дії, неявно викликаються завершенням якоїсь події в стані. Переходи можуть містити в собі умови переходу і дії. Дія виконується природною мовою або на будь-якій мові програмування.

У діаграмі діяльностей використовуватиметься стан, пов'язаний з ухваленням рішення – *рішення (decision).* Рішення використовується в тих випадках, коли залежно від умов переходу може бути обраний той або інший перехід на діаграмі.

Може здаватися, що діаграма дій є аналогом блок-схеми. Це не так. Розглянемо діаграму на рис. 2.34. Знайти різницю можна, подивившись на стан дії. Він активізує дві дії, пов'язані з очікуванням помилки і спливанням відрізка часу *T'*. Припустимо, що відбулася помилка і ми стали рухатися вниз по цьому маршруту. Цей шлях веде до *лінійки синхронізації,* з яким пов'язана активізація двох діяльностей – послати повідомлення про помилку викладачам і послати повідомлення про помилку системі керування.

Діаграма показує, що ці дві діяльності можуть виконуватися паралельно і порядок їх виконання не має значення. У цьому й полягає головна відмінність між блок-схемою і діаграмою діяльностей. Блок-схеми обмежуються послідовними процесами, а діаграми діяльностей можуть підтримувати рівнобіжні процеси.

Стан дії на діаграмі зображується як прямокутник з округленими кутами. Вираз, що описує виконувану дію, розташовується всередині прямокутника. Вирази на одній діаграмі можуть дублюватися.

Розв'язок зображується на діаграмі як ромб з одним або більше вхідним у нього переходом і з одним або більше вихідним переходом.

Діаграми діяльностей відбивають події, що відбуваються, однак вони нічого не говорять про те, хто бере участь у реалізації того або іншого процесу. Один зі способів розв'язання цієї проблеми – позначати кожен стан дії міткою класу, що за нього відповідає. Можливий ще й інший спосіб – застосування так званих *плавальних доріжок (swimlines).*

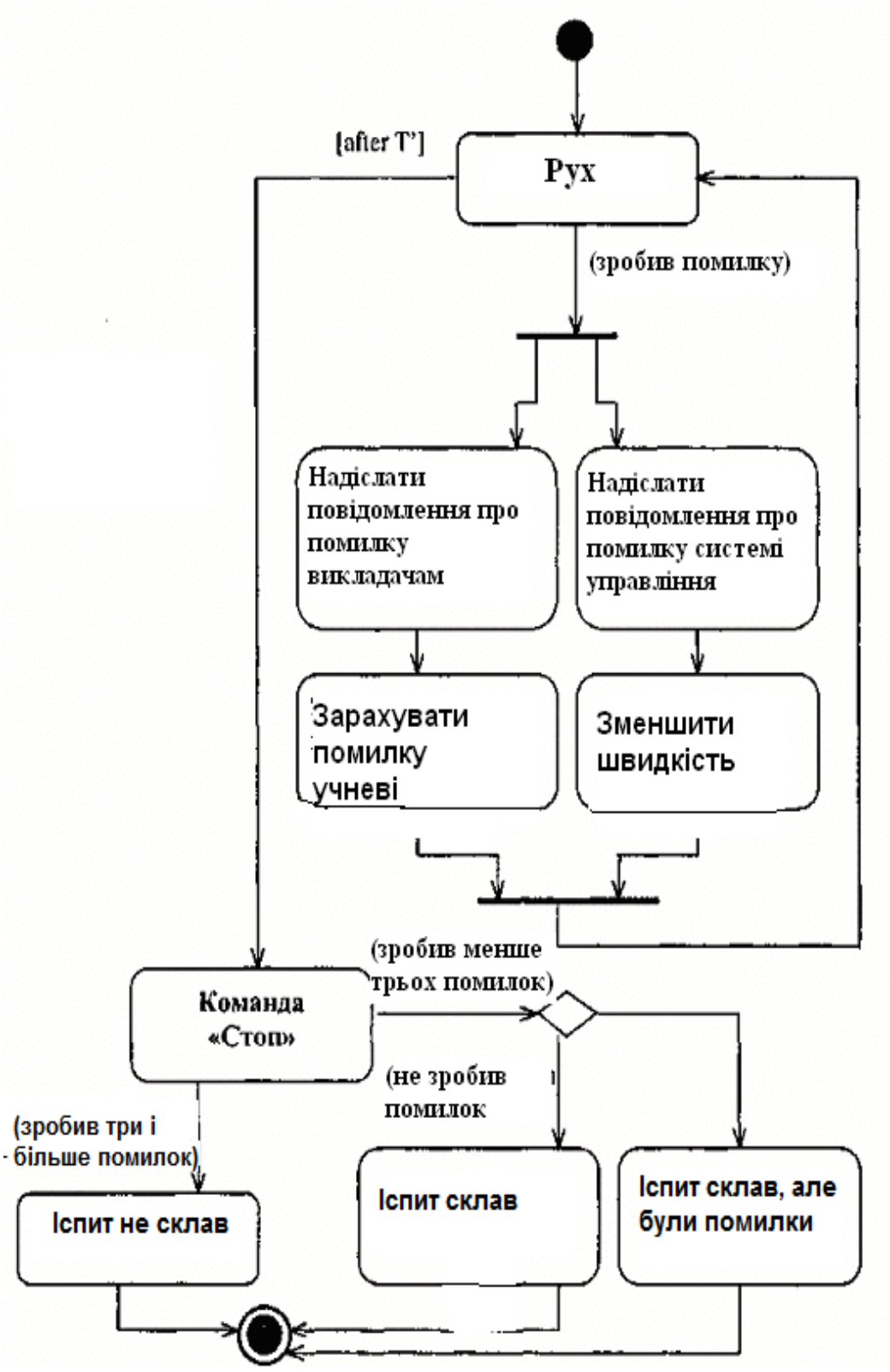


Рис. 2.34. Діаграма діяльностей

У цьому випадку діаграма діяльностей поділяється пунктирними лініями на вертикальні зони. Кожна зона являє собою зону відповідальності конкретного класу, як це зображено на рис. 2.35.

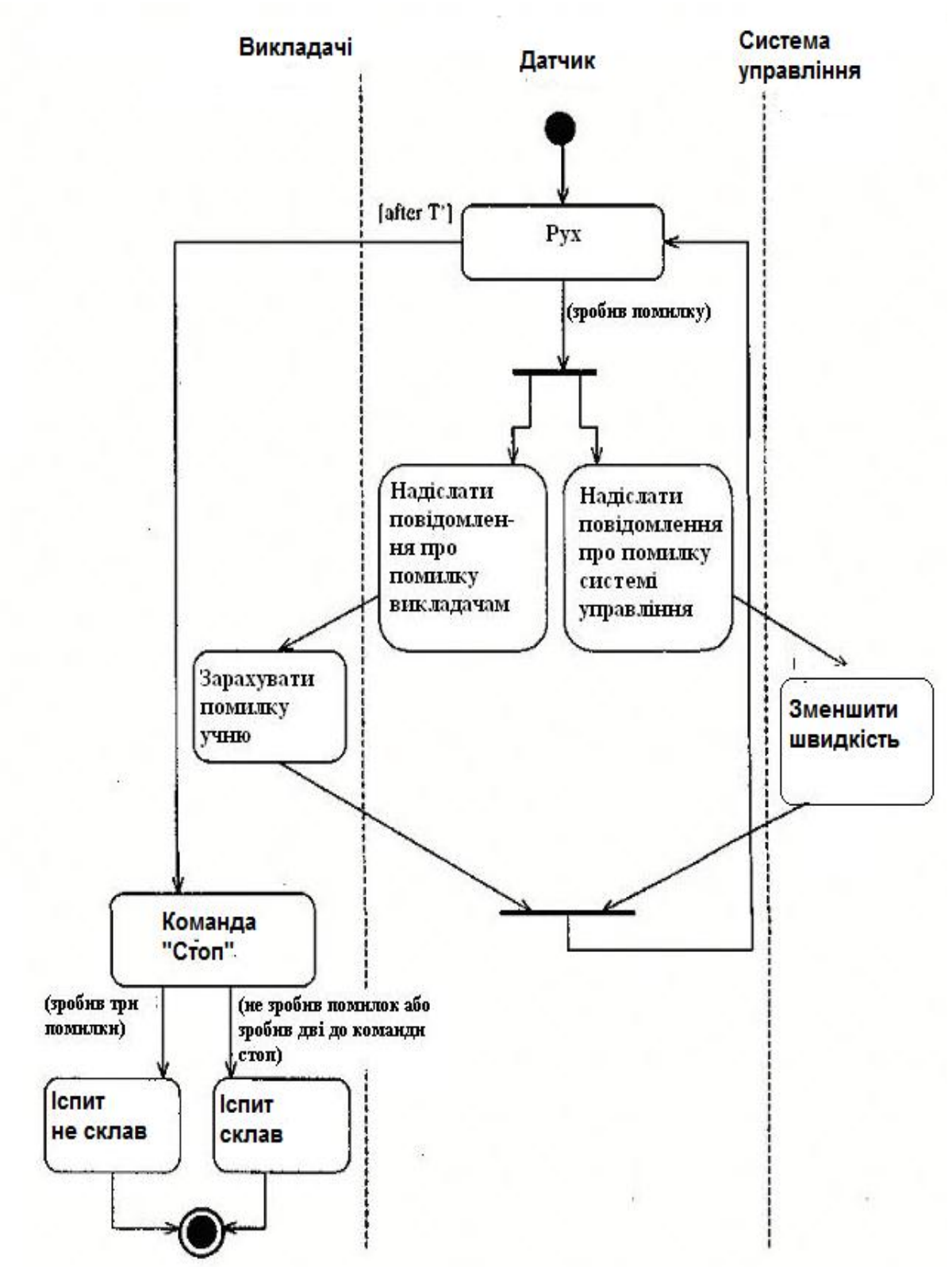


Рис. 2.35. Плавальні доріжки

Будь-який стан дії на діаграмі діяльностей може зазнати подальшої декомпозиції. Його опис можна подати як текст, код або іншу діаграму діяльностей.

Якщо діаграма діяльностей являє собою декомпозицію стану дії вищого рівня, то на такій діаграмі має бути тільки один початковий псевдостан, але може бути кілька кінцевих псевдостанів (залежно від того, скільки виходів у стану дії вищого рівня).

2.4.12. Діаграми пакетів, компонентів і розміщення

Одне з найважливіших питань методології створення програмного забезпечення – як розбити велику систему на невеликі підсистеми? Саме з цього погляду, пов'язані з переходом від структурного підходу до об'єктно-орієнтованого, є найбільш помітними. Одна з ідей полягає в групуванні класів у компоненти вищого рівня. У UML такий механізм групування має назву пакетів (package).

Діаграма пакетів містить пакети класів і залежності між ними. Строго кажучи, пакети і залежності є елементами діаграми класів, тобто діаграма пакетів – це всього лише форма діаграми класів. Проте на практиці причини побудови таких діаграм різні.

Залежність між двома елементами має місце в тому випадку, коли зміни у визначенні одного елемента можуть спричинити зміни в іншому. Що стосується класів, то причини залежностей можуть бути найрізноманітнішими: один клас посилає повідомлення іншому; один клас включає частину даних іншого класу; один клас посилається на інший як на параметр операції. Якщо клас змінює свій інтерфейс, то будь-яке повідомлення, яке він посилає, може стати неправильним.

В ідеальному випадку тільки зміни в інтерфейсі класу повинні впливати на інші класи. Мистецтво проектування великих систем включає мінімізацію залежностей, яка знижує вплив змін і потребує менших зусиль на їх внесення. Залежність між двома пакетами існує в тому випадку, якщо є якась залежність між будь-якими двома класами в пакетах.

Пакети – це життєво необхідній засіб для великих проектів. Їх слід використовувати в тих випадках, коли діаграма класів, що охоплює всю систему в цілому і розміщена на єдиному аркуші паперу формату А4, стає складною для читання. Пакети не дають відповіді на питання, яким чином можна зменшити кількість залежностей у системі, що розробляється, проте вони допомагають виділити ці залежності. Зведення до мінімуму кількості залежностей дозволяє знизити пов'язаність компонентів системи. Але евристичний підхід до цього процесу далекий від ідеалу. Пакети особливо корисні в тестуванні. Кожен пакет у тестуванні може містити один або декілька тестових класів, за допомогою яких перевіряється поведінка пакета.

Компоненти на діаграмі компонентів є фізичними модулями програмного коду (рис. 2.36). Зазвичай вони точно відповідають пакетам на діаграмі пакетів; таким чином, діаграма компонентів відбиває виконання кожного пакета в системі.

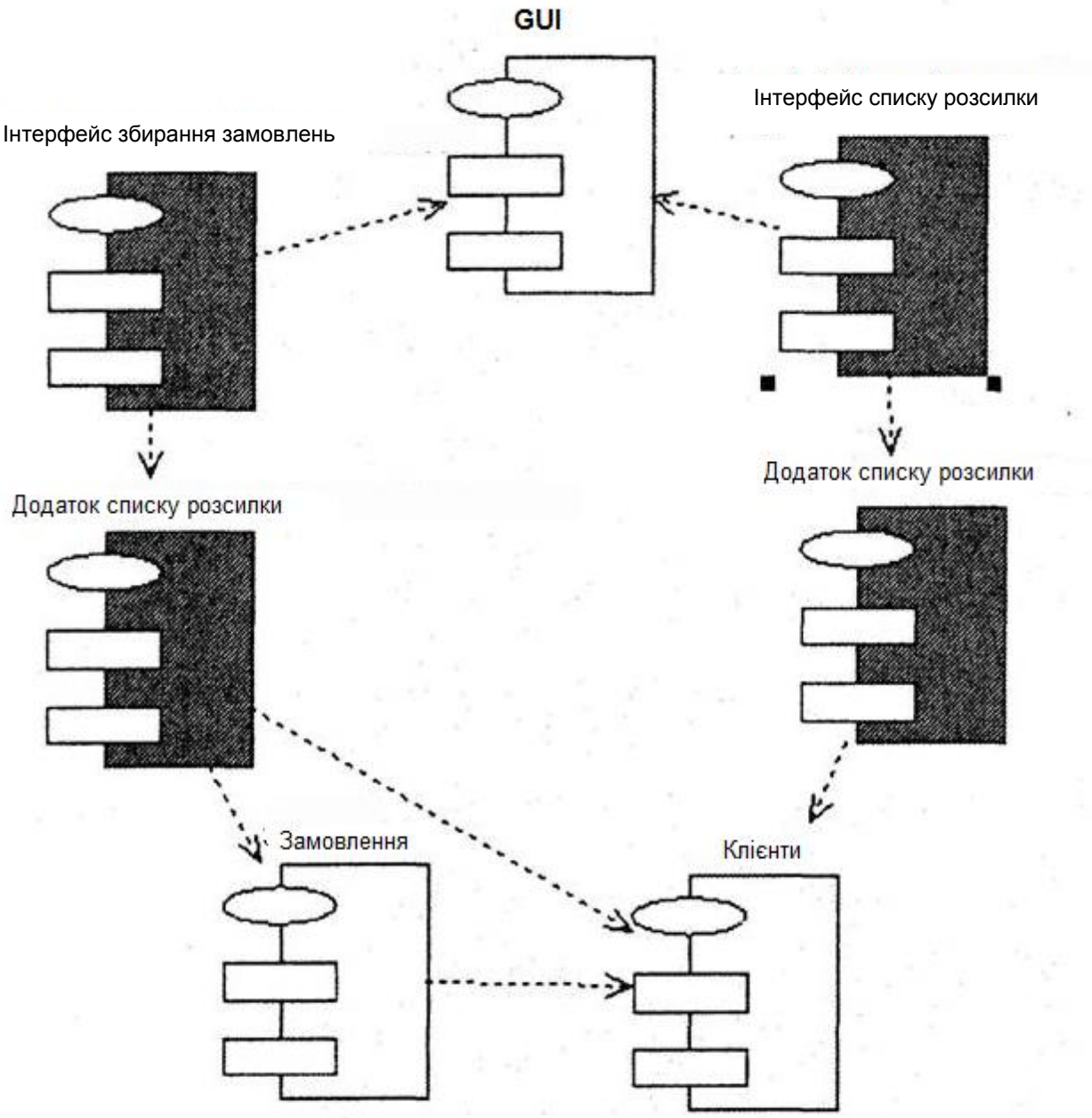


Рис. 2.36. Діаграма компонентів

Залежності між компонентами повинні збігатися із залежностями між пакетами. Ці залежності показують, яким чином одні компоненти взаємодіють з іншими. Напрям цієї залежності показує рівень обізнаності про комунікацію.

Діаграма розміщення відбиває фізичні взаємозв'язки між програмними та апаратними компонентами системи. Вона є хорошим засобом для того, щоб показати маршрути переміщення об'єктів і компонентів у розподіленій системі.

Кожен вузол на діаграмі розміщення є деяким типом обчислювального пристрою, у більшості випадків – частиною апаратури. Ця апаратура може бути простим пристроєм або датчиком і навіть великим комп'ютером.

На рис. 2.36 зображений персональний комп'ютер (ПК), пов'язаний з UNIX-сервером за допомогою протоколу TCP/IP. З'єднання між вузлами показують комунікаційні канали, за допомогою яких здійснюються системні взаємодії.

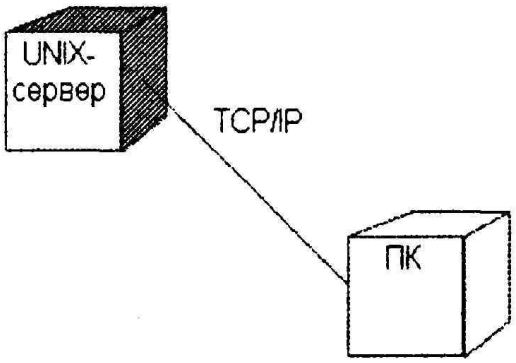


Рис. 2.36. Діаграма розміщення

На практиці такі діаграми застосовуються не надто часто. У цілому ці діаграми корисно застосовувати, щоб виділити особливі фізичні характеристики цієї системи. У міру поширення розподілених систем важливість цих діаграм зростає.

Запитання і завдання для самоконтролю

1. Які три типи моделей використовуються під час проектування?
2. Яке призначення концептуальної моделі?
3. Назвіть основний вид діаграм у концептуальній моделі.
4. Яке призначення логічної моделі?
5. Назвіть основний вид діаграм у логічній моделі.
6. Назвіть два погляди на модельовану систему в логічній моделі.
7. Яка роль діаграм взаємодії об'єктів у логічній моделі?
8. Яка роль діаграм послідовності взаємодій у логічній моделі?
9. Яке призначення фізичної моделі?
10. Назвіть основний вид діаграм у фізичній моделі.
11. У чому сенс процедури ітераційного моделювання?
12. У чому сенс варіанта використання?
13. Яке призначення діаграм варіантів використання?
14. Назвіть основні властивості варіантів використання.
15. Назвіть основні компоненти діаграм варіантів використання.
16. Що таке "дійова особа"?
17. Яку роль можуть відігравати дійові особи стосовно варіанта використання?
18. Яким чином аналіз зовнішніх подій дозволяє визначити варіанти використання системи?
19. Яке призначення діаграм класів?
20. Для чого використовується діаграма класів на стадії аналізу?

21. Для чого використовується діаграма класів на стадії проектування?
22. Назвіть основні компоненти діаграм класів.
23. Назвіть основні типи статичних зв’язків між класами.
24. Що таке асоціація?
25. У чому сенс множинності асоціацій?
26. У чому відмінність атрибутів від асоціацій?
27. Що таке ознака видимості?
28. Що таке операція класу?
29. У чому сенс узагальнення?
30. Яке призначення обмежень на діаграмах класів?
31. Яке призначення діаграм взаємодії?
32. Який зв’язок між діаграмою варіантів використання і діаграмою взаємодії?
33. Назвіть два види діаграм взаємодії.
34. Що таке “життєва лінія” на діаграмі послідовності?
35. Як на діаграмі послідовності зображуються повідомлення?
36. Що таке самоделегування?
37. Що показує активізація об’єкта?
38. У чому відмінність кооперативних діаграм від діаграм взаємодії?
39. Які переваги і недоліки кожного виду взаємодії?
40. Як відображається умовна поведінка на діаграмах взаємодії?
41. Яке призначення діаграм стану?
42. Як відображаються дії та діяльності на діаграмах стану?
43. Що таке умовний перехід і як він описується на діаграмі?
44. Які особливі стани об’єкта відображаються на діаграмі?
45. Які переваги і недоліки діаграм стану?
46. Яку проблему проектування мусять розв’язати діаграми пакетів?
47. У чому відмінність діаграм пакетів від діаграм класів?
48. У чому сенс залежності між елементами діаграми пакетів?
49. Що таке інтерфейс класу?
50. За якими ознаками класи групуються в пакети?
51. Які види елементів моделі зображено на діаграмі компонентів?
52. Як пов’язані між собою діаграми пакетів і діаграми компонентів?
53. Що показує діаграма розміщення?
54. Які сутності відображаються на діаграмах розміщення?
55. У яких випадках потрібно застосовувати діаграми розміщення?

ПРАКТИЧНІ ЗАВДАННЯ

Практичне завдання № 1

СТВОРЕННЯ ДІАГРАМИ ВАРІАНТІВ ВИКОРИСТАННЯ

Постановка завдання

Розглядається робота компанії, яка спеціалізується на виробництві та поставках продукції клієнтам. Створювана інформаційна система повинна забезпечувати можливість одержання і виконання нових замовлень, зміни старих, перевірки і поновлення інвентарних описів. Одержавши замовлення, система повинна також надсилати повідомлення бухгалтерській системі, що виписує рахунок. Якщо необхідного товару немає на складі, замовлення відхиляється.

Модель системи виконати в середовищі об'єктно-орієнтованого проектування Rational Rose.

Перетворити вимоги на діаграму Варіантів Використання, за допомогою якої почати моделювати систему.

Створення діаграми Варіантів Використання

Створіть діаграму Варіантів Використання для системи обробки замовлень. Необхідні для цього дії докладно перераховано далі. Зразок готової діаграми Варіантів Використання зображено на рис. 1.

Етапи виконання вправи

Створити діаграми Варіантів Використання (варіантів використання і Дійових осіб)

1. Двічі клацніть на Головній діаграмі Варіантів Використання (Main) у браузері, щоб відкрити її.

2. За допомогою кнопки Use Case (Варіант Використання) панелі інструментів помістіть на діаграму новий варіант використання.

3. Назвіть цей новий варіант використання "Ввести нове замовлення".

4. Повторіть етапи 2 і 3, щоб помістити на діаграму інші варіанти використання: Змінити існуючі замовлення, Надрукувати інвентарний опис, Обновити інвентарний опис, Оформити замовлення, Відхилити замовлення.

5. За допомогою кнопки Actor (Дійова особа) панелі інструментів помістіть на діаграму Нову дійову особу.

6. Назвіть її "Продавець".

7. Повторіть кроки 5 і 6, помістивши на діаграму інших дійових осіб: Керуючий магазином, Клерк магазину, Бухгалтерська система.

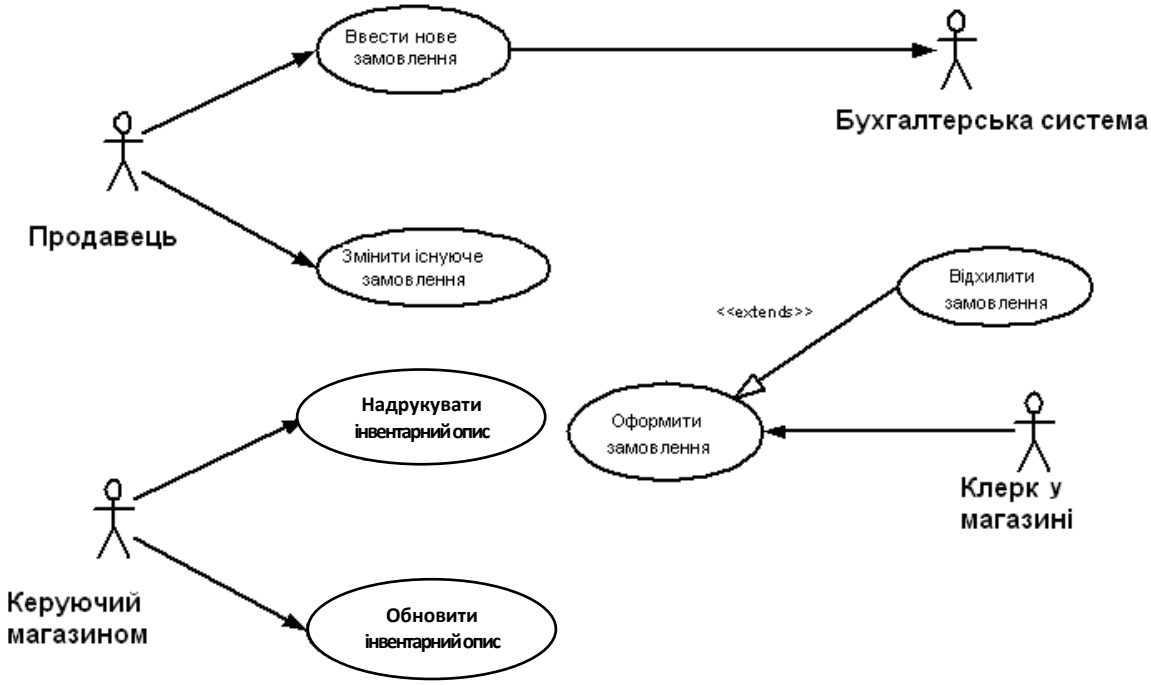


Рис. 1. Діаграма Варіантів Використання для системи обробки замовлень

Абстрактні варіанти використання

1. Клацніть правою кнопкою миші на варіанті використання "Відхилити замовлення" на діаграмі.

2. У відкритому меню виберіть пункт Open Specification (Відкрити специфікацію).

3. Позначте контрольний перемикач Abstract (Абстрактний), щоб зробити цей варіант використання абстрактним.

Додати асоціації

1. За допомогою кнопки Unidirectional Association (Односпрямована асоціація) панелі інструментів намалюйте асоціацію між дійовою особою Продавець і варіантом використання "Ввести нове замовлення".

2. Повторіть цей етап, щоб помістити на діаграму інші асоціації.

Додати зв'язок розширення

1. За допомогою кнопки Generalization панелі інструментів намалюйте зв'язок між варіантом використання "Відхилити замовлення" і варіантом використання "Оформити замовлення". Стрілка повинна простягнутися від першого варіанта використання до другого. Зв'язок розширення означає, що варіант використання "Відхилити замовлення" за необхідності доповнює функціональні можливості варіанта використання "Оформити замовлення".

2. Клацніть правою кнопкою миші на новому зв'язку між варіантами використання "Відхилити замовлення" і "Оформити замовлення".

3. У відкритому меню виберіть пункт Open Specification (Відкрити специфікацію).

4. У списку стереотипів, що розкривається, уведіть слово extends (розширення), потім натисніть ОК.

5. Слово <<extends>> з'явиться на лінії даного зв'язку.

Додати опис до варіантів використання

1. Виділіть у браузері варіант використання "Ввести нове замовлення".

2. У вікні документації Уведіть такий опис до цього варіанта використання:

Цей варіант використання дає клієнтові можливість увести нове замовлення в систему.

3. За допомогою вікна документації введіть опис до всіх інших варіантів використання.

Додати опис до дійової особи

1.Виділіть у браузері дійову особу Продавець.

2.У вікні документації Уведіть для цього дійовій особі такий опис: Продавець – це службовець, що доставляє і намагається продати продукцію.

3.За допомогою вікна документації введіть опис до дійових осіб, що залишилися.

Прикріплення файла до варіанта використання

1. Для опису головного потоку подій варіанта використання "Увести нове замовлення" створіть файл OrderFlow.doc, що містить такий текст:

1. Продавець вибирає пункт "Створити нове замовлення" з наявних меню.

2. Система виводить форму "Подробиці замовлення".

3. Продавець уводить номер замовлення, замовника і те, що замовлено.

4. Продавець зберігає замовлення.

5. Система створює нове замовлення і зберігає його в базі даних.

2. Клацніть правою кнопкою миші на варіанті використання "Ввести нове замовлення".

3. У відкритому меню виберіть пункт Open Specification (Відкрити специфікацію).

4. Перейдіть на вкладку файлів.

5. Клацніть правою кнопкою миші на білому полі і з відкритого меню виберіть пункт Insert File (Увести файл).

6. Виберіть файл OpenFlow.doc і натисніть кнопку Open (Відкрити), щоб прикріпити файл до варіанта використання.

Практичне завдання № 2
СТВОРЕННЯ ДІАГРАМ ВЗАЄМОДІЇ

У цій роботі будуть розроблені Діаграма Послідовності і Кооперативна діаграма, що описують уведення нового замовлення в систему обробки замовлень.

Постановка завдання

Проаналізуйте складові частини Діаграми Варіантів використання. Вищий пріоритет серед користувачів має варіант використання “Ввести нове замовлення”, воно ж пов’язане з найбільшим ризиком. Опис сценаріїв цього варіанта використання має такий вигляд:

1. Продавець уводить нове замовлення.
2. Продавець намагається ввести замовлення, але товару немає на складі.
3. Продавець намагається ввести замовлення, але під час його зберігання в базі даних сталася помилка.

Створіть Діаграму Послідовності і Кооперативну діаграму для сценарію “Ввести нове замовлення”.

Створення Діаграм взаємодії

Створіть Діаграму послідовності і Кооперативну діаграму, що відбиває введення нового замовлення в систему обробки замовлень. Готову Діаграму Послідовності зображено на рис. 2. Це тільки одна з діаграм, необхідних для моделювання варіанта використання “Ввести нове замовлення”. Вона відповідає успішному варіантові ходу подій. Для опису того, що, трапиться, якщо виникне помилка або якщо користувач вибере інші дії з запропонованих, доведеться розробити інші діаграми. Кожен альтернативний потік варіанта використання може бути промодельований за допомогою своїх власних Діаграм взаємодії.

Етапи виконання вправи

Настроювання

1. У меню моделі виберіть пункт Tools > Options (Інструменти > Параметри).
2. Перейдіть на вкладку діаграм.
3. Контрольні перемикачі Sequence Numbering, Collaboration Numbering і Focus of Control мають бути позначені.
4. Натисніть ОК, щоб вийти з вікна параметрів.

Створення Діаграми послідовності

1. Клацніть правою кнопкою миші на Логічному зображенні браузера.
2. У відкритому меню виберіть пункт New > Sequence Diagram.
3. Назвіть нову діаграму “Введення замовлення”.
4. Двічі клацніть на ній, щоб відкрити.

Додавання на діаграму дійової особи та об’єктів

1. Перетягніть дійову особу Продавець (Salesperson) із браузера на діаграму.
2. На панелі інструментів натисніть кнопку Object (Об’єкт).
3. Клацніть мишею у верхній частині діаграми, щоб помістити туди новий об’єкт.
4. Назвіть об’єкт “Order Options Form і Вибір варіанта замовлення”.

5. Повторіть етапи 3 і 4, щоб помістити на діаграму всі інші об’єкти:
“Order Detail Form” – Форма Деталі замовлення
“Order № 1234” – Замовлення № 1234.

Додавання повідомлень на діаграму

1. На панелі інструментів натисніть кнопку Object Message (Повідомлення об’єкта).

2. Проведіть мишею від лінії життя дійової особи Продавець до лінії життя об’єкта Вибір варіанта замовлення.

3. Виділивши повідомлення, уведіть його ім’я “Create New Order” і створіть нове замовлення.

4. Повторіть етапи 2 і 3, щоб помістити на діаграму додаткові повідомлення:

Open form і Відкрити форму (між Вибором варіанта замовлення і Деталями замовлення).

Enter order number, customer, order items і Ввести номер замовлення, замовника і кількість предметів, що замовляються (між Продавцем і Деталями замовлення).

Save the order і Зберегти замовлення (між Продавцем і Деталями замовлення).

Create new, blank order і Створити порожнє замовлення (між Деталями замовлення і Замовленням № 1234).

Set the order number, customer, order items і Ввести номер замовлення, замовника та кількість предметів, що замовляються (між Деталями замовлення і Замовленням № 1234).

Save the order і Зберегти замовлення (між Деталями замовлення Замовленням № 1234).

Рис. 2. Діаграма Послідовності введення нового замовлення після завершення першого етапу роботи

Тепер треба подбати про керівні об’єкти і взаємодію з базою даних. Як видно з діаграми, об’єкт Деталі замовлення має багато відповідальностей, з якими найкраще міг би впоратися керівний об’єкт. Крім того, нове замовлення має зберігати себе в базі даних саме. Імовірно, цей обов’язок краще було б перекласти на інший об’єкт.

Додавання на діаграму додаткових об’єктів

1. На панелі інструментів натисніть кнопку Object.
2. Клацніть мишею між об’єктами Деталі замовлення і Замовлення № 1234, щоб помістити туди новий об’єкт.
3. Уведіть ім’я об’єкта – Order Manager (Керуючий замовленнями).
4. На панелі інструментів натисніть кнопку Object.
5. Новий об’єкт розташуйте праворуч від Замовлення № 1234.
6. Уведіть його ім’я – Transaction Manager (Керуючий транзакціями).

Призначення відповідальностей об’єктам

1. Виділіть повідомлення 5 (Створити порожнє замовлення).
2. Натисніть комбінацію клавіш CTRL + D, щоб видалити це повідомлення.
3. Повторіть етапи 1 і 2, щоб видалити два останніх повідомлення:

Уведіть номер замовлення, замовника і кількість предметів, що замовляються.

Зберегти замовлення.

4. На панелі інструментів натисніть кнопку Object Message.
5. Помістіть на діаграму нове повідомлення, розташувавши його під повідомленням 4 між Деталями замовлення і Керуючим замовленнями.
6. Назвіть його Save the order (Зберегти замовлення).
7. Повторіть етапи 4–6, додавши повідомлення із шостого до дев’ятого і назвавши їх:

Create new, blank order (Створити нове замовлення) – між Керуючим замовленнями і Замовленням № 1234.

Set the order number, customer, order items (Увести номер замовлення, замовника і кількість предметів, що замовляються) – між Керуючим замовленнями і Замовленням № 1234.

Save the order (Зберегти замовлення) – між Керуючим замовленнями і Керуючим транзакціями.

Collect order information (Інформація про замовлення) – між Керуючим транзакціями і Замовленням № 1234.

8. На панелі інструментів натисніть кнопку Message to Self (Повідомлення собі).
9. Клацніть на лінії життя об’єкта Керуючий транзакціями нижче повідомлення 9, додавши туди рефлексивне повідомлення. Назвіть його Save the order information to the database (Зберегти інформацію про замовлення в базі даних).

Тепер Діаграма послідовності повинна мати вигляд, як на рис. 3.

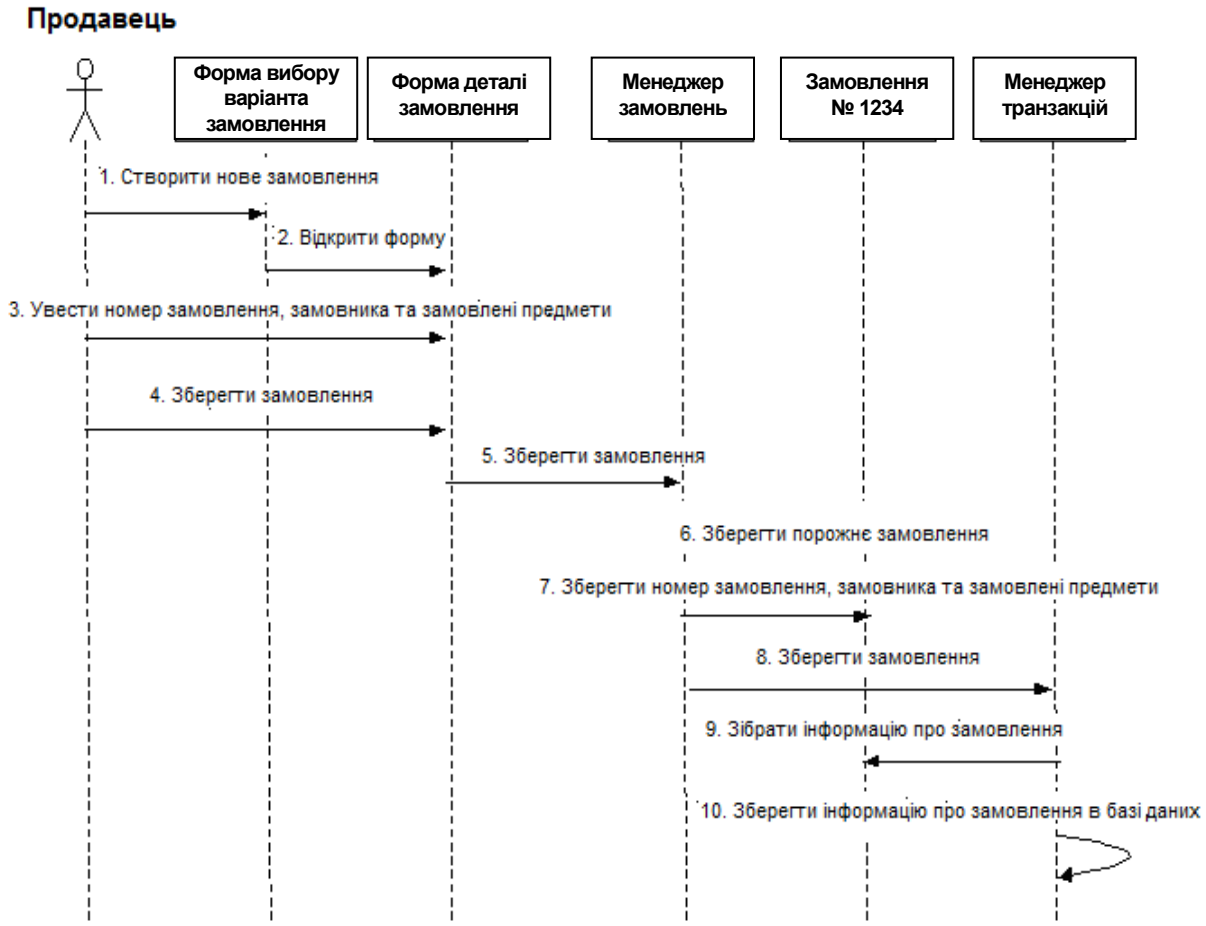


Рис. 3. Діаграма послідовності з новими об’єктами

Співвіднесення об’єктів із класами

1. Клацніть правою кнопкою миші на об’єкті Вибір варіанта замовлення.

2. У відкритому меню виберіть пункт Open Specification (Відкрити специфікацію).

3. У списку класів, що розкривається, виберіть пункт <New> (Створити). З’явиться вікно специфікації класів.

4. У полі імені Уведіть ім’я OrderOptions (Вибір замовлення).

5. Клацніть на кнопці ОК. Ви повернетеся до вікна специфікації об’єкта.

6. У списку класів виберіть тепер клас OrderOptions.

7. Клацніть на кнопці ОК, щоб повернутися до діаграми. Тепер об’єкт називається Order Options Form : OrderOptions (Вибір варіанта замовлення OrderOptions).

8. Для співвіднесення інших об’єктів із класами повторіть етапи з 1 до 7:

Клас OrderDetail співвіднесіть з об’єктом Деталі замовлення.

Клас OrderMgr – з об’єктом Керуючий замовленнями.

Клас Order – з об’єктом Замовлення № 1234.

Клас TransactionMgr – з об’єктом Керуючий транзакціями.

Після завершення цих дій діаграма міститиме імена класів.

Співвіднесення повідомлень з операціями

1. Клацніть правою кнопкою на повідомленні 1, Створити нове замовлення.

2. У відкритому меню виберіть пункт <new operation> (створити операцію). З'явиться вікно специфікації операції.

3. У полі імені Уведіть ім'я операції – Create (Створити).

4. Натисніть на кнопку ОК, щоб закрити вікно специфікації операції і повернутися на діаграму.

5. Ще раз клацніть правою кнопкою миші на повідомленні 1.

6. У відкритому меню виберіть нову операцію Create ().

7. Повторіть повідомлення з 1 до 6, поки не співвіднесете з операціями всі інші повідомлення:

Повідомлення 2: Відкрити співвіднесеність з операцією Open ().

Повідомлення 3: Увести номер замовлення, замовника і кількість предметів, що замовляються – з операцією SubmitInfo ().

Повідомлення 4: Зберегти замовлення – з операцією Save ().

Повідомлення 5: Зберегти замовлення – з операцією SaveOrder ().

Повідомлення 6: Створити порожнє замовлення – з операцією Create ().

Повідомлення 7: Увести номер замовлення, замовника і кількість предметів, що замовляються, – з операцією SetInfo ().

Повідомлення 8: Зберегти замовлення – з операцією SaveOrder ().

Повідомлення 9: Інформація про замовлення – з операцією GetInfo ().

Повідомлення 10: Зберегти інформацію про замовлення в базі даних – з операцією Commit.

Після завершення цих дій діаграма міститиме операції.

Створення Кооперативної діаграми

Для створення Кооперативної діаграми досить просто натиснути клавішу F5 або, якщо ви бажаєте самі проробити всі необхідні операції, скористайтеся планом, що далі наводиться.

1. Клацніть правою кнопкою миші на Логічному зображенні в браузері.

2. У відкритому меню виберіть пункт New > Collaboration Diagram.

3. Назвіть цю діаграму Введення замовлення.

4. Клацніть на ній двічі, щоб відкрити.

Додавання дійової особи та об'єктів на діаграму

1. Перетягніть дійову особу Продавець (Salesperson) із браузера на діа-граму.

2. На панелі інструментів натисніть кнопку Object (Об'єкт).

3. Клацніть мишею де-небудь усередині діаграми, щоб помістити туди новий об'єкт.

4. Назвіть об'єкт "Order Options Form" і Вибір варіанта замовлення.

5. Повторіть етапи 3 і 4, щоб помістити на діаграму всі інші об'єкти:

"Order Detail Form" – Форма Деталі замовлення

"Order № 1234" – Замовлення № 1234.

Додавання повідомлень на діаграму

1. На панелі інструментів натисніть кнопку Object Link (Зв'язок об'єкта).

2. Проведіть мишею від дійової особи Продавець до об'єкта Вибір варіанта замовлення.

3. Повторіть етапи 1 і 2, з'єднавши зв'язками такі об'єкти:

Дійова особа Продавець і об'єкт Деталі Замовлення.

Об'єкт Вибір варіанта замовлення та об'єкт Деталі замовлення.

Об'єкт Деталі замовлення та об'єкт Замовлення № 1234.

4. На панелі інструментів натисніть кнопку Link Message (Повідомлення зв'язку).

5. Клацніть на зв'язку між Продавцем і Вибором варіанта замовлення.

6. Виділивши повідомлення, уведіть його ім'я "Create New Order і Створити нове замовлення".

7. Повторіть етапи з 4 до 6, помістивши на діаграму всі інші повідомлення, як показано нижче:

Open form і Відкрити форму (між Вибором варіанта замовлення і Деталями замовлення).

Enter order number, customer, order items і Ввести номер замовлення, замовника та кількість предметів, що замовляються (між Продавцем і Деталями замовлення).

Save the order і Зберегти замовлення (між Продавцем і Деталями замовлення).

Create new, blank order і Створити порожнє замовлення (між Деталями замовлення і Замовленням № 1234).

Set the order number, customer, order items і Ввести номер замовлення, замовника та кількість предметів, що замовляються (між Деталями замовлення і Замовленням № 1234).

Save the order і Зберегти замовлення (між Деталями замовлення і Замовленням № 1234).

Тепер, як і раніше, треба продовжити роботу і помістити на діаграму додаткові елементи, а також розглянути відповідальності об'єктів.

Додавання на діаграму додаткових об'єктів

1. На панелі інструментів натисніть кнопку Object.

2. Клацніть мишею де-небудь на діаграмі, щоб помістити туди новий об'єкт.

3. Уведіть ім'я об'єкта – Order Manager (Керуючий замовленнями).

4. На панелі інструментів натисніть кнопку Object.

5. Помістіть на діаграму ще один об'єкт.

6. Уведіть його ім'я – Transaction Manager (Керуючий транзакціями).

Призначення відповідальностей об'єктам

1. Виділіть повідомлення 5 (Створити порожнє замовлення). Виділяйте слова, а не стрілки.

2. Натисніть комбінацію клавіш CTRL + D, щоб видалити це повідомлення.

3. Повторіть етапи 1 і 2, щоб видалити повідомлення 6 і 7:

Увести номер замовлення, замовника і кількість предметів, що замовляються

Зберегти замовлення.

4. Виділіть зв’язок між об’єктами Деталі замовлення і Замовлення № 1234.

5. Натисніть комбінацію клавіш CTRL + D, щоб видалити цей зв’язок.

6. На панелі інструментів натисніть кнопку Object Link (Зв’язок об’єкта).

7. Намалюйте зв’язок між Деталями Замовлення і Керуючим замовленнями.

8. На панелі інструментів натисніть кнопку Object Link (Зв’язок об’єкта).

9. Намалюйте зв’язок між Керуючим замовленнями і Замовленням № 1234.

10. На панелі інструментів натисніть кнопку Object Link (Зв’язок об’єкта).

11. Намалюйте зв’язок між Замовленням № 1234 і Керуючим транзакцій.

12. На панелі інструментів натисніть кнопку Object Link (Зв’язок об’єкта).

13. Намалюйте зв’язок між Керуючим замовленнями і Керуючим транзакцій.

14. На панелі інструментів натисніть кнопку Link Message (Повідомлення зв’язку).

15. Клацніть на зв’язку між об’єктами Деталі замовлення і Керуючими замовленнями, щоб увести нове повідомлення.

16. Назвіть це повідомлення Save the order (Зберегти замовлення).

17. Повторіть етапи 14–16, додавши повідомлення із шостого до дев’ятого і назвавши їх:

Create new, blank order (Створити нове замовлення) – між Керуючими замовленнями і Замовленням № 1234.

Set the order number, customer, order items (Увести номер замовлення, замовника і кількість предметів, що замовляються) – між Керуючим замовленнями і Замовленням № 1234.

Save the order (Зберегти замовлення) – між Керуючим замовленнями і Керуючим транзакціями.

Collect order information (Інформація про замовлення) – між Керуючої транзакціями і Замовленням № 1234.

18. На панелі інструментів натисніть кнопку Message to Self (Повідомлення собі).

19. Клацніть на об’єкті Керуючий транзакціями, додавши до нього рефлексивне повідомлення.

20. На панелі інструментів натисніть кнопку Link Message (Повідомлення зв’язку).

21. Клацніть на рефлексивному зв’язку Керуючого транзакціями, щоб увести туди повідомлення.

22. Назвіть нове повідомлення Save the order information to the database (Зберегти інформацію про замовлення в базі даних).

Співвіднесення об’єктів із класами
(якщо під час розробки описаної вище діаграми Послідовності самі класи ви вже створили)

1. Знайдіть у браузері клас OrderOptions.
2. Перетягніть його на об’єкт Вибір варіанта замовлення на діаграмі.
3. Повторіть етапи 1 і 2, співвідносячи інші об’єкти і відповідні їм класи:

Клас OrderDetail співвіднесіть з об’єктом Деталі замовлення.
Клас OrderMgr – з об’єктом Керуючий замовленнями.
Клас Order – з об’єктом Замовлення № 1234.
Клас TransactionMgr – з об’єктом Керуючий транзакціями.

Співвіднесення об’єктів із класами
(якщо ви не створювали описану вище діаграму Послідовності)

1. Клацніть правою кнопкою миші на об’єкті Вибір варіанта замовлення.
2. У відкритому меню виберіть пункт Open Specification (Відкрити специфікацію).
3. У списку класів, що розкривається, виберіть пункт <New> (Створити). З’явиться вікно специфікації класів.
4. У полі імені Уведіть ім’я Order Options (Вибір замовлення).
5. Клацніть на кнопці ОК. Ви повернетеся до вікна специфікації об’єкта.
6. У списку класів виберіть тепер клас Order Options.
7. Клацніть на кнопці ОК, щоб повернутися до діаграми. Тепер об’єкт називається Order Options Form : OrderOptions (Вибір варіанта замовлення : OrderOptions).
8. Для співвіднесення інших об’єктів із класами повторіть етапи з 1 до 7:

Клас OrderDetail співвіднесіть з об’єктом Деталі замовлення.
Клас OrderMgr – з об’єктом Керуючий замовленнями.
Клас Order – з об’єктом Замовлення № 1234.
Клас TransactionMgr – з об’єктом Керуючий транзакціями.

Співвіднесення повідомлень з операціями
(якщо під час розробки описаної вище Діаграми послідовності самі операції ви вже створили)

1. Клацніть правою кнопкою на повідомленні 1 Створити нове замовлення.
2. У відкритому меню виберіть пункт Open Specification (Відкрити специфікацію).
3. У списку імен, що розкривається, зазначте ім’я операції – Create (Створити).
4. Натисніть на кнопку ОК.

5. Повторіть етапи з першого до четвертого для співвіднесення з операціями інших повідомлень:

Повідомлення 2: Відкрити співвіднесеність з операцією Open ().

#Повідомлення 3: Увести номер замовлення, замовника і кількість предметів, що замовляються, – з операцією SubmitInfo ().

Повідомлення 4: Зберегти замовлення – з операцією Save ().

Повідомлення 5: Зберегти замовлення – з операцією SaveOrder ().

Повідомлення 6: Створити порожнє замовлення – з операцією Create ().

Повідомлення 7: Увести номер замовлення, замовника і кількість предметів, що замовляються, – з операцією SetInfo ().

Повідомлення 8: Зберегти замовлення – з операцією SaveOrder ().

Повідомлення 9: Інформація про замовлення – з операцією GetInfo ().

Повідомлення 10: Зберегти інформацію про замовлення в базі даних – з операцією Commit ().

Співвіднесення повідомлень з операціями (якщо ви не створювали описану вище Діаграму послідовності)

1. Клацніть правою кнопкою на повідомленні 1: Створити нове замовлення.

2. У відкритому меню виберіть пункт <new operation> (створити операцію). З'явиться вікно специфікації операції.

3. У полі імені Уведіть ім'я операції – Create (Створити).

4. Натисніть на кнопку ОК, щоб закрити вікно специфікації операції і повернутися на діаграму.

5. Ще раз клацніть правою кнопкою миші на повідомленні 1.

6. У відкритому меню виберіть пункт Open Specification (Відкрити специфікацію).

7. У списку, що розкривається, Name (ім'я) зазначте ім'я нової операції.

8. Натисніть на кнопку ОК.

Повторіть етапи з першого до восьмого, щоб створити нові операції та співвіднести з ними інші повідомлення:

Повідомлення 2: Відкрити співвіднесеність з операцією Open ().

Повідомлення 3: Увести номер замовлення, замовника і кількість предметів, що замовляються, – з операцією SubmitInfo ().

Повідомлення 4: Зберегти замовлення – з операцією Save ().

Повідомлення 5: Зберегти замовлення – з операцією SaveOrder ().

Повідомлення 6: Створити порожнє замовлення – з операцією Create ().

Повідомлення 7: Увести номер замовлення, замовника і кількість предметів, що замовляються, – з операцією SetInfo ().

Повідомлення 8: Зберегти замовлення – з операцією SaveOrder ().

Повідомлення 9: Інформація про замовлення – з операцією GetInfo ().

Повідомлення 10: Зберегти інформацію про замовлення в базі даних – з операцією Commit.

Діаграма повинна мати вигляд, як на рис. 4.

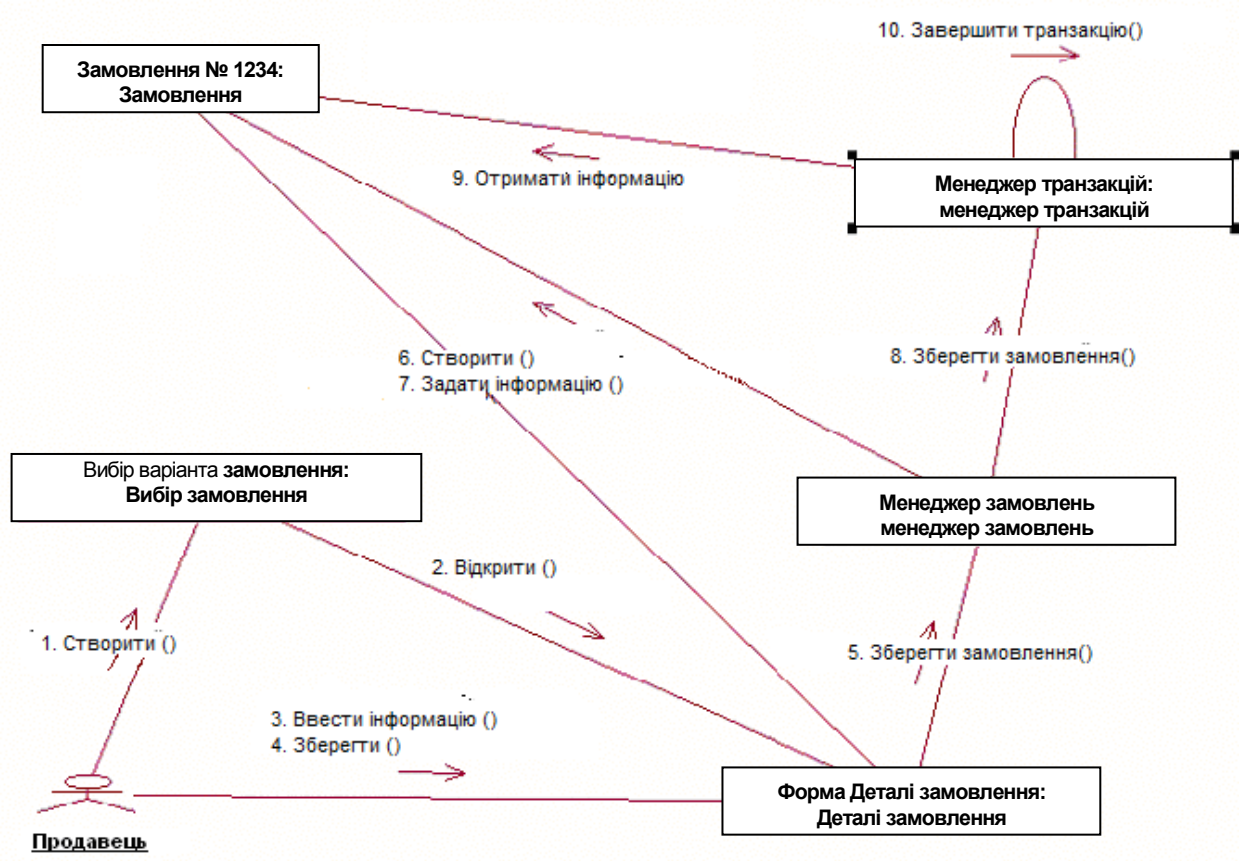


Рис. 4. Кооперативна діаграма та операції на ній

Практичне завдання № 3
СТВОРЕННЯ ДІАГРАМ КЛАСІВ

Необхідно згрупувати в пакети класи, створені під час виконання попередньої роботи. Потім необхідно створити кілька Діаграм класів, на яких також показують класи і пакети системи.

Створення Діаграми класів

Об’єднайте виявлені класи в пакети. Створіть Діаграму класів для відображення пакетів, Діаграми класів для відображення класів у кожному пакеті і Діаграму класів для відображення всіх класів варіанта використання “Ввести нове замовлення”.

Етапи виконання вправи.
Настроювання

1. У меню моделі виберіть пункт Tools > Options (Інструменти > Параметри).

2. Перейдіть на вкладку діаграм.

3. Переконайтеся, що позначено контрольний перемикач Show Stereotypes (Показати стереотипи).

4. Переконайтеся, що позначено контрольні перемикачі Show All Attributes (Показати всі атрибути) і Show All Operations (Показати всі операції).

5. Переконайтеся, що не позначено перемикачі Suppress Attributes (Знищити висновок атрибутів) і Suppress Operations (Знищити висновок операцій).

Створення пакетів

1. Клацніть правою кнопкою миші на Логічному представленні браузера.
2. У відкритому меню виберіть пункт New > Package (Створити > пакет).
3. Назвіть новий пакет Entities (Сутності).
4. Повторіть етапи з першого до третього, створивши пакети Boundaries (границі) і Control (керування).

Рис. 5. Головна Діаграма класів системи обробки замовлень

Створення Головної діаграми класів

1. Двічі клацніть на Головній діаграмі класів прямо під Логічним представленням браузера, щоб відкрити її.
2. Перетягніть пакет Entities із браузера на діаграму.
3. Перетягніть пакети Boundaries і Control із браузера на діаграму.

Головна діаграма класів повинна мати вигляд, як на рис. 5.

Створення Діаграми класів для сценарію “Ввести нове замовлення” з усіма класами

1. Клацніть правою кнопкою миші на Логічному представленні браузера.
2. У відкритому меню виберіть пункт New > Class Diagram (Створити > Діаграму класів).
3. Назвіть нову Діаграму Класів Add New Order (Уведення нового замовлення).
4. Клацніть у браузері на цій діаграмі двічі, щоб відкрити її.
5. Перетягніть з браузера всі класи (OrderOptions, OrderDetail, Order, OrderMgr і TransactionMgr).

Діаграма класів повинна мати вигляд, як на рис. 6.

Рис. 6. Діаграма класів Add New Order

Додавання стереотипів до класів

1. Клацніть правою кнопкою миші на класі OrderOptions діаграми.
2. У відкритому меню виберіть пункт Open Specification (Відкрити специфікацію).
3. У полі стереотипу введить слово Boundary.
4. Натисніть на кнопку ОК.
5. Клацніть правою кнопкою миші на класі OrderDetail діаграми.
6. У відкритому меню виберіть пункт Open Specification (Відкрити специфікацію).
7. У списку, що розкривається, в полі стереотипів тепер буде стереотип Boundary. Зазначте його.
8. Натисніть на кнопку ОК.
9. Повторіть етапи 1–4, пов’язавши класи OrderMgr і TransactionMgr зі стереотипом Control, а клас Order – зі стереотипом Entity.

Тепер Діаграма класів повинна мати вигляд, як на рис. 7.

Об’єднання класів у пакети

1. Перетягніть у браузері клас OrderOptions на пакет Boundaries.
2. Перетягніть клас OrderDetail на пакет Boundaries.
3. Перетягніть класи OrderMgr і TransactionMgr на пакет Control.
4. Перетягніть клас Order на пакет Entities.

Додавання Діаграм класів до кожного пакета

1. Клацніть правою кнопкою на пакеті Boundaries браузера.
2. У відкритому меню виберіть пункт New > Class Diagram (Створити > Діаграму Класів).

3. Уведіть ім’я нової діаграми – Main (Головна).
4. Двічі клацніть мишею на цій діаграмі, щоб відкрити її.
5. Перетягніть на неї з браузера класи OrderOptions і OrderDetail.
6. Закрийте діаграму.
7. Клацніть правою кнопкою на пакеті Entities браузера.
8. У відкритому меню виберіть пункт New > Class Diagram (Створити > Діаграму класів).
9. Уведіть ім’я нової діаграми – Main (Головна).
10. Двічі клацніть *мишею на цій діаграмі, щоб відкрити її.*
11. Перетягніть на неї з браузера клас Order.
12. Закрийте діаграму.
13. Клацніть правою кнопкою на пакеті Control браузера.
14. У відкритому меню виберіть пункт New > Class Diagram (Створити > Діаграму Класів).
15. Уведіть ім’я нової діаграми – Main (Головна).
16. Двічі клацніть мишею на цій діаграмі, щоб відкрити її.
17. Перетягніть на неї з браузера класи OrderMgr і TransactionMgr.
18. Закрийте діаграму.

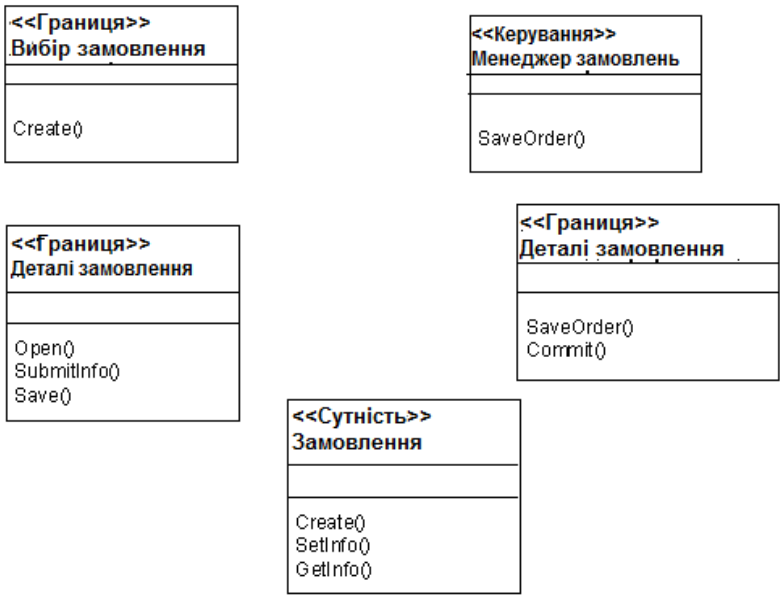


Рис. 7. Стереотипи класів для варіанта використання
Ввести нове замовлення

Практичне завдання № 4
СТВОРЕННЯ ДІАГРАМ КЛАСІВ (УРАХУВАННЯ НОВИХ ВИМОГ)

У практичному завданні № 2 було створено кілька операцій для класів розглянутої задачі. У практичному завданні № 3 на діаграму додано класи. Тепер до описів операцій додамо деталі, включаючи параметри і типи значень, що повертаються. Крім того, у класів визначатимуться атрибути.

Додавання атрибутів і операцій

Додамо атрибути й операції до класів Діаграми класів “Ввести нове замовлення”. Для атрибутів і операцій використовуємо специфічні для мови особливості. Встановимо параметри так, щоб показувати всі атрибути, всі операції та їхні сигнатури. Видимість покажемо за допомогою нотації UML.

Етапи виконання вправи

Настроювання

1. У меню моделі виберіть пункт Tools > Options.
2. Перейдіть на вкладку Diagram.
3. Переконайтеся, що перемикач Show Visibility позначений.
4. Переконайтеся, що перемикач Show Stereotypes позначений.
5. Переконайтеся, що перемикач Show Operation Signatures позначений.
6. Переконайтеся, що перемикачі Show All Attributes і Show All Operations позначені.
7. Переконайтеся, що перемикачі Suppress Attributes і Suppress Operations не позначені.
8. Перейдіть на вкладку Notation.
9. Переконайтеся, що перемикач Visibility as Icons не позначений.

Додавання нового класу

1. Знайдіть у браузері Діаграму класів варіанта використання “Увести нове замовлення”.
2. Клацніть на ній двічі, щоб відкрити.
3. Натисніть кнопку Class панелі інструментів.
4. Клацніть мишею усередині діаграми, щоб помістити там новий клас.
5. Назвіть його OrderItem (Позиція_замовлення).
6. Призначте цьому класові стереотип Entity.
7. У браузері перетягніть клас у пакет Entities.

Додавання атрибутів

1. Клацніть правою кнопкою миші на класі Order (Замовлення).
2. У відкритому меню виберіть пункт New Attribute (Створити _атрибут).
3. Уведіть новий атрибут OrderNumber : Integer (Номер_замовлення)
4. Натисніть клавішу Enter.
5. Уведіть наступний атрибут CustomerName : String (Найменування_ замовника).
6. Повторіть етапи 4 і 5, додавши атрибути OrderDate:Date (Дата_замовлення) і OrderFillDate : Date (Дата_заповнення_замовлення).
7. Клацніть правою кнопкою миші на класі OrderItem.
8. У відкритому меню виберіть пункт New Attribute (Створити_атрибут).
9. Уведіть новий атрибут ItemID : Integer (Ідентифікатор_предмета).
10. Натисніть клавішу Enter.
11. Уведіть наступний атрибут ItemDescription : String (Опис_предмета).

Додавання операцій до класу OrderItem

1. Клацніть правою кнопкою миші на класі OrderItem.

2. У відкритому меню виберіть пункт New Operation (Створити_ операцію).

3. Уведіть нову операцію Create.

4. Натисніть клавішу Enter.

5. Уведіть наступну операцію SetInfo.

6. Натисніть клавішу Enter.

7. Уведіть наступну операцію GetInfo.

Докладний опис операцій за допомогою Діаграми класів

1. Клацніть мишею на класі Order, виділивши його таким способом.

2. Клацніть на цьому класі ще раз, щоб перемістити курсор усередину.

3. Відредагуйте операцію Create(), щоб вона мала такий вигляд: Create() : Boolean

4. Відредагуйте операцію SetInfo(), щоб вона мала такий вигляд: SetInfo(OrderNum : Integer, Customer : String, OrderDate : Date, FillDate : Date) Boolean

5. Відредагуйте операцію GetInfo(), щоб вона мала такий вигляд: GetInfo() : String

Докладний опис операцій за допомогою броузера

1. Знайдіть у браузері клас OrderItem.

2. Щоб розкрити цей клас, клацніть на значку “+” поруч з ним. У браузері з’являться його атрибути й операції.

3. Двічі клацніть на операції GetIInfo(), щоб відкрити вікно її специфікації.

4. У списку, що розкривається, Return class (клас, що повертається) вкажіть String.

5. Клацніть на кнопці ОК, закривши вікно специфікації операції.

6. Двічі клацніть у браузері на операції SetInfo класу OrderItem, щоб відкрити вікно її специфікації.

7. У списку, що розкривається, Return class укажіть Boolean.

8. Перейдіть на вкладку Detail (Докладно).

9. Клацніть правою кнопкою миші на білому полі в ділянці аргументів, щоб додати новий параметр.

10. У відкритому меню виберіть пункт Insert. Rose додасть туди аргумент за назвою argname.

11. Клацніть один раз на цьому слові, щоб виділити його, і змініть ім’я аргумента на ID.

12. Клацніть на колонці Type, відкривши список типів, що розкривається. У ньому виберіть тип Integer.

13. Клацніть на колонці Default, щоб додати значення аргумента за замовчуванням. Уведіть туди число 0.

14. Натисніть на кнопку ОК, закривши вікно специфікації операції.

15. Двічі клацніть на операції Create() класу OrderItem, щоб відкрити вікно її специфікації.

16. У списку, що розкривається, Return class вкажіть Boolean.

17. Натисніть на кнопку ОК, закривши вікно специфікації операції.

Докладний опис операцій за допомогою кожного з описаних методів

1. Використовуючи броузер або Діаграму класів, уведіть таку сигнатуру операцій класу OrderDetail:

Open() : Boolean

SubmitInfo() : Boolean

Save() : Boolean

2. Використовуючи браузер або Діаграму класів, уведіть таку сигнатуру операцій класу OrderOptions:

Create() : Boolean

3. Використовуючи браузер або Діаграму класів, уведіть таку сигнатуру операцій класу OrderMgr:

SaveOrder(OrderID : Integer) : Boolean

4. Використовуючи браузер або Діаграму класів, уведіть таку сигнатуру операцій класу TransactionMgr:

SaveOrder(OrderID : Integer) : Boolean

Commit() : Integer

Практичне завдання № 5
СТВОРЕННЯ ДІАГРАМ КЛАСІВ
(ДОДАВАННЯ ЗВ'ЯЗКІВ МІЖ КЛАСАМИ)

У цій роботі буде визначено зв'язки між класами, що беруть участь у варіанті використання "Ввести нове замовлення".

Додавання зв'язків

Додамо зв'язки до класів, що беруть участь у варіанті використання "Ввести нове замовлення".

Етапи виконання вправи

Настроювання

1. Знайдіть у браузері Діаграму класів "Уведення нового замовлення".

2. Двічі клацніть на ній, щоб відкрити.

3. Перевірте, чи є на панелі інструментів діаграми кнопка Unidirectional Association. Якщо немає, продовжуйте настроювання, виконавши етапи 4 і 5. Якщо є, розпочинайте виконання самої вправи.

4. Клацніть правою кнопкою миші на панелі інструментів діаграми й у відкритому меню виберіть пункт Customize.

5. Додайте на панель кнопку, що називається Create A Unidirectional Association.

Додавання асоціацій

1. Натисніть кнопку панелі інструментів Unidirectional Association.

2. Намалюйте асоціацію від класу Вибір_замовлення (OrderOptions) до класу Деталі_замовлення (OrderDetail).

3. Повторіть етапи 1 і 2, створивши ще асоціації:

Від класу OrderDetail до класу Менеджер замовлень (OrderMgr).

Від класу OrderMgr до класу Замовлення (Order).

Від класу OrderMgr до класу Менеджер транзакцій (TransactionMgr)

Від класу TransactionMgr до класу Order

Від класу TransactionMgr до класу Позиція замовлення (OrderItem).

Від класу Order до класу OrderItem.

4. Клацніть правою кнопкою миші на односпрямованій асоціації між класами OrderOptions і OrderDetail, з боку класу OrderOptions.

5. У відкритому меню виберіть пункт Multiplicity > Zero or One.

6. Клацніть правою кнопкою миші на іншому кінці односпрямованої асоціації.

7. У відкритому меню виберіть пункт Multiplicity > Zero or One.

8. Повторіть етапи 4–7, додавши на діаграму значення множинності для інших асоціацій, як показано на рис. 8.

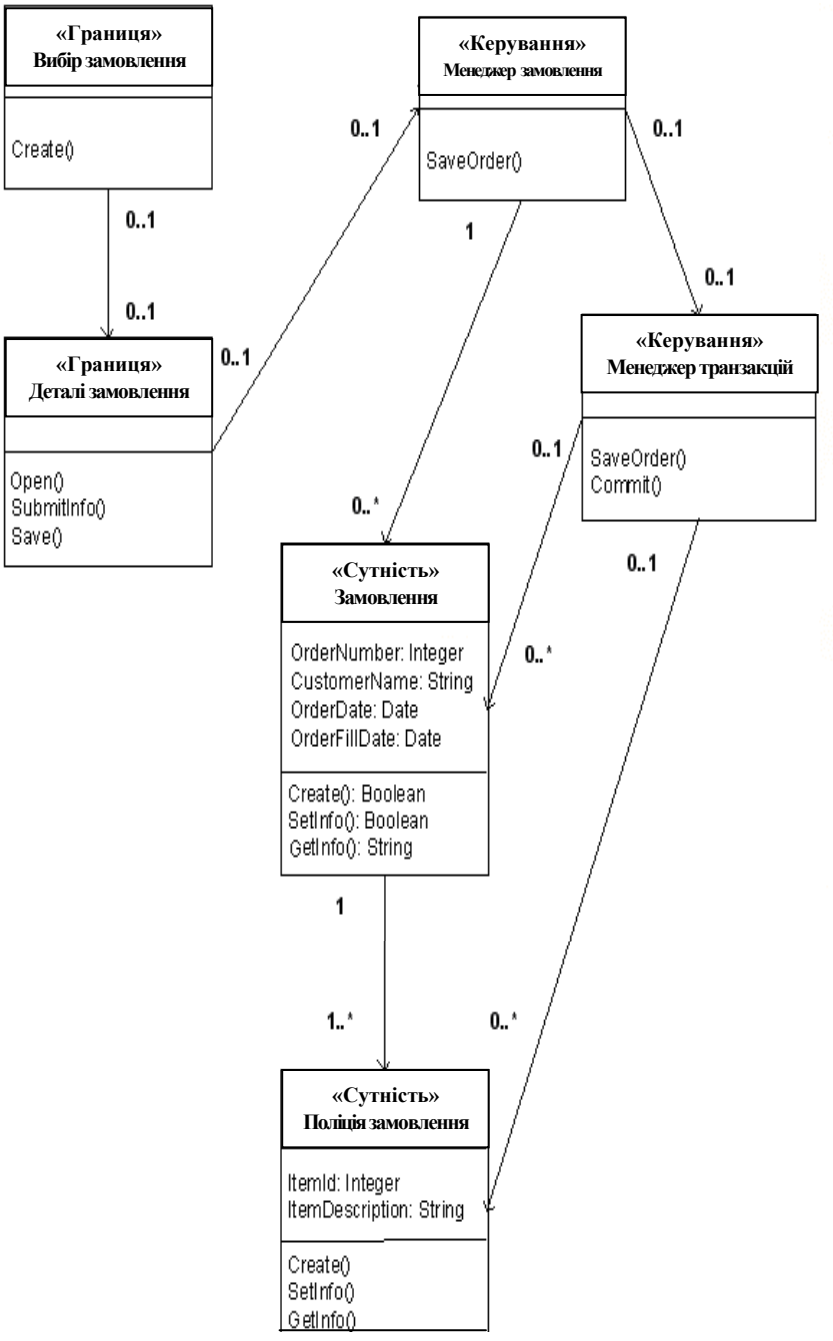


Рис. 8. Асоціації сценарію “Ввести нове замовлення”

Практичне завдання № 6
СТВОРЕННЯ ДІАГРАМИ СТАНІВ

У цьому завданні буде створено Діаграму станів для класу Order.

Створення Діаграми станів

Розробіть Діаграму станів для класу Order, показану на рис. 9.

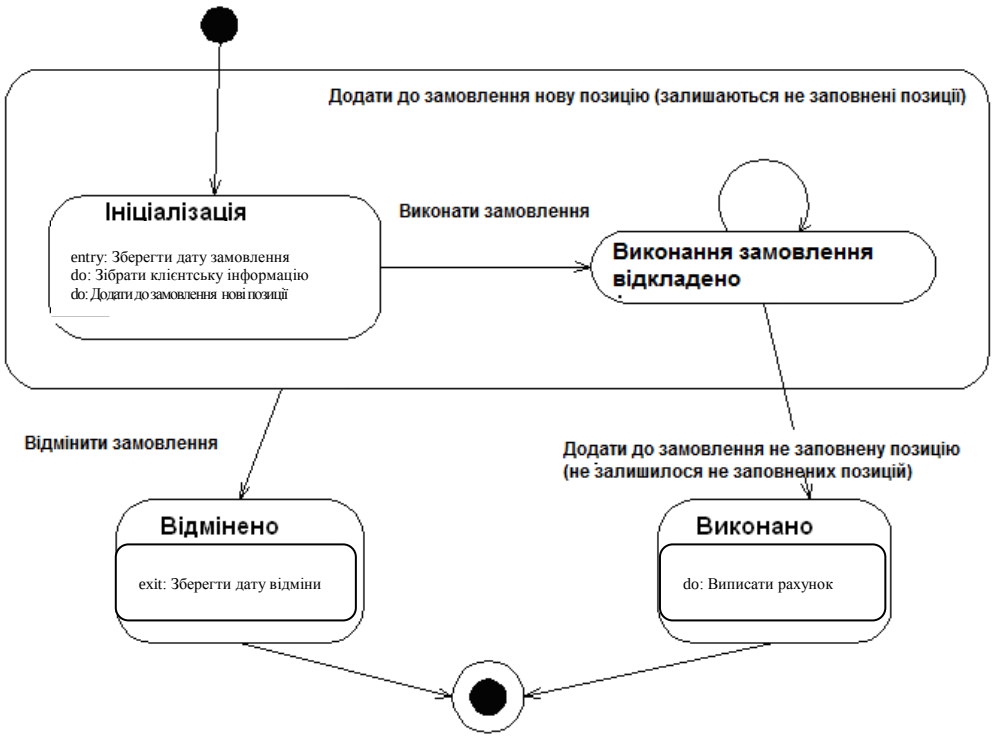


Рис. 9. Діаграма станів для класу Order

Етапи виконання вправи

Створення діаграми

1. Знайдіть у браузері клас Order.
2. Клацніть на класі правою кнопкою миші та у відкритому меню вкажіть пункт Open State Diagram.

Додавання початкового і кінцевого станів

1. На панелі інструментів натисніть кнопку Start State (Початковий стан).
2. Помістіть цей стан на діаграму.
3. На панелі інструментів натисніть кнопку End State (Кінцевий стан).
4. Помістіть цей стан на діаграму.

Додавання суперстану

1. На панелі інструментів натисніть кнопку State (Стан).
2. Помістіть цей стан на діаграму.

Додавання станів, що залишилися

1. На панелі інструментів натисніть кнопку State (Стан).
2. Помістіть цей стан на діаграму.
3. Назвіть стан Cancelled (Відмінний).
4. На панелі інструментів натисніть кнопку State (Стан).
5. Помістіть цей стан на діаграму.
6. Назвіть стан Filled (Виконаний).
7. На панелі інструментів натисніть кнопку State (Стан).
8. Помістіть цей стан на діаграму всередину суперстану.
9. Назвіть стан Initialization (Ініціалізація).
10. На панелі інструментів натисніть кнопку State (Стан).
11. Помістіть цей стан на діаграму всередину суперстану.
12. Назвіть стан Pending (Виконання замовлення припинено).

Докладний опис станів

1. Двічі клацніть на стані Initialization (Ініціалізація).
2. Клацніть правою кнопкою миші на вікні Actions (Дії).
3. У відкритому меню виберіть пункт Insert (Вставити).
4. Двічі клацніть мишею на новій дії.
5. Назвіть його Store Order Date (Зберегти дату замовлення).
6. Переконайтеся, що у вікні When (Коли) зазначено пункт On Entry (На вході).
7. Повторіть етапи 3–7, додавши такі дії:

Collect Customer Info (Зібрати клієнтську інформацію), у вікні When указати пункт Do.

Add Order Items (Додати до замовлення нові графи), у вікні When указати Do.

8. Натисніть на кнопки ОК два рази, щоб закрити специфікацію.
9. Двічі клацніть на стані Cancelled (Відмінний).
10. Повторіть етапи 2–7, додавши дію Store Cancellation Data (Зберегти дату скасування), зазначити пункт On Exit (На виході).
11. Натисніть на кнопки ОК два рази, щоб закрити специфікацію.
12. Двічі клацніть на стані Filled (Виконаний).
13. Повторіть етапи 2–7, додавши дію Bill Customer (Виписати рахунок), зазначити пункт Do.
14. Натисніть на кнопку ОК дві, щоб закрити специфікацію.

Додавання переходів

1. На панелі інструментів натисніть кнопку Transition (Перехід).
2. Клацніть мишею на початковому стані.
3. Проведіть лінію переходу до стану Initialization (Ініціалізація).
4. Повторіть етапи з першого до третього, створивши такі переходи:

Від стану Initialization (Ініціалізація) до стану Pending (Виконання замовлення припинено).

Від стану Pending (Виконання замовлення припинено) до стану Filled (Виконаний).

Від суперстану до стану Cancelled (Відмінний).

Від стану Cancelled (Відмінний) до кінцевого стану.

Від стану Filled (Виконаний) до кінцевого стану.

5. На панелі інструментів натисніть кнопку Transition to Self (Перехід до себе).

6. Клацніть на стані Pending (Виконання замовлення припинено).

Докладний опис переходів

1. Двічі клацніть на переході від стану Initialization (Ініціалізація) до стану Pending (Виконання замовлення припинено), відкривши вікно його специфікації.

2. У полі Event (Подія) уведіть фразу Finalize order (Виконати замовлення).

3. Клацніть на кнопці ОК, закривши вікно специфікації.

4. Повторіть етапи з першого по третій, додавши подію Cancel Order (Скасувати замовлення) до переходу між суперстаном і станом Cancelled (Відмінний).

5. Двічі клацніть на переході від стану Pending (Виконання замовлення припинено) до стану Filled (Виконаний), відкривши вікно його специфікації.

6. У полі Event (Подія) уведіть фразу Add Order Item (Додати до замовлення нову позицію).

7. Перейдіть на вкладку Detail (Докладно).

8. У полі Condition (Умова) Уведіть No unfilled items remaining (Не залишилося незаповнених позицій).

9. Клацніть на кнопці ОК, закривши вікно специфікації.

10. Двічі клацніть мишею на рефлексивному переході (Transition to Self) стану Pending (Виконання замовлення припинено).

11. У полі Event (Подія) уведіть фразу Add Order Item (Додати до замовлення нову позицію).

12. Перейдіть на вкладку Detail (Докладно).

13. У полі Condition (Умова) уведіть Unfilled items remaining (Залишаються незаповнені позиції).

14. Клацніть на кнопці ОК, закривши вікно специфікації.

Практичне завдання № 7
СТВОРЕННЯ ДІАГРАМИ КОМПОНЕНТІВ

У цій роботі буде створено діаграму Компонентів системи обробки замовлень. На даний момент вже визначено всі класи, необхідні для варіанта використання “Ввести нове замовлення”. У міру реалізації інших варіантів використання на діаграму слід додавати нові компоненти.

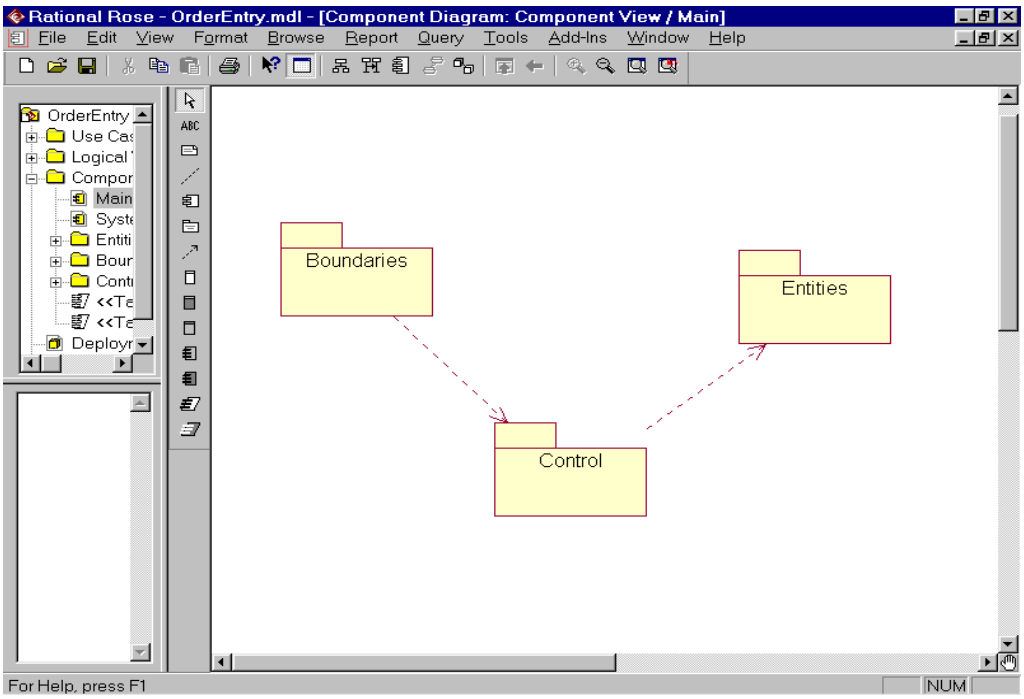


Рис. 10. Головна діаграма компонентів системи

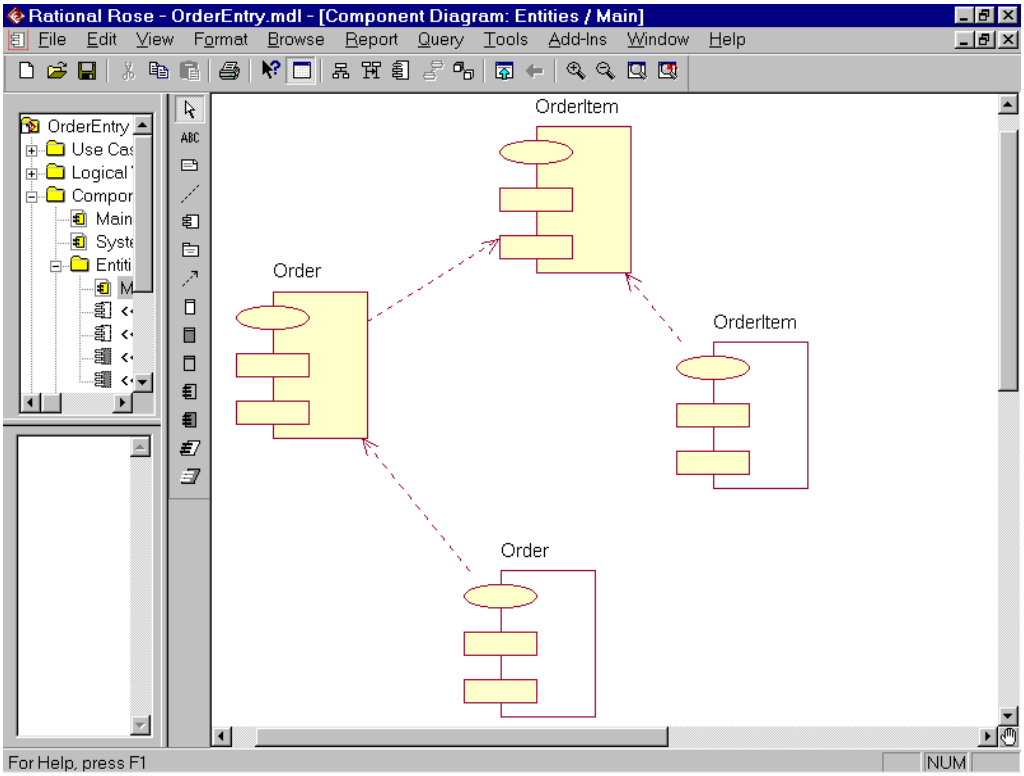


Рис. 11. Діаграма компонентів пакета Entities

На рис. 11 показано всі компоненти пакета Entities. Ці компоненти містять класи пакета Entities Логічного подання системи.

На рис. 12 показано компоненти пакета Control. Вони містять класи пакета Control Логічного подання системи.

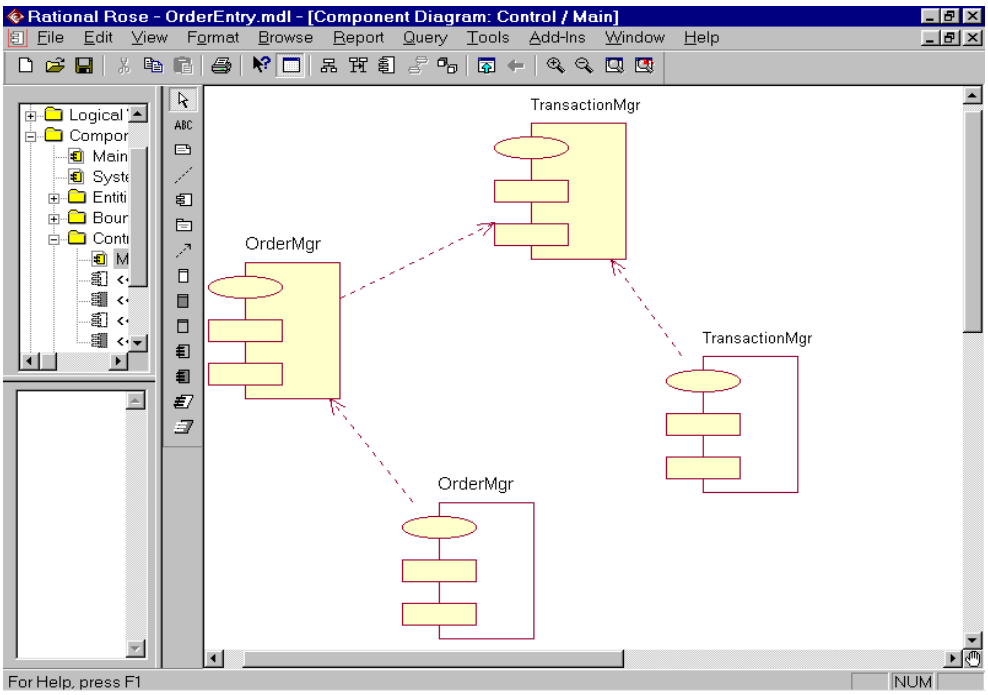


Рис. 12. Діаграма компонентів пакета Control

Нарешті, на рис. 13 показано компоненти пакета Boundaries. Вони також відповідають класам однойменного пакета Логічного подання системи.

На рис. 14 показано всі компоненти системи. Ми назвали цю діаграму Діаграмою компонентів системи. На ній можна побачити всі залежності між усіма компонентами проектованої системи.

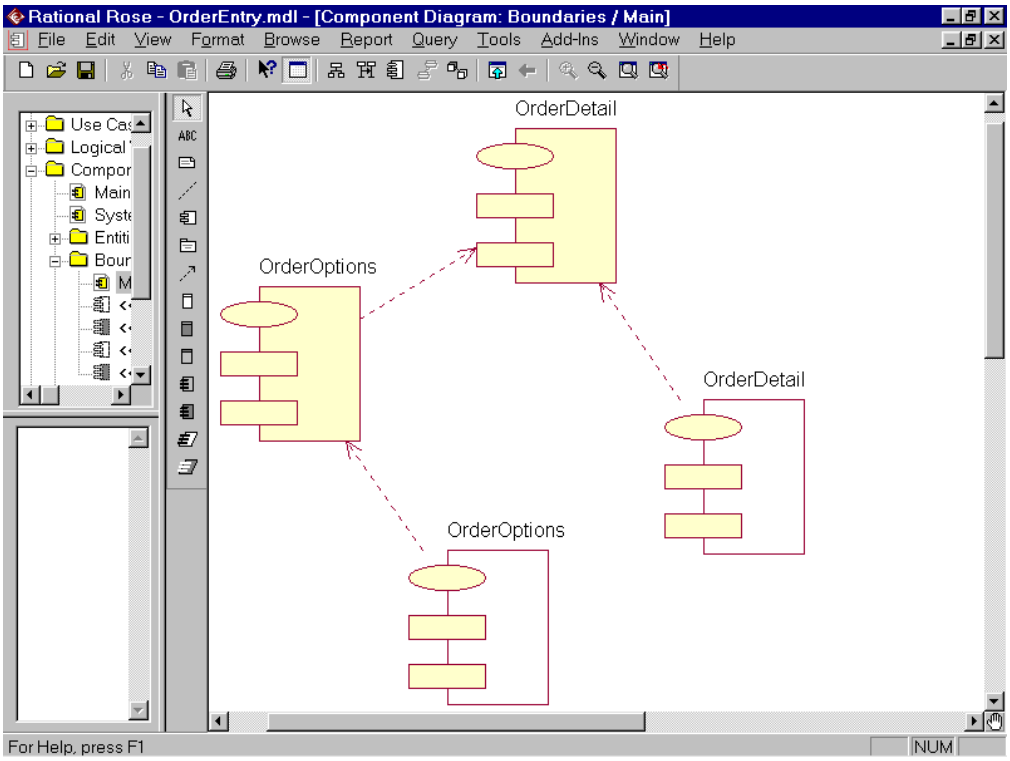


Рис. 13. Діаграма компонентів пакета Boundaries

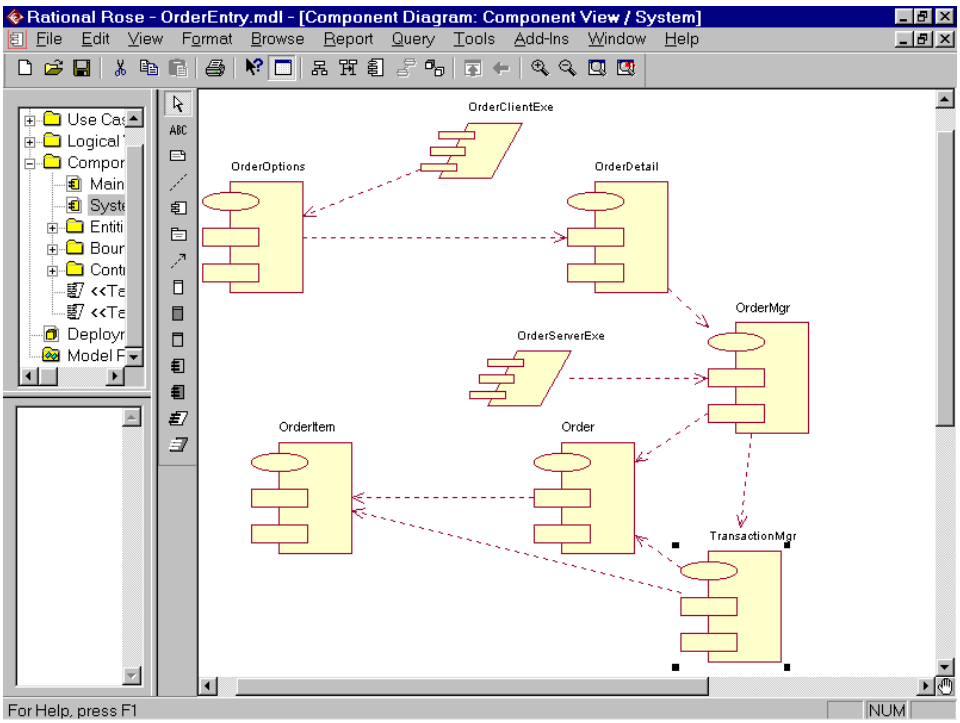


Рис. 14. Діаграма компонентів системи

Етапи виконання вправи

Створення пакетів компонентів

1. Клацніть правою кнопкою миші на зображенні компонентів у браузері.
2. У відкритому меню виберіть пункт New > Package (Створити > пакет).
3. Назвіть цей пакет Entities (Сутності).
4. Повторіть етапи з першого до третього, створивши пакети Boundaries (Границі) і Control (Керування).

Додавання пакетів на Головну діаграму компонентів

1. Відкрийте Головну діаграму компонентів, двічі клацнувши на ній.
2. Перетягніть пакети Entities, Boundary і Control із браузера на Головну діаграму.

Малювання залежностей між пакетами

1. На панелі інструментів натисніть кнопку Dependency (Залежність).
2. Клацніть мишею на упакуванні Boundaries Головної діаграми компонентів.
3. Проведіть лінію залежності до упакування Control.
4. Повторіть етапи 1–3, провівши ще залежність від пакета Control до пакета Entities.

Додавання компонентів до пакетів і малювання залежностей

1. Двічі клацніть мишею на пакеті Entities Головної діаграми компонентів, відкривши Головну діаграму Компонентів цього пакета.
2. На панелі інструментів натисніть кнопку Package Specification (Специфікація пакета).

3. Помістіть специфікацію пакета на діаграму.
4. Уведіть ім’я специфікації пакета OrderItem.
5. Повторіть етапи 2–4, додавши специфікацію пакета Order.
6. На панелі інструментів натисніть кнопку Package Body (Тіло пакета).
7. Помістить його на діаграму.
8. Уведіть ім’я тіла пакета OrderItem.
9. Повторіть етапи 6–8, додавши тіло пакета Order.
10. На панелі інструментів натисніть кнопку Dependency (Залежність).
11. Клацніть мишею на тілі пакета OrderItem.
12. Проведіть лінію залежності від нього до специфікації пакета OrderItem.
13. Повторіть етапи 10–12, додавши лінію залежності між тілом пакета Order і специфікацією пакета Order.
14. Повторіть етапи 10–12, додавши лінію залежності від специфікації пакета Order до специфікації пакета OrderItem.
15. За допомогою описаного методу створіть такі компоненти і залежності:

Для пакета Boundaries:
Специфікацію пакета OrderOptions.
Тіло пакета OrderOptions.
Специфікацію пакета OrderDetail.
Тіло пакета OrderDetail.
Залежності в пакеті Boundaries:
Від тіла пакета OrderOptions до специфікації пакета OrderOptions
Від тіла пакета OrderDetail до специфікації пакета OrderDetail
Від специфікації пакета OrderOptions до специфікації пакета Order Detail
Для пакета Control:
Специфікацію пакета OrderMgr
Тіло пакета OrderMgr
Специфікацію пакета TransactionMgr
Тіло пакета TransactionMgr
Залежності в пакеті Control:
Від тіла пакета OrderMgr до специфікації пакета OrderMgr
Від тіла пакета TransactionMgr до специфікації пакета TransactionMgr
Від специфікації пакета OrderMgr до специфікації пакета TransactionMgr.

Створення Діаграми компонентів системи

1. Клацніть правою кнопкою миші на зображенні компонентів у браузері.
2. У відкритому меню виберіть пункт New > Component Diagram
3. Назвіть нову діаграму System.
4. Двічі клацніть на цій діаграмі.

Розміщення компонентів на Діаграмі компонентів системи

1. Якщо це ще не було зроблено, розгорніть у браузері пакет компонентів Entities, щоб відкрити його.

2. Клацніть мишею на специфікації пакета Order у пакеті компонентів Entities.

3. Перетягніть цю специфікацію на діаграму.

4. Повторіть етапи 2 і 3, помістивши на діаграму специфікацію пакета OrderItem.

5. За допомогою цього методу помістіть на діаграму такі компоненти:

3 пакета компонентів Boundaries:

Специфікацію пакета OrderOptions

Специфікацію пакета OrderDetail

3 пакета компонентів Control:

Специфікацію пакета OrderMgr

Специфікацію пакета TransactionMgr

6. На панелі інструментів натисніть кнопку Task Specification (Специфікація задачі).

7. Помістіть специфікацію задачі на діаграму і назвіть її OrderClientExe.

8. Повторіть етапи 6 і 7 для специфікації задачі OrderServerExe.

Додавання залежностей, що залишилися, на Діаграму компонентів системи

Вже існуючі залежності будуть автоматично показані на Діаграмі компонентів системи після додавання туди відповідних компонентів. Тепер треба додати інші залежності.

1. На панелі інструментів натисніть кнопку Dependency (Залежність).

2. Клацніть на специфікації пакета OrderDetail.

3. Проведіть лінію залежності до специфікації пакета OrderMgr.

4. Повторіть етапи 1–3, створивши такі залежності:

Від специфікації пакета OrderMgr до специфікації пакета Order

Від специфікації пакета TransactionMgr до специфікації пакета OrderItem

Від специфікації пакета TransactionMgr до специфікації пакета Order

Від специфікації задачі OrderClientExe до специфікації пакета OrderOptions

Від специфікації задачі OrderServerExe до специфікації пакета OrderMgr

Співвідношення класів з компонентами

1. У Логічному представленні браузера знайдіть клас Order пакета Entities.

2. Перетягніть цей клас на специфікацію пакета компонента Order у представленні Компонентів браузера. У результаті клас Order буде співвіднесений зі специфікацією пакета компонента Order.

3. Перетягніть клас Order на тіло пакета компонента Order у представленні Компонентів браузера. У результаті клас Order буде співвіднесений з тілом пакета компонента Order.

4. Повторіть етапи 1–3, співвідносячи з класами такі компоненти:

Клас OrderItem зі специфікацією пакета OrderItem

Клас OrderItem з тілом пакета OrderItem

Клас OrderOptions зі специфікацією пакета OrderOptions
Клас OrderOptions з тілом пакета OrderOptions
Клас OrderDetail зі специфікацією пакета OrderDetail
Клас OrderDetail з тілом пакета OrderDetail
Клас OrderMgr зі специфікацією пакета OrderMgr
Клас OrderMgr з тілом пакета OrderMgr
Клас TransactionMgr зі специфікацією пакета TransactionMgr
Клас TransactionMgr з тілом пакета TransactionMgr.

Практичне завдання № 8
СТВОРЕННЯ ДІАГРАМИ РОЗМІЩЕННЯ

У цій роботі буде створена Діаграма розміщення для системи обробки замовлень.

Постановка задачі

Команда розробників завершила весь попередній аналіз і проектування системи. Варіанти використання, взаємодії між об'єктами і компоненти чітко описані. Проте підрозділові адміністрування мережі потрібно знати, на яких комп'ютерах буде розміщено різні компоненти системи. У зв'язку з цим довелося ще розробити Діаграму розміщення для системи обробки замовлень.

Створення Діаграми розміщення

Розробіть діаграму Розміщення для системи обробки замовлень. Готова діаграма повинна мати вигляд, як на рис. 15.

Етапи виконання вправи

Додавання вузлів до Діаграми розміщення

1. Двічі клацніть мишею на представленні Розміщення в браузері, щоб відкрити Діаграму розміщення.
2. На панелі інструментів натисніть кнопку Processor (Процесор).
3. Клацніть на діаграмі, помістивши туди процесор.
4. Уведіть ім'я процесора "Сервер бази даних".
5. Повторіть етапи 2–4, додавши такі процесори:
Сервер додатка
Клієнтська робоча станція № 1
Клієнтська робоча станція № 2.
6. На панелі інструментів натисніть кнопку Device (Пристрій).
7. Клацніть на діаграмі, помістивши на неї пристрій.
8. Назвіть його "Принтер".

Додавання зв'язків

1. На панелі інструментів натисніть кнопку Connection (Зв'язок).
2. Клацніть на процесорі "Сервер бази даних".

3. Проведіть лінію зв’язку до процесора “Сервер додатка”.

4. Повторіть етапи 1–3, додавши такі зв’язки:

Від процесора “Сервер додатка”до процесора “Клієнтська робоча станція № 1”

Від процесора “Сервер додатка”до процесора “Клієнтська робоча станція № 2”

Від процесора “Сервер додатка”до пристрою “Принтер”

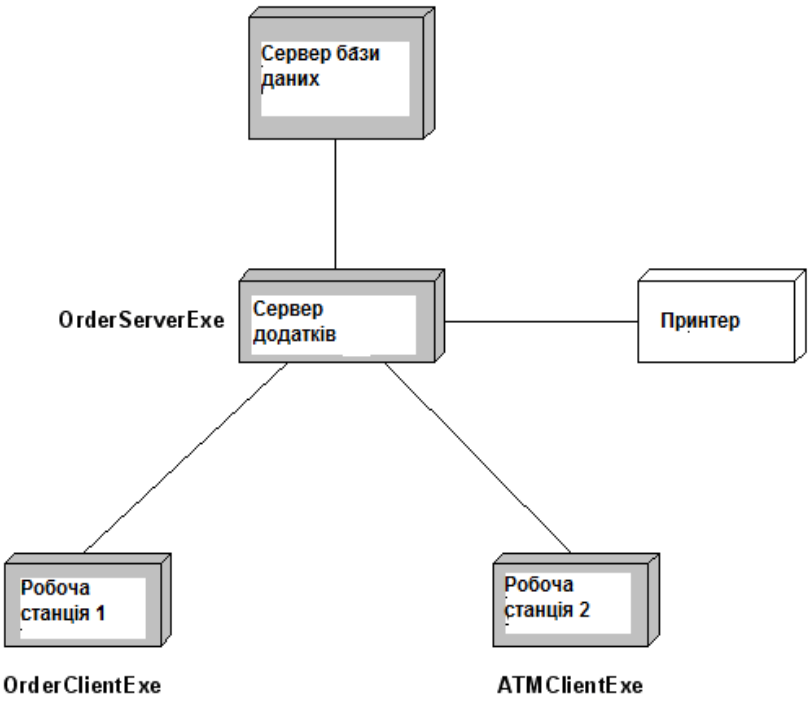


Рис. 15. Діаграма розміщення для системи обробки замовлень

Додавання процесів

1. Клацніть правою кнопкою миші на процесорі “Сервер додатка” у браузері.

2. У меню виберіть пункт New > Process (Створити > Процес).

3. Уведіть ім’я процесу OrderServerExe.

4. Повторіть етапи 1–3, додавши ще процеси:

На процесорі “Клієнтська робоча станція № 1” – процес OrderClientExe

На процесорі “Клієнтська робоча станція № 2” – процес ATM ClientEXE

Показ процесів на діаграмі

1. Клацніть правою кнопкою миші на процесорі “Сервер додатка”.

2. У відкритому меню виберіть пункт Show Processes (Показати процеси).

3. Повторіть етапи 1 і 2, показавши процеси на таких процесорах:

Клієнтська робоча станція № 1

Клієнтська робоча станція № 2.

Практичне завдання № 9
ГЕНЕРАЦІЯ КОДУ C++

У попередніх роботах було створено модель для системи обробки замовлень (Order Entry). Тепер згенеруємо програмний код C++ для цієї системи. При цьому скористаємося Діаграмою компонентів системи, показаною на рис. 16. Для генерації програмного коду слід виконати описані нижче кроки.

Етапи виконання вправи
Уведення тіл пакетів на Діаграму компонентів системи

1. Відкрийте Діаграму компонентів системи.
2. Виберіть у браузері Entities: тіло пакета Order.
3. “Перетягніть” тіло пакета Order на Діаграму компонентів системи.
4. Повторіть п. 2 і 3 для таких компонентів:

Entities: тіло пакета OrderItem.
Boundaries: тіло пакета OrderOptions.
Boundaries: тіло пакета OrderDetail.
Control: тіло пакета TransactionMgr.
Control: тіло пакета OrderMgr.

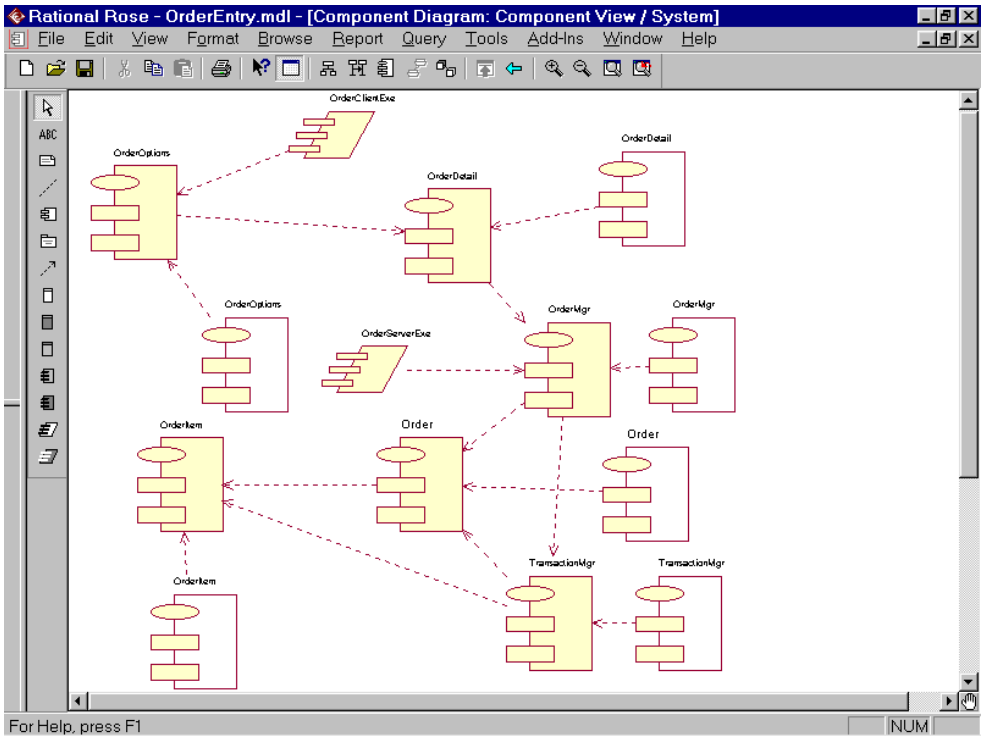


Рис. 16. Діаграма компонентів системи Order Entry

Установка мови C++

1. Відкрийте специфікацію компонента Order (специфікацію пакета) у пакеті компонентів Entities.
2. Виберіть як мову C++.
3. Повторіть п. 1 і 2 для таких компонентів:

Entities: тіло пакета Order.
Entities: специфікація пакета OrderItem.
Entities: тіло пакета OrderItem.
Boundaries: специфікація пакета OrderOptions.
Boundaries: тіло пакета OrderOptions.
Boundaries: специфікація пакета OrderDetail.
Boundaries: тіло пакета OrderDetail.
Control: специфікація пакета TransactionMgr.
Control: тіло пакета TransactionMgr.
Control: специфікація пакета OrderMgr.
Control: тіло пакета OrderMgr.
Специфікація задачі OrderClientExe.
Специфікація задачі OrderServerExe.

Генерація програмного коду C++

1. Відкрийте Діаграму компонентів системи.
2. Виберіть усі об'єкти на Діаграмі компонентів системи.
3. Виберіть Tools > C++ > Code Generation у меню.

Література

Основна:

1. Буч Г. Язык UML. Руководство пользователя / Гради Буч, Джейм Рамбо, Ивар Якобсон ; пер. с англ. Мухина Н. – Изд. 2-е. – М. : ДМК Пресс, 2006. – 496 с.

2. Бенькович Е. С. Практическое моделирование динамических систем / Е. С. Бенькович, Ю. Б. Колесов, Ю. Б. Сениченков. – СПб. : BHV-СПб, 2002. – 444 с.

3. Вендров А. М. Проектирование программного обеспечения экономических информационных систем : учебник / А. М. Вендров. – 2-е изд. перераб. и доп. – М. : Финансы и статистика, 2005. – 544 с.

4. Иванова Г. С. Технология программирования : учебник для вузов / Г. С. Иванова. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2002. – 320 с.

5. Липаев В. В. Программная инженерия. Методологические основы : учебник / В. В. Липаев. – М. : ТЕИС, 2006. – 608 с.

6. Норенков И. П. Основы автоматизированного проектирования : учеб. для вузов / И. П. Норенков. – 4-е изд., перераб. и доп. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2009. – 430 с.

7. Орлов С. Технологии разработки программного обеспечения : учебник / С. Орлов. – СПб. : Питер, 2002. – 464 с.

8. Орлов С. А. Технологии разработки программного обеспечения : учебник / С. А. Орлов, Б. Я. Цилькер. – СПб. : Питер, 2012. – 608 с.

9. Фаулер М. UML. Основы. Краткое руководство по стандартному языку объектного моделированияю / Мартин Фаулер. – Третье издание. – М. : Символ-Плюс, 2006.

10. Фаулер М. UML в кратком изложении. Пример стандартного языка объектного моделирования / Мартин Фаулер. – М. : Мир, 1999.

11. Федотова Д. Э. CASE-технологии : практикум / Федотова Д. Э., Семенов Ю. Д., Чижик К. Н. – М. : Горячая линия – Телеком, 2005. – 160 с.

Додаткова:

12. Боггс У. UML и Rational Rose / Уэнди Боггс, Майкл Боггс, И. Дранишников ; пер. с англ. И. Афанасьева. – М. : Лори, 2008. – 600 с.

13. Леоненков А. Самоучитель UML / Александр Леоненков. − СПб. : БХВ-Петербург, 2007. – 576 с.

14. Ларман К. Применение UML и шаблонов проектирования : пер. с англ. / Крэг Ларман. – 2-е изд. – М. : Вильямс, 2004. – 624 с.

15. Трофимов С. А. CASE-технологии. Практическая работа в Rational Rose / С. А. Трофимов. – М. : Бином, 2001. – 272 с.

16. Штаєр Л. О. Технології розробки програмного забезпечення : конспект лекцій / Л. О. Штаєр. – Івано-Франківськ : ІФНТУНГ, 2017. – 139 с.

17. Технологія розробки програмного забезпечення [Електронний ресурс]. – Режим доступу : http://easy-code.com.ua/2011/07/texnologiya-rozrobki-programnogo-zabezpechennya

НАВЧАЛЬНЕ ВИДАННЯ

**Яковенко В. О., Ульяновська Ю. В., Костенко В. В.,
Костенко Д. Є., Лавренюк І. В., Молотков О. Н.**

ОСНОВИ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ СКЛАДНИХ ОБ'ЄКТІВ І СИСТЕМ

Навчальний посібник

Редактори: *Т. П. Дерев'янко, Л. І. Малигіна,
О. О. Смирнова, І. В. Орищій*
Комп'ютерна верстка: *О. О. Іщенко, Т. Г. Пунтус*

**Підписано до друку 12.11.2018. Формат 60х84 1/16. Папір офсетний.
Ум. друк. арк. 7,13. Облік.-вид. арк. 6,33. Наклад 100 прим.
Замовлення № 139.**
**Дніпро: Університет митної справи та фінансів
(свідоцтво про видавничу діяльність ДК № 6198 від 24.05.2018 р.)
49000, м. Дніпро, вул. Володимира Вернадського, 2/4**