Міністерство освіти і науки України
Державний вищій навчальний заклад
“Криворізький національний університет”

МЕТОДИЧНІ ВКАЗІВКИ

для виконання лабораторних робіт
з дисципліни
„ Інженерія програмного забезпечення ”

Для студентів денної та заочної форм навчання

Спеціальність 123 «Комп’ютерна інженерія»
Факультет інформаційних технологій
Кафедра комп’ютерні системи і мережі

Кривий Ріг
2018

Методичні вказівки для студентів денної та заочної форм навчання з дисципліни " Інженерія програмного забезпечення ".

Укладач: к.т.н. Вдовиченко І.Н.
Комп'ютерний набір : к.т.н. Вдовиченко І.Н.

Схвалено на засіданні кафедри комп'ютерні системи і мережі
(Протокол № 7 від 20.02. 2018 р.)

Завкафедрою _____ д.т.н. А.І. Купін
підпис

Методичні вказівки розглянуто та схвалено вченою радою факультету інформаційних технологій
(Протокол № 7 від 21.02. 2018 р.)

Голова вченої ради ФІТ _____ В.А. Чубаров
підпис

Начальник навчально-методичного відділу _____ Г.Х.Отверченко
підпис

Методичні вказівки рекомендовано для проведення лабораторних робіт з курсу дисципліни " Інженерія програмного забезпечення ".

Наклад примірників: за вимогою

ЗМІСТ

ВСТУП

Метою лабораторних робіт з дисципліни "Інженерія програмного забезпечення" є формування у студентів навичок, необхідних для проектування, тестування, розробки, аналізу, верифікації, оцінки, атестації програмних продуктів.

Основне **завдання** лабораторних робіт з дисципліни "Інженерія програмного забезпечення" полягає в підготовці висококваліфікованих спеціалістів, здатних застосовувати на практиці знання одержані на лекційних заняттях.

Особливістю робіт є те, що особливу увагу приділяється креативності завдань. Кожний студент має свою особисту наскрізну задачу на протязі всього семестру, рішення якої, крім знань, потребує творчого підходу.

ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Для успішного виконання лабораторних робіт необхідно:

1. Ознайомитися з темою та метою лабораторної роботи.
2. Уважно вивчити теоретичний матеріал запропонований в лабораторній роботі.
3. Уважно вивчити хід роботи.
4. Виконати вказаний перелік дій даної роботи.
5. Записати результати.
6. Виконати необхідні проектні роботи за вказаним планом.
7. Проаналізувати отримані результати.
8. Оформити звіт.

Лабораторна робота № 1

Тема: «Розробка і опис каскадної моделі проектування інформаційної системи"

Мета роботи: Описати і проаналізувати інформаційну систему, побудувати її каскадну модель.

Методичні вказівки

Лабораторна робота спрямована на ознайомлення з процесом опису інформаційної системи і отримання навичок побудови каскадної моделі ІС.

Вимоги до результатів виконання лабораторної роботи:

1. наявність опису інформаційної системи;
2. проведення аналізу етапів моделі ІС;
3. наявність базових пропозицій по часу і необхідному програмному забезпеченню.

Теоретичні відомості

Загальні відомості про розробку програмного забезпечення

Неможливо описати і стандартизувати всі роботи, що виконуються в проекті по створенню ПЗ. Ці роботи залежать від організації, де виконується розробка ПЗ, від типу створюваного програмного продукту. Але завжди можна виділити наступні моменти:

1. Написання пропозицій по створенню ПЗ.
2. Вибір моделі проектування ПО.
3. Планування і складання графіка робіт зі створення ПЗ.
4. Оцінювання вартості проекту.
5. Підбір персоналу.
6. Контроль за ходом виконання робіт.
7. Написання звітів та уявлень.

Перша стадія програмного проекту може складатися з написання пропозицій щодо реалізації цього проекту. Пропозиції повинні містити опис цілей проектів і способів їх досягнення.

Написання пропозицій - дуже відповідальна робота. Не існує будь-яких рекомендацій з написання пропозицій.

Вибір моделі проектування ПЗ залежить від різних факторів. У даній роботі відзначимо + і - каскадної моделі для даної ІС. Наведемо поетапно весь хід розробки ІС. Для цього треба:

1. Вказати апаратні і програмні ресурси, необхідні для реалізації проекту.
2. Розбити роботу на етапи. Процес реалізації проекту розбивається на окремі процеси, визначаються етапи виконання проекту, наводиться опис результатів ("виходів") кожного етапу і контрольні позначки.
3. Графік робіт. У цьому графіку відображаються залежності між окремими процесами (етапами) розробки ПЗ, оцінки часу їх виконання.

Порядок виконання роботи

1.Отримати у викладача завдання згідно варіанту.

2. Написати пропозиції по реалізації цього проекту (функції ІС).
3. Вказати апаратні і програмні ресурси для проектування ІС.
4. Скласти графік робіт.
5. Побудувати каскадну модель проектування ІС.
6. Описати етапи проектування ІС.
7. Провести аналіз етапів моделі ІС.
8. Зробити висновки про позитивні і негативні сторони використання каскадної моделі проектування ІС.

Приклад каскадної моделі показаний на рис.1.

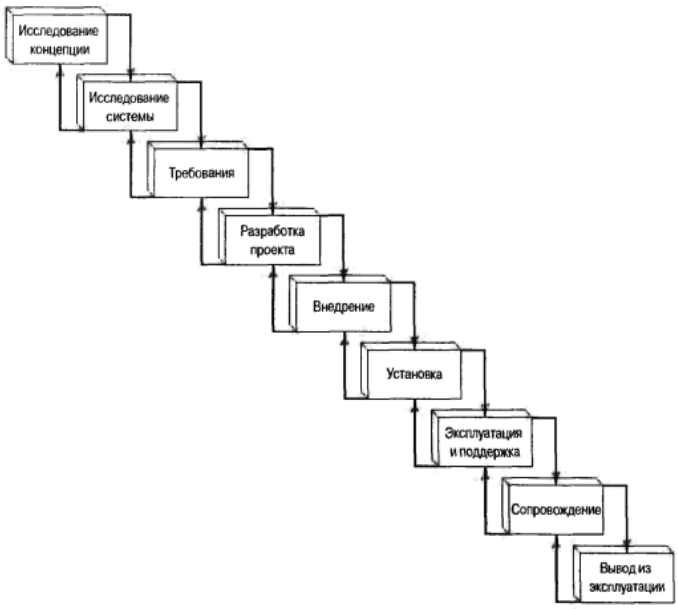


Рис.1 Класична каскадна модель зі зворотним зв'язком

Питання для самоконтролю

1. Особливості каскадної моделі
2. Основні функції системи, що розробляється
3. Якє програмне та апаратне забезпечення використовується

Лабораторна робота № 2

***Тема*:** Побудова структурної, функціональної схем та алгоритму програмної системи

***Мета роботи*:** Навчитися створювати структурні, функціональні схеми та алгоритми програмних систем

Теоретичні відомості

Фактично інформаційні потоки системи є відображенням функціональної і структурної організації досліджуваного об'єкта в ракурсі механізму прийняття рішень всередині системи.

До параметрів інформаційних потоків відносять:

• загальний час реагування;
• інтенсивність;
• надмірність;

• дублювання;
• нестабільність;
• похибка;
• форми подання.

В ході інформаційного аналізу в системі виділяють рівні ієрархії, окремі вузли (інформаційні елементи) і зв'язують їх потоки інформації. Вся система представляється у вигляді спрямованого графа, вершинами якого служать вузли управління, а ребрами - інформаційні потоки. Напрямок ребер відповідає напрямку інформаційних потоків.

Результатом інформаційного опису системи є:

• визначення складу інформаційних елементів,

• складу і структури інформаційних потоків між ними,

• кількість і цінність інформації, що надходить (вихідної) в (з) інформаційних елементів;

• алгоритмів перетворення інформації у відповідних інформаційних елементах.

Сукупність функціонального, структурного і інформаційного описів дозволяє відобразити основні характеристики систем.

Функціональні процеси в системі тісно пов'язані з інформаційними. Джерелом інформації для функціонування системи є внутрішній ресурс і середовище.

На функціональній схемі відображають поведінку системи, виконувані сервіси.

Структурні схеми

Формування структури є частиною вирішення загальної задачі опису системи. Структура виявляє загальну конфігурацію системи, а не визначає систему в цілому.

Якщо зобразити систему як сукупність блоків, які здійснюють деякі функціональні перетворення, і зв'язків між ними, то отримаємо структурну схему, яка в узагальненому вигляді описує структуру системи. Під блоком зазвичай розуміють, функціонально закінчений і оформлене у вигляді окремого цілого пристрій. Членування на блоки може здійснюватися виходячи з необхідного ступеня деталізації опису структури. Крім функціональних, в структурну схему можуть включатися логічні блоки, що дозволяють змінювати характер функціонування в залежності від того, виконуються чи ні деякі наперед задані умови.

Структурні схеми наочні і вміщають в себе інформацію про велику кількість структурних властивостей системи. Вони легко піддаються уточненню і конкретизації.

Однак, структурна схема - це ще не модель структури. Вона насилу піддається формалізації і є скоріше природним містком, що полегшує перехід від змістовного опису системи до математичного, ніж дійсним інструментом аналізу і синтезу структур.

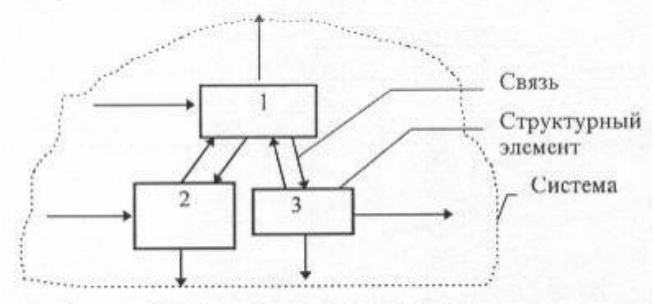


Рис.1 Приклад структурної схеми

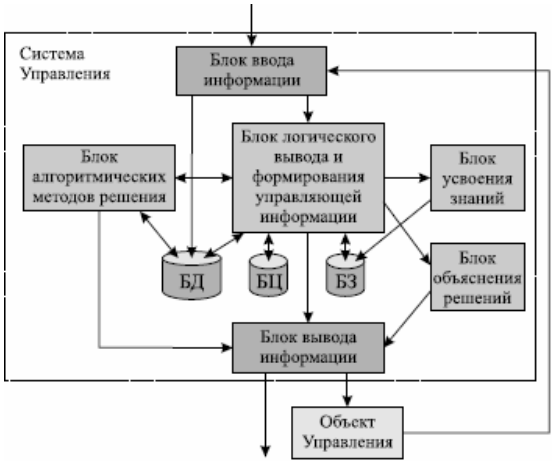


Рис.2. Структурна схема інтелектуальної робототехнічної системи

Порядок виконання роботи

1. Згідно з варіантом, побудувати структурну схему інформаційної системи в цілому та схеми окремих модулів.
2. Побудувати функціональну схему інформаційної системи.
3. Представити алгоритм інформаційної системи.
4. Зробити необхідні пояснення.
5. Зробити висновки.
6. Оформити звіт.

Питання для самоконтролю

1. Правила побудови алгоритмів
2. Компоненти структурної схеми
3. Компоненти функціональної схеми

Лабораторна робота №3

Тема: ***«Методологія функціонального моделювання»***

Мета роботи: ***Вивчити методологію функціонального моделювання IDEF0.***

Методичні вказівки

Лабораторна робота спрямована на ознайомлення з методологіями функціонального моделювання IDEF0, отримання навичок щодо застосування даної методології для побудови функціональних моделей на підставі вимог до інформаційної системи.

Вимоги до результатів виконання лабораторної роботи:

1. модель повинна відображати весь зазначений в описі функціонал, а також чітко відображати існуючі потоки даних і описувати правила їх руху;
2. наявність в моделі не менше трьох рівнів;
3. не менше двох рівнів декомпозиції в стандарті IDEF0 (контекстна діаграма + діаграми А0);
4. на діаграмі 1-го рівня (А0) не менше 4-х функціональних блоків;
5. на діаграмі 2-го і далі рівнях повинна бути декомпозиція не менше 2-х функціональних блоків.

Теоретичні відомості

Основні поняття IDEF0

IDEF0 (Integrated Definition Function Modeling) - методологія функціонального моделювання. В основі методології IDEF0 лежить поняття блоку, який відображає деяку функцію. Чотири сторони блоку мають різну роль: ліва сторона має значення "входу", права - "виходу", верхня - "управління", нижня - "механізму" (рис. 1).

Взаємодія між функціями в IDEF0 представляється у вигляді дуги, яка відображає потік даних або матеріалів, що надходить з виходу однієї функції на вхід іншого. В залежності від того, з якою стороною блоку пов'язаний потік, його називають відповідно "вхідним", "вихідних", "керуючим".

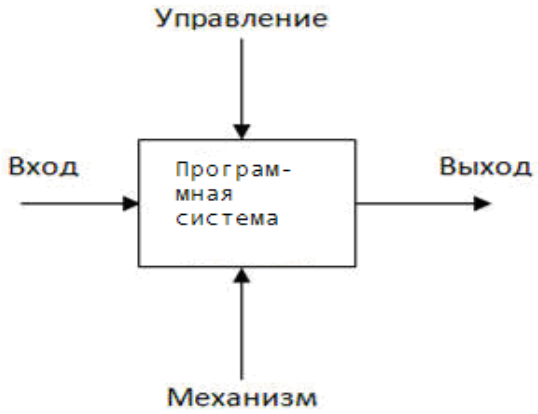


Рис. 1. Функціональний блок

Принципи моделювання IDEF0

В IDEF0 реалізовані три базових принципи моделювання процесів:
• принцип функціональної декомпозиції;
• принцип обмеження складності;
• принцип контексту.

Принцип функціональної декомпозиції являє собою спосіб моделювання типової ситуації, коли будь-яка дія, операція, функція можуть бути розбиті (декомпозованих) на більш прості дії, операції, функції, підсистеми. Іншими словами, складна система може бути представлена у вигляді сукупності елементарних підсистем. Представляючи систему графічно, у вигляді блоків, можна заглянути всередину блоку і детально розглянути її структуру і склад.

Принцип обмеження складності. При роботі з діаграм IDEF0 істотною є умова їх чіткість і зрозумілість. Суть принципу обмеження складності полягає в тому, що кількість блоків на діаграмі має бути не менше двох і не більше шести. Практика показує, що дотримання цього принципу призводить до того, що функціональні процеси, представлені у вигляді IDEF0 моделі, добре структуровані, зрозумілі і легко піддаються аналізу.

Принцип контекстної діаграми. Моделювання ділового процесу починається з побудови контекстної діаграми. На цій діаграмі відображається тільки один блок - головна

функція модельованої системи. Головна функція системи - це "місія" системи, її значення у навколишньому світі.

При визначенні головної функції необхідно завжди мати на увазі мету моделювання і точку зору на модель. Один і той же процес може бути описаний по-різному, залежно від того, з якої точки зору його розглядають: замовник і програміст бачать процес проектування системи абсолютно по-різному.

Контекстна діаграма грає ще одну роль у функціональній моделі. Вона "фіксує" межі системи, що моделюється, визначаючи те, як модельована система взаємодіє зі своїм оточенням. Це досягається за рахунок опису дуг, сполучених з блоком, що представляє головну функцію.

Приклад.

На рис. 2 і рис. 3 представлений приклад побудови функціональної діаграми, що описує виготовлення виробу. Рис. 2 - контекстна діаграма. Рис. 3 - перший рівень декомпозиції.

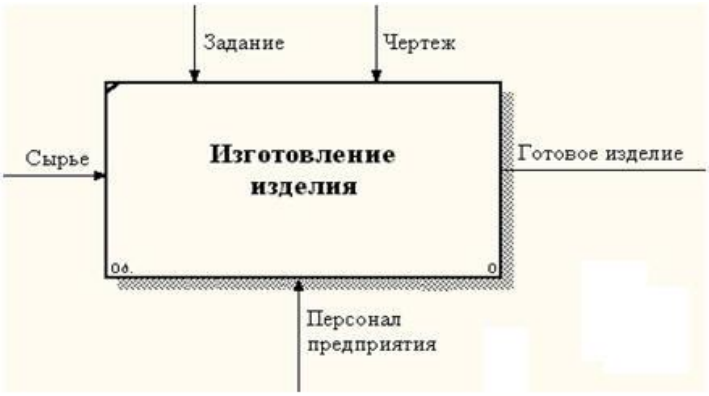


Рис. 2. Контекстна діаграма

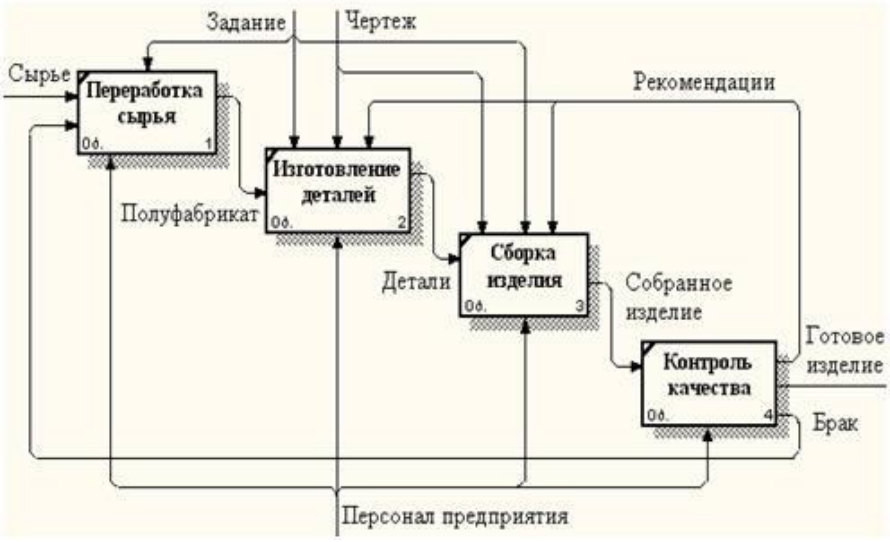


Рис.3. Діаграма першого рівня декомпозиції

Застосування IDEF0

Існує два ключових підходи до побудови функціональної моделі: побудова “як є” і побудова “як буде”.

Побудова моделі **“як є**”. Обстеження предметної області є обов'язковою частиною будь-якого проекту створення інформаційної системи.

Побудова функціональної моделі “як є” дозволяє чітко зафіксувати, які інформаційні об'єкти використовуються при виконанні процесів та окремих операцій. Функціональна модель “як є” є відправною точкою для аналізу потреб автоматизації, виявлення проблем та "вузьких" місць та розробки проекту програмної системи.

Побудова моделі **“як буде”**. Створення і впровадження інформаційної системи приводить до зміни умов виконання окремих операцій, структури ділових процесів і підприємства в цілому. Це призводить до необхідності зміни системи роботи підприємства або організації. Функціональна модель “як буде” дозволяє вже на стадії проектування майбутньої інформаційної системи визначити ці зміни. Застосування функціональної моделі “як буде” дозволяє не тільки скоротити терміни впровадження інформаційної системи, а також знизити ризики, пов'язані з неприйняттям персоналу до інформаційних технологій.

Порядок виконання роботи

1. Вивчити запропонований теоретичний матеріал.
2. Побудувати функціональну модель системи, описаної в Лабораторній роботі № 1 так, щоб вона відповідала всім пред'явленим до системи вимогам, представляла повний функціонал системи (кожній функції в описі системи повинен відповідати принаймні один функціональний блок) та її основні процеси.
3. За допомогою методології IDEF0 побудувати контекстну діаграму.
4. За допомогою методології IDEF0 побудувати діаграми 1-го рівня (A0) - модель оточення;
5. За допомогою методології IDEF0 застосувати побудову моделі “як є” та “як буде”.
6. Побудувати звіт, що включає всі отримані рівні моделі, опис функціональних блоків, потоків даних, сховищ і зовнішніх об'єктів.
7. Зробити висновки.

Питання для самоконтролю

1. Що таке декомпозиція
2. Стандарт представлення блоків системи
3. Поясніть методологію IDEF0

Лабораторна робота №4

Тема: Моделювання структури інформаційної системи на базі мови UML, за допомогою діаграми класів і діаграми прецедентів.

Мета роботи: Ознайомлення з основними елементами проектування і моделювання програмних систем, що реалізовується шляхом відношення між об'єктами, за допомогою мови UML.

Теоретичний матеріал

Якщо моделюється невелика програмна система - то стартовою діаграмою може бути діаграма класів. Проте, при проектуванні великої комерційної програми, спочатку розробляється діаграма прецедентів (use case diagram). Інформація, що міститься в діаграмі прецедентів, сприймається як початкова специфікація для розробки інших діаграм. Діаграма прецедентів (яка в деяких випадках називається діаграмою варіантів використання), в загальному випадку моделює поведінку системи, що розглядається як "чорний ящик".

Для знову проектованої системи діаграма прецедентів дозволяє специфікувати вимоги до функцій системи по відношенню до зовнішніх акторів. Діаграма прецедентів є діаграматичне представлення безлічі прецедентів, безлічі акторів і відношення між ними.

Актор (Дійова особа) - це зовнішній агент, який взаємодіє з системою на різних етапах її існування. Актором може бути або інша програмна система, або людина (користувач системи, її розробник, експерт предметної області системи), або час. Час стає дійовою особою, якщо від нього залежить запуск яких-небудь подій в системі. Дійова особа (actor) - це ***роль,*** яку користувач грає по відношенню до системи. Дійові особи є ролями, а не конкретними людьми або найменуваннями робіт.

Діаграма прецедентів є безліччю графічних символів акторів, безліччю графічних символів прецедентів і відношення між елементами цих двох великих множин.

Графічний символ актора - іконка у вигляді стилізованого зображення людини. Кожен актор на діаграмі прецедентів має унікальне ім'я.

Графічним символом прецеденту є еліпс. Кожен прецедент забезпечується унікальним ім'ям.

Графічний символ зв'язку комунікації, тобто зв'язку між прецедентом і дійовою особою - це односпрямована асоціація (суцільна лінія).

Імена повинні відбивати семантику актора або прецедента і записуються англійською мовою.

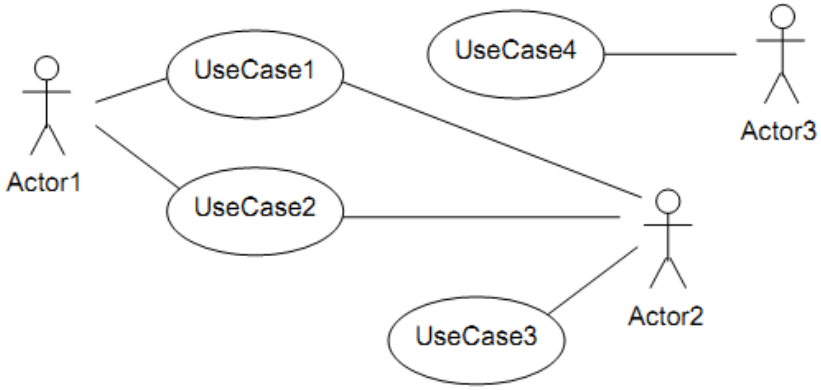


Рис. 1. Базова нотація діаграми прецедентів

Діаграма прецедентів, приведена на рис.1, моделює систему, яка включає трьох акторів і чотири прецеденти.

На рис. 2 наведений приклад діаграми прецедентів, підсистеми банківського автомата, що моделює поведінку, системи обслуговуючої клієнта.

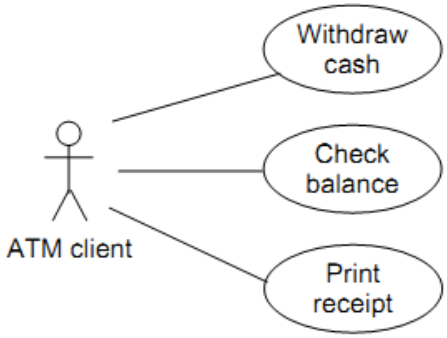


Рис. 2. Приклад діаграми прецедентів, що моделює поведінку підсистеми банкомату, обслуговуючої клієнта

На рис.2 підсистема банківського автомата, обслуговуючого клієнта, повинна реалізовувати три варіанти поведінки (прецеденту) з іменами: Withdraw cash (отримати

гроші), Check balance (перевірити суму на рахунку) і Print receipt (роздрукувати чек). Клієнт моделюється актором з ім'ям ATM client (клієнт банкомату).

Між акторами можна встановлювати відношення типу узагальнення-спеціалізація, і, отже, вводити поняття суперактор і субактор (рис. 3).

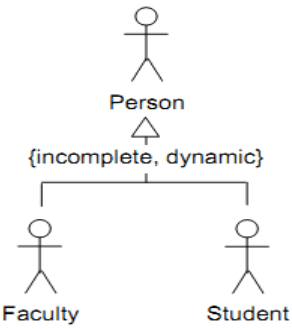


Рис.3 Відношення типу узагальнення-спеціалізація між акторами

Між прецедентами можна встановити відношення типу узагальнення-спеціалізація, для того, щоб показати, що деяка функція системи є загальнішою (чи суперфункцією) і розпадається на декілька спеціалізованих функцій (чи субфункций) Рис. 4.

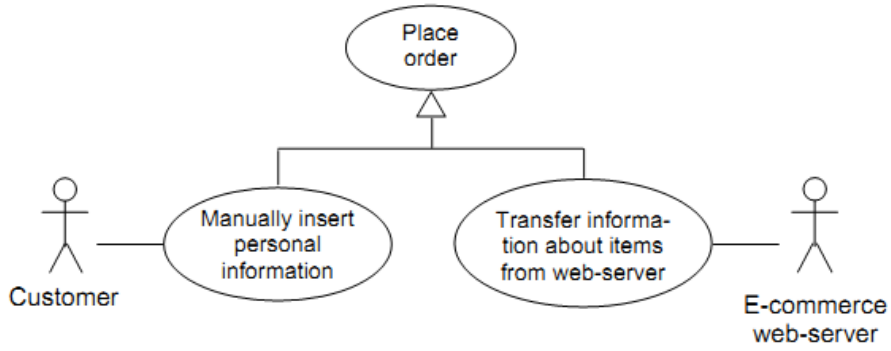


Рис.4 Використання відношення типу узагальнення-спеціалізація для прецедентів.

Діаграма прецедентів, зображена на рис. 4 моделює спрощений погляд на систему формування замовлення для випадку придбання товарів в електронному магазині.

Діаграма включає двох акторів: Customer (замовник, або користувач системи) і E - commerce Web – server (web- сервер системи електронної комерції).

З актором Customer асоційований прецедент Manually insert personal information (вручну вводить персональну інформацію), і це означає, що одна з функцій системи полягає в забезпеченні ручного введення персональної інформації замовником. З актором E - commerce асоційований прецедент Transfer information about items from web – server (передати інформацію про товари з web- сервера), що означає, що у функції системи входить також забезпечення передачі потрібної для замовлення інформації з серверного комп'ютера на клієнтський комп'ютер замовника.

За допомогою відношення типу узагальнення-спеціалізація моделюється той факт, що перераховані функції системи є спеціалізацією загальнішої функції Place order (розміщення замовлення).

Окрім відношення типу асоціація і узагальнення-спеціалізація між прецедентами може бути встановлена відношення типу залежність, уточнене за допомогою стереотипів.

У ряді випадків один прецедент включає функції іншого прецеденту. Оскільки наявність такого відношення важлива для подальшого проектування системи, воно має бути відображене на діаграмі. Для цього використовується відношення типу залежність, спрямоване на той прецедент, який моделює функцію, що включається . Характер залежності уточнюється стереотипом <<include>> (включення).

Окрім стереотипу <<include>> відношення типу залежність на діаграмі прецедентів часто забезпечується стереотипом <<extend>> (розширення).

Тоді як стереотип <<include>> означає, що один прецедент завжди включає інший прецедент, стереотип <<extend>> означає можливе розширення функцій деякого прецеденту.

Розглянемо **приклад.**

Оплата товару в магазині, як правило, може здійснюватися або готівкою, або за допомогою кредитної картки. За допомогою діаграми прецедентів легко змоделювати таку систему. Нескладно уявити, що акторами в такій системі виступають: Buyer (покупець), Cashier (касир) і Credit card system (система прийому платежів за допомогою кредитної картки), а основним прецедентом – Payment (оплата). На рис. 5 приведена діаграма прецедентів, що моделює описаний випадок.

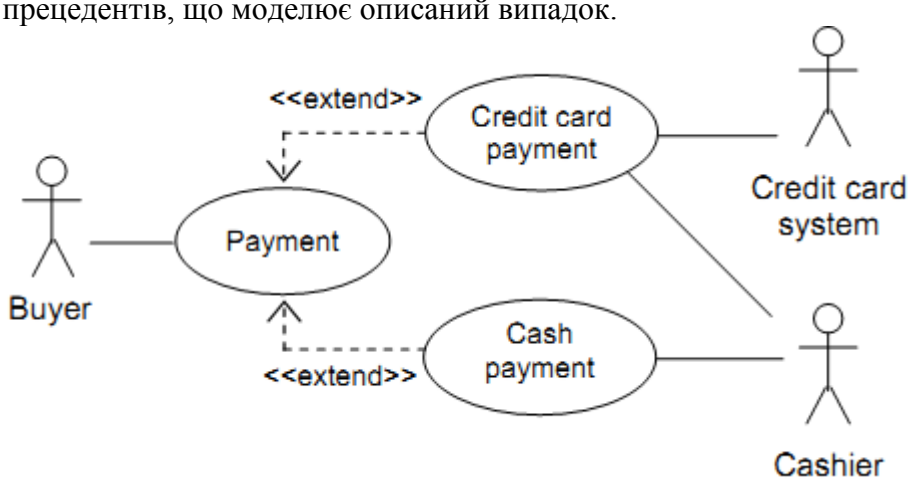


Рис. 5. Відношення типу залежність із стереотипом <<extend>> між прецедентами

Напрям відношення типу залежність вказує на прецедент, функції якого розширюються.

Порядок виконання роботи

1. Вивчити запропонований теоретичний матеріал і конспект лекцій.
2. Розробити діаграму класів, моделюючу структуру системи (згідно з варіантом). Використати відношення :
   - ✓ узагальнення-спеціалізація
   - ✓ асоціація
   - ✓ композиція
   - ✓ агрегація
   - ✓ залежність
   - ✓ реалізація.
3. Побудуйте діаграму прецедентів для вибраної інформаційної системи. При розробці діаграми використайте відношення типу асоціація і залежність із стереотипами <<include>> і <<extend>>.
4. Побудувати звіт. Зробити висновки.

Питання для самоконтролю

1. Характеристики класу
2. Префікси бачення
3. Які існують відношення між класами
4. Що таке методи класу
5. Що таке стеріотип

Лабораторна робота №5

Тема: Моделювання обміну повідомленнями між структурними елементами системи

Мета роботи: Ознайомлення з основними елементами проектування і моделювання програмних систем, що реалізуються шляхом обміну повідомленнями між об'єктами, за допомогою мови UML.

Теоретичний матеріал

Реалізація поведінки об'єктної системи здійснюється шляхом обміну повідомленнями між об'єктами. Повідомлення - це запит з боку об'єкту-посилача до об'єкту-мети про виконання деякого методу об'єкту-мети.

Діаграми взаємодії (interaction diagrams) дозволяють моделювати поведінку системи, що реалізовується шляхом обміну повідомлення між об'єктами.

У UML існує два типи діаграм взаємодії :

1. діаграма кооперації(collaboration diagram);
2. діаграма послідовності(sequence diagram).

Діаграма кооперації є варіантом діаграми об'єктів, на якій відображена логічна послідовність обміну повідомленнями між об'єктами.

Діаграма послідовності включає вісь часу і моделює тимчасову послідовність обміну повідомленнями між об'єктами.

Діаграма кооперації і діаграма послідовності взаємно конвертовані. Це означає, що будь-яка діаграма кооперації може бути перетворена в семантично еквівалентну діаграму послідовності і навпаки.

Повідомлення, використовувані в діаграмах взаємодії, класифікують як: синхронні (synchronous) і асинхронні (asynchronous).

Якщо об'єкт-посилач відправив синхронне повідомлення, то він повинен чекати закінчення виконання відповідного методу об'єктом-метою.

Якщо ж відправлено асинхронне повідомлення, то очікування закінчення виконання методу не обов’язкове.

Проста діаграма кооперації

На діаграмі кооперації повідомлення зображуються у вигляді стрілок, спрямованих від об'єкту-посилача до об'єкту-мети. Стрілка на графічному символі синхронного повідомлення зображується у вигляді зачорненого трикутника (див. рис. 1).

Графічний символ асинхронного повідомлення є "половинною" стрілкою, зображеною на рис. 1-а.

Об'єкт-посилач і об'єкт-мета, часто є екземплярами двох різних класів що знаходяться у відносинах типу асоціації.

Стрілка, що відповідає повідомленню, позначається ім'ям методу об'єкту-мети, за яким йде список фактичних вхідних аргументів і повертане значення.

Стрілка зображається уздовж лінії, що сполучає об'єкт-посилач з об'єктом-метою. Ця лінія моделює конкретну реалізацію відношення типу асоціація, встановленого між відповідними класами на діаграмі класів

Таким чином, одним із способів породження повідомлень в системі є їх незалежна генерація об'єктом-посилачем.

Проте об'єкт-посилач може отримати "завдання" згенерувати і відправити повідомлення об'єкту-мішені від деякого третього об'єкту.

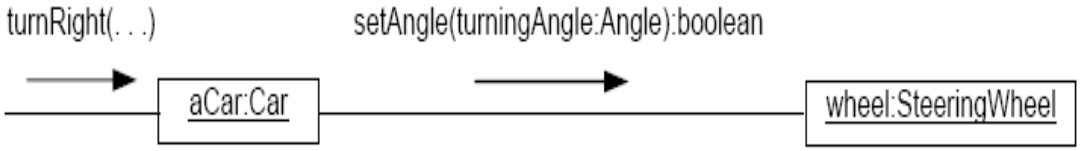


Рис. 1. Приклад простої діаграми кооперації

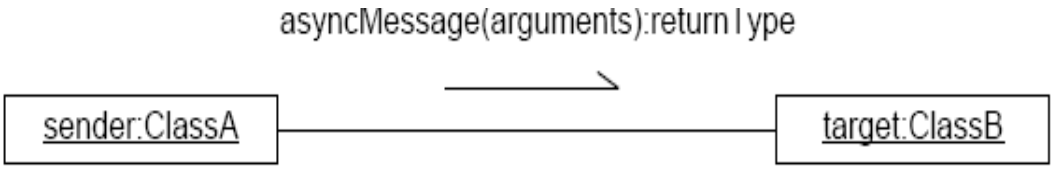


Рис. 1-а. Зображення асинхронного повідомлення на діаграмі кооперації

Діаграма, зображена на рис. 1, є ***діаграмою об'єктів*** (чи фрагмент діаграми об'єктів) що складається з двох об'єктів: aCar (деякий автомобіль), згенерований за допомогою класу Car (автомобіль), і wheel (колесо), згенерований за допомогою класу SteeringWheel (рульове колесо).

Між цими об'єктами встановлений зв'язок і, отже, передбачена можливість обміну повідомленнями. Як було відмічено раніше, цей зв'язок може розглядатися як конкретна реалізація відношення типу асоціація між класами Car і SteeringWheel.

Об'єкт aCar отримує синхронне повідомлення, що ініціює метод turnRight(повернути направо) цього об'єкту. Об'єкт посилач цього повідомлення невідомий. У відповідь на отримане повідомлення, об'єкт aCar генерує синхронне повідомлення і адресує його об'єкту wheel.

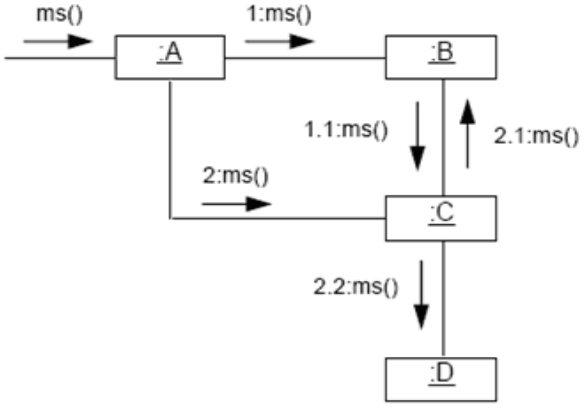
На рис. 1 повідомлення, адресоване об'єкту wheel, ініціює метод setAngle (встановити кут) і записане у виді: setAngle(turningAngle: Angle) : boolean

Метод setAngle отримує, в якості вхідного аргументу значення turningAngle (кут повороту) типу Angle, і повертає значення типу boolean.

Діаграма кооперації з вказівкою послідовності виконання повідомлень

Уздовж лінії, що сполучає об'єкти на діаграмі кооперації, впродовж деякого проміжку часу, можуть передаватися декілька повідомлень в обох напрямах. З неї легко зрозуміти зв'язки між об'єктами, проте, важче зрозуміти послідовність подій.

Рис. 2 ілюструє спосіб вказівки послідовності передачі повідомлень на діаграмі кооперації за допомогою системи багаторівневої нумерації.



Последовательность сообщений	Обозначение на рис.
Первое сообщение	ms()
Второе сообщение	1:ms()
Третье сообщение	1.1:ms()
Четвёртое сообщение	2:ms()
Пятое сообщение	2.1:ms()
Шестое сообщение	2.2:ms()

Рис.2. Вказівка послідовності передачі повідомлень на діаграмі кооперації

Багаторівнева нумерація повідомлень, приведена на рис. 2, читається таким чином:

1. Початковим (першим) повідомленням, отриманим об'єктом :A являється повідомлення ms.
2. У відповідь на повідомлення ms об'єкт :A генерує друге повідомлення 1: ms, адресуючи його об'єкту :B.
3. У відповідь на повідомлення 1: ms об'єкт :B генерує третє повідомлення 1.1: ms, адресуючи його об'єкту :C.
4. Четверте повідомлення позначене як 2: ms і передається від об'єкту :A до об'єкту :C.
5. У відповідь на це повідомлення об'єкт :C послідовно генерує два повідомлення: 2.1: ms і 2.2: ms, адресуючи їх об'єктам :B і :D відповідно.

Діаграми кооперації, зображені на рис.1, - 2, припускають безумовну генерацію повідомлень і передачу їх від об'єкту-посилача до об'єкту-мети. У ряді випадків повідомлення має бути передане тільки при виконанні деякої умови.

Розглянемо приклад діаграми кооперації на рис. 3

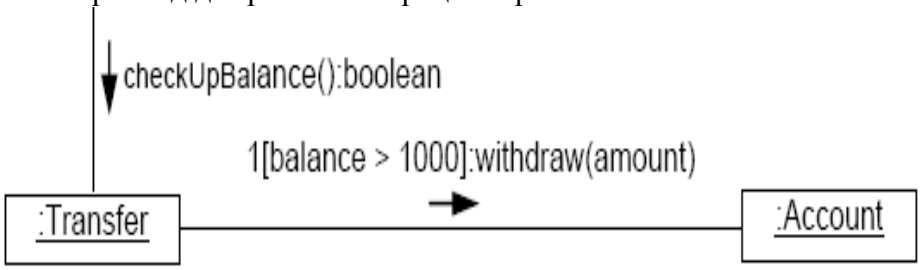


Рис.3. Зображення умовних повідомлень на діаграмі кооперації

Діаграма кооперації, приведена на рис. 3, описує наступну послідовність подій. Об'єкт :Transfer (переклад) отримує синхронне повідомлення, яке ініціює метод checkUpBalance (перевірити баланс) цього об'єкту. Метод checkUp - Balance повертає значення типу boolean, яке набуває значення рівне true у тому випадку, якщо виконується умова [balance > 1000].

В цьому випадку генерується повідомлення, адресоване об'єкту :Account (банківський рахунок) і що ініціює метод withdraw (зняти гроші з рахунку) об'єкту :Account.

Умова, при якій генерується повідомлення, ***записується у вигляді логічного виразу в квадратних дужках***, після номера повідомлення. Вираз в квадратних дужках має тип boolean і, тому, набуває тільки два значення true і false. Цей вираз часто називають вираження-захист або просто захист (guard). Захист є обмеженням, використовуваним в таких діаграмах як діаграма кооперації, діаграма послідовності і діаграма станів.

За допомогою виразу-захисту можна легко зображувати два взаємовиключних спрямування («розвилки») передачі повідомлень на діаграмі співробітництва. Для цього необхідно використовувати два умовних повідомлення з одним і тим же захистом з запереченням і без нього. Доповнимо діаграми, наведеної на рис.3, напрямком, моделюючим випадок, коли обчислене значення методу checkUpBalance стає рівним false. Діаграма, відповідна ситуації наведена на рис.4.

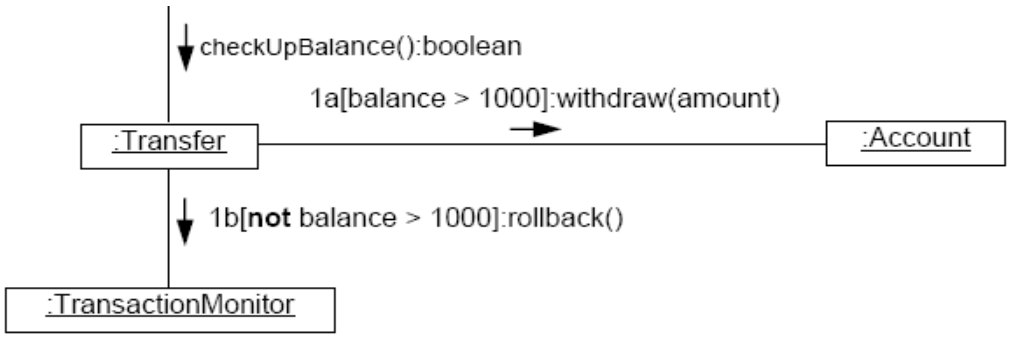


Рис. 4. Зображення «розвилки» на діаграмі кооперації

Захист, на рис.4, записаний таким чином, що якщо один захист приймає значення true, то інший - значення false. Повідомлення, з номерами 1a і 1b є альтернативними. У тому випадку, якщо баланс на рахунку менше 1000, то об'єкту :TransactionMonitor (монітор транзакцій) передається повідомлення, яке ініціює метод rollback (повернути в початковий стан).

Ітераційні повідомлення

Ітераційним повідомленням (*iterated* message) будемо називати таке повідомлення, яке послідовно передається декільком об'єктам-цілям.

Набір об'єктів, яким передається ітераційне повідомлення, називається мультиобъектом (multiobject). Мультиобъектом може бути, наприклад, безліч об'єктів класів-конституэнтов**.**

Графічно на діаграмі кооперації мультиобъект зображується у вигляді «подвійного» прямокутника, як це показано у правій частині рис. 5.

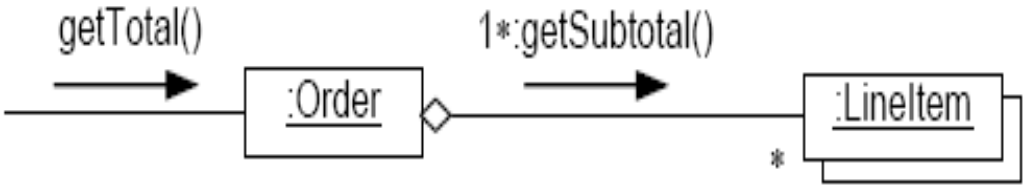


Рис. 5. Повідомлення getSubtotal() розсилається ітераційно всім елементам мультиобъекта :LineItem

Діаграма на рис. 5 моделює ситуацію, коли певний об'єкт класу-агрегату Order (замовлення) отримує синхронне повідомлення, яке ініціює метод get-Total (отримати загальну вартість).

Відповідь на отримане повідомлення, об'єкт :Order посилає ітераційне повідомлення 1*:getSubtotal (отримати вартість) мультиобъекту :LineItem (товари, що продаються).

Ітераційне повідомлення забезпечується префіксом у вигляді символу «*» (зірочка, що позначає, множинність), який слідує за порядковим номером повідомлення.

Розсилка ітераційного повідомлення здійснюється в циклічному процесі. Параметри циклу можуть бути явно вказані при запису ітераційного повідомлення.

Діаграма на рис. 6 відрізняється від діаграми на рис. 5 тим, що ітераційне повідомлення забезпечено ітеративним оператором у квадратних дужках, який задає параметри циклу.

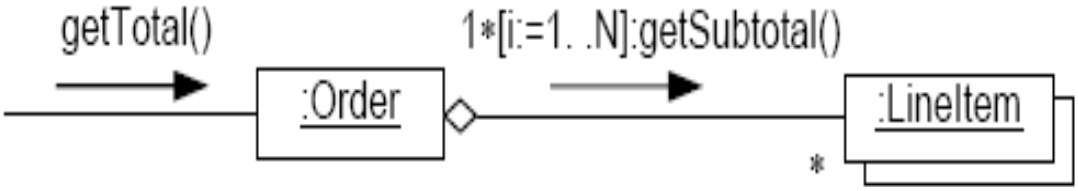


Рис. 6. Ітераційне повідомлення з зазначенням параметрів циклу

Діаграма послідовності

Діаграма послідовності пов'язує між собою: безліч взаємодіючих об'єктів, безліч повідомлень, що циркулюють між об'єктами і безліч методів, що викликаються повідомленнями.

Діаграма послідовності показує, яким чином розгортається у ***часі*** процес обміну повідомленнями між об'єктами системи. Структура діаграми послідовності наведена на рис.7.

Всі дійові особи показано у верхній частині діаграми. Стрілки відповідають повідомленям, переданим між дійовою особою і об'єктом або між об'єктами для виконання необхідних функцій.

На діаграмі послідовності об'єкт зображується у вигляді прямокутника, від якого вниз проведена вертикальна пунктирна лінія. Ця лінія називається ***лінією життя*** (lifeline) об'єкта (див.нижче). Вона являє собою фрагмент життєвого циклу об'єкта в процесі взаємодії.

Кожне повідомлення представляється у вигляді стрілки між лініями життя двох об'єктів. Повідомлення з'являються в тому порядку, як вони показані на сторінці зверху вниз. Кожне повідомлення позначається як мінімум ім'ям повідомлення.

Основним структурним елементом діаграми послідовності є ***графічний символ об'єкта***, з індивідуальною віссю часу, званою ***лінією життя об'єкта*** (object lifeline).

Лінія життя об'єкта зображується пунктирною лінією і спрямована вертикально вниз. Точка виходу лінії життя з графічного символу об'єкта відповідає нульового моменту часу.

На лінії життя об'єкта розташовуються ***графічні символи*** ініційованих методів, що представляють собою ***вертикальні прямокутники***.

Графічні символи методів розташовуються у порядку їх виклику. Довжина прямокутника пропорційна тривалості виконання методу.

Повідомлення моделюються ***суцільними горизонтальними лініями*** зі стрілками, спрямованими від методу об'єкта-відправника до методу об'єкта-цілі.

Повідомлення підписуються іменами методів, що викликаються в об'єкті-цілі. Синхронні повідомлення на діаграмі послідовності зображаються точно також, як і на діаграмі кооперації.

На діаграмі послідовності можна зображати значення, що повертаються, за допомогою пунктирних ліній зі стрілками.

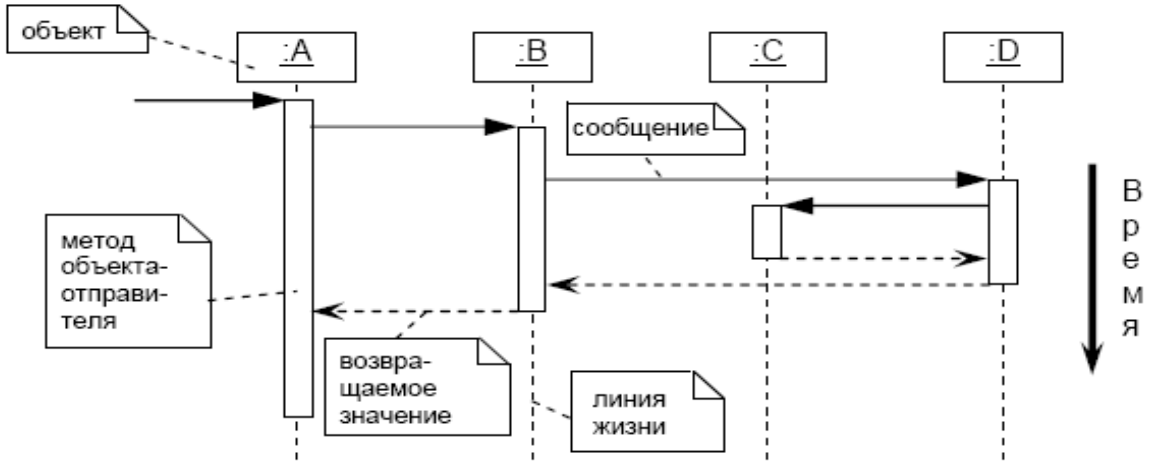


Рис. 7. Базові структурні елементи діаграми послідовності

Діаграма послідовності, зображена на рис.7, моделює тимчасову послідовність обміну повідомленнями між чотирма об'єктами: :A :B :C :D, а також послідовність виклику методів цих об'єктів.

З діаграми випливає, що

деякий метод об'єкта :A передає повідомлення об'єкту :B,

яке викликає відповідний метод цього об'єкта.

Почавши працювати, метод об'єкта :B передає повідомлення об'єкту :D,

який, у свою чергу, передає повідомлення об'єкту :C.

Пунктирні стрілки на діаграмі показують, що всі методи повертають деякі значення.

З діаграми також випливає, що всі передані повідомлення - синхронні, а також, що найбільш тривалим методом є метод об'єкта :A.

На рис. 7 всі об'єкти розташовуються на одній горизонталі. Це означає, що їх існування почалося одночасно, і жоден з них не був згенерований в процесі обміну повідомленнями.

Якщо в процесі обміну повідомленнями створюється новий об'єкт (наприклад, за допомогою методу create (створити)), то його графічний символ розташовується на горизонталі, що відповідає моменту його створення. Діаграма на рис. 8 ілюструє спосіб зображення згенерованого об'єкта.

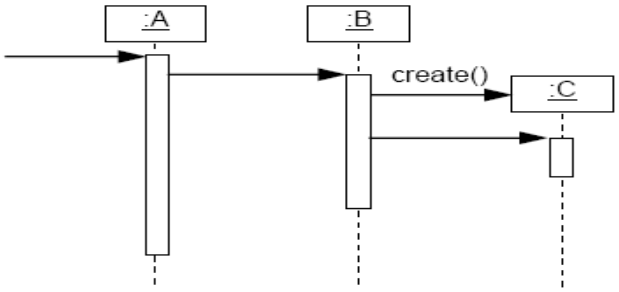


Рис. 8. Зображення створення об'єкта на діаграмі послідовності

На рис. 8 показано, що певний метод об'єкта :B спочатку генерує новий об'єкт класу C, а потім ініціює один з методів цього об'єкту шляхом передачі йому повідомлення.

Умовні повідомлення можуть породжувати два альтернативних повідомлення, або розвилку на діаграмі кооперації.

Рис. 9 ілюструє, яким чином альтернативні повідомлення і розвилка зображуються на діаграмі послідовності. Метод об'єкта :A, в залежності від значення виразу-захисту a (true або false) породжує два альтернативних повідомлення. Одне з цих альтернативних повідомлень викликає метод в об'єкті :B, а інше - в об'єкті :C.

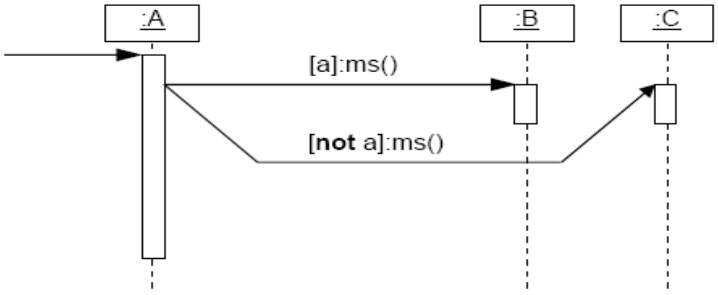
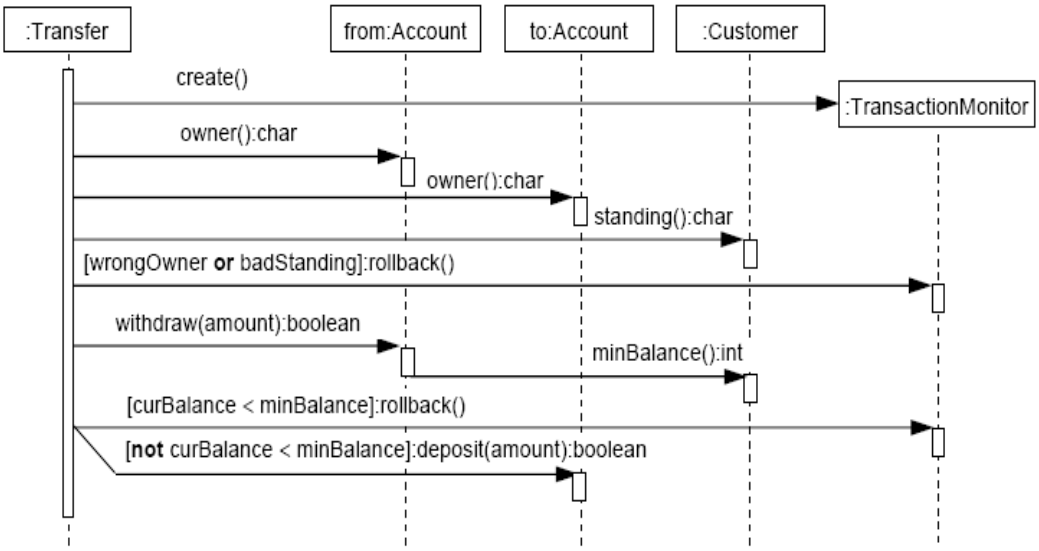


Рис.9. Зображення альтернативних повідомлень на діаграмі послідовності

На рис. 10 приведена діаграма послідовності, що моделює процес переказу грошових коштів одного і того ж клієнта з одного банківського рахунку на інший.



:T

Рис. 10. Діаграма послідовності, що моделює процес переказу грошових коштів

З діаграми, наведеної на рис. 10, випливає, що анонімний об'єкт :Transfer, який виконує функції керуючого елемента системи, що виконує певний метод (зображений у вигляді довгого вертикального прямокутника в лівій частині рис. 10) з переказу грошових коштів з банківського рахунку, модельованого об'єктом from:Account, на банківський рахунок, який моделюється об'єктом to:Account.

Будемо, для визначеності, називати цей метод makeTransfer. Будемо вважати, що раніше інший метод об'єкта :Transfer (наприклад, setUpTransfer) детермінував номери рахунків, залучених в операцію, розмір пересилаємих грошових коштів і т.д.

Метод makeTransfer створює об'єкт :TransactionMonitor, необхідний для спостереження за виконанням транзакції. Моніторинг транзакцій необхідний, оскільки у разі виникнення помилки, система повинна повернути в первісний стан частково виконані всі зміни за допомогою методу rollback. Наприклад, не можна допустити, щоб гроші були зняті з рахунку, але не поміщені на інший рахунок.

Два наступних повідомлення викликають метод owner в об'єктах from:Account і to:Account. За допомогою цих методів об'єкт :Transfer переконується, що обидва рахунки належать одному і тому ж клієнтові. Таке повідомлення викликає метод standing, за допомогою якого об'єкт :Transfer визначає репутацію клієнта. Якщо з'ясовується, що рахунки належать різним клієнтам, або клієнт має погану репутацію, то пересилається умовне повідомлення, що викликає метод rollback.

Наступне повідомлення викликає метод withdraw в об'єкті fromt:Account, який, у свою чергу, викликає метод minBalance, для перевірки того, що зняття грошей з рахунку не зменшить загальну суму на рахунку менше дозволеного мінімуму.

Якщо, в результаті зняття грошей, залишок на рахунку перевищує дозволений мінімум, то чергове повідомлення викликає метод deposit, який збільшує суму, збереженої на to:Account на величину amount. В іншому випадку (залишок на рахунку менше дозволеного мінімуму) викликається метод rollback, який відміняє всі зроблені зміни і повертає систему в початковий стан.

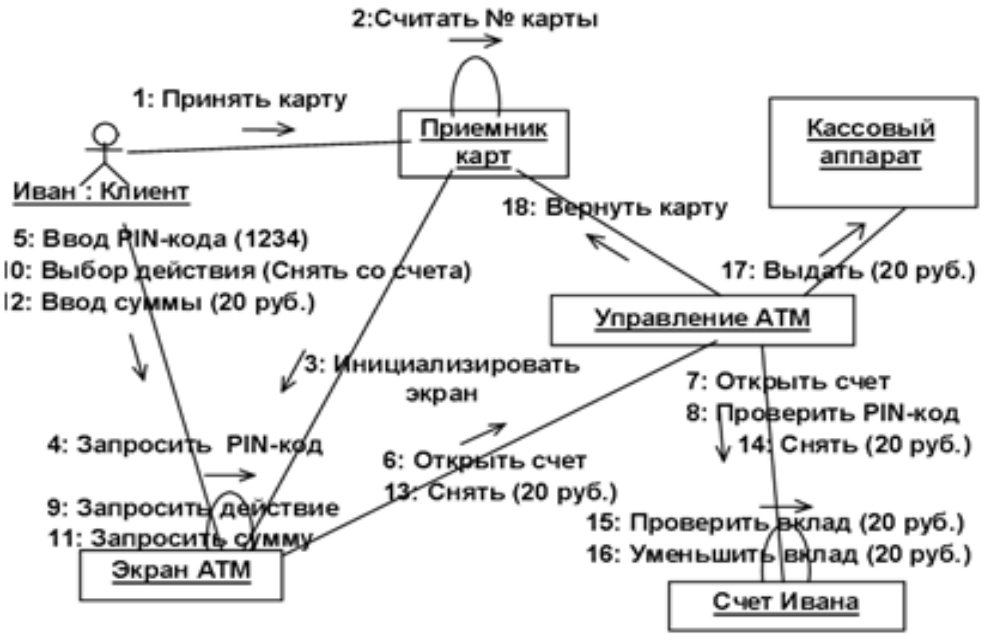


Рис. 11. Прилад діаграми кооперації

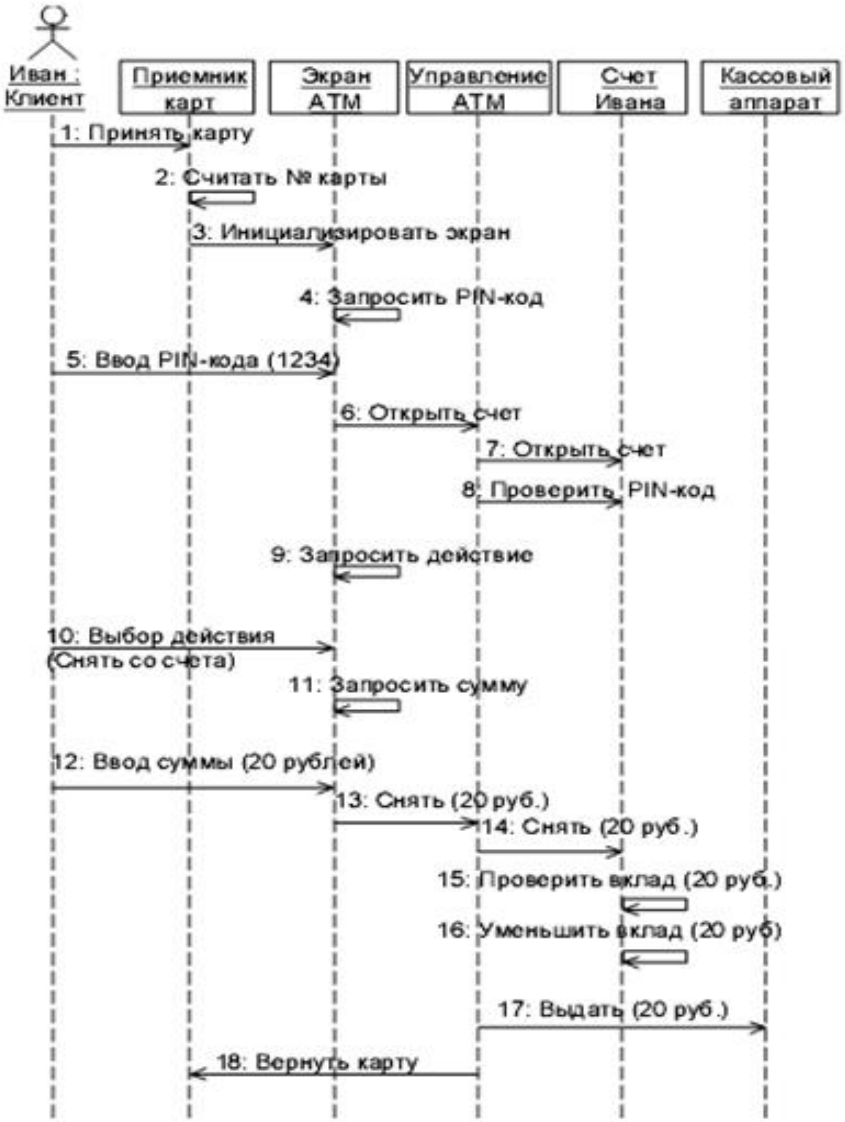


Рис.12 Прилад діаграми послідовності

Порядок виконання роботи

1. Вивчити запропонований теоретичний матеріал
2. Розробіть модель можливої поведінки системи, зображеної у вигляді діаграми класів в лабораторній роботі № 4. Представте цю модель у вигляді діаграми кооперації.
   2.1 Відобразити синхронні і асинхронні повідомлення.
   2.2 Використовувати багаторівневої системи нумерації.
   2.3 Використовувати умовні повідомлення на діаграмі кооперації.
   2.4 Продемонструвати використання ітераційних повідомлень і мультиобъектов.
3. Розробіть модель можливої поведінки системи, зображеної у вигляді діаграми класів в лабораторній роботі № 4 та описаної в лабораторній роботі № 1. Представте цю модель у вигляді діаграми послідовності.
   4. Побудувати звіт. Зробити висновки.

Питання для самоконтролю

1. Призначення лінії життя
2. Відмінність діаграми послідовності від діаграми кооперації
3. Призначення діаграми послідовності
4. Призначення діаграми кооперації

Лабораторна робота №6

Тема: Моделювання поведінки системи за допомогою діаграми діяльності та моделювання поведінки структурного елементу системи за допомогою діаграми станів.

Мета роботи: Ознайомлення з елементами моделювання поведінки програмних систем і їх елементів, що реалізовується за допомогою мови UML.

Теоретичний матеріал

Поведінка системи можна розглядати як покрокову діяльність, що приводить до досягнення мети. У найпростішому випадку діяльність в межах шага- це виконання одного методу. У загальному випадку крок - це виконання певної, логічно завершеною діяльності.

Модель поведінки системи може бути представлена послідовно-паралельними потоками виконуваних діяльностей. Графічне представлення послідовно-паралельних потоків виконуваних діяльностей називається діаграма діяльності (activity diagram).

Діаграма діяльності являє собою безліч графічних символів станів діяльностей, пов'язаних графічними символами переходів, що реалізують відношення типу «попередній наступний» на безлічі діяльностей. Перехід до наступної діяльності здійснюється після завершення попередньої діяльності.

У даній діаграмі не важливо, який об'єкт і за допомогою якого повідомлення ініціює той чи інший метод, важлива логічна послідовність виконуваних методів.

Графічний символ стану діяльності являє собою овал, у якого верхня і нижня частини - горизонтальні. Усередині графічного символу стану діяльності записується ім'я діяльності.

Для розрізнення простих і складних видів діяльності використовують такі терміни: стан діяльності та стан дії.

Стан дії (action state) є атомарної активністю, воно не може бути розділене на більш прості активності, а процес виконання стану дії не може бути перерваний. Стан дії - це стан виконання деякого методу.

Стан діяльності (activity state) не є атомарної активністю, воно може бути розбите на кілька простіших діяльностей і представлено окремою діаграмою діяльності, а процес

виконання стану діяльності може бути перерваний. Стан дії є окремим випадком стану діяльності.

Існують два спеціальних символу стану діяльності: символ початкового стану (initial state) і символ кінцевого стану (final state). Символ початкового стану зображується у вигляді жирної крапки . Символ кінцевого стану зображується у вигляді жирної крапки, укладеної в коло .

Графічний символ переходу (transition) являє собою стрілку, спрямовану від символу попередньої діяльності до символу подальшої діяльності.

Для моделювання переходу до однієї з двох альтернативних діяльностей використовується символ розгалуження (branching), що представляє собою ромб з одним вхідним переходом і двома вихідними переходами. Для моделювання точки завершення розгалуження використовується символ з'єднання, що представляє собою ромб з двома вхідними переходами і одним вихідним переходом.

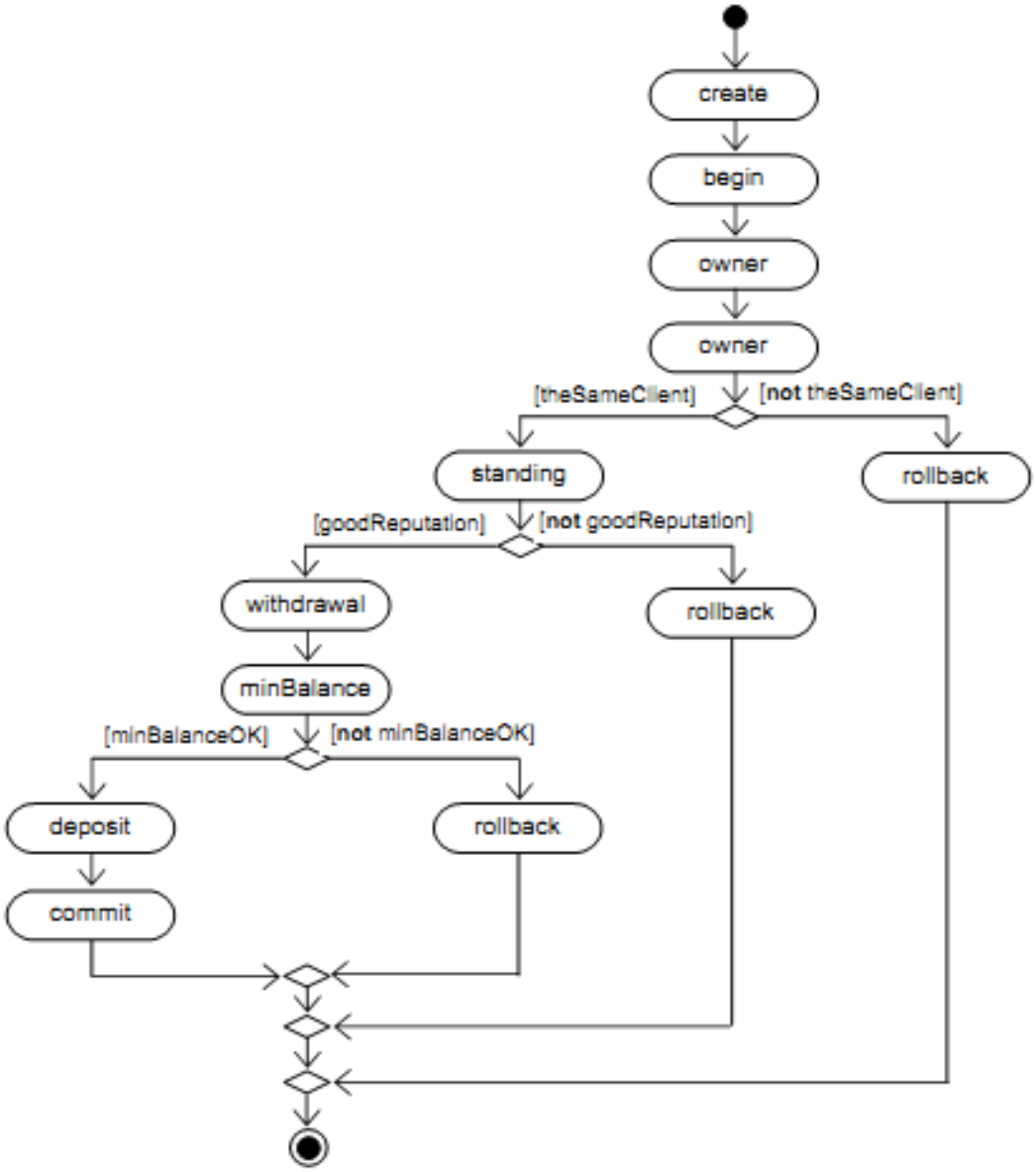


Рис. 1. Діаграма діяльності

Альтернативні переходи, які виходять із графічного символу розгалуження надписуються логічними виразами захисту в квадратних дужках. Вирази захисту повинні бути складені таким чином, що якщо одне з них приймає значення true, то інше - значення false.

Однією з істотних відмітних характеристик діаграми діяльності є можливість моделювання не тільки послідовних, але і паралельних потоків діяльностей / дій.

Діаграма на рис. 2 показує, яким чином на діаграмі діяльності зображуються паралельні потоки діяльностей.

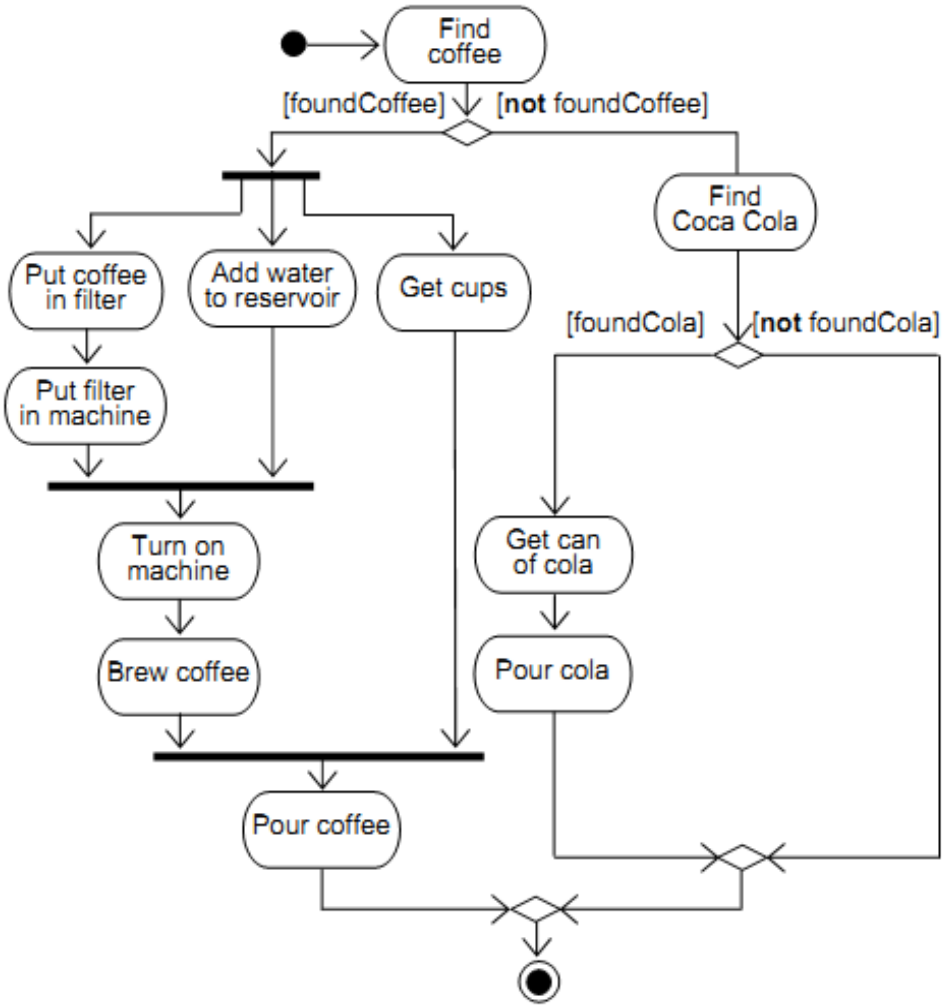


Рис. 2. Моделювання паралельних потоків діяльностей на діаграмі діяльності

Діаграма, наведена на рис. 2, моделює поведінку кухонного робота, який отримав завдання приготувати один з двох напоїв: каву або кока-колу.

Ім'я стану діяльності записують у вигляді короткої фрази англійською мовою, яка передає семантику діяльності.

Жирна горизонтальна лінія називається лінією синхронізації (synchronization bar). Лінії синхронізації використовуються при моделюванні паралельних дій. Розрізняють два типи ліній синхронізації:

Лінія поділу (forking synchronization bar) використовується для вказівки точки, починаючи з якої можливо паралельне і одночасне виконання діяльностей, а лінія злиття (joining synchronization bar) - для вказівки точки, починаючи з якої виконання діяльностей здійснюється послідовно.

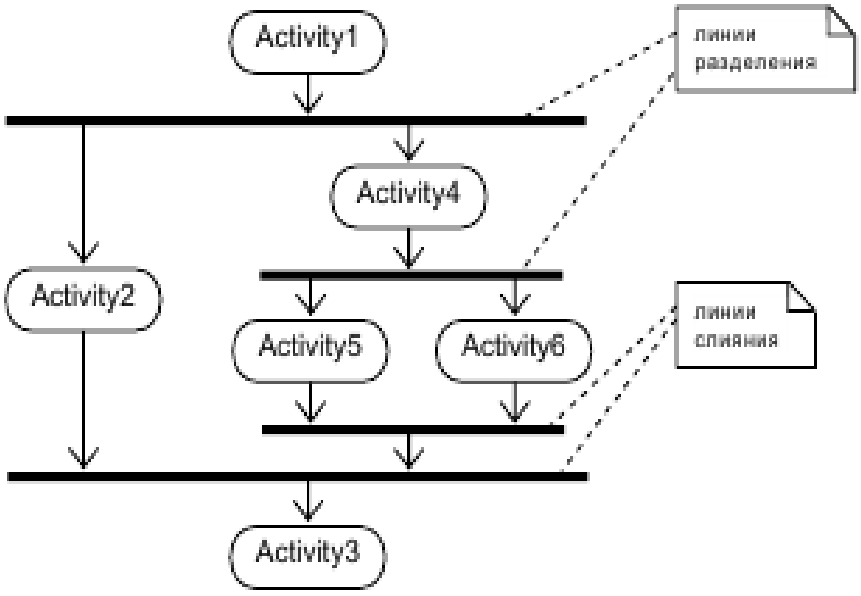


Рис. 3. Лінії синхронізації на діаграмі діяльності

Лінія поділу має один вхідний перехід і кілька вихідних.

Лінія злиття має кілька вхідних переходів і один вихідний.

Як правило, на діаграмах діяльності зберігається баланс ліній поділу та ліній злиття. Баланс означає, що, в межах однієї діаграми діяльності, кількість ліній поділу дорівнює кількості ліній злиття.

Можливість паралельного виконання діяльностей не обов'язково означає, що паралельно зображені діяльності, повинні виконуватися паралельно, вони можуть виконуватися послідовно і в довільному порядку.

Будь-яка діяльність може бути уточнена і представлена більш докладною діаграмою діяльності. Процедура подання складної діяльності кількома простими діяльностями називається декомпозиція діяльності.

Стан діяльності, яке піддається декомпозиції, називають композитним станом діяльності, а стану, на які розбивається композитне стан - станами – компонента.

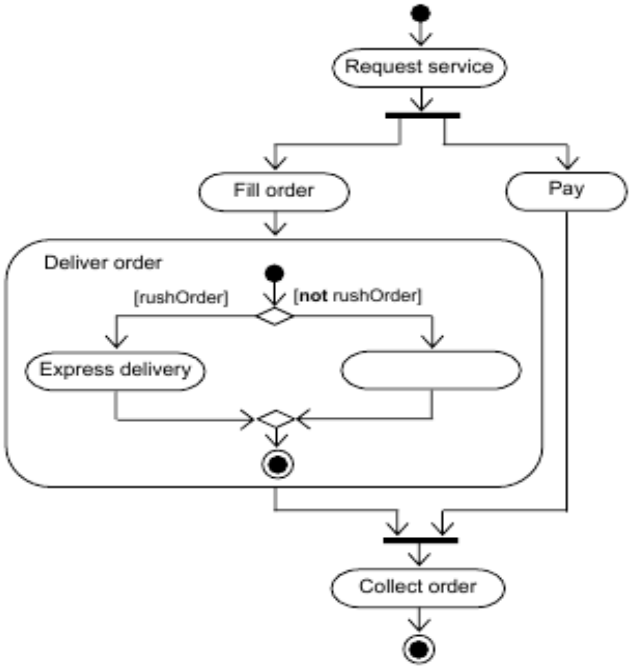


Рис. 4. Подання уточненого стану Deliver order у вигляді композитного стану

На рис.4 бачимо, що стан Deliver order (доставити замовлення) представимо у вигляді двох станів-компонентів: Express delivery (експрес доставка) і Regular delivery (звичайна доставка). Альтернативність вказується за допомогою символу розгалуження і виразів захисту: [rushOrder] (термінове замовлення), [not rushOrder] (Не термінове замовлення).

Графічний символ композитного стану зображується у вигляді прямокутника з закругленими кутами. Ім'я композитного стану розташовується у верхній частині графічного символу композитного стану. Усередині символу розташовується результат декомпозиції у вигляді діаграми діяльності.

Для відображення діяльності, що виконується ітераційно, UML дає можливість уявити ітераційне виконання діяльності без явного зображення петлі циклу (див. Рис.5). Діяльність Fill order line (заповнити рядок замовлення) - багаторазово повторюється діяльність.

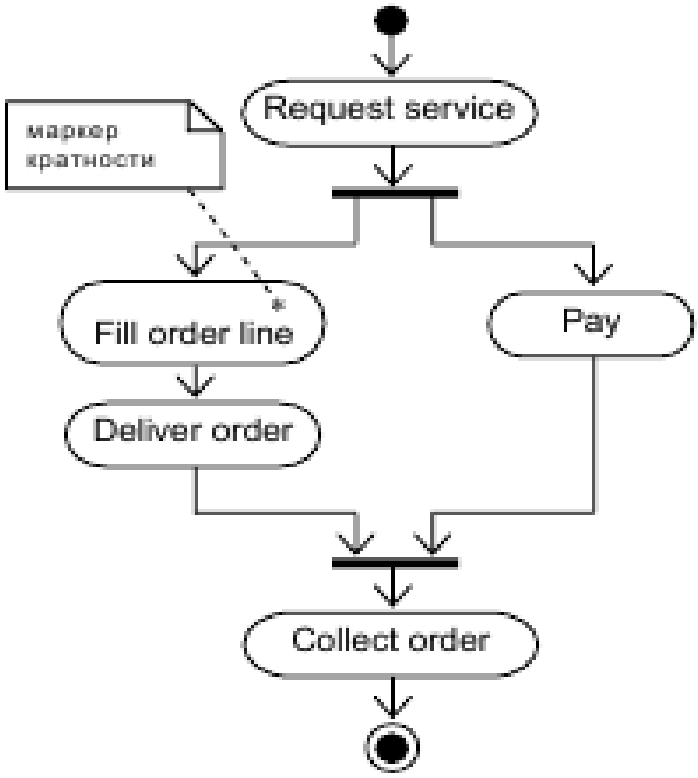


Рис. 5 Спосіб зображення ітераційної діяльності

Символ «зірочка» у верхній правій частині графічного символу діяльності вказує на те, що ця діяльність є ітераційної, отже, перехід до діяльності Deliver order (доставити замовлення) буде виконаний тільки в тому випадку, коли заповнені всі рядки замовлення.

Діаграма діяльності, дозволяє моделювати покрокову і цілеспрямовану поведінку системи, що складається з декількох структурних елементів: об'єктів, класів, пакетів та ін.

Моделювання поведінки структурного елементу системи за допомогою діаграми станів

Діаграми станів, дають можливість окреслити покрокову і цілеспрямовану поведінку одного структурного елементу системи, в якості якого найчастіше виступає об'єкт.

Діаграми станів визначають всі можливі стани, в яких може перебувати конкретний об'єкт, а також процес зміни станів об'єкта в результаті настання деяких подій.

Базові поняття діаграми станів

Стан (state) об'єкта - це сукупне значення його полів для деякого моменту часу.

Стан об'єкта зручно представляти як точку в n-вимірному просторі станів (state space), де n - кількість полів об'єкта. Кожне з полів об'єкта визначає один з вимірів простору станів.

Ім'я поля задає ім'я вимірювання, тип поля - задає безліч точок уздовж цього виміру.

Поведінка об'єкта може розглядатися як послідовність переходів з попереднього стану в наступний, або як переміщення репрезентативної точки (точки, що представляє стан об'єкта в даний момент часу) в просторі станів об'єкта.

Розглянемо поняття простір станів. Нехай є прямокутний брусок, про який відомі колір, вага і температура. Тоді, зазначений брусок може моделюватися об'єктом, що є екземпляром класу (рис.6)

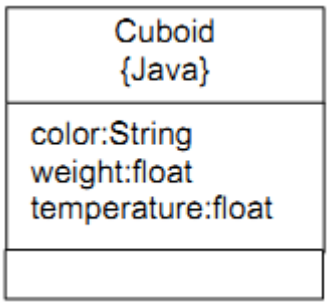


Рис. 6. Клас прямокутних паралелепіпедів (Cuboid) з полями: color (колір), weight (вага) і temperature (температура)

Нехай в деякий момент часу, за допомогою класу Cuboid, згенерований об'єкт bar1: Cuboid. Поведінка цього об'єкта може розглядатися як рух репрезентативної точки в тому ж просторі станів - просторі станів класу Cuboid.

Нехай, в момент генерації об'єкта, його поля проініціалізовані наступним чином: color = "yellow"; weight = 10; temperature = 24;

Ці значення задають початковий стан об'єкта, якому відповідала би репрезентативна точка в просторі станів з координатами:

bar1 < "yellow", 10, 24>

Об'єкт може знаходитися в цьому стані невизначено довго, до тих пір, поки не відбудеться подія, що змінює значення одного або декількох полів. Після завершення такої дії, стан об'єкта зміниться, що призведе до переміщення репрезентативної точки bar1 в просторі станів.

Кроком поведінки називають одноразове переміщення репрезентативної точки в просторі станів.

Діаграмою станів (state diagram) називають цілеспрямоване і послідовне поведінку об'єкта, що включає безліч станів, необхідне і достатнє для досягнення мети, безліч переходів між станами і безліч дій, що викликають перехід з попереднього стану в наступне.

Діаграма станів дає графічне представлення про рух репрезентативної точки в просторі станів, від початкового стану до цільового.

Простір станів може бути безперервним, тобто включати одне або кілька вимірів з значеннями, що безперервно змінюються. Безперервні вимірювання простору станів породжуються полями з числовими типами. Це має на увазі нескінченне (або дуже велике) кількість значень цих полів. Ясно, що діаграма станів, що включає велику кількість станів стає практично непридатною.

Спрощення поняття станів зводиться до того, що воно визначається не по відношенню до всіх полів класу, а по відношенню до деяких «обраних» полях, які назвають полями стану (state fields). Поле стану повинно володіти такими властивостями:

- безліч значень поля невелика (не більше десяти);
- поле описує істотну якість класу / об'єкта.

Діаграми станів UML описують поведінку класу / об'єкта по відношенню до одного або декількох полях станів.

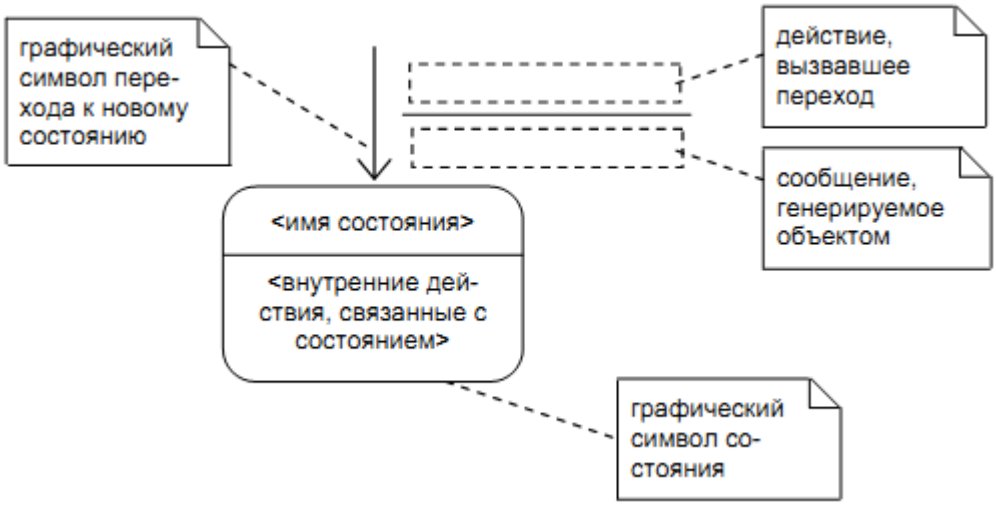


Рис. 7. Зображення кроку поведінки на діаграмі станів

Графічний символ переходу (transition), на діаграмі станів, має точно такий же вигляд, як і на діаграмі діяльності - стрілка, спрямована від попереднього стану до наступного. Перехід надписується спеціальним чином. Напис складається з двох частин, розділених горизонтальною лінією.

Над горизонтальною лінією записується подія, яка викликала перехід від попереднього стану до даного стану. Якщо над лінією вказано кілька подій, то будь-яка з них викликає перехід. Ця частина напису є обов'язковою.

Під горизонтальною лінією записується повідомлення, що генерується об'єктом, при переході в новий стан. Якщо під лінією вказано кілька повідомлень, то всі вони генеруються при переході в новий стан. Ця частина напису не є обов'язковою.

Графічний символ стану являє собою прямокутник із закругленими кутами, розділений горизонтальною лінією на два відділення. У верхньому відділенні записується ім'я стану. Ім'я стану являє собою значення поля стану, по відношенню до якого розглядається поведінка об'єкта. У нижньому відділенні можуть записуватися внутрішні (internal) дії, пов'язані зі станом. Ці дії називаються внутрішніми, оскільки вони пов'язані тільки з даним станом і виконуються незалежно від подій, що призводять до переходу від одного стану до іншого. Існують кілька типів внутрішніх дій. Внутрішні дії записуються в наступній нотації:

entry / <action1>
do / <action2>
exit / <action3>

Дія <action1> виконується відразу ж після того, як об'єкт перейшов в даний стан, дію <action2> виконується весь час, поки об'єкт знаходиться в даному стані, а дія <action3> виконується відразу ж після того, як об'єкт залишає дане стан.

Приклад простої діаграми станів. Ця діаграма моделює поведінку об'єкту класу Goods (товар) у просторі станів, що задається значеннями єдиного поля стану inspectionStatus (статус інспектування товару). Це поле вибрано в якості поля стану, оскільки воно є істотним для класу Goods, і приймає невелику кількість значень:

- received (товар отриманий);
- inInspection (товар інспектується);
- accepted (товар прийнятий);
- rejected (товар відхилений).

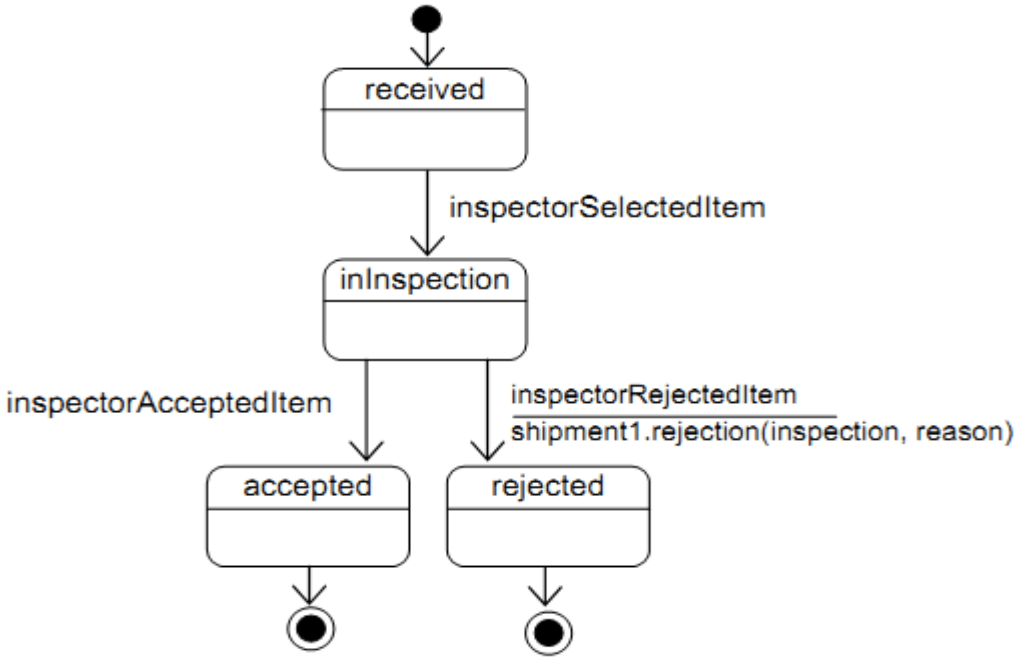


Рис. 8. Приклад діаграми станів для класу, поведінка якого описуються значеннями єдиного поля inspectionStatus

Діаграма, наведена на рис. 8 описує поведінку класу, метою якого є досягнення одного з двох станів: accepted (товар прийнятий) або rejected (товар відхилений).

Вважається, що з початкового стану об'єкт миттєво переходить в наступний стан. Простір станів об'єкта класу Goods по відношенню до значень поля inspectionStatus складається з чотирьох стійких станів. Кожен з стійких станів зображено окремим символом стану з ім'ям, тотожним значенням поля inspectionStatus.

На рис. 8 кожен з переходів (за винятком переходу з початкового стану) надписаний подією, що викликає відповідний перехід, і тільки один з переходів позначений також повідомленням, що генерується при переході зі стану inInspection в стан rejected. Це повідомлення адресовано методу rejection (відхилення) об'єкта shipment1 (поставка 1), якому, у вигляді аргументів, передається інформація про поточне інспектування (inspection) і причини відхилення товару (reason).

Перехід зі стану received в стан inInspection здійснюється в тому випадку, якщо відбувається подія inspectorSelectedItem (інспектор вибрав товар). Зі стану inInspection можливі два переходи: (1) в стан accepted, якщо відбулася подія inspectorAcceptedItem (інспектор прийняв товар); і (2) в стан rejected, якщо відбулася подія inspectorRejectedItem (інспектор відхилив товар). Обидва зазначених стану є цільовими, що відзначено символами кінцевого стану.

Порядок виконання роботи

1. Вивчити запропонований теоретичний матеріал
2. Розробіть модель можливої поведінки системи. Представте цю модель у вигляді діаграми діяльності.
   2.1 Показати паралельні потоки діяльностей.
   2.2 Вказати 2 типи лінії синхронизации.
   2.3 Використовувати композитні стани діяльності.
3. Розробіть модель поведінки структурного елементу системи за допомогою діаграми станів.
4. Побудувати звіт. Зробити висновки.

Питання для самоконтролю

1. Характеристика діаграми діяльності
2. Характеристика діаграми станів
3. Що таке композитні стани діяльності
4. Що таке паралельні потоки діяльностей

Лабораторна робота №7

Тема: Моделювання засобів, за допомогою яких специфицируется програмна реалізація системи.
Мета роботи: Ознайомлення з елементами моделювання архітектури системи, реалізовані на мові UML за допомогою діаграми компонентів.

Теоретичний матеріал

Діаграма компонентів використовується для моделювання структури системи, представленої у вигляді безлічі фізичних програмних компонентів різної природи і відносин між ними.

Наприклад, відносини між компонентами, що представляють собою вихідний код програми і компонентом, що представляє її виконуваний код, або відношення між програмним процесором і підлеглими йому програмними драйверами.

Діаграма компонентів

За допомогою діаграми компонентів моделюється статична структура програмної системи, і в цьому сенсі діаграма компонентів ідеологічно близька до діаграми класів. Однак, на відміну від діаграми класів, яка відображає логічну структуру системи, представлену абстракціями, які називали класами, діаграма компонентів моделює фізичну структуру системи у вигляді безлічі фізичних програмних компонентів, що використовуються на етапі її реалізації. Тому, відносини між програмними компонентами, на діаграмі компонентів можуть моделюватися за допомогою відомих нам типів відносин, розглянутих раніше, однак частіше за інших використовується відношення типу залежність.

Компонентом може бути така фізична одиниця програмної системи, як проект, пакет, специфікації, файл. Діаграма компонентів часто використовується в наступних випадках:

- моделювання безлічі компонентів, між якими розподілено вихідний код програмної системи, і відносини між ними;
- моделювання структури сімейства версій програмної системи;
- моделювання безлічі компонентів, які компілюються в виконуваний код.

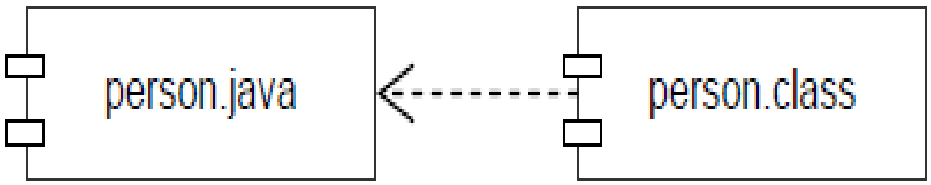


Рис.1 Приклад простий діаграми компонентів ілюструє графічний символ компонента зі ставленням типу залежність

Графічним символом програмного компонента на діаграмі компонентів є прямокутник з двома маленькими прямокутниками, розташованими на одній з його бічних сторін (часто на лівій стороні). Усередині прямокутника записується або просте, або складене ім'я компонента.

Наприклад, простим ім'ям компонента може бути ім'я файлу. Діаграма на рис. 1 показує, що програмний компонент з ім'ям person.java, (байт код класу Person) отримано шляхом компіляції програмного компонента з ім'ям person.class (вихідний код класу Person) і залежить від останнього.

Графічні символи відносини типу залежність на діаграмі компонентів можуть бути надписані за допомогою стандартних або додаткових стереотипів.

На рис.2 приведена діаграма компонентів, що ілюструє використання стереотипів для відносини типу залежність.

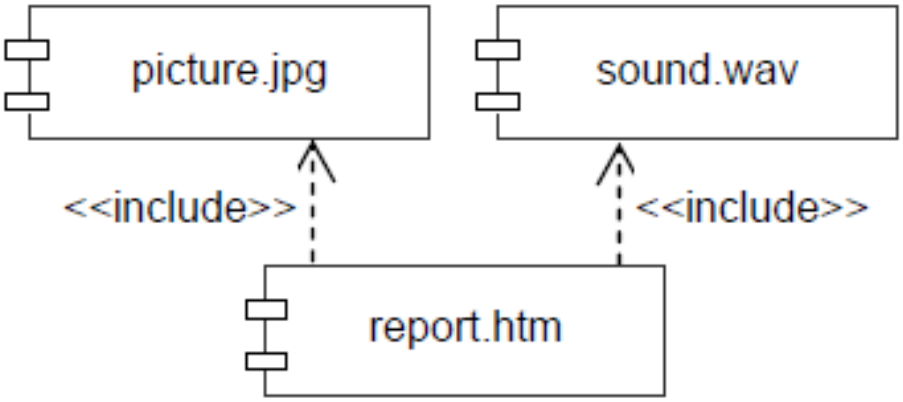


Рис. 2. Приклад діаграми компонентів зі стереотипом << include >>, використовуваним для уточнення відносини типу залежність

Діаграма компонентів, зображена на рис.2 читається в такий спосіб.

Є програмна система, що складається з трьох програмних компонентів, представлених файлами: report.htm (звіт, у вигляді HTML - тексту), picture.jpg (графічне зображення у форматі .jpg) і sound.wav (звуковий супровід у форматі .wav).

Файли picture.jpg і sound.wav є незалежними, а файл report.htm - залежить від файлів picture.jpg і sound.wav. Характер залежності між зазначеними файлами уточнюється стереотипом << include >> (включає). При зображенні діаграми компонентів на рис.2 використовувався стандартний стереотип << include >>, проте розробник проекту може вводити власні стереотипи, що відображають специфіку проекту. За допомогою стереотипів можуть уточнюватися не тільки відносини між компонентами, а й самі компоненти.

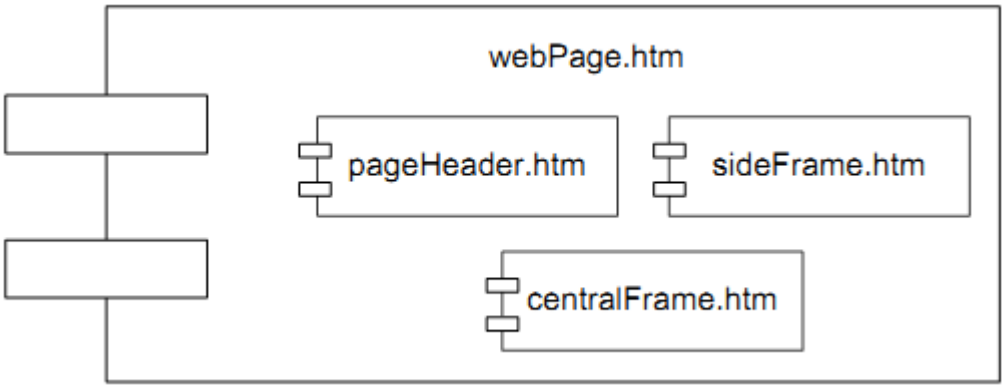


Рис. 3. Вкладення графічних символів програмних компонентів

Діаграма компонентів, зображена на рис.3, читається в такий спосіб.

Програмний компонент webPage.htm (web-сторінка) складається з наступних програмних компонентів: pageHeader.htm (заголовок сторінки), sideFrame.htm (бічний фрейм сторінки) і centralFrame.htm (центральний фрейм сторінки).

Компонент є фізичною реалізацією різних логічних структурних елементів системи. Часто компонент є фізичною реалізацією одного або декількох класів. UML пропонує кілька спосіб відображення зв'язку між компонентом і класами, які він реалізує.

На рис.4 показано, яким чином на діаграмі компонентів можна зобразити класи, використовувані для реалізації компонента.

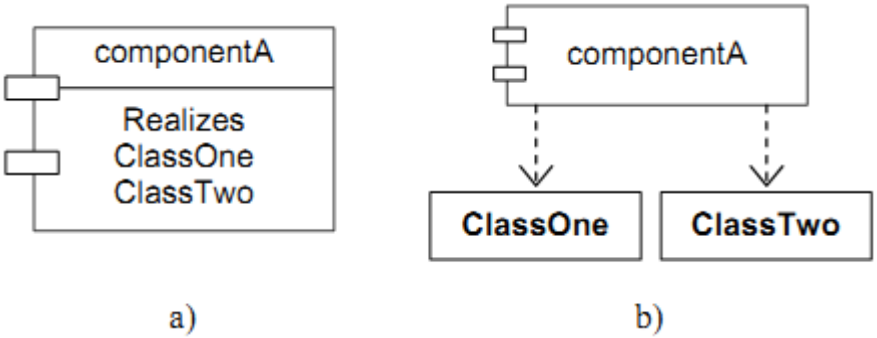


Рис. 4 - Зображення компонента із зазначенням класів, які він реалізує.
а) за допомогою перерахування класів всередині графічного символу компонента
в) за допомогою відносини типу залежність

У лівій частині рис. 4 графічний символ компонента розділений на два відділення горизонтальною лінією. У верхньому відділенні записується ім'я компонента, а в нижньому відділенні, після слова Realizes, перераховуються імена класів, які необхідно використовувати при реалізації компонента. У правій частині рис. 4 кожен клас зображується у вигляді окремого графічного символу (використовується скорочена форма графічного символу класу), а відношення типу залежність, моделює залежність компонента (componentA) від класів (ClassOne і ClassTwo).

Приклад діаграми компонентів

При проектуванні початковій версії діаграми компонентів можна користуватися наступними рекомендаціями. Спочатку формується безліч компонентів, необхідних і достатніх для реалізації функцій програмної системи. Потім, якщо безліч вихідних компонентів велике, то формуються більші компоненти, в які вкладаються вихідні компоненти. На наступному етапі моделюються класи і інтерфейси, що використовуються при реалізації компонентів. На заключному етапі, за допомогою відносини типу залежність моделюються відносини між компонентами системи. У процесі послідовних уточнень первісної версії діаграми можуть редагуватися будь-які елементи первісної версії.

На рис. 5 наведено приклад діаграми компонентів, яка, на фізичному рівні, моделює структуру програмної системи, призначеної для автоматичного тестування знань студентів. Передбачається, що система реалізується в рамках однієї з платформ мови програмування Java. Діаграма компонентів на рис. 5 може бути прочитана наступним чином. Система складається з двох великих компонентів у вигляді Java-пакетів, з розширенням .jar (Java Achives): examRegistration.jar (реєстрація перед іспитом) і subjectExam.jar (випробування з предмета).

Компонент-пакет examRegistration.jar, в свою чергу, складається з трьох компонентів-класів: registrationInterface.class (інтерфейс реєстрації), registrationControl.class (управління реєстрацією) і registrationHelp.class (допомога в реєстрації).

Компонент registrationInterface.class містить два вкладених компонента-класу: GUIPrimitives.class (примітиви графічного призначеного для користувача інтерфейсу) і GUIRegistration.class (графічний користувальницький інтерфейс реєстрації). Ясно, що компонент-клас GUIRegistration.class залежить від компонента-класу GUIPrimi-tives.class. Компонент GUIRegistration.class реалізує інтерфейс Registration. Компонент-пакет subjectExam.jar складається з трьох компонентів-класів: scriptGenerator.class (генератор сценарію тестування), dialogueAgent.class (діалоговий агент) і evaluationAgent.class (агент, що оцінює знання).

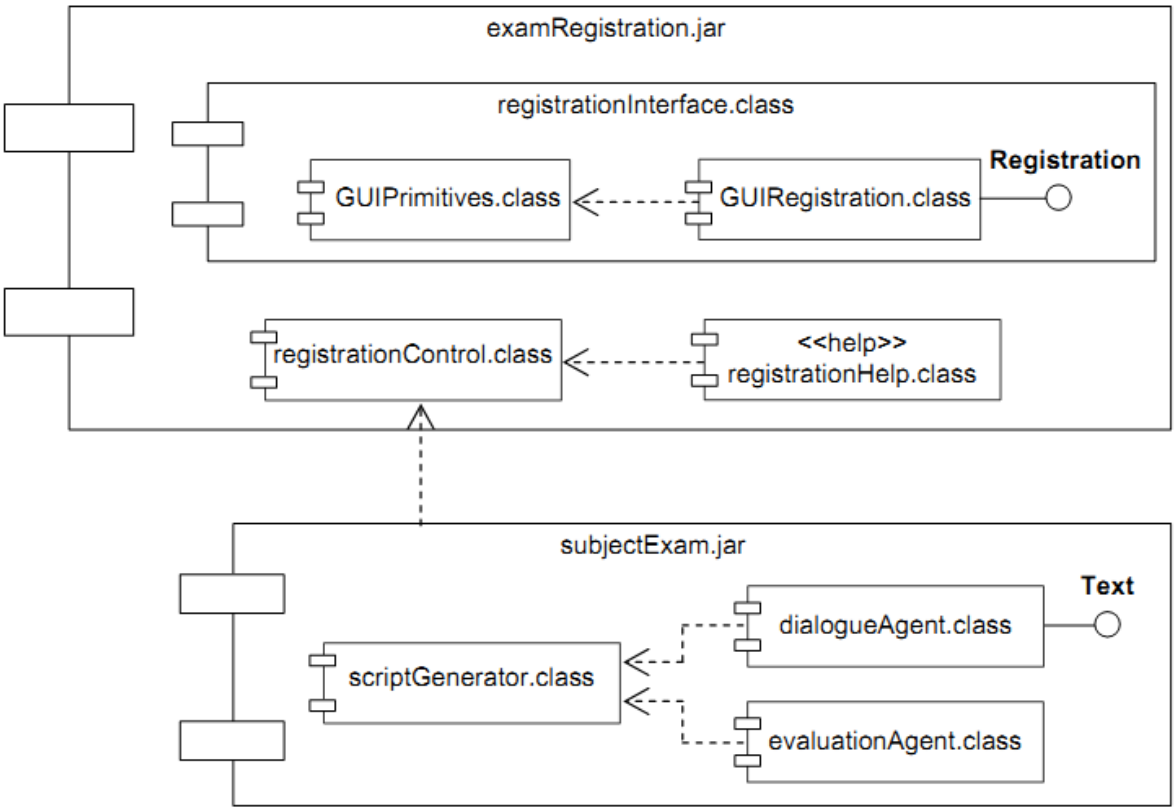


Рис. 5 Діаграма компонентів, що моделює структуру програмної системи, для автоматичного тестування знань студентів

Компоненти-класи dialogueAgent.class і evaluationAgent.class є залежні і залежать від компонента scriptGenerator.class. Компонент dialogueAgent.class реалізує інтерфейс Text. З діаграми, на рис. 5 також видно, що компонент subjectExam.jar залежить від компонента registrationControl.class.

Порядок виконання роботи

1. Вивчити пропонований теоретичний матеріал
2. Розробіть модель архітектури системи. Уявіть цю модель у вигляді діаграми компонентів.
3. Побудувати звіт. Зробити висновки.

Питання для самоконтролю

1. Характеристика діаграми компонентів
2. Графічний символ діаграми компонентів

Лабораторная работа №8

Тема: Моделювання засобів, за допомогою яких специфицируется апаратна реалізація системи.

Мета роботи: Ознайомлення з елементами моделювання архітектури системи, реалізовані на мові UML за допомогою діаграми розгортання**.**

Теоретичний матеріал

Діаграма розгортання використовується, головним чином, для моделювання структури апаратної середовища, призначеної для впровадження проекту. Структура апаратної середовища, на діаграмі розгортання, моделюється за допомогою безлічі апаратних вузлів і ліній комунікацій між ними.

Діаграма розгортання

Діаграма розгортання являє собою сукупність графічних символів фізичних апаратних вузлів (nodes), на яких передбачається експлуатувати проектовану програмну систему, і відносин між вузлами. Апаратними вузлами на діаграмі розгортання моделюються будь апаратні блоки комп'ютера або комп'ютерной мережі.

Наприклад, керуючі процесори; комп'ютери, що виконують функції робочої станції або сервера; стандартні периферійні пристрої; датчики, підключені до комп'ютера і т.п. Фізичні апаратні блоки з'єднані між собою за допомогою різних ліній комунікації. На діаграмі розгортання ці лінії комунікації можуть моделюватися усіма типами відносин, вивченими раніше, однак частіше за інших використовується відношення типу асоціація.

Графічним символом апаратного вузла, на діаграмі розгортання, є зображення прямокутного паралелепіпеда. Кожен вузол має своє унікальне ім'я. Як ім'я вузла може бути зазначено або ім'я типу апаратного блоку, або ім'я конкретного екземпляра апаратного блоку. Правила запису імен апаратних вузлів такі ж, як і правила запису імен класів і об'єктів на діаграмах класів або об'єктів.

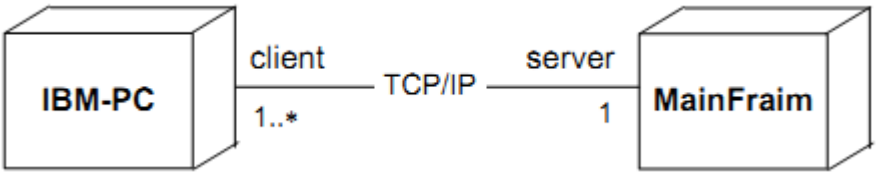


Рис. 1 - Приклад діаграми розгортання. Як ім'я вузла використовується ім'я класу

Діаграма розгортання, наведена на рис. 1, моделює апаратну структуру найпростішої системи типу клієнт-сервер на рівні класів обладнання. Діаграма показує, що система складається з двох класів обладнання: IBM-PC (персональний комп'ютер в стандарті фірми IBM) і MainFraim (універсальна обчислювальна машина). Лінії зв'язку між зазначеними класами обладнання моделюється за допомогою відносини типу асоціація з ім'ям TCP / IP. У цій асоціації клас IBM-PC грає роль клієнта (client), а клас MainFraim - роль сервера (server). Потужності ролей вказують на те, що один екземпляр класу MainFraim пов'язаний з декількома екземплярами класу IBM-PC. Іншими словами до однієї універсальної обчислювальної машини за допомогою системи комунікації типу TCP / IP підключено кілька персональних комп'ютерів в стандарті фірми IBM.

Діаграма розгортання, наведена на рис. 2 розвиває систему типу клієнт-сервер, наведену на рис. 1, і моделює структуру «триярусна» (three-tier) комп'ютерної мережі типу клієнт-сервер на рівні примірників класів обладнання.

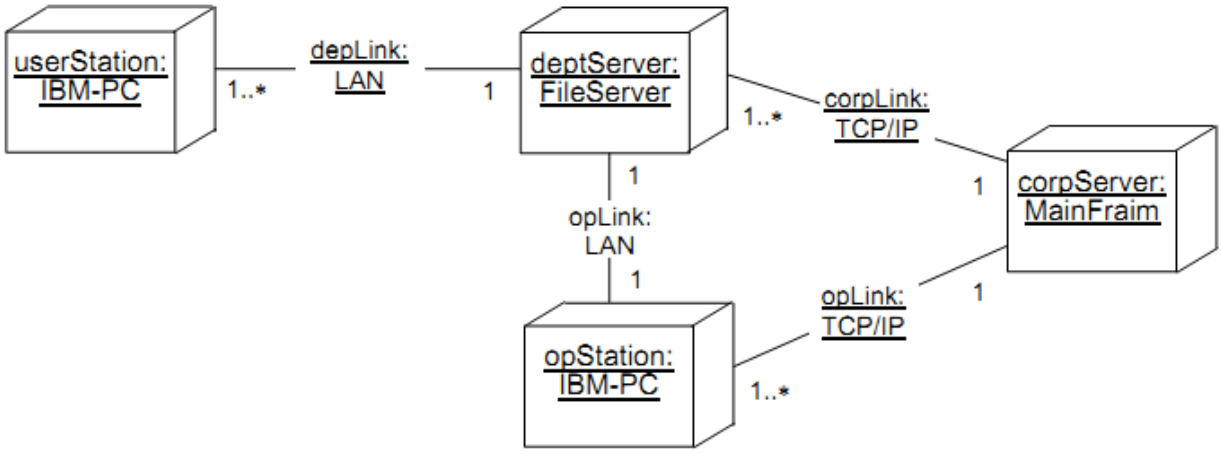


Рис. 2 - Діаграма розгортання, що моделює «триярусна» мережу типу клієнт-сервер. Як ім'я вузла використовується ім'я екземпляра класу обладнання.

Апаратна структура «триярусної» системи типу клієнт-сервер має наступну організацію. Робочі станції, розташовані на столах користувачів, що працюють в деякому відділі, об'єднані з сервером відділу за допомогою локальної комп'ютерної мережі. На рис. 2 робочі станції користувача моделюються графічним символом вузла з ім'ям userStation: IBM-PC, сервери відділів - графічним символом вузла з ім'ям deptServer: FileServer, а лінії комунікації локальної обчислювальної мережі моделюються за допомогою асоціації з ім'ям depLink: LAN. Сервери відділів, в свою чергу, об'єднані з сервером корпорації за допомогою корпоративної комп'ютерної мережі. На рис. 2 сервер корпорації моделюється графічним символом вузла з ім'ям corpServer: MainFrame, а його зв'язок з серверами відділів - за допомогою відносини типу асоціація з ім'ям corpLink: TCP / IP.

Для адміністрування системою в структуру «триярусної» системи включені також робочі станції оператора, з'єднані як з серверами відділів, так і з сервером корпорації. На рис 2 робочі станції оператора моделюються графічним символом вузла з ім'ям opStation: IBM-PC, а зв'язку робочих станцій оператора з серверами - асоціаціями з іменами: opLink: LAN і opLink: TCP / IP. Потужності ролей для відносин типу асоціація на рис. 2 показують, що деяка кількість робочих станцій користувачів з'єднані з одним сервером відділу, деяка кількість серверів відділів з'єднані з одним сервером корпорації, з одним сервером відділу з'єднана одна робоча станція оператора і деяку кількість робочих станцій оператора пов'язані з сервером корпорації.

Діаграми розгортання, що включають програмні компоненти

Діаграми компонентів і розгортання, кожна окремо, дозволяють моделювати структуру системи, або з точки зору її програмних компонентів, або з точки зору апаратних блоків. Коли ми говоримо про архітектуру системи ми, як правило, маємо на увазі її інтегровану апаратно-програмну структуру. Тому, часто, з метою моделювання саме архітектури системи, діаграма розгортання поєднується з діаграмою компонентів. Таке поєднання дозволяє показати відповідність між окремими апаратними та програмними блоками, які виконуються на цій апаратурою, а також відносини між ними. Наприклад, зв'язок між контролером периферійного пристрою комп'ютера і класом драйверів, які, на програмному рівні реалізують частину функцій контролера. На рис. 3 наведено приклад поєднання графічних символів програмних компонентів і апаратних вузлів в одній діаграмі.

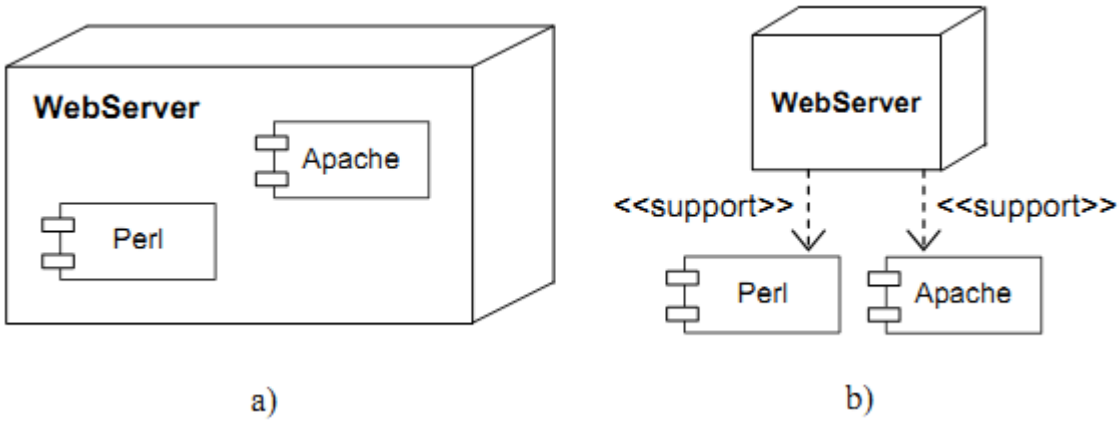


Рис. 3 - Приклад поєднання графічних символів програмних компонентів і апаратних вузлів
a) програмні компоненти вкладені в символ апаратного вузла;
b) програмні компоненти та апаратний вузол пов'язані ставленням типу залежність

На рис. 4 наведено ще один приклад моделювання архітектури системи шляхом поєднання діаграми компонентів і діаграми розгортання. Мал. 4 моделює архітектуру гіпотетичної системи управління маніпулятором промислового робота. Тому, всі типи обладнання, наведені на рис.4 - вигадані.

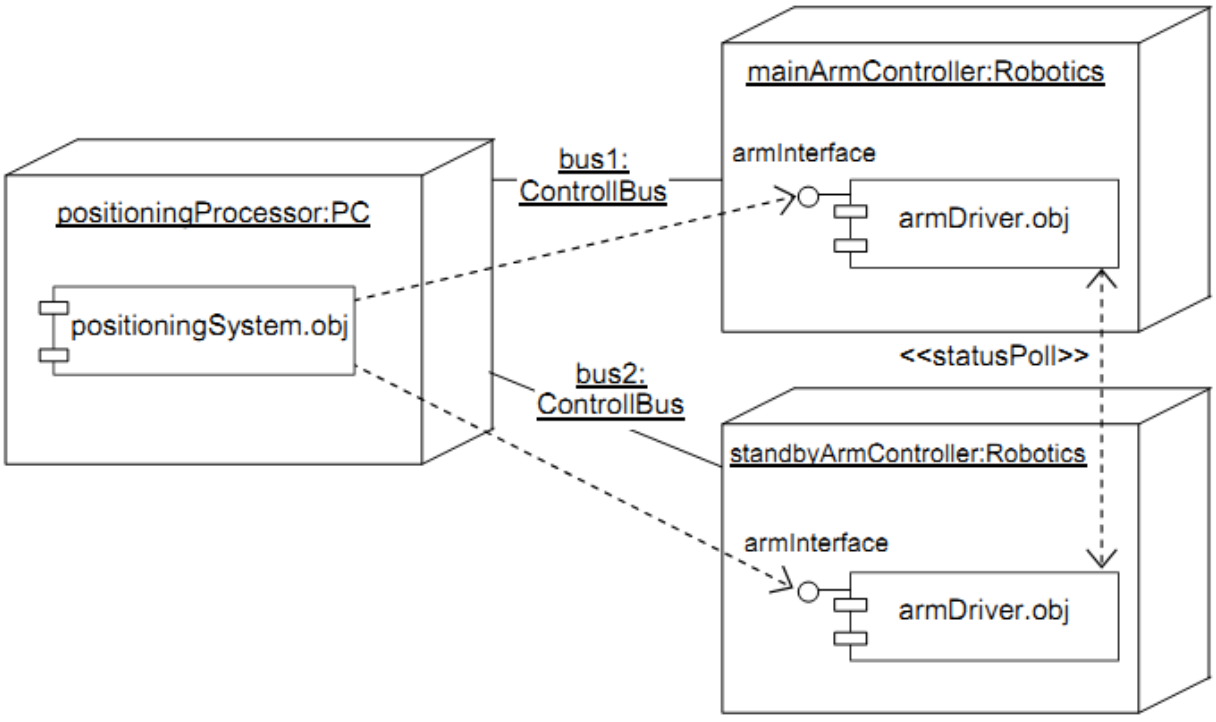


Рис.4 - Діаграма, що моделює архітектуру гіпотетичної системи управління маніпулятором промислового робота

Діаграма, наведена на рис. 4, містить досить багато інформації про архітектуру системи управління маніпулятором промислового робота, і може бути прочитана наступним чином. З точки зору структури апаратних блоків, система включає процесор позиціонування маніпулятора, який моделюється вузлом з ім'ям positioningProcessor: PC. З способу запису імені цього вузла видно, що він належить до типу PC. Система включає, також, два однотипних контролера, призначених для управління маніпулятором. Обидва контролера належать до одного типу обладнання: Robotics. Контролер з ім'ям mainArmController:

Robotics є основним, а контролер з ім'ям standbyArmController: Robotics - резервним. Основний і резервний контролери маніпулятора підключені до процесора позиціонування за допомогою двох однотипних ліній зв'язку з іменами: bus1: ControllBus і bus2: ControllBus. На діаграмі, використовується відношення типу асоціація.

З точки зору структури програмного забезпечення, система включає програмний компонент, керуючий позиціонуванням маніпулятора як файл з ім'ям positioningSystem.obj. Видно, що цей програмний компонент розміщений на процесорі позиціонування маніпулятора positioningProcessor: PC. В структуру системи програмного забезпечення входять також два однакових драйвера контролерів маніпулятора, які моделюються за допомогою програмних компонентів з іменами: armDriver.obj (драйвер маніпулятора). Кожен із зазначених компонентів реалізує інтерфейс з ім'ям armInterface (інтерфейс маніпулятора). Ці інтерфейси використовує компонент positioningSystem.obj для передачі керуючих команд драйверам.

Драйвери розміщені на вузлах mainArmController: Robotics і standbyArmController: Robotics. На діаграмі також показано, що обидва драйвера взаємодіють один з одним. Цей факт моделюється двонаправленим ставленням типу залежність. Характер взаємодії (відносини) уточнюється за допомогою стереотипу << statusPoll >> (опитування стану). Іншими словами, передбачається, що кожен з контролерів може періодично опитувати стан протилежної контролера.

Порядок виконання роботи

1. Вивчити пропонований теоретичний матеріал
2. Розробіть модель архітектури системи. Уявіть цю модель у вигляді діаграми розгортання.
3. Уявіть структуру вашої системи у вигляді діаграми розгортання, поєднаної з діаграмою компонентів.
3. Побудувати звіт. Зробити висновки.

Питання для самоконтролю

1. Характеристика діаграми розгортання
2. Графічний символ діаграми розгортання

Рекомендована література

1. Соммервиль Иан. Инженерия программного обеспечения, 6-е издание.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 624 с.
2. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. – С-П.: Издательство «Питер», 2003. – 432 с.
3. Роберт Т. Фатрелл. Управление программными проектами. – М.: Издательский дом “Вильямс”, 2013. – 1125 с.
4. Лешек Мацяшек. Анализ требований и проектирование систем. – М.: Издательский дом “Вильямс”, 2003. – 651 с.
5. Орлов С.А. Технологии разработки программного обеспечения: Разработка сложных программных систем Изд. 3-е, 2004.
6. Липаев, В.В. Программная инженерия. Методологические основы : Учеб. / В. В. Липаев ; Гос. ун-т — Высшая школа экономики. — М. : ТЕИС, 2016. — 608 с.
7. Чмырь И.А. “Объектно-ориентированное моделирование. Учебное пособие. Одесса, 2016.