Національний авіаційний університет
Факультет економіки та бізнес-адміністрування
Кафедра економічної кібернетики

Ю.П. Бойко

Методичні рекомендації до виконання практичних робіт з дисципліни «Моделювання в цифровій економіці»

Київ
2019

Зміст

Вступ

Головною метою дисципліни «Моделювання в цифровій економіці» є освоєння знань з питань моделювання та проектування цифрових систем.

Об'єктами вивчення в даній дисципліні є: технології проектування, моделі і методи підтримки життєвого циклу програмних систем; засоби і методи створення та реалізації проектів по створенню цифрових систем.

Особливістю методичних вказівок є те, що його можна застосовувати як основу при вивченні теоретичного матеріалу і отримання навичок моделювання та проектування цифрових систем з використанням CASE-моделей на практичних заняттях або в рамках самостійної роботи студента.

Методичні вказівки призначені для студентів, які вивчають курс «Моделювання в цифровій економіці», і фахівців, які працюють в різних областях розробки цифрових систем.

Практична робота №1
(4 год.)

Тема: Введення в CASE-засіб Rational Rose. Діаграми варіантів використання

Мета роботи – вивчення основних етапів проектування та основних елементів нотації, застосовуваних в CASE-засобі Rational Rose, оволодіння інтерфейсом і створення нового проекту, вивчення діаграм варіантів використання та їх застосування в процесі постановки вимог до проектованої системи.

1. Теоретичні відомості

1.1.Етапи проведення моделювання в Rational Rose

Моделювання системи проводиться як спуск по рівнях: від концептуальної моделі до логічної, а потім до фізичної моделі програмної системи (ПС).

Концептуальна модель виражається у вигляді діаграм варіантів використання (use case diagram). Цей тип діаграм служить для проведення ітераційного циклу загальної постановки завдання разом із замовником. Часто можна почути таке: «Замовник і раніше не знав, і тепер не знає, і в майбутньому не буде точно знати, що йому потрібно». Діаграми варіантів використання саме і є основою для досягнення взаєморозуміння між програмістами, що розробляють проект системи, і замовниками проекту. Усередині кожного варіанта використання можуть бути визначені:

- вкладена діаграма варіантів використання;
- діаграма взаємодії об'єктів (collaboration diagram);
- діаграма послідовності взаємодій (sequence diagram);
- діаграма класів (class diagram);
- діаграма переходу станів (state diagram).

Логічна модель дозволяє визначити два різних погляди на систему: статичний і динамічний. Ці погляди перетворюються у різні підходи до моделювання системи. Статичний підхід виражається діаграмами класів (class diagram). Саме діаграми класів є основою для генерації програмного коду цільовою мовою програмування. Можливе дуже гнучке настроювання генерації коду з врахуванням конкретних угод (наприклад, за префіксами імен ідентифікаторів), прийнятих у команді розробників проекту.

Динамічний підхід описується двома типами діаграм:

- діаграмами взаємодії об'єктів;
- діаграмами послідовності взаємодій.

У засобі Rational Rose ці діаграми не впливають на код, що генерується, однак фірми-партнери Rational Software застосовують ці діаграми у своїх додатках. Так, діаграми послідовності взаємодій використовуються в пакеті SQA Suite для автоматизованого тестування компонентів, розроблених в

Rational Rose. Класи, введені на цих діаграмах, потрапляють у список класів моделі й можуть використовуватися при конструюванні діаграм класів [4].

Динаміка конкретного класу може бути виражена за допомогою діаграм переходу станів, що визначають модель скінченного автомата, який описує поведінку класу. Кожен стан задається своєю вершиною; визначені вхідний і вихідний стани, а також умови переходу із стану в стан.

Фізична модель задається компонентною діаграмою (component diagram), що описує розподіл реалізації класів за модулями, і діаграмою розгортання (deployment diagram).

Після побудови першого та наступних рівнів статичної моделі з використанням діаграм класів можна провести генерацію коду на мові програмування. На рівні коду можна ввести нові уточнюючі класи, змінити атрибути й методи класів моделі і синхронізувати код і модель, виконавши зворотне проектування, тобто за модифікованим кодом Rational Rose дозволяє побудувати нову логічну модель взаємозв'язків класів між собою. Повторення такої процедури кілька разів називається ітераційним моделюванням (round-trip modeling). Воно становить основу м'якого й поступового уточнення постановки завдання й узгодження вимог замовника з наявними ресурсами (обчислювальними, фінансовими тощо). На рис.1.1 зображено зовнішній вигляд головного вікна Rational Rose.

Створення нового проекту в Rational Rose виконується в меню File/New. При цьому створюється кілька порожніх діаграм верхнього рівня: діаграма варіантів використання, діаграма класів та ін. Кожну діаграму можна вибрати для редагування, при цьому на інструментальній панелі відображаються елементи, доступні для певного виду діаграм. Вибір типу поточної діаграми виконується в меню Browse. Описання елементів керування наведено в табл.1.1.

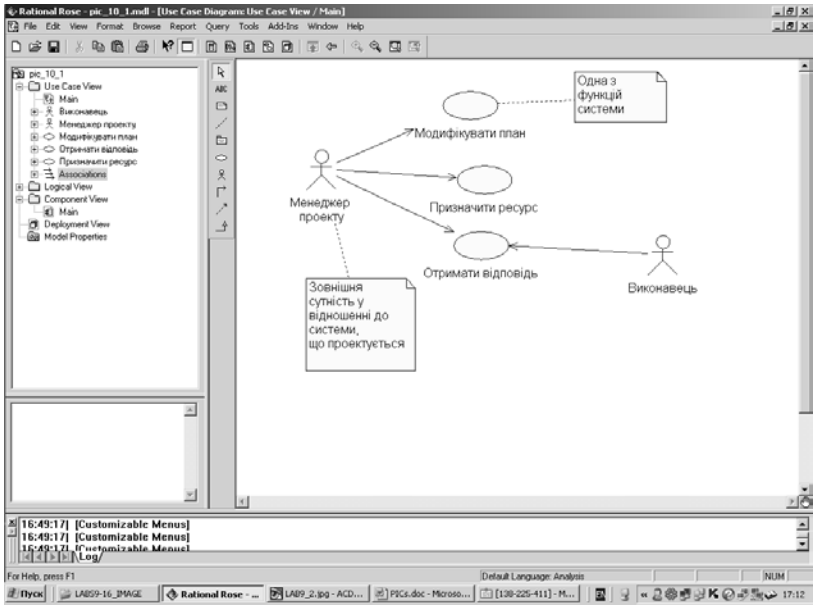

Рис. 1.1. Головне вікно Rational Rose

Таблиця 1.1

Описання елементів керування основної панелі Rational Rose

Елемент керування	Описання	Відповідний пункт меню
	Створити нову модель	File->New
	Відкрити модель	File->Open
	Зберегти модель	File->Save
	Надрукувати модель	File->Print
	Перемикання між типами діаграм	Browse-> Diagram
	Одержання довідки	Help
	Відкриття вікна для введення коментарів	View-> Documentation
	Навігація по діаграмах	Browse-> Previous
	Масштабування	View->Zoom

1.2.Кількісна оцінка діаграм

Методика кількісної оцінки й порівняння діаграм UML будується на присвоєнні елементам діаграм оцінок, що залежать від їхньої інформаційної цінності, а також від внесеної ними в діаграму додаткової складності. Цінність окремих елементів змінюється залежно від типу діаграми, на якій вони перебувають.

Словник мови UML містить два види будівельних блоків: сутності й відносини. Сутності - це абстракції, що є основними елементами моделі. Відносини зв'язують різні сутності. Кількісну оцінку діаграми можна провести за формулою:

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

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

Якщо діаграма містить велику кількість зв'язків одного типу (наприклад, модель бази даних (БД)), то їх можна не враховувати, і формула розрахунку набуває вигляду:

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

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

$$S_{cls} = \frac{\sqrt{Op} + \sqrt{Atr}}{0{,}3(Op + Atr)},$$

де S_{cls} - оцінка операцій й атрибутів для класу; *Op* - кількість операцій у класі, *Atr* - кількість атрибутів у класі.

При цьому враховуються тільки атрибути й операції, відображені на діаграмі. Далі в табл.1.2 і 1.3 наводяться оцінки для різних типів елементів і зв'язків.

Таблиця 1.2

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 (note)	2

Таблиця 1.3

Основні типи зв'язків мови UML

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

Інші типи зв'язків розглядаються як асоціації. Недоліком діаграми є як занадто низька оцінка (недостатня інформативність), так і занадто висока (діаграма занадто складна). Діапазони оптимальних оцінок для діаграм наведені у табл.1.4.

Таблиця 1.4

Діапазони оцінок для діаграм UML

Тип діаграми	Діапазон оцінок
Класів (class) - з атрибутами й операціями	5-5,5
Класів (class) - без атрибутів й операцій	3-3,5
Компонентів (component)	3,5-4
Варіантів використання (use case)	2,5-3
Розгортання (deployment)	2-2,5

Закінчення таблиці 1.4

Тип діаграми	Діапазон оцінок
Послідовності (sequence)	3-3,5
Кооперативна (cooperative)	3,5-4
Пакетів (package)	3,5-4
Станів (state)	2,5-3

Наведемо приклад оцінки простої діаграми класів, яка показана на рис.1.2.

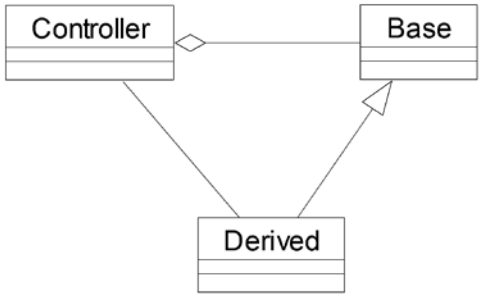

Рис. 1.2. Діаграма класів

Діаграма містить три класи без операцій та атрибутів. Отже, $T_{Obj}=1$, ΣS_{obj} = 15 і Obj = 3. Як зв'язки використовуються асоціація, агрегування й узагальнення; отже, $\Sigma S_{Lnk} = 6$ й $T_{Lnk} = 3$.

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

Тобто числова оцінка для даної діаграми дорівнює 3,5.

На рис.1.3 і 1.4 показані діаграми класів моделі системи «Служба зайнятості в рамках вищого навчального закладу (ВНЗ)» Ці діаграми реалізують один і той самий фрагмент системи, але перша з них більш повно

реалізує принципи об’єктно-орієнтованого підходу. Знайдемо числову оцінку кожної з діаграм.

Діаграма 1

Проведемо розрахунок оцінки атрибутів й операцій для класів «Роботодавець», «БД студентів» і «Студент».

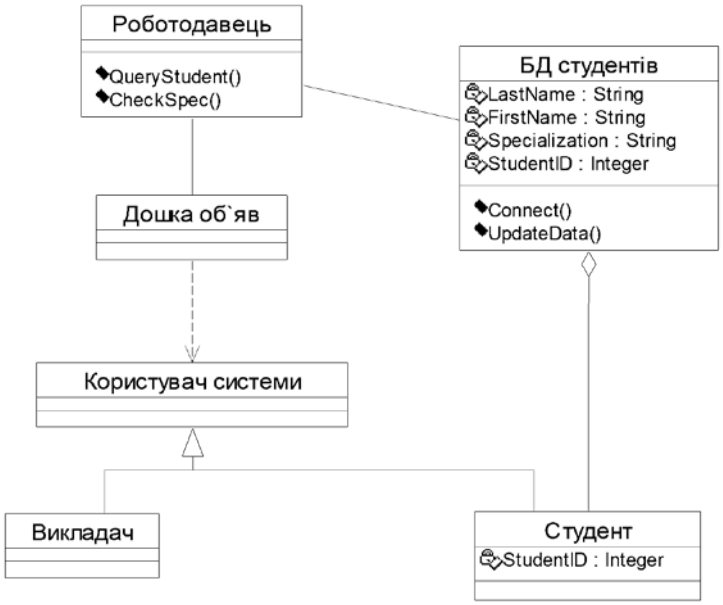

Рис. 1.3. Діаграма класів 1

«Роботодавець»:

$$S_{cls} = \frac{\sqrt{Op} + \sqrt{Art}}{0.3*(Op + Art)} = \frac{\sqrt{2} + \sqrt{0}}{0,3*(2+0)} = 2,36.$$

Аналогічно для класу «БД студентів» одержуємо 2,53; для класу «Студент» - 3,33. Розрахуємо повне значення для діаграми:

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{38,33+9}{1+6+\sqrt{5}} = 5,11.$$

Діаграма 2

Проведемо розрахунок оцінки атрибутів й операцій для класів «Деканат», «Група» й «Користувач системи». Для класу «Деканат» одержуємо 2,36; для класу «Група» - 3,33; для класу «Користувач системи» - 1,11. Розрахуємо повне значення для діаграми:

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{31,8+8}{1+5+2} = 4,85.$$

У результаті оцінка для діаграми 1 потрапляє в середину оптимального діапазону для діаграм класів, а оцінка для діаграми 2 виявляється нижче оптимального діапазону.

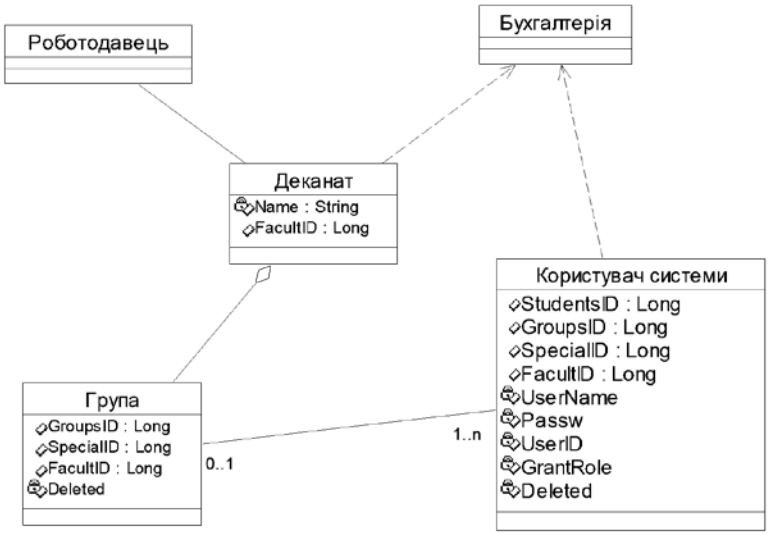

Рис. 1.4. Діаграма класів 2

Такий результат можна пояснити такими причинами:

1. Діаграма 2 містить зайво деталізований клас «Користувач системи», тоді як у діаграмі 1 він спрощений за допомогою побудови ієрархії класів.

2. Клас «Деканат» на діаграмі 2 бере на себе занадто багато функцій, наслідком чого є надлишок зв'язків.

3. Клас «Бухгалтерія» на діаграмі 2 не належить прямо до фрагмента, змодельованого на діаграмі, тобто ускладнює модель, не вносячи при цьому корисної інформації.

2. Діаграми варіантів використання (use case diagrams)

Одна з моделей формалізації процесу постановки цілей і завдань проекту була запропонована фірмою Rational і ввійшла в стандарт мови UML. Це діаграми варіантів використання (use case), які іноді називають діаграмами прецедентів. Варіант використання являє собою типову взаємодію користувача й проектованої системи і характеризується такими властивостями:

- варіант охоплює деяку очевидну для користувачів функцію;
- варіант може бути як невеликим, так і досить великим;
- варіант вирішує деяке дискретне завдання користувача.

У найпростішому випадку варіант використання створюється в процесі обговорення з користувачами тих речей, які вони хотіли б одержати від системи [5]. При цьому кожній окремій функції привласнюється ім'я й записується її короткий текстовий опис.

Це все, що необхідно у фазі аналізу. Знання деяких деталей може знадобитися, якщо передбачається, що даний варіант використання містить важливі архітектурні відгалуження. Більшість варіантів використання може бути деталізована під час конкретної ітерації в процесі проектування.

На рис.1.5 показано варіант використання, що описує одну з функцій системи керування проектами - зворотний зв'язок між менеджером проекту й виконавцем. Основними елементами діаграми варіантів використання є діючі особи, варіанти використання й відносини між ними. Діюча особа - це роль, що користувач грає відносно системи.

Рис. 1.5. Діаграма варіантів використання системи керування проектами

На діаграмі наявні дві діючі особи: «Менеджер проекту» й «Виконавець». Менеджерів і виконавців може бути багато, але з погляду системи вони виконують однакову роль. Говорячи про діючі особи, важливо бачити в них ролі, а не конкретних людей або найменування робіт. Діючі особи зовсім не зобов'язані бути людьми, незважаючи на те, що на діаграмах варіантів використання вони зображуються у вигляді стилізованих людських фігурок. Діюча особа може також бути зовнішньою системою, якій необхідна деяка інформація від нашої системи.

На діаграмі наявні також три варіанти використання: «Модифікувати план», «Призначити ресурс» і «Одержати звіт». Всі варіанти використання так чи інакше пов'язані із зовнішніми вимогами до функціональності системи. Варіанти використання завжди варто аналізувати разом із діючими особами системи, визначаючи при цьому реальні завдання користувачів і розглядаючи альтернативні способи вирішення цих завдань.

Діючі особи можуть грати різні ролі щодо варіанта використання. Вони можуть застосовувати його результати або самі безпосередньо в ньому брати участь (табл.1.5).

Гарним джерелом для ідентифікації варіантів використання служать зовнішні події. Необхідно перелічити всі події, що відбуваються у зовнішньому світі, на які система повинна реагувати. Конкретна подія може викликати реакцію системи, що не потребує втручання користувачів, або, навпаки, викликати суто користувальницьку реакцію. Ідентифікація подій, на які необхідно реагувати, допоможе ідентифікувати варіанти використання.

Таблиця 1.5

Кнопки панелі інструментів діаграм варіантів використання

Кнопка	Описання	Назва
	Вибір елемента моделі	Selection Tool
ABC	Введення тексту	Text Box
	Коментар	Note
	Зв'язок елемента моделі з коментарем	Anchor Note to Item
	Додавання пакета	Package

Закінчення таблиці 1.5

Кнопка	Описання	Назва
	Додавання варіанта використання	Use Case
	Додавання діючої особи	Actor
	Односпрямований зв'язок	Unidirectional Association
	Залежність	Dependency
	Успадкування (спадкування)	Generalization

Дві діаграми варіантів використання, що описують одну з функцій системи «Служба зайнятості в рамках ВНЗ», зображено на рис.1.6 і 1.7. Відмінність цих діаграм у тім, що перша з них більш докладно описує процес взаємодії користувача й системи. Знайдемо кількісну оцінку для кожної з діаграм.

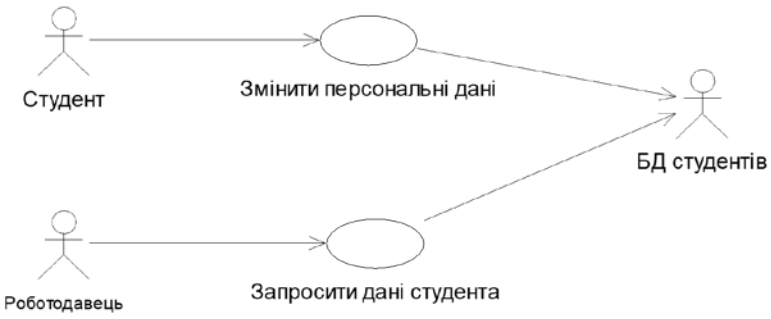

Рис. 1.6. Діаграма варіантів використання 1

Діаграма 1

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{17 + 4}{1 + 5 + \sqrt{3}} = 2{,}72.$$

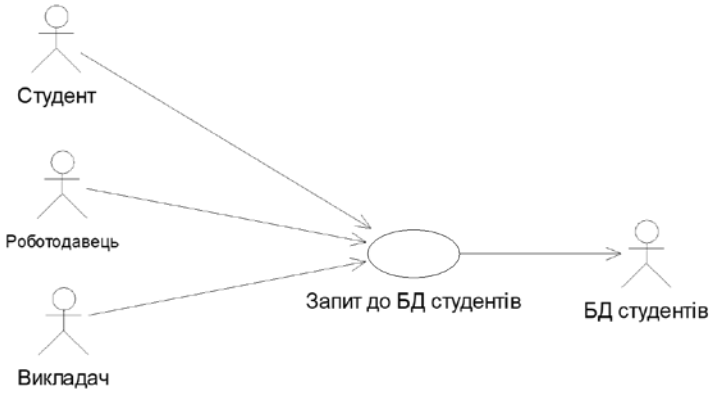

Рис. 1.7. Діаграма варіантів використання 2

Діаграма 2

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{21 + 4}{1 + 5 + \sqrt{3}} = 3{,}23.$$

Оцінка для діаграми 1 потрапляє в оптимальний для діаграм варіантів використання діапазон, а оцінка для діаграми 2 перевищує верхню границю діапазону. Отриманий результат можна пояснити тим, що діаграма 2 описує взаємодію системи з користувачем на занадто високому рівні. Фактично діаграма 1 є деталізованим до рівня простих операцій варіантом діаграми 2.

3. Завдання

За допомогою CASE-засобу Rational Rose створити проект інформаційної системи, для чого виконати перший крок – побудувати діаграми варіантів використання для заданого опису предметного середовища.

Постановка задачі

Перед керівником інформаційної служби Навчального центру ставиться завдання розробки автоматизованої системи реєстрації студентів на додаткові курси. Система повинна дозволяти студентам реєструватися на курси і переглядати свої табелі успішності з персональних комп'ютерів, підключених до локальної мережі академії.

Професори повинні мати доступ до системи, щоб вказати курси, які вони будуть читати, і проставити оцінки за курси.

В даний час в Навчальному центрі функціонує база даних, що містить всю інформацію про курси (каталог курсів). Реєстрація на курси відбувається

наступним чином: спочатку кожного семестру студенти можуть вимагати від реєстратора каталог курсів, що містить список курсів, пропонованих в даному семестрі. Інформація по кожному курсу повинна включати ім'я професора, найменування кафедри і вимоги до попереднього рівня підготовки (прослуханих курсів).

Студент може вибрати 4 курси в майбутньому семестрі. На додаток до цього кожен студент може вказати 2 альтернативних курсу на той випадок, якщо який-небудь з обраних ним курсів виявиться вже заповненим або був скасований. На кожен курс може записатися не більше 10 і не менше 3 студентів (якщо менше 3, то курс буде скасовано). У кожному семестрі існує період часу, коли студенти можуть змінити свої плани (додати або відмовитися від вибраних курсів). Після того, як процес реєстрації деякого студента завершено, реєстратор надсилає інформацію в розрахункову систему, щоб студент міг внести плату за семестр.

Якщо курс виявиться заповненим в процесі реєстрації, студент повинен бути повідомлений про це до остаточного формування його особистого навчального плану. В кінці семестру студенти можуть переглянути свої табелі успішності.

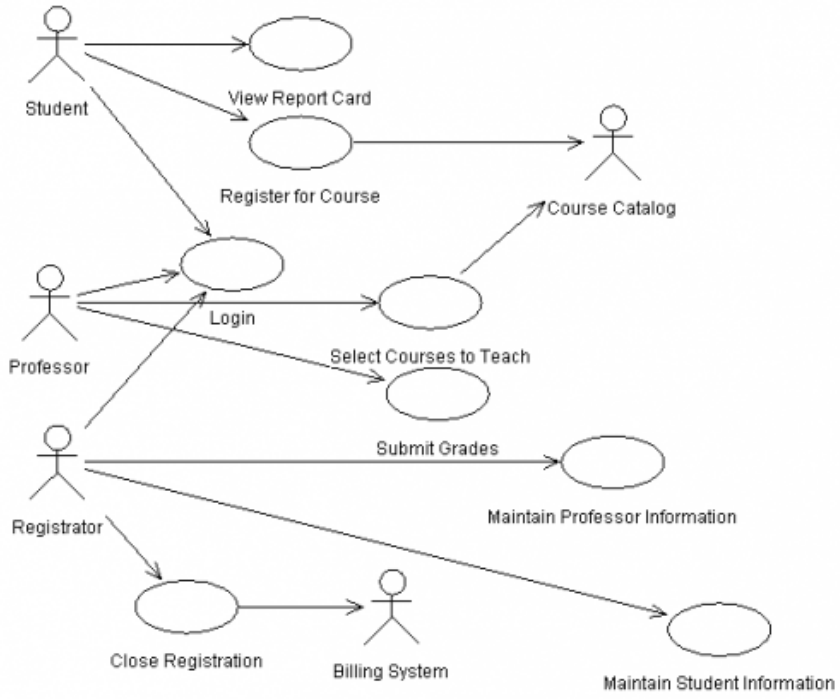

Рис. 1.8. Діаграма варіантів використання для системи реєстрації на курси

Контрольні питання

1. Які три типи моделей використовуються під час проектування? Основний вид діаграм у концептуальній моделі.

2. Яке призначення логічної моделі? Назвіть основний вид діаграм у логічній моделі. Яка роль діаграм взаємодії об'єктів у логічній моделі?

3. Назвіть основний вид діаграм у фізичній моделі.
4. У чому зміст процедури ітераційного моделювання?

Практична робота №2

(4 год.)

Тема: Побудова діаграм класів

Мета роботи – вивчення діаграм класів та їх застосування в процесі проектування.

1. Теоретичні відомості

Побудова діаграм класів (class diagrams) є центральною ланкою методології об’єктно-орієнтованого аналізу й проектування.

Діаграма класів відображає класи та їх взаємовідносини, тим самим представляючи логічний аспект проекту. Кожна діаграма класів представляє певний ракурс структури класів. На стадії аналізу діаграми класів використовуються, щоб виділити загальні ролі й обов'язки сутностей, які забезпечують необхідну поведінку системи. На стадії проектування діаграми класів застосовують, щоб передати структуру класів, які формують архітектуру системи.

Кожен клас повинен мати ім'я і, якщо ім'я занадто довге, його можна скоротити або збільшити сам значок на діаграмі. Ім'я кожного класу повинно бути унікальним в поточному проекті.

Діаграма класів визначає типи об'єктів системи й різного роду статичні зв'язки, які існують між ними. Є два основних види статичних зв'язків:

- асоціації (наприклад, менеджер може вести кілька проектів);
- підтипи (працівник є різновидом особистості).

На діаграмах класів зображуються також атрибути класів, операції та обмеження, які накладаються на зв'язки між об'єктами. Типова діаграма класів зображена на рис.2.1. Далі розглянемо фрагменти діаграми.

Асоціації

Асоціації являють собою зв'язки між екземплярами класів (особистість працює в компанії, компанія має ряд офісів).

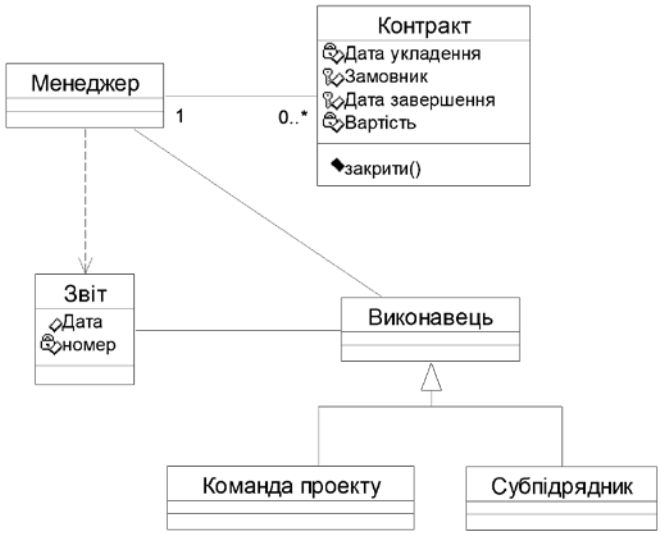

Рис. 2.1. Типова діаграма класів

Будь-яка асоціація володіє двома ролями; кожна роль являє собою напрямок асоціації. Таким чином, асоціація між «Виконавцем» і «Звітом» містить дві ролі: одна від «Виконавця» до «Звіту»; інша – від «Звіту» до «Виконавця». Роль може бути явно пойменована за допомогою позначки (мітки). Якщо позначка відсутня, ролі привласнюється ім'я класу-мети. Таким чином, роль асоціації від «Виконавця» до «Звіту» може бути названа «Звітом».

Роль також має множинність, яка вказує, скільки об'єктів може брати участь у даному зв'язку. На рис.2.1 символ «0..*» над асоціацією між «Менеджером» й «Контрактом» вказує, що з одним «Менеджером» може бути зв'язано багато «Контрактів», а символ «1» показує, що будь-який «Контракт» управляється одним «Менеджером».

У загальному випадку множинність вказує нижню й верхню границі кількості об'єктів, які можуть брати участь у зв'язку. Для цього можуть використовуватися однина, діапазон або дискретна комбінація із чисел і діапазонів.

Для асоціації може бути зазначений напрямок навігації. Якщо навігація зазначена тільки в одному напрямку, то така асоціація називається односпрямованою (рис.2.1, асоціація між «Менеджером» та «Звітом»). У двонапрямленій асоціації навігація зазначена в обох напрямках. У мові UML відсутність стрілок в асоціації трактується наступним чином: напрямок навігації невідомий або асоціація є двонапрямленою.

Атрибути

Атрибути багато в чому подібні до асоціацій. Різниця між ними полягає в тому, що атрибути припускають єдиний напрямок навігації – від типу до атрибута.

На рис.2.1 атрибути зазначені для класів «Контракт» і «Звіт». Залежно від ступеня деталізації діаграми, позначення атрибута може включати ім'я атрибута, тип і значення за замовчуванням. У синтаксисі UML це виглядає так: <ознака видимості> <ім'я> : <тип> = <значення за замовчуванням>, де ознака видимості може набувати одне з чотирьох значень:

• загальний (public) - атрибут доступний для всіх клієнтів класу;

• захищений (protected) - атрибут доступний тільки для підкласів і друзів класу;

• секретний (private) - доступний тільки для друзів класу;

• реалізація (implementation) - атрибут доступний тільки усередині пакета, що обрамляє.

Операції

Операції являють собою процеси, реалізовані класом. Найбільш очевидна відповідність існує між операціями й методами класу. Повний синтаксис UML для операцій виглядає так: <ознака видимості> <ім'я> (<список-параметрів>) : <тип виразу, що повертає значення> = <рядок-властивостей>, де:

• ознака видимості може набувати ті ж значення, що й для атрибутів;

• ім'я являє собою символьний рядок;

• список параметрів містить необов'язкові аргументи, синтаксис яких збігається із синтаксисом атрибутів;

• тип виразу, що повертає значення, є необов'язковою специфікацією й залежить від конкретної мови програмування;

• рядок властивостей показує значення властивостей, які застосовуються до даної операції. Прикладом операції на рис.2.1 є операція закриття класу «Контракт».

Узагальнення

Типовий приклад узагальнення містить «Команду проекту» й «Субпідрядника» (див. рис.2.1). Вони мають деякі розходження, однак у них є також багато спільного. Однакові характеристики можна помістити в узагальнений клас «Виконавець» (супертип), при цьому класи «Команда проекту» й «Субпідрядник» будуть виступати як підтипи.

Зміст узагальнення полягає в тім, що інтерфейс підтипу повинен включати всі елементи інтерфейсу супертипу. Інша сторона узагальнення пов'язана із принципом підстановки. Субпідрядника можна підставити в будь-який код, де потрібен «Виконавець», і при цьому все повинно нормально працювати. Це означає, що, розробивши код, який припускає використання «Виконавця», можна вільно вживати екземпляр будь-якого підтипу «Виконавця». Субпідрядник може реагувати на деякі команди відмінним від іншого «Виконавця» чином (відповідно до принципу поліморфізму), але ця відмінність не повинна турбувати об'єкт, що викликає «Виконавця».

Узагальнення з погляду реалізації пов'язане з поняттям успадкування в мовах програмування. Підклас успадковує всі методи й поля суперкласу і може перевизначити наслідувані методи. Підтип можна також реалізувати, використовуючи механізм делегування.

Обмеження

При побудові діаграм класів основним завданням є відображення різних обмежень. На рис.2.1 показано, що «Контракт» може управлятися тільки одним «Менеджером».

За допомогою конструкцій асоціації, атрибута й узагальнення можна специфікувати найбільш важливі обмеження, але неможливо виразити їх усі.

В UML відсутній чіткий синтаксис опису обмежень, за винятком поміщення їх у фігурні дужки {}. Описання інструментів для рисування діаграм класів наведено в табл.2.1.

Таблиця 2.1

Кнопки панелі інструментів діаграм класів у Rational Rose

Кнопка	Описання	Назва
	Вибір елемента моделі	Selection Tool
	Введення тексту	Text Box
	Коментар	Note
	Зв'язок коментарю з елементом	Anchor Note to Item
	Клас	Class
	Інтерфейс	Interface
	Асоціація	Association
	Прив'язка атрибута	Link Attribute
	Додавання пакета	Package
	Залежність	Dependency
	Успадкування	Generalization
	Реалізація	Realize

2. Приклад

На рис. 2.2 і 2.3 зображено дві діаграми класів, що реалізують однаковий фрагмент системи «Служба зайнятості в рамках ВНЗ»: взаємодія користувача з БД, що містить персональні відомості.

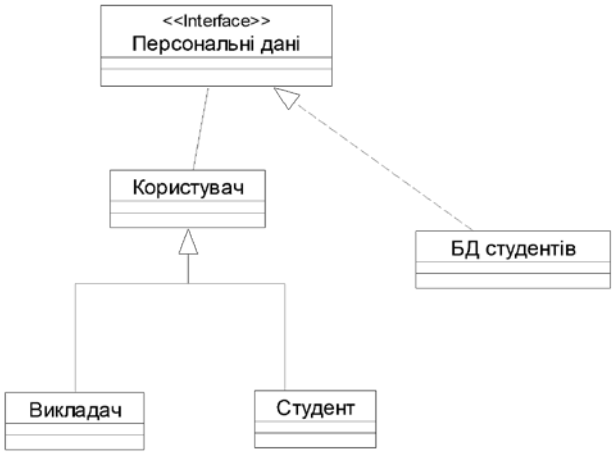

Рис. 2.2. Діаграма класів 1

Проведемо розрахунок оцінки для кожної з діаграм.

Діаграма 1

Атрибути й операції на діаграмі не зазначені, тому відразу розрахуємо повне значення для діаграми:

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{25+6}{1+5+\sqrt{1+3}} = 3{,}875.$$

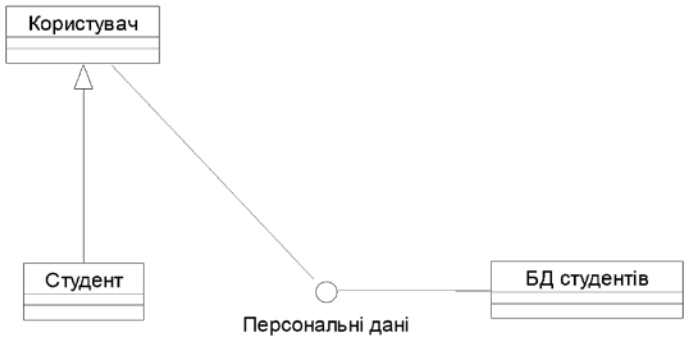

Рис. 2.3. Діаграма класів 2

Діаграма 2

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{19 + 5}{1 + 4 + \sqrt{1 + 3}} = 3{,}43.$$

Значення для діаграми 2 перебуває в допустимих межах, а значення для діаграми 1 перевищує припустиму величину. Такий результат можна пояснити двома причинами.

1. На діаграмі 1 відображений клас «Викладач», який хоча і є нащадком класу «Користувач системи», але не бере участь у взаємодії із БД. Відповідно він ускладнює модель, не вносячи при цьому корисної інформації.

2. На діаграмі 1 інтерфейс «Особисті дані» зображений у розгорнутій формі (як клас-стереотип), а на діаграмі 2 він показаний значком інтерфейсу. Остання форма відображення краща, коли не вказуються операції інтерфейсу.

3. Завдання

Виділити основні класи об'єктів у системі, що проектується і побудувати діаграму класів, яка у загальному вигляді демонструє архітектуру системи. Побудувати діаграми класів, для моделі бізнес-аналізу, яка описує Business Use Case «Зареєструватися на курси». Вказати для класів основні атрибути, операції, вид і напрямок асоціацій.

Контрольні питання

1. Призначення діаграм класів. Для чого використовується діаграма класів на стадії аналізу і на стадії проектування?
2. Назвіть основні компоненти діаграм класів та основні типи статичних зв'язків між класами.
3. Що являє собою асоціація? У чому зміст множинності асоціацій? У чому відмінність атрибутів від асоціацій?
4. Що являє собою операція класу? У чому зміст узагальнення?

Практична робота №3

(2 год.)

Тема: Розробка діаграм взаємодії

Мета роботи – вивчення діаграм взаємодії та їх застосування в процесі проектування.

1. Теоретичні відомості

1.1. Діаграми взаємодії (interaction diagrams)

Діаграми взаємодії є моделями, що описують поводження взаємодіючих груп об'єктів.

Як правило, діаграма взаємодії охоплює поводження тільки одного варіанта використання. На такій діаграмі відображається ряд об'єктів і ті повідомлення, якими вони обмінюються між собою в рамках даного варіанта використання.

Цей підхід буде проілюстрований на прикладі простого варіанта використання, що описує таку поведінку:

• Студент - записується на курси і переглядає свій табель успішності.

• Професор - вибирає курси для викладання і ставить оцінки за курси.

• Розрахункова система - отримує інформацію по оплаті за курси.

• Каталог курсів - база даних, що містить інформацію про курсах.

Існує два види діаграм взаємодії: діаграми послідовності і кооперативні діаграми.

1.2.Діаграми послідовності (sequence diagrams)

На діаграмі послідовності об'єкт зображується у вигляді прямокутника на вершині пунктирної вертикальної лінії (рис.3.1).

Ця вертикальна лінія називається лінією життя об'єкта (lifeline). 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 діаграмі (зверху вниз). Кожне повідомлення може бути позначено ім'ям, за бажанням можна вказати також аргументи й деяку керуючу інформацію. Також можна використовувати самоделегування - повідомлення, яке об'єкт посилає самому собі, при цьому стрілка повідомлення вказує на ту ж саму лінію життя.

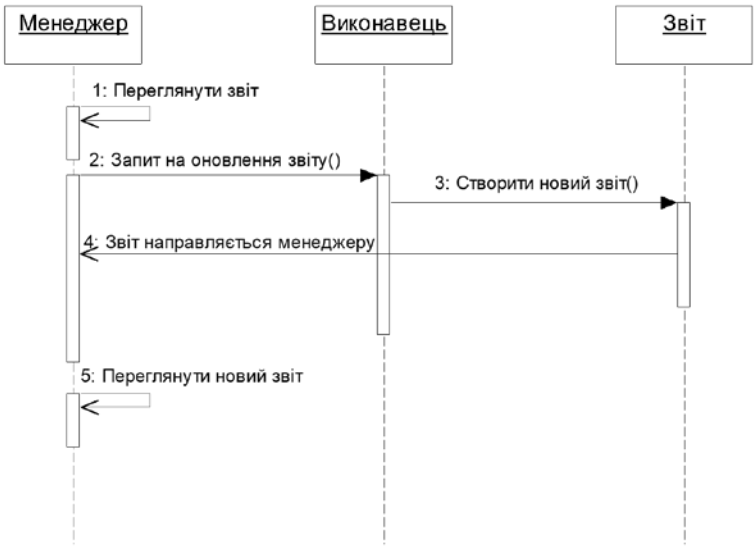

Рис. 3.1. Діаграма послідовності

Із всієї можливої керуючої інформації два її види мають істотне значення. По-перше, це умова, що показує в якому випадку посилається повідомлення. Наприклад, можна ввести умову: [Звіт_застарів() = true]. Тоді запит на оновлення звіту буде посилатися тільки при виконанні цієї умови. По-друге, корисним керуючим маркером є маркер ітерації, який показує, що повідомлення посилається багато разів для множини об'єктів-адресатів (наприклад, *оновити).

Активізації - прямокутники на лініях життя - показують, коли метод стає активним (під час його виконання або при очікуванні результату виконання якої-небудь процедури). Використовуючи механізм активізацій, можна більш чітко показати зміст самоделегування. Без цього досить важко визначити, де саме виконуються наступні після самоделегування звернення - у методі, який викликає, чи у методі, який викликається. Описання інструментів діаграм послідовності наведено в табл.3.1.

Таблиця 3.1

Кнопки панелі інструментів діаграм послідовності

Кнопка	Описання	Назва
	Вибір елемента моделі	Selection Tool
ABC	Введення тексту	Text Box
	Коментар	Note
	Зв'язок коментарю з елементом	Anchor Note to Item
	Об'єкт	Object
→	Повідомлення	Object Message

→	Виклик процедури	Procedure Call
↵	Самоделегування	Message to Self

1.3.Кооперативні діаграми (collaboration diagrams)

Другим видом діаграми взаємодії є кооперативна діаграма (рис. 3.2).

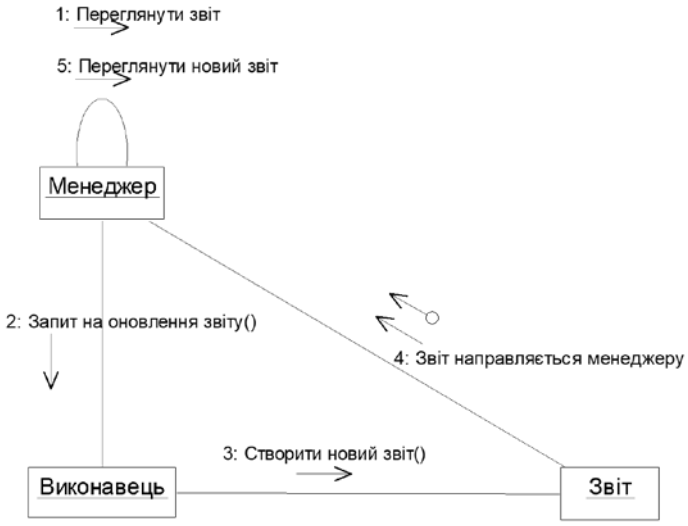

Рис. 3.2. Діаграма кооперації

На кооперативній діаграмі екземпляри об'єктів показані у вигляді піктограм. Лінії між ними позначають повідомлення, обмін якими здійснюється в межах даного варіанта використання.

Кожен вид діаграм взаємодії має свої переваги, тому вибір здійснюється, виходячи з пріоритетів розробника. На діаграмах послідовності робиться акцент саме на послідовності повідомлень, при цьому легше спостерігати порядок, у якому відбуваються різні події. У випадку кооперативних діаграм можна використати просторове розміщення об'єктів для того, щоб показати їхню статичну взаємодію.

Однією з головних властивостей будь-якої діаграми взаємодії є її простота. Подивившись на діаграму, можна легко побачити всі повідомлення. Однак при спробі зобразити щось більш складне, аніж єдиний послідовний процес без множини умовних переходів або циклів, цей підхід може не спрацювати.

Для відображення умовної поведінки на діаграмах взаємодії існує два підходи. Один з них складається із використання окремих діаграм для кожного сценарію. Другий полягає в тому, що повідомлення супроводжуються умовами, які показують поведінку об'єктів. Описання інструментарію діаграм кооперації наведено в табл.3.2.

Таблиця 3.2

Описання кнопок панелі інструментів кооперативних діаграм

Кнопка	Описання	Назва
	Вибір елемента моделі	Selection Tool
ABC	Введення тексту	Text Box
	Коментар	Note
	Зв'язок коментарю з елементом	Anchor Note to Item
	Об'єкт	Object
	Представник класу	Class Instance
	Зв'язок	Object Link
	Самоделегування	Link to Self
	Повідомлення	Link Message

Закінчення таблиці 3.2

Кнопка	Описання	Назва
	Відповідь	Reverse Link Message
	Потік даних	Data Flow
	Зворотний потік даних	Reverse Data Flow

2. Приклад

Діаграми послідовності моделі системи «Служба зайнятості», що показують взаємодію двох класів моделі: “Студент” і “БД студентів”, зображено на рис.3.3 і 3.4. На рис.3.5 і 3.6 та сама взаємодія показана за допомогою кооперативних діаграм, причому об’єкт “ Студент” визначений локально (літера L), а об’єкт “БД студентів” передається параметром в інший об’єкт (літера P).

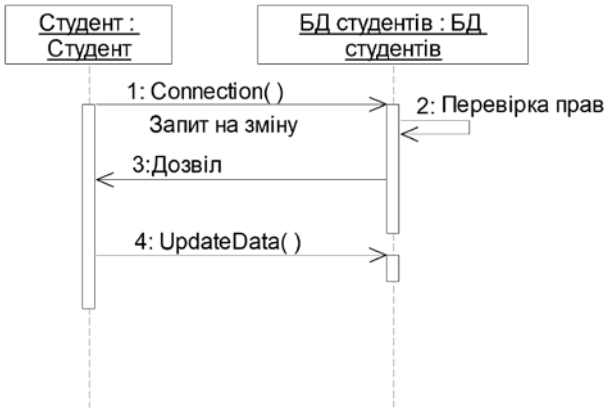

Рис. 3.3. Діаграма 1

Знайдемо числову оцінку для кожної з діаграм.

Діаграма 1

Оскільки на діаграмі послідовності зв'язки відсутні, проведемо розрахунок за скороченою формулою:

4

$$S = \frac{\Sigma S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}} = \frac{34}{1 + 6 + \sqrt{2}} = 4{,}04.$$

Діаграма 2

$$S = \frac{\Sigma S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}} = \frac{22}{1 + 4 + \sqrt{2}} = 3{,}43.$$

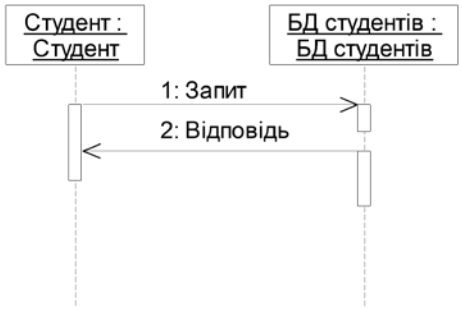

Рис. 3.4. Діаграма 2

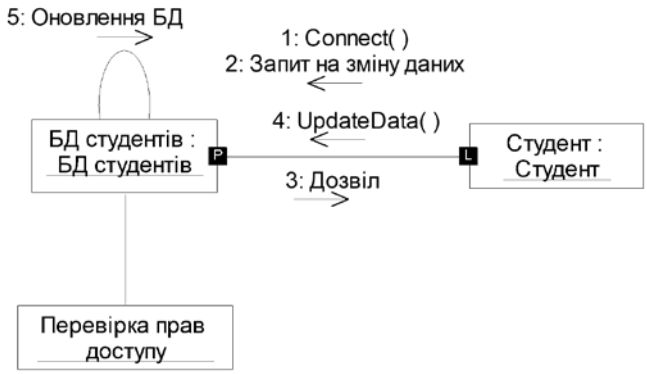

Рис. 3.5. Діаграма 3

Тепер розрахуємо оцінку для кооперативних діаграм.

Діаграма 3

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

Рис. 3.6. Діаграма 4

Діаграма 4

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} \frac{10+1}{1+2+1} = 2{,}75.$$

Таким чином, значення для діаграм 1 та 3 відповідають оптимальним, а для діаграм 2 і 4 - нижче оптимальних. Це можна пояснити низькою інформативністю діаграм 2 і 4, тому що взаємодія класів показана на них на занадто високому рівні.

3. Завдання

Для декількох варіантів використання з моделі інформаційної системи побудувати відповідні діаграми послідовності та кооперативні діаграми. Створимо діаграми послідовності і кооперативні діаграми для основного потоку подій варіанту використання Register for Courses «Зареєструватися на курси».

Контрольні питання

1. Яке призначення діаграм взаємодії? Як співвідносяться між собою діаграми варіантів використання й діаграми взаємодії?
2. Назвіть два види діаграм взаємодії. Що таке «життєва лінія» на діаграмі послідовності?
3. Як на діаграмі послідовності зображені повідомлення? Що таке самоделегування? Що показує активізація об'єкта?
4. Відмінність кооперативних діаграм від діаграм послідовності. Переваги й недоліки кожного виду взаємодії.

Практична робота №4

(2 год.)

Тема: Побудова діаграм станів

Мета роботи – вивчення діаграм станів та їх застосування в процесі проектування інформаційних систем.

1. Теоретичні відомості

Діаграми станів (state diagrams) є добре відомим засобом описання поведінки систем. Вони визначають усі можливі стани, в яких може перебувати конкретний об'єкт, а також процес зміни станів об'єкта в результаті впливу деяких подій.

На рис.4.1 показана діаграма станів UML, яка відображає поведінку звіту в системі керування проектами. На діаграмі зображені різні стани, в яких може перебувати звіт.

Рис. 4.1. Діаграма станів звіту в системі керування проектами

Процес починається з початкової точки, далі слідує найперший перехід у стан «Перевірка дати звіту». У поведінці об'єкта в системі можна виділити дії, що відображаються переходами, і діяльності, які відображаються станами. Хоча і перше і друге – це процеси, реалізовані, як правило, деяким методом класу «Звіт», вони трактуються по різному.

Дії завжди пов'язані з переходами і розглядаються, як миттєві й такі, що не перериваються. Діяльності пов'язані зі станами й можуть тривати досить довго. Діяльність може бути перервана в результаті настання деякої довільної події.

Перехід може містити позначку (мітку). Синтаксично мітка переходу складається із трьох частин, кожна з яких є необов'язковою: <Подія> [< Умова>]/< Дія >. Якщо мітка переходу не містить ніякої події, це означає, що перехід відбувається, як тільки завершується певна діяльність, пов'язана з цим станом.

Зі стану «Перевірка дати звіту» можливі два переходи. Мітка одного з них містить умову. Мається на увазі логічна умова, яка може приймати два значення: «істина» або «неправда». Умовний перехід виконується тільки в тому випадку, якщо умова набуває значення «істина», у протилежному випадку виконується перехід, не позначений умовою. З конкретного стану в цей момент часу може бути здійснений тільки один перехід; таким чином, умови є такими, що взаємно виключають одна одну для будь-якої події.

Існує два особливих стани: вхід і вихід. Будь-яка дія, пов'язана з подією входу, виконується, коли об'єкт входить у даний стан. Подія виходу виконується в тому випадку, коли об'єкт виходить із даного стану.

Діаграми станів добре використовувати для опису поведінки об'єкта в декількох різних варіантах використання. Вони не занадто придатні для опису поведінки ряду взаємодіючих об'єктів.

Рекомендується будувати діаграми станів тільки для тих класів, поведінка яких впливає на загальну поведінку системи, наприклад, для класів користувальницького інтерфейсу й керуючих об'єктів.

Описання панелі інструментів діаграм станів наведено в табл.4.1.

Таблиця 4.1

Описання кнопок панелі інструментів діаграм станів

Кнопка	Описання	Назва
	Вибір елемента моделі	Selection Tool
	Введення тексту	Text Box
	Коментар	Note
	Зв'язок коментарю з елементом	Anchor Note to Item
	Стан	State
	Вхід	Start State
	Вихід	End State
	Перехід у стан	State Transition
	Повернення	Transition to Self

2. Приклад

Діаграми станів екземпляра класу «Студент» зображено на рис.4.2 і 4.3. Ці діаграми показують стани екземпляра в ході взаємодії об'єкта класу «Студент» із БД студентів. Перша діаграма розписує стани об'єкта докладно, а друга показує тільки загальний стан взаємодії із БД.

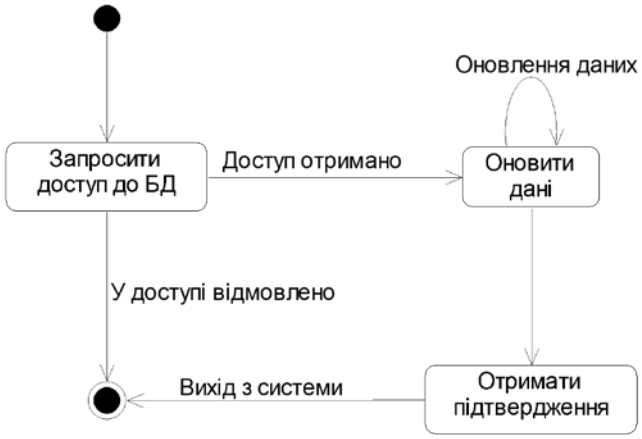

Рис. 4.2. Діаграма станів 1

Знайдемо кількісну оцінку для кожної з діаграм.

Діаграма 1

Оскільки на діаграмі станів зв'язки відсутні, проведемо розрахунок за скороченою формулою:

$$S = \frac{\Sigma S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}} = \frac{20}{1+5+1} = 2{,}86.$$

Діаграма 2

$$S = \frac{\Sigma S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}} = \frac{12}{1+3+1} = 2{,}4.$$

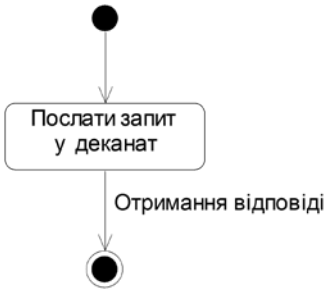

Рис. 4.3. Діаграма станів 2

Отриманий результат пояснюється наявністю недостатньо деталізованого стану на діаграмі 2.

3. Завдання

Вибрати в модельованій системі класи, для об'єктів яких побудувати діаграми станів, що характеризують поведінку об'єктів у декількох варіантах використання. Уточнення зв'язків між класами

- В процесі проектування зв'язку між класами (асоціації, агрегації та узагальнення) підлягають уточненню.

- Асоціації між граничними і керуючими класами відображають зв'язку, динамічно що виникають між відповідними об'єктами в потоці управління. Для таких зв'язків досить забезпечити видимість класів, тому вони перетворюються в залежності.
- Якщо для деяких асоціацій немає необхідності в двобічної зв'язку, то вводяться напрямки навігації.
- Агрегації, що володіють властивостями композиції, перетворюються в зв'язку композиції.

Контрольні питання

1. Призначення діаграм стану. Як відображаються дії й діяльності на діаграмах стану?
2. Що таке умовний перехід і як він описується на діаграмі?
3. Які особливі стани об'єкта відображаються на діаграмі? Які переваги й недоліки діаграм стану?

Практична робота №5

(2 год.)

Тема: Діаграми пакетів, компонентів і розгортання

Мета роботи – вивчення діаграм пакетів, діаграм компонентів і діаграм розгортання та особливостей їхнього застосування в процесі проектування програмних та апаратних частин системи.

1. Теоретичні відомості

1.1.Діаграми пакетів (package diagrams)

Одне з найважливіших завдань методології створення програмних систем – це розбиття великої системи на невеликі підсистеми. Саме з цього погляду зміни, пов'язані з переходом від структурного підходу до об’єктно-орієнтованого, є найбільш помітними. Одна з ідей полягає в групуванні класів у компоненти більш високого рівня. В UML такий механізм зветься пакетом.

Діаграма пакетів (package diagrams) – це діаграма, котра містить пакети класів і залежності між ними. Насправді, пакети й залежності є елементами діаграми класів, тобто діаграма пакетів – це одна з форм діаграми класів. Однак ці діаграми є корисними на практиці, а причини побудови таких діаграм різні.

Залежність між двома елементами має місце в тому випадку, якщо зміни у визначенні одного елемента викликають зміни в іншому. Щодо класів, то причини залежностей можуть бути самими різними: 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 містить у собі мінімізацію залежностей, що знижує вплив змін і потребує менше зусиль на їхнє внесення.

Класи предметної області, що моделюють діяльність організації, згруповані у два пакети: «Клієнти» й «Замовлення». Ці пакети зображено в числі інших елементів на рис.5.1.

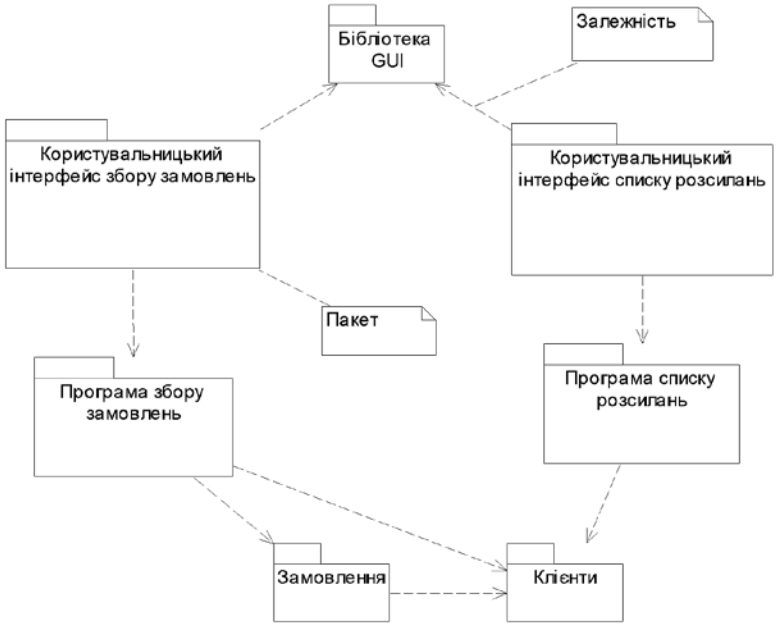

Рис. 5.1. Діаграма пакетів діяльності організації

«Програма збору замовлень» має залежності з обома пакетами предметної області. «Користувальницький інтерфейс збору замовлень» має залежності з «Програмою збору замовлень» і «Бібліотекою GUI».

Залежність між двома пакетами існує в тому випадку, якщо є певна залежність між будь-якими двома класами в цих пакетах. Наприклад, якщо деякий клас у пакеті «Список розсилання» залежить від якого-небудь класу в пакеті «Клієнти», то між відповідними пакетами існує залежність.

Пакети є життєво необхідним засобом для великих проектів. Їх варто використовувати в тих випадках, коли діаграма класів, що охоплює всю систему в цілому й розміщена на єдиному аркуші паперу формату А4, стає важкою для читання.

Пакети не дають відповіді на питання, яким чином можна зменшити кількість залежностей у системі, що проектується, однак вони допомагають виділити ці залежності. Зведення кількості залежностей до мінімуму знижує зв'язаність компонентів системи. Але евристичний підхід до цього процесу далекий від ідеалу.

Пакети особливо корисні при тестуванні. Кожен пакет при тестуванні може містити один або декілька тестових класів, за допомогою яких перевіряється поведінка пакета.

1.2.Діаграми компонентів (component diagrams)

Компоненти на діаграмі компонентів являють собою фізичні модулі програмного коду (рис.5.2). Зазвичай вони в точності відповідають пакетам на діаграмі пакетів (див. рис.5.1). Таким чином, діаграма компонентів відображає виконання кожного пакета в системі.

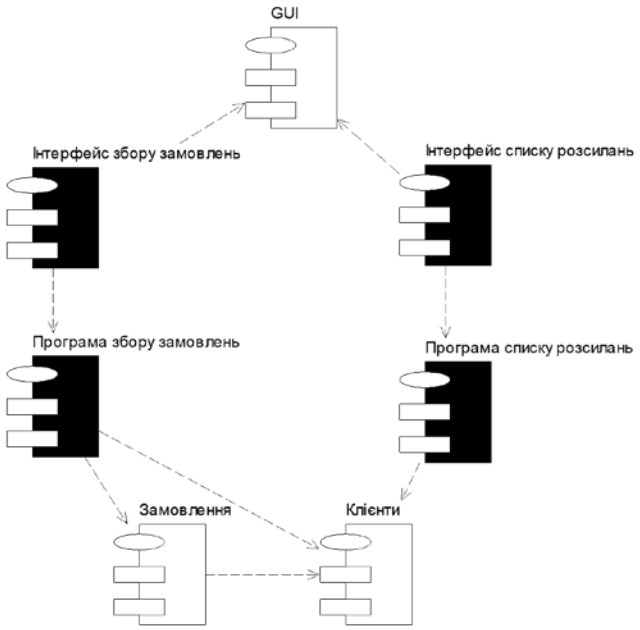

Рис. 5.2. Діаграма компонентів діяльності організації

Залежності між компонентами повинні збігатися із залежностями між пакетами. Ці залежності показують, яким чином одні компоненти взаємодіють з іншими. Напрямок цієї залежності вказує рівень поінформованості про комунікації. Описання панелі інструментів діаграм компонентів наведено в табл.5.1.

Таблиця 5.1

Описання кнопок панелі інструментів діаграм компонентів

Кнопка	Описання	Назва
	Вибір елемента моделі	Selection Tool

ABC	Введення тексту	Text Box
	Коментар	Note
	Зв'язок коментарю з елементом	Anchor Note to Item
	Компонент	Component
	Пакет	Package
	Залежність	Dependency
	Специфікація підпрограми	Subprogram Specification
	Тіло підпрограми	Subprogram Body
	Головна програма	Main Program
	Специфікація пакета	Package Specification
	Тіло пакета	Package Body
	Специфікація завдання	Task Specification
	Тіло завдання	Task Body

1.3.Діаграми розгортання (deployment diagrams)

Діаграма розгортання відображає фізичні взаємозв'язки між програмними й апаратними компонентами системи. Вона є гарним засобом для того, щоб показати маршрути переміщення об'єктів і компонентів у розподіленій системі.

Кожен вузол на діаграмі розгортання являє собою деякий тип обчислювального пристрою, у більшості випадків – частину апаратури. Ця апаратура може бути простим пристроєм або датчиком, а може бути й великим комп'ютером.

Персональний комп'ютер (ПК), пов'язаний з UNIX-сервером за допомогою протоколу TCP/IP, зображений на рис.5.3. З'єднання між вузлами зображують комунікаційні канали, за допомогою яких здійснюються системні взаємодії.

На практиці такі діаграми застосовуються рідко. Загалом ці діаграми корисно застосовувати, щоб виділити особливі фізичні характеристики конкретної системи. По мірі поширення розподілених систем важливість даних діаграм зростає. Описання панелі інструментів діаграм розгортання наведено у табл.5.2.

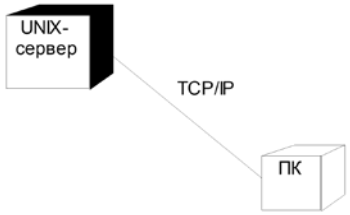

Рис. 5.3. Приклад діаграми розгортання

Таблиця 5.2

Кнопки панелі інструментів діаграм розгортання

Кнопка	Опис	Назва
	Вибір елемента моделі	Selection Tool
ABC	Введення тексту	Text Box
	Коментар	Note
	Зв'язок коментарю з елементом	Anchor Note to Item
	Процесор	Processor
	З'єднання	Connection
	Пристрій	Device

2. Приклади

Порівнювати діаграми пакетів, компонентів і розгортання в загальному випадку безглуздо, тому що ці діаграми не існують самі по собі, а є інтерпретацією деякої діаграми класів, для якої й доречно проводити порівняння з іншими діаграмами класів.

Діаграма пакетів містить один тип елементів – пакет і один тип зв'язків – залежність, а тому числова оцінка для діаграми пакетів не настільки важлива, як для діаграми класів.

Діаграма пакетів зображена на рис.5.4, а на рис.5.5 – діаграма компонентів системи «Служба зайнятості в рамках ВНЗ».

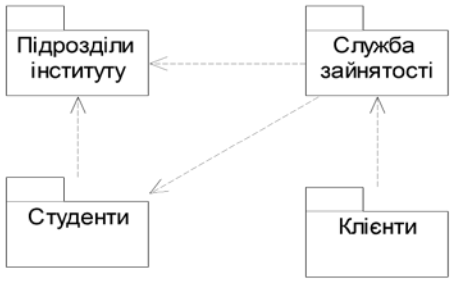

Рис. 5.4. Діаграма пакетів «Служби зайнятості в рамках ВНЗ»

Діаграми компонентів і розгортання будуються й використовуються на етапі реалізації й супроводу, коли базова архітектура системи вже зазвичай визначена; тому вони однозначно можуть бути отримані із діаграми класів і для них досить навести по одному прикладу.

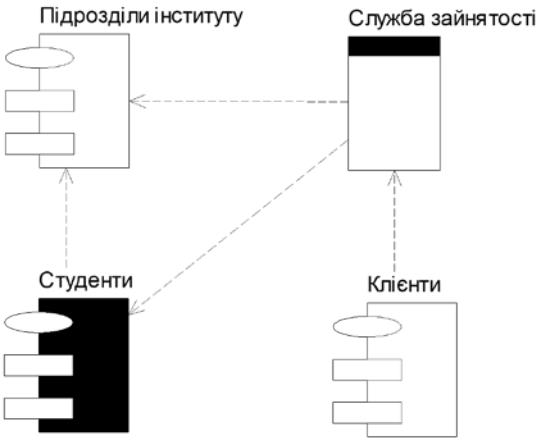

Рис. 5.5. Діаграма компонентів «Служби зайнятості в рамках ВНЗ»

На рис.5.5 зображена діаграма компонентів, що побудована на основі діаграми пакетів, зображеної на рис.5.4. На рис.5.6 зображена діаграма розгортання системи «Служба зайнятості в рамках ВНЗ». Оцінка для діаграми компонентів дорівнює:

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{16+8}{1+4+\sqrt{2}} = 3{,}74.$$

Очевидно, що ця оцінка буде дорівнювати оцінці для діаграми пакетів з рис.5.4. Оцінка для діаграми розгортання дорівнює:

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{12+4}{1+5+2} = 2.$$

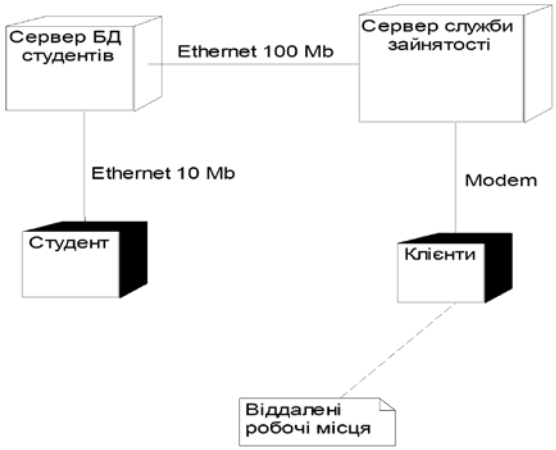

Рис. 5.6. Діаграма розгортання «Служби зайнятості в рамках ВНЗ»

3. Завдання

Побудувати для системи, що моделюється, загальну діаграму пакетів, визначити на ній пакети з необхідними системними бібліотеками, відобразити залежності між пакетами. Побудувати для цієї системи діаграму компонентів, що відповідає побудованій діаграмі пакетів, системні пакети зобразити у вигляді специфікацій пакетів. Побудувати для проектованої системи кілька варіантів діаграми розгортання (для архітектури «клієнт-сервер», трирівневої архітектури тощо), обґрунтувати кожен варіант, запропонувати найбільш оптимальний.

Контрольні питання

1. Яку проблему проектування покликані вирішити діаграми пакетів? У чому відмінність діаграм пакетів від діаграм класів?
2. У чому зміст залежності між елементами діаграми пакетів?
3. Що таке інтерфейс класу? За якими ознаками класи групуються в пакети?
4. Які види елементів моделі зображені на діаграмі компонентів? Як зв'язані між собою діаграми пакетів і діаграми компонентів?
5. Які сутності відображаються на діаграмах розгортання та у яких випадках необхідне застосування цих діаграм?

Практична робота №6

(2 год.)

Тема: Генерація вихідних текстів програм (code generation) та зворотне проектування (REVERSE ENGINEERING)

Мета роботи – вивчення можливостей кодогенерації та засобів зворотного проектування CASE-засобу Rational Rose.

1. Теоретичні відомості

1.1. Генерація вихідних текстів програм (code generation)

Усі розробники зіштовхуються із ситуацією, коли доводиться проектувати великі класи. Під час ручного введення та оголошення є ряд “підводних каменів”: по-перше, постановник задач, як правило, описує “що потрібно” на словах, у крайньому випадку з мінімальним паперовим супроводом; по-друге, розробник, що створює систему, в більшості випадків ігнорує коментарі, якими необхідно супроводжувати програмний код. Система кодогенерації Rational Rose дозволяє, поряд з іншими засобами проектування, побудувати процес розробки програмного забезпечення як виробничий процес із строгим розподілом ролей та повноважень.

Для демонстраційних цілей спроектуємо тільки один клас. Назвемо його MyString. В його обов'язки входять основні операції над масивами (одержання розміру, друк, порівняння, копіювання). Як приклад опишемо спочатку цей клас на C++:

```
  Class MyString  {
Protected:
  Char *TmpString;
Public:
  Int Stat;
  Int Count;
  Int GetStringSize(Char *);
  Int PrintString(Char *);
  Int CmpString(Char *, Char *);
  Int CpyString (Char *, Char *);
  };
```

Тепер засобами Rational Rose спроектуємо клас в графічному вигляді. Кожен атрибут задається окремо з коментарем і записом типу (public, protected, private). На рис.6.1 показані специфікації для TmpString. Подібним чином записуються всі змінні.

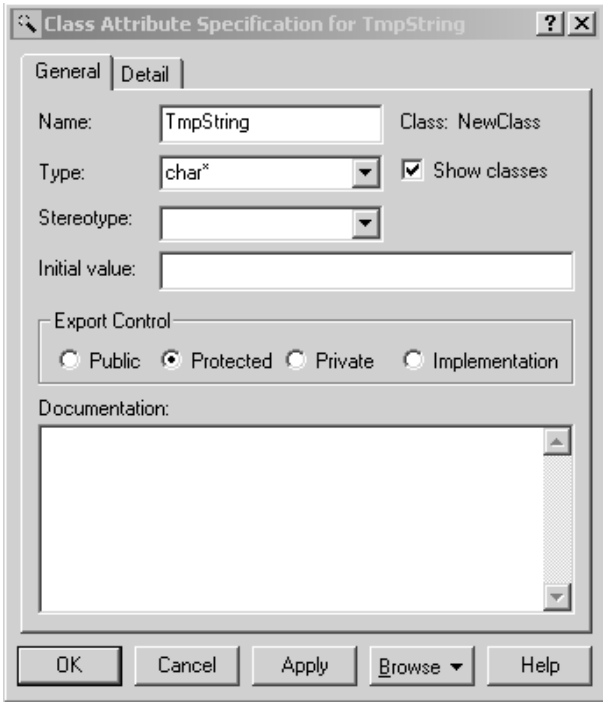

Рис. 6.1. Специфікації для змінної TmpString

У плані описання функцій все аналогічно, тільки крім опису самої функції (тип значення, що вона повертає) необхідно розписати специфіку кожного вхідного параметра, доповнивши все це найдокладнішими коментарями. У

всіх продуктах компанії Rational прийнято давати коментарі для будь-якої малопомітної операції, оскільки згодом, при генерації звітів не потрібно буде ще раз вручну доводити документ, щоб показати його керівництву або передати розробникові як технічне завдання.

Результатом виконання вищеописаних дій буде поява класу з розписаними специфікаціями. Сам клас показаний на рис.6.2. Зауважимо, що в графічному вигляді можна оцінити основні властивості кожного елемента.

Наступний крок у роботі – одержання коду на C++. Тут потрібно розвіяти існуючий міф про стовідсоткову генерацію коду. Rational Rose у принципі не може дати готового коду, вона здатна лише спроектувати клас і розписати специфікацію кожного елемента, підставити шаблони членів класу для подальшого заповнення кодом. А для стовідсоткової генерації робочого коду на C++ використовується Rational Rose RealTime, який у цьому лабораторному практикумі не розглядається.

Отже, повернемося до кодогенерації (точніше сказати, до класогенерації). Через систему меню (Tools) вибираємо підтримуючу мову для описання спроектованого класу (у цьому випадку це C++), і викликаємо Code Generation. Результатом роботи буде поява двох файлів: MyString.h і MyString.cpp. У першому розписується сам клас, а другий є шаблоном для подальшого заповнення відповідним кодом.

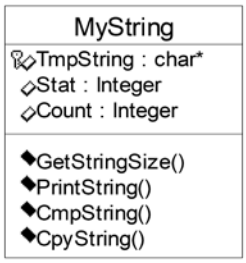

Рис. 6.2. Описання класу MyString

Нижче наводиться роздрук файлу MyString.h. Весь наведений матеріал отриманий без зміни настроювань і без додаткового виправлення. Маючи подібний шаблон, стає не важливо, який саме розробник почав створювати кодування логіки класу.

Для одержання ж докладного звіту за класом або технічним завданням можна скористатися інструментом Rational SoDA.

Наступне завдання, з яким допоможе впоратися Rational Rose – це аналіз існуючої системи. Навіщо переписувати й документувати великі системи заново, якщо можна скористатися функцією зворотного проектування, що дозволить із наявного коду побудувати візуальну модель і вже візуально дописати необхідні властивості й атрибути, а також нові класи. А наприкінці згенерувати весь спектр файлів, необхідних для подальшої роботи програмістів. Такий підхід називається ітераційним моделюванням (round-trip modeling) і повністю підтримується в Rational Rose.

1.2. Зворотне проектування (reverse engineering)

Зворотним проектуванням називається процес перетворення в модель коду, записаного на якій-небудь мові програмування. У результаті цього процесу виходить величезний обсяг інформації, частина якої перебуває на більш низькому рівні деталізації, ніж необхідно для побудови корисних моделей. Водночас зворотне проектування ніколи не буває повним. Оскільки пряме проектування веде до втрати інформації, повністю відновити модель на основі коду не вдасться, якщо тільки інструментальні засоби не включали в коментарях до вихідного тексту інформацію, що виходить за межі мови реалізації.

Одна з незаперечних переваг Rational Rose – зворотне проектування, оскільки розробнику й проектувальнику важливо побачити перед змінами вже працюючу систему в нормальному графічному вигляді. Як правило, візуально-графічний ряд має куди більший вплив, ніж перегляданння технічних завдань і текстів програм. Тим більше, що проект, який піддався зворотному проектуванню, може бути дороблений і знову згенерований (а згодом і скомпільований). Rational Rose надає для цього всі необхідні засоби.

Для здійснення зворотного проектування в Rational Rose передбачений потужний модуль Analyzer, основне призначення якого (що випливає з назви) – аналіз програм, написаних на C і C++. Цей модуль здатний проаналізувати наявний файл на одній з вище згаданих мов і перетворити його у візуальну модель, привласнивши вихідному файлу розширення mdl. Далі файл можна спокійно відкрити для модифікації в Rational Rose вже у візуальному режимі.

Analyzer являє собою окремий програмний файл, який викликається як із самої Rose, так і звичайним способом. Модуль входить не в усі поставки Rational Rose, а тільки в Enterprise, Professional і RealTime. У поставку Data Modeler цей модуль не входить, оскільки специфіка поставки не передбачає генерації коду та зворотного проектування.

Для правильного перетворення коду в модель необхідно провести кілька настроювань. На рис.6.3 показаний зовнішній вигляд програми із залученням стандартних настроювань.

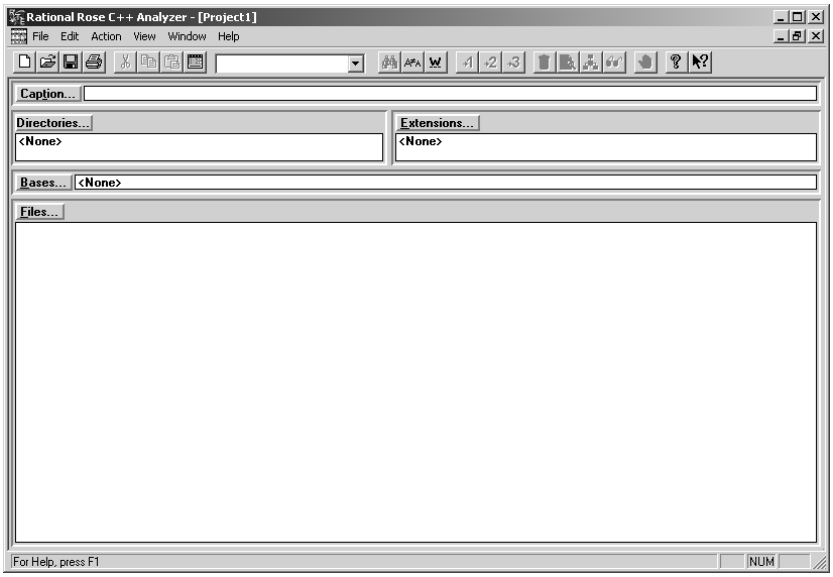

Рис. 6.3. Зовнішній вигляд програми Analyzer

Основні поля, що підлягають обов'язковому заповненню на першому етапі такі:

• Caption - ім'я проекту. Згодом ім'я моделі буде визначено за ім'ям проекту;

• Directories - шлях до вихідної директорії. За замовчуванням Rational Rose використовує для зберігання вихідних модельних файлів директорію C++\Source домашньої директорії, що іноді може приносити деякі незручності;

• Extensions - типи використовуваних розширень. Тут можна настроїти систему так, щоб вона розпізнавала тільки певні види розширень;

• Bases - місце збереження поточного проекту.

• Files - список з файлів, що підлягають генерації.

Для проведення правильного зворотного проектування необхідно заповнити вищеописані поля. Всі файли, які підлягають зворотному проектуванню, вказуються в полі Files. Варто враховувати, що при цьому ми одержимо візуальну модель взаємодії класів і структур; а отже, мова не йде про те, щоб на візуальній моделі відбився існуючий код системи. Крім того, всі нестандартні конструкції не будуть виведені в модель (аналізатор їх просто проігнорує), а це значить, що будь-яке відхилення від заздалегідь відомих конструкцій приводить до того, що в початковому варіанті Rose не зможе правильно проаналізувати код. Це не є недоліком, оскільки в арсеналі Analyzer є інструменти тонкого настроювання, що дозволяють настроїти все таким чином, щоб специфіка конкретного проекту була повністю врахована.

Процес зворотного проектування ділиться на два етапи: аналіз і генерацію моделі. На першому етапі виконуються всі підготовчі операції з аналізу тексту програми на відсутність синтаксичних помилок. Другий етап – це перетворення коду в модель.

Усі операції виконуються незалежно, що дає більший маневр для розробника, який, наприклад, хоче провести тільки синтаксичний розбір тексту, без генерації моделі.

За відсутності помилок у файлі можна приступити до генерації моделі. З метою оптимізації часу генерації в Rose передбачено три способи проведення зворотного проектування, кожен з яких може охопити й виконати певний сегмент робіт. Якщо користувачеві за якимись причинами не підходить жоден із трьох наперед встановлених способів, то Rose допускає створення власного способу зворотного проектування.

Поговоримо докладніше про такі три стандартні способи:

• FirstLook - наближений перегляд тіла програми;

• DetailedAnalysis - детальний аналіз проекту;

• RoundTrip - комбінація двох перерахованих вище способів.

Останній дозволяє безболісно будувати й перебудовувати застосування, які розробляються за принципом кругової розробки.

Усі настройки можуть бути змінені на розсуд користувача. У разі збереження змін можна вказати нове ім'я шаблона або перезаписати вже існуюче, що дозволить при частому використанні зворотного проектування не втрачати час на встановлення потрібного пункту. Вибір відповідного пункту обов'язково позначається на швидкості аналізу.

Необхідно відзначити таку особливість модуля Analyzer: після аналізу створюється не тільки модель, але й Log-файл із повідомленнями, що виникли в результаті сканування програми. Цей файл може містити як попередження, так і помилки. Особливість генерації моделі полягає в тому, що вона відбудеться незважаючи на помилки в тексті програми (звісно, мова не йде про правильну модель). Цю особливість варто враховувати й уважно аналізувати файл звіту після генерації моделі.

Ще одна важлива нотатка. Як правило, зворотному проектуванню піддається повноцінний проектний файл, що містить у собі й директиви #INCLUDE для визначень, і коментарі, а також інші супровідні інструкції. Природно, що розробнику хочеться мати такий інструмент, який адекватно буде реагувати на всі складові. Для цього модуль Analyzer у режимі (DetailedAnalysis) забезпечує таке:

• аналіз і перетворення у візуальну модель класів і структур;

• генерацію зв'язків у моделі (між класами або структурами);

• знаходження у вихідному тексті коментарів і перенесення їх як атрибутів компонентів моделі. Тобто, якщо вихідний текст містить коментарі, то вони всі перейдуть у вигляді атрибутів до відповідного елемента (змінної, масиву тощо);

• завантаження в проект всіх файлів-заголовків (один за одним, по ланцюгу).

Тепер перейдемо до практики. Нашою метою буде одержання графічної моделі із класу, що записаний на мові програмування. Особливо хочеться ще раз звернути увагу на коментарі. Кожен рядок містить коментар. Зміст

зворотного проектування полягає не тільки в тому, щоб коректно зобразити модель, але й правильно описати специфікації кожної складової класу.

За основу програми візьмемо такий клас:

```
//It's main class
class string {
public:
  char  *string; //Structure's pointer
  int  buffer[100]; //Temporary buffer
  char  name[10]={"Massiv"}; //Name of data
  int  a; //Integer
  int  b; //Integer
  void string(void); //constructor
  void ~string(void); //destructor
char StringCopy(char *, //Buffer
char *,  //source1
char *); //source2
private:
  int tmp_a;
  int tmp_b;
};
```

Результат зворотного проектування зображено на рис.6.4.

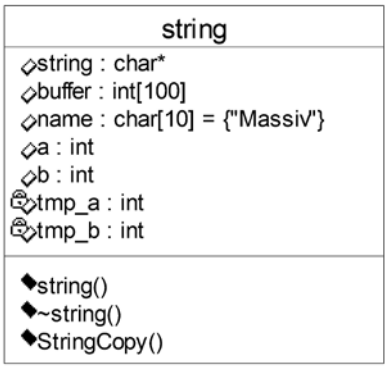

Рис. 6.4. Зворотне проектування класу string

Рис.6.4 показує модель класу string. А на рис.6.5 відображається вкладка, що описує функції класу. Як і у випадку зі змінними, імена функцій відображаються на екрані. Також доступний вхід у специфікації конкретної функції. Якщо ще раз повернутися до лістингу, то можна звернути увагу на декларацію функції StringCopy, у якій вхідні параметри докладно документовані. Отже, якщо був застосований подібний підхід до документування, то коментар кожного параметра перенесеться як описовий коментар у відповідну частину атрибута моделі класу.

Тобто виходить, що дуже вигідно піддавати обробці вихідні тексти, написані за всіма правилами програмування.

Виразні засоби візуального проектування й аналізу роблять Rational Rose незамінним інструментом при створенні великих інформаційних систем. Особливо повно Rose розкриває свої можливості під час аналізу ефективності не нової системи, а вже існуючої. Вищенаведені приклади показують, що дає інструмент під час аналізу проекту на предмет підвищення його ефективності. Ця проблема не є надуманою, оскільки подібний аналіз потрібний компаніям, що переводять, наприклад, старе програмне забезпечення на нові платформи й нові технології.

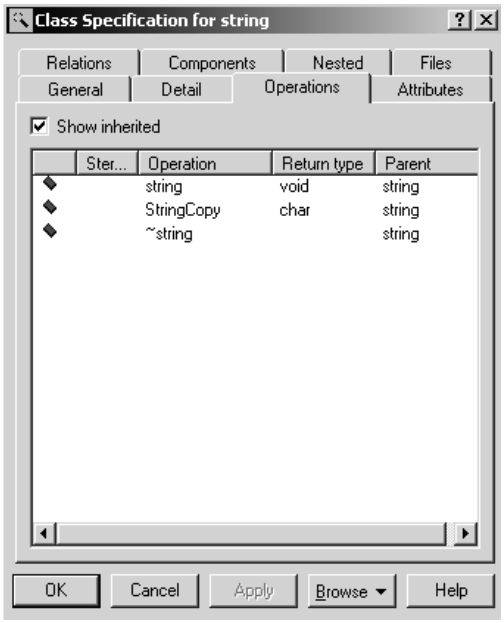

Рис. 6.5. Функції класу string

Як видно, модель нічого не втрачає у результаті зворотного проектування, у той час як саме зворотне проектування представляє потужний механізм аналізу ефективності існуючих програмних напрацювань шляхом натискання на одну-дві кнопки.

Rational Rose має у своєму арсеналі можливість прямого й зворотного проектування на ADA, Java, C++, COM, DDL, Basic, XML; схеми Oracle і SQL Server. Rose має відкрите, добре документоване API, яке дозволяє будь-якій людині створити додатковий модуль (міст) для будь-якої мови програмування. Тому на сьогодні Rational Rose - це унікальний продукт у плані відкритої архітектури.

2. Приклад кодогенерації

Генеруються файли MyString.h та MyString.cpp. Приклад файлу MyString.h :

```
//## begin module%1.3%.codegen__version preserve=yes
```

```
// Read the documentation to learn more about C-f + code generator versioning.
//## end module%1.3%.codegen_version
//## begin module%395AF70D0321.cm preserve=no
//  %X% %Q% %Z% %W%
//## end module%395AF70D0321.cm
//## begin module%395AF70D0321.cp preserve=no
//## end module%395AF70D0321.cp
//## Module: MyString%395AF70D0321; Pseudo Package specification
//## Source file: C: \ProgramFiles\Rational\Rose\C++ \source\MyString.h
#ifndef MyString_h
#define MyString__h 1
//## begin module%395AF70D0321.additionalIncludes preserve=no
//## end module%395AF70D0321.additionalIncludes
//## begin module%395AF70D0321.includes preserve=yes
//## end module%395AF70D0321.includes
//## begin module%395AF70D0321.additionalDeclarations preserve=yes
//## end module%395AF70D0321.additionalDeclarations
//## begin MyString%395AF70D0321.preface preserve=yes
//## end MyString%395AF70D0321.preface
//## Class: MyString%395AF70D0321
// Даний клас дозволяє проводити різні операції над масивами символів.
//## Category: <Top Level>
//## Persistence: Transient
//## Cardinality/Multiplicity: n
class MyString {
//## begin MyString%395AF70D0321.initialDeclarations preserve=yes
//## end MyString%395AF70D0321.initialDeclarations
public:
//## Constructors (generated)
MyString();
//## Destructor (generated) ~MyString();
//## Assignment Operation (generated) MyString & operator^ (const MyString &right);
//## Equality Operations (generated)
int operator==(const MyString &right) const;
int operator!=(const MyString &right) const;
//## Other Operations (specified)
//## Operation: GetStringSize%395AF87900E9
// Підраховує кількість символів у переданому масиві
Int GetStringSize (Char *massiv); // Покажчик на масив
//## Operation: PrintString%395AF88800B9
……..
```

3. Приклад створення комплекту діаграм UML для системи реєстрації на курси

Виконати зворотне проектування і порівняти вихідні тексти програми й отриману модель.

3.1. Генерація коду

Процес генерації коду складається з чотирьох кроків:

1. Перевірка коректності моделі.
2. Встановлення властивостей генерації коду.
3. Вибір класу, компонента або пакета.

4. Генерація коду. Для перевірки моделі:

1. Виберіть в менюTools / Check Model.
2. Проаналізуйте всі знайдені помилки в вікні журналу.

До найбільш поширених помилок відносяться такі, наприклад, як повідомлення на діаграмі послідовності або кооперативної діаграмі, що не співвіднесені з операцією, або об'єкти діаграм, що не співвіднесені з класом.
Щоб виявити порушення правил доступу, виконайте наступне.
1. Виберіть в меню Report / Show Access Violations. Проаналізуйте всі порушення правил доступу у вікні.
Для аналізу властивостей генерації коду виберіть Tools / Options, а потім вкладку відповідної мови. У вікні списку можна вибрати клас, атрибут, операцію і інші елементи моделі
Під час генерації коду Rose вибирає інформацію з логічного і компонентного уявлень моделі і генерує великий обсяг "скелетного" коду:
- Класи. Генеруються всі класи моделі.
- Атрибути. Код включає атрибути кожного класу, в тому числі видимість, тип даних і значення за замовчуванням.

- сигнатури операцій. Код містить визначення операцій з усіма параметрами, типами даних параметрів і типом значення, що повертається операції.
- Зв'язки. Деякі з зв'язків моделі викликають створення атрибутів при генерації коду.
- Компоненти. Кожен компонент реалізується у вигляді відповідного файлу з вихідним кодом.

3.2. Генерація коду C ++

1. Відкрийте діаграму компонентів Main системи.
2. Виберіть всі об'єкти на діаграмі компонентів.
3. ВиберітеTools / C ++ / Code Generationвменю.
4. Чи відбудеться генерація коду.
5. У вікні Log внизу подивіться помилки і попередження в побудованій нами моделі.
6. Перегляньте результати генерації (меню Tools / C ++ / Browse Header і Tools / C ++ / Browse Body).
7. Збережіть модель File / Save. Останній етап створення моделі системи для реєстрації курсів збережеться в файлі ФаміліяАнгл5.

4. Завдання

Згенерувати код для декількох класів системи, порівняти модель та отриманий вихідний код.

Контрольні питання

1. Які переваги автоматичної кодогенерації?
2. Які види діаграм використовуються для генерації коду?

3. Які компоненти вихідного коду генерує Rational Rose?

4. Як у вихідному коді відбиваються атрибути й операції класу? Чому важливо детально коментувати компоненти моделі?

5. Для чого призначене зворотне проектування? Які основні способи зворотного проектування наявні в Rational Rose?

6. Як побудувати модель в Rational Rose на основі довільної програми на C++?

7. Який модуль використовується для аналізу вихідного коду?

Рекомендовані джерела:

Основна література

1.Буч Г., Якобсон А., Рамбо Дж. UML. Классика компьютерных технологий: Пер. с англ. – СПб.: Питер, 2006. – 736 с.

2.Леоненков А.В. Самоучитель UML. – СПб.: БХВ-Петербург, 2006. – 432 с.

3.Боггс У., Боггс М. UML и Rational Rose 2002: Пер. с англ. – М.: Лори, 2004. – 509 с.

4.Трофимов С.А. CASE-технологии: Практическая работа в Rational Rose. – М.: Бином-Пресс, 2002. – 288 с.

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

6.Соммервилл И. Инженерия программного обеспечения: Пер. с англ. – М.: Изд. дом Вильямс, 2002. – 624 с.

7.Бабенко Л.П., Лаврищева К.М. Основи програмної інженерії. Навч. посіб. – К.: Знання, КОО, 2001. – 269 с.

Додаткова література

1. Харченко О.Г., Райчев І.Е. Організація баз даних і знань // Лабораторний практикум. – К.: НАУ, 2006. – 52 с.

2.Райчев І.Е. Принципи проектування відкритих розподілених систем: Структурний системний аналіз і проектування інформаційних систем // Лабораторний практикум – К.: НАУ, 2007. – 80 с.

Додаткові ресурси:

1. http://www.softforfree.com/programs/rational_rose-34201.html