МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
КІРОВОГРАДСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСІТЕТ

МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Технологія проектування програмних систем

Методичні вказівки до виконання лабораторних робіт

з елементами кредитно – модульної
системи організації навчального процесу

для студентів денної форми навчання за напрямом підготовки
7.050102.01, 8.050102.01 «Комп'ютерні системи та мережі»
7.050102.02, 8.050102.02 «Системне програмування»

Укладачі:
Доцент Смірнов В.В.
Ст. викладач Смірнова Н.В.

Кіровоград 2013

Технологія проектування програмних систем: Методичні вказівки до виконання лабораторних робіт для студентів денної форми навчання за напрямом підготовки 7.050102.01, 8.050102.01 «Комп'ютерні системи та мережі», 7.050102.02, 8.050102.02 «Системне програмування» / Укл.: / Смірнов В.В., Смірнова Н.В. – Кіровоград: КНТУ, 2013. – 90 с.

Затверджено на засіданні кафедри ПЗ:
11 вересня 2013 р. протокол № 2;
17 вересня 2014 р., протокол № 4.

Укладачі:
Смірнов Володимир Вікторович, к.т.н., доцент кафедри ПЗ,
Смірнова Наталія Володимирівна, к.т.н., старший викладач кафедри ПЗ.

Для студентів денної форми навчання, що вивчають навчальну дисципліну "Технологія проектування програмних систем" за напрямом підготовки 7.050102.01, 8.050102.01 «Комп'ютерні системи та мережі», 7.050102.02, 8.050102.02 «Системне програмування».

Лабораторні роботи розроблені на сучасному науково - методичному рівні. Відповідають вимогам до курсу практичного навчання студентів, у доступній формі викладені теоретичні відомості і описані практичні кроки оволодіння основами технології проектування програмних систем

ЗМІСТ

1. Опис навчальної дисципліни
“Технологія проектування програмних систем"

Дисципліна вивчає методи проектування локальних та розподілених програмних систем.

Робоча програма складена на базі освітньо – професійної програми для студентів денної форми навчання за напрямом підготовки 7.050102.01, 8.050102.01 «Комп’ютерні системи та мережі» та 7.050102.02, 8.050102.02 «Системне програмування»

Дисципліна відноситься до вибіркового блоку програми.

Мета вивчення дисципліни

Основна мета курсу полягає в придбанні досконалих знань в області розробки та проектування сучасного програмного забезпечення і програмних систем, з використанням сучасних технологій.

Завдання вивчення дисципліни

- вивчення процесу розробки програмного забезпечення програмних систем;
- аналіз встановлення і специфікація вимог до програмної системи;
- заглиблений аналіз та обґрунтування основ проектування систем;
- проектування користувальницького інтерфейсу;
- проектування баз даних, програм і транзакцій;
- тестування і керування змінами.

Предметом дисципліни є методи і алгоритми проектування та створення програмних систем.

Студент повинен знати і вміти після опанування дисципліни:

- володіти процесом розробки програмного забезпечення;
- проводити аналіз специфікації вимог до програмної системи;
- проводити заглиблений аналіз та обґрунтування методів проектування систем;
- вміти проектувати користувальницький інтерфейс;
- володіти проектуванням баз даних, програм і транзакцій.

Перелік дисциплін, необхідних для вивчення даної дисципліни.

- об'єктно-орієнтоване програмування
- мережі ЕОМ
- програмне забезпечення управляючих мікро ЕОМ
- операційні системи

2. Теми і зміст лекційних занять

№ теми	Тематика і зміст лекцій	Години
1	2	3
1	**Процес розробки програмних систем.** Стандарт ISO 9000. UML.CASE-засоби та удосконалення процесу. Планування розробки системи. Підхід SWOT. Підхід VCM. Підхід BPR. Підхід ISA. Системи для трьох рівнів керування. Етапи життєвого циклу програмного забезпечення. Планування проекту протягом життєвого циклу ПЗ	0,66
2	**Підстави аналізу вимог** Основи об'єктної технології. Аналіз об’єктів. Моделювання прецедентів. Моделювання видів діяльності. Моделювання класів. Моделювання взаємодій. Діаграма станів.	0,66
3	**Встановлення вимог** Принципи Встановлення вимог. Виявлення вимог. Прототипування. Спільна розробка додатків (JAD-метод) Системні сервіси. Системні обмеження.	0,66
4	**Принципи специфікації вимог** Специфікації вимог. Специфікації станів. Моделювання класів. Виявлення класів. Підхід на основі використання іменних груп. Підхід на основі використання загальних шаблонів для класів. Підхід на основі використання прецедентів. Комплексний підхід.	0,66
5	**Системне проектування** Архітектура програмного забезпечення. Розподілена архітектура. Триланкова архітектура. Рівні BCED. Стратегія повторного використання. Компоненти. Діаграма компонентів. Розгортання. Реалізація Web-додатків. Проект розгортання. Розгортання Web-додатків.	0,66

6	**Проектування баз даних** Рівень постійних об'єктів бази даних. Моделі даних. Відображення об'єктів у базу даних. Модель об'єктної бази даних. Об'єктно-реляційна модель бази даних. Модель реляційної бази даних. Елементарні типи моделі РБД. Реляційні таблиці.	0,66
7	**Проектування програмної системи** Зв'язність та ув'язування класів. Види ув'язування класів. Закон Деметра. Методи відкриття доступу. Динамічна класифікація і зв'язність класів зі змішаними екземплярами. Проектування клієнт-серверних кооперативних взаємодій. Збережені процедури. Тригери. Проектування транзакцій. Короткі транзакції. Рівні ізольованості. Автоматичне відновлення. Програмувальне відновлення. Точка збереження. Тригерний відкат.	1
8	**Тестування і керування змінами** Тестування системних сервісів. Наскрізний контроль. Інспекція. Тестування відносно специфікації. Тестування відносно програмного коду. Тестування системних обмежень. Тестування користувальницького інтерфейсу. Тестування баз даних. Тестування авторизації. Тестування загальних обмежень. Документація по тестуванню і керуванню змінами.	1
	Всього:	**6**

3. Практичні заняття з дисципліни

“Технологія проектування програмних систем"

№ заняття	Назва практичного заняття	Кільк. годин
1.	Планування розробки системи. Аналіз вимог і моделювання	1
2.	Специфікації вимог. Прототипування і спільна розробка додатків	1
3.	Системне проектування. Проектування баз даних	1
4.	Проектування програмної системи	1
	Всього годин	**4**

4. Змістовні модулі

Навчальне навантаження складається з 3 – змістовних модулів, які включають в себе лекції, практичні роботи, самостійну роботу і контроль знань. Система оцінки успішності в балах включає тестовий поточний контроль при виконанні практичних робіт, модульний контроль і оцінку самостійної роботи.

Перший модуль.

Процес розробки програмних систем. Підстави аналізу вимог. Встановлення вимог. Принципи специфікації вимог. Системне проектування

Лабораторні роботи:

№1 Планування розробки системи. Аналіз вимог і моделювання

Другий модуль.

Проектування баз даних. Проектування програмної системи

Лабораторні роботи:

№2 Специфікації вимог. Прототипування і спільна розробка додатків

№3 Системне проектування. Проектування баз даних

Третій модуль.

Тестування і керування змінами

Лабораторні роботи:

№4 Проектування програмної системи

5. Оцінка успішності в балах при повному виконанні умов і графіку навчального процесу

№ модуля	Матеріал лекцій	Кількість лекцій	Бали за вивчення лекційного матеріалу	Лабораторні роботи				Реферат 8б за 1	Науково дослідна робота	Максимальна сума балів
				Теми лабораторних робіт	Години	Тестовий контроль	Виконання л.р 5б за 1 л.р.			
1.	Процес розробки програмних систем. Підстави аналізу вимог. Встановлення вимог.	2	3	**№1** Планування розробки системи. Аналіз вимог і моделювання	2	14	5			22
2.	Принципи специфікації вимог. Системне проектування. Проектування баз даних.	2	4	**№2** Специфікації вимог. Прототипування і спільна розробка додатків **№3** Системне проектування. Проектування баз даних	2	14	10	8		36
3.	Проектування програмної системи. Тестування і керування змінами.	2	3	**№4** Проектування програмної системи	2	14	5		20	42
	Всього:	6	10		6	42	20	8	20	100

6. Розподіл балів за змістовими модулями для визначення оцінки за результатами вивчення навчальної дисципліни

Модулі	Оцінка		
	«3»	«4»	«5»
Змістовий модуль 1	13-17	18-21	22-25
Змістовий модуль 2	15-19	20-25	26-29
Змістовий модуль 3	32-36	37-41	42-46
Всього за семестр	**60 - 74**	**75 - 89**	**90 - 100**

Шкала оцінювання

За шкалою ECTS	За національною шкалою	За шкалою навчального закладу
A	відмінно	90-100
BC	добре	75-89
DE	задовільно	60-74
FX	незадовільно з можливістю повторного складання	35-59
F	незадовільно з обов’язковим повторним курсом	1-34

Лабораторна робота № 1

Тема: Планування розробки системи. Аналіз вимог і моделювання

Ціль: Придбання практичних навичок планування системи, проведення аналізу і моделювання

Короткі теоретичні відомості:

Планування розробки системи

Проекти з розробки інформаційних систем (ІС) повинні бути заздалегідь сплановані. ІС необхідно ідентифікувати, класифікувати, ранжувати і вибирати для первісної розробки, для вдосконалення або, можливо, для ліквідації. Питання полягає в наступному: "Які технології ІС і додатки принесуть найбільші ділові вигоди?" В ідеалі рішення, якому необхідно слідувати, повинне базуватися на бізнес-стратегії і ретельному і методичному плануванні.

Бізнес-стратегію можна визначити за допомогою різних процесів, відомих як стратегічне планування, бізнес моделювання, реінжиніринг бізнес-процесів, стратегічне погоджування, керування інформаційними ресурсами і т.п. Ми не ставимо метою пояснювати відмінності між названими підходами. Досить сказати, що всі ці підходи пов'язані з вивченням фундаментальних бізнес-процесів в організації, метою яких є визначення довгострокового ведення бізнесу з наступним призначенням пріоритетів різним проблемам ведення бізнесу, які можуть бути дозволені за допомогою інформаційної технології (ІТ).

Існує багато способів планування розробки систем. Один з традиційних підходів одержав назву SWOT (Strengths, Weaknesses, Opportunities, Threats - сильні сторони, слабкі сторони, сприятливі можливості, погрози). Ще одна популярна стратегія базується на моделі VCM (Value Chain Model - модель ланцюжків цінності). Більш сучасні варіанти підходів до розробки бізнес-стратегій відомі як BPR (Business Process Reengineering - реінжиніринг бізнес-процесів). Інформацію, необхідну для діяльності організації, також оцінюють з використанням так званих проектних шаблонів для ISA (Information System Architecture - архітектура інформаційних систем). Подібні проектні шаблони можна одержати за аналогією з

описових схем, що успішно зарекомендували себе в дисциплінах, відмінних від IT (наприклад, у будівельній промисловості).

Усі підходи до планування розробки систем мають один спільний знаменник - вони спрямовані скоріше на досягнення ефективності (робити те, що потрібно), ніж продуктивності (робити так, як потрібно).

Підхід SWOT

Підхід SWOT дозволяє ідентифікувати, класифікувати, ранжувати і вибирати проекти з розробки ІС таким чином, щоб вони були у'язані з сильними і слабкими сторонами організації, а також можливостями і погрозами. Це підхід зверху - униз, застосування якого починається з визначення місії організації.

Формулювання місії фіксує унікальний характер організації і визначає її бачення того, де вона хотіла б опинитися в майбутньому. Вірно сформульована місія відводить головне місце потребам клієнтів, а не товарам або послугам, які надає організація.

Формулювання місії і розроблювана на її основі бізнес-стратегія враховують те, якими внутрішніми сильними і слабкими сторонами володіє компанія в таких областях, як керування, виробництво, кадрове забезпечення, фінанси, маркетинг, дослідження і розробка і т.д. Ці сильні і слабкі сторони повинні бути детально обговорені і погоджені, після чого їм призначаються пріоритети. Процвітаючі організації здатні в будь-який момент ідентифікувати поточний набір сильних і слабких сторін, які направляють розробку їх бізнес-стратегій.

Підхід VCM

Метод ціннісних ланцюжків - VCM - дозволяє оцінити конкурентні переваги за допомогою аналізу всього ланцюжка видів діяльності в організації, починаючи від одержання сировини до кінцевої продукції, що продається і доставляється споживачам. Метафора ланцюжка підсилює те положення, що єдина слабка ланка приводить до розриву всього ланцюга.

Модель служить меті з'ясування того, яка конфігурація ланцюжка цінності

обіцяє найбільші конкурентні переваги. Проекти по розробці ІС можуть з часом указати на ті сегменти, операції, канали розподілу, маркетингові підходи і т.д., які дозволяють завоювати найбільшу конкурентну перевагу.

У первісному варіанті методу VCM організаційні функції були розділені на основні види діяльності і допоміжні види діяльності. Основні види діяльності створюють або додають цінність кінцевому продукту.

Вони розділяються на п'ять послідовних етапів:

1. вхідне постачання;
2. обробка;
3. вихідне постачання;
4. збут і маркетинг;
5. обслуговування.

Підтримуючі види діяльності не додають цінності, принаймні, прямо. Вони також відіграють істотну роль, але не "збагачують" продукт.

Підтримуючі види діяльності включають:

1. адміністрацію і інфраструктуру;
2. керування кадрами;
3. дослідження;
4. розробку;
5. розробку ІС.

Хоча модель VCM являє собою корисний інструмент для планування розробки ІС, всюдисуща комп'ютеризація може сприяти змінам у способах ведення бізнесу, що, у свою чергу, створює конкурентні переваги. Інакше кажучи, ІТ може перетворити ціннісні ланцюжки організації. Таким чином, між ІТ і моделлю VCM може бути встановлений позитивний зворотний зв'язок.

Портер (Porter) і Міллар (Millar) виділяють п'ять кроків, які повинна зробити організація, щоб скористатися перевагами, наданими ІТ.

1. Оцінити інформаційну ємність продуктів і процесів.

2. Оцінити роль IT у галузевій структурі.

3. Виявити і ранжувати способи, за допомогою яких IT створює конкурентна перевага.

4. Розглянути, яким чином IT може створити новий напрямок у бізнесі.

5. Розробити план, спрямований на вилучення вигод від використання IT.

Підхід BPR

Підхід до планування розробки ІС з використанням методів реінжинірингу бізнес-процесів (BPR-методів) заснований на допущенні, що сучасні організації повинні реконструювати себе і відмовитися від функціональної декомпозиції, ієрархічних структур і принципів пріоритетності повсякденних потреб, які вони сьогодні використовують. Розглянута концепція була введена в 1990 році Хаммером (Hammer) і Девенпортом (Davenport), і Шортом (Short) і відразу звернула на себе увага, викликавши полеміку.

Сьогодні більшість організацій мають структуру з вертикальним підпорядкуванням підрозділів, зосереджених на функціях, товарах або регіонах. Ці структури і методи роботи прослідковуються аж до вісімнадцятого століття і беруть свій початок у принципі поділу праці і наступної фрагментації роботи, сформульованому ще Адамом Смитом. Ніхто з працівників або підрозділів не відповідає за бізнес-процес, який визначається як "...сукупність видів діяльності, що одержують на вході один або кілька типів ресурсів, що і створюють на виході продукцію, що представляє цінність для споживача".

Методологія реінжинірингу бізнес-процесів ставить під сумнів індустріальні принципи Смита поділу праці, ієрархічного керування і економії на масштабах. У сучасному світі організація повинна бути здатна адаптуватися до швидких змін ринку, новим технологіям, конкурентним факторам, вимогам споживачів і т.д.

Жорсткі організаційні структури, у яких бізнес-процеси розірвані між багатьма підрозділами, застаріли. Організації повинні зосереджуватися на бізнес-процесах, а не на окремих завданнях, фахівцях і функціях підрозділів. Ці процеси розділені по горизонталі між видами діяльності і завершуються в точках контакту зі споживачами. "Найбільш видима відмінність між підприємством, орієнтованим на

бізнес-процеси, і традиційною організацією полягає в існуванні в кожного з процесів "хазяїна".

Основна мета реінжинірингу бізнес-процесів полягає в радикальній реконструкції бізнес-процесів в організації (тому підхід BPR найчастіше називається реконструкцією або перепроектуванням процесів). Бізнес-процеси необхідно ідентифікувати, добре налагодити і вдосконалити. Поведінка процесів фіксується у вигляді діаграм потоків робіт (workflow diagrams) і вивчається в рамках дисципліни "Аналіз потоків робіт". Потоки робіт охоплюють потоки подій, документів і інформації в рамках бізнес-процесів і можуть використовуватися для підрахунку часу, ресурсів і фінансових коштів, необхідних для цих видів діяльності.

Основна перешкода на шляху реалізації BPR-підходу в організації лежить у необхідності впровадження горизонтального процесу в традиційну вертикальну структуру керування. Серйозна ініціатива по впровадженню BPR-підходу вимагає перемикання організації на проектні бригади як основні організаційні одиниці. Ці бригади відповідають за один або більше наскрізних бізнес-процесів.

Підхід ISA

На відміну від уже описаних підходів, підхід з використанням уніфікованої схеми для архітектури інформаційних систем (Information Systems Architecture - ISA) заснований на проектуванні знизу-вгору. Цей підхід пропонує нейтральну архітектурну концептуальну схему для проектних рішень по створенню ІС, яка може підходити для різних бізнес-стратегій. По суті, підхід з використанням ISA не включає методології планування розробки систем. Він просто пропонує схему, яка може служити в якості важеля для досягнення цілей більшості бізнес-стратегій.

Схема ISA являє собою таблицю з тридцяти гнізд, організованих у вигляді п'яти рядків (позначених від 1 до 5) і шести стовпців (позначених від A до F). Рядки представляють різні точки зору (або ракурси), використовувані при конструюванні складних інженерних продуктів, таких як інформаційні системи.

Ці точки зору належать п'яти основним "гравцям" - п'ятьом учасникам розробки ІС.

1. Планувальник (визначає границі системи).

2. Власник (визначає концептуальну модель підприємства).
3. Проектувальник (задає фізичну модель системи).
4. Конструктор (забезпечує деталізоване технологічне рішення).
5. Субпідрядник (поставляє компоненти системи).

Шість стовпців представляють шість різних описів або архітектурних моделей, з якими "відіграє" кожний учасник. Подібно точкам зору описи значно відрізняються між собою, але в той же час вони внутрішньо зв'язані між собою. Описи покликані дати відповідь на шість питань щодо модельованих сутностей, якими задається кожний з учасників.

1. Що становить сутність (тобто, у випадку ІС, дані).
2. Як сутність функціонує (тобто бізнес-процеси).
3. Де сутність розташована (тобто розташування оброблювальних компонентів).
4. Хто працює з сутністю (тобто користувачі).
5. Коли з сутністю щось відбувається (тобто розподілу подій і станів у часі).
6. Чому сутність існує (тобто мотивація підприємства).

Комбінація точок зору і описів, представлених у тридцяти комірках таблиці Захмана, являє собою потужну систематизацію, на основі якої можна вибудувати повну архітектуру для розробки ІС. Розташовані по вертикалі ракурси можуть відрізнятися ступенем деталізації, але, що більш важливо, вони відрізняються по суті і використовують різні представлення моделі. Різні моделі відображають різні погляди учасників. Аналогічно розташовані по горизонталі описи підготовлені, виходячи з різних міркувань. Кожне з цих описів покликано відповісти на один з шести питань.

Найбільш привабливою стороною підходу ISA є те, що він пропонує схему, яка, цілком імовірно, може виявитися досить гнучкої для адаптації до майбутніх змін в умовах ведення бізнесу і у ресурсах, якими володіє підприємство. Це є наслідком того, що рішення на основі ISA не базуються на якій-небудь певній бізнес-стратегії. Це просто схема для повного опису ІС. Сама схема отримана на

підставі досвіду, накопиченого більш устояними дисциплінами, деякі з них нараховують тисячі років (наприклад, класична архітектура).

Системи для трьох рівнів керування

Планування розробки системи пов'язане з представленням про наявність в організації трьох рівнів керування.

1. Стратегічного.
2. Тактичного.
3. Оперативного.

Ці три рівні організаційного керування характеризуються власним рівнем прийнятих рішень, різним набором необхідних додатків ІС і специфічними вимогами до підтримки з боку ІТ-підрозділів. Одне з завдань, що виникають при плануванні розробки системи, полягає у визначенні комплексу додатків ІС і ІТ-рішень, який є найбільш ефективним для організації в певний момент часу. У табл. 1.1 визначені пов'язані з встановленням відповідності додатків ІС і ІТ-рішень рівню прийнятих рішень.

Таблиця 1.1 Підтримка різних рівнів прийняття рішень з боку ІС і ІТ

Рівень прийняття рішень	**Спрямованість прийнятих рішень**	**Типові додатки ІС**	**Типові ІТ-рішення**
Стратегічний	Стратегії, що підтримують довгострокові цілі організації	Аналіз ринку і збуту; планування розробки товарів; оцінка ефективності	Видобуток даних; керування знаннями

Тактичний	Стратегії, що підтримують короткострокові завдання і розподіл ресурсів	Аналіз бюджету; прогнозування фонду зарплати; планування запасів; обслуговування споживачів	Сховища даних; аналіз даних; електронні таблиці
Оперативний	Підтримка повсякденної діяльності персоналу і виробництво	Виплата платні; виписка рахунків; закупівлі; бухгалтерський облік	Бази даних; обробка транзакцій; генератори додатків

Реалізовані ІС-додатки і ІТ-рішення, які здатні забезпечити найбільші вигоди організації, відносяться до стратегічного рівня. Однак ці рішення найбільш важко реалізувати - вони використовують "прикордонні" технології і вимагають дуже кваліфікованого і спеціалізованого проектування. Проте, саме ці системи здатні забезпечити організації конкурентну перевагу на ринку.

На протилежному кінці шкали розташовуються системи, що підтримують оперативний рівень керування. Ці системи відрізняються одноманітністю дій і процедур, використовують традиційні технології баз даних і найчастіше збираються з готових до застосування пакетних програмних рішень. Дані системи безперспективні з погляду забезпечення конкурентної переваги, однак без них організація не здатна функціонувати належним чином.

Будь-яка сучасна організація має у своєму розпорядженні повний комплект систем оперативного керування, але тільки організації, які досяглися більших висот у мистецтві керування, мають інтегрований набір додатків ІС стратегічного рівня. Основна технологія, яка використовується для зберігання і обробки даних у процесі прийняття високорівневих стратегічних і тактичних рішень, відома як технологія сховищ даних.

Аналіз об'єктів

Моделювання аналізу

На самому загальному рівні можна виділити три типи UML-моделей - кожна зі своїм власним набором діаграм і пов'язаних з ними конструкцій.

1. Модель станів (state model), яка представляє статичний погляд на систему, - це модель вимог до даних. Модель стану представляє структури даних і відношення на них. Основний метод візуалізації моделі станів полягає у використанні діаграми класів.

2. Модель поведінки (behavior model), яка представляє операційний погляд на систему, - це модель функціональних вимог. Модель поведінки представляє бізнес-транзакції, операції і алгоритми над даними. Для візуалізації моделі поведінки існує кілька способів - діаграми прецедентів, діаграми послідовностей, діаграми кооперації і діаграми видів діяльності.

3. Модель зміни станів (state change model), яка представляє динамічний погляд на систему, - це модель еволюції об'єктів з часом. Модель зміни станів представляє можливі зміни станів об'єкта (де під станом розуміються поточні значення атрибутів і асоціативних зв'язків з іншими об'єктами). Основний метод візуалізації моделі зміни станів полягає у використанні діаграми станів.

Моделювання прецедентів

Поведінка системи - це її реакція у відповідь на зовнішні події. У мові UML зовні спостережувана поведінка, що і допускає тестування, фіксується у вигляді прецедентів. Прецедент (use case) виконує бізнес-функцію, яку може спостерігати зовнішній суб'єкт і яка може бути з часом окремо протестована в процесі розробки.

Суб'єкт (actor) - це хтось або щось (людина, машина і т.д.), взаємодіюче з прецедентом. Суб'єкт взаємодіє з прецедентом, очікуючи одержати якийсь корисний результат.

Діаграма прецедентів - це наочне представлення суб'єктів і прецедентів разом

з будь-якими додатковими визначеннями і специфікаціями. Діаграма прецедентів являє собою не просто якусь схему, а є повністю документованою моделлю передбачуваної поведінки системи. Таке ж розуміння застосовується у відношенні інших діаграм мови UML. Якщо тільки не обумовлено протилежне, то UML-діаграма використовується як синонім UML-моделі.

Суб'єкти

Суб'єкти і прецеденти визначаються в результаті аналізу функціональних вимог. Функціональні вимоги втілюються в прецедентах. Прецеденти задовольняють функціональні вимоги за рахунок надання суб'єкту корисного результату. При цьому не має значення, у якій послідовності вирішує бізнес-аналітик свої завдання: спочатку позначає суб'єктів, а потім прецеденти, або навпаки. У нашому наставлянні спочатку вибираються суб'єкти.

Типовим графічним зображенням суб'єкта є "штриховий чоловічок" (див. нижче). У загальному випадку суб'єкт може бути показаний у вигляді прямокутного символу класу. Подібно звичайному класу суб'єкт може володіти атрибутами і операціями (пов'язаними з подіями, повідомлення про яких він відправляє і одержує).

На рис. 1.1 показано три суб'єкти, які явно представлені в специфікації. Це суб'єкти Customer (Клієнт), Salesperson (Продавець) і Warehouse (Склад).

Рис. 1.1. Суб'єкти (Internet-магазин)

Наставляння з аналізу: (Internet-магазин)

Розширені вимоги для встановлення суб'єктів у додатку lntemet-магазин.

1. Для знайомства зі стандартною конфігурацією сервера, що обирається, настільного або портативного комп'ютера клієнт використовує Web-сторінку

Internet-магазину. При цьому також приводиться ціна конфігурації.

2. Клієнт вибирає деталі конфігурації, з якими він прагне познайомитися, можливо, з наміром купити готову або скласти більш підходящу конфігурацію. Ціна для кожної конфігурації може бути підрахована на вимогу користувача.

3. Клієнт може вибрати варіант замовлення комп'ютера по Internet або попросити, щоб продавець зв'язався з ним для пояснення деталей замовлення, домовився про ціну і т.п. перш, ніж замовлення буде фактично розміщене.

4. Для розміщення замовлення клієнт повинен заповнити електронну форму з адресами для доставки товару і відправлення рахунок-фактури, а також деталями, що стосуються оплати (кредитна картка або чек).

5. Після введення замовлення клієнта в систему продавець відправляє на склад електронна вимога, що містить деталі замовленої конфігурації.

6. Деталі угоди, включаючи номер замовлення, номер рахунку клієнта, відправляються по електронній пошті клієнту, так що замовник може перевірити стан замовлення через Internet.

7. Склад одержує рахунок-фактуру від продавця і відвантажує комп'ютер клієнту.

Прецеденти

Прецедент (use case) являє собою якийсь цілісний набір функцій, що мають певну цінність для суб'єкта. Суб'єкт, який не спілкується з прецедентом, не має сенсу, однак зворотне твердження не завжди вірне (тобто прецедент, який не спілкується з суб'єктом - річ допустима). Можуть існувати деякі прецеденти, які узагальнюють або уточнюють основний прецедент і не взаємодіють безпосередньо з суб'єктами. Вони використовуються як внутрішні в моделі прецедентів і допомагають основному прецеденту виробити результат, надаваний суб'єкту.

Прецеденти можна вивести в результаті ідентифікації завдань для суб'єкта. Для цього слід задатися питанням: "Які обов'язки суб'єкта відносно системи і чого він очікує від системи?" Прецеденти також можна визначити в результаті безпосереднього аналізу функціональних вимог. У багатьох випадках функціональна вимога відображається безпосередньо в прецедент.

У табл. 1.2 функціональні вимоги, (*Суб'єкти* / Наставляння по аналізу: (Internet-магазин)), розподілені по суб'єктам і прецедентам. Складські завдання збірки конфігурації комп'ютера і відправлення його клієнту відносяться до функцій, які в цьому випадку виходять за рамки додатка.

Таблиця 2.1 Розподіл вимог по суб'єктах і прецедентам

№ п/п	**Вимога**	**Суб'єкт**	**Прецедент**
1.	Для знайомства зі стандартною конфігурацією сервера, що обирається, настільного або портативного комп'ютера клієнт використовує Web-сторінку Internet-магазину. При цьому також приводиться ціна конфігурації	Customer (Клієнт)	Display Standard Computer Configuration *(Відображення стандарт-ний конфігурації комп'ютера)*
2.	Клієнт вибирає деталі конфігурації, з якими він прагне познайомитися, можливо, з наміром купити готову або скласти більш підходящу конфігурацію. Ціна для кожної конфігурації може бути підрахована на вимогу користувача	Customer	Build Computer Configuration *(Складання конфігурації комп'ютера)*

3.	Клієнт може вибрати варіант замовлення комп'ютера по Internet або попросити, щоб продавець зв'язався з ним для пояснення деталей замовлення, домовився про ціну і т.п. перш, ніж замовлення буде фактично розміщене	Customer, Salesperson (Продавець)	Order Configured Computer *(Замовлення конфігурованого комп'ютера).* Request Salesperson Contact *(Звернення з проханням до продавця)*
4.	Для розміщення замовлення клієнт повинен заповнити електронну форму з адресами для доставки товару і відправлення рахунок -фактури, а також деталями, що стосуються оплати (кредитна картка або чек)	Customer	Order Configured Computer,Verify and Accept Customer Payment *(Перевірка і прийняття платежу від клієнта)*
5.	Після введення замовлення клієнта в систему продавець відправляє на склад електронну вимогу, що містить деталі, що стосуються замовленої конфігурації	Salesperson, Warehouse (Склад)	Inform Warehouse About Order *(Інформування складу про замовлення)*
6.	Деталі угоди, включаючи номер замовлення, номер рахунку клієнта, відправляються по електронній пошті клієнту, так що замовник може перевірити стан замовлення через Internet	Salesperson, Customer	Order Configured Computer, Update Order States *(Відновлення стану замовлення)*
7.	Склад отримує рахунок-фактуру від продавця, і відправляє комп'ютер клієнту	Salesperson, Warehouse	Print Invoice *(Друк рахунок-фактури)*

На рис. 1.2 показане графічне позначення прецедентів. Прецедент зображується у вигляді еліпса, усередині або нижче якого міститься ім'я

прецеденту. Можна використовувати також інші представлення, включаючи позначення класу у вигляді прямокутника, усередині якого можна привести перелік усіх необхідних атрибутів і операцій прецеденту.

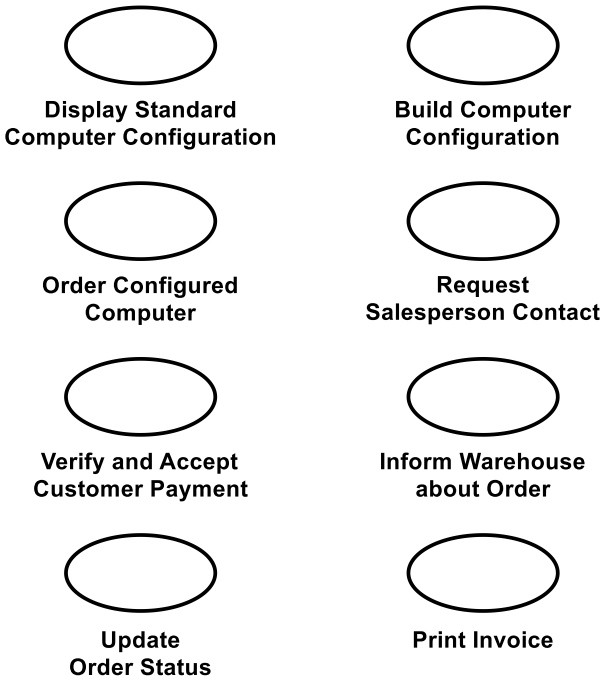

Рис. 1.2 Прецеденти (Internet-магазин)

Діаграми прецедентів

Діаграма прецедентів приписує прецеденти до суб'єктів. Вона також дозволяє користувачу встановити відношення між прецедентами, зазвичай, якщо такі відношення існують.

Діаграми прецедентів - основний метод візуалізації для моделі поведінки системи. Щоб представити повну модель прецедентів, необхідно більш докладний опис елементів діаграми (прецедентів і суб'єктів).

Документування прецедентів

Кожний прецедент повинен бути описаний за допомогою документально зафіксованого потоку подій (flow of events). Відповідний текстовий документ визначає, що повинна робити система, коли суб'єкт ініціює прецедент. Структура

документа, що описує прецеденти, може варіюватися, однак типовий опис повинен містити наступні розділи.

1. Короткий опис.
2. Суб'єкти, що беруть участь.
3. Предумови, необхідні для ініціювання прецеденту.
4. Деталізований опис потоку подій, який включає:
   - основний потік, який можна розбити для того, щоб показати підлеглі потоки подій (підлеглі потоки можуть бути розділені далі на ще більш дрібні потоки, з метою поліпшити удобочитаємість документа);
   - альтернативні потоки для визначення виняткових ситуацій.
5. Постумови, що визначають стан системи, по досягненню якого прецедент завершується.

Документ, що містить описи прецеденту, розвивається по ходу розробки. На ранній стадії визначення вимог складається тільки короткий опис. Інші частини документа створюються поступово і ітеративно. Повний документ виникає наприкінці етапу специфікації вимог. На цій стадії документ може бути доповнений прототипами GUI-екранів. Пізніше документ по прецедентах використовується для створення користувацької документації для реалізованої системи.

Моделювання видів діяльності

Модель видів діяльності (activity model) може представляти в графічній формі потік подій для прецеденту. Цей тип моделі був введений тільки в більш пізні версії UML і дозволив подолати розрив між високорівневим представленням поведінки системи за допомогою моделей прецедентів і набагато більш низьким рівнем представлення поведінки за допомогою моделей взаємодій (діаграм послідовностей і діаграм кооперації).

Діаграма видів діяльності показує кроки обчислення. Кожний крок відповідає стану (state), у якому що-небудь виконується. Тому кроки виконання називаються станами виду діяльності. Діаграма описує, які кроки виконуються послідовно, а

які - паралельно. Передача керування від одного стану виду діяльності до іншого називається переходом (transition).

Після завершення документа з описом прецеденту стану виду діяльності можна встановити по опису основного і альтернативних потоків. Однак, між описом прецедентів і моделлю видів діяльності існує важлива відмінність. Опис прецеденту створюється з погляду зовнішнього суб'єкта. Модель видів діяльності відображає внутрішньосистемну точку зору.

Моделі видів діяльності можуть знаходити і інше застосування при розробці систем крім моделювання прецедентів. Вони можуть використовуватися для аналізу бізнес-процесів на високому рівні абстракції до виробітку прецедентів. І навпаки, їх можна використовувати на більш низькому рівні абстракції для розробки складних послідовних алгоритмів або засобів розпаралелювання в багатопоточних додатках.

Види діяльності

Якщо моделювання видів діяльності використовується для візуалізації послідовності видів діяльності, пов'язаних з прецедентом, то стану виду діяльності можна встановити на основі документа опису прецеденту. Як було відзначено вище, імена діям слід привласнювати, виходячи з системних міркувань, а не з погляду суб'єкта.

Стан виду діяльності представляється в UML у вигляді прямокутника з закругленими кутами. Слід відразу уточнити, що той самий графічний символ використовується для візуалізації стану виду діяльності (activity state) і стану дії (action state). Відмінність між діяльністю і дією реакцією полягає в їхньому тимчасовому масштабі. Для здійснення діяльності потрібний визначений час; дія ж завершується настільки швидко, що в масштабах нашої тимчасової шкали - може вважатися, що відбуваються миттєво. (Отже, у моделі станів) види діяльності можуть бути визначені тільки в рамках стану об'єкта, а дії можуть з'являтися також при переході між станами об'єкта.

У табл. 1.3 викладені події, що відносяться до основного і альтернативних потоків, запозичених з документа опису прецеденту, а також позначені стану видів діяльності. Зверніть увагу, що ім'я кожному виду діяльності привласнене, виходячи

з системної точки зору, а не з погляду суб'єкта.

Таблиця 1.3 Встановлення дій в основному і в альтернативних потоках

№ п/п	*Формулювання прецеденту*	*Стан виду діяльності*
1.	Початок прецеденту збігається з рішенням клієнта замовити конфігурований комп'ютер за допомогою вибору функції Continue (або аналогічної функції) при відображенні на екрані деталізованої інформації, що відноситься до замовлення	Display Current Configuration *(Відображення поточної конфігурації);* Get Order Request *(Одержання запиту на замовлення)*
2.	Система просить клієнта ввести деталізовану інформацію про покупку, у тому числі: - ім'я продавця (якщо воно відомо); - деталі, що стосуються доставки (ім'я і адреса клієнта); - детальну інформацію з оплати (якщо вона відрізняється від інформації по доставці); - спосіб оплати (кредитна картка або чек) і довільні коментарі	Display Purchase Form *(Відображення закупівельної форми)*
3.	Клієнт вибирає функцію Purchase (або аналогічну функцію) для відправлення замовлення виробнику	Стан виду діяльності Get Purchase Details *(Деталізувати інформацію про покупку)*

4.	Система привласнює унікальний номер замовлення і клієнтський обліковий номер замовленню на покупку і запам'ятовує інформацію про замовлення в базі даних	Store Order *(Запам'ятати замовлення)*
5.	Система відправляє клієнту по електронній пошті номер замовлення і клієнтський номер клієнту разом з усіма деталями, що відносяться до замовлення, у якості підтвердження прийняття замовлення	Email Order Details *(Відправити детальну інформацію з замовлення)*
6.	Клієнт ініціює функцію Purchase до того, як введе всю обов'язкову інформацію. Система відображає на екрані повідомлення про помилку і просить ввести пропущену інформацію	Get Purchase Details; Display Purchase Form
7.	Клієнт вибирає функцію Reset (або аналогічну) для того, щоб повернутися до вихідної форми замовлення на покупку. Система дає можливість клієнту знову ввести інформацію	Display Purchase Form

Діаграма видів діяльності

Діаграма видів діяльності (activity diagram) показує переходи між видами діяльності. Якщо вид діяльності не являє собою замкненого циклу, то діаграма містить початковий стан виду діяльності і один або більше кінцевих станів виду діяльності. Повністю зафарбована окружність представляє початковий стан. Кінцевий стан зображується у вигляді окружності з зафарбованою центральною частиною (автор образно називає цей символ "бичачим оком").

Переходи можуть розгалужуватися за умовою і поєднуватися. У результаті виникають альтернативні (alternative) обчислювальні потоки (thread). Умова

розгалуження позначається ромбом.

Переходи можуть також розділятися і зливатися. У результаті виникають паралельні (які одночасно виконуються) (concurrent) потоки. Розпаралелювання і возз'єднання переходів представляється у вигляді жирної лінії або смуги. Помітимо, що діаграма виду діяльності, у якій відсутні паралельні процеси, схожа на звичайну блок-схему. (Поведінка з паралелізмом у наставлянні не використовується).

Моделювання класів

Систему утворює системний стан. Стан є функцією вмісту системної інформації в заданий момент часу - це функція системного поточного набору об'єктів-екземплярів.

Визначення внутрішнього стану системи дається в моделі класів (class model). До елементів, що брав участь у моделюванні класів, відносяться самі класи, атрибути і операції класів, асоціації, агрегації і композиції, а також узагальнення. Діаграма класів (class diagram) дає узагальнене візуальне представлення про всі ці елементи моделі.

Хоча в наставлянні моделювання класів розглядається після моделювання прецедентів, на практиці ці два види діяльності проводяться в паралель. Дві моделі взаємно доповнюють одна іншу допоміжною інформацією. Прецеденти полегшують вибір класів, і навпаки - моделі класів можуть допомогти виявити упущені прецеденти.

Класи

Класи для визначення бізнес-об'єктів відносяться до довгоживучим (постійним або перманентним) сутностям бізнес-процесів, таким як Order (Замовлення), Shipment (Доставка), Customer (Клієнт), Student (Студент) і т.д. Це класи, які визначають модель бази даних для прикладної області. з цієї причини подібні класи часто називають класами-сутностями (entity class) або модельними класами. Вони представляють постійно збережені об'єкти бази даних.

Класи-сутності визначають суть будь-якої інформаційної системи. Аналіз

вимог спрямований переважно на виявлення класів-сутностей. Однак, для функціонування системи потрібні також класи іншого типу. Користувачам системи необхідні класи, які визначають GUI-об'єкти (наприклад, такі як екранні форми), називаємі прикордонними класами (boundary classes) (класами представлення (view classes)). Щоб функціонувати належним чином, системі також необхідні класи, які управляють програмною логікою - керуючі класи (control classes)

Залежно від конкретного підходу до моделювання прикордонні і керуючі класи можуть розглядатися або не розглядатися на деякому рівні представлення в ході аналізу вимог. Моделювання класів цього типу може бути відкладене до етапу проектування системи.

Виділення класів являє собою ітеративне завдання, і первісний перелік передбачуваних класів, як правило, зазнає змін.

При визначенні того, чи є поняття, присутні у вимогах, шуканими класами, можуть допомогти відповіді на деякі питання. Ось ці питання.

1. Чи є поняття "вмістиліщем" даних?
2. Чи володіє воно окремими атрибутами, здатними приймати різні значення?
3. Чи можна створити для нього множину об'єктів-екземплярів?
4. Чи входить воно в границі прикладної області?

Діаграма класів

Діаграма класів, так сказати «серце» і «душа» об'єктно-орієнтованої системи. У даному напрямку ставиться мета тільки продемонструвати можливості статичного моделювання відносно до моделі класів.

У класи поки що не введено ні однієї нової операції. Операції належать скоріше до сфери проектування, ніж аналізу. Після того, як операції в остаточному підсумку вводяться в класи, модель класів у явному вигляді визначає поведінку системи.

На рис. 1.3 показана діаграма класів для додатка Internet-магазин.

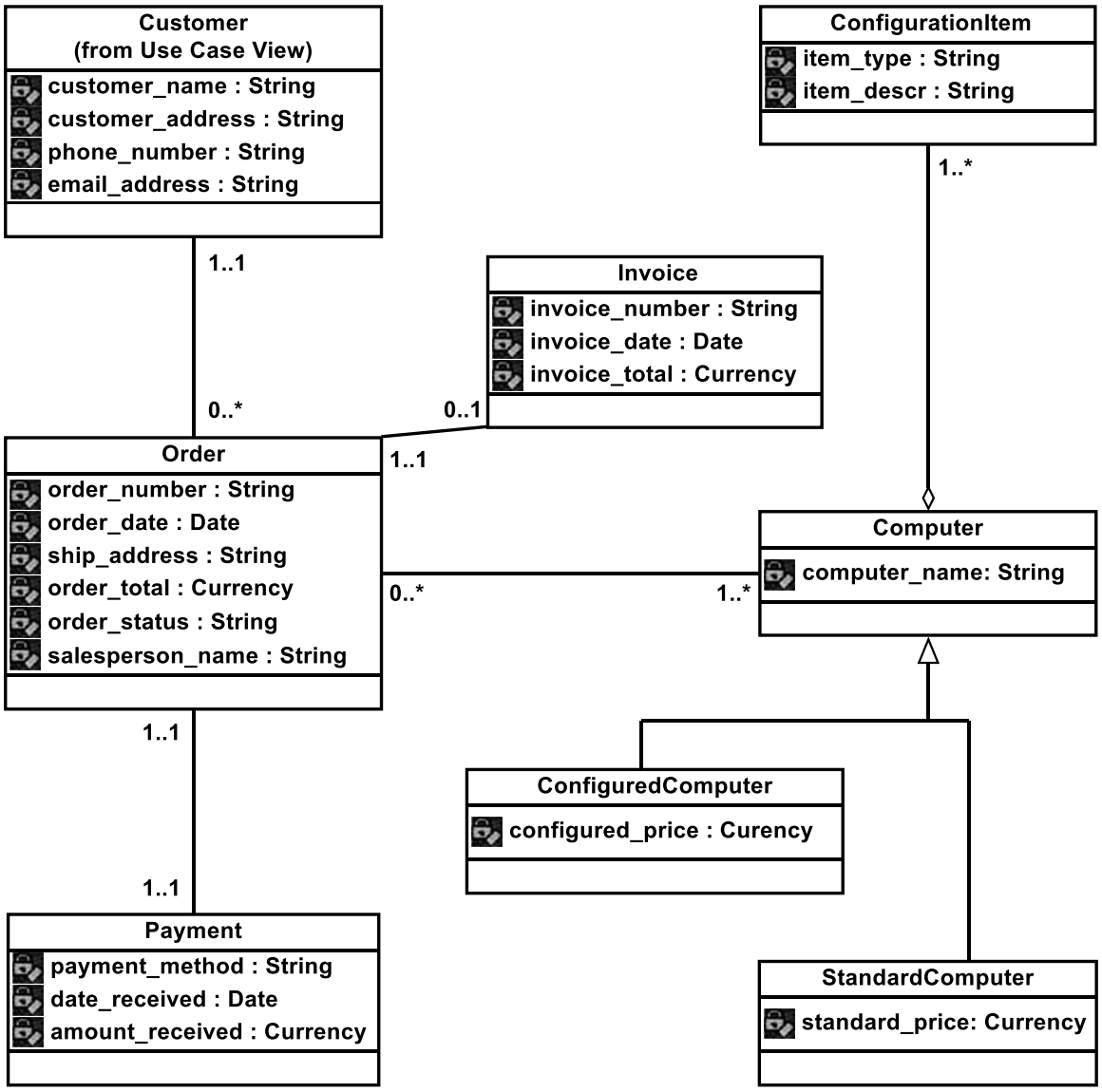

Рис. 1.3. Діаграма класів (Internet-магазин).

Моделювання взаємодій

Моделювання взаємодій (interaction modeling) охоплює питання взаємодії між об'єктами, необхідними для виконання прецеденту. Моделі взаємодії використовуються на більш розвинених стадіях аналізу вимог, коли стає відомою модель класів, так що посилання на об'єкти опираються на модель класів.

Наведене вище спостереження служить опорою для встановлення основної відмінності між моделюванням видів діяльності і моделюванням взаємодій. Моделі обох типів описують поведінку одного прецеденту (як правило). Однак, моделювання видів діяльності здійснюється на більш високому рівні абстракції воно

відображає послідовність подій поза зв'язком подій з об'єктами. Моделювання взаємодій відображає послідовність подій (повідомлень) у їхньому зв'язку з діючими в кооперації об'єктами.

Діаграми взаємодій розділяються на два види - діаграми послідовностей і діаграми кооперації. Вони можуть використовуватися як взаємозамінні, і, звичайно, багато CASE-засобів підтримують автоматичне перетворення однієї моделі в іншу. Різниця між моделями полягає в акцентах. Моделі послідовностей концентруються на тимчасових послідовностях подій, а в моделях кооперації основна увага приділяється відношенням між об'єктами.

Взаємодії

Взаємодія (interaction) являє собою набір повідомлень, властивих поведінці деякої системи, якими обмінюються об'єкти та відповідності з встановленими між ними зв'язками (останні можуть бути постійними або тимчасовими). Діаграма послідовностей представляється двовимірним графом. Об'єкти розташовуються по горизонталі. Послідовності повідомлень розташовуються зверху вниз по вертикалі. Кожна вертикальна лінія називається лінією життя (lifeline) об'єкта.

Стрілки представляють кожне повідомлення, що направляється від об'єкта, що викликається (відправника) до операції (методу) об'єкта, що викликається (одержувача). Для кожного повідомлення, як мінімум, вказується його ім'я. Крім того, для повідомлення можуть бути зазначені фактичні аргументи повідомлення і інша керуюча інформація. Фактичні аргументи відповідають формальним аргументам методу об'єкта-одержувача.

Фактичні аргументи можуть бути вхідними аргументами (передаються від відправника до одержувача) або вихідними аргументами (передаються від одержувача назад до відправника). Вхідні аргументи можуть бути позначені ключовим словом in (якщо ключове слово відсутнє, то передбачається, що аргумент - вхідний). Вихідні аргументи позначаються ключовим словом out. Допускаються також аргументи типу inout ("вхідні-вихідні"), однак, для об'єктно-орієнтованого підходу вони не характерні. Повідомлення getcoursename, відправлене об'єкту, позначеному змінної `crs_ref`, має один вихідний аргумент і жодного вхідного:

```
crs_ref.getcoursename(out crs_name)
```

Показувати на діаграмі повернення керування від об'єкта-одержувача об'єкту-відправнику не обов'язково. Стрілка, що вказує на об'єкт-одержувач, припускає автоматичне повернення керування відправнику. Одержувач знає унікальний ідентифікатор об'єкта (OID) відправника.

Повідомлення може бути відправлене колекції (collection) об'єктів (колекція може бути набором, списком, масивом об'єктів і т.д.). Досить частою є ситуація, коли викликаючий об'єкт пов'язаний, з декількома об'єктами-одержувачами (оскільки кратність асоціації зазначена як "один до багатьох" або "багато до багатьох"). Ітеративний маркер - зірочка перед позначенням повідомлення - вказує на процес ітерації повідомлення по всій колекції.

Завдання:

Виконати планування розробки системи. Виконати аналіз вимог і моделювання.

Надати звіт, що містить результати проведеного планування і результати моделювання системи.

Контрольні питання:

1. Реінжиніринг бізнес - процесів (BPR) проводить ясну відмінність між бізнес - процесом і бізнес - функцією. У чому полягає ця відмінність? Приведіть приклад бізнес - процесу, який розірваний по горизонталі по всій організації.
2. Чому розуміння методу ISA (архітектура інформаційної системи) важливо для системної розробки?
3. Поясніть різницю між етапами визначення вимог і розробки специфікації.
4. Поясніть взаємозв'язок двох етапів проектування (архітектурне проектування і деталізоване проектування) з першими двома етапами життєвого циклу - етапом визначення вимог і етапом розробки специфікації.
5. Які основні причини зсуву від структурного підходу до проектування объектно - орієнтованому?

Лабораторна робота № 2

Тема: Специфікації вимог. Прототипування і розробка додатків

Ціль: Придбання практичних навичок проведення специфікації вимог, прототипування і розробка додатків

Короткі теоретичні відомості:

У порівнянні з іншими етапами розробки системи встановлення вимог у найменшому ступені стосується технічних сторін проекту. У дійсності мова йде про успішне рішення соціальних, комунікативних і управлінських питань. Недостатньо ретельне виконання цього етапу може привести до більш серйозних наслідків, ніж у випадку з іншими етапами. Лавиноподібне зростання витрат, спричинених не зафіксованими, упущеними або невірно понятими вимогами замовників, може з часом виявитися просто нестерпним для процесу розробки.

Принципи встановлення вимог

Встановлення вимог - перший етап життєвого циклу розробки системи. Система, що підлягає розробці, визначається за допомогою виду діяльності, який одержав назву системного планування. Ціль встановлення вимог полягає в тому, щоб дати розгорнуте визначення функціональних - а також нефункціональних - вимог, які учасники проекту очікують затвердити в системі що реалізовується і розгортається.

Вимоги визначають послуги, очікувані від системи (формулювання сервісів) і обмеження, яким система повинна підкорятися (формулювання обмежень). Ці формулювання сервісів можна об'єднати в кілька груп: одна з груп описує границі системи, друга - необхідні бізнес-функції (функціональні вимоги), а третя - необхідні структури даних (вимоги до даних). Формулювання обмежень можна класифікувати відповідно до різних категорій обмежень, що накладаються на систему, такі як необхідне "враження і відчуття від використання програми",

продуктивність, безпека і т.д.

Вимоги необхідно отримати від замовників (користувачів і власників системи). Цей вид діяльності, називаэмий виявленням вимог (requirements elicitation), здійснюється аналітиком бізнес-процесів (або системним аналітиком). Для цього підходять різні методи, починаючи з традиційних інтерв'ю з замовником і закінчуючи (при необхідності) побудовою програмного прототипу, за допомогою якого розкриваються додаткові вимоги.

Зібрані вимоги повинні бути піддані ретельному аналізу для виявлення в них повторів і протиріч. Це незмінно приводить до перегляду вимог і їх повторному узгодженню з замовниками.

Вимоги, задовільні з точки зору замовника, документуються. При цьому вимогам дають визначення, класифікують, нумерують і привласнюють їм пріоритети. Структура документа, що описує вимоги, відповідає шаблону (template), обраному в організації для цієї мети.

Хоча документ, що фіксує вимоги, носить значною мірою описовий характер, цілком можливо включити в нього високорівневу схематичну бізнес-модель. Як правило, бізнес-модель складається з моделі границь системи, моделі бізнес - прецедентів і моделі бізнес-класів.

Вимоги користувача подібні рухомій цілі. Щоб упоратися з мінливими вимогами, необхідно вміти управляти змінами. Керування вимогами включає такі види діяльності як оцінка впливу змін одних вимог на інші, а також на незмінну частину системи.

Виявлення вимог

Бізнес-аналітик виявляє вимоги до системи за допомогою консультацій. До участі в консультаціях залучаються замовники і експерти в проблемній області. У деяких випадках бізнес-аналітик має достатній досвід у проблемній області, і допомога експерта може не знадобитися. У цьому випадку Бізнес-аналітик являє собою різновид Експерта проблемної області, що відображено в моделі, показаної на рис. 2.1, за допомогою відношення узагальнення. (Слід, однак, мати на увазі, що рис.2.1 - це не модель прецедентів: нотація для прецедентів використовується тут

тільки для зручності.)

Вимоги, виявлені за допомогою експерта проблемної області, становлять основу знань про проблему області. Вони фіксують широко визнані, що не залежать від часу бізнес-правила, застосовні до більшості організацій і систем. Вимоги, виявлені в ході консультацій з замовниками, виражаються в сценаріях прецедентів. Вони виходять за рамки базових знань про проблемну область і фіксують унікальні риси організації - спосіб ведення бізнесу "тут і зараз" або побажання у відношенні того, як слід вести бізнес.

Завдання бізнес-аналітика полягає в тому, щоб об'єднати два набори вимог у бізнес-моделі. Як показано на рис. 2.1, Бізнес-модель містить Модель бізнес-класів і Модель бізнес-прецедентів. Модель бізнес-класів являє собою діаграму класів верхнього рівня, яка ідентифікує і зв'язує між собою бізнеси-об'єкти. Модель бізнес-прецедентів - це діаграма прецедентів верхнього рівня, яка ідентифікує основні функціональні будівельні блоки системи.

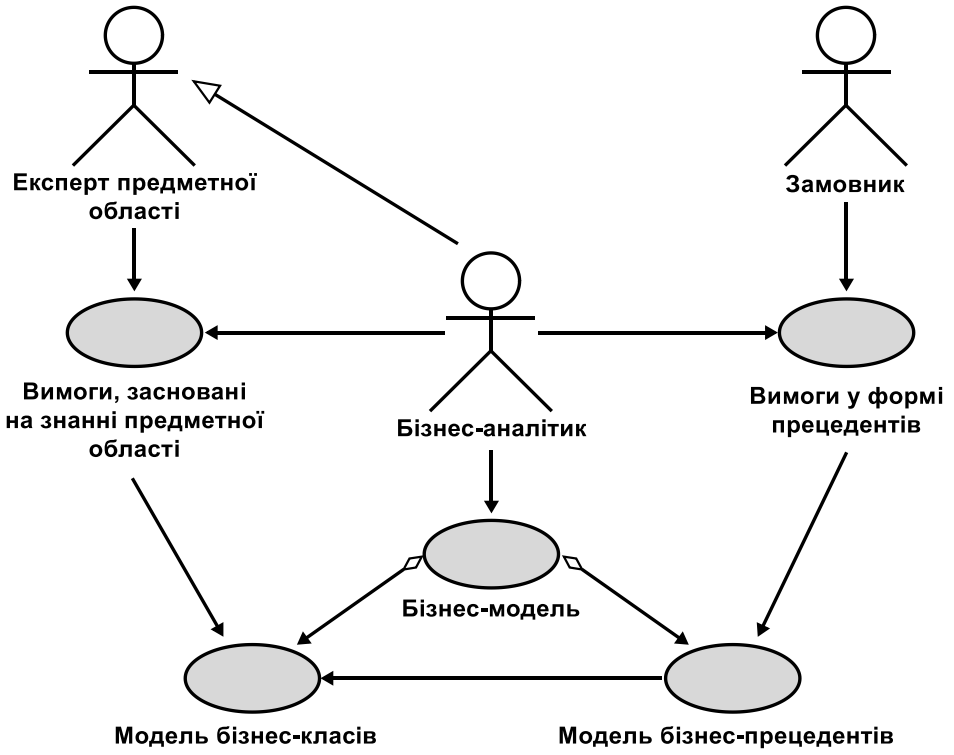

Рис. 2.1. Взаємні впливи моделей, характерні для процесу визначення вимог

У загальному випадку, класи проблемної області (бізнес-об'єкти) не повинні виводитися з прецедентів. Однак, на практиці правильність моделі бізнес-класів

повинна підтверджуватися за допомогою порівняння з Моделлю бізнес-прецедентів. Це порівняння, як правило, приводить до деякого настроювання або розширення Моделі бізнес-класів.

Прототипування

Прототипування (prototyping) - це найбільше часто використовуваний сучасний метод виявлення вимог. Програмні прототипи конструюються для візуалізації системи або її частини для замовників з метою одержання їх відкликань.

Прототип являє собою демонстраційну систему - "нашвидкуруч і грубо" зроблену робочу модель рішення, яка представляє користувацький GUI-інтерфейс і моделює поведінку системи при ініціюванні користувачем різних подій. Інформаційне наповнення екранів частіше жорстко запрограмоване в програмі прототипу, ніж виходить автоматично з бази даних.

Складність (і зростаючі "апетити" замовників) сучасних GUI-інтерфейсів роблять прототипування обов'язковим елементом розробки ПЗ. Прототипи дозволяють досить непогано оцінити реалізуємість і корисність системи до початку її реалізації.

У загальному випадку, прототип - це досить ефективний спосіб виявлення вимог, які важко одержати від замовника за допомогою інших засобів. Найчастіше така ситуація зустрічається для систем, які повинні надати в розпорядження користувачів нові бізнес-функції. Подібна ситуація також характерна для випадків суперечливих вимог і наявності проблем у кооперації між замовниками і розробниками.

Існують два основні різновиди прототипів.

- *"Одноразовий" прототип ("throw-away" prototype),* який після того, як виявлення вимог завершене, просто відкидається. Розробка "одноразового" прототипу націлена тільки на етап встановлення вимог ЖЦ ПЗ. Як правило, цей прототип концентрується на найменш зрозумілих вимогах.
- *Еволюційний прототип (evolutionary prototype),* який зберігається після

виявлення вимог і використовується для створення кінцевого програмного продукту. Еволюційний прототип націлений на прискорення поставки продукту. Як правило, він концентрується на ясно викладених вимогах, так що першу версію продукту можна надати замовнику досить швидко (хоча її функціональні можливості, як правило, неповні).

Додатковим доводом на користь використання саме "одноразового" прототипу може служити прагнення уникнути ризику "консервації" швидких і грубих і, як наслідок, неефективних рішень у кінцевому продукті. Однак міць і гнучкість сучасних засобів створення ПЗ послабляють цей довід. Якщо керування проектом здійснюється належним чином, то причини, по якій не можна було б позбутися неефективних запропонованих для прототипу рішень, не існує.

Системні сервіси

Основна частина документа опису вимог присвячена визначенню системних сервісів Ця частина може займати до половини всього обсягу документа.

Рамки системи можна моделювати за допомогою діаграми контексту. У поясненнях до діаграми контексту повинні бути чітко визначені рамки системи. Без подібного визначення проект не може бути застрахований від спроб "розтягти" його рамки.

Функціональні вимоги можна моделювати за допомогою діаграми бізнес-прецедентів Однак, діаграма охоплює перелік функціональних вимог тільки в самому загальному виді. Як відзначалося, усі вимоги необхідно позначити, класифікувати і визначити.

Вимоги до даних можна моделювати за допомогою діаграми бізнес-класів. Так само, як і у випадку функціональних вимог, діаграма бізнес-класів не дає повного визначення структур даних для бізнес-процесів. Кожний бізнес-клас вимагає подальших пояснень. Необхідно описати атрибутне наповнення класів і визначити ідентифікуючі атрибути класів. А якщо ні, то неможливо правильно представити асоціації.

Системні обмеження

Системні сервіси визначають, що повинна робити система. Системні обмеження визначають, наскільки система обмежена при виконанні обслуговування. Системні обмеження зв'язані з наступними видами вимог.

- Вимоги до інтерфейсу.
- Вимоги до продуктивності.
- Вимоги до безпеки.
- Експлуатаційні вимоги.
- Політичні і юридичні вимоги.

Вимоги до інтерфейсу визначають, як система взаємодіє з користувачами. У документі опису вимог визначається тільки "враження і відчуття" від GUI-інтерфейсу. Початкове проектування (зафарбовування екрана) GUI-іитерфейса проводиться під час специфікації вимог і пізніше під час системного проектування.

Залежно від області додатка вимоги до продуктивності можуть відіграти досить значну роль в успіху проекту. У вузькому змісті вони задають швидкість (час відгуку системи), з якої повинні виконуватися різні завдання. У широкому змісті, вимоги до продуктивності включають інші обмеження - відносно надійності, готовності, пропускній здатності і т.д.

Вимоги до безпеки описують користувацькі права доступу до інформації, контрольовані системою. Користувачам може бути наданий обмежений доступ до даних або обмежені права на виконання визначених операцій з даними.

Експлуатаційні вимоги визначають програмно-технічне середовище, якщо вона відома на етапі проектування, у якій повинна функціонувати система. Ці вимоги можуть впливати на інші сторони проекту, такі як підготовка користувачів і супровід системи

Політичні вимоги і юридичні вимоги найчастіше маються на увазі, ніж явно формулюються в документі опису вимог. Подібна помилка може обійтися дуже дорого. Поки ці вимоги не виведені явно, програмний продукт може бути важко розгорнути по політичних або юридичним причинам.

Можливі і інші види обмежень. Наприклад, у відношенні деяких систем

можуть пред'являтися підвищені вимоги до легкості їх використання (вимоги відносно придатності до використання) або легкості їх супроводу (вимоги відносно придатності до супроводу).

Значення виробітку недвозначних визначень для системних обмежень важко переоцінити. Існує чимало прикладів проектів, які провалилися через упущені або невірно понятих обмежень. Ця проблема рівною мірою відноситься як до замовників, так і до розробників. Несумлінні або нерозсудливі розробникі можуть розіграти "карту системних обмежень", щоб одержати перевагу у своєму прагненні ухилитися від відповідальності.

Проектні питання

Заключна частина документа опису вимог звертається до інших проектних питань. Один з важливих розділів цієї частини називається “Відкриті питання”. Тут піднімаються всі питання, які можуть позначитися на успіху проекту і які не розглядалися в інших розділах документа. Сюди відноситься очікуване зростання значення деяких вимог, які в теперішній момент виходять за рамки проекту. Сюди можна віднести також будь-які потенційні проблеми і відхилення в поведінці системи, які можуть початися у зв'язку з розгортанням системи.

Додатки

Додатки містять іншу, корисну для розуміння вимог, інформацію. Основним додаванням тут служить глосарій. Глосарій визначає терміни, скорочення і абревіатури, використовувані в документі описи вимог. Значення розумного глосарія важко переоцінити. Невірне тлумачення термінології таїть у собі велику небезпеку для проекту.

Одна з особливостей, яку часто випускають з уваги при складанні документа опису вимог, полягає в тому, що в проблемній області, обумовленої документом, можна досить непогано розібратися за допомогою вивчення документів і форм, використовуваних у процесах діловодства. При можливості слід включати в документ заповнені форми - "порожні" форми не дають такого ж рівня розуміння

бізнес - процесів.

Розділ посилань містить перелік документів, які згадуються або використовуються при підготовці документа опису вимог. До них можуть відноситися книги і інші опубліковані джерела інформації, але - що, мабуть, навіть більш важливо - необхідно також згадати протоколи нарад, службові записки і внутрішні документи.

Специфікація вимог

Специфікація вимог пов'язана з доскональним моделюванням вимог замовників, визначених у процесі встановлення вимог. При цьому розглядаються тільки послуги, які прагнуть одержати від системи замовники (формулювання сервісів)

На етапі специфікації вимог формулювання обмежень не підлягають подальшому опрацювання,, хоча і можуть перетерпіти зміни як результат звичайного циклу ітерації.

У якості вхідної інформації процесу специфікації вимог виступають неформальні вимоги замовників, а результатом цього процесу є моделі специфікації проектних конструкцій.

Ці моделі дають більш формальне визначення різних сторін (представлення) системи. Зазвичай вимоги користувачів у процесі специфікації підрозділяються на дві основні категорії: функціональні вимоги і вимоги до даних.

У якості результату етапу специфікації виступає розширений ("детально пророблений") документ опису вимог. Новий документ часто називають документом специфікації вимог (або просто "специфікацією" на жаргоні розробників). Структура вихідного документа не змінюється, однак зміст значний розширюється за рахунок глав, які визначають вимоги замовників. Поступово для цілей проектування і реалізації документ специфікації вимог заміняє документ опису вимог (на практиці, розширений документ може як і раніше називатися документом опису вимог).

Моделі специфікацій можна розділити на три групи.

1. Моделі станів.
2. Моделі поведінки.
3. Моделі зміни станів.

Моделі станів "деталізують" вимоги до даних. Моделі поведінки забезпечують деталізовані специфікації для функціональних вимог. Моделі зміни станів охоплюють два види вимог. Вони покликані пояснити, яким чином дія функцій приводить до зміни даних.

Моделі представляються у вигляді діаграм мовою візуального моделювання (Visual Modeling Language) - у нашому випадку це мова UML. Зазвичай діаграма служить цілям моделювання однієї зі сторін системи - станів, поведінки або зміни станів. Помітне виключення становить діаграма класів, яка визначає всі три аспекти - стан і поведінку об'єктів, і, опосередковано, зміни станів об'єктів.

Кожна діаграма дає представлення про певну сторону системи. Узяті разом діаграми дають можливість розробникам і користувачам поглянути на запропоноване рішення з різних точок зору, виділяючи одні його сторони і ігноруючи інші. Жодна з діаграм окремо не дає повного визначення системи. Систему можна зрозуміти тільки через взаємозалежний набір діаграм.

Аналогічно випадку інтерпретації завершених моделей конструювання діаграм - це не послідовний процес побудови однієї діаграми за іншою. Діаграми розробляються в паралель, і в результаті кожної наступної ітерації до них додаються нові деталі. У той час, як розробники повинні додержуватися строго певному процесу розробки, рішення про те, яка з моделей повинна відігравати роль "рушійної сили" розробки, значною мірою залежить від особистих переваг аналітика. Зазвичай діаграми прецедентів і моделі класів - як найбільш важливі типи моделей - конструюються паралельно, взаємно "збагачуючи" один одного ідеями.

З кожної новою ітерацією розробки глибина і ступінь деталізації специфікації зростає. Чимало більш глибокі властивості об'єктів моделі виражаються скоріше в текстовому, ніж графічному виді. Деякі властивості визначають задум об'єкта моделі, а не результат аналізу. Деякі інші властивості можуть відображати особливості CASE-засобів.

Специфікації станів

Стан об'єкта визначається значеннями його атрибутів і асоціацій. Наприклад, об'єкт Bankaccount (Банківський рахунок) може перебувати в стані "перевищення кредитного ліміту", якщо значення атрибута balance (баланс) негативне. Оскільки стани об'єкта визначаються структурам даних, моделі структур даних називаються специфікаціями станів.

Специфікація станів дає статичний погляд на систему (тому моделювання станів часто називають статичним моделюванням). Тут основним завданням є визначення класів проблемної області, їх атрибутів і відношень з іншими класами. Спочатку операції класів зазвичай не розглядаються. Вони виводяться з моделей специфікації поведінки.

У типовій ситуації спочатку визначаються класи-сутності, тобто класи, які визначають проблемну область і характеризуються постійною присутністю в базі даних системи. Подібні класи іноді називаються "бізнес-класами". Класи, які обслуговують системні події (керуючі класи) і класи, які представляють GUI-інтерфейс (класи представлення або прикордонні класи) не встановлюються доти, поки не стануть відомі поведінкові характеристики системи.

Моделювання класів

Модель класів - це наріжний (краеугольный) камінь розробки об'єктно-орієнтованої системи. Класи лежать в основі наблюдаємості властивостей і поведінки системи. На жаль, класи завжди важко піддаються визначенню, а властивості класів не завжди очевидні. Досить малоймовірно, щоб два аналітики прийшли до однієї і тієї ж множини класів і їх властивостей для однієї і тієї ж нетривіальної проблемної області. Хоча моделі класів можуть відрізнятися, кінцевий результат і ступінь задоволеності користувача можуть бути рівною мірою достатніми (або рівною мірою недостатніми).

Моделювання класів - це не детермінований процес. Не існує загального рецепта відшукання і визначення найкращих класів. Цей процес значною мірою носить ітеративний і покроковий нарощуваний характер. До факторів, що

визначають успішне проектування класів, відноситься рівень кваліфікації і досвід аналітика, зокрема, перераховані нижче можливості.

1. Знання в області моделювання класів.
2. Розуміння проблемної області.
3. Досвід в області аналогічних і успішних проектів.
4. Здатність дивитися вперед і передбачити наслідки рішень.
5. Готовність до перегляду моделі з метою усунення недоліків і т.д.

Останній момент пов'язаний з використанням CASE-засобів. Широкомасштабне застосування CASE-технології може стати перешкодою на шляху розробки системи в технологічно незрілих організаціях. Однак використання CASE-засобів для підвищення особистої продуктивності завжди виправдане.

Виявлення класів

Два різні аналітики, як правило, не можуть прийти до ідентичних моделей класів для однієї і тієї ж проблемної області, і точно так само два різні аналітики не користуються тим самим розумовим процесом при виділенні класів. Література буяє підходами, пропонованими для виявлення класів. Аналітики можуть спочатку навіть слідувати одному з цих підходів, однак наступні ітерації, як правило, обов'язково приводять до використання нешаблонних і в чомусь навіть випадкових механізмів.

Барамі (Bahrami) докладно вивчив головні особливості чотирьох основних підходів до виявлення класів (class discovery). Нижче перераховані ці підходи.

1. Підхід на основі використання іменних груп.
2. Підхід на основі використання загальних шаблонів для класів.
3. Підхід на основі використання прецедентів.
4. Підхід CRC (class-responsibility-collaborators - клас-обов'язку - "співробітники").

Барамі поставив у відповідність кожному з підходів опубліковані роботи, однак - на нашу думку - тільки останній підхід має незаперечне джерело.

Підхід на основі використання іменних груп

Підхід на основі використання іменних груп (тобто імен іменників у пропозиціях) припускає, що аналітик читає формулювання документа опису вимог у пошуках іменних груп. Кожне ім'я іменник розглядається як потенційний клас (candidate class). Потім список усіх класів розділяється на наступні три групи.

1. Релевантні або підходящі класи.
2. Нечіткі або сумнівні класи.
3. Нерелевантні або невідповідні класи.

До нерелевантних (irrelevant) відносяться класи, які виходять за рамки проблемної області. Для них не вдається дати формулювання їх призначення. Досвідчені аналітики-практики найімовірніше не включають невідповідні класи в первісний список потенційних класів. У такий спосіб вдається уникнути формальних кроків по ідентифікації і виключенню нерелевантних класів.

До релевантних (relevant) відносяться класи, які безумовно належать проблемній області. Імена іменники, що представляють імена цих класів, часто зустрічаються в документі опису вимог. Крім того, значення і призначення цих класів можна обґрунтувати на основі загальних знань про прикладну область, а також на основі вивчення аналогічних систем, інструкцій, документів і патентованих програмних пакетів.

До нечітких (fuzzy) відносяться класи, які не можна впевнено і беззастережно визнати підходящими. Вони становлять найбільшу проблему. Їх необхідно проаналізувати більш глибоко, а потім або включити до списку релевантних класів, або виключити зі списку нерелевантних. Остаточне віднесення цих класів до тієї або іншій групі, властиво, і проводить відмінність між гарною і поганою моделлю класів.

Підхід на основі використання іменних груп припускає наявність повного і коректного документа опису вимог. На практиці це припущення рідко відповідає дійсності. Але навіть якщо воно обґрунтовано, кропітке вивчення великих обсягів тексту не обов'язково повинне привести до одержання вичерпного і точного результату.

Підхід на основі використання загальних шаблонів для класів

Підхід на основі використання загальних шаблонів для класів дозволяє вивести потенційні класи на основі теорії родової класифікації об'єктів. Теорія класифікації стосується поділу світу об'єктів на зручні групи, що дозволяє більш ефективно будувати міркування про них.

Барами приводить наступний перелік груп (шаблонів) для виявлення потенційних класів.

- *Понятійний (або концептуальний) клас (concept class)* являє собою ідею, яку розділяє або з якою згодна значна спільність людей. Під час відсутності понять люди не здатні ефективно спілкуватися або навіть спілкуватися на якомусь задовільному рівні. Наприклад, Reservation (Резервування) – це понятійний клас, що відноситься до системи резервування місць в авіакомпаніях.
- *Подійний клас (events class).* Подія - це щось, що не вимагає часу відносно до нашої тимчасової шкали. Наприклад, Arrival (Прибуття) - це подійний клас, що відноситься до системи резервування місць в авіакомпаніях.
- *Організаційний клас (organization class).* Організація – це будь-який вид цілеспрямованого об'єднання або сукупності сутностей. Наприклад, Travelagency (Бюро подорожей) – це клас, що відноситься до системи резервування місць в авіакомпаніях.
- *Клас "людей" (people class).* Під "людьми" тут розуміється скоріше роль, яку людина відіграє в тій або іншій системі, а не фізична особа. Наприклад, Passenger (Пасажир) - це клас, що відноситься до системи резервування місць в авіакомпаніях.
- *Клас місць розташування (places class).* Місце розташування визначає фізичне розташування об'єктів, пов'язаних з інформаційною системою. Наприклад, Traveloffice (Офіс бюро подорожей) - подібний клас, що відноситься до системи резервування місць в авіакомпаніях.

Дж. Рамбау (J. Rumbaugh), А. Джекобсон (I. Jacobson) і Г. Буч (G. Booch) пропонують іншу схему класифікації.

- Фізичний клас (physicalclass) (наприклад, Airplane (Літак)).
- Бізнес-клас (business class) (наприклад, Reservation).
- Логічний клас (logical class) (наприклад, Flighttimetable (Розклад рейсів)).
- Прикладний клас (application class) (наприклад, Reservationtransaction (Операція резервування)).
- Комп'ютерний клас (computerclass) (наприклад, Index (Індекс)).
- Поведінковий клас (behavioral class) (наприклад, Reservationcancellation (Скасування резервування)).

Підхід на основі використання загальних шаблонів класів служить в якості корисного керівництва, але не визначає систематичного процесу, за допомогою якого можна було б виділити надійну і повну множину класів. Цей підхід можна з успіхом використовувати для визначення початкової множини класів або для перевірки того, чи повинні деякі класи (отримані іншими способами) бути присутніми у нашій множині або, навпаки, відсутнім у ньому. Однак, підхід на основі використання загальних шаблонів класів занадто слабко пов'язаний зі специфічними вимогами користувачів, щоб претендувати на вичерпне рішення.

Особлива небезпека, пов'язана з підходом на основі використання загальних шаблонів класів, полягає в невірному тлумаченні імен класів. Наприклад, що означає Arrival (Прибуття)? Чи означає це прибуття на злітно-посадочну смугу (час приземлення), прибуття до термінала (час висадження), прибуття в зал повернення багажу (час митного огляду) і т.д.? Аналогічно, слово Reservation (у цьому випадку резервація) у середовищі північноамериканських індіанців має зовсім інше значення в порівнянні з тим, що малося на увазі дотепер.

Підхід на основі використання прецедентів

Підходу на основі використання прецедентів надається особливе значення в мові UML. Можна навіть сказати, що цей підхід рекомендується використовувати в рамках UML (якщо бути точним - то в рамках методології RUP (Rational Unified Process)). Графічна модель прецедентів супроводжується неформальними описами, а також діаграмами послідовностей і кооперації для окремих прецедентів.

Ці додаткові описи і кроки визначення діаграм (і об'єктів) потрібно виконати для кожного прецеденту. На основі цієї інформації можна прийти до узагальнень, необхідних для виявлення потенційних класів.

Підхід, що направляється прецедентами, має особливості, властиві підходу знизу-нагору. Після того, як прецеденти стають відомі, а представлення про систему з погляду взаємодії, щонайменше, частково визначене за допомогою діаграм послідовностей, об'єкти, використовувані в цих діаграмах, приводять до виявлення класів.

У дійсності цей підхід у чомусь схожий на підхід, що використовує іменні групи. Їх поєднує той факт, що прецеденти специфікують вимоги. Обидва підходи спрямовані на вивчення формулювань, викладених у документі опису вимог, щоб виявити в підсумку потенційні класи. Те, що ці формулювання викладаються в оповідальній формі або представлені графічно, має другорядне значення. У кожному разі на цьому етапі ЖЦ розробки ПЗ більшу частину прецедентів можна описати тільки в текстовій формі без діаграм взаємодії.

Підхід, заснований на прецедентах, страждає тими ж недоліками, що і підхід, що використовує іменні групи. Будучи по суті підходом знизу-нагору в змісті точності, він опирається на повноту і коректність моделей прецедентів. У результаті, він навіть може привести до небажаного розбалансування ітеративного і нарощуваного процесу розробки ПЗ, при якому моделі прецедентів повинні бути завершені ще до побудови моделей класів. Загалом, хоч би які були цілі і засоби, це приводить до функціонального підходу (function-driven approach) (прихильники об'єктно-орієнтованого підходу воліють називати його проблемно-орієнтованим (problem-driven)).

Комплексний підхід

На практиці процес виявлення класів у різний час, найімовірніше, додержується різним підходів. Найчастіше, він включає елементи всіх чотирьох підходів, розглянутих вище. Немаловажними факторами при цьому виступають загальна ерудиція експерта, його досвід і інтуїція. Процес у чистому вигляді не слідує ні методу зверху-униз, ні методу знизу-нагору - він увесь час іде "з

середини". Подібний підхід до виявлення класів ми називаємо комплексним підходом (mixed approach).

От один з можливих сценаріїв. Початкову множину класів можна сформувати на основі загальних знань і досвіду експерта. При цьому додатково можна керуватися підходом на основі загальних шаблонів класів. Інші класи можна додати, ґрунтуючись на аналізі узагальненого опису проблемної області з використанням підходу на основі іменних груп. Якщо в розпорядженні аналітика є прецеденти, можна скористатися підходом, що направляються прецедентами, щоб додати нові і перевірити спроможність існуючих класів.

Деякі правила виявлення класів

Нижче наведений перелік керівних принципів або правил, яким повинен слідувати аналітик при виборі потенційних класів.

1. Для кожного класу повинне бути ясно сформульоване його призначення в системі.

2. Кожний клас - це шаблон опису множини об'єктів. Одиничні класи, для яких можна представити існування тільки одного об'єкта, досить малоймовірні серед "бізнес-об'єктів". Подібні класи зазвичай складають у додатку "загальне знання" і як правило твердо запрограмовані в програмах додатка. Наприклад, якщо система спроектована для єдиної організації, існування класу Organization (Організація) може бути не виправдано.

3. Кожний клас (тобто клас-сутність) повинен містити набір атрибутів. Гарним прийомом є встановлення ідентифікуючих атрибутів (ключів), щоб допомогти нам судити про потужність (cardinality) класу (тобто очікуваній кількості об'єктів даного класу в базі даних). Слід, однак, пам'ятати про те, що клас не обов'язково повинен володіти користувальницьким ключем. Об'єкти класів ідентифікуються за допомогою ідентифікаторів об'єктів (OID) .

4. Кожний клас повинен відрізнятися від атрибута. Чи представляється поняття класом або атрибутом який залежить від області додатка. Колір автомобіля зазвичай сприймається як атрибут класу Car (Автомобіль). Однак на фабриці по виробництві фарб Color (Колір) - це визначений клас зі своїми власними атрибутами

(яскравістю, насиченістю, прозорістю і т.д.).

Завдання: Виконати специфікацію вимог проекту. Виконати прототипування і розробку додатків проекту.

Надати звіт, що містить результати специфікації, прототипування і розробки додатків системи.

Контрольні питання:

1. Які принципи встановлення вимог?
2. Що таке домінантний клас?
3. Що таке відношення з'єднання?
4. Поясніть, у чому полягають основні відмінності чотирьох підходів до виявлення класів.
5. У чому полягає сутність підходу на основі використання загальних шаблонів класів?

Лабораторна робота № 3

Тема: Системне проектування. Проектування баз даних.

Ціль: Придбання практичних навичок системного проектування й проектування баз даних

Короткі теоретичні відомості

Системне проектування містить у собі два основні питання: архітектурне проектування і деталізоване проектування. Архітектурне проектування охоплює багаторівневу організацію класів і пакетів, розподіл процесів по обчислювальних засобах, повторне використання і керування компонентами. Деталізоване проектування звернене до моделей кооперації, необхідним для реалізації функціональних можливостей системи, зафіксованих у прецедентах.

Архітектура програмного забезпечення

Проект являє собою низькорівневу модель архітектури системи і її внутрішніх функції. Проектування здійснюється в термінах програмно-апаратної платформи, на якій належить реалізувати систему. При ітеративній і нарощуваній розробці моделі аналізу безупинно "обростають" технічними подробицями. Як тільки технічні подробиці включають міркування, що стосуються програмного і апаратного забезпечення, модель аналізу стає проектною моделлю.

Між аналізом і проектуванням не можна провести різкої границі. Можна заглиблюватися в технічні питання, не торкаючись специфічних програмно-апаратних рішень. У цьому змісті більшу частину питань, можна віднести скоріше до аналізу, ніж до проектування.

Опис системи в термінах її модулів називається архітектурним проектуванням (architectural design). Архітектурний проект включає вибір стратегії рішень у відношенні клієнтської і серверної компонентів системи.

Опис внутрішніх функцій кожного модуля (прецеденту) називається

деталізованим проектуванням (detailed design). Деталізований проект спрямований на розробку завершених алгоритмів і структур даних для кожного модуля. Ці алгоритми і структури даних пристосовуються до всіх (підсилюючих і нав'язуваних) обмеженням базової платформи реалізації.

Розподілена архітектура

Архітектурне проектування пов'язане з вибором стратегії рішень і модулярізацією системи. Стратегія рішення покликана розв'язати проблеми, пов'язані з побудовою клієнтської і серверної частин системи, а ПЗ проміжного шару (middleware) необхідно для "склеювання" клієнта і сервера. Рішення по основних будівельних блоках (модулям) тільки частково залежить від обраної стратегії рішення.

Клієнт і сервер - логічні поняття. Клієнт (client) - це обчислювальний процес, який здійснює запити до процесу сервера. Сервер (server) - це обчислювальний процес, який обслуговує запити сервера. Зазвичай процеси клієнта і сервера виконуються на різних комп'ютерах, але цілком можливо реалізувати систему клієнт/сервер на одній машині.

У типовому сценарії клієнтський процес відповідає за керування відображенням інформації на екрані користувача і за обробку подій, ініційованих користувачами. Процес сервера - це будь-який комп'ютерний вузол з базою даних, з якої дані можуть бути запитані клієнтським процесом.

Архітектуру клієнт/сервер можна розширити для представлення довільної розподіленої системи. Будь-який комп'ютерний вузол з базою даних може відігравати роль клієнта в одних ділових операціях, а сервер - в інших операціях. З'єднання подібних вузлів за допомогою мережі зв'язку дає початок архітектурі системи розподіленої обробки, як показано на рис. 3.1.

Рис. 3.1. Архітектура системи розподіленої обробки

У системі розподіленої обробки клієнт може здійснювати доступ до будь-якої кількості серверів. Однак клієнту може бути дозволений доступ одночасно тільки до одному серверу. Це значить, що він може бути не в змозі об'єднати дані від двох або більше серверів баз даних в одному запиті. Якщо це можливо, то архітектура підтримує систему розподілених баз даних.

Триланкова архітектура

Аналогічно клієнтському і серверному процесу прикладний процес являє собою логічне поняття, яке може підтримуватися або не підтримуватися спеціально виділеним для цієї мети апаратним забезпеченням. Логіка додатка може з рівним успіхом виконуватися на клієнтському або серверному вузлі, тобто може бути вбудована в клієнтський або серверний процес і реалізована у вигляді бібліотеки DLL (Dynamic Link Library - динамічно компонуєма бібліотека), API-інтерфейсу (Application Programming Interface - інтерфейс прикладного програмування), RPC-викликів (Remote Procedure Calls - віддалений виклик процедури) і т.д.

Якщо логіка додатка скомпільована з клієнтом, говорять про архітектуру товстого клієнта (thick client architecture) ("клієнт на стероїдах"). Якщо вона

скомпільована з сервером, говорять про архітектуру тонкого клієнта (thin client architecture) ("клієнт" "шкіра так кістки"). Можливі також проміжні архітектури, у яких логіка додатка частково скомпільована з клієнтом, а частково - з сервером. Логіку додатка можна також розгорнути на окремих обчислювальних вузлах, як показано на рис. 3.2.

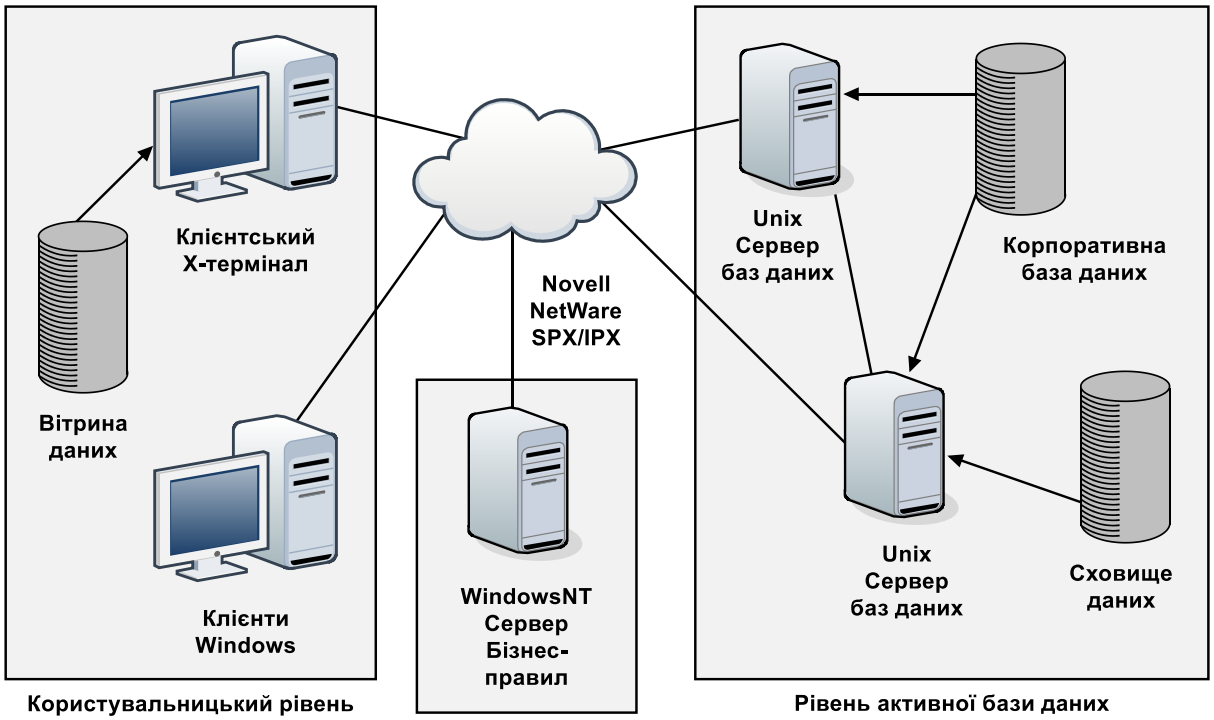

Рис. 3.2. Триланкова архітектура

Це триланкова архітектура в самому чистому вигляді. До її кращих сторін відносяться висока гнучкість, розширюваність, незалежність користувача, готовність і низька вартість відновлення. Однак подібна архітектура може відрізнятися високою початковою вартістю, а крім того може зазнавати деякі проблеми з продуктивністю.

Програмування баз даних

Незалежно від того, де розташована логіка додатка, програма (клієнт) взаємодіє з базою даних (сервером), щоб одержати інформацію для її відображення і надання в розпорядження користувача для подальших маніпуляцій. Однак сучасні

бази даних також можна програмувати. Говорять, що ці сучасні бази даних активні.

Програми баз даних називаються збереженими процедурами (stored procedure). Збережені процедури зберігаються в самій базі даних (вони є постійними або персистентними). Їх можна викликати з клієнтської програми (або з іншої збереженої процедури) за допомогою звичайного оператора виклику процедури/функції.

Існують збережені процедури спеціального виду - тригери, які не можна викликати явно. Тригер спрацьовує автоматично при спробі змінити вміст бази даних. Тригери використовуються для реалізації бізнес-правил масштабу підприємства, які повинні бути проведені в життя способом, незалежним від клієнтських програм (або збережених процедур). Тригери підсилюють цілісність і несуперечність баз даних. Вони не дозволяють окремим додаткам порушувати бізнес-правила, закладені в базу даних.

Взаємодія "додаток - база даних"

Нам необхідно розв'язати, яка частина системи буде запрограмована у клієнті, а яка - у базі даних. При цьому розглядаються наступні програмувальні частини системи.

- Користувацький інтерфейс.
- Презентаційна логіка.
- Прикладні функції.
- Інтегральна логіка.
- Функції доступу до даних.

Частина програми, яка називається користувацьким інтерфейсом, відповідає за відображення інформації на конкретний GUI-інтерфейс, такий як GUI-інтерфейс Microsoft Windows, GUI-інтерфейс Unix Motif, GUI-інтерфейс Macintosh. Презентаційна логіка (або логіка представлення) відповідає за обробку об'єктів GUI-інтерфейсу (форм, меню, кнопок дій і т.д.), як того вимагають функції додатка.

Функції додатка містять основну логіку програми. Вони фіксують дії додатка і являють собою сполучну ланку, що з'єднує разом клієнта і базу даних. З погляду

підходу всі функції додатка реалізуються класами керуючого пакета.

Інтегральна логіка відповідає за бізнес-правила масштабу підприємства. Це правила, які застосовуються до всіх прикладних програм, тобто ***BCE*** програми повинні функціонувати відповідно до них. Функції доступу до даних володіють питаннями доступу до постійних об'єктів даних на диску.

***** Підхід *BCE* (Boundary-Control-Entity - межа, управління, сутність) являє собою підхід до об'єктного моделювання, заснований на трьохфакторному представленні класів. В UML на класах зумовлені три стереотипи: boundary (межа), control (управління), entity (сутність). Це і визначило вибір абревіатури BCE.

Стратегія повторного використання

UML визначає повторне використання (reuse) як "використання раніше існуючих артефактів". Стратегія впливає на ступінь деталізації, з якою здійснюється повторне використання. Можуть застосовуватися наступні ступені деталізації повторного використання.

- Клас.
- Компонента.
- Ідея рішення.

У зв'язку зі ступенем деталізації існують три відповідні стратегії повторного використання в основі яких лежать наступні програмні сутності.

1. Інструментальні засоби (бібліотеки класів).
2. Каркаси.
3. Шаблони аналізу і проектування.

Компоненти

Компонента (component) - це фізична частина системи, фрагмент реалізації, програма. Компоненти найчастіше сприймаються як двійкові виконувані (EXE-файли) частини системи. Але компонента може бути також частиною системи, яка

не є модулем, що безпосередньо виконується (наприклад, файлом вихідного тексту програми, файлом даних, динамічно компонуємою бібліотекою (Dynamic Link Library - DLL) або збереженою процедурою бази даних).

У цей час UML визначає п'ять стандартних стереотипів для компонентів.

1. Виконувана (тобто модуль, що безпосередньо виконується).
2. Бібліотека (тобто модуль статичної або динамічної об'єктної бібліотеки).
3. Таблиця (тобто таблиця бази даних).
4. Файл (тобто файл вихідного тексту або даних).
5. Документ (тобто документ, призначений для сприйняття людиною).

Нижче приводиться перелік характеристик компонент.

- Компонент являє собою програмний блок, що розгортається незалежно (компонента ніколи не розгортається частково).
- Компонента може служити будівельним блоком для стороннього розробника (тобто компонента в достатній мері документована і самодостатня, щоб сторонній розробник міг "вмонтувати" її в інші компоненти).
- Компонент не має тупиковий стан (тобто компоненту неможливо відрізнити від її власних копій; у рамках будь-якого даного додатка присутнє якнайбільше одна копія конкретного компонента).
- Компонент – замінна частина системи, тобто її можна замінити іншим компонентом, який узгоджується з тим же інтерфейсом.
- Компонент виконує чітку функцію і з логічної і фізичної точки зору утворює єдине ціле.
- Компонента може бути вкладена в інші компоненти.

Розгортання

У мові UML триланкова архітектура (на зразок тієї, що показана на рис. 3.2) або будь-яка інша архітектура для системи представлена у вигляді діаграми розгортання (deployment diagram). Фактично діаграма, представлена на рис. 3.2, є

допустимою діаграмою розгортання UML, на якій обчислювальні ресурси представлені у вигляді спеціальних піктограм.

Проект розгортання

Характер Internet-систем без встановлення прямого з'єднання робить розгортання (deployment) Web-додатків значно складнішим завданням, ніж розгортання додатків баз даних в архітектурі клієнт/сервер. Щоб приступити до розгортання, потрібно встановити Web-сервер як пункт маршрутизації між усіма броузерами клієнтів і базою даних.

Якщо проблема керування сеансами не може бути задовільно вирішена за допомогою технології cookie, необхідно залучати на допомогу технологію розподілених об'єктів. Розгортання розподілених об'єктів може вимагати розміщення окремих архітектурних елементів - сервера додатків - між Web-сервером і сервером баз даних.

Проект розгортання повинен звертатися до питання безпеки. Безпечна передача даних і протоколи шифрування - ще один аспект вимог з боку проекту розгортання. Крім того, необхідно ретельне планування з урахуванням мережного завантаження, Internet-з'єднань, резервних копій і т.д.

Розгортання Web-додатків

Архітектура розгортання, здатна підтримувати більш складні Web-додатки, включає чотири ланки обчислювальних вузлів.

1. Клієнтський Web-броузер.
2. Web-сервер.
3. Сервер додатків.
4. Сервер баз даних.

Броузер клієнтського вузла можна використовувати для відображення статичних або динамічних Web-сторінок. Сторінки, що включають сценарії і аплети, можна завантажувати і виконувати в рамках броузера. Клієнтський броузер можна

оснастити додатковими функціональними можливостями, такими як елементи керування Activex або Javabeans. Виконання програми додатка на клієнтській машині, але поза броузера, може задовольнити інші вимоги до GUI-інтерфейсу.

Web-сервер обробляє запити на сторінці, що надходять від броузера, і динамічно генерує сторінки і програмний код для виконання і відображення на клієнту. Web-сервер також забезпечує настроювання і параметризацію сеансів роботи користувача.

Сервер додатків необхідний у тому випадку, коли в реалізації використовуються розподілені об'єкти. Він управляє бізнес-логікою. Бізнес-компоненти публікують свої інтерфейси для інших вузлів через інтерфейси компонентів, такі як CORBA, DCOM або EJB.

Бізнес-компоненти інкапсулюють постійні об'єкти, збережені в базі даних, найчастіше в реляційній базі. Вони взаємодіють з сервером баз даних через протоколи зв'язку з базами даних, наприклад, такими як JDBC або ODBC. Вузол бази даних забезпечує масштабоване сховище даних і багатокористувацький доступ до нього.

Проектування баз даних

Можна сказати, що інформаційні системи є багатокористувацькими системами за визначенням. Ця властивість сама по собі вимагає наявності бази даних, з якою можуть одночасно працювати багато користувачів. Прикладні програми залежать від бази даних, однак, зворотне твердження невірне. Висновок очевидний - належний проект бази даних, який може об'єднати і підтримувати прикладні програми, є необхідною умовою реалізації інформаційною системою передбачених функціональних можливостей.

У мові UML діаграма класів визначає структури даних, що вимагаються додатком. Структури даних, які постійно знаходяться у базі даних, моделюються за допомогою класів-сутностей, а також як відношення між класами-сутностями. Класи-сутності необхідно відобразити в структури даних, які розпізнаються базою даних. Ці структури даних змінюються залежно від базової моделі бази даних, яка може бути об'єктно-орієнтованою, об'єктно-реляційною або реляційною.

Тут розглядається відображення об'єктів у бази даних і пояснюється, яким чином здійснюються перетворення класів-сутностей, асоціацій, агрегацій і узагальнень у структури даних, наявні в розпорядженні кожної з трьох моделей баз даних.

Моделі даних

Фахівці з баз даних виробили своє власне уявлення про світ моделювання. Бази даних зберігають дані. Історично фахівці з баз даних концентрувалися на моделях даних (тобто на мові UML - моделях стану). Сучасні можливості баз даних по зберіганню і виконанню програм поширили цю перспективу на моделі поведінки (орієнтовані на тригери і збережені процедури), однак моделювання даних як і раніше залишається альфою і омегою розробки баз даних.

Модель даних (data model) (названа також схемою бази даних (database schema) - це абстракція, яка представляє структури бази даних у термінах більш зрозумілих, ніж ці страшні біти і байти. Широко відома класифікація рівнів моделі даних виділяє три рівні абстракції.

1. Зовнішня (концептуальна модель).
2. Логічна модель даних.
3. Фізична модель даних.

Зовнішня схема (external schema) представляє узагальнену концептуальну модель даних, необхідну для єдиного додатка. Оскільки база даних зазвичай підтримує багато додатків, конструюються кілька зовнішніх схем. Потім вони інтегруються у вигляді однієї концептуальної моделі даних. Найбільш відомим методом концептуального моделювання даних є ER-діаграми (entity-relationship - сутність зв'язок).

Логічна схема (logicalschema) (іноді називаєма також глобальною концептуальною схемою) представляє модель, яка відображає структури зберігання СУБД. Саме глобальна інтегрована модель підтримує поточні і перспективні додатки, яким потрібен доступ до інформації, збереженої в базі даних.

Фізична схема (physical schema) відображає специфіку конкретної СУБД. Вона визначає спосіб фактичного зберігання даних в енергонезалежних пристроях пам'яті, як правило, це дискові накопичувачі. Фізична схема стосується таких питань, як використання індексів і кластеризація даних для підвищення ефективності обробки даних.

Конструкторські CASE-засоби (тобто засоби, спрямовані на проектування і реалізацію систем) надають розробникам єдиний метод моделювання даних для логічної і фізичної схеми. Зазвичай такі засоби трактують подібну комбіновану модель як фізичну модель даних.

На рис. 3.3 показано, яким чином в UML моделюється відношення додатка і моделі постійно збережених об'єктів бази даних. Класи пакетів-сутностей представляють "бізнес-об'єкти" додатка. Діаграма класів UML (для класів-сутностей) може замінити ER-діаграму як альтернативний засіб концептуального моделювання баз даних.

Пакети баз даних не "направляють" процес моделювання баз даних, вони скоріше направляються ним. Пакет баз даних ізолює модель додатка від моделі бази даних, проектує одночасно з визначенням рівня архітектури постійно збережених об'єктів бази даних або після визначення цього рівня. Пакет баз даних роз'єднує класи-сутності від схеми бази даних. Він встановлює відображення між об'єктами і базою даних

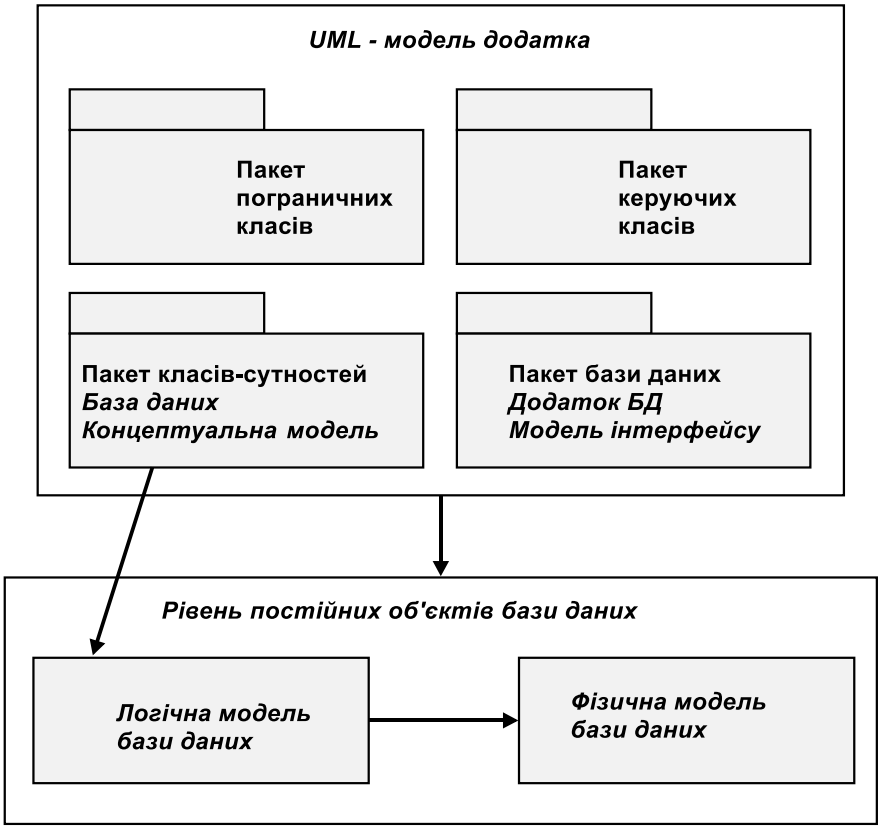

Рис. 3.3. UML і моделі постійно збережених об'єктів

Модель об'єктної бази даних

Модель об'єктної бази даних (ОБД) доставляє найменше клопоту при відображенні структур даних між прикладною програмою і базою даних. У дійсності, головною метою об'єктної СУБД є прозора інтеграція бази даних з мовою програмування, на якій написаний додаток.

Консорціум ODMG (Object Data Management Group - група по керуванню об'єктними даними) стандартизував модель ОБД. Організації, що входять до складу ODMG, представляють всіх основних постачальників ПЗ СУБД. Зовсім недавно ODMG змінила напрямки своєї діяльності, сконцентрувавши основні зусилля на відображенні об'єктів у реляційні і інші типи баз даних. Ці зусилля знайшли своє вираження в розробці стандартного API-інтерфейсу об'єктної пам'яті (Object Storage API), який здатний працювати з будь-якими постійними джерелами даних. По суті, цей стандарт можна використовувати в якості пакета баз даних для відображення між додатком і базою даних. Останній стандарт одержав назву об'єктного стандарту даних (Object Data Standard): ODMG 3.0.

Стандарт визначає, що система управління об'єктними базами даних (СУОБД) не забезпечує окремої мови для баз даних (на зразок SQL) для маніпулювання даними в межах середовища мови програмування. Замість цього він передбачає появу об'єктів бази даних у мові програмування додатків в якості звичайних об'єктів мови програмування. Інакше кажучи, мова програмування розширюється за рахунок об'єктів бази даних, які реалізують функції постійного зберігання об'єктів, керування транзакціями, навігаційні запити (тобто запити, які дозволяють "переміщатися" уздовж відношень) і т.д.

Об'єктно-реляційна модель бази даних

Наступною "великою хвилею" в області технологій баз даних є об'єктно-реляційна модель. Як ясно з назви, об'єктно-реляційна база даних (ОРБД) поєднує в собі старомодну реляційну модель і новомодну об'єктну модель. Та сама об'єктно-реляційна система керування базами даних (СУОРБД) здатна обробляти реляційні структури даних (реляційні таблиці) і об'єктні структури даних (об'єктні таблиці).

Стандарт для моделі ОРБД був погоджений в 1999 році, після більш ніж шестирічної розробки (цей стандарт відомий під неофіційним іменем SQL3). Даний стандарт є результатом праці Американського національного інституту стандартів (American National Standards Institute - ANSI) і Міжнародної організації по стандартизації (International Organization for Standardization- ISO). Офіційна назва цього стандарту SQL:1999. Стандарт залишив багато питань, що стосуються ОРБД, невирішеними і повинен по положенню переглядатися приблизно раз у три роки.

Модель ОРБД сумісна "знизу вгору" з останнім стандартом для реляційних баз даних - так званим стандартом SQL92. Модель розширює традиційні можливості реляційних таблиць за рахунок нового механізму, що дозволяє зберігати об'єкти в SQL таблицях. Модель також розширює обмежену реляційну підтримку для обумовлених користувачем типів за рахунок введення довільних складних структурованих типів (щоб інкапсулювати атрибути і операції в одному об'єктному типі - класі).

Хоча стандарт розвивався (і продовжує розвиватися), більшість постачальників реляційних баз даних (Oracle, IBM, Informix) взялися за завдання

поставки продуктів для СУОРБД, що забезпечують щонайменше часткову підтримку моделі ОРБД. Однієї з основних проблем для постачальників залишається інтеграція раніше існуючих реляційних можливостей з новими об'єктно-орієнтованими, щоб уможливити безперешкодний перенос реляційних систем у рішення, засновані на використанні ОРБД. Стандарт SQL: 1999 фактично не торкається цієї проблеми.

Елементарні типи моделі РБД

Елементарні типи моделі РБД, зазвичай ж, елементарні по визначенню. Простота моделі РБД, яка випливає з математичного поняття множини, становить як сильну, так і слабку сторони цієї моделі. Математичний фундамент, на якому ґрунтується реляційна модель, визначив її декларативний характер (на противагу процедурному). Користувач просто повідомляє, що йому потрібно одержати від бази даних, а не інструктує систему про те, як саме відшукати потрібну інформацію (СУРБД знає, як шукати дані у своїй базі даних).

Однак те, що видасться простим спочатку, стає досить складним разом з ростом складності розв'язуваної проблеми. Для складних проблем немає простих рішень. Щоб розв'язати складну проблему, потрібні тонкі механізми. Щоб приступитися до моделювання, будуть потрібні розвинені елементарні типи даних.

Напевно кращий шлях охарактеризувати модель РБД - сформулювати, які з елементарних типів вона не підтримує. З основних елементарних типів моделювання, застосовуваних у моделях ОБД і ОРБД, реляційна модель не підтримує наступних.

1. Об'єктні типи і пов'язані з ними поняття (такі, як спадкування або методи).
2. Структуровані типи.
3. Колекції.
4. Посилання.

Основним елементарним типом моделі РБД є реляційна таблиця (relational table), яка складається зі стовпців. Стовпці, таблиці можуть приймати тільки атомічні значення - структуровані значення або значення колекцій не допускаються.

Модель РБД поводиться досить жорстко у відношенні будь-яких видимих

користувачу навігаційних зв'язків між таблицями - вони виключаються. Відношення між таблицями підтримуються за допомогою порівняння значень у стовпцях. Постійні зв'язки відсутні. Корисна властивість ОРБД по підтримці визначених відношень між таблицями називається *посилальною цілісністю (referential integrity).*

На рис. 3.4 показані елементарні типи моделі РБД і залежності між ними.

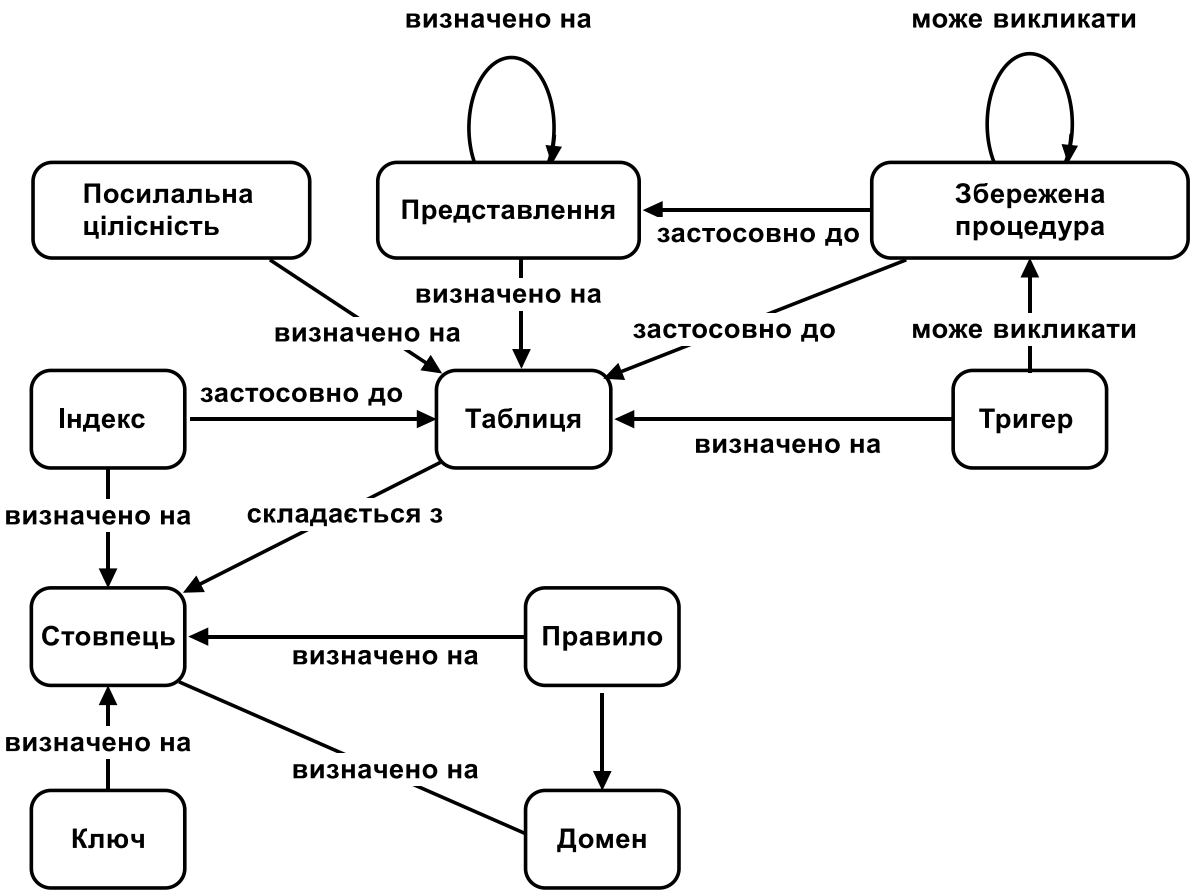

Рис. 3.4. Залежності між елементарними типами моделі РБД

Реляційні бази даних визначають дані в стовпцях і рядках. Значення даних, збережених на перетинанні будь-якого стовпця і рядка, повинне бути простим (неподільним) і однократним (не повторюваним) значенням. Ми говоримо, що стовпці утворюють атомічні домени (atomic domains) (типи даних).

Домен (domain) визначає допустиму множину значень, яка може приймати стовпець. Домен може бути безіменним (наприклад, gender char (1)) або може бути пойменований (наприклад, gender Gender). В останньому випадку домен Gender був визначений раніше і використовується у визначенні стовпця. Нижче приводиться

можливий синтаксис визначення домена.

```
create domain Gender char(1);
```

Іменований домен можна використовувати для визначення багатьох стовпців у різних таблицях. Це сприяє досягненню несуперечності між цими визначеннями. Зміни, внесені у визначення домена, автоматично відображаються на визначенні стовпця. Хоча подібна можливість і виглядає на перший погляд досить привабливо, її використання відразу стає перешкодою, як тільки база даних заповнюється, тобто завантажується даними.

Зі стовпцями і доменами можуть бути зв'язані деякі бізнес-правила, що накладають на них обмеження. Бізнес-правила можуть визначати наступні характеристики даних.

- Значення, використовуване за замовчуванням (наприклад, якщо для міста (city) не задане ніякого значення, передбачається значення "Сідней").
- Діапазон значень (наприклад, допустиме значення для віку (age) лежить у межах від 18 до 80).
- Список значень (наприклад, допустимими кольорами (color) можуть бути "зелений", "жовтий" або "червоний").
- Регістр, використовуваний для представлення значення (наприклад, значення повинне складатися з символів верхнього або нижнього регістру клавіатури).
- Формат значення (наприклад, значення повинне починатися з букви " ДО").

За допомогою засобів завдання правил можна визначити тільки прості правила, що відносяться до єдиного стовпця або домену. Більш складні правила, що охоплюють таблиці, можна визначити за допомогою правил, обмеження цілісності. Основним механізмом визначення бізнес-правил є тригери.

Реляційні таблиці

Реляційна таблиця (relational table) визначається фіксованою множиною своїх стовпців. Типи даних, які зберігаються в стовпцях, відносяться до вбудованих або

визначених користувачем типам (тобто доменам). Таблиця може містити довільну кількість рядків (або записів). Оскільки таблиця є не що інше, як математична множина, повторювані рядки в таблиці відсутні.

Для деяких стовпців допускається, що значення стовпця в визначеному рядку може приймати значення NULL. Значення NULL означає одну з двох можливостей: "не визначене в цей момент значення" або "непридатне значення".

Одним з наслідків вимоги до моделі РБД, який полягає у неприпустимості дублювання рядків, є наявність у кожної таблиці первинного ключа (primary key). Ключ - ця мінімальна множина стовпців (можливо один) таких, що значення в цих стовпцях єдиним способом ідентифікують один рядок у таблиці. Таблиця може містити багато подібних ключів. Один з цих ключів довільним чином вибирається як найбільш важливий - це первинний ключ, (primary key). Інші ключі називаються потенційними (candidate) або альтернативними (alternative) ключами.

На практиці таблиця СУРБД не зобов'язана мати ключ. Це значить, що таблиця (без унікального ключа) може містити ідентичні рядки - дивна марна властивість реляційної бази даних, коли два рядки, що містять однакові значення в усіх своїх стовпцях ніяк не можна відрізнити. У системах ОБД і ОРБД подібне розрізнення забезпечується за рахунок наявності OID (два об'єкти можуть бути рівними, але не ідентичними, наприклад, як дві копії однієї книги).

Хоча існує можливість увести в UML стереотипи для моделювання реляційних баз даних, більш зручно використовувати спеціально призначені для цього діаграмні методи, що дозволяють здійснити логічне моделювання реляційних баз даних.

На рис. 3.5 показано одна з подібних систем нотації. У якості цільової бази даних тут використовувалася база даних DB2 компанії IBM.

Employee			
emp_id	CHAR(7)	<pk>	not null
family_name	VARCHAR(30)	<ak>	not null
first_name	VARCHAR(20)		not null
date_of_birth	DATE	<ak>	not null
gender	Gender		not null
phone_num1	VARCHAR(12)		null
phone_num2	VARCHAR(12)		null
salary	DEC(8,2)		null

Рис. 3.5. Визначення таблиць реляційної бази даних

Таблиця Employee складається з восьми стовпців. Останні три стовпці допускають в якості значень використання значення NULL. Стовпець `emp_id` являє собою первинний ключ. Стовпці `{family_name, date_of_birth}` визначають потенційні (альтернативні) ключі. Стовпець gender визначений на домені Gender.

Оскільки на РБД накладається обмеження, яке полягає в тому, що стовпці можуть приймати тільки атомічні однократні значення, моделювання імені і телефонного номера працівника викликає в нас певні труднощі. У попередньому випадку ми використовували два стовпці : `family_name` і `firstname`. Стовпці не згруповані і ніяк не зв'язані в моделі. В останньому випадку ми віддали перевагу рішенню з двома стовпцями `(phone_numl, phone_num2)`, що допускають наявність максимум двох телефонних номерів у кожного працівника.

Після того, як таблиця визначена за допомогою CASE-засобів, можна автоматично згенерувати програмний код для створення таблиці, як показано нижче. Згенерований текст програми включає визначення домена Gender і визначення бізнес-правила, визначеного на цьому домені.

Завдання: Виконати системне проектування програмної системи. Обґрунтувати вибір бази даних і виконати її проектування.

Надати звіт, що містить результати системного проектування і проектування бази даних

Контрольні питання:

1. Поясніть, у чому полягає відмінність між розподіленою системою обробки і розподіленою системою баз даних.
2. Що таке триланкова архітектура? У чому її переваги і недоліки?
3. Що таке домінантний клас?
4. Що таке відношення з'єднання?
5. Які відмінності між об'єктною і реляційною моделлю БД?

Лабораторна робота № 4

Тема: Проектування програмної системи

Ціль: Оволодіння навичками по проектуванню програмної системи

Короткі теоретичні відомості

Проектування програмних модулів - невід'ємна частина проектування всієї системи. Архітектурне проектування пакетів класів і компонент встановлює вихідну схему роботи додатка. Деталізоване проектування GUI-інтерфейсу і баз даних визначає клієнтську і серверну складові цієї схеми. Проектування програми заповнює прогалини в початковій схемі і перетворює її в проектний документ, який можна передати програмістам для реалізації.

Проектування концентрується одночасно на одній програмі додатка. У цьому сенсі проектування програми є прямим продовженням проектування користувацького інтерфейсу. При проектуванні програми використовується фрагмент (підсхема) проекту бази даних для визначення пов'язаних з ним процедурних аспектів - збережених процедур і обумовлених програмно тригерів.

Логіка виконання програми розділена між клієнтським і серверним процесами. Клієнтський процес реалізує більшу частину динаміки кооперативних взаємодій об'єктів, втілених у програмі. Вірний вибір співвідношення між зв'язністю і зв'язаністю об'єктів може допомогти впоратися зі складністю цих взаємодій. Серверний процес, крім іншого, стежить за виконанням бізнес-транзакцій, ініційованих клієнтським процесом.

Зв'язність і ув'язування класів

Наріжним *(Краеугольным)* каменем створення зрозумілих, зручних у супроводі і масштабованих програм є введення ієрархії класів. Щоб уникнути створення об'єктно-орієнтованих програм, які застарівають наступного дня після передачі їх замовникам, необхідно правильно користуватися такими потужними (у

тому числі по своїй руйнівній силі!) засобами, як спадкування і делегування.

Належне проектування програми припускає збалансованість між зв’язністю (cohesion) і ув’язуванням (coupling) класів. Терміни зв'язність і ув'язування були вироблені в рамках методів структурного проектування. Однак, ці терміни мають схожий сенс і значення і для об'єктно-орієнтованого проектування

Під зв’язністю класів розуміють ступінь внутрішнього самовизначення класу. Зв’язність є заходом незалежності класу. Класи, що володіють сильною зв’язністю, виконують одну дію або їх функціонування підпорядковано одній меті. Чим вище рівень зв’язності, тем краще.

Під ув'язуванням класів розуміють, наскільки тісно зв'язані між собою класи. Ув'язування є заходом взаємозалежності класів. Чим нижче рівень ув'язування, тем краще (однак, не слід забувати, що для кооперації класи повинні бути "ув'язані"!).

Зв’язність і ув’язування знаходяться у протиріччі один з одним. Краща зв’язність приводить до погіршення ув’язування і навпаки. Завданням конструктора є досягнення їх найкращої збалансованості. Ріль (Riel) запропонував кілька евристик, присвячених цій проблемі.

Два класи повинні бути або незалежні друг від друга, або один з класів повинен залежати тільки від відкритого інтерфейсу іншого класу.

Атрибути і зв'язані методи повинні знаходитися в одному класі (ця евристика часто порушується класами, що володіють великою кількістю методів відкриття доступу (get, set), визначених у їхньому відкритому інтерфейсі).

Клас повинен охоплювати одну і тільки одну абстракцію. Інформація, що не має відношення до даної абстракції, коли підмножина методів працює на відповідній підмножині атрибутів, повинна бути перенесена в інший клас.

Системна логіка повинна бути розподілена по можливості рівномірно (так, щоб класи були рівномірно завантажені спільною роботою).

Види ув’язування класів

Щоб взаємодіяти, класи повинні бути "ув'язані". Між класом X і класом Y існує ув’язування, якщо клас X може безпосередньо посилатися на клас Y. Пейдж-Джонс (Page-Jones) приводить вісім типів ув'язування класів (які він називає

набором безпосередніх міжкласових посилань).

1. X успадковує від Y.
2. X має атрибут класу Y.
3. X має атрибут шаблону з параметром класу Y.
4. X має метод з вхідним аргументом класу Y.
5. X має метод з вихідним аргументом класу Y.
6. X обізнаний про глобальну змінну класу Y.
7. X обізнаний про метод, що містить локальну змінну класу Y.
6. X - клас, дружній класу Y.

Закон Деметра

Ув'язування класів необхідне для взаємодії об'єктів, однак, воно повинне бути по можливості обмежене рамками рівнів ієрархії класів (тобто зведена до ув’язування усередині рівня). Міжрівнева взаємодія повинна бути зведена до мінімуму і ретельно направлятися. Додаткові керівні принципи для обмеження довільної взаємодії класів сформульовані в законі Деметра (Demeter).

Закон Деметра визначає, на що можуть бути націлені повідомлення в рамках методів класу. Він говорить, що метою повідомлень може бути тільки один з перерахованих нижче об'єктів.

1. Об'єкт, у якому визначений метод (тобто в C++ - це this, в Java - self, а в Smalltalk - super).
2. Об'єкт, який є аргументом сигнатури методу.
3. Об'єкт, на який посилається атрибут об'єкта (включаючи об'єкт, на який посилається один з атрибутів колекції).
4. Об'єкт, створений методом.
5. Об'єкт, на який посилається глобальна змінна.

Для обмеження ув’язування, викликаної спадкуванням, можна обмежити третє правило атрибутами, визначеними в самому класі. Тоді атрибут, успадкований класом, не може використовуватися для позначення цільового об'єкта для повідомлення. Це обмеження відоме як строге правило Деметра.

Методи відкриття доступу і безглузді класи

Атрибути і пов'язані з ними методи повинні перебувати в одному класі. Клас повинен сам "вирішити свою долю". Він може обмежити доступ інших класів до свого стану за рахунок заборони методів відкриття доступу у своєму інтерфейсі. Методи відкриття доступу (accessor methods) визначають операції спостерігача (observer(get)) або мутатора (mutator(set)).

Методи відкриття доступу "відкривають" клас для внутрішніх маніпуляцій з боку іншого класу. Хоча ув’язування і має на увазі певну частку використання об'єкта іншого класу "у своїх інтересах", надмірна поширеність методів відкриття доступу може привести до нерівномірного розподілу "інтелектуального" навантаження на класи. Клас, що містить занадто багато методів, ризикує виявитися безглуздим - інші класи визначають, що для нього "добре".

У світлі сказаного стає ясно, що існують ситуації, коли клас повинен "відкритися" іншим класам. Це має місце щораз, коли необхідно реалізувати деяку політику (або стратегію) "відношень" між двома або більше класами. За прикладом недалеко ходити.

Припустимо, що в нас є два класи INTEGER і REAL і нам необхідно реалізувати "політику" перетворення цілих (integer) чисел у дійсні (real) і навпаки. Для цього потрібно відповісти на деякі питання. У якому з двох класів повинна бути реалізована політика? Чи потрібний нам спеціальний клас-перетворювач CONVERTER для реалізації цієї політики? А якщо ні, то, принаймні, один з класів повинен допускати використання методів відкриття доступу і внаслідок цього може стати "безглуздим" стосовно цієї політики.

Тут доречно буде привести знаменитий вислів Пейдж-Джонса: "На об'єктно-орієнтованої фермі і молоко об'єктно-орієнтоване. Чи означає це, що об'єктно-орієнтована корова повинна відправляти об'єктно-орієнтованому молоку повідомлення `uncow_yourself` ("Звільнитися від корови"), або об'єктно-орієнтоване молоко повинне відправляти об'єктно-орієнтованій корові повідомлення unmilk_yourself ("позбутися молока")" (з виступу Пейдж-Джонса на конференції OOPSLA'87).

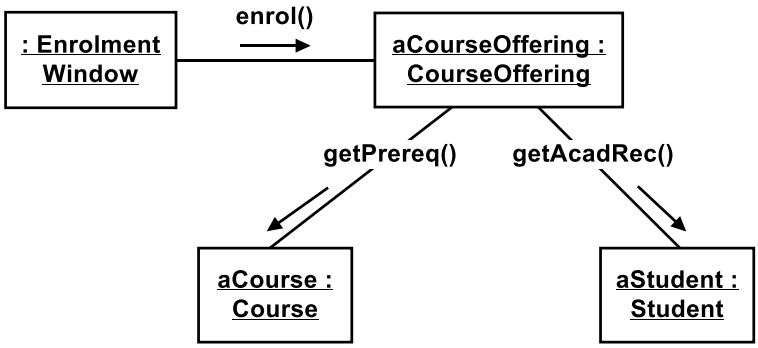

Рис. 4.1. Об'єкт acourseoffering

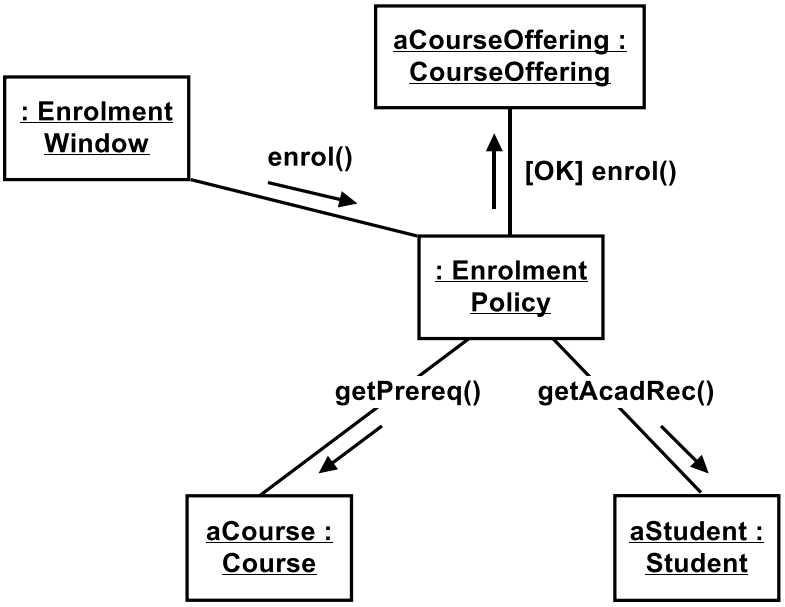

Рис. 4.2. Клас: Enrotmentpoticy

Проектування клієнт-серверних кооперативних взаємодій

Щоб одержати доступ до даних програми, ІС взаємодіють з базами даних. Для доступу до даних у базі даних і їх модифікації клієнтська програма повинна використовувати мову баз даних - зазвичай SQJL. Щоб зрозуміти, яким чином клієнтська програма зв'язується з сервером баз даних, потрібно визнати існування цілого ряду діалектів мови SQL, які, крім того, можна використовувати на різних рівнях програмної абстракції.

На рис. 4.3 показано п'ять різних рівнів SQL-інтефейсу. На рівні 1 SQL використовується як мова визначення даних (data definition language - DDL). DDL -

мова специфікації для визначення структур баз даних (схем баз даних). Основними користувачами мови SQL є проектувальник і адміністратор баз даних (database administrator - DBA).

На рівні 2 SQL використовується як мова маніпулювання даними (data manipulation language - DML) або мова запитів (query language). Термін мова запитів, однак, невірний, оскільки SQL на рівні 2 служить не тільки меті доступу до даних, але і їх модифікації (включаючи операції вставки, відновлення і видалення).

SQL рівня 2 застосовує широке коло користувачів, починаючи з недосвідчених випадкових користувачів і закінчуючи досвідченими DBA. На цьому рівні SQL являє собою мову інтерактивної взаємодії, а це означає, що користувач може сформувати запит поза яким-небудь середовищем програмування і негайно виконати його над базою даних. SQL рівня 2 є відправною точкою у вивченні більш розвинених SQL наступних рівнів.

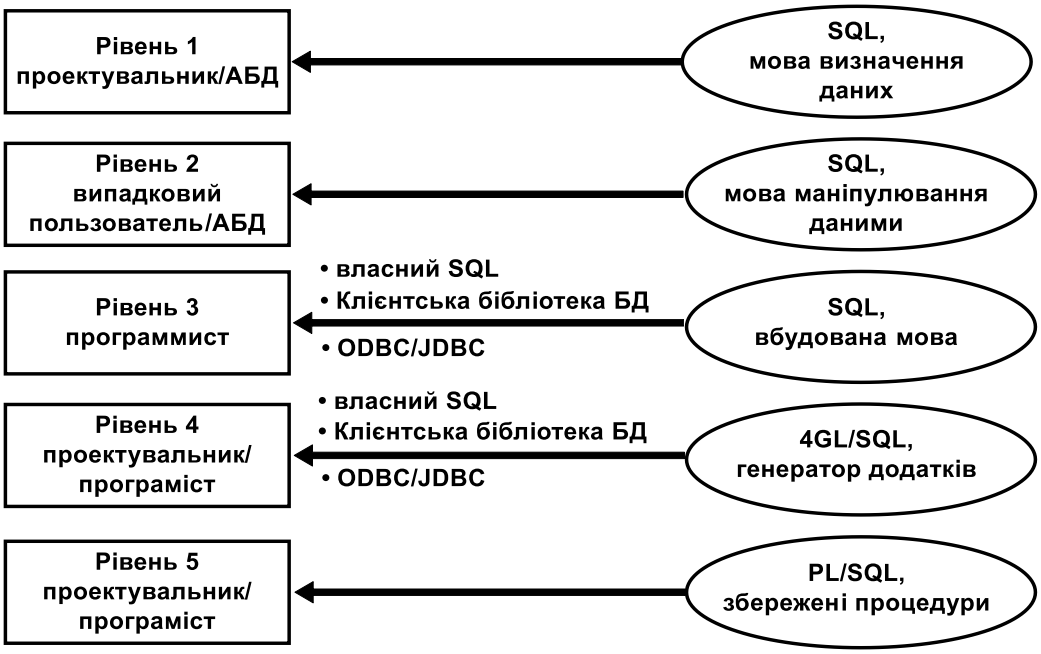

Рис. 4.3. Класифікація рівнів SQL-інтерфейсу

Прикладні програмісти використовують SQL рівні вище 2-го. На цих більш високих рівнях SQL дозволяє працювати в режимі послідовної обробки записів (record-at-a-time processing) на додаток до можливостей послідовної обробки наборів записів (set-at-a-time processing) (єдино доступних) на рівні 2. При роботі в режимі послідовної обробки наборів записів СУБД бере в якості входу для запиту одну або

більше таблиць (набір записів) і повертає в якості виходу таблицю. Хоча це досить потужний засіб, використовувати його в складних запитах не тільки важко, але і небезпечно.

Щоб бути впевненим, що запит повертає вірний результат, програміст повинен мати можливість переглядати одну за одною записи, що вертаються як результат запиту, і приймати рішення щодо того, що робити з кожним окремим записом. Подібна можливість послідовної обробки записів, яка називається курсором (cursor), доступна в SQL рівнів вище 2-го.

SQL рівня 3 вбудований у традиційну мову програмування, на зразок C або Cobol. Оскільки компілятор мови програмування не розуміє SQL, необхідний прекомпілятор (препроцесор) для трансляції SQL-операторів у виклики функцій DB-бібліотеки, що поставляється виробником СУБД. Програміст може віддати перевагу програмі, що безпосередньо використовує функції DB-бібліотеки, при цьому необхідність у прекомпіляторі відпадає.

Одним з найпоширеніших способів забезпечення інтерфейсу клієнтської програми з базами даних є використання стандартів ODBC (Open Database Connectivity - відкритий інтерфейс доступу до баз даних) або/JDBC (Java Database Connectivity - інтерфейс організації доступу Java-додатків до баз даних). Для програмування з використанням цих інтерфейсів вимагається програмний драйвер ODBC або JDBC для визначеної СУБД. Інтерфейси ODBC і JDBC забезпечують стандартну надбудову над мовою SQL, яка за допомогою драйвера транслюється в "рідний" SQL СУБД.

Інтерфейси ODBC/JDBC володіють тією перевагою, що забезпечують автономізацію програми від "рідної" мови SQL СУБД. Якщо в майбутньому потрібне перенесення програми на іншу платформу СУБД, в якості основного прийому здійснення цього процесу служить проста заміна драйверів. Що більш важливо, робота з інтерфейсами ODBC/JDBC дозволяє одному додатку видавати запит більше, ніж до однієї СУБД,

Недоліком стандартів ODBC/JDBC є те, що вони виступають по відношенню SQL "найменшим спільним знаменником". Клієнтський додаток не може скористатися ніякими перевагами спеціальних засобів SQL або розширень, підтримуваних постачальником конкретної СУБД.

SQL рівня 4 використовує ту ж стратегію вбудовування SQL у клієнтські програми, що і SQL рівня 3. Однак, SQL рівня 4 забезпечує більш потужне середовище програмування на основі генератора додатків або графічної мови четвертого покоління (fourth generation language - 4GL). Як правило, мова 4GL поставляється оснащеним засобами побудови екранних зображень конструювання GUI-інтерфейсу. Оскільки додатки IC вимагають розвиненого GUI, для побудови подібних додатків найчастіше вибирають комплект 4GL/SQL.

SQL рівня 5 доповнює SQL рівнів 3 і 4, забезпечуючи можливість переносу деяких SQL-операторів з клієнтської програми на активний (програмувальний) сервер баз даних. SQL використовується в якості мови програмування (PL/SQL). Програми сервера можна викликати зсередини клієнтської програми, як розглядається далі.

Збережені процедури

Збережені процедури (storedprocedure) вперше були введені в СУБД Sybase і зараз становлять невід'ємну частину всіх основних комерційних СУБД. Збережені процедури перетворюють базу даних в активну програмовану систему.

Збережена процедура являє собою розширення SQL, що допускає такі конструкції, як змінні, цикли, розгалуження і оператори присвоювання. Збережені процедури отримують ім'я, можуть приймати вхідні і вихідні параметри, вони компілюються і зберігаються в базі даних. Клієнтська програма може викликати збережену процедуру значною мірою аналогічно тому, як вона викликає підпрограму..

Рис. 4.4 є ілюстрацією переваг виклику клієнтською програмою збереженої процедури замість відправлення повного запиту серверу. Запит, сформований у клієнтській програмі, пересилається серверу по мережі.

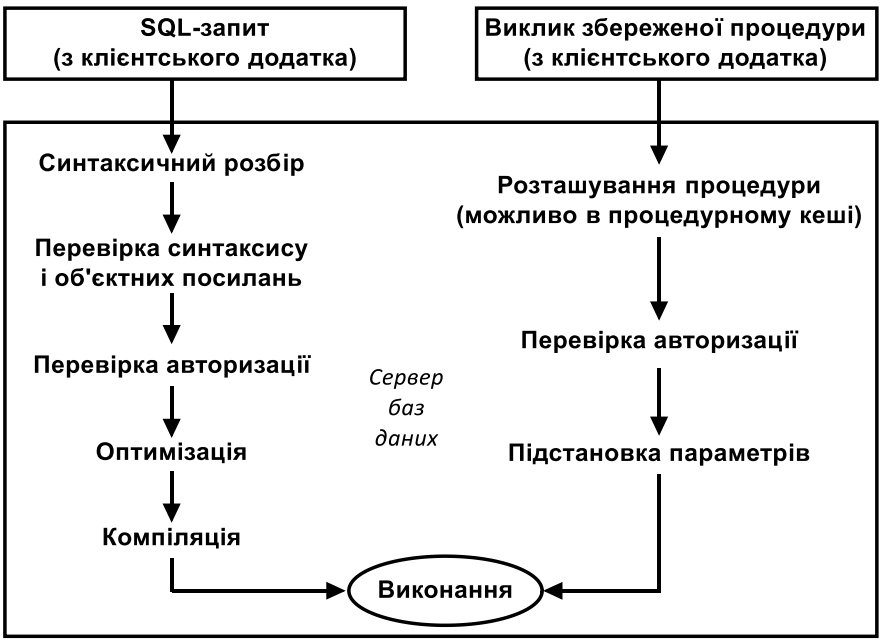

Рис. 4.4. Порівняння виклику з клієнтської програми SQL-запиту і збереженої процедури

Запит може містити синтаксичні помилки і помилки інших типів, але клієнт не в змозі виключити їх - єдине місце, де можна здійснити подібну верифікацію, - система бази даних. Після верифікації СУБД перевіряє, чи володіє викликаюча сторона правами, достатніми для виконання запиту. Якщо це так, то запит оптимізується для визначення найкращого плану доступу до даних. Тільки після цього запит компілюється, виконується, і результат вертається клієнту.

З іншого боку, якщо запит (або весь набір запитів) написаний у вигляді збереженої процедури, він оптимізується і компілюється з наступним збереженням на сервері баз даних. Клієнтській програмі немає необхідності пересилати (можливо, досить об'ємний) запит по мережі - замість цього вона відправляє короткий виклик, що містить ім'я процедури і список фактичних параметрів. Якщо повезе, процедура може розташовуватися в кеш-пам'яті СУБД. Якщо ні, вона переноситься в пам’ять з бази даних.

Повноваження користувача уважно аналізуються так само, як і у випадку SQL-запиту. Усі фактичні параметри підставляються замість формальних параметрів, і збережена процедура виконується. Результат повертається програмі, що викликається.

Як видно з наведеного вище сценарію, збережені процедури дають значно більш ефективний спосіб доступу до бази даних з клієнтської програми. Переваги в продуктивності є наслідком економії мережного трафіка і відсутності необхідності синтаксичного розбору і компіляції щораз при одержанні клієнтського запиту. Що ще більш важливо, супровід збереженої процедури локалізовано в єдиному місці, а викликатися вона може з багатьох клієнтських програм.

Тригери

Тригери являють собою особливий вид збереженої процедури, яку не можна викликати, - вони або запускають себе, або додають (insert), обновляють (update) або видаляють (delete) події з таблиці. Це значить, що кожній таблиці можна зіставити до трьох тригерів. Зазвичай, у деяких системах це так і є (наприклад, Sybase). В інших системах (наприклад, Oracle) виділяються додаткові варіанти подій, що приводить до можливості тримати для кожної таблиці більше трьох тригерів. (Проте, доступність великої кількості типів тригерів не надає більшої виразної сили тригерним програмам).

Тригери можна програмувати для введення деяких бізнес-правил, які застосовуються до бази даних і не можуть бути порушені ні однією клієнтською програмою або інтерактивним DML-Оператором мови SQL. Це означає, що тригери можна використовувати поверх процедурного способу забезпечення обмежень посилальної цілісності. Наприклад, можна створити тригер, який заборонить доступ до бази даних протягом визначених періодів часу або в деякі дні.

Користувач клієнтського додатка може навіть не знати про те, що тригери "очікують", коли в базі даних відбудуться які-небудь модифікації. Якщо модифікації не порушують бізнес-правил, тригери невидимі для програм. Тригер "виявляє" себе користувачу при спробі виконати неприпустиму DML-команду. Він сповіщає користувача про проблему, відображаючи інформаційне повідомлення на екрані програми і відмовляючись виконати DML-операцію.

Проектування транзакцій

Транзакція - це логічна одиниця роботи, яка складається з одного або більше операторів SQL, виконуваних користувачем. Транзакція також є одиницею виміру несуперечності бази даних - стан бази даних після завершення транзакції несуперечливий. Для забезпечення цієї несуперечності використовується програма менеджера транзакцій, яка служить двом цілям: відновленню бази даних (database recovery) і керуванню паралельним виконанням операцій (concurrency control).

Згідно зі стандартами SQL транзакція починається з першого, що виконується SQL-оператора (у деяких системах потрібен явний оператор початку транзакції на зразок begin transaction). Транзакція завершується операторами commit або rollback. Оператор commit записує зміни в базу даних як постійні. Оператор rollback стирає будь-які зміни, зроблені транзакцією.

Транзакція відрізняється властивістю атомарності (atomic) - результат усіх SQL-операторів, що утримуються в транзакції, або повністю підтверджується (commit), або відкидається (rollback). Користувач визначає тривалість (довжину) транзакції. Залежно від характеру бізнес-вимог, області додатка, стилю взаємодій користувача з комп'ютером транзакція може бути зовсім короткою, що складається з одного SQL-оператора, або містити послідовність SQL-операторів.

Песимістичне керування паралельністю

Архітектура традиційних СУБД - ОСУБД становлять тут помітне виключення - орієнтована на короткі транзакції. Ці системи працюють відповідно до алгоритму песимістичного керування паралельністю (pessimistic concurrency control}. При обробці транзакцією кожного постійного об'єкта вона запитує блокування (lock). Існує чотири типи блокувань по об'єктах.

1. Привілейоване блокування (exclusive lock) (по запису) – інші транзакції повинні очікувати, поки транзакція, що утримує подібне блокування, завершиться і звільнить блокування.

2. Блокування відновлення (update lock) (по можливому запису) - інші транзакції можуть читати об'єкт, однак, транзакції, що утримує блокування,

гарантується можливість виконати відновлення в привілейованому режимі, як тільки в неї виникне в цьому потреба.

3. Блокування читання (read lock) (з поділом) - інші транзакції можуть читати і, можливо, одержати блокування відновлення по об'єкту.

4. Відсутність блокування (no lock) - інші транзакції можуть обновляти об'єкт у будь-який момент; підходить тільки для додатків, що допускають "чорнове читання", - тобто транзакція зараховує дані, які можуть бути модифіковані або навіть вилучені (іншою транзакцією) до завершення транзакції.

Точка збереження

Точка збереження (savepoint) - це оператор програми, який ділить довгу транзакцію на більш короткі частини. Іменована точка збереження міститься в стратегічно важливих пунктах програми. Згодом програміст має можливість здійснити відкат виконаної роботи до точки збереження, а не до початку транзакції.

Наприклад, програміст може помістити точку збереження прямо перед операцією update. Якщо операція update завершилася неуспішно, програма виконує відкат до точки збереження і намагається повторити відновлення. Замість цього програма може почати будь-які інші дії, щоб уникнути повного аварійного припинення транзакції.

У програмах великого обсягу точки збереження можна поміщати перед кожною підпрограмою. Якщо підпрограма відмовляє, є можливість здійснити відкат до початків підпрограми і повторити її виконання з виправленими параметрами. При необхідності спеціально спроектовані і запрограмовані підпрограми відновлення можуть виконати очищення, так що транзакція може відновити виконання.

Триггерный відкат

Тригерний відкат (trigger rollback) являє собою особливий вид точки збереження. Тригер можна використовувати для програмування бізнес-правил будь-якої складності. У деяких випадках відкат усієї транзакції неприйнятний, коли тригер відмовляється модифікувати таблиці. (через те, що транзакція намагається

порушити бізнес-правило). Транзакції може знадобитися почати коригувальні дії.

Внаслідок наведених вище причин СУБД може забезпечити можливості програмування тригера для відкату або всієї транзакції, або тільки тригера. В останньому випадку програма (можливо, збережена процедура) може проаналізувати проблему і приймають рішення щодо подальших дій. Навіть якщо необхідно поступово згорнути всю транзакцію, програма буде мати можливості більш ретельної інтерпретації причини помилки і відображення більш осмисленого повідомлення для користувача.

Тестування баз даних

Аналогічно тестуванню GUI-інтерфейсу тестування баз даних зв'язане з багатьма іншими видами тестування. Так, тестування по методу чорного ящика (тестування по відношенню до специфікації) найчастіше засноване на використанні вхідної і вихідної інформації для бази даних. Однак, це не виключає необхідності в окремій методиці тестування баз даних.

Післяреалізаційне тестування баз даних включає всебічне тестування по методу прозорого ящика (відповідно до програмного коду). Найбільш істотною частиною тестування баз даних є тестування транзакцій. Тестування деяких інших аспектів баз даних - продуктивності, паралелізму, перевірка повноважень користувачів - іноді виділяють в окремі групи тестів.

Аналогічно тестуванню GUI-інтерфейсу деякі тести баз даних потрібно проводити повторно для всіх різних функцій додатка. Питання, що вимагають розгляди при тестуванні баз даних, необхідно оформити у вигляді загального документа. Цей загальний документ слід потім додати до опису всіх функціональних тестів (тестів системних послуг). Нижче приводиться зразковий перелік питань, на які слід звернути увагу при тестуванні баз даних.

- Перевірити, чи виконується транзакція належним чином при правильних вхідних даних. Чи коректний зворотний зв'язок системи з GUI-інтерфейсом? Чи коректний вміст бази даних після транзакції?
- Перевірити, чи виконується транзакція належним чином при неправильних вхідних даних. Чи коректний зворотний зв'язок системи з GUI-

інтерфейсом? Чи коректний вміст бази даних після транзакції?

- Перервати транзакцію до її завершення. Чи коректний зворотний зв'язок системи з GUI-інтерфейсом? Чи коректний вміст бази даних після транзакції?
- Запустити ту саму транзакцію паралельно в багатьох процесах. Свідомо змусити одну з транзакцій заблокувати ресурс, необхідний іншим транзакціям. Чи одержують користувачі зрозуміле пояснення ситуації? Чи коректний вміст бази даних після транзакцій?
- Витягти кожний з клієнтських SQL-операторів з клієнтської програми і виконати їх над базою даних в інтерактивному режимі. Чи збігаються отримані результати з очікуваними, а також чи відповідають вони результатам при виконанні SQL-операторів у складі програми?
- Виконати інтерактивне тестування кожного більш складного SQL-запиту (видаваного з клієнтської програми або збереженої процедури) по методу прозорого ящика, включаючи зовнішні з'єднання, об'єднання, підзапити, значення типу null, функції агрегації і т.д.

Тестування авторизації

Тестування авторизації можна розглядати як природнє розширення тестування перших двох типів системних обмежень. Як клієнтські (користувацький інтерфейс), так і серверні (бази даних) об'єкти повинні бути захищені від несанкціонованого використання. Тестування авторизації повинне забезпечити перевірку того, що механізми безпеки, вбудовані в клієнтську і серверну частині системи, у дійсності захищають систему від несанкціонованого проникнення.

Хоча порушення безпеки безумовно позначається на базах даних, захист починається з клієнтської частини системи. Користувацький інтерфейс програми повинен могти динамічно набудовувати власну конфігурацію відповідно до рівня повноважень поточного користувача (який підтверджується (аутентифікується) ідентифікатором і паролем користувача). Пункти меню, командні кнопки і навіть цілі вікна можуть стати недоступні користувачам, якщо в них немає відповідних прав.

Не всі "проходи" у захисті системи можуть бути звернені убік користувачів. Підтримка авторизації є істотним компонентом будь-який СУБД. Користувачу можуть бути надані вибіркові права доступу до серверних об'єктів, при цьому права (повноваження) користувача по відношенню до сервера сервера розпадаються на дві категорії.

Доступ до окремих серверних об'єктів (таблицям, представленням, стовпцям, збереженим процедурам і т.д.).

Виконання SQL-операторів (select, update, insert, delete і т.д.). Повноваження для користувачів можна призначати безпосередньо на рівні користувачів або на груповому рівні. Групи дозволяють адміністратору системи захисту призначати права доступу групі користувачів за допомогою однократного введення відповідних параметрів. Користувач може як не належати до жодної групи, так і належати до декількох груп.

Для забезпечення більшої гнучкості при роботі з авторизацією більшість СУБД вводять додатковий рівень авторизації - рольовий рівень. Ролі дозволяють адміністратору системи захисту призначати права доступу всім користувачам, які відіграють певну роль в організації. Ролі можуть носити вкладений характер, тобто права, надані різним рольовим іменам, можуть перекриватися.

Для великих додатків ІС проектування авторизації вимагає скрупульозної роботи. Найчастіше поряд з прикладною базою даних створюється база даних авторизації для зберігання і маніпулювання клієнтськими і серверними повноваженнями. Після входу користувача в систему прикладна програма звертається до бази даних, щоб установити рівень повноважень користувача і настроїти свою конфігурацію по відношенню до цього користувача.

Будь-які зміни в правах доступу до баз даних здійснюються через базу даних авторизації - тобто ніхто, включаючи адміністратора системи захисту, не має можливості безпосередньо змінити права доступу до бази даних без попереднього відновлення бази даних авторизації. На рис. 4.5 наведений приклад проекту бази даних авторизації.

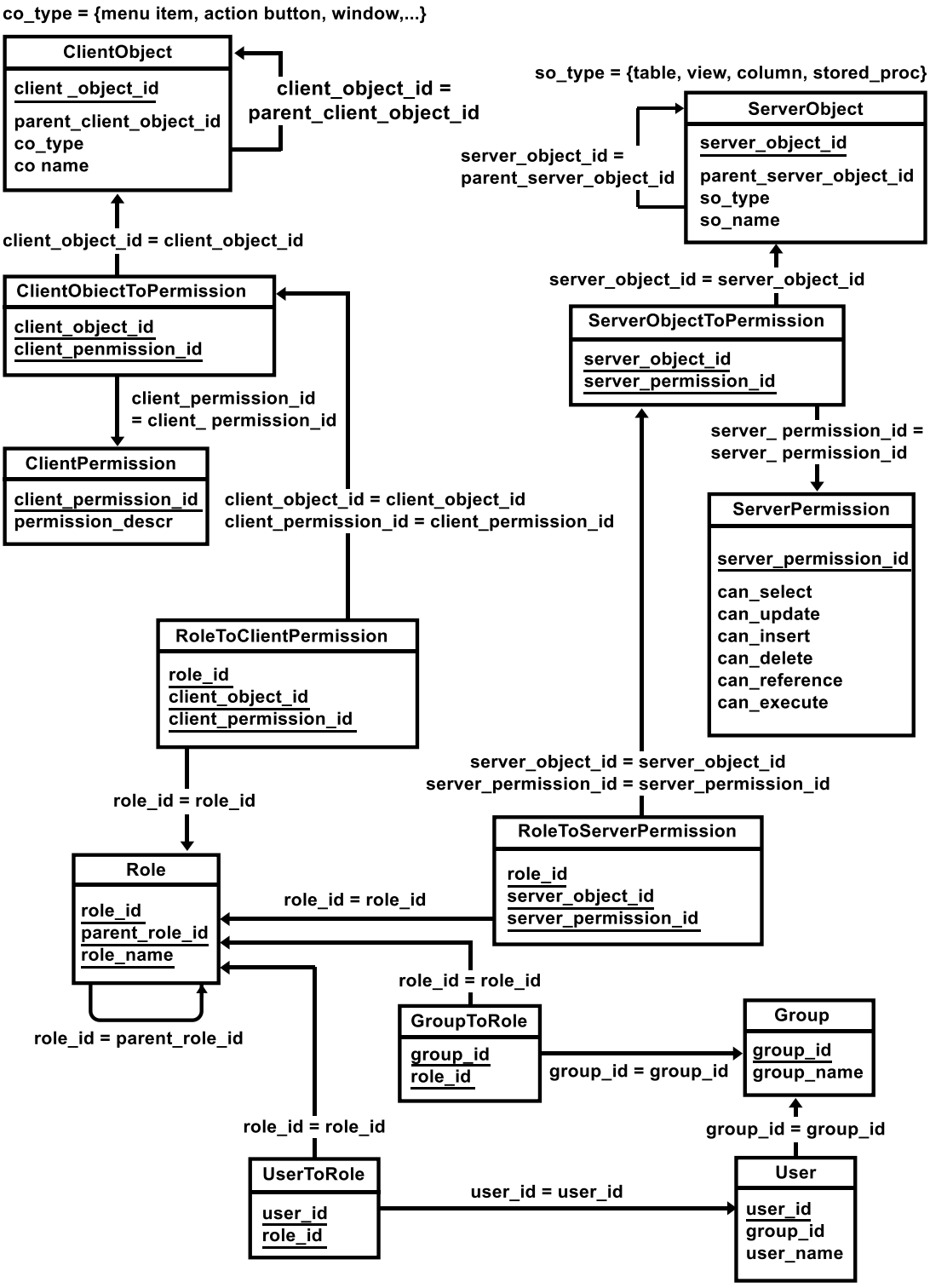

Рис. 4.5. Проект бази даних авторизації

Тестування інших обмежень

Крім описаних вище тестування системних обмежень включає наступні види тестування.

- Тестування продуктивності.
- Тестування в обтяжувальному режимі.

- Тестування при відмові.
- Конфігураційне тестування.
- Інсталяційне тестування.

Тестування продуктивності спрямовано на вимір обмежень продуктивності, необхідних замовником. Обмеження пов'язані зі швидкістю транзакцій і пропускною здатністю. Тестування проводиться при різному робочому завантаженні систем, включаючи передбачуване пікове завантаження. Тестування продуктивності є важливого складового настроювання системи.

Тестування в обтяжувальному режимі проектується таким чином, щоб вивести зі стоя систему при пред'явленні до неї завищених вимог - ерез нестачу ресурсів, незвичайної конкуренції за ресурси, непередбачуваної частоти, величини або обсягу вимог до ресурсів. Тестування в обтяжувальному режимі часто поєднується з тестуванням продуктивності і може вимагати відповідного до апаратного і програмного оснащення.

Тестування при відмові спрямоване на вивчення реакції системи на різні апаратні, мережні або програмні збої. Цей вид тестування тісно пов'язаний з процедурами відновлення, підтримуваними СУБД.

Конфігураційне тестування пов'язане з перевіркою функціонування системи при різній апаратної і програмної конфігурації. Для більшості виробничих середовищ передбачається, що система здатна функціонувати на різних клієнтських робочих станціях, які підключаються до бази даних з використанням різних мережних протоколів. На клієнтських робочих станціях може бути інстальоване різне ПЗ (наприклад, драйвери), яке може конфліктувати з передбаченими установками.

Інсталяційне тестування є розширенням конфігураційного тестування. Воно пов'язане з перевіркою належного функціонування системи на кожній з платформ, на яких вона інсталюється. Це означає фактичне повторення тестування системних послуг.

Документація по тестуванню і керуванню змінами

Документація по тестуванню і керуванню змінами становить невід'ємну частину іншої системної документації, включаючи документацію по прецедентах, рис. 4.6. Системні функції, визначені в моделі бізнес-прецедентів, можна використовувати для написання первісного тест-плану.

Потім модель прецедентів використовується для написання документації по тест-прецедентах і визначення тестових вимог. Дефекти, виявлені під час тестування, фіксуються в документації по дефектах. У документації по вдосконаленню відображаються всі нереалізовані вимоги-прецеденти. При використанні CASE-засобів у розробників існують наступні можливості по створенню документації.

- Розробка описових документів і їх наступне використання для створення вимог (тестові вимоги, вимоги-прецеденти і т.д.) у CASE-репозиторії.
- Використання CASE-засобів для введення вимог в репозиторій і подальша генерація документації на їх основі

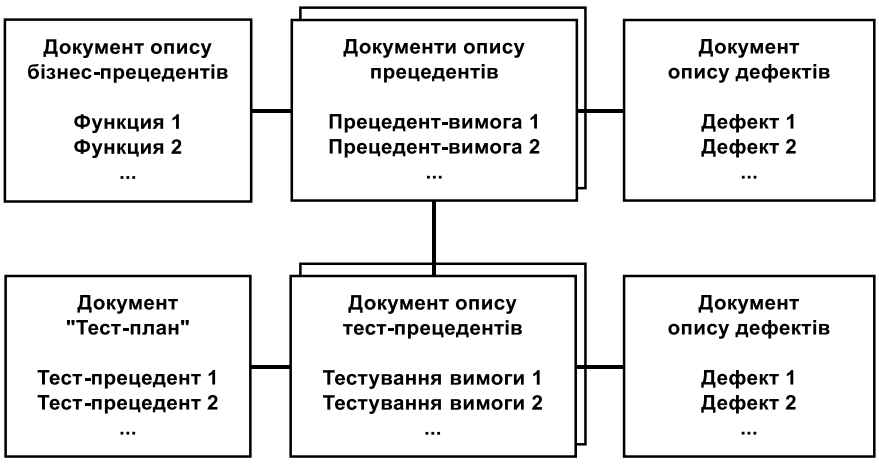

Рис. 4.6. Документація по тестуванню і керуванню змінами

Завдання: Виконати повне проектування програмної системи. Виконати проектування тестуючої системи.

Надати звіт, що містить результати проектування програмної системи

Контрольні питання:

1. Який вплив на проектування роблять принципи, пов'язані зі зв'язністю і ув'язуванням?
2. Які об'єкти можуть виступати як цільові об'єкти для повідомлень згідно із законом Деметра?
3. Коротко опишіть п'ять рівнів SQL - інтерфейсів.
4. У чому перевага виклику із клієнтської програми збереженої процедури в порівнянні з SQL - запитом пересилається базі даних? Існують ситуації, при яких ми змушені використовувати SQL - запит замість виклику вилученої процедури?
5. Коротко опишіть види блокувань при песимістичнім керуванні паралельністю.
6. Що таке точка збереження? Як її можна використовувати при проектуванні програми?
7. Які дії можливі у відповідь на відправлений запит на зміни?

Список літератури

1. Смірнов В.В. Технологія проектування програмних систем. Лекції / В.В. Смірнов, Н.В. Смірнова. – Кіровоград: КНТУ.

2. Лешек А. Мацяшек. Анализ и проектирование информационных систем с помощью UML 2.0 / Лешек А. Мацяшек. – М.: Вильямс, 2008. – 816 с.

3. Шалыто А.А. SWITCH - технология. Алгоритмизация и программирование задач логического управления / А.А. Шалыто. – СПб.: Наука, 1998. – 628 с.

4. Карло Гецци. Основы инженерии программного обеспечения / Карло Гецци, Мехди Джазайери, Дино Мандриоли. – СПб.: БХВ-Петербург, 2005. – 832 с.

5. Соммервилл, Иан. Инженерия программного обеспечения, 6-е издание, пер. с англ.
А.А. Минько. – М.: Издательский дом "Вильямс", 2002. – 624 с.

6. Эдвард Йордон. Объектно-ориентированный анализ и проектирование систем / Эдвард Йордон, Карл Аргила. – М.: Лори, 2010. – 264 с.

7. Эрик Эванс. Предметно – ориентированное проектирование (DDD). Структуризация сложных программных систем / Эрик Эванс, пер. с англ. В. Бродов. – К.: Вильямс, 2010. – 448 с.

8. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. пер. с англ. А. Слинкин. – К.: Питер, 2007. – 366 с.

9. Joey F. George. Object-Oriented Systems Analysis and Design. [Joey F. George, Dinesh Batra, Joseph S. Valacich, Jeffrey A. Hoffer]; (2nd Edition). – Prentice Hall; 2 edition (October 27, 2006). – 550 p.

10. Noushin Ashrafi. Object Oriented Systems Analysis and Design / Noushin Ashrafi, Hessam Ashrafi. – Prentice Hall; 1 edition (September 20, 2008). – 648 p.

11. Michele Lanza. Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems / Michele Lanza, Radu Marinescu. – Springer; Softcover reprint of hardcover 1st ed. 2006 edition (December 2, 2010). – 220 p.

12. Grady Booch. Object-Oriented Analysis and Design with Applications (3rd

Edition) / [Grady Booch, Robert A. Maksimchuk, Michael W. Engel, Bobbi J. Young, Jim Conallen, Kelli A. Houston. – Addison-Wesley Professional; 3 edition (April 30, 2007). – 720 p.

13. Jeffrey Whitten. Systems Analysis and Design Methods / Jeffrey Whitten, Lonnie Bentley. – McGraw-Hill/Irwin; 7th edition (November 22, 2005). – 768 p.

14. Alan Dennis. Systems Analysis and Design / Alan Dennis, Barbara Haley Wixom, Roberta M. Roth. – Wiley; 4 edition (December 10, 2008). – 576 p.