Основні відомості про мову UML

Найкращий засіб – це велика діаграма, приколота до стіни.
Даг Скотт

1.1. Цілі та історія створення мови UML

Уніфікована мова моделювання UML (Unified Modeling Language)- це наступник того покоління методів об'єктно-орієнтованого аналізу та проектування, які з'явилися наприкінці 80-х та на початку 90-х років. Створення UML фактично почалося наприкінці 1994 р., коли Граді Буч та Джеймс Рамбо розпочали роботу щодо об'єднання їх методів Booch [Буч-1999] та OMT (Object Modeling Technique) під егідою компанії Rational Software. До кінця 1995 вони створили першу специфікацію об'єднаного методу, названого ними Unified Method, версія 0.8. Тоді ж у 1995 р. до них приєднався автор методу OOSE (Object-

Oriented Software Engineering) Івар Якобсон. Таким чином, UML є прямим поєднанням та уніфікацією методів Буча, Рамбо та Якобсона, проте доповнює їх новими можливостями.

UML знаходиться в процесі стандартизації, що проводиться консорціумом OMG (Object Management Group), в даний час він прийнятий як стандартна мова моделювання і отримав широку підтримку. UML використаний практично всіма найбільшими компаніями - виробниками програмного забезпечення (Microsoft, IBM, Hewlett-Packard, Oracle, Sybase та ін.). Крім того, практично всі світові виробники CASE-засобів, окрім Rational Software (Rational Rose), підтримують UML у своїх продуктах (Paradigm Plus (CA), System Architect (Popkin Software), Microsoft Visual Modeler та ін.). Повний опис UML можна знайти на сайтахhttp://www.omg.org ihttp://www.rational.com.

1.2 Засоби UML

Автори UML представляють його як мову для визначення, представлення, проектування та документування програмних систем, організаційно-економічних систем, технічних систем та інших систем різної природи. UML містить стандартний набір діаграм та нотацій найрізноманітніших видів. Стандарт UML версії 1.1, прийнятий OMG 1997 р., пропонує наступний набір діаграм для моделювання:

- **діаграми варіантів використання (use case diagrams)**- для моделювання бізнес-процесів організації та вимог до створюваної системи);
- **діаграми класів (class diagrams)**- для моделювання статичної структури класів системи та зв'язків між ними;
- діаграми поведінки системи (behavior diagrams):
  - діаграми взаємодії (interaction diagrams):
    - ♦ діаграми послідовності (sequence diagrams) та
    - ♦ **кооперативні діаграми (collaboration diagrams)**- Для моделювання процесу обміну повідомленнями між об'єктами;
  - **діаграми станів (statechart diagrams)**- для моделювання поведінки об'єктів системи під час переходу з одного стану до іншого;
  - **діаграми діяльності (activity diagrams)**- для моделювання поведінки системи в рамках різних варіантів використання, або моделювання діяльності;
- діаграми реалізації (implementation diagrams):
  - **діаграми компонентів (component diagrams)**- Для моделювання ієрархії компонентів (підсистем) системи;
  - **діаграми розміщення (deployment diagrams)**- Для моделювання фізичної архітектури системи.

1.3 Діаграми варіантів використання

Поняття варіанта використання (use case) вперше запровадив Івар Якобсон і надав йому такої значущості, що в даний час варіант використання перетворився на основний елемент розробки та планування проекту.

Варіант використанняє послідовністю дій (транзакцій), що виконуються системою у відповідь на подію, що ініціюється деяким зовнішнім об'єктом (дійовою особою). Варіант використання описує типову взаємодію між користувачем та системою. У найпростішому випадку варіант використання визначається процесі обговорення з користувачем тих функцій, які він хотів би реалізувати.

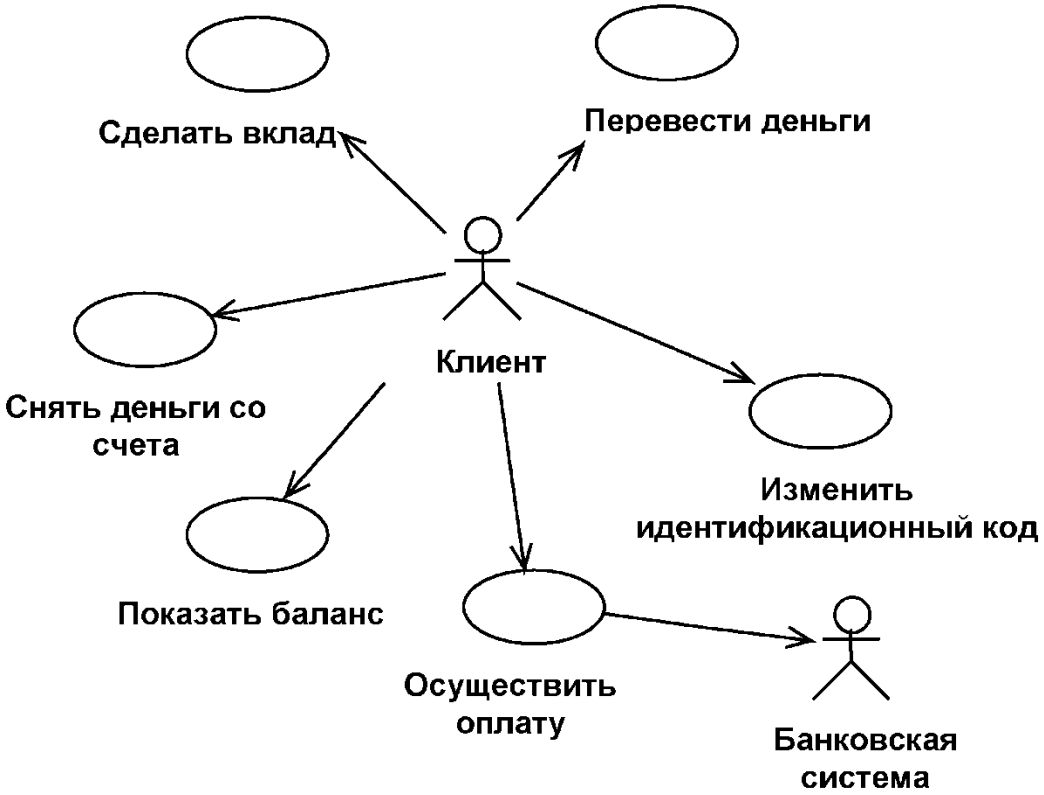
Чинна особа(Actor) - це роль, яку користувач грає по відношенню до системи. Діючі особи є ролі, а чи не конкретних людей чи найменування робіт. Незважаючи на те, що на діаграмах варіантів використання вони зображуються у вигляді стилізованих людських фігурок, дійова особа може бути зовнішньою системою, якій необхідна деяка інформація від даної системи. Показувати на діаграмі дійових осіб слід лише тому випадку, коли їм дійсно необхідні деякі варіанти використання.

Діючі особи діляться на три основні типи - користувачі системи, інші системи, що взаємодіють з цією, та час. Час стає дійовою особою, якщо від неї залежить запуск будь-яких подій у системі.

Для наочного подання варіантів використання як основних елементів процесу розробки програмного забезпечення (ПЗ) застосовуються діаграми варіантів використання. На рис. 1.1 показано приклад такої діаграми для банкомату (Automated Teller Machine, ATM).

На даній діаграмі людські фігурки позначають дійових осіб, овали - варіанти використання, а лінії та стрілки - різні зв'язки між дійовими особами та варіантами використання.

На цій діаграмі показані дві дійові особи: клієнт та кредитна система. Існує також шість основних дій, що виконуються системою, що моделюється: перевести гроші, зробити вклад, зняти гроші з рахунку, показати баланс, змінити ідентифікаційний код і здійснити оплату.



Мал. 1.1. Приклад діаграми варіантів використання

На діаграмі варіантів використання показано взаємодію між варіантами використання та дійовими особами. Вона відображає вимоги до системи з погляду користувача. Таким чином, варіанти використання – це функції, що виконуються системою, а дійові особи – це зацікавлені особи (stakeholders) по відношенню до створюваної системи. Такі діаграми показують, які дійові особи ініціюють варіанти використання. З них також видно, коли дійова особа одержує інформацію від варіанта використання. Дана діаграма, наприклад, відображає взаємодію між варіантами використання та дійовими особами системи АТМ. Власне, діаграма варіантів використання ілюструє вимоги до системи. У нашому прикладі клієнт банку ініціює велику кількість різних варіантів використання: «Зняти гроші з рахунку», «Перевести гроші», «Зробити вклад», «Показати баланс» та «Змінити ідентифікаційний код». Від варіанта використання "Здійснити оплату" стрілка вказує на Банківську систему. Чинними особами можуть бути зовнішні системи, і тому в даному випадку банківська система показана саме як дійова особа - вона зовнішня для системи АТМ. Спрямована від варіанту використання до дійової особи стрілка показує, що варіант використання надає деяку інформацію, що використовується дійовою особою. У разі варіант використання «Здійснити оплату» надає Банківській системі інформацію про оплату за кредитною карткою.

Всі варіанти використання, так чи інакше, пов'язані із зовнішніми вимогами до функціональності системи. Варіанти використання завжди слід аналізувати разом із

дійовими особами системи, визначаючи при цьому реальні завдання користувачів та розглядаючи альтернативні способи вирішення цих завдань.

Діючі особи можуть відігравати різні ролі по відношенню до варіанта використання. Вони можуть скористатися його результатами чи можуть самі у ньому брати участь. Значимість різних ролей дійової особи залежить від цього, як використовуються її зв'язку.

Конкретна мета діаграм варіантів використання - це документування варіантів використання (все, що входить у сферу застосування системи), дійових осіб (все поза цією сферою) та зв'язків між ними. Розробляючи діаграми варіантів використання, намагайтеся дотримуватись наступних правил:

- Не моделюйте зв'язок між дійовими особами. За визначенням дійові особи перебувають поза сферою дії системи. Це означає, що зв'язок між ними також не належать до її компетенції.
- Не сполучайте суцільною стрілкою (комунікаційним зв'язком) два варіанти використання безпосередньо. Діаграми цього типу описують лише, які варіанти використання доступні системі, а чи не порядок їх виконання. Для відображення порядку виконання варіантів використання застосовують діаграми діяльності.
- Варіант використання має бути ініційований дійовою особою. Це означає, що має бути суцільна стрілка, що починається на дійовій особі та закінчується на варіанті використання.

Хорошим джерелом для ідентифікації варіантів використання є зовнішні події. Слід розпочати з переліку всіх подій, що відбуваються у зовнішньому світі, на які система має якимось чином реагувати. Будь-яка конкретна подія може спричинити реакцію системи, яка не потребує втручання користувачів, або, навпаки, викликати реакцію користувача. Ідентифікація подій, на які потрібно реагувати, допомагає ідентифікувати варіанти використання.

Варіанти використання починають описувати, що має робити система. Щоб фактично розробити систему, проте, будуть потрібні більш конкретні деталі. Ці деталі описуються в документі, який називається «потік подій» (flow of events). Метою потоку подій є документування процесу обробки даних, що реалізується в рамках варіанта використання. Цей документ докладно описує, що робитимуть користувачі системи, і що сама система.

Хоча потік подій і описується докладно, він також повинен залежати від реалізації. Мета - описати, що робитиме система, а не як вона робитиме це. Зазвичай потік подій включає:

- короткий опис;
- передумови (pre-conditions);
- основний потік подій;

- альтернативний потік подій (або кілька альтернативних потоків);
- постумови (post-conditions).

Послідовно розглянемо ці складові.

Опис

Кожен варіант використання повинен мати пов'язаний з ним короткий опис того, що він робитиме. Наприклад, варіант використання «Перевести гроші» системи АТМ може містити такий опис:

Варіант Використання «Перевести гроші» дозволяє клієнту або службовцю банку переказувати гроші з одного рахунку до запитання або ощадного рахунку на інший.

Передумови

Передумови варіанта використання - це умови, які мають бути виконані, як варіант використання почне виконуватися сам. Наприклад, такою умовою може бути виконання іншого варіанта використання або наявність у користувача прав доступу, потрібних для запуску цього. Не всі варіанти використання бувають попередні умови.

Раніше згадувалося, що діаграми варіантів використання повинні відображати порядок їх виконання. За допомогою передумов можна документувати і таку інформацію. Наприклад, передумовою одного варіанта використання може бути те, що в цей час має виконуватись інший.

Основний та альтернативний потоки подійКонкретні деталі варіантів використання описуються переважно і альтернативних потоках подій. Потік подій поетапно визначає, що має відбуватися під час виконання закладеної у варіанти використання функціональності. Потік подій приділяє увагу тому, що робитиме система, а не як вона робитиме це, причому описує все це з погляду користувача. Основний та альтернативний потоки подій включають наступний опис:

- спосіб запуску варіанта використання;
- різні шляхи виконання варіанта використання;
- нормальний, чи основний, потік подій варіанти використання;
- відхилення від основного потоку подій (так звані альтернативні потоки);
- потоки помилок;
- спосіб завершення варіанта використання.

Наприклад, потік подій варіанта використання «Зняти гроші» може виглядати так:

Основний потік

1 . Варіант використання починається, коли клієнт вставляє картку в АТМ.

2. АТМ виводить вітання та пропонує клієнту запровадити свій персональний ідентифікаційний номер.

3. Клієнт вводить номер.

4. АТМ підтверджує введений номер. Якщо номер не підтверджено, виконується альтернативний потік подій А1.

5. АТМ виводить список доступних дій:

- покласти гроші на рахунок;
- зняти гроші з рахунку;
- переказати гроші.

6. Клієнт обирає пункт "Зняти гроші".

7. АТМ просить, скільки грошей треба зняти.

8. Клієнт запроваджує необхідну суму.

9. АТМ визначає, чи є на рахунку достатньо грошей. Якщо грошей недостатньо, виконується альтернативний потік А2. Якщо під час підтвердження суми виникають помилки, виконується потік помилок Е1.

1 0. АТМ віднімає необхідну суму з рахунку клієнта.

11. АТМ видає клієнту необхідну суму готівкою.

1 2. АТМ повертає клієнту його картку.

13. АТМ друкує чек клієнта.

1 4. Варіант використання завершується.

Альтернативний потік А1. Введіть неправильний ідентифікаційний номер.

1 . АТМ повідомляє клієнта, що ідентифікаційний номер введений неправильно.

2. АТМ повертає клієнту його картку.

3. Варіант використання завершується.

Альтернативний варіант використання А2. Недостатньо грошей на рахунку.

1. АТМ інформує клієнта, що грошей на його рахунку недостатньо.

2. АТМ повертає клієнту його картку.

3. Варіант використання завершується.

Потік помилок Е1. Помилка в підтвердженні суми, що запитується.

1. АТМ повідомляє користувачеві, що при підтвердженні суми, що запитується, сталася помилка і дає йому номер телефону служби підтримки клієнтів банку.

2. АТМ заносить відомості про помилку до журналу помилок. Кожен запис містить дату та час помилки, ім'я клієнта, номер його рахунку та код помилки.

3. АТМ повертає клієнту його картку.

4. Варіант використання завершується.

Постуслів'я

Постуслів називаються такі умови, які завжди повинні бути виконані після завершення варіанту використання. Наприклад, наприкінці варіанта використання можна позначити

прапорцем якийсь перемикач. Інформація такого типу входить до складу постумов. Як і передумов, за допомогою постумов можна вводити інформацію про порядок виконання варіантів використання системи. Якщо, наприклад, після одного з варіантів використання завжди повинен виконуватися інший, це можна описати як постумову. Такі умови є у кожного варіанта використання.

Зв'язки між варіантами використання та дійовими особами

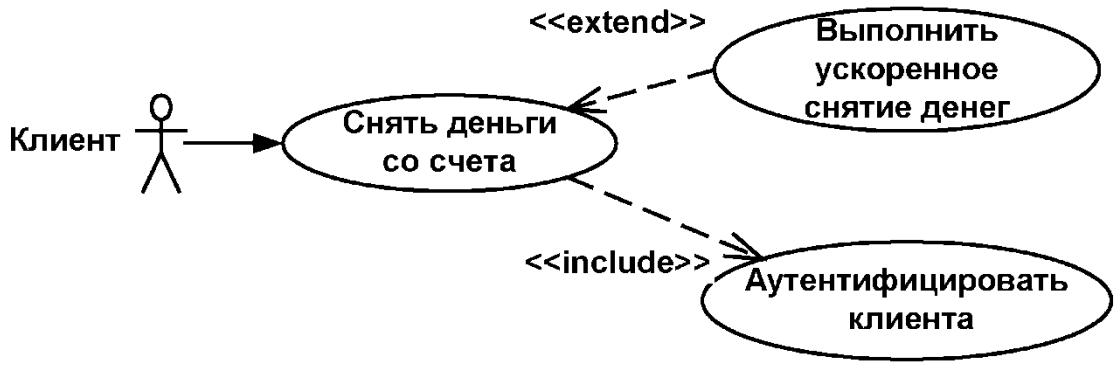
У UML на діаграмах варіантів використання підтримується кілька типів зв'язків між елементами діаграми. Це зв'язки комунікації (communication), включення (include), розширення (extend) та узагальнення (generalization).

Зв'язок комунікації - це зв'язок між варіантом використання та дійовою особою. Мовою UML зв'язку комунікації показують за допомогою односпрямованої асоціації (суцільної лінії зі стрілкою). Напрямок стрілки дозволяє зрозуміти хто ініціює комунікацію.

Зв'язок включення застосовується у тих ситуаціях, коли є якийсь фрагмент поведінки системи, який повторюється більш ніж одному варіанті використання. За допомогою таких зв'язків зазвичай моделюють функціональність, що багато разів використовується. У прикладі АТМ варіанти використання «Зняти гроші» та «Покласти гроші на рахунок» мають упізнати (аутентифікувати) клієнта та його ідентифікаційний номер перед тим, як допустити здійснення самої транзакції. Замість того, щоб детально описувати процес аутентифікації для кожного з них, можна помістити цю функціональність у свій власний варіант використання під назвою «Аутентифікувати клієнта».

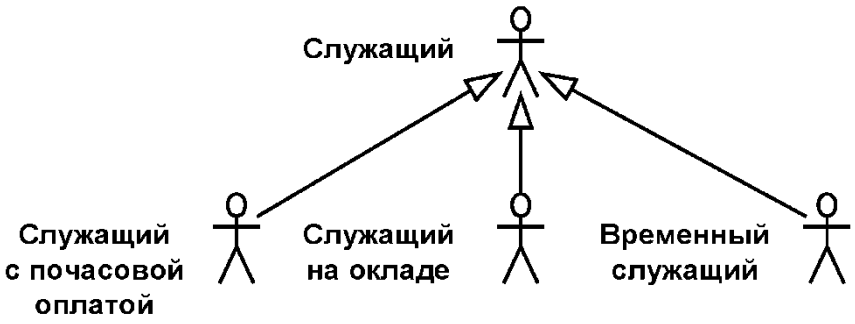
Зв'язок розширення застосовується при описі змін у нормальній поведінці системи. Вона дозволяє варіанту використання лише за необхідності використовувати функціональні можливості іншого.

Мовою UML зв'язку включення та розширення показують у вигляді залежностей з відповідними стереотипами, як показано на рис. 1.2.



Мал. 1.2. Зв'язки використання та розширення

З допомогою зв'язку узагальнення показують, що з кількох дійових осіб є спільні риси. Наприклад, клієнти можуть бути двох типів: корпоративні та індивідуальні. Цей зв'язок можна моделювати за допомогою нотації, показаної на рис. 1.3.



Мал. 1.3. Узагальнення дійової особи

Немає потреби завжди створювати зв'язки цього типу. Загалом, вони потрібні, якщо поведінка дійової особи одного типу відрізняється від поведінки іншого остільки, оскільки це стосується системи. Якщо обидва підтипи використовують одні й самі варіанти використання, показувати узагальнення дійової особи не потрібно.

Варіанти використання є необхідним засобом на стадії формування вимог до ПЗ. Кожен варіант використання - це потенційна вимога до системи, і доки вона не виявлена, неможливо запланувати її реалізацію.

1.4. Діаграми взаємодії

Діаграми взаємодії (interaction diagrams) описують поведінку взаємодіючих груп об'єктів.

Як правило, діаграма взаємодії охоплює поведінку об'єктів у межах лише одного варіанта використання. На такій діаграмі відображається ряд об'єктів і повідомлення, якими вони обмінюються між собою.

Повідомлення (message)- це засіб, з допомогою якого об'єкт-відправник запитує в об'єкта одержувача виконання однієї з його операцій.

Інформаційне (informative) повідомлення- це повідомлення, що забезпечує об'єкт-одержувач деякою інформацією оновлення його стану.

Повідомлення-запит (interrogative)- це повідомлення, що просить видачу деякої інформації про об'єкт-одержувача.

Імперативне (imperative) повідомлення- це повідомлення, що запитує об'єкта-одержувача виконання деяких дій.

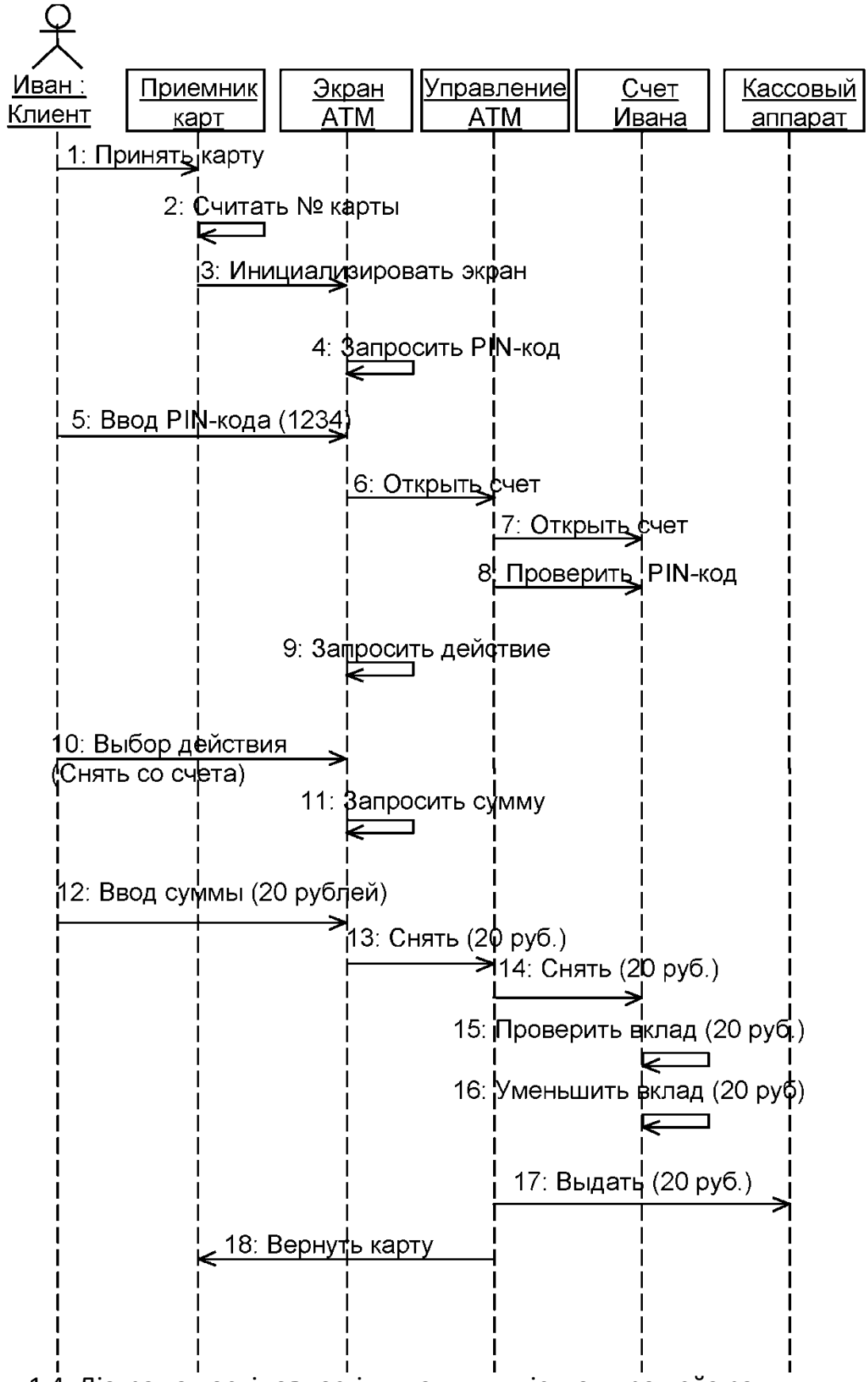
Існує два види діаграм взаємодії: діаграми послідовності (sequence diagrams) та кооперативні діаграми (collaboration diagrams).

1.4.1. Діаграми послідовності

Діаграми послідовності відбивають потік подій, які у рамках варіанта використання. Наприклад, варіант використання «Зняти гроші» передбачає кілька можливих послідовностей, такі як зняття грошей, спроба зняти гроші, не маючи їх достатньої кількості на рахунку, спроба зняти гроші за неправильним ідентифікаційним номером та деякі інші. Нормальний сценарій зняття грошей з рахунку (за відсутності таких проблем, як неправильний ідентифікаційний номер або нестача грошей на рахунку), показаний на рис. 1.4.

Ця діаграма послідовності показує потік подій у рамках варіанта використання "Зняти гроші". Усі дійові особи показані у верхній частині діаграми; у наведеному вище прикладі зображено дійову особу Клієнт. Об'єкти, необхідні системі для виконання варіанта використання "Зняти гроші", також представлені у верхній частині діаграми. Стрілки відповідають повідомленням, що передаються між дійовою особою та об'єктом або між об'єктами для виконання потрібних функцій.

На діаграмі послідовності об'єкт зображується у вигляді прямокутника, від якого вниз проведено пунктирну вертикальну лінію. Ця лінія називається лінією життя (lifeline) об'єкта. Вона є фрагментом життєвого циклу об'єкта у процесі взаємодії.



Мал. 1.4. Діаграма послідовності для зняття клієнтом грошейз рахунку

Кожне повідомлення представляється як стрілки між лініями життя двох об'єктів. Повідомлення з'являються в порядку, показаному на сторінці зверху вниз. Кожне повідомлення позначається щонайменше ім'ям повідомлення; при бажанні можна додати

також аргументи і деяку керуючу інформацію, і, крім того, можна показати само-делегування (self-delegation) - повідомлення, яке об'єкт посилає самому собі, при цьому стрілка повідомлення вказує на ту саму лінію життя.

Хороший спосіб початкового виявлення деяких об'єктів - це вивчення іменників у потоці подій. Також можна прочитати документи, що описують конкретний сценарій. Під сценарієм розуміється конкретний екземпляр потоку подій. Потік подій для варіанта використання "Зняти гроші" говорить про людину, яка знімає деяку суму грошей з рахунку за допомогою АТМ.

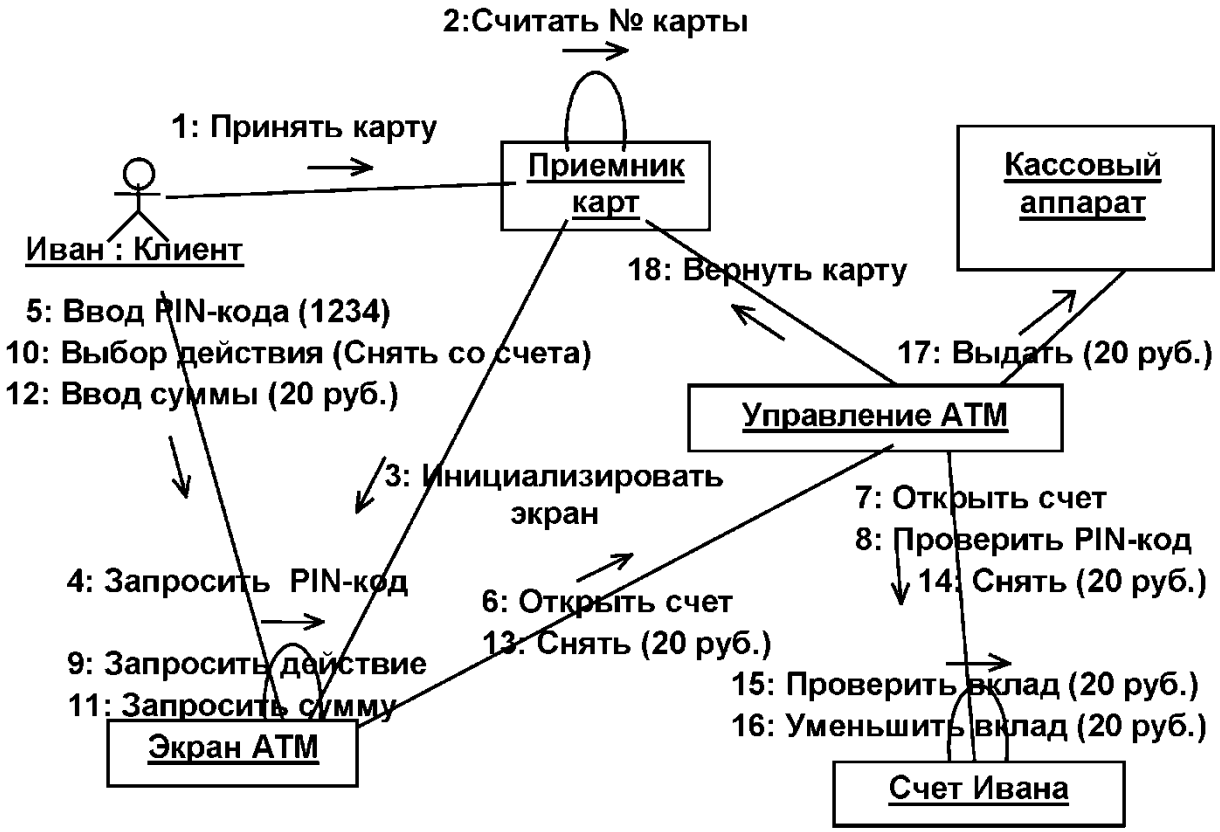
Не всі об'єкти з'являються у потоці подій. Там, наприклад, може бути форм для заповнення, але їх необхідно показати на діаграмі, щоб дозволити дійовій особі ввести нову інформацію у систему чи переглянути її. У потоці подій, швидше за все, також не буде і об'єктів, що управляють (control objects). Ці об'єкти керують послідовністю потоку у варіанті використання.

1.4.2. Кооперативні діаграми

Другим видом діаграми взаємодії є кооперативна діаграма.

Подібно до діаграм послідовності, кооперативні діаграми (collaborations) відображають потік подій через конкретний сценарій варіанта використання. Діаграми послідовності впорядковано за часом, а кооперативні діаграми більше уваги загострюють на зв'язках між об'єктами. На рис. 1.5 наведено кооперативну діаграму, яка описує, як клієнт знімає гроші з рахунку.

Як видно з малюнка, тут представлена вся та інформація, яка була і на діаграмі послідовності, але кооперативна діаграма по-іншому описує потік подій. З неї легше зрозуміти зв'язок між об'єктами, проте, важче усвідомити послідовність подій.



Мал. 1.5. Кооперативна діаграма, що описує процес зняття клієнтом грошей зі свого рахунку

Тому часто для будь-якого сценарію створюють діаграми обох типів. Хоча вони служать однієї і тієї ж мети і містять одну й ту саму інформацію, але представляють її з різних точок зору.

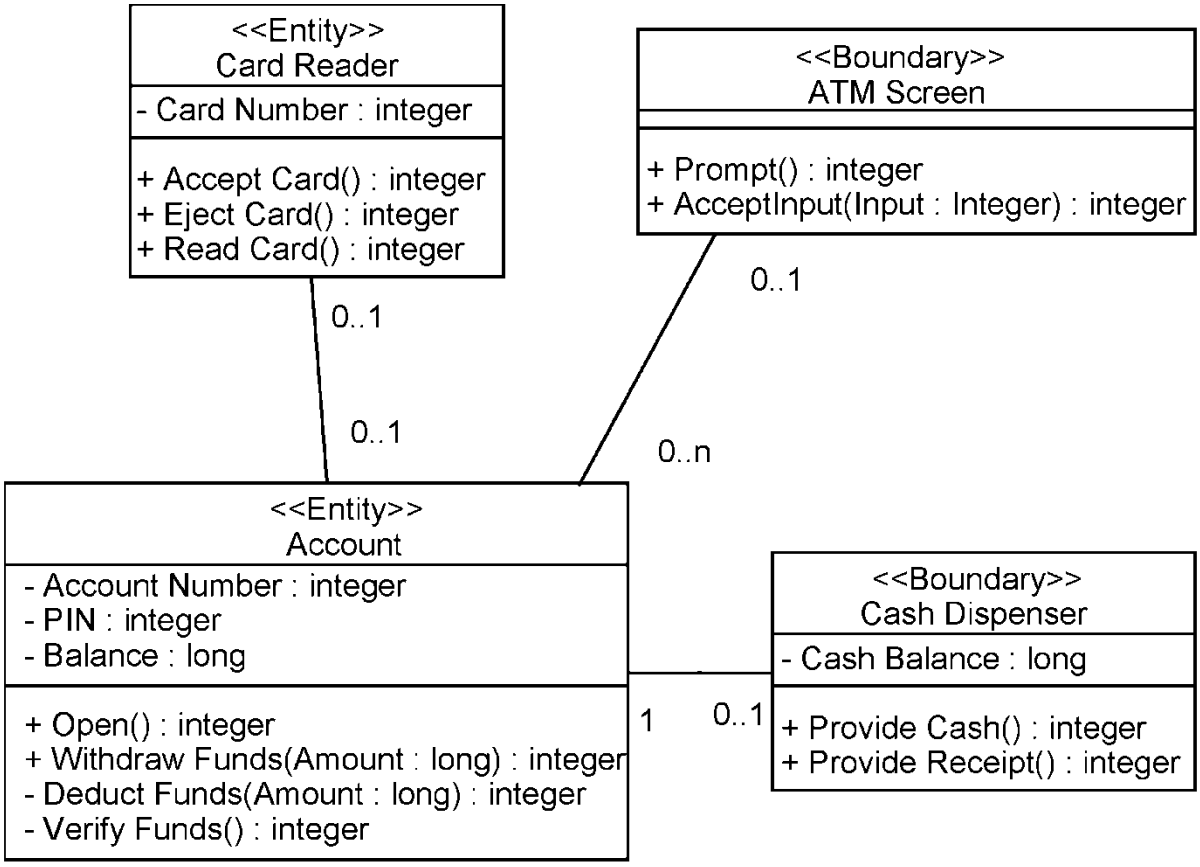
На кооперативній діаграмі так само, як і на діаграмі послідовності, стрілки позначають повідомлення, обмін якими здійснюється в рамках цього варіанта використання. Їхня тимчасова послідовність, однак, вказується шляхом нумерації повідомлень.

1.5. Діаграми класів

1.5.1. Загальні відомості

Діаграма класіввизначає типи класів системи та різного роду статичні зв'язки, які існують між ними. На діаграмах класів зображуються також атрибути класів, операції класів та обмеження, що накладаються на зв'язки між класами.

Діаграма класів для варіанта використання "Зняти гроші" показана на рис. 1.6.



Мал. 1.6. Діаграма класів для варіанта використання "Зняти гроші"

На цій діаграмі класів показані зв'язки між класами, що реалізують варіант використання "Зняти гроші". У цьому процесі задіяно чотири класи: Card Reader (пристрій для читання карток), Account (рахунок), ATM Screen (екран АТМ) та Cash Dispenser (касовий апарат). Кожен клас на діаграмі має вигляд прямокутника, розділеного на три частини. У першій міститься ім'я класу, у другій – його атрибути. В останній частині містяться операції класу, що відображають його поведінку (дії класу).

На діаграмах класів та всіх наступних діаграмах використовуються англійські імена, оскільки лише такі імена підтримуються у мовах програмування. Використання російських імен об'єктів, операцій, атрибутів тощо. буд. пов'язані з великими труднощами, оскільки CASE-средства їх підтримують належним чином.

Зв'язуючі класи лінії відбивають взаємодію між класами. Так, клас Account пов'язаний із класом ATM Screen (екран АТМ), тому що вони безпосередньо повідомляються та взаємодіють один з одним. Клас Card Reader (пристрій для читання карток) не пов'язаний із класом Cash Dispenser (касовий апарат), оскільки вони не повідомляються один з одним безпосередньо.

1.5.2 Стереотипи класів

Стереотипи – це механізм, що дозволяє розділяти класи на категорії. У мові UML визначено три основні стереотипи класів: Boundary (кордон), Entity (сутність) та Control (управління).

Граничні класи

Межовими класами (boundary classes) називаються такі класи, які розташовані на межі системи та всього навколишнього середовища. Це екранні форми, звіти, інтерфейси з апаратурою (такі як принтери або сканери) та інтерфейси з іншими системами.

Щоб знайти граничні класи, слід досліджувати діаграми варіантів використання. Кожній взаємодії між дійовою особою та варіантом використання повинен відповідати принаймні один граничний клас. Саме такий клас дозволяє дійовій особі взаємодіяти із системою.

Класи-сутності

Класи-сутності (entity classes) містять інформацію, що зберігається. Вони мають найбільше значення для користувача, і тому в назвах часто використовують терміни з предметної області. Зазвичай кожному за класу-сутности створюють таблицю у базі даних.

Керівні класи

Керівні класи (control classes) відповідають за координацію дій інших класів. Зазвичай кожен варіант використання є один управляючий клас, який контролює послідовність подій цього варіанта використання. Керуючий клас відповідає за координацію, але сам несе в собі ніякої функціональності, тому що інші класи не надсилають йому великої кількості повідомлень. Натомість він сам посилає безліч повідомлень. Керуючий клас просто делегує відповідальність іншим класам, тому його часто називають класом-менеджером.

У системі можуть бути й інші класи, що управляють, загальні для декількох варіантів використання. Наприклад, може бути клас SecurityManager (менеджер безпеки), який відповідає за контроль подій, пов'язаних з безпекою. Клас TransactionManager (менеджер транзакцій) займається координацією повідомлень, які стосуються транзакцій з базою даних. Можуть бути інші менеджери для роботи з іншими елементами функціонування системи, такими як поділ ресурсів, розподілена обробка даних або обробка помилок.

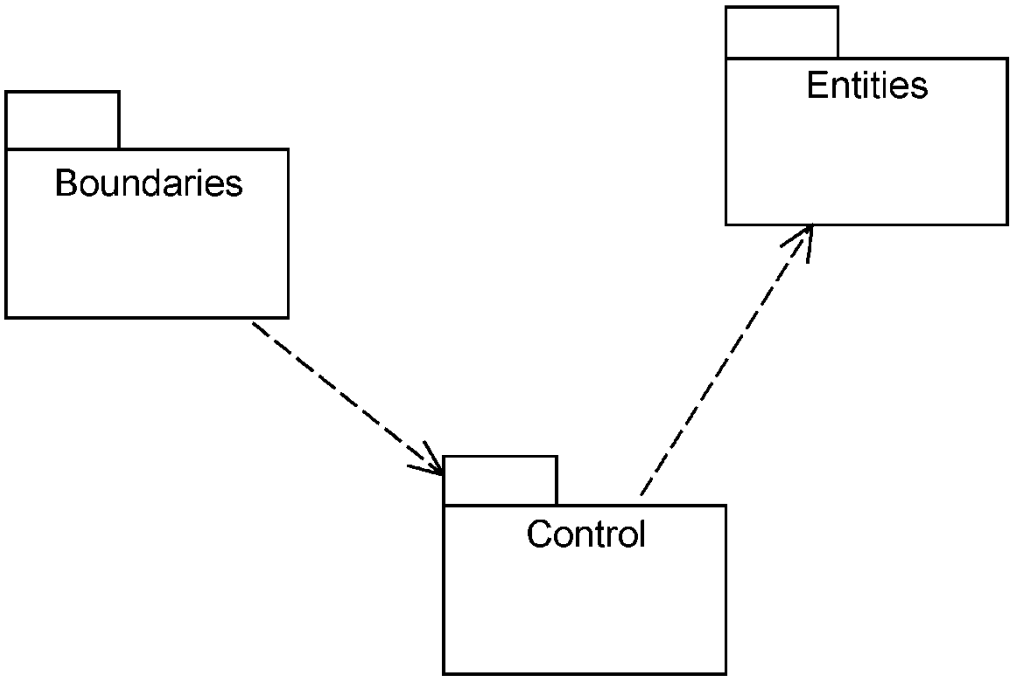
Крім згаданих вище стереотипів, можна створювати і свої власні.

1.5.3. Механізм пакетів

Пакети застосовують, щоб згрупувати класи, які мають деяку спільність. Існує кілька найпоширеніших підходів до угруповання. По-перше, можна групувати їх за стереотипом. У такому випадку виходить один пакет з класами-сутностями, один з граничними класами, один з керуючими класами і т.д. .

Інший підхід полягає в об'єднанні класів з їхньої функціональності. Наприклад, у пакеті Security (безпека) містяться всі класи, які відповідають за безпеку програми. У такому разі інші пакети можуть називатися Employee Maintenance (Робота зі співробітниками), Reporting (Підготовка звітів) та Error Handling (Обробка помилок). Перевага цього підходу полягає у можливості повторного використання.

Механізм пакетів застосовується до будь-яких елементів моделі, а не тільки до класів. Якщо для угруповання класів не використовувати деякі евристики, вона стає довільною. Одна з них, яка переважно використовується в UML, - це залежність. Залежність між двома пакетами існує у тому випадку, якщо між будь-якими двома класами у пакетах існує будь-яка залежність. Таким чином, діаграма пакетів (рис. 1.7) є діаграмою, що містить пакети класів і залежності між ними. Строго кажучи, пакети та залежності є елементами діаграми класів, тобто діаграма пакетів – це форма діаграми класів.



Мал. 1.7. Діаграма пакетів

Залежність між двома елементами має місце в тому випадку, якщо зміни у визначенні одного елемента можуть спричинити зміни в іншому. Що стосується класів, то причини для залежностей можуть бути різними: один клас посилає повідомлення іншому; один клас включає частину даних іншого класу; один клас використовує інший як параметр операції. Якщо клас змінює свій інтерфейс, будь-яке повідомлення, яке він посилає, може втратити свою силу.

Пакети не дають відповіді на питання, яким чином можна зменшити кількість залежностей у вашій системі, проте вони допомагають виділити ці залежності, а після того, як вони опиняться, залишається тільки попрацювати над зниженням їх кількості. Діаграми пакетів вважатимуться основним засобом управління загальної структурою системи.

Пакети є життєво необхідним засобом великих проектів. Їх слід використовувати у тих випадках, коли діаграма класів, що охоплює всю систему загалом і розміщена на єдиному аркуші паперу формату А4, стає нечитаною.

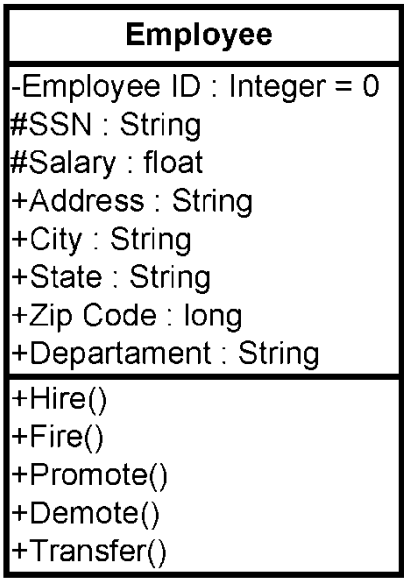
1.5.4. Атрибути

Атрибут – це елемент інформації, пов'язаний із класом. Наприклад, у класу Company (компанія) можуть бути атрибути Name (Назва), Address (Адреса) та NumberOfEmployees (Кількість службовців).

Оскільки атрибути містяться всередині класу, вони приховані з інших класів. У зв'язку з цим може знадобитися вказати, які класи мають право читати та змінювати атрибути. Ця властивість називається видимістю атрибуту (attribute visibility).

У атрибута можна визначити чотири можливі значення цього параметра. Розглянемо кожен із них у контексті прикладу (рис. 1.8). Нехай у нас є клас Employee з атрибутом Address та клас Company:

- **Public (загальний, відкритий).**Це значення видимості передбачає, що атрибут буде видно усіма іншими класами. Будь-який клас може переглянути чи змінити значення атрибута. У такому випадку клас Company може змінити значення атрибута Address класу Employee. Відповідно до нотації UML загальному атрибуту передує знак "+".
- **Private (закритий, таємний).**Відповідний атрибут не видно жодним іншим класом. Клас Employee знатиме значення атрибута Address і зможе змінювати його, але клас Company не зможе його побачити, ні редагувати. Якщо це знадобиться, він повинен попросити клас Employee переглянути чи змінити значення цього атрибута, що зазвичай робиться за допомогою загальних операцій. Закритий атрибут позначається символом - - відповідно до нотації UML.
- **Protected (захищений).**Такий атрибут доступний лише самому класу та його нащадкам. Припустимо, що у нас є два різні типи співробітників - з погодинною оплатою та на окладі. Таким чином, ми отримуємо два інші класи HourlyEmp та SalariedEmp, які є нащадками класу Employee. Захищений атрибут Address можна переглянути або змінити з класів Employee, HourlyEmp та SalariedEmp, але не з класу Company. Нотація UML для захищеного атрибута – це знак «#».
- Package or Implementation (пакетний). Припускає, що цей атрибут є загальним, але лише в межах його пакета. Припустимо, що атрибут Address має пакетну видимість. У такому разі він може бути змінений із класу Company, тільки якщо цей клас знаходиться у тому самому пакеті. Цей тип видимості не позначається спеціальним значком.



Мал. 1.8. Видимість атрибутів

Загалом, атрибути рекомендується робити закритими або захищеними. Це дозволяє краще контролювати сам атрибут та код. За допомогою закритості чи захищеності вдається уникнути ситуації, коли значення атрибута змінюється всіма класами системи. Натомість логіка зміни атрибуту буде укладена в тому ж класі, що і сам цей атрибут. Параметри видимості, що задаються, вплинуть на генерований код.

1.5.5. Операції

Операції реалізують пов'язану з класом поведінку. Операція включає три частини - ім'я, параметри та тип значення, що повертається. Параметри – це аргументи, які отримують операція «на вході». Тип значення, що повертається відноситься до результату дії операції.

На діаграмі класів можна показувати як імена операцій, так і імена операцій разом з їх параметрами та типом значення, що повертається. Щоб зменшити завантаженість діаграми, корисно буває на деяких із них показувати лише імена операцій, а на інших їхню повну сигнатуру.

У мові UML операції мають таку нотацію:

Ім'я Операції (аргумент1: тип даних аргументу1, аргумент2: тип даних аргументу2,...): тип значення, що повертається

Слід розглянути чотири різні типи операцій.

Операції реалізації

Операції реалізації (implementor operations) реалізують деякі бізнес-функції. Такі операції можна визначити, досліджуючи діаграми взаємодії. Діаграми цього фокусуються на бізнес- функціях, і кожне повідомлення діаграми, швидше за все, можна співвіднести з операцією реалізації.

Кожна операція реалізації має бути легко простежуваною до відповідної вимоги. Це досягається на різних етапах моделювання. Операція виводиться із повідомлення на діаграмі взаємодії, повідомлення виходять із докладного опису потоку подій, що створюється на основі варіанта використання, а останній - на основі вимог. Можливість простежити весь цей ланцюжок дозволяє гарантувати, що кожна вимога буде реалізована в коді, а кожен фрагмент коду реалізує якусь вимогу.

Операції управління

Операції управління (manager operations) управляють створенням та знищенням об'єктів. У цю категорію потрапляють конструктори та деструктори класів.

Операції доступу

Атрибути зазвичай бувають закритими чи захищеними. Тим не менш, інші класи іноді повинні переглядати чи змінювати їх значення. І тому існують операції доступу (access operations).

Нехай, наприклад, ми маємо атрибут Salary класу Employee. Ми не хочемо, щоб усі інші класи могли змінювати цей атрибут. Натомість до класу Employee ми додаємо дві операції доступу – GetSalary та SetSalary. До першої з них, що є спільною, можуть звертатися інші класи. Вона просто отримує значення атрибуту Salary і повертає його класу, що викликав її. Операція SetSalary також є загальною, вона допомагає класу, що викликав її, встановити нове значення атрибуту Salary. Ця операція може містити будь-які правила та умови перевірки, які необхідно виконати, щоб зарплата могла бути змінена.

Такий підхід дає можливість безпечно інкапсулювати атрибути всередині класу, захистивши їх від інших класів, але все ж таки дозволяє здійснити до них контрольований доступ. Створення операцій Get та Set (отримання та зміни значення) для кожного атрибуту класу є стандартом.

Допоміжні операції

Допоміжними (helper operations) називаються такі операції класу, які необхідні для виконання його відповідальності, але про які інші класи не повинні нічого знати. Це закриті та захищені операції класу.

Щоб ідентифікувати операції, виконайте такі дії:

1 . Вивчіть діаграми послідовності та кооперативні діаграми. Більшість повідомлень цих діаграмах є операціями реалізації. Рефлексивні повідомлення будуть допоміжними операціями.
2. Розгляньте керуючі операції. Може знадобитися додати конструктори та деструктори.
3. Розгляньте операції доступу. Для кожного атрибута класу, з яким повинні працювати інші класи, треба створити операції Get і Set.

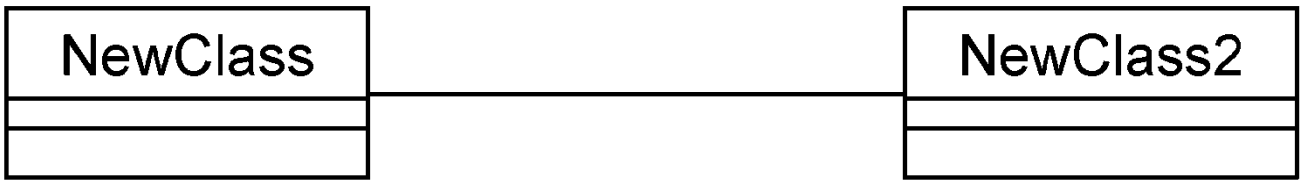
1.5.6. Зв'язки

Зв'язок є семантичну взаємозв'язок між класами. Вона дає класу можливість дізнаватися про атрибути, операції та зв'язки іншого класу. Іншими словами, щоб один клас міг надіслати повідомлення іншому на діаграмі послідовності або кооперативній діаграмі, між ними має існувати зв'язок.

Існують чотири типи зв'язків, які можуть бути встановлені між класами: асоціації, залежності, агрегації та узагальнення.

Асоціації

Асоціація (association) – це семантичний зв'язок між класами. Їх малюють на діаграмі класів як звичайної лінії.



Мал. 1.9. Асоціація

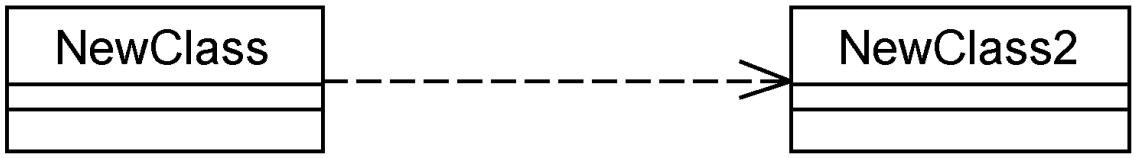
Асоціації можуть бути двоспрямованими, як у прикладі, або односпрямованими. На мові UML двонаправлені асоціації малюють у вигляді простої лінії без стрілок або зі стрілками з обох сторін. На односпрямованій асоціації зображують лише одну стрілку, що показує її напрямок.

Напрямок асоціації можна визначити, вивчаючи діаграми послідовності та кооперативні діаграми. Якщо всі повідомлення ними відправляються лише одним класом і приймаються лише іншим класом, але з навпаки, між цими класами має місце односпрямований зв'язок. Якщо хоча б одне повідомлення відправляється у зворотний бік, асоціація має бути двоспрямованою.

Асоціації можуть бути рефлексивними. Рефлексивна асоціація передбачає, що один екземпляр класу взаємодіє з іншими екземплярами цього класу.

Залежно

Зв'язки залежності (dependency) також відбивають зв'язок між класами, але вони завжди односпрямовані і показують, що клас залежить від визначень, зроблених іншому. Залежно зображують у вигляді стрілки, проведеної пунктирною лінією.

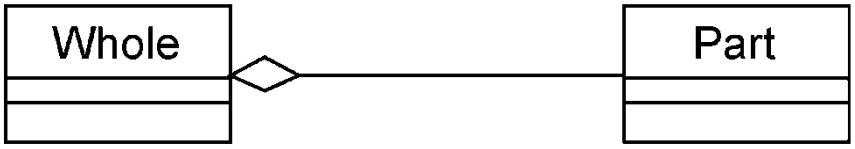


Мал. 1.10. Залежність

При генерації коду для цих класів до них не додаватимуться нові атрибути. Проте, буде створено специфічні для мови оператори, необхідні підтримки зв'язку. Наприклад, мовою C++ код увійдуть необхідні оператори #include.

Агрегації

Агрегації (aggregations) є тіснішу форму асоціації. Агрегація - це зв'язок між цілим та його частиною. Наприклад, у вас може бути клас Автомобіль, а також класи Двигун, Покришки та класи для інших частин автомобіля. В результаті об'єкт класу Автомобіль складатиметься з об'єкта класу Двигун, чотирьох об'єктів Покришок і т.д.



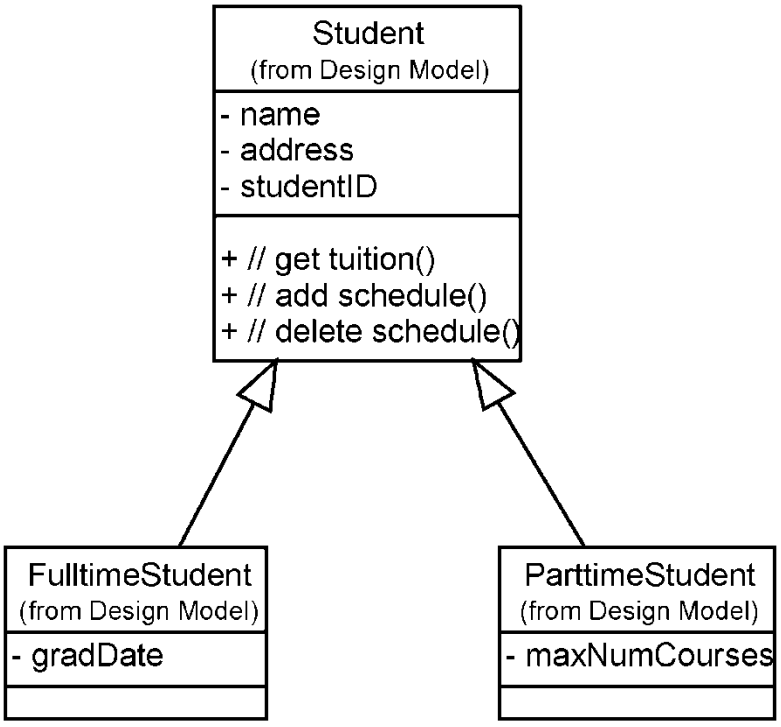
Мал. 1.11. Агрегація

На додаток до простої агрегації UML вводить більш сильний різновид агрегації, що називається композицією. Згідно з композицією, об'єкт-частина може належати лише єдиному цілому, і, крім того, як правило, життєвий цикл частин збігається з циклом цілого: вони живуть і вмирають разом із ним. Будь-яке видалення цілого поширюється з його частини.

Таке каскадне видалення нерідко сприймається як частина визначення агрегації, проте воно завжди мається на увазі у разі, коли множинність ролі становить 1..1; наприклад, якщо необхідно видалити Клієнта, то це видалення поширюється і на Замовлення (і, у свою чергу, на Рядки замовлення).

Узагальнення

За допомогою узагальнень (generalization) показують зв'язки наслідування між двома класами. Більшість об'єктно-орієнтованих мов безпосередньо підтримують концепцію спадкування. Вона дозволяє одному класу успадковувати всі атрибути, операції та зв'язки іншого. Мовою UML зв'язку успадкування називають узагальненнями і зображують у вигляді стрілок від класу-нащадка до класу-предка:



Мал. 1.12. Узагальнення

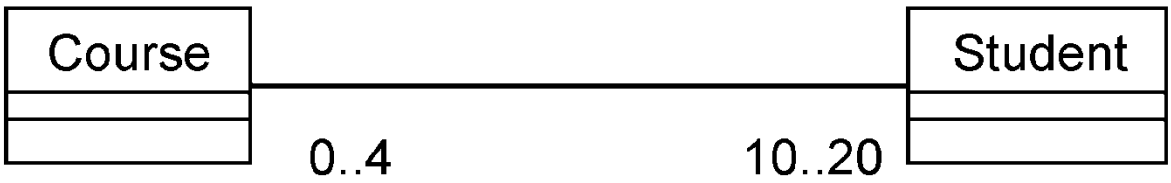
Окрім успадкованих, кожен підклас має свої власні унікальні атрибути, операції та зв'язки.

Множинність

Множинність (multiplicity) показує, скільки екземплярів одного класу взаємодіють за допомогою зв'язку з одним екземпляром іншого класу в даний момент часу.

Наприклад, при розробці системи реєстрації курсів в університеті можна визначити класи Course (курс) та Student (студент). Між ними встановлений зв'язок: у курсів можуть бути студенти, а у студентів – курси. Запитання, на яке має відповісти параметр множинності: «Скільки курсів студент може відвідувати зараз? Скільки студентів може за раз відвідувати один курс?

Оскільки множинність дає у відповідь обидва ці питання, її індикатори встановлюються обох кінцях лінії зв'язку. У прикладі реєстрації курсів ми вирішили, що один студент може відвідувати від нуля до чотирьох курсів, а один курс можуть слухати від 10 до 20 студентів. На діаграмі класів можна зобразити, як показано на рис. 1.13.



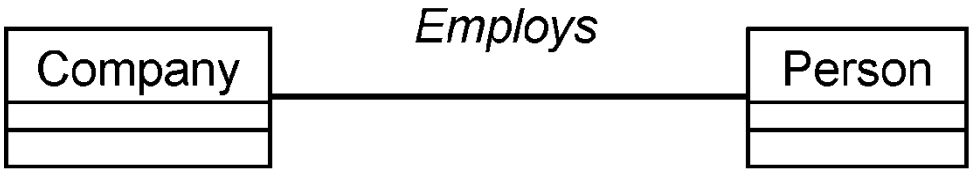
Мал. 1.13. Множинність

У мові UML прийнято такі нотації для позначення множинності:

Множинність	Значення
0..*	Нуль чи більше
1..*	Один чи більше
0..1	Нуль чи один
1.. 1 (скорочений запис: 1)	Рівно один

Імена зв'язків

Зв'язки можна уточнити за допомогою зв'язків або рольових імен. Ім'я зв'язку - це зазвичай дієслово чи дієслівна фраза, яка описує, навіщо вона потрібна. Наприклад, між класом Person (людина) та класом Company (компанія) може існувати асоціація. Чи можна поставити у зв'язку з цим питання, чи є об'єкт класу Person клієнтом компанії, її співробітником або власником? Щоб визначити це, асоціацію можна назвати "employs" (наймає):

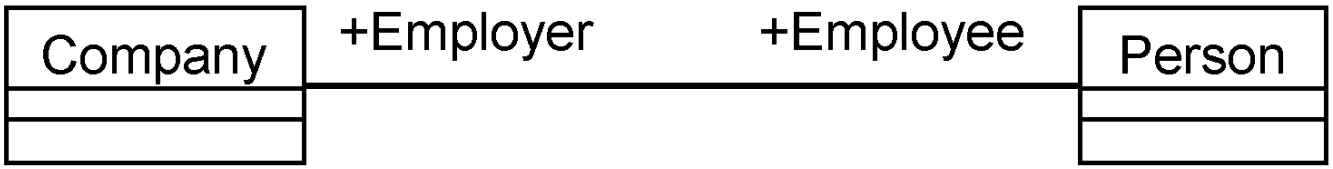


Мал. 1.14. Ім'я зв'язку

Імена у зв'язків визначати необов'язково. Зазвичай це роблять, якщо причина створення зв'язку не є очевидною. Ім'я показують біля лінії відповідного зв'язку.

Ролі

Рольові імена застосовують у зв'язках асоціації або агрегації замість імен для опису того, навіщо ці зв'язки потрібні. Повертаючись наприклад із класами Person і Company, можна сказати, що клас Person грає роль співробітника класу Company. Рольові імена - це зазвичай іменники або засновані на них фрази, їх показують на діаграмі поруч із класом, що грає відповідну роль. Як правило, користуються або рольовим ім'ям, або ім'ям зв'язку, але не обома відразу. Як і імена зв'язків, рольові імена не є обов'язковими, їх дають, тільки якщо мета зв'язку не очевидна. Приклад ролей наводиться нижче:

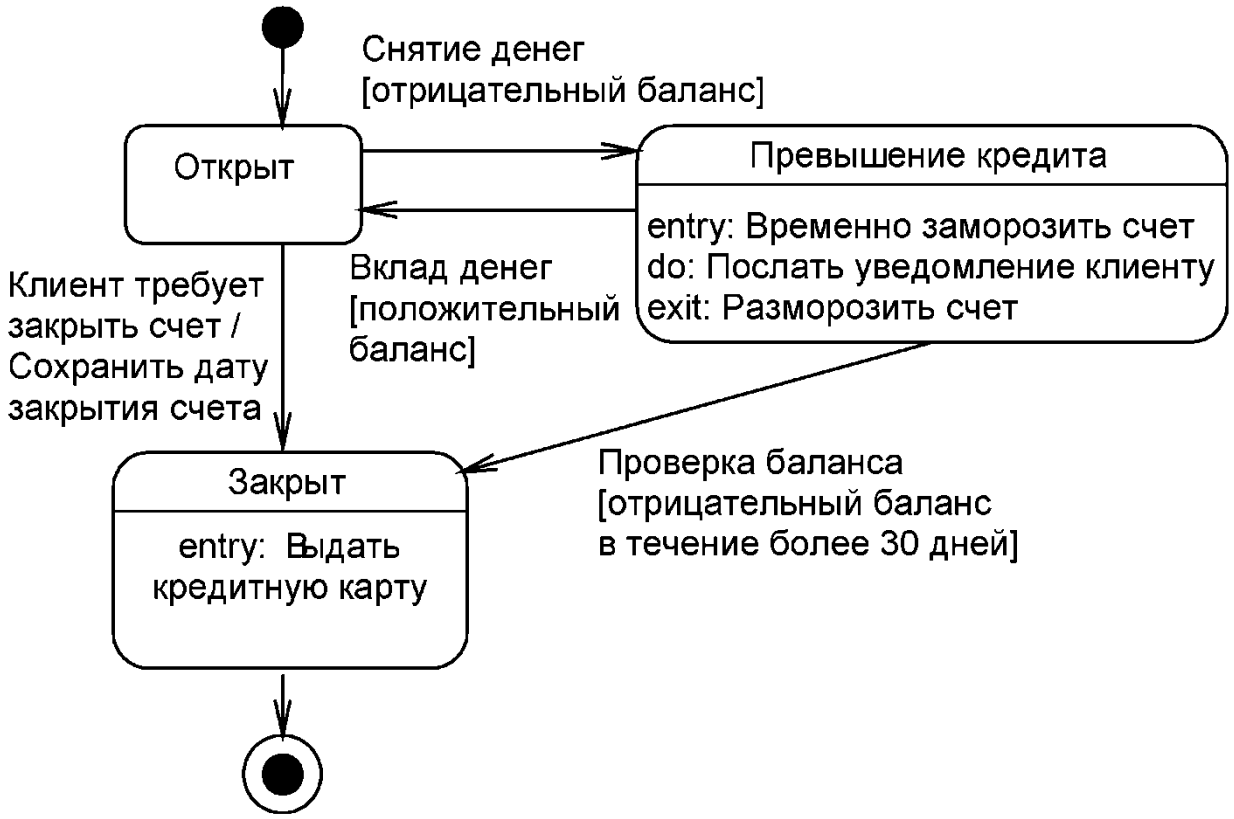


Мал. 1.15. Рольові імена

1.6. Діаграми станів

Діаграми станів визначають усі можливі стани, в яких може бути конкретний об'єкт, а також процес зміни станів об'єкта в результаті настання деяких подій. Існує багато форм діаграм станів, що незначно відрізняються один від одного семантикою. Найбільш поширена форма, яка використовується в об'єктно-орієнтованих методах, вперше застосовувалася у методі ЗМТ і згодом була адаптована Граді Бучем.

На рис. 1.16 наводиться приклад діаграми станів банківського рахунку. З цієї діаграми видно, у яких станах може бути рахунок. Можна також бачити процес переходу рахунку з одного стану до іншого. Наприклад, якщо клієнт вимагає закрити відкритий рахунок, він переходить у стан "Закрити". Вимога клієнта називається подією (event), саме такі події і викликають перехід із одного стану в інший.



Мал. 1.16. Діаграма станів для класу Account

Якщо клієнт знімає гроші з відкритого рахунку, він може перейти у стан «Перевищення кредиту». Це відбувається, тільки якщо баланс по цьому рахунку менший за нуль, що відображено умовою [негативний баланс] на нашій діаграмі. Укладена у квадратних дужках умова (guard condition) визначає, коли може чи може статися перехід із одного стану до іншого.

На діаграмі є два спеціальні стани - початковий (start) і кінцевий (stop). Початковий стан виділено чорною точкою, воно відповідає стану об'єкта, коли він щойно був створений. Кінцевий стан позначається чорною точкою в білому гуртку, він відповідає стану об'єкта безпосередньо перед знищенням. На діаграмі станів може бути один і лише один початковий стан. У той же час може бути стільки кінцевих станів, скільки вам потрібно, або їх може не бути взагалі. Коли об'єкт перебуває у певному стані, можуть виконуватися різні процеси. У прикладі при перевищенні кредиту клієнту надсилається відповідне повідомлення. Процеси, що відбуваються, коли об'єкт перебуває у певному стані, називаються діями (actions).

Зі станом можна пов'язувати дані п'яти типів: діяльність, вхідна дія, вихідна дія, подія та історія стану. Розглянемо кожен із них у контексті діаграми станів для класу Account системи ATM.

Діяльність

Діяльністю (activity) називається поведінка, що реалізується об'єктом, доки він перебуває в даному стані. Наприклад, коли рахунок перебуває у стані «Закритий», відбувається повернення кредитної картки користувачеві. Діяльність - це поведінка, що переривається. Воно може виконуватися до завершення, поки об'єкт перебуває у цьому стані, чи може бути перервано переходом об'єкта в інший стан. Діяльність зображують усередині самого стану, їй має передувати слово do (робити) та двокрапка.

Вхідна дія

Вхідною дією (entry action) називається поведінка, яка виконується, коли об'єкт переходить у цей стан. У прикладі рахунку в банку, коли він переходить у стан «Переважений рахунок», виконується дія «Тимчасово заморозити рахунок», незалежно від того, звідки об'єкт перейшов у цей стан. Таким чином, дана дія здійснюється не після того, як об'єкт перейшов у цей стан, а швидше як частина цього переходу. На відміну від діяльності, вхідна дія розглядається як безперервна.

Вхідну дію також показують усередині стану, йому передує слово entry (вхід) та двокрапка.

Вихідна дія

Вихідна дія (exit action) подібна до вхідної. Однак воно здійснюється як складова частина процесу виходу з даного стану. У нашому прикладі при виході об'єкта Account зі стану «Перевищений рахунок», незалежно від того, куди він переходить, виконується дія «Розморозити рахунок». Воно є частиною такого переходу. Як і вхідна, вихідна дія є безперервною.

Вихідну дію зображують усередині стану, йому передує слово exit (вихід) та двокрапка.

Поведінка об'єкта під час діяльності, при вхідних та вихідних діях може включати надсилання події іншому об'єкту. Наприклад, об'єкт account (рахунок) може надсилати подію об'єкту card reader (пристрій читання карти). У цьому випадку опис діяльності, вхідної дії або вихідної дії передує знак «^». Відповідний рядок на діаграмі виглядає як

Do: ^Мета.Подія(Аргументи)

Тут Мета - це об'єкт, що отримує подію, Подія - це повідомлення, що надсилається, а Аргументи є параметрами повідомлення, що посилається.

Діяльність може виконуватися в результаті отримання об'єктом деякої події. Наприклад, об'єкт account може бути в стані Відкрито. При отриманні певної події виконується певна діяльність.

Переходом(Transition) називається переміщення з одного стану до іншого. Сукупність переходів діаграми показує, як об'єкт може рухатися між своїми станами. На діаграмі всі переходи зображують у вигляді стрілки, що починається на початковому стані і закінчується наступним.

Переходи може бути рефлексивними. Об'єкт може перейти в той самий стан, в якому він зараз перебуває. Рефлексивні переходи зображують у вигляді стрілки, що починається і завершується на тому самому стані.

У переходу є кілька специфікацій. Вони включають події, аргументи, що захищають умови, дії та події, що посилаються. Розглянемо кожне їх у контексті прикладу АТМ.

Події

Подія (event) - те, що викликає перехід із одного стану до іншого. У прикладі подія «Клієнт вимагає закрити» викликає перехід рахунку з відкритого в закритий стан. Подію розміщують на діаграмі вздовж лінії переходу.

На діаграмі для відображення події можна використовувати ім'я операції, так і звичайну фразу. У прикладі події описані звичайними фразами. Якщо ви хочете використовувати операції, то подію «Клієнт вимагає закрити» можна назвати RequestClosure().

Події можуть мати аргументи. Так, подія «Зробити внесок», що викликає перехід рахунку зі стану «Переважений рахунок» у стан «Відкритий», може мати аргумент Amount (Кількість), що описує суму депозиту.

Більшість переходів повинні мати події, оскільки саме вони насамперед змушують перехід здійснитися. Тим не менш, бувають і автоматичні переходи, які не мають подій. При цьому об'єкт сам переміщається з одного стану до іншого зі швидкістю, що дозволяє здійснитися вхідним діям, діяльності та вихідним діям.

Огороджувальні умови

Огороджувальні умови (guard conditions) визначають коли перехід може, а коли не може здійснитися. У нашому прикладі подія «Зробити внесок» переведе рахунок зі стану «Перевищення рахунку» у стан «Відкритий», але тільки якщо баланс буде більшим за нуль. В іншому випадку перехід не здійсниться.

Огороджувальні умови зображують на діаграмі вздовж лінії переходу після імені події, укладаючи в квадратні дужки.

Огороджувальні умови задавати необов'язково. Однак якщо існує декілька автоматичних переходів зі стану, необхідно визначити для них умови, що взаємно виключають огороджувальні умови. Це допоможе читачеві діаграми зрозуміти, який шлях переходу буде автоматично обрано.

Дія

Дія (action), як говорилося, є безперервне поведінка, що здійснюється як частина переходу. Вхідні та вихідні дії показують усередині станів, оскільки вони визначають, що відбувається, коли об'єкт входить або виходить із нього. Більшість дій, однак, зображують вздовж лінії переходу, оскільки вони не повинні здійснюватися при вході або виході зі стану.

Наприклад, при переході рахунку з відкритого в закритий стан виконується дія «Зберегти дату закриття рахунку». Ця безперервна поведінка здійснюється лише під час переходу зі стану «Відкритий» у стан «Закритий».

Дія малюють уздовж лінії переходу після імені події, йому передує коса межа.

Подія або дія можуть бути поведінкою всередині об'єкта, а можуть являти собою повідомлення, яке надсилається іншому об'єкту. Якщо подія чи дія надсилається іншому об'єкту, перед ним на діаграмі поміщають знак «^».

Діаграми станів не треба створювати для кожного класу, вони застосовуються лише у складних випадках. Якщо об'єкт класу може існувати в кількох станах і в кожному з них поводиться по-різному, йому може знадобитися така діаграма.

1.7. Діаграми діяльності

На відміну від більшості інших засобів UML, діаграми діяльностей не мають явно вираженого джерела в попередніх роботах Буча, Рамбо та Якобсона, і запозичують ідеї з різних методів, зокрема, методу моделювання станів SDL і мереж Петрі. Ці діаграми особливо корисні в описі поведінки, що включає велику кількість паралельних процесів [Фаулер-1999].

Подібно до більшості інших засобів, що моделюють поведінку, діаграми діяльностей мають певні переваги і недоліки, тому їх найкраще використовувати в поєднанні з іншими засобами.

Найбільшою перевагою діаграм діяльності є підтримка паралелізму. Завдяки цьому є потужним засобом моделювання потоків робіт і, по суті, паралельного програмування. Найбільший їх недолік полягає в тому, що зв'язки між діями та об'єктами проглядаються не надто чітко.

Ці зв'язки можна спробувати визначити, використовуючи для діяльності мітки з іменами об'єктів, але цей спосіб не має такої ж простої безпосередності, як у діаграм взаємодії. Діаграми діяльностей краще використовувати у таких ситуаціях:

- Аналіз варіантів використання. На цій стадії нас не цікавить зв'язок між діями та об'єктами, а потрібно лише зрозуміти, які дії повинні мати місце та які залежності у

поведінці системи. Зв'язування методів та об'єктів виконується пізніше за допомогою діаграм взаємодії.

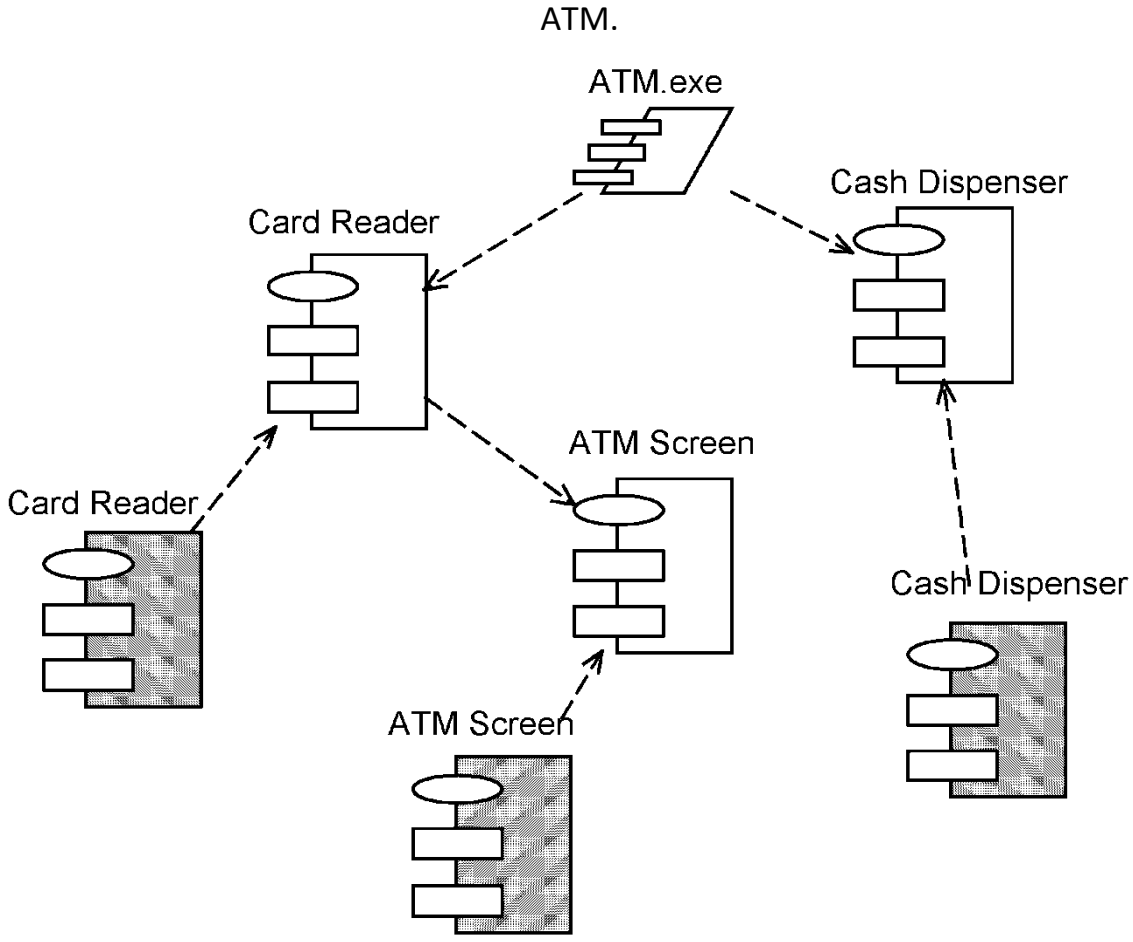
- Аналіз потоків робіт (workflow) у різних варіантах використання. Коли варіанти використання взаємодіють друг з одним, діаграми діяльностей є сильним засобом уявлення та аналізу їхньої поведінки.

1.8. Діаграми компонентів

Діаграми компонентів показують, як виглядає модель фізично. На них зображені компоненти програмного забезпечення та зв'язку між ними. При цьому на такій діаграмі виділяють два типи компонентів: компоненти, що виконуються, і бібліотеки коду.

Кожен клас моделі (або підсистема) перетворюється на компонент вихідного коду. Після створення вони одразу додаються до діаграми компонентів. p align="justify"> Між окремими компонентами зображують залежності, відповідні залежностям на етапі компіляції або виконання програми.

На рис. 1.17 зображено одну з діаграм компонентів для системи ATM.



Мал. 1.17. Діаграма компонентів для клієнта ATM

На цій діаграмі показано компоненти клієнта системи ATM. У разі система розробляється мовою З++. Кожен клас має свій власний заголовний файл і файл з

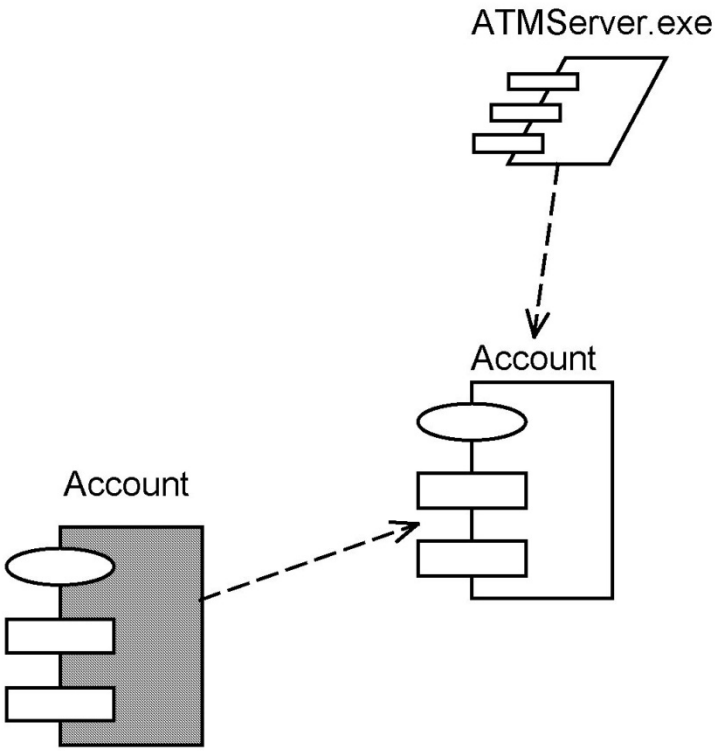
розширенням .CPP, отже кожен клас перетворюється на власні компоненти на діаграмі. Наприклад, клас ATM screen перетворюється на компонент ATM Screen діаграми. Він перетворюється також і на другий компонент ATM Screen. Разом ці два компоненти представляють тіло та заголовок класу ATM Screen. Виділений темним компонентом називається специфікацією пакета (package specification) і відповідає файлу тіла класу ATM Screen мовою C++ (файл з розширенням .CPP). Невиділений компонент також називається специфікацією пакета, але відповідає заголовному файлу класу мови C++ (файл з розширенням .H). Компонент ATM.exe є специфікацією завдання та представляє потік обробки інформації (thread of processing). В даному випадку потік обробки є програмою, що виконується.

Компоненти з'єднані штриховою лінією, що відповідає залежностям між ними. Наприклад, клас Card Reader залежить від класу ATM Screen. Це означає, що, щоб клас Card Reader міг бути скомпільований, клас ATM Screen має вже існувати. Після компіляції всіх класів може бути створений виконуваний файл ATMClient.exe.

Приклад ATM містить два потоки обробки і, таким чином, виходять два виконувані файли. Один з них – це клієнт ATM, він містить компоненти Cash Dispenser, Card Reader та ATM Screen. Другий файл - це сервер ATM, що включає компонент Account. Діаграма компонентів для сервера ATM показано на рис. 1.18.

Як видно з прикладу, система може мати кілька діаграм компонентів, залежно від числа підсистем або виконуваних файлів. p align="justify"> Кожна підсистема є пакетом компонентів. У випадку, пакети - це сукупності компонентів. Приклад ATM містить два пакети: клієнт ATM та сервер ATM.

Діаграми компонентів застосовуються тими учасниками проекту, хто відповідає за компіляцію системи. З неї видно, в якому порядку треба компілювати компоненти, а також які компоненти, що виконуються, будуть створені системою. На такій діаграмі показано відповідність класів реалізованим компонентам. Вона потрібна там, де починається створення коду.



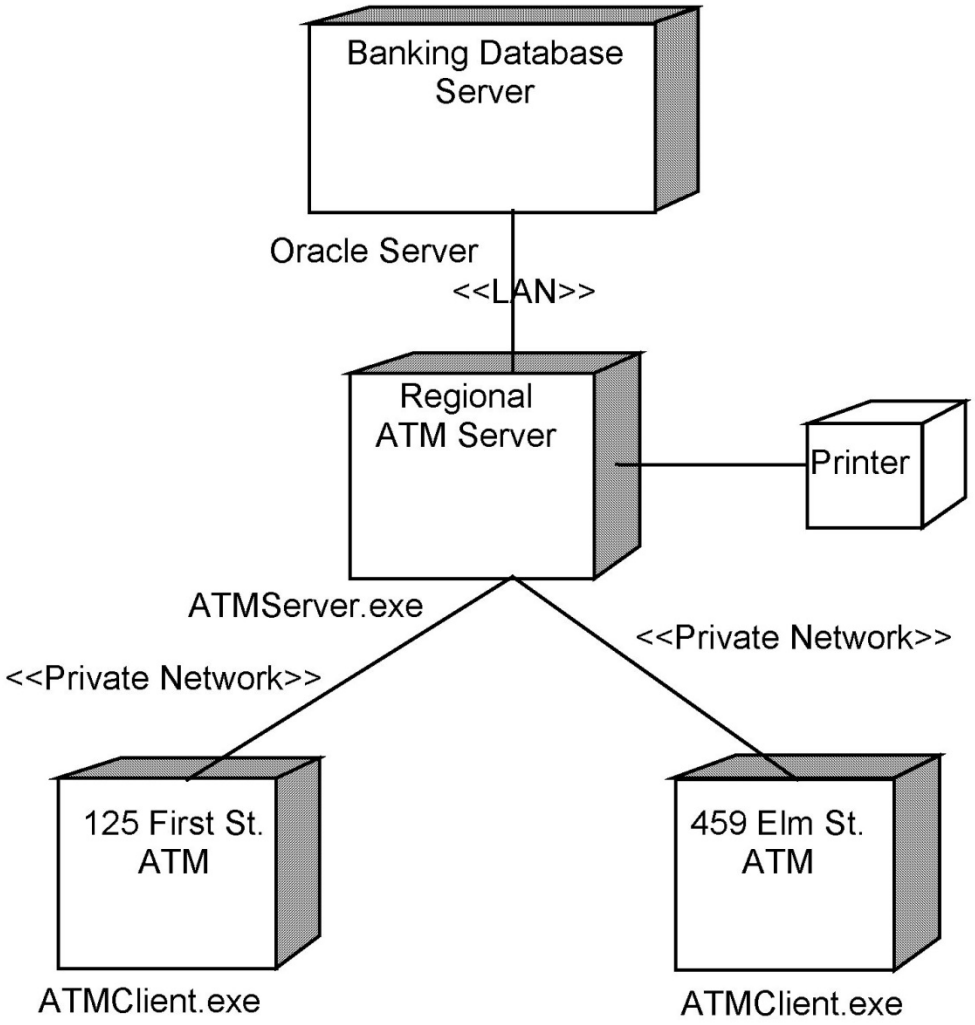
Мал. 1.18. Діаграма компонентів для сервера ATM

1.9. Діаграми розміщення

Діаграма розміщення(Deployment diagram) відображає фізичні взаємозв'язки між програмними та апаратними компонентами системи. Вона є гарним засобом для того, щоб показати маршрути переміщення об'єктів та компонентів у розподіленій системі.

Кожен вузол на діаграмі розміщення є деяким типом обчислювального пристрою - в більшості випадків частина апаратури. Ця апаратура може бути простим пристроєм або датчиком, а може бути мейнфреймом.

Діаграма розміщення показує фізичне розташування мережі та місцезнаходження в ній різних компонентів. У прикладі система ATM складається з великої кількості підсистем, виконуваних окремих фізичних пристроях, чи вузлах (node). Діаграма розміщення системи ATM показано на рис. 1.19.



Мал. 1.19. Діаграма розміщення системи ATM

З цієї діаграми можна дізнатися про фізичне розміщення системи. Клієнтські програми ATM працюватимуть у кількох місцях різних сайтах. Через закриті мережі здійснюватиметься їхнє повідомлення з регіональним сервером ATM. На ньому працюватиме програмне забезпечення сервера ATM. У свою чергу, за допомогою локальної мережі регіональний сервер буде повідомляти сервер банківської бази даних, що працює під управлінням Oracle. Нарешті, з регіональним ATM з'єднаний принтер.

Діаграма розміщення використовується менеджером проекту, користувачами, архітектором системи та експлуатаційним персоналом, щоб зрозуміти фізичне розміщення системи та розташування її окремих підсистем.