Уніфікована мова моделювання (UML)

1 Основні засади

Створення Уніфікованої мови моделювання (Unified Modeling Language, UML) почалося в 1994 році. У цей час Грейді Буч (Grady Booch) і Джеймс Рамбо (James Rumbaugh) почали поєднувати кілька методів об'єктно-орієнтованого моделювання у фірмі Rational Software. У 1995 році була репрезентована специфікація методу, який отримав назву Unified Method. Пізніше до розроблювачів приєднався Івар Джекобсон (Ivar Jacobson). Раніше всі три автори працювали самостійно над різними концепціями об'єктно-орієнтованого моделювання і проектування.

Перша версія UML була прийнята консорціумом OMG (Object Management Group) у січні 1997 року як міжнародний стандарт. Затверджена ж у вересні версія UML 1.1 була застосована основними компаніями – виробниками програмного забезпечення, такими, як Microsoft, IBM, Hewlett-Packard і виробниками CASE-засобів, що реалізували підтримку UML у своїх програмних продуктах.

Автори і розробники UML представляють її як мову для *візуалізації*, *визначення*, *проектування* та *документування* програмних систем, бізнес-систем та інших систем, в першу чергу пов'язаних з програмним забезпеченням. UML визначає нотацію, що представляє собою сукупність графічних об'єктів, які використовуються в моделях. Універсальна мова об'єктного моделювання UML не залежить від мов програмування і, внаслідок цього, може підтримувати будь-яку об'єктно-орієнтовану мову програмування.

Формальна специфікація останньої версії UML 2.0 опублікована в серпні 2005 року.

2 Конструктивні блоки UML

Сутності є основою моделі. Прив'язку сутностей одну до одної забезпечують *відносини*, а *діаграми* групують набори сутностей. У UML представлені чотири типи сутностей: структурні (structural), поведінкові (behavioral), що групують (group), анотаційні (annotational).

Графічне зображення окремих структурних і поведінкових сутностей, прийняте в UML, приводиться нижче в зв'язку з діаграмами, на яких ці сутності найчастіше зображуються. Разом з тим, більшість елементів може бути присутніми практично на усіх діаграмах.

Єдиним представником сутності, що групує, є *пакет* (package). Пакет – це механізм загального призначення для організації елементів у вигляді єдиної групи. Структурні, поведінкові і навіть інші пакети, можуть бути розміщені всередині пакету.

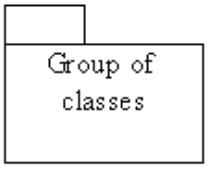

Рисунок 2.1 – Пакет

Анотації – це сутності, які в UML вживають для представлення пояснень і коментарів. Єдиним типом анотаційної сутності є *примітка* (note). Примітка з'єднується пунктирною лінією із сутністю, якої вона стосується:

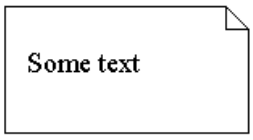

Рисунок 2.2 – Анотація

Відносини є елементами, які використовують для сполучення окремих сутностей. У UML є чотири типи відносин:

- залежність (dependency);
- асоціація (association); її різновидом є агрегація (aggregation);
- узагальнення (generalization);
- реалізація (realization).

Рисунок 2.3 – Відносини (відповідно залежність, узагальнення та асоціація)

Третім компонентом UML є діаграми.

Діаграма у UML – це графічне зображення набору елементів у вигляді зв'язаного графа з вершинами (сутностями) і ребрами (відносинами).

UML 1 пропонує такий набір діаграм:

- діаграми варіантів використання (use case diagrams)
- діаграми класів (class diagrams)
- діаграми об'єктів (object diagrams)
- діаграми взаємодії (interaction diagrams), у тому числі
  - діаграми послідовності (sequence diagrams)
  - діаграми кооперації (collaboration diagrams)
- діаграми станів (statechart diagrams)
- діаграми діяльності (activity diagrams)
- діаграми компонентів (component diagrams).
- діаграми розгортання (deployment diagrams).

UML 2 пропонує такий набір діаграм для моделювання:

- структурні діаграми (structure diagrams);
  - діаграми класів (class diagrams) - для моделювання статичної структури класів системи і зв'язків між ними;
  - діаграми об'єктів (object diagrams) - для моделювання статичної структури екземплярів класів (об'єктів) і зв'язків між ними;
  - діаграми пакетів (package diagrams) - для моделювання ієрархії логічних підсистем (тільки в UML 2);
  - діаграми внутрішньої структури (composite structure diagrams) - для моделювання внутрішньої структури сутності (тільки в UML 2);
    - діаграми кооперації (collaboration diagrams) - окремий вид діаграм внутрішньої структури (тільки в UML 2);
  - діаграми компонентів (component diagrams) - для моделювання ієрархії фізичних компонентів (підсистем) системи;

    - діаграми розміщення (deployment diagrams) - для моделювання фізичної архітектури системи;
- діаграми відображення поведінки (behavior diagrams):
    - діаграми варіантів використання (use case diagrams) - для моделювання бізнес-процесів організації та вимог до системи;
    - діаграми взаємодії (interaction diagrams):
        - діаграми послідовності (sequence diagrams) для моделювання процесу обміну повідомленнями між об'єктами в часі;
        - діаграми комунікацій (communications diagrams, у UML 1 - діаграми кооперації, collaboration diagrams) - для моделювання процесу обміну повідомленнями між об'єктами;
        - діаграми огляду взаємодії (interaction overview diagrams) - діаграма діяльності, вузлами якої виступають діаграми взаємодії (тільки в UML 2);
        - діаграми відображення за часом (timing diagrams), або діаграми синхронізації, використовують для відображення зміни стану чи значень параметрів елементів під час функціонування (тільки в UML 2);
    - діаграми станів (statechart diagrams) - для моделювання поведінки об'єктів системи при переході з одного стану в інший;
    - діаграми діяльності (activity diagrams) - для моделювання поведінки системи в рамках різних варіантів використання, або моделювання діяльності системи.

3 Діаграми варіантів використання

Діаграми варіантів використання описують поведінку системи з точки зору користувача. Вони дозволяють визначити межі системи, а також відношення між системою та зовнішнім середовищем. Діаграми варіантів використання використовують для візуалізації вимог до системи.

На діаграмах варіантів використання зображують варіанти використання, акторів та відношення між ними.

Варіант використання, або *прецедент* (use case) відповідає за конкретний спосіб взаємодії з системою. Варіанти використання відображають функціональність системи. Варіант використання має ім'я, яке може бути розміщене всередині еліпсу або під ним.

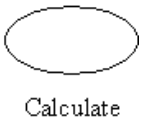

Рисунок 2.4 – Варіант використання

Актор, або *дійова особа* (actor) описує роль, яку грає персона або інша зовнішня сутність під час взаємодії з системою. Один актор може представляти багатьох фізичних осіб, у яких збігаються функції по відношенню до системи. І навпаки, одна фізична особа може грати декілька ролей, отже її функції описуються багатьма акторами.

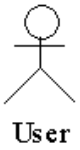

Рисунок 2.5 – Актор

UML визначає такі типи зв'язків між акторами та варіантами використання:

1. Асоціація (між актором та варіантом використання):

Рисунок 2.6 – Асоціація

Стрілка починається з елементу, до якого належить ініціатива. Іноді це буває не актор, а варіант використання. Наприклад, комп'ютеризована система може автоматично повідомляти користувача про системні помилки. UML дозволяє зображати асоціацію лінією без стрілки.

2. Залежність зі стереотипом "включає" ("include") показує, що додатковий варіант використання є обов'язковим. Стрілка починається з основного варіанту використання та вказує на додатковий.

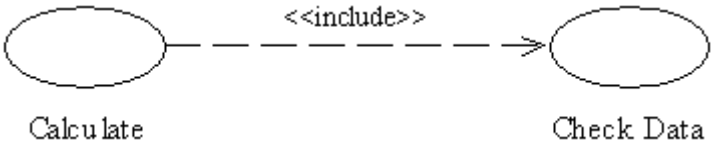

Рисунок 2.7 – Залежність зі стереотипом "включає"

3. Залежність зі стереотипом "розширює" ("extend") показує, що додатковий варіант використання не є обов'язковим і може бути реалізований пізніше. Стрілка починається з додаткового варіанту використання та вказує на основний.

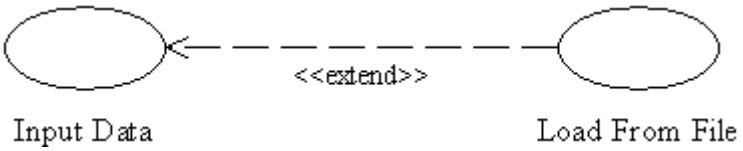

Рисунок 2.8 – Залежність зі стереотипом "розширює"

4. Узагальнення (проміж конкретним та більш узагальненим акторами).

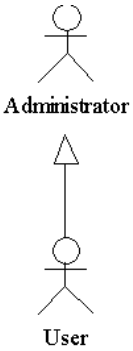

Рисунок 2.9 – Узагальнення

Узагальнення також можна застосувати до варіантів використання.

Наступна діаграма представляє вимоги щодо програми, яка розв'язує квадратне рівняння. Єдиним актором є користувач. Він може вводити дані, розв'язувати рівняння, та переглядати результати на екрані. Базові варіанти можна розширити додатковими - введення даних з файлу та зберігання результатів у іншому файлі. Крім того, розв'язання рівняння передбачає обов'язкову перевірку дискримінанту.

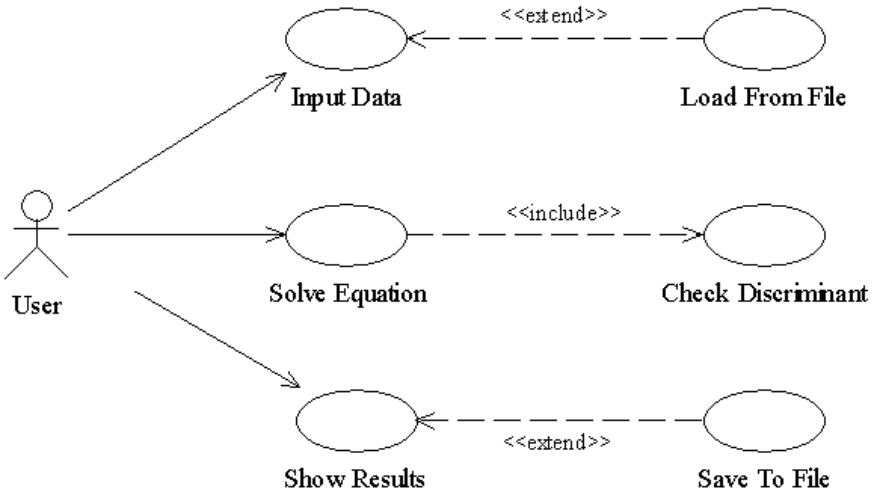

Рисунок 2.10 – Діаграма варіантів використання програми розв'язання квадратного рівняння.

Наступна діаграма представляє вимоги до комп'ютеризованої системи, яка підтримує проектування та розробку web-сайтів.

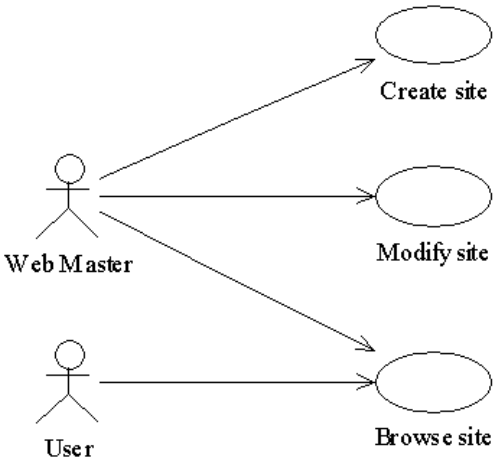

Рисунок 2.11 – Розробка web-сайту (версія 1)

Оскільки web-майстер є спеціальним типом користувача, можна змінити діаграму з додаванням відношення типу "узагальнення":

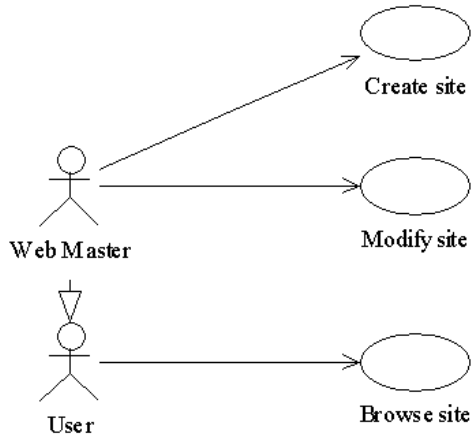

Рисунок 2.12 – Розробка web-сайту (версія 2)

Для написів на діаграмах варіантів використання (імена акторів та варіантів використання, коментарі, тощо) використовують мову, зручну та зрозумілу як для розробників, так і для майбутніх користувачів програмного забезпечення.

4 Діаграми класів

Клас представляє множину об'єктів, яку об'єднує спільна структура та поведінка. Клас є абстракцією сутностей реального світу. Реальні екземпляри сутностей (instances) це так звані *об'єкти*.

Діаграма класів відображає статичну структуру системи у термінах класів та відношень між цими класами.

Графічним зображенням класу є прямокутник, розділений на три частини. Перша частина - це ім'я класу. Друга частина - список так званих атрибутів, третя частина - список операцій.

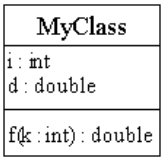

Рисунок 2.13 – Клас

Атрибути визначають дані, які описують екземпляр класу та визначають його стан. У об'єктно-орієнтованому програмуванні атрибути відображаються у поля, або елементи даних. Повний формат атрибуту має такий вигляд:

```
видимість ім'я : тип = значення_за_умовчанням
```

Видимість може бути такою:

```
- закритий (приватний)
+ відкритий (публічний)
# захищений
```

Можна використовувати скорочений формат – без видимості, без типу, без початкового значення, або взагалі тільки ім'я. Атрибут - масив визначається квадратними дужками [ ] біля імені.

Операції визначають дії, які можна застосувати до класу або його об'єкту. Операція описується рівнем доступу (видимості), списком параметрів та типом результату:

```
видимість ім'я(список_параметрів) : тип
```

Параметри у списку відокремлюють один від одного комами. Кожний параметр має такий формат:

```
напрямок ім'я : тип = значення_за_умовчанням
```

Для того, щоб вказати напрямок, вживають такі службові слова: **in** (вихідні дані), **out** (результат), **inout** (потік даних в обох напрямках). Найчастіше цей специфікатор взагалі випускають.

Можна опустити усі частини специфікації операції, за винятком імені та круглих дужок.

UML 2 дозволяє у випадку необхідності додавати четверту частину до зображення класу. У цій частині розташовують піктограми вкладених класів.

Асоціації - це найбільш загальний тип зв'язків між класами. За допомогою асоціацій зв'язують класи, об'єкти яких створюються та існують незалежно. У мовах програмування звичайна асоціація реалізується за допомогою покажчиків або посилань. Асоціація може бути спрямованою (один покажчик або посилання) або неспрямованою (екземпляри обох класів обізнані про екземпляри протилежного класу). Відношення типу "*множинність*" (multiplicity) показує кількість можливих зв'язків між екземплярами класів. Явне зазначення множинності можна опускати.

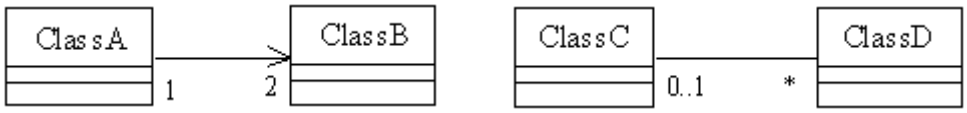

Рисунок 2.14 – Асоціації

Агрегація - це спеціальна форма асоціації, яка показує, що сутність (екземпляр) міститься в іншій сутності або не може бути створена та існувати без охопної сутності. Іноді використовується більш жорстка форма - композиція, яка передбачає що сутність повністю міститься в іншій сутності. Агрегація та композиція дозволяють вказувати множинність.

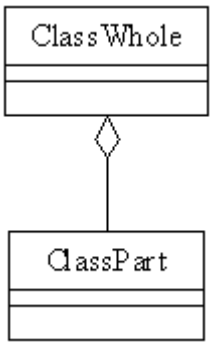

Рисунок 2.15 - Агрегація та композиція

Відношення типу "залежність" на діаграмі класів вказує, що залежний клас користується функціональністю іншого класу. На рівні мови програмування це може бути реалізовано через виклик статичних функцій іншого класу або створення тимчасового об'єкту.

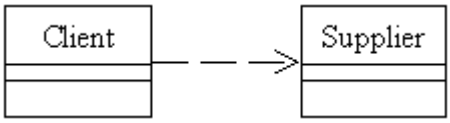

Рисунок 2.16 – Залежність

Узагальнення (*generalization*) використовується для моделювання спадкування. Більш загальний клас є базовим. Стрілка починається з похідного класу.

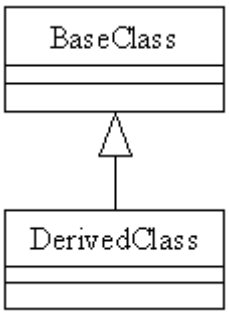

Рисунок 2.17 – Узагальнення

Відношення типу "*реалізація*" - це відношення між сутністю, яка декларує певну поведінку (наприклад, інтерфейсом) та класом, який цю поведінку реалізує.

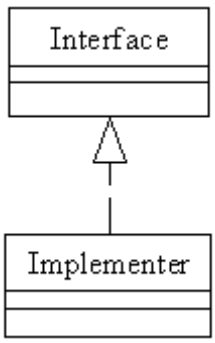

Рисунок 2.18 – Реалізація

Для написів на діаграмах класів можна використовувати будь-яку мову, але в тих випадках, коли діаграма класів використовується для генерації вихідного коду, слід вживати латинські літери та переважно англійську мнемоніку.

5 Діаграми об'єктів

Діаграма об'єктів представляє екземпляри класів (об'єкти). Графічне зображення об'єкту схоже на зображення класу. Ім'я екземпляру слід підкреслювати:

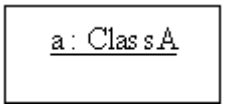

Рисунок 2.19 – Об'єкт

Можна вказувати тільки тип без імені:

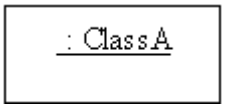

Рисунок 2.20 – Безіменний об'єкт

Зазвичай діаграма об'єктів представляє ієрархію екземплярів. Така ієрархія найчастіше відображає відношення агрегації, представлене на діаграмі класів.

6 Діаграми взаємодії

Діаграми взаємодії описують послідовність повідомлень, якими обмінюються між собою об'єкти. Завдяки діаграмам взаємодії описується поведінка системи під час реалізації різних варіантів використання.

Діаграма послідовності – це графічне зображення послідовної взаємодії об'єктів. Діаграма послідовності має два виміри: по горизонталі розміщуються об'єкти, які взаємодіють між собою; вертикальний вимір – це вимір часу.

Типовими елементами діаграми послідовності є:

- об'єкт
- лінія життя
- повідомлення
- фокус управління
- повідомлення до себе

Лінія життя - це пунктирна лінія, яка починається з піктограми об'єкту та спрямована униз.

Повідомлення , яке один об'єкт надсилає до іншого (приймача), обумовлює дію, яку потім виконує об'єкт-приймач. Графічно повідомлення зображується у вигляді стрілки, яка поєднує лінії життя двох об'єктів. Ім'я повідомлення - це фактично ім'я операції об'єкта-приймача.

Фокус управління (focus of control) відображає період часу, протягом якого об'єкт виконує операцію або чекає на результат операцій інших об'єктів. Фокус управління зображають у вигляді прямокутника, який розташовано на лінії життя.

Повідомлення до себе є по суті викликом з однієї операції певного об'єкту іншої операції цього ж об'єкту.

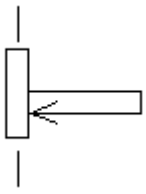

Рисунок 2.21 – Повідомлення до себе

Наступний приклад демонструє взаємодію користувача з елементами управління вікна програми під час реалізації варіанту використання "Розв'язання рівняння".

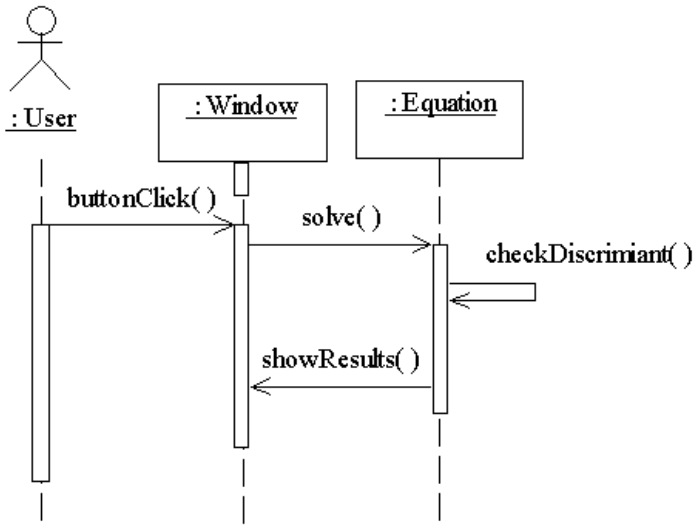

Рисунок 2.22 – Діаграма послідовності

Діаграми послідовності семантично зв'язані з *діаграмами комунікацій* (у UML 1 - діаграмами кооперації). Фактично це альтернативне представлення діаграм послідовності. Діаграми комунікацій застосовують у тих випадках, коли необхідно відобразити велику кількість об'єктів, які взаємодіють між собою, а фактор часу має менше значення.

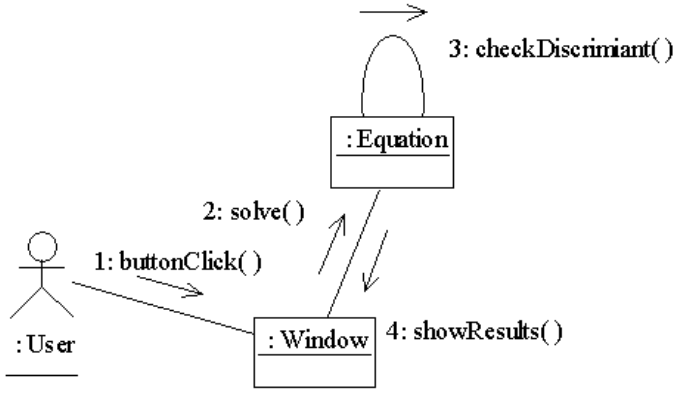

Рисунок 2.23 – Діаграма комунікацій

Числа перед іменами повідомлень вказують на відносну послідовність повідомлень.

7 Діаграми діяльності

Діаграми діяльності надають можливість моделювання послідовності виконання дій у певному процесі. Діаграми діяльності аналогічні блок-схемам та можуть бути застосовані для представлення алгоритмів.

Зазвичай діаграми діяльності включають такі елементи:

- початковий стан
- діяльність
- перехід
- розгалуження
- кінцевий стан

Кожна діаграма діяльності повинна мати один і тільки один початковий стан:

Рисунок 2.24 – Початковий стан

Усі види діяльності (введення, виведення, розрахунки) слід зображати у вигляді овалу (слід відрізняти від еліпсу та прямокутника з закругленими кутами):

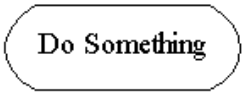

Рисунок 2.25 – Діяльність

Перехід між окремими діяльностями завжди зображують у вигляді стрілки:

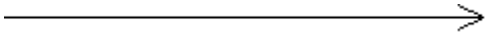

Рисунок 2.26 – Перехід

Розгалуження, як і у блок-схемі, зображується ромбом. Але, на відміну від блок-схем, умови переходу записують не всередині ромбу, а біля стрілок у квадратних дужках:

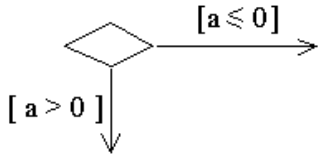

Рисунок 2.27 – Розгалуження

Кожана діаграма повинна мати один або декілька кінцевих станів:

Рисунок 2.28 – Кінцевий стан

На відміну від традиційних блок-схем, на діаграмі діяльності можна зобразити дії, які виконуються паралельно. Розгалуження на такі дії починається та закінчується лінійкою синхронізації.

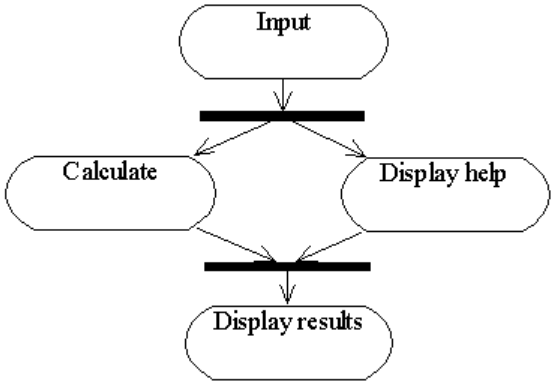

Рисунок 2.29 – Паралельність

Наступна діаграма представляє алгоритм методу дихотомії для розв'язання рівняння на інтервалі [a, b]:

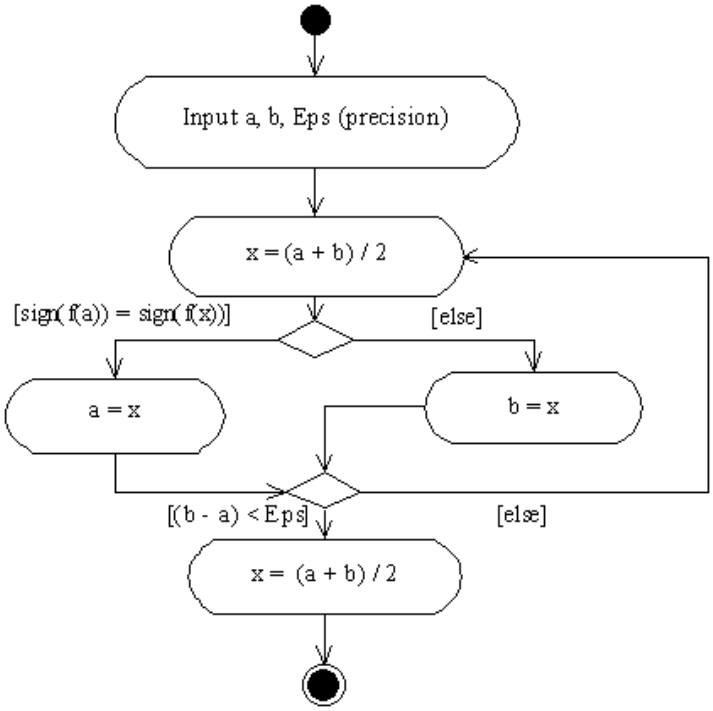

Рисунок 2.30 - Метод дихотомії

Для того, щоб відобразити відповідальність окремих акторів (пакетів, компонентів, підсистем) за певні дії, вводиться поняття доріжки (swimlane). Вони зазвичай мають імена.

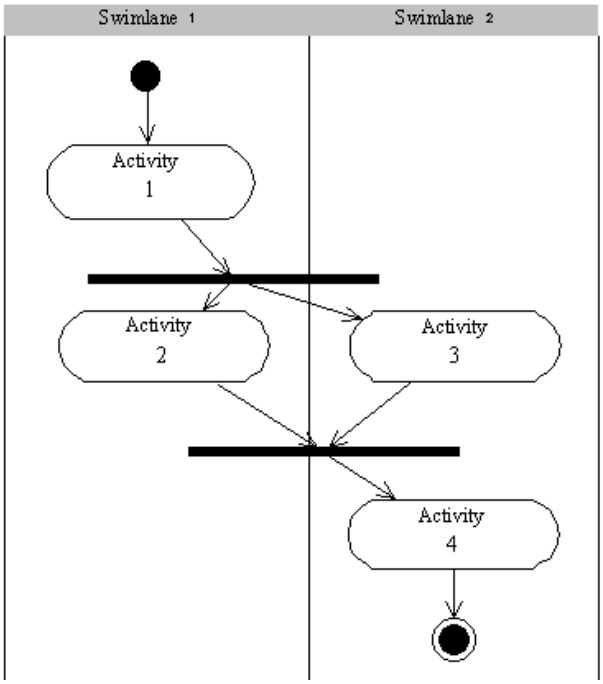

Рисунок 2.31 – Доріжки

У UML 2 замість доріжок вводиться поняття секції (Partition). Секції можуть бути розташовані в обох вимірах.

8 Діаграми станів

Діаграми станів представляють скінчений автомат набором станів та переходів. Об'єкт можна описати з точки зору станів, у яких він може знаходитись, а також можливих переходів з одного стану в інший у випадку виникнення певних подій. Свій скінчений автомат може бути пов'язаний з будь-яким класом моделі.

Стан зображують у вигляді прямокутника з закругленими кутами:

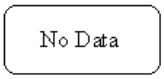

Рисунок 2.32 – Стан

Завжди є один і тільки один початковий стан об'єкту. З іншого боку, може бути декілька кінцевих станів, які відповідають різним умовам завершення життєвого циклу об'єкта. Наступна діаграма представляє можливі стани об'єкту, який представляє рівняння.

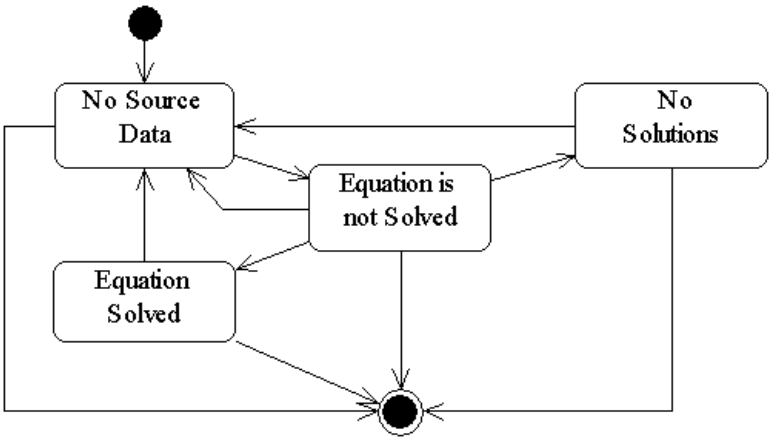

Рисунок 2.33 – Діаграма станів

9 Діаграми компонентів

Діаграма компонентів описує фізичні компоненти програмного забезпечення (файли, динамічні бібліотеки та ін.) та відношення між ними у середовищі виконання програми. Не слід плутати компоненти з вихідними текстами та бібліотеками часу компіляції.

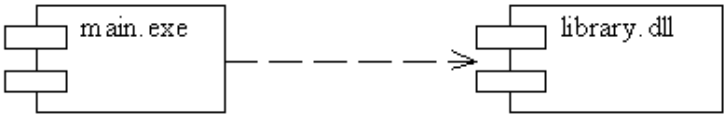

Рисунок 2.34 – Діаграма компонентів

10 Діаграми розгортання

Діаграма розгортання показує фізичне розміщення компонентів апаратного забезпечення, необхідного для нормального функціонування комп'ютеризованої системи. Для окремих компонентів - вузлів (nodes) - можна вказувати, які саме програмні елементи на них розташовані.

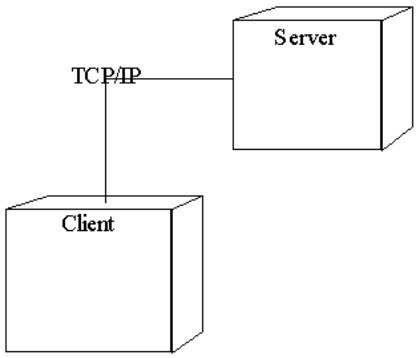

Рисунок 2.35 – Діаграма розгортання

Біля ліній-асоціацій слід вказувати .

11 Використання Visusal Paradigm Community Edition для створення UML-діаграм

Існує велика кількість програмних засобів, що дозволяють створювати і редагувати діаграми UML. Зокрема, це можна зробити за допомогою програмних засобів Visusal Paradigm for UML. Розробник цієї програми - компанія Visual Paradigm International. Існує декілька варіантів реалізації Visusal Paradigm, які відрізняються набором функцій, а також змістом ліцензійної угоди. Це такі варіанти реалізації:

- Community Edition
- Personal Edition
- Modeler Edition
- Standard Edition
- Professional Edition
- Enterprise Edition

Реалізація Visusal Paradigm Community Edition може бути вільно завантажена та з сайту http://www.visual-paradigm.com/solution/freeumltool використана для створення діаграм UML з некомерційною метою. Станом на вересень 2014 року поточна версія програми - 11.2.

Після завантаження програми в робочій частині головного вікна з'являється стартова сторінка (`Start Page`), з якої можна почати безпосереднє створення діаграм. Обираємо групу функцій `UML Modeling` та тип необхідної діаграми.

Взаємозв'язані діаграми створюються та зберігаються в проекті, ім'я якого можна визначити під час першого виклику функції `Save Project` (`Ctrl+S`). Новий проект також можна створити за допомогою функції головного меню `File | New Project`, або клавішною комбінацією `Ctrl+N`. Далі в діалоговому вікні вводимо ім'я проекту та залишаємо інші опції за умовчанням. За умовчанням проекти Visual Paradigm зберігаються у теці `Documents\VPProjects` поточного користувача. Проект зберігається у файлі з розширенням `vpp`.

Варіант Standard Edition (комерційний) підтримує генерацію з діаграми класів вихідного коду різними мовами програмування (C++, Java, C# тощо).

12 Використання програми NClass для створення діаграм класів і генерації вихідного коду

Безкоштовні програмні засоби NClass дозволяють створювати діаграми класів та потім генерувати вихідний код мовами C# і Java. Ця програма також підтримує *зворотне проектування* (reverse engineering), яке полягає в автоматичному створенні діаграми класів на підставі вихідного коду.

Засоби NClass можна безкоштовно завантажити з сайту http://nclass.sourceforge.net.

13 Приклад проектування програми та створення діаграм

Припустимо, необхідно спроектувати програму, яка здійснює читання, модифікацію, додавання, сортування, пошук та зберігання даних про книги на книжковій полиці. Дані слід читати з текстового файлу та записувати модифіковані дані в інший файл. Подальший розвиток програми повинен передбачати взаємодію з XML-документами.

Перелічені вимоги можна представити у вигляді діаграми варіантів використання:

Рисунок 3.1 – Вимоги щодо програми обробки даних про книги

У нашому випадку майже очевидним є необхідний набір класів. Це класи "Книжкова полиця" (Shelf) та "Книга" (Book). Книжкова полиця може мати свою назву та містити набір книг (масив, список, або інша послідовність). Книгу можна охарактеризувати назвою (), списком авторів (authors), роком видання (yearOfPublication), видавництвом (publishingHouse), номером видання (edition) та кількістю сторінок (pages). Дані про автора та видавництво в свою чергу мають свою внутрішню структуру - ім'я та прізвище автора, назва та місто знаходження видавництва. Для автора та видавництва доцільно створити окремі класи. Можна запропонувати таку діаграму класів:

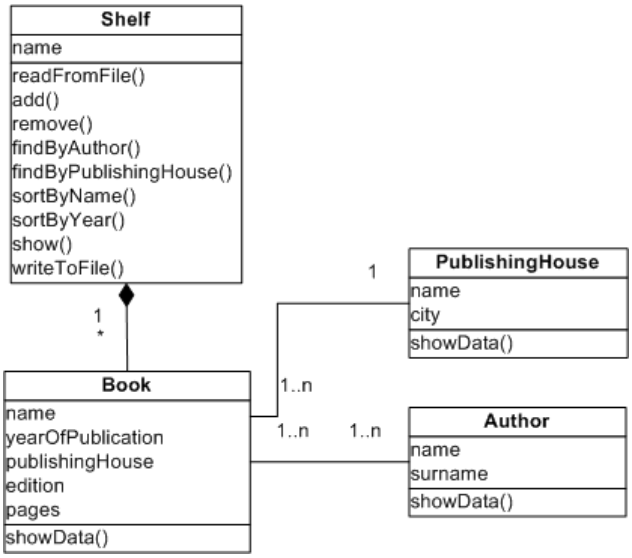

Рисунок 3.2 – Структура класів програмного забезпечення

Під час подальшого проектування та реалізації програмного забезпечення діаграми зазвичай зазнають змін, іноді істотних змін.

Кожному варіанту використання можна поставити у відповідність діаграму послідовності. Іноді діаграми послідовності малюють для окремих операцій. Наприклад, для варіанту використання "Сортувати за роком" можна запропонувати таку діаграму послідовності:

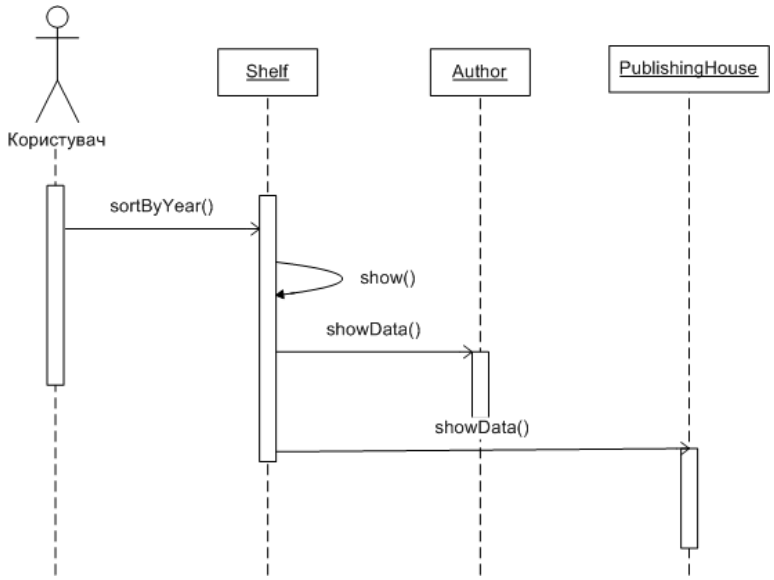

Рисунок 3.3 – Взаємодія класів під час реалізації варіанту використання "Сортувати за роком"

Окремі складні операції вимагають створення діаграм діяльності. Наприклад, це операція пошуку книг з певним автором (findIfAuthor()):

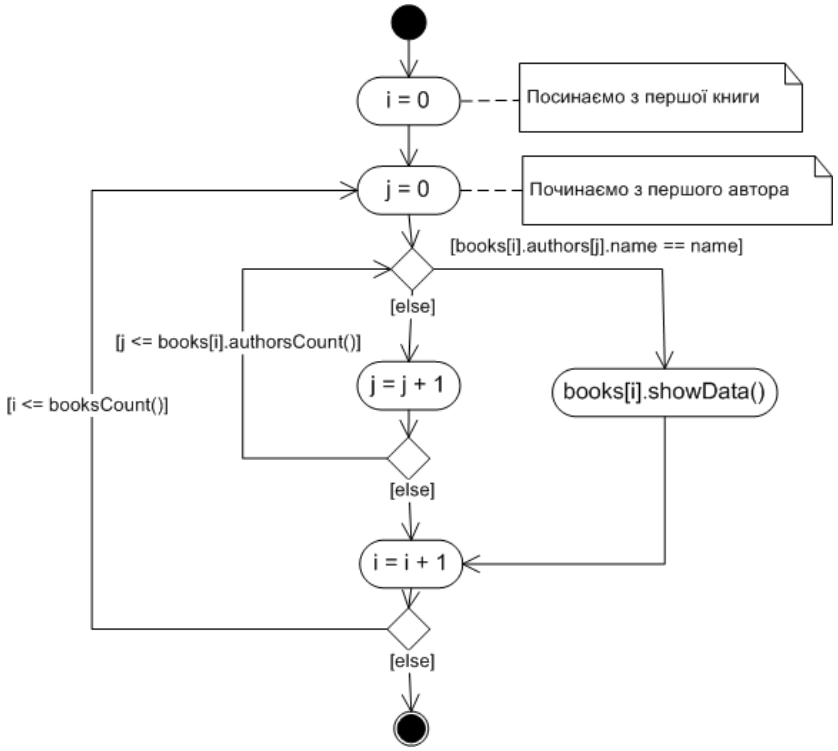

Рисунок 3.4 – Алгоритм пошуку книг певного автора

Аналогічні діаграми можна створити для інших операцій.