**Лекція 4**

 **4UML. Діаграма компонентів**

Діаграма компонентів служить частиною фізичного представлення моделі системи та грає важливу роль в процесі об'єктно-орієнтованого аналізу і проектування, є необхідною для генерації програмного коду. Вона показує різні компоненти системи і залежності між ними.

Компонент являє собою фізичний модуль програмного коду. Компонент часто вважають синонімом пакету, але ці поняття можуть відрізнятися, оскільки компоненти являють собою фізичне об'єднання програмного коду. Хоча окремий клас може бути представлений в цілій сукупності компонентів, цей клас повинен бути визначений тільки в одному певному пакеті. Наприклад, клас String в мові Java є частиною пакета java.lang, але він може бути виявлений в ряді компонентів.

Компонент є фізичною частиною системи, який може бути замінений та має один набор інтерфейсів і забезпечує реалізацію будь-якого іншого компоненту. Приклади зображення компонента представлені на рис. 6.1.





Рис. 6.1. Приклади зображення компонентів

Залежності між компонентами показують, як зміни одного компонента можуть вплинути на зміни інших компонентів. Кількість видів залежностей обмежена.

У CASE-засобах діаграми компонентів, як правило, представлені в Component View.

Часто компоненти містять або використовують інтерфейси. Інтерфейс – це набір операцій, які описують послуги, що надаються класом або компонентом. Істотним є ставлення між компонентом та інтерфейсом. Всі популярні компонентні засоби операційних систем (такі як СОМ+, CORBA та Enterprise JavaBeans) використовують інтерфейси для "склеювання" різних компонентів.

Відносини між компонентом та його інтерфейсами можна зобразити двома способами. Перший, найбільш поширений, полягає в тому, що інтерфейс малюється в згорнутій (elided) формі. Компонент, який реалізує інтерфейс, приєднується до нього за допомогою відносини згорнутої реалізації. У другому випадку інтерфейс малюється в розгорнутому вигляді, можливо з розкриттям операцій. Компонент, який реалізує його, приєднується за допомогою відносини повної реалізації. В обох випадках компонент, який одержує доступ до послуг інших компонентів через цей інтерфейс, підключається до нього за допомогою відносини залежності.



Рис. 6.2. Компоненти та інтерфейси

Інтерфейс, що реалізується компонентом, називається інтерфейсом, що експортується (export interface, component implements interface). Це означає, що компонент через даний інтерфейс надає ряд послуг для інших компонентів. Компонент може експортувати багато інтерфейсів.

Інтерфейс, яким компонент користується, називається інтерфейсом, що імпортується (import interface, component uses interface). Це означає, що компонент сумісний з таким інтерфейсом та залежить від нього при виконанні своїх функцій. Компонент може імпортувати різні інтерфейси, причому йому дозволяється одночасно експортувати та імпортувати різні інтерфейси (рис. 6.1).

На рис. 6.2 компонент, який реалізується в component.java, надає інтерфейс ImageObserver, який використовується компонентом з image.java.

Компоненти можна розділити на три види.

1. Компоненти розгортання (deployment components), які необхідні ідостатні для побудови системи, яка виконується. До їх числа відносяться спільні бібліотеки (DLL) та виконувані програми (EXE). Визначення компонентів в UML досить широко, щоб охопити як класичні об'єктні моделі (на зразок COM+, CORBA та Enterprise JavaBeans), так і альтернативні, що можливо містять динамічні Web-сторінки, таблиці БД та модулі для виконання, де використовуються закриті механізми комунікації.
2. Компоненти-робочі продукти (work product components). По суті, цепобічний результат процесу розробки. Сюди можна віднести файли з вихідними текстами програм і даними, з яких створюються компоненти розгортання. Такі компоненти не беруть безпосередньої участі в роботі системи, але є робочими продуктами, з яких система створюється.
3. Компоненти виконання (execution components). Вони створюються якнаслідок роботи системи. Прикладом може служити об'єкт COM+, екземпляр якого створюється з DLL.

Компоненти, як і класи, можуть бути згруповані в пакети. При організації компонентів між ними можна уточнювати відносини залежності, узагальнення, асоціації (включаючи агрегування) та реалізації.

Стосовно до компонентів, в UML існують такі типи стереотипів:

* executable (для виконання) – визначає компонент, який може

виконуватися у вузлі;

* library (бібліотека) – визначає статичну або динамічну бібліотеку;
* table (таблиця) – визначає компонент, який представляє таблицю БД;
* file (файл) – визначає компонент, який представляє документ, щомістить вихідний текст або дані;
* document (документ) – визначає компонент, який представляє документ.

З виходом нових версій UML, можливе розширення цього списку.

На рис. 6.3 показаний набір компонентів, що входить до складу інструментальної програми, яка працює на одному ПК: програма та чотири бібліотеки. На діаграмі представлені також залежності між компонентами.



Рис. 6.3. Діаграма компонентів програми з бібліотеками

Безпосередній показ залежності між двома компонентами – це насправді згорнута форма подання реальних відносин між ними. Компонент рідко залежить від іншого компоненту безпосередньо. Частіше він імпортує один або декілька інтерфейсів, що експортуються іншим компонентом. Можна було б, наприклад, змінити представлену на рис. 6.3 діаграму, явно вказавши інтерфейси, які реалізує (експортує) бібліотека render.dll та використовує (імпортує) програма animator.exe. Однак, для простоти ці деталі можна приховати, показавши лише, що існує залежність між компонентами.

Можливий розширений показ компонента з елементами діаграми класів

(рис. 6.4).



Рис. 6.4. Розгорнуте представлення компонента

При зображенні компонента в UML рекомендується керуватися наступними правилами:

1. Застосовувати згорнуту форму інтерфейсу, якщо тільки не виникаєгострої необхідності розкрити операції, які пропоновані цим інтерфейсом.
2. Показувати тільки ті інтерфейси, які необхідні для розумінняпризначення компонента в даному контексті.
3. У тих випадках, коли використовуються компоненти для моделюваннябібліотек і вихідного коду, вказувати помічені значення, які стосуються контролю версій.