Тема 6 АРХІТЕКТУРА ПРОГРАМНИХ СИСТЕМ

6.1. ПОНЯТТЯ АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ

Розробка архітектури системи — це процес розбиття великої системи більш дрібні частини. Для позначення цих частин вигадано безліч назв: програми, компоненти, підсистеми...
Процес розробки архітектури — етап, необхідний під час проектування систем чи комплексів, але необов'язковий під час створення програми. Якщо зовнішні специфікації (екранні форми, організація файлів тощо. буд.) описують програмну систему з погляду користувача, наступний крок проектування полягає у розробці архітектури, а й слід проектування структури кожної програми.
Незважаючи на те, що немає точного визначення програмної системи, можна сказати, що вона є набором рішень безлічі різних, але пов'язаних між собою завдань, і далі покластися на інтуїцію у випадках, коли треба відрізнити програму від системи.
Приклади систем: ОС, СУБД, система продажу авіаквитків та ін.
Приклади програм: редактор текстів, компілятор, програми надсилання запитів від касира та ін.
Поняття архітектури програмної системи можна проілюструвати на прикладі. Нехай є на якомусь підприємстві якась САПР. Припустимо, що підприємство досить велике, і САПР буде цілим комплексом різних програмних продуктів, причому найчастіше різних виробників. Архітектурою цієї системи буде опис зв'язків цих програмних засобів в одне ціле. Очима програміста: САПР - комплекс комплексів програм.

6.2. СИСТЕМИ З ОКРЕМИХ ПРОГРАМ

Програмна система може складатися з окремих розроблених різними організаціями програм, що виконуються. Об'єднання функцій цих програм у цілу єдину програму може призвести до нестачі оперативної пам'яті машини, а сама технологія може бути економічно невиправданою.
Близький аналог цієї системи – система, керована командним файлом. Найпростіша архітектура такої системи реалізується послідовним викликом кожної програми. Програми обмінюються даними через файли, записані на диску, або через елементи даних, що знаходяться в оперативній пам'яті ЕОМ за відомими абсолютними адресами.
Вже наприкінці 1970-х років можна було швидко реалізувати дуже зручний введення даних у програму. Стосовно пізнішої операційної системи MS DOS достатньо було написати текстовий файл maket.txt з текстами пояснень суті даних та символом «?» позначити поля даних, що вводяться. Далі готувався командний файл із послідовністю команд:
1) видалення файлу work.txt з диска;
2) копіювання файлу maket.txt у файл work.txt;
3) запуск готової програми текстового редактора із параметром work.txt;
4) запуск програми користувача обробки даних (вхідна інформація програми – файл work.txt, вихідна інформація програми – файл result.txt);
5) запуск готової програми перегляду текстових файлів із параметром result.txt.

Після старту командного файлу користувач у вікні текстового редактора міг читати пояснення щодо введення інформації, знаходити поля введення даних пошуком підрядка із символом «?», вводити значення поля введення з коригуванням. Після введення та коригування даних користувач виходить із програми текстового редактора, що автоматично запускає програму обробки даних, а після завершення її роботи автоматично запускається програма перегляду текстових файлів, яка забезпечує користувачеві можливість перегляду результатів роботи програми обробки даних.
Якщо потрібно реалізувати меню вибору окремих програм, неможливо обійтися послідовним викликом команд командного файла. Для цього випадку можна написати програму, яка візуалізує меню на екрані та повертає номер обраного користувачем меню операційної системи. Повернути номер вибраної теми меню можна викликом підпрограми Halt (номер_мен) модуля DOS Turbo Pascal, де номер_мен — значення вибраного номера теми меню.
Далі, використовуючи команди DOS, організуйте виклик потрібних програм командним файлом:

```
IF ERRORLEVEL 2 GOTO ITEM2
IF ERRORLEVEL 1 GOTO ITEM1
.........................

:ITEM1
.........................
:ITEM2
.........................
```

Використовуючи процедуру Halt для присвоєння системної змінної ERRORLEVEL необхідного значення в кожній із програм та командний файл з вибором потрібної програми, можна створити найпростіший механізм управління порядком виконання програм, коли кожна програма, що завершує роботу, визначає, яка програма повинна виконуватися наступною.
Замість стандартного монітора командних файлів для виклику у довільному порядку вже готових програм можна написати програму свого монітора на основі підпрограми виклику готових програм.
Хоча проектування систем з окремих програм виконується, принаймні, останні 30 років, але не вироблено жодної методики, крім особливо корисної поради: зобразіть функціональну схему процесу, а потім розбийте процес на програми.

6.3. СИСТЕМИ З ОКРЕМИХ РЕЗИДЕНТНИХ ПРОГРАМ

Резидентна програма – програма, яка постійно перебуває в оперативній пам'яті машини та не перешкоджає запуску нових програм. Після запуску резидентна програма стає частиною операційної системи MS DOS шляхом зміни значення межі пам'яті операційної системи, далі вона налаштовує якесь переривання на передачу управління в свою точку входу, а потім завершує роботу. Можна запустити ще кілька резидентних програм та звичайну програму. Після виконання заданих переривань MS DOS запускаються резидентні програми.
Кожна резидентна програма може бути завантажена в будь-якій послідовності. Резидентні програми можуть містити вектор переривань, які вказують на блоки даних кожної з програм. Ці блоки можуть містити ідентифікатор програми контролю наявності програми та дані міжпрограмного обміну. Непотрібні програми можуть бути видалені за допомогою спеціальних програм, так і за допомогою універсальної програми Release.

6.4. СИСТЕМИ З ПРОГРАМ, ОБМІНЮЮЧИХ ДАНИМИ ЧЕРЕЗ ПОРТИ

Такий обмін зазвичай реалізується при багатопроцесорній (багатомашинної) обробці. Порт кожної із програм представляє програму накопичення та верифікації як вхідних, так і вихідних даних у відповідних чергах. У міру виконання поточної роботи з вхідного порту береться чергова порція інформації, обробляється, результати записуються у вихідний порт і далі програма приступає до обробки наступної роботи. Інші програми надсилають інформацію у вхідний порт і забирають результати роботи з вихідного порту.

6.5. ПІДХІД ДО ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ НА ОСНОВІ АБСТРАКТНИХ МАШИН ДЕЙКСТРИ

Найнижчий рівень абстракції – це рівень апаратури. Кожен рівень реалізує абстрактну машину з дедалі більшими можливостями.
1. Принцип 1. На кожному рівні абсолютно нічого не відомо про властивості вищих рівнів. Цим досягається скорочення зв'язків між рівнями.
Принцип 2 На кожному рівні нічого не відомо про внутрішню будову інших рівнів. Зв'язок рівнів здійснюється лише через певні заздалегідь сполучення.
Принцип 3 Кожен рівень є окремо відкомпільованими програмами. Деякі з цих модулів є внутрішніми для рівня, тобто недоступними іншим рівням. Імена інших модулів відомі на вищому рівні і являють собою сполучення з цим рівнем.
Принцип 4 Кожен рівень має певні ресурси і або приховує їх з інших рівнів, або надає іншим рівням деякі їх абстракції. Наприклад, у системі управління файлами один із рівнів може містити фізичні файли, приховуючи їх організацію від решти системи. Інші рівні можуть володіти ресурсами: у каталозі, у словнику даних та ін.
Принцип 5 Кожен рівень може забезпечувати деяку абстракцію даних у системі. Наприклад, файли послідовного та прямого доступу на одному рівні однаково реалізуються на іншому рівні.
Принцип 6 Припущення, які на кожному рівні робляться щодо інших рівнів, повинні бути мінімальними, ці припущення можуть набувати вигляду угод, які повинні дотримуватися перед виконанням функцій, або належати до подання даних або факторів зовнішнього середовища.
*Принцип*7. Зв'язки між рівнями обмежені явними аргументами, що передаються з одного рівня на інший. Неприпустиме спільне використання глобальних даних кількома рівнями. Більше того, бажано повністю виключити використання глобальних даних (навіть усередині рівня) у системі.
Принцип 8 Будь-яка функція, що виконується рівнем абстракції, має бути представлена єдиним входом. Аргументи, що пересилаються між рівнями, мають бути окремими елементами даних, а чи не складними структурами.
Підхід до проектування архітектури системи з урахуванням абстрактних машин Дейкстри можна пояснити на прикладі.
Процесор фірми «Intel» може виконувати операції арифметики і, здійснюючи порівняння двох величин, може виконувати команди переходу на команди в заданих адресах пам'яті. Програмувати таку ЕОМ можна як прямий запис двійкових команд.
ЕОМ IBM PC має спеціальний постійний пристрій з програмами BIOS. Після встановлення BIOS виходить машина з додатковими командами завантаження програми з дисків, читання інформації з будь-якого сектора дисків, читання символу з клавіатури, виведення інформації на екран і т.д. та відсутності співпроцесора.
Після встановлення операційної системи MS DOS на машину IBM PC виходить машина з новою підтримкою даних як файлів і з новими командами роботи з файлами і директоріями (копіювання, видалення тощо. буд.). Нова машина може виконувати операції над речовими числами з плаваючою точкою. З'являються команди запуску файлів та інші нові команди.
Після встановлення операційної системи MS Windows 3.1 (при встановленні операційної системи MS Windows 95 одночасно встановлюється і MS DOS) з'являються нові команди керування вікнами для роботи фахівців зі столами, завареними паперами. З'являється можливість одночасного запуску різних програм у різних вікнах із можливістю між віконного обміну інформацією.
Після інсталяції програмного комплексу Microsoft Office з'являються середовища та команди роботи над документами, розрахунковими таблицями тощо.

6.6. COM - ТЕХНОЛОГІЯ РОЗРОБКИ ПРОГРАМ, ЩО РОЗВИВАЮТЬСЯ І РОЗПОДОРОЧЕНИХ КОМПЛЕКСІВ

COM - Component Object Model (модель компонентних об'єктів) - це специфікація методу створення компонентів та побудови з них програм.
У літературних джерелах можна знайти безліч теорій та пропозицій щодо так званої технології еволюційного програмування. Однак до COM практично невідомі вдалі приклади розробки програм, що еволюціонують у часі. Це неможливістю однозначного передбачення людьми майбутнього. Тому поради на кшталт «передбач то в програмі для майбутнього розвитку» виявлялися безглуздими через те, що в ході супроводу з'ясовувалась потреба в якихось інших доробках, але не в апріорі закладених.

Традиційно програма проектувалася з окремих файлів, модулів чи класів, які компілювалися та компонувалися в єдине ціле.

Компоненти COM є виконуваний код, зазвичай поширюється як динамічно компонованих бібліотек (DLL). Компоненти COM підключаються один до одного динамічно.

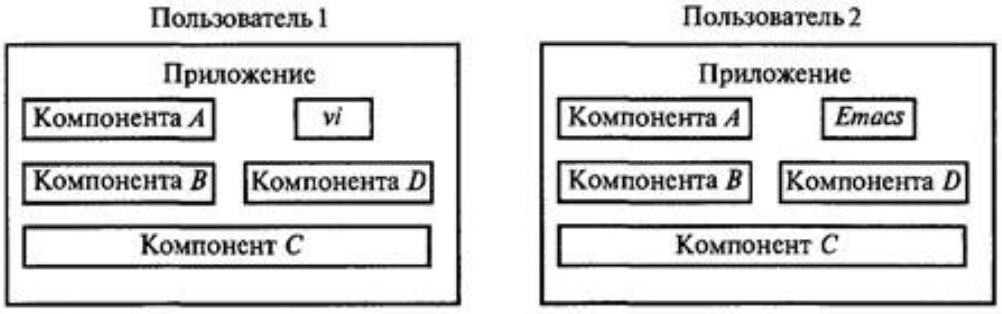
Розробка програм із компонентів про додатків компонентної архітектури відбувається зовсім інакше. З появою COM єдиного цілого немає. Програми складаються з окремих компонентів. Компонента постачається користувачеві як двійковий код, скомпільований, скомпонований та готовий до використання. Доступ до цього коду здійснюється через документований точно інтерфейс. Під час виконання компоненти підключаються до інших компонентів, утворюючи програму.

COM - це інструмент розробки та розосереджених (багатомашинних) комплексів програм, заснована на моделі компонентних об'єктів.

Головне в COM - дотримання стандартних специфікацій інтерфейсу компонент. Якось прийнятий стандарт специфікацій інтерфейсу ніколи не змінюється. Однак після розробки нового стандарту нові компоненти самі будуть впізнавати старі та нові стандарти. Після заміни деякого числа компонентів програма раптом запрацює по-новому!

У міру розвитку програми компоненти, що становлять програму, можуть замінюватися новими. Програма більше не є статичною, приреченою застаріти ще до появи. Натомість вона поступово еволюціонує із заміною старих компонентів новими. З існуючих компонентів легко створити і абсолютно нові програми.

Користувачі часто хочуть адаптувати програми до своїх потреб. Кінцеві користувачі вважають за краще, щоб програма працювала так, як вони звикли. Програмістам у великих організаціях потрібні програми, що адаптуються, щоб створювати спеціалізовані рішення на основі готових продуктів. Компонентні архітектури добре пристосовані для адаптації, оскільки будь-яку компоненту можна замінити на іншу, більш відповідну потребам користувача. Припустимо, що ми маємо компоненти з урахуванням редакторів vi і Emacs (рис. 6.1). Користувач 1 може налаштувати програми використання vi, тоді як користувач 2 віддасть перевагу Emacs. Програми можна легко налаштовувати, додаючи нові компоненти або замінюючи наявні.



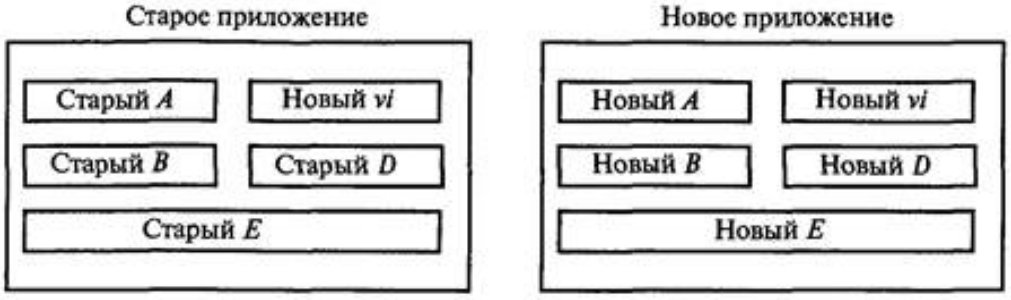
*Мал. 6.1.*Збираються під час виконання програми з окремих компонентів

Один із найперспективніших аспектів впровадження компонентної архітектури — швидка розробка програм. Ви зможете вибирати компоненти з бібліотеки та складати з них, як із деталей конструктора, цілісні додатки методом морфологічного синтезу! Практично всі програми Microsoft, які продаються сьогодні, використовують COM. Технологія ActiveX цієї фірми побудована на основі компонентів COM. Програмісти Visual Basic, C++, Delphi і Java можуть скористатися керуючими елементами ActiveX для прискорення розробки своїх додатків і сторінок Web. Звичайно, кожному додатку, як і раніше, будуть потрібні і деякі спеціалізовані компоненти, але для побудови простих додатків можна обійтися стандартними.

Створити зі звичайної програми розподілену програму легше, якщо ця проста програма складається з компонентів. По-перше, вона вже розділена на функціональні частини, які можуть розташовуватися далеко один від одного. По-друге, оскільки компоненти замінюються, замість деякої компоненти можна підставити іншу, єдиним завданням якої забезпечуватиме зв'язок з віддаленою компонентою.

Переваги використання компонентів безпосередньо випливають із здатності останніх підключатися до додатку і відключатися від нього. Для цього компоненти повинні відповідати двом вимогам. По-перше,

вони мають компонуватися динамічно. По-друге, повинні приховувати (або інкапсулювати) деталі реалізації.
Щоб зрозуміти, як це з інкапсуляцією, необхідно визначити деякі терміни. Програма або компонент, що використовує іншу компоненту, називається клієнтом (client). Клієнт приєднується до компонента через інтерфейс (interface). Якщо компонент змінюється без зміни інтерфейсу, то змін у клієнті не потрібно. Аналогічно якщо клієнт змінюється без зміни інтерфейсу, немає необхідності змінювати компоненту. Однак якщо зміна або клієнта, або компоненти викликає зміну інтерфейсу, то й інший бік інтерфейсу також необхідно змінити.
Таким чином, для того, щоб скористатися перевагами динамічного компонування, компоненти та клієнти повинні намагатися не змінювати свої інтерфейси та бути інкапсулюючими. Деталі реалізації клієнта або компоненти не повинні відображатись в інтерфейсі. Чим надійніше інтерфейс ізольований від реалізації, тим менш імовірно, що він зміниться під час модифікації клієнта чи компоненти. Якщо інтерфейс не змінюється, зміна компоненти надає лише незначне впливом геть додаток загалом. Необхідність ізоляції клієнта від деталей реалізації накладає на компоненти низку важливих обмежень, список яких наведено нижче.
Обмеження 1.Компонента повинна приховувати мову програмування, що використовується. Компоненти можуть бути розроблені за допомогою практично будь-якої процедурної мови, включаючи Ada, C, Java, Modula-3, Oberon та Pascal. Будь-яку мову, у тому числі Smalltalk та Visual Basic, можна пристосувати до використання компонентів COM. Будь-який клієнт повинен мати можливість використовувати компоненту незалежно від мов програмування, на яких написано ту чи іншу.
Обмеження 2.Компоненти повинні поширюватись у двійковій формі. Дійсно, оскільки вони повинні приховувати мову реалізації, їх необхідно постачати вже відкомпільованими та готовими до використання (DLL).
Обмеження 3.Компоненти COM можна модернізувати, не порушуючи роботи старих клієнтів. COM надає стандартний спосіб реалізації різних версій компонентів. Нові версії компонентів повинні працювати як з новими клієнтами, так і старими.
Обмеження 4.Компоненти COM є переміщуються по мережі, причому переміщення по мережі має бути прозорою. Компонент на віддаленій системі розглядається так само, як компонент на локальному комп'ютері. Необхідно, щоб компонент і програма, що використовує її, могли виконуватися всередині одного процесу, в різних процесах або на різних машинах. Клієнт повинен розглядати віддалену компоненту так само, як локальну. Якби з віддаленими компонентами треба було б працювати інакше, ніж з локальними, то потрібно було б перекомпіляція клієнта щоразу, коли локальна компонента переміщається в інше місце мережі.
Користувач може мати дві клієнтські програми, які використовують одну й ту саму компоненту. Припустимо, що один додаток застосовує нову версію цієї компоненти, а інший стару. Інсталяція нової версії не повинна порушувати роботу програми, яка використовувала стару версію. Стара програма використовує нову компоненту vi абсолютно так само, як це робить нове (рис. 6.2).
Проте зворотна сумісність має обмежувати розвиток компонент. Потрібно, щоб поведінка компоненти нових додатків можна було радикально змінювати, не порушуючи підтримку старих додатків.



*Мал. 6.2.*Поетапне отримання нових додатків

Таким чином, технологія передбачає взаємозамінність компонентів під час виконання за допомогою встановлення стандарту, якому повинні дотримуватися компоненти; практично прозорої підтримки

кількох версій компоненти; забезпечення можливості роботи із подібними компонентами однаковим способом; визначення архітектури, незалежної від мови; підтримки прозорих зв'язків із віддаленими компонентами.

ВИСНОВКИ

• Розробка архітектури - це процес розбиття великої системи на дрібніші частини. Процес розробки архітектури — етап, необхідний під час проектування систем чи комплексів, але необов'язковий під час створення програми. Якщо зовнішні специфікації (екранні форми, організація файлів...) описують програмну систему з погляду користувача, наступний крок проектування полягає у розробці архітектури, а потім слідує проектування структури кожної програми.
• Програмна система може складатися з окремих, розроблених різними організаціями програм, що виконуються; із програм, що обмінюються даними через порти; з окремих резидентних програм.
• Традиційно програма проектувалась з окремих файлів, модулів або класів, які компілювалися та компонувалися в єдине ціле.
• Розробка програм з компонентів - так званих додатків компонентної архітектури - відбувається зовсім інакше. З появою технології розробки та розосереджених (багатомашинних) комплексів програм, заснованої на моделі компонентних об'єктів (COM), єдиного цілого більше немає: програми складаються з окремих компонентів. Компонента постачається користувачеві як двійковий код, скомпільований, скомпонований та готовий до використання. Доступ до цього коду здійснюється через документований інтерфейс. Під час виконання компоненти підключаються до інших компонентів, утворюючи програму.

Контрольні питання

1. Що таке архітектура програм?
2. Чи є синонімами поняття «структура» та «архітектура»?
3. У чому полягає процес розробки архітектури програми?
4. Як реалізується архітектура системи окремих програм?
5. Що таке резидентна програма?
6. Як здійснюється обмін даними через порти?
7. Перерахуйте принципи підходу до проектування архітектури системи з позиції рівнів абстракції Дейкстри.
8. Чому зі звичайної програми створити розподілену програму легше, якщо вона складається із компонентів?
9. Перерахуйте низку обмежень, що накладаються на компоненти.
10. За допомогою чого передбачається взаємозамінність компонентів?