Phát triển phần mềm phân tán

CHƯƠNG II.
NGUYÊN TẮC THIẾT KẾ

I. Mô hình ứng dụng phân tán
II. Nguyên lý thiết kế chung
III. Một số mẫu thiết kế tiêu biểu

Nhắc lại một số khái niệm

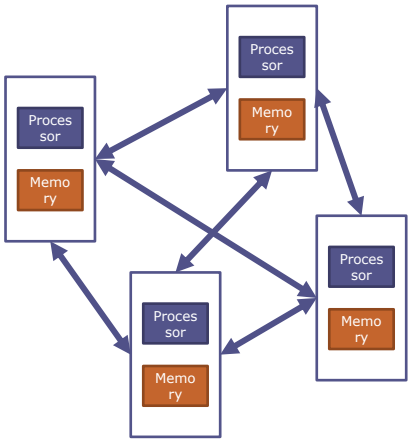
- Program:
- Process:
- Message:
- Packet:
- Protocol:

- Mạng:

- Thành phần:

I. MÔ HÌNH ỨNG DỤNG PHÂN TÁN

1. Mở đầu



- Ứng dụng phân tán: tập các tiến trình cùng hoạt động trên các máy nối mạng nhằm giải quyết một vấn đề cụ thể.
- Các đặc điểm cần lưu ý:
  - Truyền thông giữa các tiến trình (Interprocess communication):
  - Đồng bộ các sự kiện (Event synchronization):

CHƯƠNG II. NGUYÊN TẮC THIẾT KẾ

I. Mô hình ứng dụng phân tán

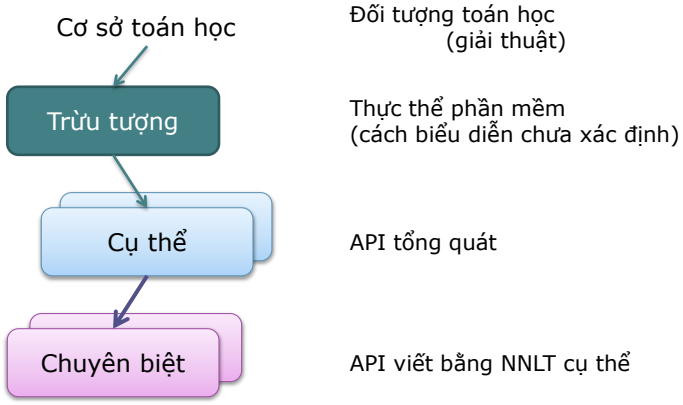
1. Mở đầu

1.1. Mô hình kiến trúc phần mềm

- Là gì ?
  - Miêu tả một khía cạnh (quan điểm, chức năng) của kiến trúc phần mềm
- Dùng để làm gì ?
  - Giải thích
  - Dự báo
  - Chứng minh
  - Hướng dẫn thực hiện

  các thao tác thực hiện của 1 CT

1.2. Phân cấp các mô hình kiến trúc phần mềm



CHƯƠNG II. NGUYÊN TẮC THIẾT KẾ
I. Mô hình ứng dụng phân tán
1. Mở đầu

1.3. Phân loại các mô hình kiến trúc ứng dụng phân tán

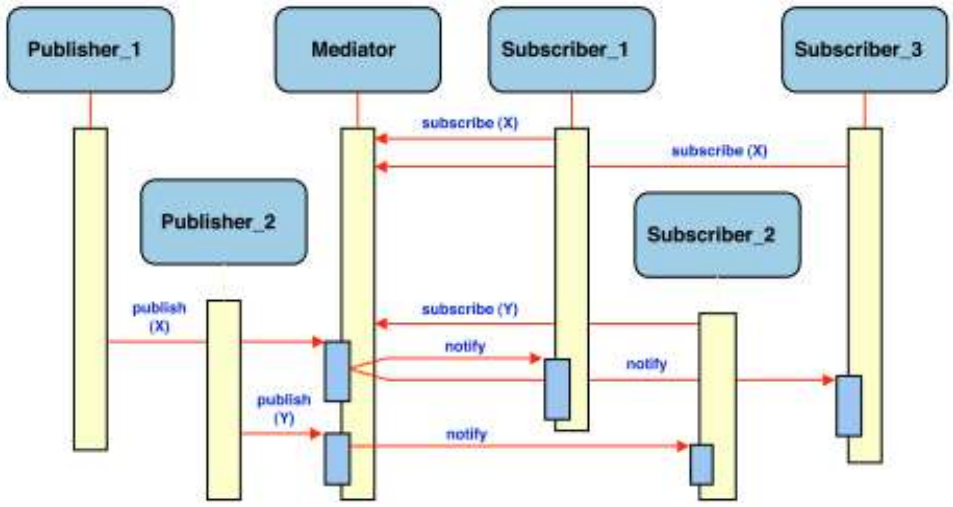
- Theo bản chất của luồng điều khiển thực hiện công việc
  - Đồng bộ (client – server)
  - Không đồng bộ (truyền thông điệp, sự kiện)
  - Hỗn hợp
- Theo đơn vị tổ chức
  - Đối tượng phân tán
  - Thành phần phân tán
  - Dịch vụ phân tán
- Theo cấu trúc tĩnh hoặc động
  - Khả năng di động của các phần tử
  - Khả năng cấu hình lại

→ Phần lớn đều là các mô hình chuyên biệt hoặc cụ thể

2.1. Bối cảnh và vấn đề

- Bối cảnh:
  - Tập các thực thể cần phối hợp thực hiện thông qua cách truyền các sự kiện và phản ứng lại các sự kiện này
- Vấn đề:
  - Các đặc tính cần có:
    - Độc lập, có khả năng tiến hóa
    - Không cần định nghĩa trước vai trò
    - Lựa chọn các thực thể phối hợp thực hiện dựa trên bản chất của sự kiện
  - Ràng buộc
    - Dễ mở rộng phạm vi
    - Dung lỗi, giao dịch, bền vững, có thứ tự

2.2. Giải pháp

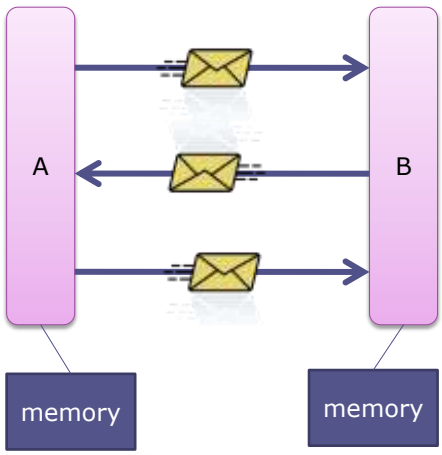


2.2. Giải pháp

- 2 vai trò:
  - Publisher: Sinh ra dữ liệu dưới dạng các sự kiện (event)
    - Sơ đồ các sự kiện là biết trước và cố định
  - Subscriber: Nhận các dữ liệu tương ứng với kiểu thuê bao đã đăng ký
    - Đăng ký nhận dữ liệu theo chủ đề (tĩnh) hoặc theo nội dung (động)
- 1 thực thể trung gian (mediator)
  - Chức năng thông báo sự kiện thường được cài đặt theo:
    - Mô hình tập trung: duy nhất 1 mediator
    - Mô hình phân tán: nhiều mediator, mỗi mediator phụ trách nhiều nhất N đăng ký.
  - Các cơ chế định tuyến sự kiện:
    - Thông báo cho tất cả các thuê bao: Flooding
    - Thông báo có lựa chọn: Filter-based; Rendez-Vous

→ Có thể xây dựng nhiều kiểu ứng dụng phân tán, tùy thuộc vào mô hình đăng ký nhận dữ liệu và mô hình mediator

3. Mô hình trao đổi thông điệp (message passing model)



- Tập các tiến trình sử dụng bộ nhớ tại chỗ
  - Giao tiếp bằng cách gửi và nhận các thông điệp
  - Việc trao đổi dữ liệu thể hiện hoạt động cộng tác của các tiến trình:
    - Tiến trình A gửi thông điệp đến tiến trình B, thông điệp này thể hiện yêu cầu của A
    - Thông điệp được chuyển tới B, B xử lý thông điệp này và có thể gửi thông điệp trả lời A
    - Việc trả lời một thông điệp có thể kích hoạt việc gửi một yêu cầu khác, và cứ thế tiếp mãi

Các phép toán hỗ trợ

- Send: gửi thông điệp
- Receive: nhận thông điệp
- Connect/disconnect: tạo/hủy kết nối

→ Đóng vai trò của các phép toán vào/ra

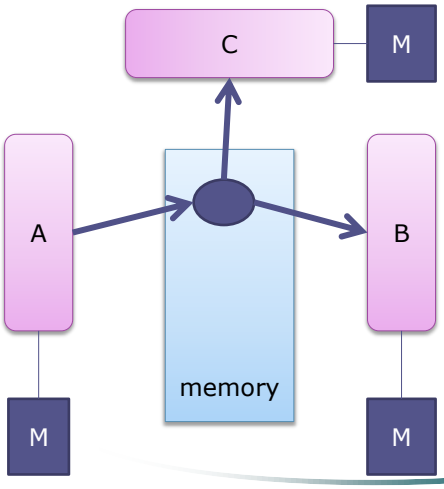
→ Che giấu quá trình truyền thông mạng ở tầng hệ điều hành

- Ví dụ: giao diện lập trình socket

Cài đặt mô hình truyền thông điệp

- Process management routines
  - Khởi tạo và kết thúc tiến trình
  - Xác định số tiến trình liên quan
  - Định danh các tiến trình
- Point-to-point communication routines
  - Gửi và nhận dữ liệu giữa 2 tiến trình bất kỳ
- Process group/collective communication routines
  - Gửi dữ liệu cho tất cả các tiến trình
  - Tìm kiếm các tiến trình
  - Gửi dữ liệu đến một số tiến trình
  - Đồng bộ các tiến trình
- Ví dụ:
  - IBM Message Queue Series: http://www-4.ibm.com/software/ts/mqseries/
  - Microsoft's Message Queue (MSQ), http://msdn.microsoft.com/library/psdk/msmq/msmq_overview_4ilh.htm
  - Java's Message Service http://developer.java.sun.com/developer/technicalArticles/Networking/messaging/

4. Mô hình chia sẻ dữ liệu (shared memory model)



- Nhiều tiến trình chia sẻ bộ nhớ
- Việc truy nhập vào vùng nhớ lưu trữ các biến dùng chung thể hiện hoạt động cộng tác của các tiến trình:
  - A (publisher) viết dữ liệu lên vùng nhớ dùng chung
  - B, C (requestor) đăng ký chia sẻ bộ nhớ và lấy dữ liệu từ vùng nhớ dùng chung về để xử lý
  - Kết quả xử lý có thể được viết lên vùng nhớ dùng chung

Các phép toán hỗ trợ

- Create/Delete: tạo/hủy biến dùng chung
- Write: ghi dữ liệu (gán giá trị cho biến dùng chung)
- Read: đọc dữ liệu (đọc giá trị biến dùng chung)
- Modify: thay đổi dữ liệu (đọc giá trị 1 biến, xử lý rồi cập nhật giá trị biến đó)
- Compare-and-swap: thay đổi giá trị một biến nếu thỏa mãn điều kiện nhất định
- Swap: ghi giá trị mới vào biến và trả về giá trị cũ của biến

Cài đặt mô hình chia sẻ bộ nhớ

- Che giấu chi tiết của các thao tác tìm kiếm đối tượng dùng chung, cũng như sự phối hợp của các tài nguyên dùng chung
- Ví dụ:

http://java.sun.com/products/javaspaces/

Lưu ý

- Các mô hình này độc lập với kiến trúc và nền tảng thực thi ứng dụng phân tán
- Các mô hình này có thể được cài đặt trên nền các hệ điều hành tương thích với các thiết bị phần cứng khác nhau
- Hiệu quả của việc cài đặt này phụ thuộc vào sự phù hợp giữa các đặc điểm của mô hình với các thiết bị phần cứng

II. NGUYÊN LÝ THIẾT KẾ CHUNG

Các vấn đề thường gặp khi thiết kế phần mềm và ứng dụng phân tán

- Kiến trúc phần mềm
  - Đơn vị tổ chức, quan hệ giữa chúng
- Thiết kế và ghép nối
- Đảm bảo an toàn an ninh
- Dung lỗi
- Chất lượng dịch vụ
  - Hiệu năng của phần mềm
  - Khả năng mở rộng phạm vi
- Quản lý

1. Quản lý kiến trúc phần mềm

- Encapsulation: phân tách phần giao diện và phần thực hiện
- Abstraction: phân chia thành nhiều mức, che giấu những chi tiết không thích đáng ở mỗi mức độ xét đến
- Phân tách giữa chính sách và kỹ thuật
  - Không cài đặt lại các kỹ thuật khi thay đổi chính sách
  - Không thậm xưng hóa phần đặc tả các kỹ thuật
- Cô lập và biểu diễn các khía cạnh vượt chức năng một cách độc lập (bên ngoài giao diện chức năng)

2. Thực hành thiết kế và ghép nối

- Chia tách các việc cần làm trước
  - Tách biệt các khía cạnh độc lập (hoặc gần như độc lập) và xử lý chúng riêng rẽ
  - Khảo sát lần lượt từng vấn đề
  - Loại bỏ các phần việc giao thoa
  - Tham số hóa các khía cạnh tiến hóa một cách độc lập

3. Đảm bảo an toàn an ninh

- Tính toán cẩn thận số lượng dữ liệu truyền qua mạng và đặt mục tiêu giảm thiểu nó.
- Thời gian trễ (latency): khoảng thời gian tính từ khi khởi tạo một yêu cầu về dữ liệu cho đến khi dữ liệu được yêu cầu được truyền qua mạng. 2 lựa chọn để giảm thiểu thời gian trễ:
  - Gửi nhiều yêu cầu truyền lượng nhỏ dữ liệu
  - Gửi 1 yêu cầu truyền lượng lớn dữ liệu
- Không đặt giả thiết là dữ liệu được bên gửi gửi đi qua mạng đúng là dữ liệu bên nhận nhận được. Phải kiểm tra xem dữ liệu có bị thay đổi trong quá trình truyền thông hay không.

4. Xử lý lỗi

- Tránh việc đặt giả thiết là một thành phần nào đó trong hệ thống phải luôn ở một trạng thái nhất định thì hệ thống mới thực hiện được
  - Tiến trình trên máy A gửi dữ liệu đến tiến trình trên máy B để thực hiện.
  - A nhận kết quả do B trả về, xử lý và gửi trả lại kết quả cho B
  - Việc A giả sử B luôn sẵn sàng nhận kết quả này là không hợp lý: A phải lường trước các lỗi có thể xảy ra
- Chỉ rõ các kịch bản lỗi và xác định các lỗi tương tự có khả năng xảy ra. Đảm bảo là mã nguồn CT có tính đến hầu hết các lỗi này
- Cả bên gửi và bên nhận đều phải sẵn sàng đối phó với tình huống trong đó đối tác không trả lời.

5. Đảm bảo chất lượng

- Các chiến lược tạo bộ đệm và sao lưu phải nhằm mục đích giảm thiểu số lượng các "stateful components" trong HPT
  - Tạo bộ đệm (cache): tái cấu trúc trạng thái của một tiến trình tại một địa điểm khác với địa điểm mà tiến trình đó đang thực thi.
    - Dữ liệu trong bộ đệm có thể bị cũ, nên cần có chính sách kiểm tra trước khi sử dụng
    - Nếu tiến trình P lưu thông tin I không thể tái cấu trúc được, và P là duy nhất, làm thế nào để khai thác I khi P không chạy thông ?
  - Sao lưu (replication): giảm thiểu rủi ro khi khai thác thông tin không thể tái cấu trúc được.
    - A, B, C là 3 bản sao của tiến trình P. Nếu 1 tiến trình P' gọi A và thay đổi 1 số dữ liệu, sau đó gọi B, vậy:
      - Sự thay đổi về dữ liệu này đã được cập nhật ở B hay chưa?
      - Điều gì xảy ra khi các bản sao của P không thể giao tiếp với nhau?
      - Có bản sao nào của P xủa lý được yêu cầu của P' hay không?

5. Đảm bảo chất lượng

- Tốc độ và hiệu năng: xác định phần nào của hệ thống cần cải thiện hiệu năng
  - Có nút cổ chai (bottle neck) hay không? Tại sao?
  - Lựa chọn giải pháp thích hợp nhất bằng cách thử các trường hợp khác nhau.
- Tránh sử dụng giao thức đòi hỏi xác nhận là đã nhận được dữ liệu (acks) nếu có thể
- Việc truyền lại dữ liệu (retransmission) là rất tốn kém, và mất thời gian. Cần tối ưu hóa việc này

III. MỘT SỐ MẪU THIẾT KẾ TIÊU BIỂU

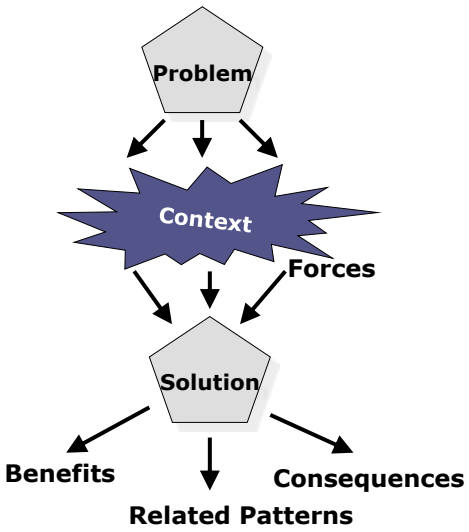
Mở đầu

- Mẫu thiết kế (design pattern): miêu tả vấn đề và bản chất của những cách giải quyết vấn đề đó trong bối cảnh cụ thể
- Dựa trên các đặc tính của đối tượng như kế thừa hay đa hình
- Đủ trừu tượng để tái sử dụng nhằm đáp ứng một lớp yêu cầu đặt ra trong các bối cảnh khác nhau

→ Cách thức để tái sử dụng các tri thức về một vấn đề và các cách giải quyết vấn đề đó

  - Ở mức độ trừu tượng cao hơn
  - Không liên quan đến các tiêu chí thiết kế cụ thể như sử dụng CTDL nào: danh sách liên kết hay bảng băm
  - Miêu tả tổng quát đối tượng và cách truyền thông

Cấu tạo của mẫu thiết kế



- Name
  - Định danh mẫu thiết kế
- Problem description
- Solution description
  - Không phải là một thiết kế cụ thể
  - Là một template cho một giải pháp thiết kế; giải pháp này có thể được cài đặt riêng cho từng trường hợp cụ thể theo các cách khác nhau
  - Khía cạnh tĩnh: các thành phần, quan hệ giữa chúng, có thể miêu tả bằng biểu đồ lớp hay biểu đồ cộng tác
  - Khía cạnh động: hành vi thực hiện, chu kỳ sống, có thể miêu tả bằng biểu đồ tuần tự hay biểu đồ trạng thái
- Consequences
  - Kết quả hay lợi ích của việc áp dụng mẫu

CHƯƠNG II. NGUYÊN TẮC THIẾT KẾ

I. Mô hình ứng dụng phân tán

II. Nguyên lý thiết kế chung

III. Một số mẫu thiết kế tiêu biểu

1. Bao gói (Wrapper)

- Chuyển đổi các giao diện không tương thích
- Dễ dàng:
  - Sửa đổi giao diện
  - Mở rộng các hành vi của một đối tượng
  - Hạn chế truy nhập vào dữ liệu mà đối tượng đó bao gói
- Lớp bao gói phụ trách phần lớn các công việc:

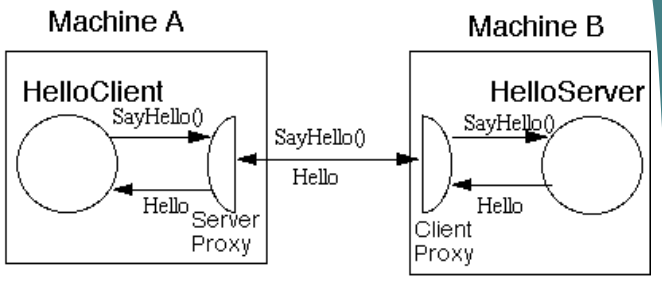
Mẫu	Chức năng	Giao diện
Adapter	Giống nhau	Khác nhau
Decorator	Khác nhau	Giống nhau
Proxy	Giống nhau	Giống nhau

a. Đại diện (Proxy)

- Bối cảnh:
  - ứng dụng bao gồm các đối tượng phân tán, 1 đối tượng (client) có thể truy cập đến các chức năng cung cấp bởi các đối tượng từ xa (server)
- Vấn đề: định nghĩa một kỹ thuật truy cập trong môi trường phân tán tránh cho client phải
  - Chỉ rõ vị trí của server trong mã nguồn (hard code)
  - Quan tâm đến các chi tiết kỹ thuật của giao thức truyền thông
- Các đặc tính mong muốn:
  - Truy cập hiệu quả, an toàn
  - Lập trình phía client đơn giản, không phân biệt việc truy cập các đối tượng tại chỗ hay ở xa
- Giải pháp
  - Sử dụng một đại diện của server trên máy client (tách biệt client của server và client của hệ truyền thông)
  - Giữ nguyên giao diện của đại diện và giao diện của server
  - Định nghiã một cấu trúc đồng nhất để tự đọng sinh ra đại diện này

Ví dụ: remote proxy

Đối tượng cần truy cập nằm trên máy khác (không sử dụng không gian địa chỉ tại chỗ)
Che giấu các chi tiết khi truy cập đối tượng này
Sử dụng trong CORBA, Java RMI




```
public class HelloClient {
    public static void main(String args[]) {
        try { String server = getHelloHostAddress( args);
                Hello proxy = (Hello) Naming.lookup( server );
                String message = proxy.sayHello();
                System.out.println( message );
        }
        catch ( Exception error) { error.printStackTrace(); }
    }
}
```

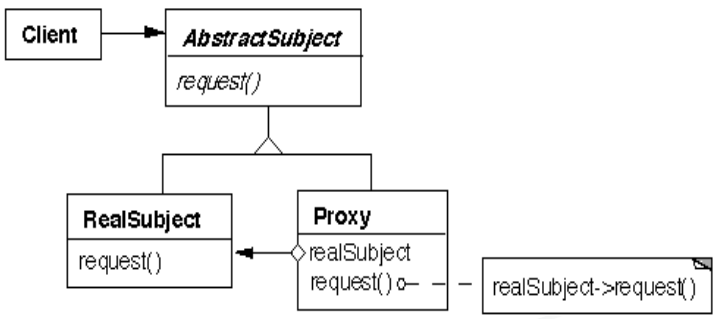
a. Đại diện (Proxy)

- Vật thay thế
- Có quyền hoạt động nhân danh đối tượng khác
- Điều khiển truy nhập đến các đối tượng khác
  - Truyền thông: quản lý mạng khi sử dụng các đối tượng ở xa
  - Khóa (locking): tuần tự hóa sự truy nhập của nhiều khách hàng
  - Đảm bảo an toàn: chỉ cho phép truy nhập nếu đủ tin cậy
  - Tạo mới: các đối tượng chưa tồn tại
    - Che giấu thời gian trễ khi tạo mới đối tượng
    - Tránh kích hoạt các đối tượng không được sử dụng

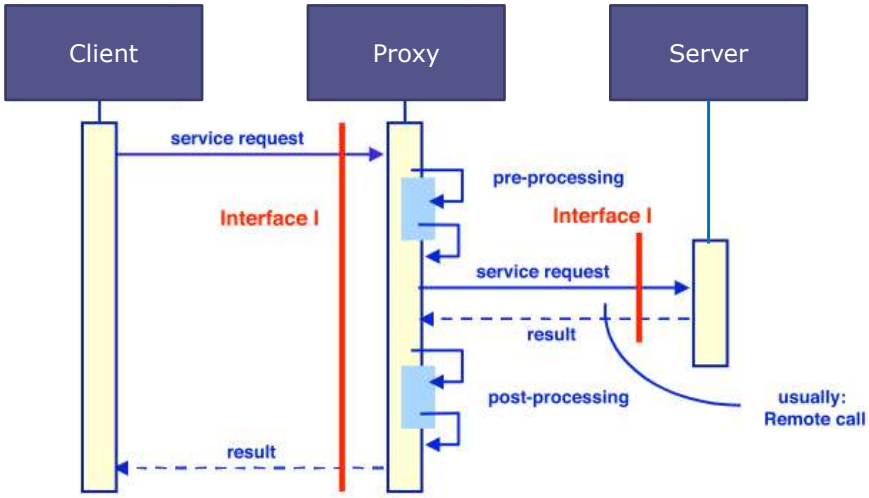
CHƯƠNG II. NGUYÊN TẮC THIẾT KẾ
I. Mô hình ứng dụng phân tán
II. Nguyên lý thiết kế chung
III. Một số mẫu thiết kế tiêu biểu
1. Bao gói

a. Đại diện (Proxy)

- Có giao diện và chức năng giống như đối tượng gốc
- Sử dụng giao diện chung (hoặc lớp ảo) cho cả proxy và đối tượng gốc
- Proxy chứa tham chiếu đến dối tượng gốc, do đó có khả năng chuyển truy vấn đến đối tượng gốc



Sử dụng proxy



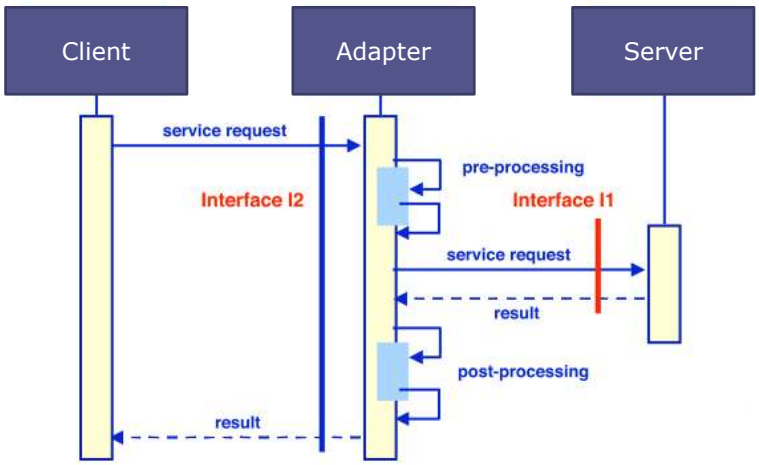
b. Chuyển đổi (adapter)

- Bối cảnh
  - Client yêu cầu các server cung cấp các chức năng thông qua giao diện
- Vấn đề
  - Tái sử dụng server hiện có, chỉ thay đổi giao diện của server này để thỏa mãn nhu cầu của một hoặc 1 lớp khách hàng
- Các đặc tính mong muốn
  - Hiệu quả
  - Thích nghi với các nhu cầu sử dụng khác nhau
  - Có thể thay đổi mà client không biết
  - Có khả năng tái sử dụng
- Giải pháp:
  - Bộ chuyển đổi chặn các lời gọi đến các phương thức qua giao diện của server.
  - Client gọi chức năng của bộ chuyển đổi thay vì gọi chức năng của server
  - Các thông số và kết quả có thể được chuyển đổi

b. Chuyển đổi (adapter)

- Thay đổi giao diện không thay đổi chức năng
  - Đổi tên phương thức
  - Chuyển đổi đơn vị tính
  - Cài đặt phương thức thay thế cho phương thức khác

Sử dụng adapter



Ví dụ: scaling rectangle

```
interface Rectangle {
// grow or shrink this by the
   given factor
void scale(float factor);
...
float getWidth();
float area();
}
class myClass {
void myMethod(Rectangle r) {
...   r.scale(2);   ...
}
}
```

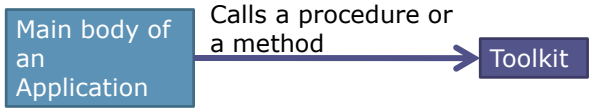
- Có thể sử dụng lớp này để thay thế hay không?

```
class NonScaleableRectangle {
void setWidth(float width) {
 ... }
void setHeight(float height) {
... }
...
}
```

2. Đại lý (Factory)

- Bối cảnh
  - ứng dụng = tập các đối tượng trong môi trường phân tán
- Vấn đề: tạo mới một cách linh hoạt nhiều instance của một lớp đối tượng
- Đặc tính mong muốn
  - Phải tham số hóa được các instance
  - Việc tiến hóa phải dễ dàng (các quyết định không được « hard code »
- Giải pháp
  - Abstract Factory : định nghĩa giao diện và cách tổ chức chung nhằm tạo mới các đối tượng; giao việc tạo đối tượng thực sự cho các đại lý cài đặt các phương thức khởi tạo cụ thể
  - Cài đặt Abstract Factory bằng các Factory Methods (định nghĩa lại các phương thức tạo mới bằng các lớp con)
  - Cung cấp khả năng tham số hóa các đại lý (Factory Factory)

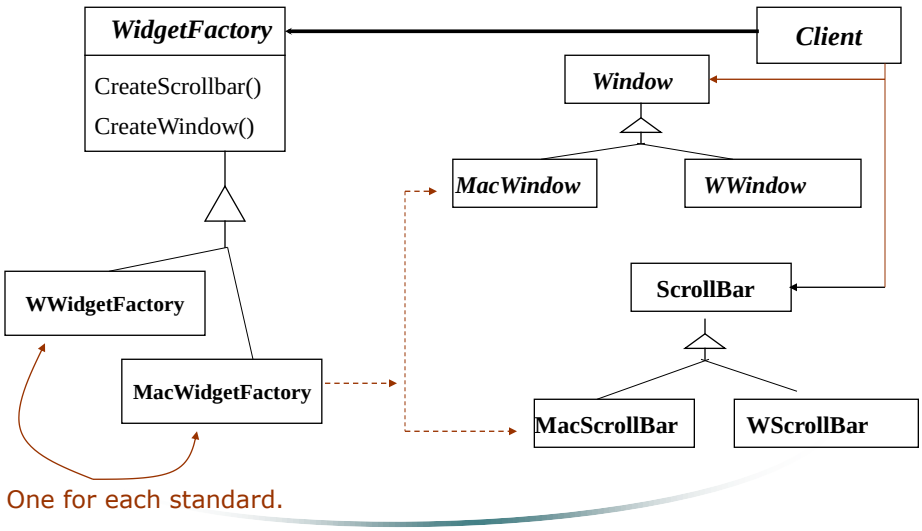
Ví dụ về factory



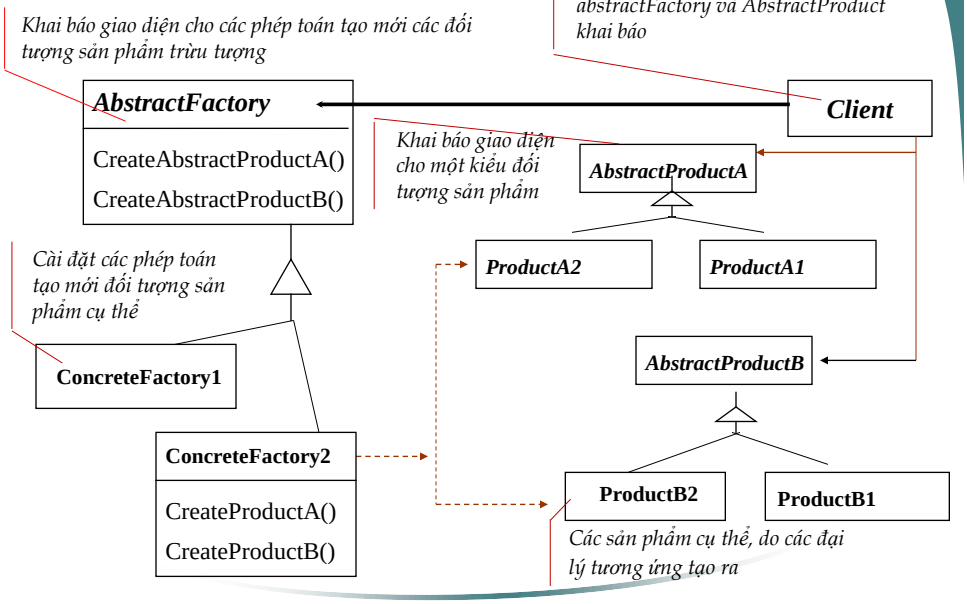
- Toolkits: tập các lớp có quan hệ với nhau và có khả năng tái sử dụng, e.g. C++ I/O stream library
- Xét toolkits hỗ trợ xây dựng giao diện người dùng theo nhiều chuẩn "look-and-feel"
- Để khả chuyển, một ứng dụng không thể "hard code" các widgets cho một look-and-feel duy nhất.

→ Làm thế nào để thiết kế ứng dụng sao cho việc chèn thêm các yêu cầu "look-and-feel" mới trở nên dễ dàng?

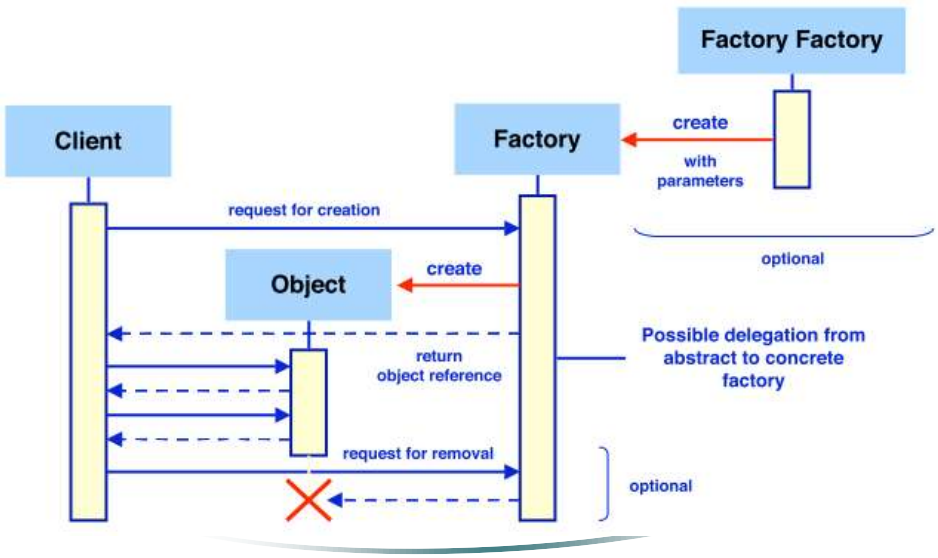
Ví dụ về factory



2. Đại lý



Sử dụng factory

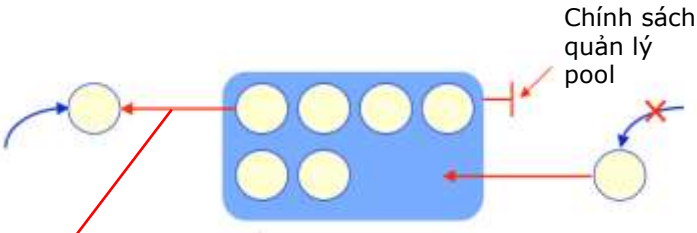


3. Nhóm trực (Pool)

- Bổ sung cho mẫu thiết kế Factory
- Ý tưởng: giảm chi phí quản lý tài nguyên
  - Kỹ thuật: tái sử dụng các mẫu sẵn có

CHƯƠNG II. NGUYÊN TẮC THIẾT KẾ
I. Mô hình ứng dụng phân tán
II. Nguyên lý thiết kế chung
III. Một số mẫu thiết kế tiêu biểu

3. Nhóm trực (Pool)



Tạo mới:
Nếu pool không rỗng:
lấy một phần tử từ pool ra
Xóa phần tử đó trong pool/
khởi tạo lại pool
Ngược lại
tạo ra một instance mới

Xóa:
Đặt instance đó vào pool

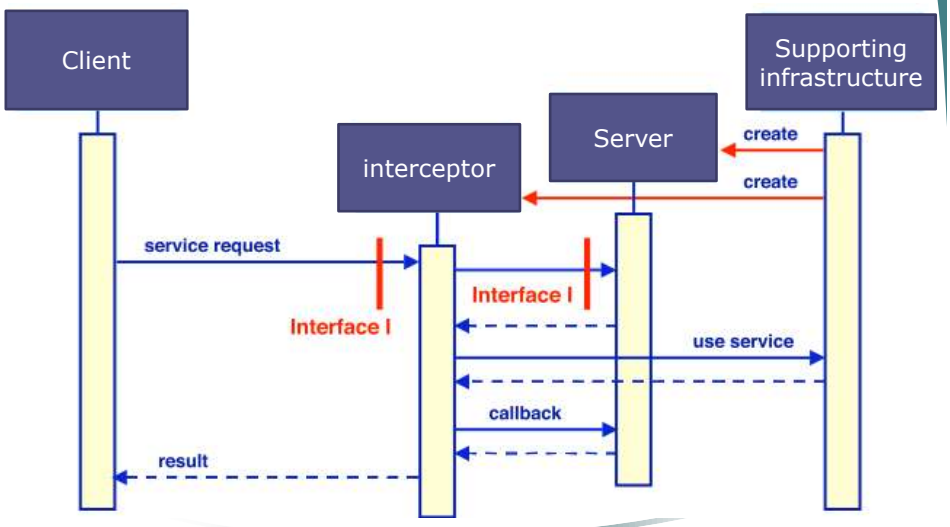
Sử dụng pool

- Quản lý bộ nhớ
  - Pool của các vùng nhớ với các kích thước khác nhau
  - Tránh các thao tác copy không cần thiết
  - Tránh chi phí để phân mảnh/ tổ chức lại bộ nhớ
  - de zones (plusieurs tailles possibles)
- Quản lý các hoạt động
  - Thread pool
  - Tránh chi phí tạo mới luồng công việc
- Quản lý truyền thông
  - Pool của các kết nối
- Quản lý các thành phần trong hệ thống

4. Chặn (Interceptor)

- Bối cảnh
  - Cung cấp dịch vụ (chung)
  - Truyền thông theo các mô hình khác nhau: client-server, P2P, phân cấp
  - Tra đổi thông tin 1 chiều hoặc 2 chiều
- Vấn đề
  - Biến đổi dịch vụ (thêm chức năng) bằng các phương tiện khác nhau
  - Chèn thêm 1 tầng xử lý
  - Thay đổi điều kiện gọi/ đích cuộc gọi
- Ràng buộc
  - Không được thay đổi trình client và server
  - Thêm được các chức năng
  - Chèn thêm các đối tượng (tĩnh hoặc động) để chặn các lời gọi/ trả lời và thêm vào các thao tác xử lý cần thiết tùy theo quy trình nghiệp vụ
    - Chuyển hướng lời gọi đến server khác
    - Sử dụng các kết quả trả về

Sử dụng interceptor

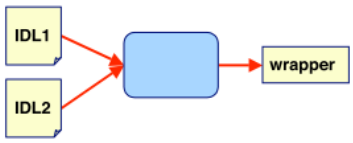


So sánh các mẫu thiết kế

- Adapter vs. Proxy
  - Cấu trúc tương tự nhau
  - Proxy giữ nguyên giao diện, adapter biến đổi giao diện
  - Truy nhập vào Proxy thường là truy nhập từ xa, truy nhập vào adapter thường là truy nhập tại chỗ
- Adapter vs. Interceptor
  - Chức năng tương tự nhau
  - Adapter thay đổi giao diện, Interceptor thay đổi chức năng (việc thay đổi này đôi khi có thể làm thay đổi hoàn toàn lời gọi đối tượng đích ban đầu)
- Proxy vs. Interceptor
  - Proxy là dạng biểu diễn đơn giản của Interceptor
  - Có thể thêm 1 Interceptor vào 1 Proxy → smart proxy

Cài đặt các mẫu thiết kế

- Sinh tự động từ khai báo



- Tối ưu
  - Loại bỏ các quan hệ gián tiếp giữa các đối tượng – nguyên nhân làm giảm hiệu quả thực thi
  - Chèn mã nguồn (chèn mã nguồn liên quan đến mẫu thiết kế vào mã của ứng dụng)
  - Sinh mã mức thấp (ví dụ java bytecode)
  - Dịch ngược (để thích nghi)