Slide tuan 3

Đã qua 2 slide đầu.

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)

1. Quản lý kiến trúc phần mềm

- Encapsulation: phân tách giữa giao diện và phần thực hiện
- Abstraction: Phan chia thành nhiều mức, che dấu những chi tiết không thích đáng ở mỗi mức độ xét đến.
- Phân tách giữa chính sách và kỹ thuật:
  - Ko cài đặt lại các kỹ thuật khi thay đổi chính sách.
  - Không thậm xưng hóa các 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 (haowcj gần như độc lập) và xử lý 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 1 yêu cầu về dữ liệu cho đến khi dl được ycau dduwwcj 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 lluoon ở một trạng thái nhất định thì hệ thống mới thực hiện được.
  - Tiến trình bên máy A gửi dữ liệu đến tiến trình máy B ddeer thực hiện.
  - A nhận kết quả do B trả về, xử lý và trả lại kết quả cho B.
  - Việc A giả sử B luôn sẵn sàng nhận kết quả này là ko 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 luwcj tạo bộ đệm và sao lưu phải nhằm mục đích giảm thiểu số lượng các “stateful component” 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.
    - Dl tring bộ ddeemj có thể bị cũ, nên cần có chính sách kiểm tra trước khi sử dụng.
    - Nếu tiến trình P lưu thông tin I ko 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 ko thể tái cấu trúc được.
    - A, B, C là 3 bản sao của tiến trình P, Neeys ttrinh P’ gọi A và thay đổi 1 số dlieuj, sau đó gọi B, vậy:
    - - Sự thau đổi (thiếu 3 ý)

5. Đảm bảo chất lượng

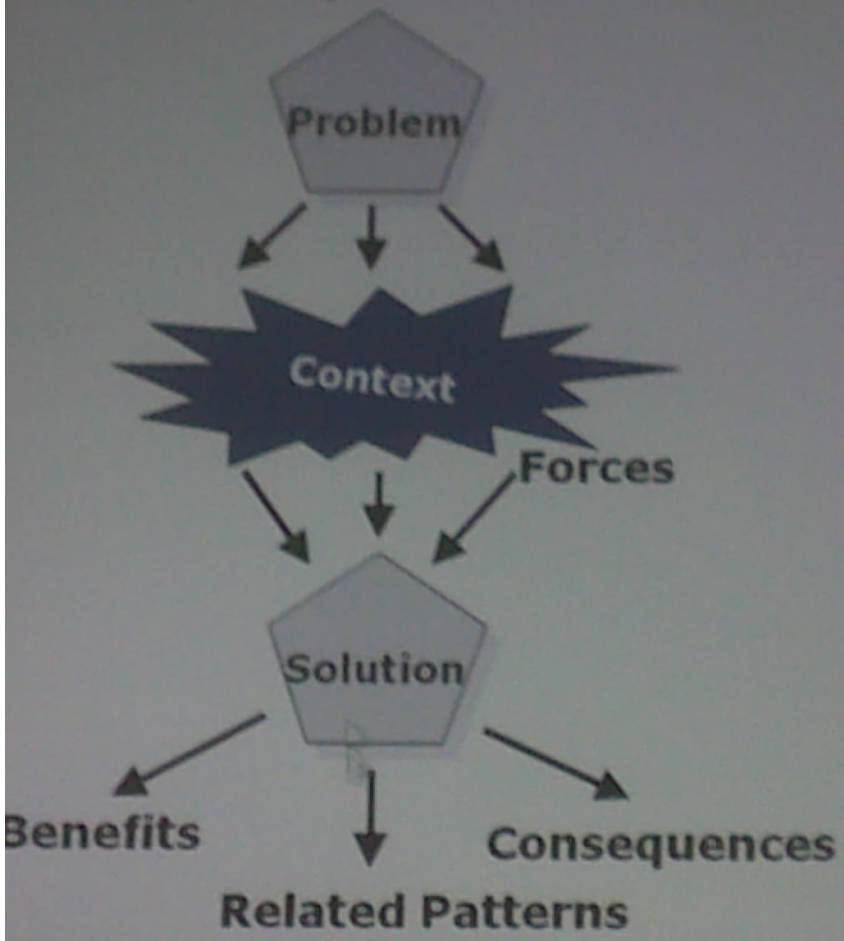
- Tốc độ và hiệu năng: xác định phần nào của hệ thống cần cải thiện hiệu năng.
  - Có nút cổ chai hay ko? Tại sao?
  - Lựa chọn giải pháp thích hợp nhất bằng cách thử các trường hợp khác nhau.
- Tránh sử dụng giao thức đòi hỏi xác nhận là đã nhận được dữ liệu (ack) nếu có thể.
- Việc truyền lại dữ liệu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

III.

- Mở đầu
- Mẫu thiết kế (design pattern): miêu tả vấn đề và bản chất của những cách giải quyết vấn đề tđó trong bối cảnh.
- Dựa trên các đăck tính của đối tượng như kế thừa hay đa hình.
- Đủ trừu tượng để taisusse nhằm đáp ứng 1 lớp ycau đặt ra trong các bối cảnh khác nhau.

-> cách thức để tái sử dụng các tri thức về vấn đề và cách giải quyết nó….

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

- Consequences
  - Kết quả hay lợi ích của việc áp dụng mẫu

1. Bao gói (wrapper)

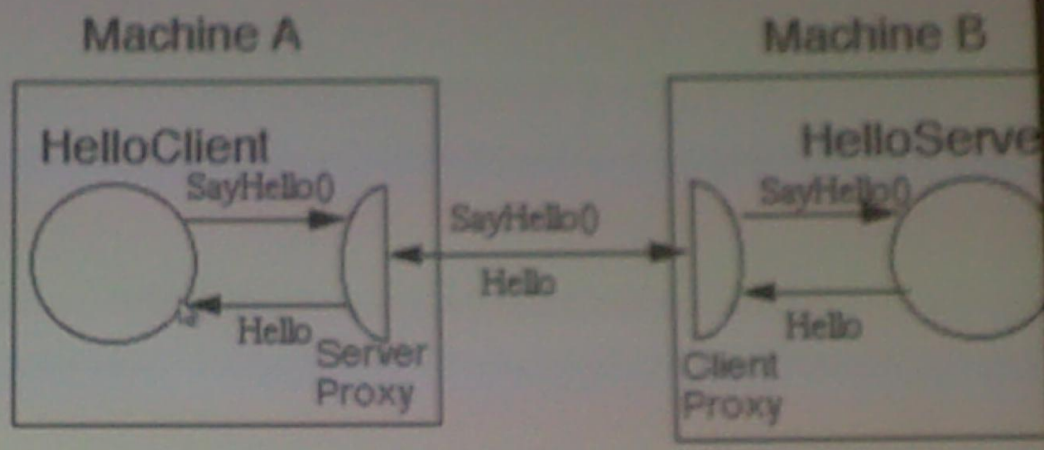
- Bchuyển đổi các gia diện không tương thích.
- Dễ dàng
- Lớp bao gói phụ trahc sphaanf lớn các công việc
- Adapter, Decorator, Proxy

a. Đại diện (Proxy)

- Bối cảnh:
  - Ud bao gồm các đối tượng phân tán, 1 đối tượng (client) có thể truy cập đến cách chức năng cung cấp bởi các đối tượng từ xa.

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(); }
    }
}
```

- Vật thay thế.
- Có quyên hoạt động nhân danh các đối tượng khác.
- Điều khiển truy nhâp đến các đối tương khác.
  - Truyền thông….

ản

a. Đại diện (Proxy)

CHƯƠNG II.
NGUYÊN TẮC
THIẾT KẾ

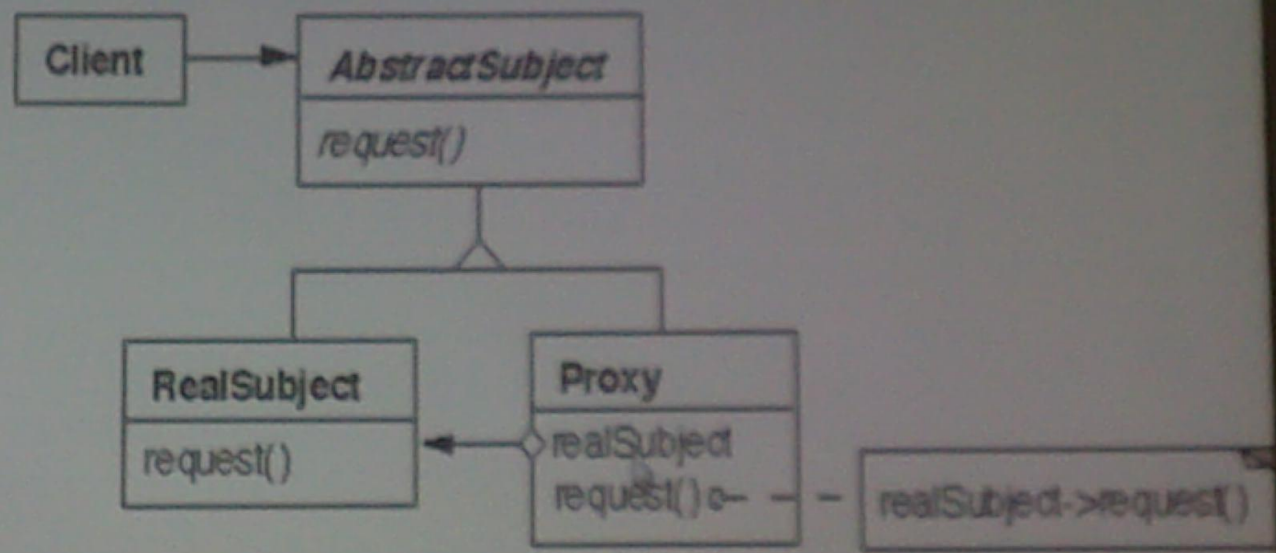
I. Mô hình
ứng dụng phân
tán

II. Nguyên
lý thiết kế
chung

III. Một số
mẫu thiết kế
điển biểu

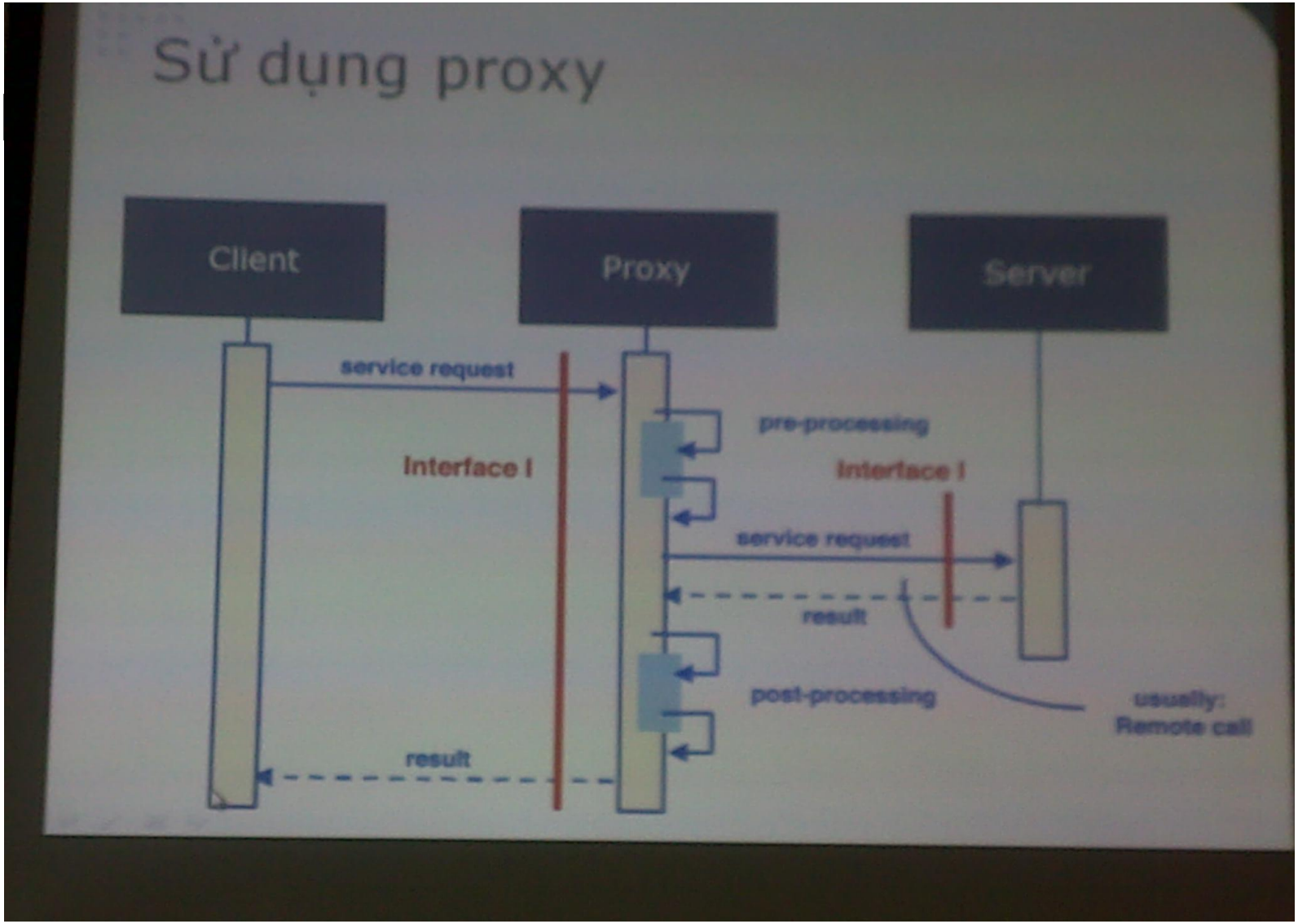
1. Bao gói

- 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 đối tượng gốc, do đó có khả năng chuyển truy vấn đến đối tượng gốc



Sử dụng proxy

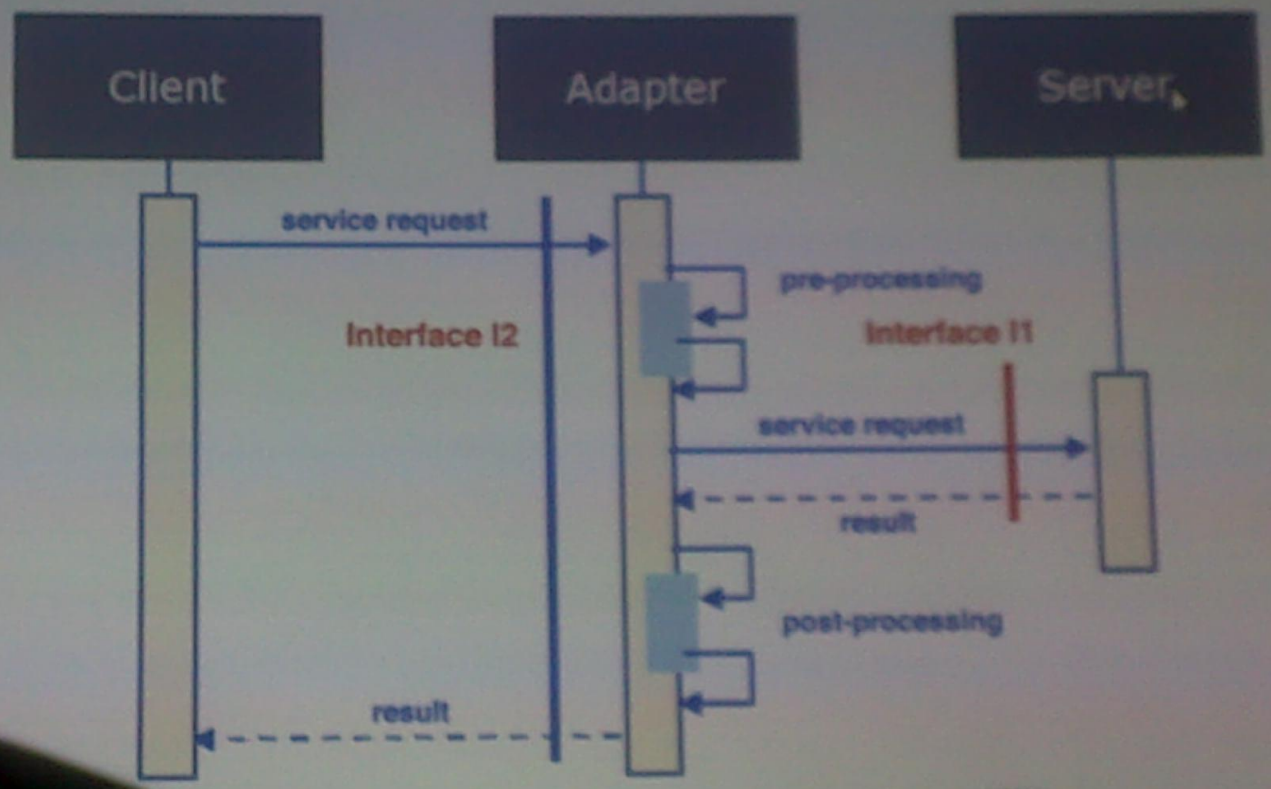
ản



6. Chuyển đỏi (adapter)

- Bối cảnh:
  - Client ycau các server cung cấp các chức năng thông qua giao diện
- Vấn đề:
- Các đặt tính mong muốn.
- Giải pháp:
- Chỉ được thay đổi giao diện➔ko được thay đổi chức năng.

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);   ...
}
}
```

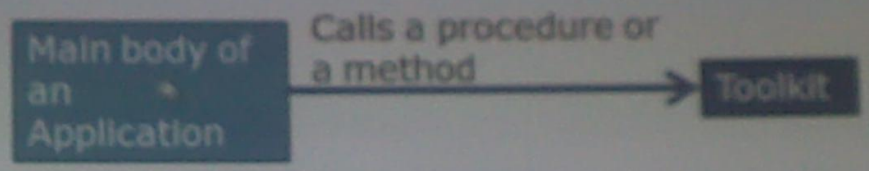
- Có thể sử dụng lớp này để thay thế hay không?

```
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 phần tán,

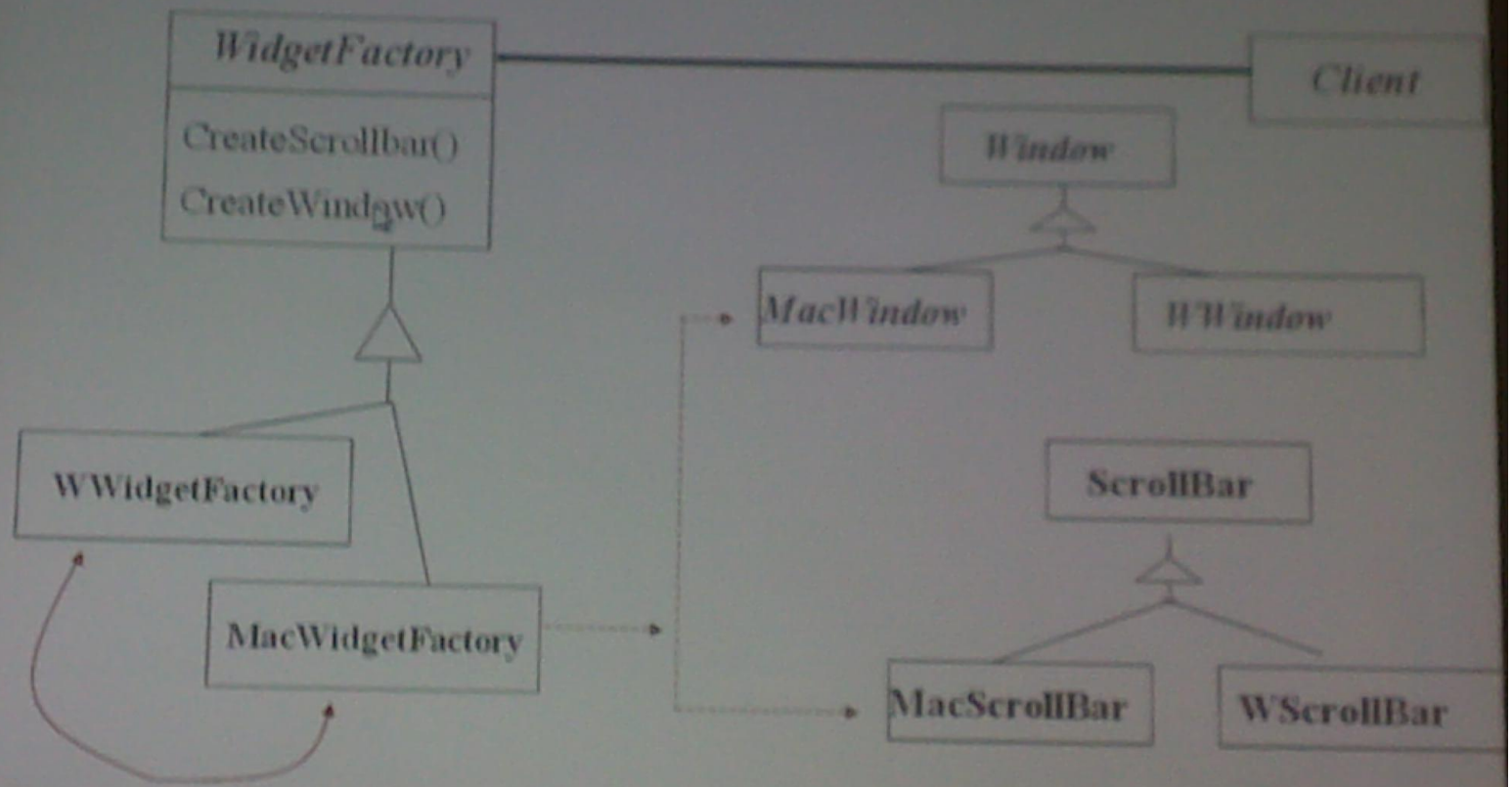
Ví dụ về factory



- 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

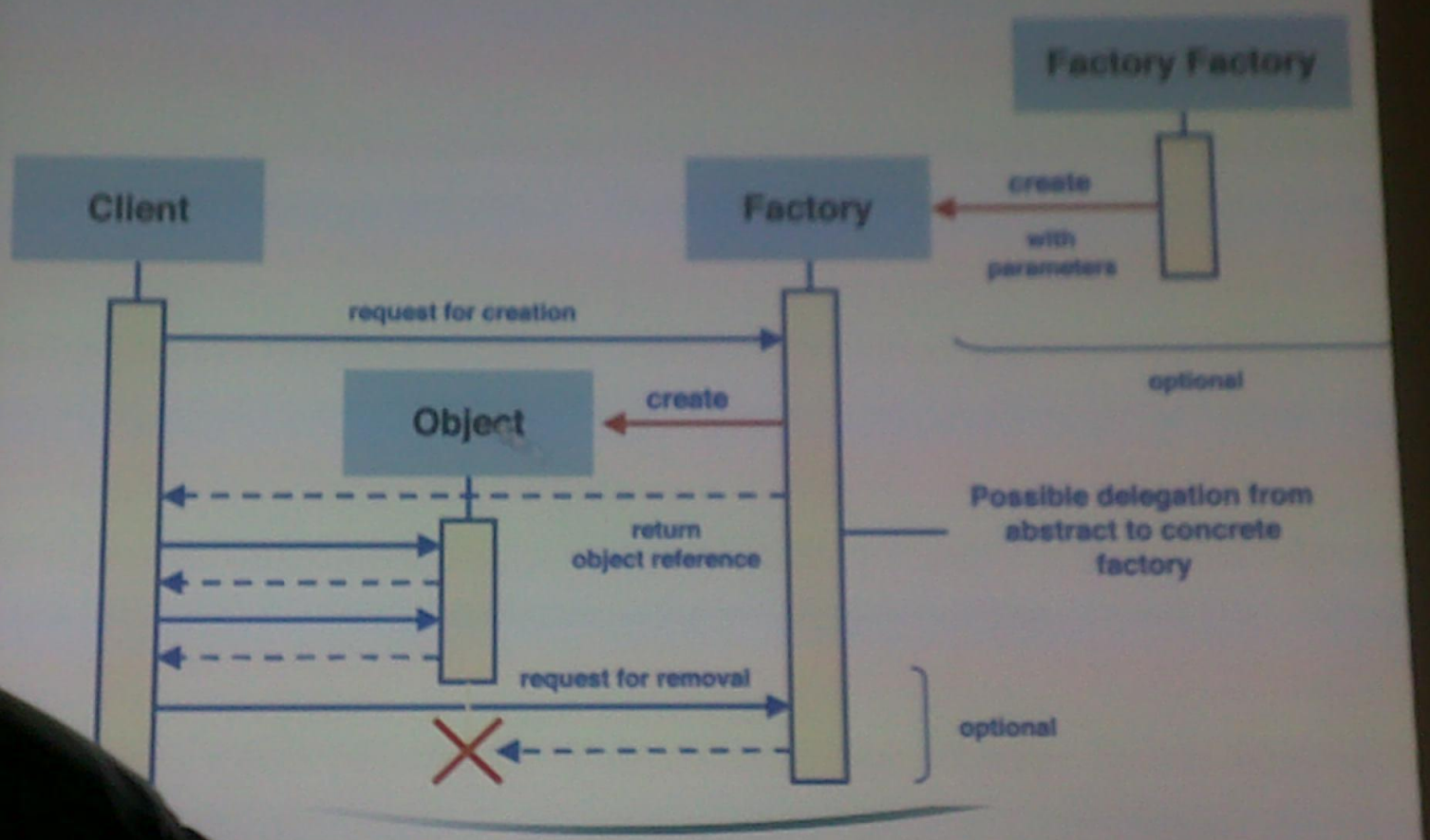


One for each standard.

2. Đại lý

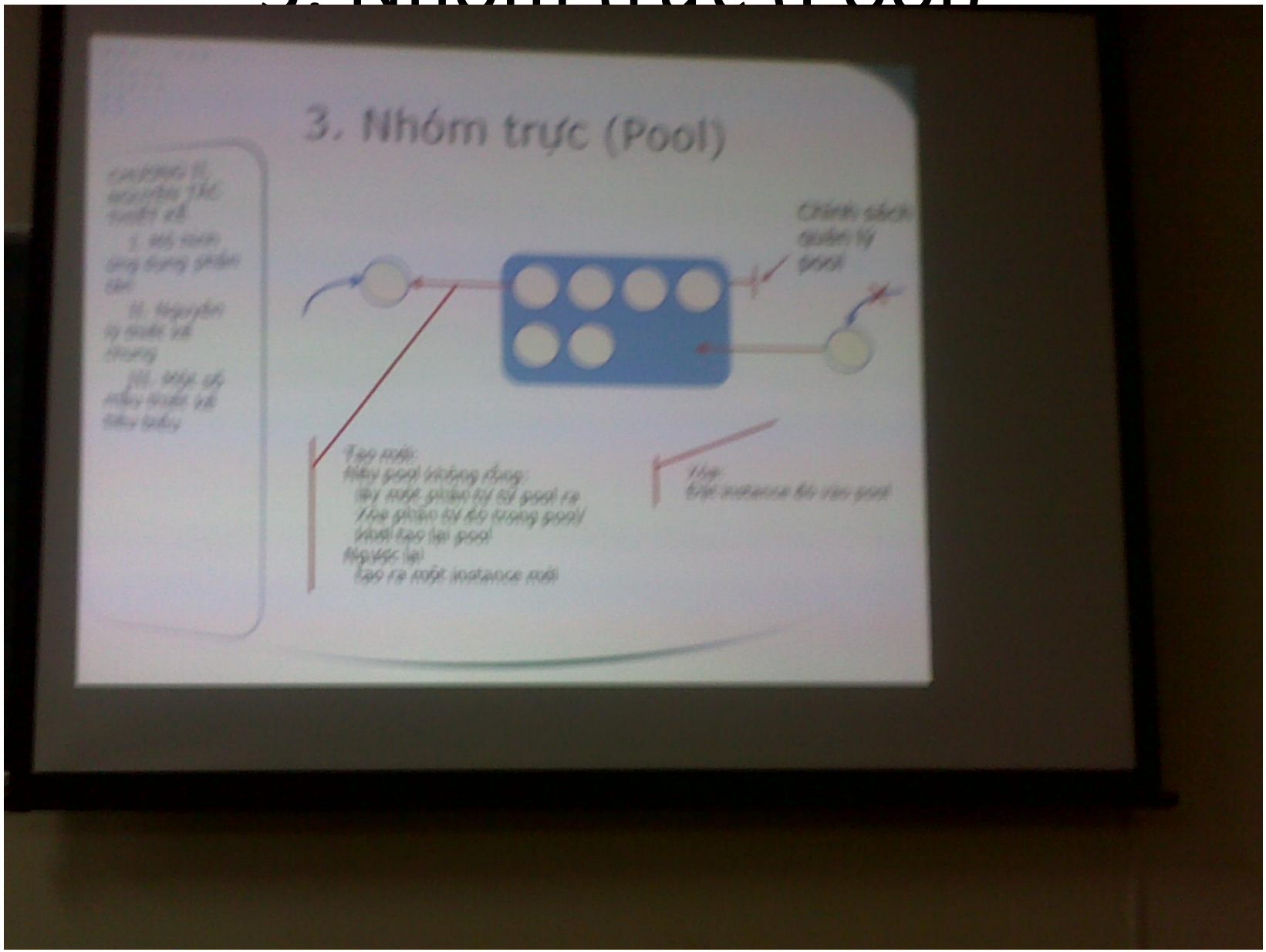
ảnh 10 (sơ đồ cụ thể-ko chụp được)

Sử dụng factory



3. Nhóm trực (Pool)

ả

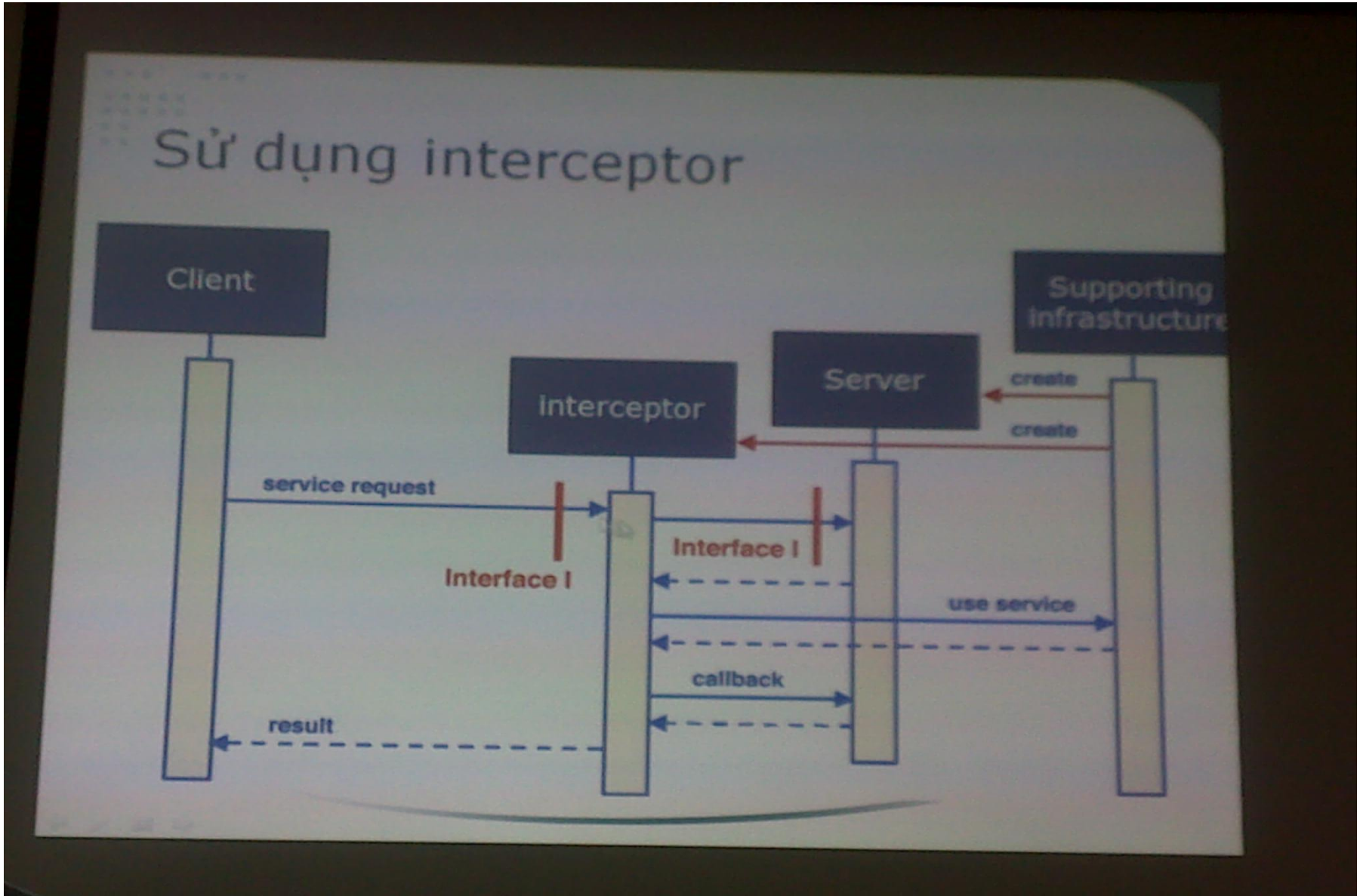


- Quản lý bộ nhớ.
  - Tránh các thao tác copy ko cần thiết.
  - Tránh chi phí để phân mảnh/ tổ chức lại bộ nhóe.
  - De zones
- Quản lý các hoạt động.
  - Thread Pool,
- Quản lý truyền thông.
- Quản lý các thành phần hệ thống.

4. Chặn (Interceptor)

- Bối cảnh: (thêm chức năng)
  - Cung cấp dịch vụ,
  - Truyền thông các
- Vấn đề:
- Ràng buộc:
  - Ko được thay đổi trình client và server

Sử dụng interceptor



So sánh các mẫu thiết kế

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