ÁP DỤNG MẪU THIẾT KẾ HƯỚNG ĐỐI TƯỢNG TRONG PHÁT TRIỂN PHẦN MỀM HƯỚNG DỊCH VỤ

APPLICATION OF OBJECT-ORIENTED DESIGN PATTERNS TO DEVELOP SERVICE-BASED SOFTWARE

Trần Đan Thư,
ĐH Khoa Học Tự Nhiên TP. HCM.
Lê Văn Vinh,
ĐH Sư Phạm Kỹ Thuật TP. HCM.

TÓM TẮT

Trong những năm gần đây, các ứng dụng hướng dịch vụ ngày càng trở nên phổ biến và trở thành xu hướng nổi bật trong ngành công nghiệp phần mềm. Cũng như khi xây dựng các phần mềm truyền thống, nhà phát triển phải đối mặt với những vấn đề khó khăn thường lặp lại khi thiết kế và cài đặt phần mềm hướng dịch vụ. Vì vậy, sử dụng các mẫu thiết kế luôn là một nhu cầu cần thiết trong xây dựng phần mềm. Trong bài báo này, chúng tôi trình bày kết quả nghiên cứu việc vận dụng các mẫu thiết kế hướng đối tượng trong phát triển phần mềm hướng dịch vụ, đồng thời đề xuất cải tiến cho mẫu thiết kế Observer để áp dụng một cách hiệu quả hơn.

***Từ khóa**: Mẫu thiết kế, dịch vụ web, phần mềm hướng dịch vụ, mẫu Observer.*

ABSTRACT

In recent years, service-based applications are more popular and become prominent trend in the software industry. As well as developing traditional software, developers face with the difficult problems often repeated when designing and coding service-based software. So, using design patterns is necessary for developing software. In this paper, we present our research results in applying object-oriented design patterns to develop service-based software, and propose general improvements for Observer design pattern.

***Keywords**: Design pattern, web services, service-based software, Observer pattern.*

I. GIỚI THIỆU

Mẫu thiết kế là một giải pháp để giải quyết một vấn đề khó khăn nào đó thường lặp lại trong các dự án phần mềm. Khái niệm mẫu thiết kế được đưa ra đầu tiên bởi 0 và được áp dụng cho lĩnh vực xây dựng. Trong lĩnh vực phát triển phần mềm hướng đối tượng, rất nhiều mẫu thiết kế hướng đối tượng đã được đề xuất 000. Đặt biệt, các mẫu GoF0 có tầm quan trọng và ảnh hưởng lớn đối với giới nghiên cứu cũng như giới công nghiệp phần mềm. Phần lớn các mẫu thiết kế hướng đối tượng được công bố mới là sự biến thể hoặc kết hợp từ các mẫu GoF. Lợi ích mang lại của các mẫu thiết kế là giúp phần mềm dễ bảo trì, dễ mở rộng và có tính tái sử dụng cao. Hơn nữa, sử dụng các mẫu thiết kế giúp cho việc trao đổi và truyền đạt ý tưởng giữa các thành viên trong cùng một dự án phần mềm trở nên dễ dàng hơn.

Thời gian gần đây, nhu cầu xây dựng các ứng dụng dựa trên nền tảng dịch vụ web ngày càng nhiều. Kiến trúc của dạng phần mềm này mang một số đặc trưng khác biệt so với kiến trúc phần mềm hướng đối tượng truyền thống. Nhà thiết kế phần mềm phải quan tâm đến vấn đề tương tác giữa các phần mềm, cách thức gọi và sử dụng chức năng được cung cấp bởi các dịch vụ khác, vấn đề bảo mật, vấn đề tương thích mã nguồn, v.v… Điều này dẫn đến một số giải pháp truyền thống không đáp ứng được nhu cầu tạo ra bản thiết kế tốt cho phần mềm. Vì vậy, việc phát triển ứng dụng hướng dịch vụ cần phải áp dụng các mẫu thiết kế mới một cách phù hợp.

Một số mẫu thiết kế với tên gọi "mẫu thiết kế cho kiến trúc hướng dịch vụ - SOA design pattern" 000 đã được đề xuất. Những mẫu thiết kế này cung cấp giải pháp cho một vấn đề thiết kế cụ thể, thiết lập cấu trúc ở mức độ thấp (low level) cho các dịch vụ dựa trên các công nghệ đã có; hay đề xuất kỹ thuật thiết kế cho một nhóm các phần mềm doanh nghiệp, nhấn mạnh ở khía cạnh trình tự thực thi của của các dịch vụ theo quy trình nghiệp vụ.

Nhóm mẫu thiết kế khác cung cấp giải pháp giúp nhà phát triển thiết kế các lớp đối tượng cho phần mềm hướng dịch vụ, từ đó có thể cài đặt bằng các ngôn ngữ lập trình hướng đối tượng 0000. Các mẫu thiết kế này phần lớn có nguồn gốc từ các mẫu GoF. Khi đề xuất những mẫu thiết kế này, các tác giả hoặc là diễn đạt lại cho phù hợp với ngữ cảnh của phần mềm hướng dịch vụ, hoặc là xây dựng mẫu thiết kế phức hợp (composit design pattern 0) dựa trên một số mẫu GoF. Đây cũng chính là hướng nghiên cứu mà chúng tôi quan tâm và đã có một số đóng góp trước đây 0.

Trong bài báo này, chúng tôi trình bày nghiên cứu áp dụng mẫu thiết kế hướng đối tượng nhằm hỗ trợ nhà phát triển trong việc thiết kế các lớp đối tượng cho phần mềm hướng dịch vụ, giúp giải quyết những vấn đề khó khăn thường gặp. Phần 2 của bài báo trình bày các công nghệ phát triển ứng dụng hướng dịch vụ và phân tích các vấn đề nảy sinh trong quá trình phát triển phần mềm. Trong phần 3, chúng tôi giới thiệu những mẫu thiết kế GoF thường được áp dụng cho ứng dụng hướng dịch vụ để giải quyết các vấn đề nảy sinh. Kế đến chúng tôi trình bày việc nghiên cứu cải tiến mẫu Observer 0 để áp dụng cho phát triển phần mềm hướng dịch vụ. Sau cùng, trong phần 5, chúng tôi tổng kết vấn đề đã nghiên cứu và xác định hướng phát triển cho chủ đề nghiên cứu này.

II. PHẦN MỀM HƯỚNG DỊCH VỤ VÀ NHỮNG VẤN ĐỀ NẢY SINH

A. Phần mềm hướng dịch vụ và dịch vụ web

Phần mềm hướng dịch vụ (service-based software hay service-oriented software 0) là dạng phần mềm được xây dựng dựa trên các dịch vụ có sẵn, hay bản thân nó cũng cung cấp các chức năng dưới dạng dịch vụ. Đây là một dạng của mô hình phát triển phần mềm phân tán mà thường được hiện thực bởi các công nghệ như RPC (Remote Procedure Call), DCOM, CORBA, RMI (Remote Method Invocation) 0 . Trước đây, dạng phần mềm này không được áp dụng nhiều bởi những hạn chế của các công nghệ. Dịch vụ web 0 ra đời mang lại sự thay đổi lớn bởi những ưu điểm nổi bật so với các công nghệ trước đó. Sự phát triển mạnh mẽ và ứng dụng của kiến trúc hướng dịch vụ (SOA – Service-Oriented Architecture 0) và gần đây là mô hình điện toán đám mây (Cloud Computing) cho thấy ngày càng nhiều phần mềm dựa trên nền tảng dịch vụ web được phát triển và có xu hướng thay thế dần các phần mềm truyền thống.

Về mặt công nghệ, dịch vụ web có thể được xây dựng theo những cách khác nhau, nhưng đều dựa trên nền tảng các giao thức hay công nghệ cơ bản như HTTP, XML. Cụ thể, dịch vụ web dạng SOAP (SOAP web service 00) là công nghệ sử dụng các chuẩn mở như XML, SOAP, WSDL, và UDDI . Trong đó, XML để mô tả dữ liệu, SOAP - được hiện thực trên nền tảng giao thức HTTP - dùng để truyền tải dữ liệu, WSDL giúp mô tả thông tin cho dịch vụ web và UDDI hỗ trợ người dùng tìm kiếm dịch vụ web. Một kiểu dịch vụ web khác là dịch vụ web dạng RESTful (RESTful web service 0). Dạng dịch vụ này không sử dụng tài liệu mô tả thông tin WSDL. Thông tin cần thiết để truy xuất dịch vụ là dựa trên định danh tài nguyên URI (Uniform Resource Identifier).

Nhìn ở khía cạnh cài đặt ứng dụng, hầu hết các môi trường hay ngôn ngữ lập trình hiện đại (Java, .NET, PHP, …) đều hỗ trợ việc xây

dựng các phần mềm dựa trên dịch vụ web. Quá trình thiết kế và cài đặt vẫn sử dụng phương pháp hướng đối tượng. Những đặc điểm khác biệt với phần mềm hướng đối tượng truyền thống, cùng với những đặc tính của công nghệ dịch vụ web có thể làm nảy sinh những vấn đề khó khăn trong quá trình phát triển phần mềm. Chủ đề này nhận được nhiều sự quan tâm của giới công nghiệp phần mềm cũng như giới nghiên cứu. Một trong những cách giải quyết là sử dụng các mẫu thiết kế hay kiến trúc lập trình phù hợp.

B. Các vấn đề nảy sinh trong quá trình thiết kế và hiện thực ứng dụng hướng dịch vụ

Trong phần này, chúng tôi hệ thống lại các vấn đề nảy sinh trong quá trình thiết kế và cài đặt ứng dụng hướng dịch vụ dựa trên những nghiên cứu, quan sát và phân tích theo quan điểm phát triển công nghệ phần mềm một cách bền vững 0. Những vấn đề này, nếu không được giải quyết tốt sẽ khiến ứng dụng trở nên cồng kềnh, khó bảo trì, khó mở rộng.

- *Trùng lặp mã nguồn*: Kết quả của việc sao chép vật lý những đoạn mã mang tính chất thủ tục (thao tác đăng nhập, thao tác kết nối hay nhận dữ liệu trả về từ dịch vụ web). Các hệ thống phần mềm trùng lặp mã nguồn rất khó bảo trì và chỉnh sửa.
- *Mã hóa cứng (hard coding)* tên các đối tượng và phương thức được cung cấp bởi các dịch vụ vào mã nguồn xử lý của ứng dụng. Mã hóa cứng quá nhiều sẽ khiến các đoạn mã chỉ dùng được một lần, không thể tái sử dụng hay mở rộng để dùng trong các tình huống tương tự nhưng có thay đổi chút ít. Vì vậy, mã hóa cứng càng ít thì ứng dụng càng dễ mở rộng hay nâng cấp.
- *Nhúng trực tiếp các đoạn mã xử lý kết nối, triệu gọi dịch vụ* vào mã nguồn xử lý nghiệp vụ. Khi có một vài thay đổi những đối tượng, phương thức cung cấp bởi dịch vụ, hay khi cần thêm hoặc hủy bỏ kết nối với một dịch vụ nào đó, lập trình viên có thể phải chỉnh sửa một số lượng lớn những đoạn mã nguồn của ứng dụng.
- *Sự không tương thích giữa các công nghệ dịch vụ web*. Vấn đề này buộc người phát triển ứng dụng phải đưa ra cách giải quyết thích hợp cho từng tình huống cụ thể.
- *Sự khác biệt về giao tiếp (interface) của các dịch vụ*. Điều này gây khó khăn khi lập trình viên có nhu cầu tích hợp nhiều dịch vụ hỗ trợ cùng một chức năng hay cùng phục vụ cho một quy trình nghiệp vụ.

Nếu không có cách thiết kế phù hợp, ứng dụng có thể bị trùng lắp mã nguồn và trở nên cồng kềnh, khó chỉnh sửa.

Để giải quyết các vấn đề trên, việc áp dụng các mẫu thiết kế cho từng tình huống cụ thể là một giải pháp mang lại hiệu quả cao. Phần tiếp theo của bài báo trình bày kết quả phân tích một số mẫu thiết kế tiêu biểu được áp dụng trong xây dựng ứng dụng hướng dịch vụ.

III. VẬN DỤNG MẪU THIẾT KẾ TRONG TIẾN TRÌNH XÂY DỰNG ỨNG DỤNG HƯỚNG DỊCH VỤ

Qua quá trình nghiên cứu, chúng tôi nhận thấy rằng một số mẫu thiết kế GoF cũng có thể được vận dụng hiệu quả trong phát triển phần mềm hướng dịch vụ. Điều này cũng đã được trình bày chi tiết trong các bài báo khác nhau 000. Các mẫu này được giới nghiên cứu diễn đạt lại cho phù hợp với ngữ cảnh xây dựng ứng dụng hướng dịch vụ. Ngoài ra, những mẫu thiết kế mới được đề xuất hầu hết từ việc cải tiến các mẫu GoF hay kết hợp các mẫu GoF để tạo ra mẫu mới 00.

- Mẫu *Proxy* là mẫu được áp dụng mặc định trong hiện thực dịch vụ web 00. Bản chất của việc cung cấp các dịch vụ là cung cấp các hàm thực thi và lớp đối tượng cần thiết để ứng dụng client có thể gọi và thi hành. Tại server, các lớp đối tượng này được chuyển đổi sang dạng dữ liệu trung gian (như WSDL, XML) và được truyền tải qua môi trường mạng. Tại client, dữ liệu được chuyển đổi ngược lại sang lớp đối tượng

theo ngôn ngữ lập trình đang được sử dụng. Các lớp đối tượng tại client này được gọi là các lớp proxy. Nói một cách khác, lớp đối tượng proxy tại client là đại diện cho lớp đối tượng được cung cấp bởi server.

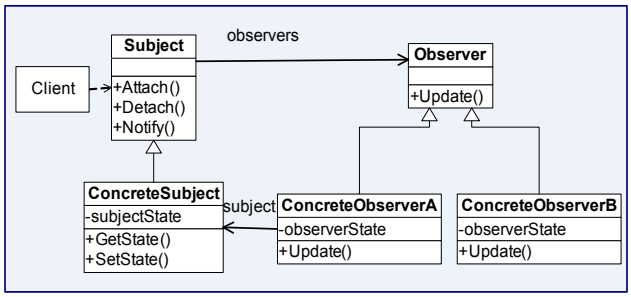
- Các phương thức (web methods) được cung cấp bởi dịch vụ web có thể có giao tiếp (interface) không phù hợp với giao tiếp mà client yêu cầu. Mẫu *Adapter* thường được áp dụng để giải quyết vấn đề này 0. Mẫu này giúp tạo ra sự tương thích giữa mã nguồn đã có của client và các đoạn mã được cung cấp bởi server.
- Mẫu *Façade* được vận dụng cho trường hợp xây dựng dịch vụ web cho hệ thống phần mềm sẵn có 00. Dịch vụ web được xem như một giao tiếp (interface) chung cho tất cả các đối tượng và chức năng bên trong hệ thống.

Bên cạnh các mẫu có thể áp dụng trực tiếp, nhiều mẫu GoF khác đóng vai trò là một thành phần hình thành nên các mẫu thiết kế phức hợp (composite design pattern 0) được đề xuất trong những năm gần đây. Có thể kể đến một số mẫu tiêu biểu như Service Factory, Product Manager 0, Composite Pattern for Web service Invocation 0. Chúng tôi cũng đã đề xuất một số mẫu thiết kế trong các nghiên cứu trước đây 0.

IV. MẪU THIẾT KẾ OBSERVER CẢI TIẾN

Từ quá trình cài đặt thử nghiệm, chúng tôi nhận thấy một số mẫu thiết kế GoF không giải quyết vấn đề một cách triệt để khi áp dụng trong ngữ cảnh ứng dụng hướng dịch vụ. Giải pháp chúng tôi áp dụng là xây dựng mẫu thiết kế phức hợp dựa trên các mẫu đã có để giúp nhà phát triển có một bản thiết kế hoàn thiện. Một trong những kết quả chúng tôi đã nghiên cứu là việc cải tiến mẫu Observer 0 nhờ vận dụng kết hợp với mẫu Factory Method 0.

C. Mẫu thiết kế Observer

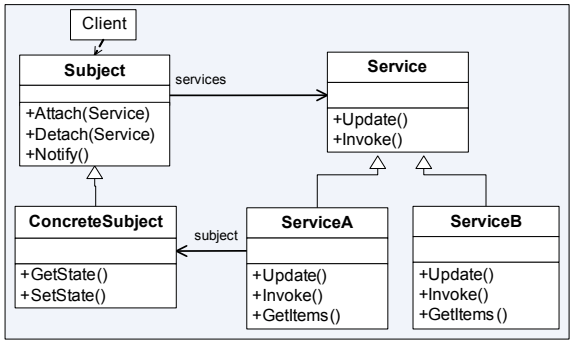


Hình 1. Mẫu thiết kế Observer
trong phát triển ứng dụng hướng đối tượng.

Mẫu Observer (Hình 1) định nghĩa mối quan hệ một – nhiều giữa các đối tượng. Trong đó một đối tượng đóng vai trò chính và ảnh hưởng đến các đối tượng còn lại (các đối tượng đóng vai trò quan sát). Khi đối tượng chính thay đổi trạng thái, các đối tượng quan sát cũng thay đổi theo. Mẫu này cho phép xử lý, thêm, bớt các lớp đối tượng quan sát (ConcreteObserverA, ConcreteObserverB) mà không cần quan tâm đích danh là đối tượng nào.

Trong ngữ cảnh ứng dụng hướng dịch vụ, mẫu Observer có thể được dùng ở tình huống client cần triệu gọi cùng lúc đến nhiều dịch vụ web khác nhau mà cung cấp chức năng giống nhau. Chẳng hạn, client có chức năng tìm kiếm thông tin sản phẩm của các công ty khác nhau, tìm kiếm thông tin dự báo thời tiết từ nhiều trung tâm, xem giá cổ phiếu, v.v... Client cung cấp giá trị của tham số đầu vào (còn gọi là tiêu chí đầu vào), các dịch vụ web có vai trò xử lý và trả về kết quả tương ứng. Hình 2 mô tả mẫu Observer được diễn đạt lại theo tình huống vận dụng cho ứng dụng hướng dịch vụ.

ServiceA và *ServiceB* là lớp đối tượng đại diện của hai dịch vụ được gọi. Hai dịch vụ này cung cấp chức năng giống nhau (*Invoke()*). Khi tiêu chí đầu vào thay đổi, các dịch vụ xử lý lại và trả về kết quả mới.

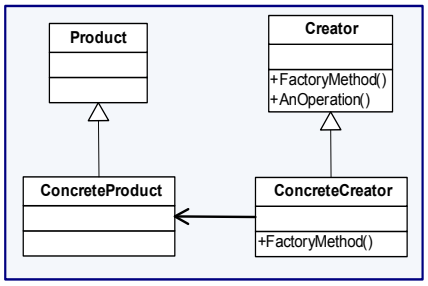


Hình 2. Mẫu thiết kế Observer vận dụng trong phát triển ứng dụng hướng dịch vụ

Về mặt ý nghĩa, mẫu thiết kế này đã giải quyết tốt việc tách biệt mã nguồn xử lý nghiệp vụ và mã nguồn xử lý thực thi các dịch vụ. Nó cũng cho phép ứng dụng gửi yêu cầu thực thi cùng lúc đến nhiều dịch vụ mà không cần quan tâm đích danh đó là dịch vụ nào. Có thể thêm, bớt dịch vụ tại thời điểm thực thi ứng dụng. Tuy nhiên, vấn đề khác biệt về giao tiếp (interface) giữa các dịch vụ chưa được giải quyết một cách triệt vì kiểu dữ liệu trả về bởi các dịch vụ có thể khác nhau. Lập trình viên khi ứng dụng mẫu này vẫn phải xử lý riêng biệt với từng dịch vụ cụ thể (*ServiceA*, *ServiceB*) để xác định kiểu dữ liệu trả về tương ứng. Để giải quyết vấn đề này, chúng tôi cải tiến mẫu này bằng cách vận dụng kết hợp với mẫu Factory Method.

D. Mẫu thiết kế Factory Method

Mẫu Factory Method (hình 3) cho phép tạo một đối tượng mà không cần thiết phải chỉ ra đích danh đối tượng nào được tạo. Nhiệm vụ tạo đối tượng được giao phó cho các lớp đối tượng kế thừa. Vận dụng mẫu này vào ứng dụng hướng dịch vụ có thể giúp tạo ra một giao tiếp chung cho các kiểu dữ liệu trả về client từ các dịch vụ.

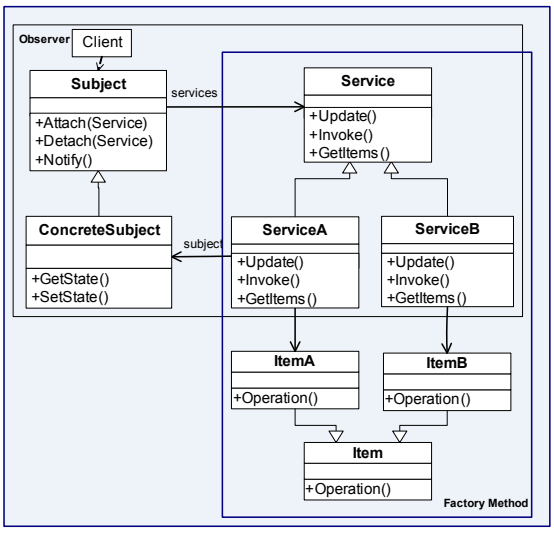


Hình 3. Mẫu thiết kế Factory Method trong phát triển ứng dụng hướng đối tượng

E. Mẫu thiết kế Observer cải tiến cho ứng dụng hướng dịch vụ

1) Cấu trúc

Hình 4 minh họa mẫu Observer cải tiến nhờ vận dụng kết hợp với mẫu Factory Method. Lớp *Subject* là một lớp ảo được hiện thực bởi lớp *ConcreteSubject*. Phương thức *Notify()* có chức năng cập nhật tiêu chí đầu vào cho các dịch vụ. Phương thức *Attach(Sevice)*, *Detach(Service)* cho phép thêm hoặc hủy dịch vụ đang gọi tại thời điểm thực thi. Trong lớp *ConcreteSubject*, phương thức *GetState()* và *SetState()* có chức năng thiết lập hoặc kiểm tra sự thay đổi tiêu chí đầu vào của các dịch vụ.



Hình 4. Mẫu Observer cải tiến, áp dụng cho ứng dụng hướng dịch vụ

Lớp *Service* là lớp ảo có vai trò tạo một giao tiếp chung cho các dịch vụ khác nhau.

Các lớp kế thừa của nó (*ServiceA*, *ServiceB*) là đại diện cho các dịch vụ được gọi. Trong hai lớp này, hàm chức năng *Invoke()* của mỗi lớp có cùng chức năng nhưng được cài đặt theo cách riêng của từng dịch vụ. Phương thức *Update()* có vai trò cập nhật tiêu chí đầu vào cho từng dịch vụ cụ thể (được gọi bởi phương thức *Notify()* trong lớp *Subject*).

Mỗi dịch vụ trả về kiểu dữ liệu khác nhau. Cụ thể, kiểu dữ liệu sử dụng bởi *ServiceA* là *ItemA*, *ServiceB* là *ItemB*, v.v… Tuy nhiên, sự khác biệt này được giải quyết nhờ sử dụng phương thức *GetItems()* trong các lớp *ServiceA*, *ServiceB*. Phương thức này (là một factory method) cho phép mỗi dịch vụ tạo và trả về kết quả theo cấu trúc riêng. Ở đây, lớp *Item* là một lớp ảo và được hiện thực bởi các lớp *ItemA*, *ItemB* tùy theo kiểu dữ liệu sử dụng của từng dịch vụ.

2) Cài đặt

Mẫu thiết kế có thể được cài đặt bằng các ngôn ngữ lập trình hướng đối tượng như C++, Java hay C#.NET. Ở đây, chúng tôi sử dụng ngôn ngữ C#.NET để minh họa cho việc cài đặt mẫu.

Trước hết, ứng dụng cần tạo một đối tượng của lớp *ConcreteSubject* và các đối tượng tương ứng với các dịch vụ cần gọi.

```
ConcreteSubject mConSub=new ConcreteSubject();
mConSub.Attach(new ServiceA());
mConSub.Attach(new ServiceB());
```

Để thực thi các dịch vụ, tiêu chí đầu vào cần được cung cấp. Việc này được thực hiện cùng lúc cho tất cả các dịch vụ bởi phương thức *Notify()*.

```
foreach Service svc in services
{
   svc->Update();
}
```

Đoạn mã nguồn sau thể hiện việc gọi thực thi và nhận kết quả trả về từ các dịch vụ bằng cách gọi phương thức *Invoke()* và *GetItems()*.

```
List<List<Item>> arrIts=new List<List<Item>>;
foreach Service svc in services
{
   svc.Invoke();
   List<Item> items=svc.GetItems();
   arrIts.add(items);
}
```

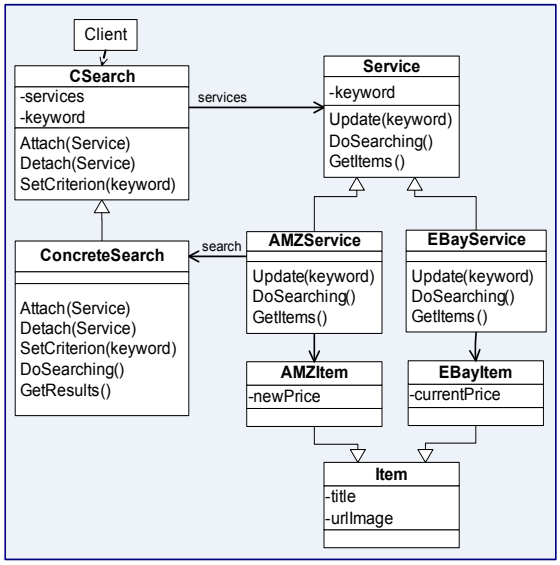
Factory method

Khi sử dụng mẫu thiết kế này, lập trình viên cần tạo thêm hai lớp nếu có thêm một dịch vụ khác được gọi và thực thi. Ví dụ, nếu thêm một dịch vụ thứ ba tên là ServiceC thì thêm hai lớp *ServiceC* (kế thừa lớp *Service*) và *ItemC* (kế thừa lớp *Item*).

3) Ý nghĩa khi vận dụng mẫu thiết kế Observer cải tiến

Trong phần này, chúng tôi trình bày những lợi ích mang lại khi vận dụng mẫu thiết kế Observer cải tiến thông qua cài đặt ứng dụng minh họa. Chúng tôi áp dụng để xây dựng trang web hỗ trợ tìm kiếm sản phẩm qua mạng.

Ứng dụng này cung cấp cho người dùng chức năng tìm kiếm thông tin sản phẩm từ các website thương mại điện tử như Amazon, eBay bằng cách sử dụng dịch vụ web cung cấp sẵn. Chức năng tìm kiếm đòi hỏi người dùng cung cấp tiêu chí tìm kiếm (theo từ khóa). Các dịch vụ thực thi và trả về danh sách sản phẩm theo yêu cầu. Sản phẩm của các nhà cung cấp có thuộc tính không hoàn toàn giống nhau về tên, kiểu dữ liệu, và số lượng. Hình 5 mô tả kiến trúc cài đặt chính của ứng dụng.



Hình 5. Mô hình cài đặt ứng dụng tìm kiếm sản phẩm, sử dụng mẫu Observer cải tiến

a) ***Tính dễ bảo trì của ứng dụng (nhờ hạn chế mã hóa cứng tên đối tượng phương thức của các dịch vụ)***

Dưới đây, chúng tôi minh họa và phân tích mã nguồn của ứng dụng trong hai trường hợp (sử dụng mẫu thiết kế và không sử dụng mẫu thiết kế) để từ đó rút ra những nhận xét, so sánh một cách cụ thể.

Ứng dụng được thực thi qua ba giai đoạn: Từ giai đoạn *khởi tạo dịch vụ* đến giai đoạn *gửi yêu cầu thực thi* và cuối cùng là *nhận kết quả trả về*. Trong trường hợp lập trình viên không sử dụng mẫu thiết kế, ứng dụng vẫn có thể được cài đặt theo một cách nào đó. Chúng tôi đưa ra trường hợp lập trình đơn giản và dễ hiểu nhất đứng trên quan điểm của lập trình viên như sau:

- Khởi tạo hai lớp đối tượng *AMZService*, *EBayService* tương ứng với hai dịch vụ, và hai lớp đối tượng *AMZItem*, *EbayItem* cho hai loại sản phẩm trả về từ Amazon, eBay.
- Cung cấp tiêu chí đầu vào và gọi thực thi lần lượt cho từng dịch vụ.
- Nhận kết quả trả về sao cho phù hợp kiểu dữ liệu của từng dịch vụ.

Sau đây là một số đoạn mã nguồn trong trường hợp này:

```
//Khởi tạo dịch vụ
AMZService azmSrv=new AMZService();
EBayService ebySrv=new EBayService();
//Gửi yêu cầu thực thi
amzSrv.SetCriterion(keyword);
amzSrv.DoSearching();
ebySrv.SetCriterion(keyword);
ebySrv.DoSearching();
//Nhận kết quả trả về
List<AMZItem> amzItm=new AMZItem();
amzItm=amzSrv.GetItems();
List<EBayItem> ebyItm=new EBayItem();
ebyItm=ebySrv.GetItems();
```

Nhận xét: Tất cả mã nguồn trong phần này phụ thuộc cứng vào tên đối tượng và phương thức của các dịch vụ.

Trong trường hợp áp dụng mẫu thiết kế, mã nguồn chương trình có thể được cài đặt lại như sau:

```
//Khởi tạo dịch vụ
ConcreteSearch cntSrc=new ConcreteSearch();
cntSrc.Attach(new AMZService());
cntSrc.Attach(new EBayService());

//Gửi yêu cầu thực thi
cntSrc.SetCriterion(keyword);
cntSrc.DoSearching();

//Nhận kết quả trả về
List<List<Item>>arrIts=cntSrc.
GetResults();
```

Đoạn mã này phụ thuộc cứng vào tên đối tượng của các dịch vụ

```
//Các phương thức sau được cài đặt trong lớp //ConcreteSearch

//Phương thức SetCriterion(keyword) cập nhật tiêu chí //đầu vào cho các dịch vụ
foreach(Service srv in services)
{
   srv.Update(keyword);
}

//Phương thức DoSearching() thực hiện chức năng tìm //kiếm
foreach(Service srv in services)
{
   srv.DoSearching();
}

//Phương thức GetResults()lấy kết quả trả về theo các //danh sách sản phẩm ứng với các dịch vụ
List<List<Item>> GetResults()
{
  List<List<Item>> arrIts=new List<List<Item>>();
  foreach(Service srv in services)
  {
     List<Item>items=srv.GetItems();
     arrIts.Add(items);
  }
  return arrIts;
}
```

Nhận xét: Chỉ duy nhất phần mã khởi tạo dịch vụ phụ thuộc cứng vào tên đối tượng của các dịch vụ.

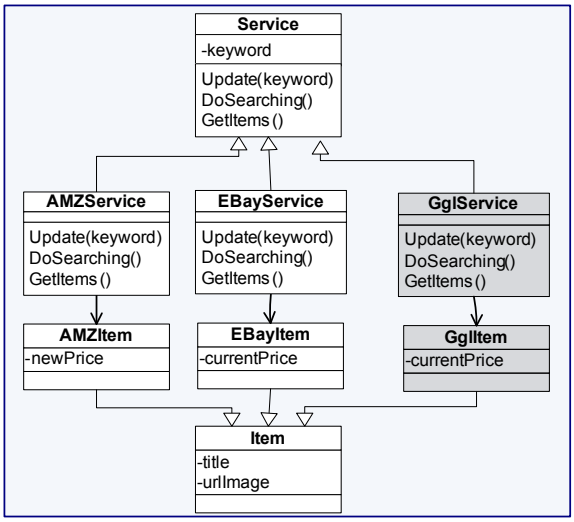
Rõ ràng, cũng với chức năng này, việc sử dụng mẫu tuy không giúp mã nguồn ngắn gọn hơn nhưng giúp ứng dụng dễ bảo trì khi có những thay đổi hoặc có lỗi phát sinh. Điều này có được nhờ hạn chế tối đa việc phụ thuộc cứng vào tên đối tượng và phương thức của các dịch vụ.

b) ***Tính tiến hóa của ứng dụng***

Khi cần cài đặt để sử dụng thêm một dịch vụ mới, lập trình viên chỉ phải tác động đến mã nguồn ở giai đoạn khởi tạo dịch vụ. Chẳng hạn thêm dịch vụ GoogleService như sau:

```
//Khởi tạo dịch vụ
ConcreteSearch cntSrc=new ConcreteSearch();
cntSrc.Attach(new AMZService());
cntSrc.Attach(new EBayService());
cntSrc.Attach(new GglService());
```

Phần mã nguồn cài đặt riêng cho dịch vụ GoogleService độc lập với các dịch vụ còn lại (thể hiện ở hình 6). Quá trình loại bỏ kết nối đến một dịch vụ được thực hiện ngược lại với quá trình thêm dịch vụ.



Hình 6. Hình minh họa trường hợp thêm dịch vụ mới cho ứng dụng tìm kiếm sản phẩm

c) Ngoài ra, sử dụng mẫu thiết kế này giúp ứng dụng ***tách biệt mã nguồn xử lý kết nối, triệu gọi dịch vụ và mã nguồn xử lý nghiệp vụ.*** Hơn nữa, nhiệm vụ tương tác với các dịch vụ được giao cho các lớp riêng biệt thực hiện. Đó cũng là cơ sở để giải quyết ***vấn đề khác biệt công nghệ dịch vụ web*** nếu xảy ra.

V.KẾT LUẬN

Phần mềm dựa trên nền tảng dịch vụ ngày

càng được phát triển và ứng dụng rộng rãi. Những mẫu thiết kế mới ra đời ở các mức độ trừu tượng khác nhau là rất cần thiết nhằm hỗ trợ những người phát triển phần mềm. Trong phạm vi nghiên cứu của mình, chúng tôi quan tâm đến việc vận dụng những mẫu thiết kế cơ sở, cụ thể là các mẫu GoF để giải quyết những vấn đề nảy sinh. Một số mẫu có thể giải quyết tốt, một số khác cần được cải tiến hay kết hợp với các mẫu khác một cách hợp lý để giải quyết triệt để vấn đề. Chúng tôi tin rằng những mẫu thiết kế này có thể giúp các phương pháp mô hình hóa hiện tại vẫn áp dụng tốt cho các kiến trúc phần mềm dựa trên nền tảng dịch vụ.

TÀI LIỆU THAM KHẢO

C. Alexander. *A Pattern Language*. Oxford University Press, 1977.

Chris Pelts, "Applying Design Patterns to Web Services Architectures", *XML Journal*, 2003.

Christian Thilmany , *NET Patterns Architechture, Design and Process*, Addison Wesley, 2003.

D. Riehle. "Composite Design Pattern", *Proc. of the 1997 Conference on Object-Oriented Programming Systems, Languages, and Application (OOPSLA'97)*, ACM Press, 1997.

E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Elements of Reuseable Object-Oriented Software*, Addison-Wesley, 1994.

F. Buschman, *Pattern-oriented Software Architecture – A System of Patterns*, John Wiley & Sons, 1996.

Hartwig Gunzer, "Introduction to Web Services", White paper by Borland Corporation, 2002.

Kai Qian, Orlando Karam, Jorge Diaz, Jiang Liu. "Design pattern for web service", *ACSI*, 2004.

Leonard Richardson and Sam Ruby, "RESTful Web Services", *O'reilly Media*, 2007.

Mark D. Hansen, *SOA Using Java Web Services*, Prentice Hall, 2007.

Massimilliano Bigatti, "Web service Integration Pattern, Part 1 – 2", Article on http://webservices.xml.com , 2004.

Michael Bell, *Service-oriented modeling: service analysis, design, and architecture*, John Wiley & Sons, 2008.

Nicolas Gold, Andrew Mohan, Claire Knight, Malcolm Munro, "Understanding Service-Oriented Software", *IEEE Computer Society*, 2004.

Nikola Milanovic, *Service Engineering Design Patterns*, *SOSE'06*, 2006.

P. A. Buhler, C. Starr, W. H. Schroder and J. M. Vidal, "Preparing for Service-Oriented Computing: A Composite Design Pattern for Stubless Web Service Invocation", *LNCS*, Vol 3140, Springer, 2004.

Paul B. Monday. *Web Services Patterns* – Java Edition, Apress, 2003.

Qusay H. Mahmoud, "Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)", Article on *Sun Microsytem*, 2005.

Rahim Lalani, "State-of-the-Art Web Services", MC Press online, 2006.

Thomas Erl, "Introducing SOA Design Patterns", *SOA World Magzine*, Vol.8, Issue 6, 2006.

Thomas Erl, SOA, *Principles of Service Design*, Prentice Hall, 2007.

Tran Dan Thu and Huynh Thuy Bao Tran, "A Composite Design Pattern for Object Frameworks", *COMPSAC* 2007, Beijing, China, 2007.

Tran Dan Thu and Huynh Thuy Bao Tran, "Composite design patterns to integrate available services", *SOSE'08*, 2008.