객체지향 컴포넌트 개발을 위한 동질 Hot Spot 설계 기법

°박동혁, 김수동
숭실대학교 대학원 컴퓨터학과

A Design Technique of Homogenous Hot Spot for Object-Oriented Components

°Dong Hyuk Park, Soo Dong Kim
Dept. of Computing, Soongsil University

요 약

컴포넌트 기반의 개발과 컴포넌트 개발에 대한 관심이 높아지면서 이를 위한 개발 방법론과 개발 도구들의 도입이 확산되고 있다. 또한 컴포넌트의 재사용 범위를 확장 시키기 위해 컴포넌트 개발 시 나타나는 Hot Spot 처리를 위한 분석, 설계 기법에 대한 관심이 높아 지고 있다. 따라서 본 논문에서는 Hot Spot 처리 시에 나타나는 의존적인 Hot Spot을 디자인 하는 방법을 제시하고 그 방법을 적용하여 의존적인 Hot Spot을 디자인 할 때 발생되는 Hot Spot간의 결합도 줄이고 디자인 모델의 가독성을 높인다. 본 논문에서는 디자인 모델링을 위한 표기법으로서 객체지향 모델링 기법인 UML(Unified Modeling Language)를 이용하여 모델링한다.

1. 서 론

소프트웨어 재사용에 대한 관심이 높아 지면서 관련된 기술들, 즉 컴포넌트 소프트웨어, 객체 지향 프레임 워크 같은 기술들에 이목이 집중되고 있다. 이런 기술들을 위한 분석 방법인 Commonality Analysis 와 같은 비슷한 여러 도메인 간의 공통점과 차이점을 분석하는 분석 방법들에 대한 관심 또한 높아지고 있다. 이러한 분석 기법을 적용하여 시스템의 안정적인 부분을 추출하여 재사용 가능한 모듈로 제작을 하게 된다. 이런 재사용 가능한 모듈들의 재사용 범위를 확장 하기 위해서 시스템의 변하기 쉬운 부분들, 즉 Hot Spot부분들을 모듈에 반영함으로써 보다 융통성을 갖는 모듈을 구성 할 수 있게 된다 본 논문에서는 객체 지향 컴포넌트 개발 시 나타날 수 있는 Hot Spot들이 밀접한 관계를 맺고 있을 때 나타나는 문제점들, 즉, 모델의 이해도 저하, Hot Spot 간의 결합도 증가, 모델 변경 시의 난해함, 같은 문제점들을 해결 하기 위해 디자인 패턴을 적용한 기법을 제시한다.

본 논문의 구성은 다음과 같다. 2 장에서는 컴포넌트 개발과 객체지향 프레임 워크 개발에서 많이 쓰이는 Commanlity Analysis 분석 방법과 추상 팩토리 설계 패턴에 대한 관련 연구를 기술 하고, 3 장에서는 의존적인 관계를 갖는 Hot Spot의 모델링 시 나타나는 문제점과 추상 팩토리 설계 패턴을 적용하여 의존관계를 맺는 Hot Spot의 모델링 기법을 제시한다.

2. 관련 연구

2.1. Commonality Analysis

Commonality Analysis에서는 비슷한 도메인으로부터 개발된 어플리케이션들을 어플리케이션 패밀리(Application Family)라 통틀어 지칭하는데 이들 사이에서 공통적인 부분들은 강조하고 공통적이지 않고 세부적인 부분들은 상대적으로 무시하는 도메인 분석 기법이다.[1] 그러므로 공통적인 부분들은 모든 패밀리 구성원들에 대하여 유효한 부분들이다.[2]

Commonality Analysis를 수행 함으로써 얻는 이점이 세가지가 있다. 첫째, 하나의 추상 개념하에 많은 클래스를 함께 묶음으로써 디자인의 복잡도를 줄일 수 있다. 둘째, commonality analysis를 통하여 묶여진 그룹들은 그들 간의 낮은 공통점으로 인해 자연스럽게 서로 결합도가 낮아지게 된다. 셋째, 공통점이 많을수록 시스템의 생명 주기가 길어지므로 유지 보수 비용이 줄어든다.[1]

그리고 Variability Analysis를 통해 각각의 패밀리 구성원들이 어떻게 다른지를 정의 하게 된다.

2.2. 추상 팩토리(Abstract Factory) 설계 패턴

추상 팩토리 설계 패턴은 패턴을 구분 짓는 기준으로 보면

객체 생성적 패턴으로 구분된다. 추상 팩토리 패턴의 목적은 관련된 혹은 의존적인 객체들의 패밀리를 구체적인 클래스들의 타입을 기술하지 않고 만들기위한 인터페이스(Interface)를 제공하는 것이다.[3] 아래에 있는 그림 1은 추상 팩토리 설계 패턴의 UML 클래스 다이어그램이다.

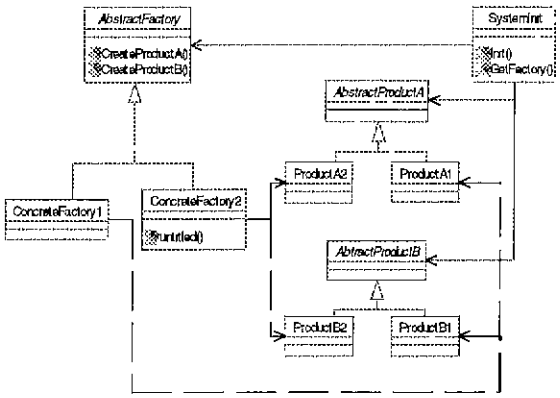

그림 1. 추상 팩토리 설계 패턴의 클래스 다이어그램

3. 문제점 분석 및 해결 기법

3.1. 동질 Hot Spot 의 모델링 시 문제점

본 논문에서는 Commonality Analysis, Variability Analysis 와 같은 분석 작업을 통하여 개념적 디자인 모델이 완성 되었다고 가정한다. 분석 작업을 통하여 생성된 디자인 모델 중 에는 의존 관계를 맺고 있는 Hot Spot 부분들이 있을 수 있다. 즉, 어떤 기능이 완전하게 수행되기 위해서, 시스템 초기화 작업 시에 어떤 Hot Spot A 에 대해 선택된 처리방법 A_1 과 그에 상응하는 다른 Hot Spot B 에 대한 처리방법 B_1 이 함께 선택되어야 하는 상황이 있을 수 있다. 이러한 의존관계를 갖는 Hot Spot 을 동질 Hot Spot 이라 정의한다. 이러한 동질 Hot Spot 을 [4] 에서는 Hot Spot 간의 Dependency Link 라 정의했다.

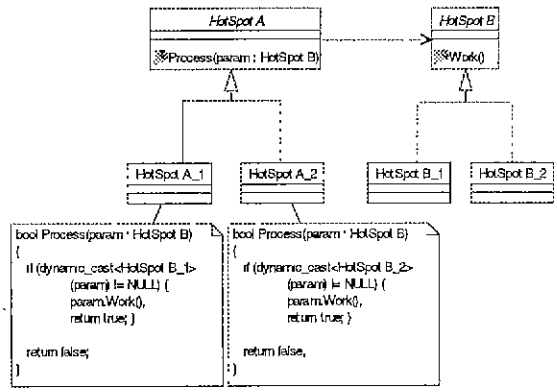

그림 2. 동질 Hot Spot 들의 클래스 다이어그램

그림 2에 있는 UML 클래스 다이어그램에서 나타난 것처럼 HotSpot A 클래스가 HotSpot B 클래스는 의존관계를 가지고 있다 HotSpot A 클래스의 자식 클래스인 HotSpot A_1 과 HotSpot A_2 클래스의 멤버 함수 Process 는 부모 클래스의 Process 함수를 오버라이딩(Overriding)해서 각각 HotSpot B_1 과 HotSpot B_2 타입의 객체가 인자로 전달 되었는지를 검사함으로써 적절한 객체가 인자로 전달되었는지 검사한다.

위의 그림의 Note 부분에 HotSpot A_1 과 HotSpot A_2 가 어떻게 인자의 타입을 검사하는지 C++ 코드로 표현되어 있다.

그림 2에서 나타난 모델의 단점은 다음과 같다.

- 첫째, HotSpot A_1 은 HotSpot B_1 의 타입을, 그리고 HotSpot A_2 는 HotSpot B_2 의 타입을 알아야 하기 때문에 HotSpot B_2 나 HotSpot B_1 의 클래스가 변경 되었을 경우 각각 HotSpot A_1 또는 HotSpot A_2 가 다시 컴파일이 되어야 한다.
- 둘째, 동적 타입 정보를 통한 타입 체크로 인하여 동적 바인딩 사용함으로써 얻는 이점이 상실된다.
- 셋째, 모델에서 HotSpot A 와 HotSpot B 사이의 관계가 명확 하지 않으므로 새로운 HotSpot A 와 HotSpot B 의 자식 클래스를 추가 시에 의미적으로 정확하지 않게 구현된 클래스를 추가 하기 쉽다.

3.2. Abstract Factory Pattern 을 적용한 모델링 기법

위에서 언급했던 동질 Hot Spot 을 가진 디자인 모델의 문제점들을 해결하기 위해 추상 팩토리 설계 패턴을 적용하여 모델링한다. 동질 Hot Spot 을 모델링 하기 위해 추상 팩토리 설계 패턴을 적용한 이유는 서로 관련된 클래스들이 함께 사용되어 한다는 제약 사항을 클래스 간에 두고 이러한 제약 사항에 대한 정보를 동질 Hot Spot 이 외의 클래스로부터 감추기 위함이다.

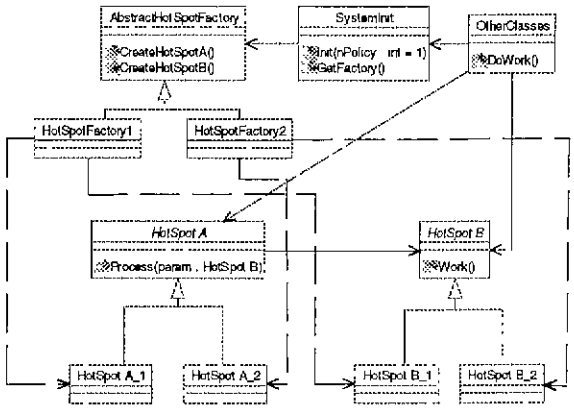

그림 3. 추상 팩토리 패턴을 적용한 동질 Hot Spot

그림 3은 그림 2의 모델에 추상 팩토리 설계 패턴을 적용하여 재구성한 모델이다 위의 그림에서 AbstractHotSpotFactory 는 HotSpot A 와 HotSpot B 의 관계에 대한 정보를 갖는 클래스들, 즉 HotSpotFactory1 과 HotSpotFactory2 의 부모클래스로서 HotSpot A 와 HotSpot B 타입의 추상 객체를 생성하기 위한 인터페이스를 선언하는 역할을 한다 실제로 HotSpot A 와 HotSpot B 클래스를 생성하는 작업은 AbstractHotSpotFactory 의 자식클래스들이 부모클래스의 인터페이스를 상속 받은 후 그 각각의 함수들을 오버라이딩(Overriding)함으로써 구현된다.

위에서 언급한 추상 팩토리 설계 패턴의 구성요소와 그림 3의 클래스들과의 연관관계는 표 1과 같다.

표 1. **Abstract Factory** 패턴과 재구성된 모델과의 연관관계

Abstract Factory Pattern	재구성한 모델
AbstractFactory	AbstractHotSpotFactory
ConcreteFactory	HotSpotFactory1 & 2
AbstractProduct	HotSpot A & B
ConcreteProduct	HotSpot A_1,2 & B_1,2
Client	SystemInit, OtherClasses

이렇게 재구성된 디자인 모델이 어떠한 순서로 메소드 호출이 일어나는지는 다음과 같다. 우선 컴포넌트가 제공하는 기능의 동작방식이 두 가지가 있고 컴포넌트의 인터페이스 중 하나에 컴포넌트의 동작 방식을 설정하는 함수가 있다고 가정한다. 컴포넌트 사용자는 컴포넌트가 동작 방식 2번 방식으로 동작 하기를 원한다고 가정하면, SystemInit 클래스의 InitPolicy 함수를 호출할 때 인자로서 2를 전달함으로써 컴포넌트의 동작방식을 설정한다. 이렇게 동작 방식이 설정된 다음 그 외 다른 객체들이 Hot Spot 부분들을 처리하는 객체를 생성하기 위해 GetFactory 함수를 호출 함으로써 팩토리 객체를 얻어온 후 팩토리 객체의 CreateHotSpotA 와 CreateHotSpotB 함수를 호출함으로써 Hot Spot 부분들을 처리하는 객체를 생성한다. 이런 모델의 동적인 면을 나타내는 UML 시퀀스 다이어그램(Sequence Diagram)이 그림 4에 나타나 있다.

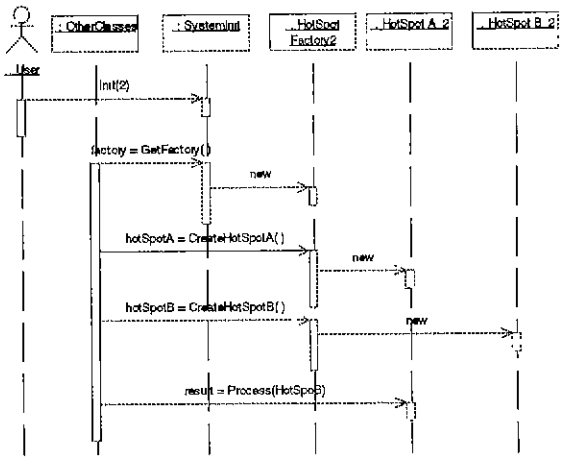

그림 4. 재구성된 모델의 동적 모델

이렇게 재구성된 모델의 장점은 다음과 같다.

- 첫째, 팩토리 클래스에 HotSpot A 와 HotSpot B 클래스의 의존관계에 대한 정보를 캡슐화함으로써 그 외 다른 클래스들은 그 의존관계에 대한 정보 필요 없이 구현될 수 있다
- 둘째, 새로운 Hot Spot의 자식 클래스가 추가 됨과 동시에 팩토리 클래스도 추가해야 함으로 의미적으로 잘못된 클래스를 구현할 가능성이 줄어든다.
- 셋째. Hot Spot 클래스 내에서 타입 체크를 하지 않으므로 동적 타입 정보를 실행 파일에 넣지 않아도 되므로 실행 파일의 크기를 줄일 수 있고 동적바인딩(Dynamic Binding)의 이점을 살릴 수 있다

3.3. 동질 Hot Spot의 문서화

대부분의 디자인된 모델은 만든 사람과 차후에 수정을 하는 사람이 다를 경우가 많다. 그러한 경우 모델을 수정할 사람이 동질 Hot Spot의 새로운 자식 클래스의 추가를 용이하게 하기 위한 동질 Hot Spot에 대한 문서의 양식을 BNF표기 법을 사용하여 정규화 한다. 다음은 그 문서에 대한 BNF이다.

```
<Hot Spot Description> .:= <Hot Spot>
        <Use Case>
        <Class Structure>

<Hot Spot>  .= "Hot Spot Name = " <Hot Spot Name> ";"

<Use Case> ::= "Use Case Name = " <Use Case Name> ";"

<Class Structure> ::=
        "Concrete Factory Name = " < Concrete Factory Name > "{"
        < Create Method Name> "CREATE" <Hot Spot Class Name>
        { < Create Method Name> "CREATE" <Hot Spot Class Name>
                }+ "}"
```

그림 5. 동질 **Hot Spot** 문서의 **BNF**

그림 5에서 Hot Spot Name 은 분석 단계에서 Hot Spot을 지칭하기 위해 정의된 이름을 기입한다. 그리고 Use Case Name은 Hot Spot이 Use Case 다이어그램에서 어떤 Use Case로부터 추출 되었는지를 명시하는 것이다 이렇게 분석 모델의 Use Case를 명시함으로써 분석 단계의 산출물과 설계 단계의 산출물을 연관지음으로써 모델의 수정이 용이해진다.

4. 맺음말

본 논문에서는 동질 Hot Spot들의 모델링 시에 나타나는 문제점들을 보완하기 위해 설계 패턴을 적용하였다.

향후 연구 분야로서 컴포넌트 개발 시 나타나는 Hot Spot 부분들의 처리방법 대한 기법 뿐만 아니라 컴포넌트의 사용자가 컴포넌트의 행위, 속성, 웍플로우(Workflow)등을 커스터마이즈 할 수 있도록 하는 기법 또한 개발 되어야 할 것이다.

5. 참고문헌

[1] James Coplien, *Multi-Paradigm DESIGN for C++*, Addison Welsley, 1999.

[2] David M. Weiss, *Commonality Analysis : A Systematic Process for Defining Families*, 2nd International Workshop on Development and Evolution of Software Architectures for Product Families, February 1998

[3] Gammar et al, *Design Pattern : Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994

[4] Ivar Jacobson et al, *Software Reuse : Architecture, Process and Organization for Business Success*, Addison Wesley, 1997

[5] Martin Fowler, UML Distilled, Addison Wesley, 1997.

[6] Clemens Szyperski, Component Software, Addison Wesley, 1997