계층적 접근 권한의 객체지향 모델링 사례

°박동혁, 김수동
숭실대학교 컴퓨터학과

A Case Study of Object-Oriented Modeling Hierarchical Access Right

°Dong Hyuk Park, Soo Dong Kim
School 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 이러한 분석,설계기법을 이용하여 융통적인 계층적 접근 권한에 대한 분석,설계 모델을 제시한다. 본 논문에서는 디자인 모델링을 위한 표기법으로서 객체지향 모델링 기법인 UML(Unified Modeling Language)를 이용하여 모델링 한다.

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 부분들을 모듈에 반영함으로써 보다 융통성을 갖는 모듈을 구성 할 수 있게 된다. 본 논문에서는 이러한 분석,설계기법을 이용하여 융통적인 계층적 접근 권한에 대한 분석,설계 모델을 제시한다.

본 논문의 구성은 다음과 같다. 2 장에서는 컴포넌트 개발과 객체지향 프레임 워크 개발에서 많이 쓰이는 Commanlity Analysis 분석 방법과 디자인 패턴에 대한 관련 연구를 기술 하고, 3 장에서는 계층적 접근 권한의 용어를 정의하고 그에 대한 개념적 모델을 제시한다. 그리고, 4 장에서는 3 장에서 소개된 개념적 모델의 설계모델과 구현모델을 사례로 제시한다.

2. 관련 연구

2.1. Commonality Analysis

Commonality Analysis 에서는 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]

패밀리를 정의 하기 위해서는 패밀리를 서술하기 위해 사용되는 용어들을 정규화 하는 작업과 패밀리의 범위를 정의하기 위해 각각의 패밀리 구성원들이 어떻게 변화될 지를 예상하는 작업을 할 필요가 있다. Commonality Analysis 를 수행 함으로써 얻는 이점이 세가지가 있다. 첫째, 하나의 추상 개념하에 많은 클래스를 함께 묶음으로써 디자인의 복잡도를 줄일 수 있다. 둘째, commonality analysis 를 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 Analysis 를 통해 각각의 패밀리 구성원들이 어떻게 다른지를 정의 하게 된다. 이렇게 발견된 차이점들은 그 각각의 종류에 따라 나뉘어 지게 되고 각각의 차이점들이 가질 수 있는 값들의 범위가 정의된다.

2.2. 설계 패턴(Design Pattern)

디자인 패턴이란 어떤 특정한 조건에서 일반적인 디자인 문제점을 해결하기 위해 고안된 객체와 클래스의 기술서이다. 일반적으로 디자인 패턴은 네 개의 필수 요소를 가지고 있는데, 그 각각은 패턴 이름, 패턴을 적용할 시점을 나타내는 디자인 문제, 디자인 모델을 구성하는 요소들을 나타내는 해결 법, 패턴을 적용한 후의 결과 또는 장단점들이다. 그리고 디자인 패턴을 분류하는 기준에는 두 가지가 있는데, 그 중 하나는 목적에 따라 분류하는 방법으로, 생성적(Creational), 구조적(Structural), 행위적(Behavioral)으로 분류된다. 그리고 또 다른 분류 기준은 패턴을 적용하는 대상에 따라 분류하는 방법으로, 적용 대상이 객체(Object)인 패턴과 클래스(Class)인 패턴으로 나뉘어진다.[3]

3. 개념적 모델

본 논문에서 계층적 접근 권한이란 용어는 다음과 같은 경우를 지칭하기 위하여 사용한다. 일반적으로 기업에서는 어떤 명령어나 자원, 혹은 문서등을 그것의 중요도에 따라 접근할 수 있는 사용자를 여러 계층으로 나누어 관리하는 경우가 많다. 그리고 그러한 사용자 계층들 사이에서 어떤 특정 계층의 권한은 다른 계층의 권한을 포함하는 경우가 많다. 즉, 포함하는 계층의 사용자는 포함되는 계층의 사용자가 접근할 수 있는 영역 또한 접근할 수 있다는 것이다.

이러한 계층적 접근 권한을 모델링 하는데 있어서 고려할 문제 요구사항은 다음과 같다. 먼저, 제시될 모델은 새로운 사용자 계층의 추가 혹은 수정과 같은 문제 요구사항의 변화에 의해 추가 혹은 수정 되는 부분은 해당 모듈로, 여기서는 클래스, 한정되야한다. 이러한 모듈의 연속성(Modular continuity)은 제시된 모델의 확장성과 직접적인 관련이 있다[7] . 그리고, 계층적 접근 권한과 접근 대상이 되는 서비스 사이의 관계에 대한 정보는 접근 권한 클래스와 접근 대상 클래스로 한정되도록 해야한다.

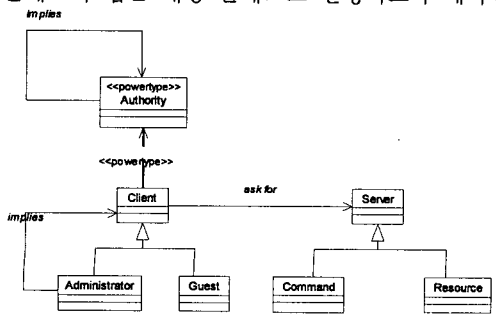

그림 1. 계층적 접근 권한의 개념적 모델

그림 1은 계층적 접근 권한의 개념적 모델을 UML 표기법으로 나타내고 있다. 그림 좌측에 나타난 Client 와 Authority 는 계측적 접근 권한을 나타내는 클래스이며 Client 클래스의 하위 클래스들이 Authority 클래스의 인스탄스가 되는 것을 모델링 하기위해 두 클래스 사이에 <powertype> 스테레오 타입의 의존관계를 설정했다. 그림 좌측에 나타난 Sever 와 그 하위 클래스들은 Client 객체가 필요한 서비스를 제공하는 클래스로서, Client 객체의 권한의 수준에 따라 이용할 수 있는 서비스의 종류가 다르다. 그리고 각각의 Server 클래스의 하위 클래스들이 추출된 문제 도메인에 따라 각각의 서비스들은 하나의 클래스의 다른 메소드로 혹은 서로 다른 클래스로 할당되어 모델링 될 수 있다.

4. 설계 및 구현 사례

그림 1과 같은 개념적 모델을 바탕으로 일반적인 기업의 조직 관리 시스템 중에서 직원관리 시스템의 설계 모델과 EJB 를 구현 아키텍처로 한 구현 사례를 이 장에서 제시한다.

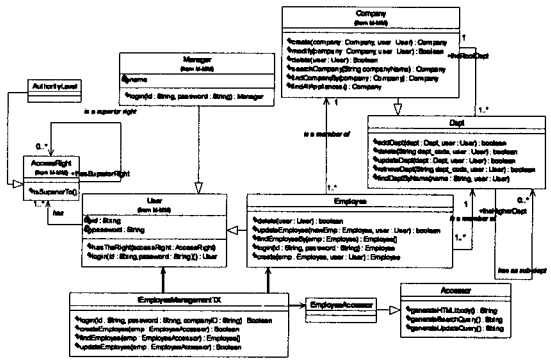

그림 2. 조직관리 시스템의 설계 모델

그림 2는 조직관리 시스템의 설계 모델을 나타내고 있다. 그림에서 User 클래스는 계층적 접근 권한을 갖고 있는 즉 개념적 모델에서의 Client 클래스가 된다. User 클래스는 User 객체마다 다른 접근 권한을 부여 하기 위해 AccessRight 객체를 소유하고 있다. 이러한 AccessRight 객체들은 객체들 사이에 포함 관계를 갖고 있어서 어떤 User 객체가 어떤 서비스를 사용할 권한이 있는가를 검사하기 위해서는 반복적으로 AccessRight 객체에 요청을 보내어 검사한다. User 클래스의 하위 클래스인 Employee 클래스는 개념적 모델의 Client 클래스와 Server 클래스의 역할을 동시에 하고 있다. 즉 어떤 특정 Employee 객체는 다른 Employee 객체를 생성,수정, 조회와 같은 권한을 갖을 수 있고, 또 다른 Employee 객체는 조회만 할 수 있는 권한을 갖을 수 있다는 것이다. 그 외 개념적 모델의 Server 클래스 역할을 하고 있는 것들은 Company, Dept 클래스이며, 이들 각각은 회사와 부서를 나타낸다.

이들 Server 개념적 클래스 역할을 하고 있는 클래스들은 각각의 메소드에 User 클래스의 객체를 파라메터로

갖고 있으며 실행시에 User 객체가 오퍼레이션을 수행할 권한이 있는지를 검사 한 후 권한을 갖고 있을 때에만 오퍼레이션을 수행한다.

표 1. 클래스의 빈으로의 맵핑

빈의 종류	클래스
Session Bean	IEmployeeManagement
Entity Bean	Employee, Company, Dept

표 1은 설계 모델에 나타난 클래스들이 EJB 구현에서 어떤 Bean 으로 구현되야 하는지를 부여주고 있다. Employee, Compandy 그리고 Dept 클래스는 모두 지속적인 특성을 가지고 있기 때문에 EntityBean 으로 구현하고, IEmployeeManagement 클래스는 직원 관리 모듈을 사용하는 사용자의 수만큼 생성되어 각각의 사용자에 대한 정보를 보관한다. 즉 IEmployeeManagement 의 login()메소드를 이용하여 접속한 사용자, 즉 Employee 혹은 Manager 객체의 인스탄스는 IEmployeeManagement 객체에 보관되어 Employee, Company, Dept 객체의 함수의 인자로 전달되어 login() 메소드를 호출한 사용자가 접근 권한 소유 여부를 판단한다.

```
class IEmployeeManagement implements EntityBean {
  User user;
  public boolean login(string id, string passwd) {
            ....
            user = userHome.findByPrimaryKey(new UserPK(id) );
            user.login(passwd);
  }
  public boolean createEmployee(EmployeeAccessor emp) {
            try {
                emp = empHome.create(emp, user);
            } catch (CreateException e) {
                        ...
            }
  }
            ...
}
```

```
Class Employee implements EntityBean {
  public void ejbCreate(EmployeeAccessor emp, User usr)
                                    throws CreateException {
            static AccessRight createRight = new AccessRight(2);
            if (! usr.hasTheRight( createRight ) ) {
                        throw new CreateException("Not Allowed");
            }
            ....
  }
            ....
}
```

그림 3. 접근 권한 검사 코드 예

그림 3은 접근 권한 검사 코드의 일부를 보여주고 있다. 그림에서 상단에 있는 코드는 직원 관리 시스템에 접속한 사용자 정보를 어떻게 보관하는지를 보여주고 있으며, 하단에 있는 코드는 사용자가 새로운 Employee 객체를 생성할 권한이 있는지를 어떻게 검사하는지를 보여주고 있다.

5. 맺음말

본 논문에서는 계층적 접근 권한에 대한 개념적 모델과 설계 모델을 제시하였다. 제시된 모델은 계층적 접근 권한은 나타내는 부분과 서비스를 제공하는 부분을 서로 분리시켜 새로운 계층의 접근 권한이 추가 되어도 서비스를 제공하는 클래스의 구조나 구현이 변경될 필요가 없으며, 새로운 서비스가 추가 되어도 접근 권한을 나타내는 부분이 영향을 받지 않으므로 시스템의 유지 보수와 확장이 용이하다.

향후 연구 분야로서 이러한 객체지향 설계 모델을 COM,EJB,Corba Component Model 과 같은 컴포넌트 환경에 맞게 정제하는 기법이 개발 되야 할 것이다.

6. 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

[7] Bertrand Meyer, *Object-Oriented Software Construction 2nd Edition*, Prentice Hall, 1997.

[8] Sun, *Enterprise Java Beans 1.1 Spec*, Sun, 1999