역할 객체를 이용한 응용 프레임워크의 동적 생성

한익주
한국산업 기술대학교 컴퓨터 공학과
{ijhan}@kpu.ac.kr

Dynamic Instantiation of Application Framework Using Role Object

Ikjoo Han
Dept. of Computer Eng., Korea Polytech. University

요 약

응용 프레임워크를 이용하여 응용 프로그램을 작성할 때, 응용 프레임워크와 함께, 그 프레임워크와 별도로 기존에 작성된 클래스를 이용하려면 다중 상속을 이용하거나 단순한 복사/붙이기와 같은 코드 재작성을 이용하여야 한다. 그러나 다중 상속은 여러 문제점을 가지고 있고, 코드 재작성은 단순한 짜깁기 이상은 될 수 없다. 또한 컴포넌트 기반의(Component-Based) 개발 방식을 따를 경우 프로그램 코드를 획득하거나 이해한다는 것도 쉬운 문제가 아니다. 역할 객체는 어떤 객체가 특정 콘텍스트에서 필요한 행동 양태(Behavior)를 가지는 객체를 의미한다. 본 논문에서는 이러한 프레임워크에 기작성된 클래스를 적용하기 위해 역할 객체를 이용하기를 제안하였다.

1. 서론

객체 지향 개발(Object-Oriented Development)에서 언급되고 있는 재사용(Reusability) 중, 응용 프레임워크(Application Framework)를 이용한 재사용은 소프트웨어의 설계의 재사용을 제공할 뿐만 아니라, 프레임워크 자체가 프로그래밍 언어로 작성되어 있기 때문에 소스 코드(Source Code) 수준의 재사용까지 보장한다. 응용 프레임워크에서 필요한 응용 프로그램을 개발하는 일반적인 방법은 프레임워크에서 응용 프로그램에 적합하게 개발되도록 구현이 유보되어 있는 추상클래스(Abstract Class)들에 하위 클래스(Subclass)들을 상속을 이용, 작성하여 추가함으로써 개발하고자 하는 응용 프로그램을 완성하는 방법을 이용한다. 이러한 방법은 기존의 개발되어 있는 클래스들을 가지고 프레임워크와 연계하여 새로운 프로그램을 개발하고자 할 때는 적잖은 문제점이 일어 난다. 우선 기존에 작성된 클래스와 프레임워크의 하부 클래스로 들어가 있는 추상클래스, 둘 다의 형질을 가지는 클래스를 필요로 하기 때문에 이를 충족시킬 수 있는 구조(Construct)가 필요하다. 여러 객체 지향 프로그래밍 언어에서 한 개 이상의 클래스의 형질을 상속받는 구조로 다중 상속(Multiple Inheritance)을 제공하고는 있지만 다중 상속은 [1]에서 볼 수 있는 것처럼 여러 문제점을 가지고 있고, 다중 상속의 문제점을 해결하기 위해 제안된 여러 방법들도 나름대로의 한계를 가지고 있다. 이러한 기작성된 클래스들을 이용한 프레임워크의 적용은 기작성된 이진 코드 수준의 재사용을 바탕으로 하는 컴포넌트 기반의 프로그램 개발에서는 더군다나 필수 불가결하다. 컴퍼넌트 기반의 소프트웨어 개발에서는 각 컴퍼넌트를 개발하는 곳이나, 프레임워크를 개발하는 곳이 같은 개발자일 수 없으며, 개발된 컴퍼넌트나 프레임워크가 소스 코드 수준에서 제공된다는 보장도 없기 때문이다.

본논문에서는 프레임워크를 기존의 클래스들을 이용하

여 적용하기 위해서 기존 클래스의 일부 소스코드를 단순히 복사하여 짜깁기하는 수준의 재사용이나, 다중 상속에 의지하지 않고, 기존 클래스를 프레임워크에 접목하여 프레임워크를 재사용하는 방법에 대하여 언급한다. 이 방법은 역할(Role) 객체의 개념을 프레임워크의 추상 클래스에 확대하여 프레임워크의 동적인 생성을 지원한다.

2. 역할(Role)

역할의 개념은 특정 콘텍스트(Context)에서 한 객체 또는 여러 개의 객체들이 연합하여 수행하는 행동 양태(Behavior)를 의미한다. 역할은 소프웨어의 분석, 설계, 그리고 구현 또는 관계형 데이터베이스 등에서 연구되었지만 역할을 하나의 객체로 보면 Kristensen[2]의 역할과 같이, 다음의 개념을 가진다. 지면 관계상 모든 특징을 나열하지는 않았다.

1) 역할은 내적(Intrinsic) 객체가 외적(Extrinsic) 객체에 참가하므로써 생성되어 진다. 즉, 내부 객체가 없는 역할은 생성될 수 없다.
2) 하나의 내적 객체는 여러 개의 외적 객체(역할)에 참가할 수 있다.
3) 역할은 원하는 콘텍스트에서 외적 객체를 통하여 내부 객체의 내용을 접근한다.
4) 외적 객체는 단순하게 내적 객체의 반영(Projection)이 아니다.

예를 들어 어떤 사람의 객체는 콘텍스트에 따라 학생,

(a) (b)

그림 1. 역할과 내적 객체

회사원 등의 역할을 수행할 수 있으며, 그 객체가 집 전화 번호를 가지고 있다면, 회사원의 역할을 수행할 동안 그 사람의 전화 번호는 직장 내에서의 구내 전화번호가 될 것이며, 집 전화 번호는 따로 집 전화 번호로 다루어질 것이다. 그림 1의 (a)에서는 하나의 역할 클래스(Class)와 그의 내적 객체가 될 수 있는 클래스를 도시했으며 (b)에서는 그에 따른 하나의 학생 역할 객체의 생성(내적 객체는 '철수')을 보여 주고 있다.

3. 역할을 이용한 프레임워크의 동적 생성

응용 프레임워크는 일반적으로 추상 클래스의 집합으로 이루어진 패키지(Package)로 볼 수 있다. 프레임워크의

그림2. 프레임워크 클래스

추상 클래스가 그 프레임워크 내에서 추상 클래스와 다른 클래스 간의 관계를 포함한 행동 양식을 나타내며, 추상 클래스의 구현이 유보되어 있다는 점에서 역할 클래스 개념의 부분 집합적인 성격을 가진다. 이러한 프레임 워크의 추상 클래스들은 프레임워크라는 콘텍스트 내부에서 어떤 객체가 수행하여야 하는 역할로 볼 수 있다. 객체들을 어떤 프레임워크의 추상클래스의 역할에 참여 하게 하기 위해서, 프레임워크 자체도 하나의 클래스로 정의되어 각 추상 클래스의 역할을 객체들이 맡을 수 있도록 한다. 이렇게 프레임워크를 클래스로 만듦으로써 프레임워크의 동적인 생성을 가능하게 할 뿐만이 아니라, VanHilst와 Smaragdakis의 Collaboration 연구[3, 4]에서 볼 수 있는, 한 역할 클래스의 종류가 다른 역할 클래스의 수의 증대에 따라 폭발적으로 증대하는 Scalability의 문제도 해결할 수 있다. 예를 들어 하나의 프레임워크 클래스는 그림 2 와 같이 도식될 수 있다. 그림에서 각 역할 클래스는 프레임워크 클래스의 내부에 선언되어 있다. 다시 말하면 각 역할 클래스는 그 프레임워크의 콘텍스트에서만 의미를 가지는 내부 클래스(Inner Class)로 존재한다. 그러므로 어떤 객체가 한 프레임워크의 생성에 참여하면, 프레임워크 내부의 콘텍스트에서 그 객체의 내용은 참여한 역할 클래스를 통해서만 접근할 수 있다. 그림 3 에서는 하나의 생산자/소비자(Producer/Consumer) 관계를 나타내는 프레

임워크 클래스의 생성된 프레임워크 객체를 보여 주고 있다. 이 간단한 프레임워크에서는 생산자는 일단 생산된 데이터들을 저장하고 있다가 데이터를 소비할 때 소비자가 생산자가 가지고 있는 데이터 중 하나를 생산자

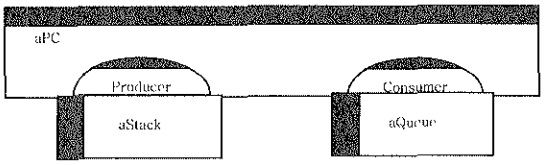

그림3. 생산자/소비자 프레임워크의 생성

의 구현 방법에 따라 전달 받아 소비하고 자신의 저장소에 저장한다. 이 프레임워크의 생산자와 소비자는 스택, 큐 등과 같은 자료 구조 클래스로 구현될 수 있는데 그림 3 에서는 생산자와 소비자가 각각 하나의 스택과 큐로 구현된 경우를 보여 주고 있다. 여기서 내적 객체가 되고 있는 스택과 큐 객체들은 기존에 작성되어 있는 클래스들에서 생성된 객체들이다. 그러므로 이렇게 역할 객체를 이용할 경우, 단순한 짜깁기나 다중 상속과 같은 불완전한 방법을 이용하지 않고도 기작성된 클래스를 이용하여 프레임워크를 적용할 수 있게 된다.

위와 같이 개념적으로 설명된 역할 클래스를 이용한 프레임워크의 생성을 C++ 언어와 같은 객체 지향 언어로 구현하고자 한다면, 프레임워크를 하나의 클래스로 정의하고, 각 역할 클래스를 프레임 클래스의 내부 추상 클래스로 정의하고, 내적 객체를 인수로 받아야만 생성될 수 있는 역할 클래스의 하위클래스를 정의하여야 한다. 다음 코드에서는 C++로 작성된 생산자/소비자 예의 프레임워크 클래스의 뼈대를 보여 주고 있다.

```
class PCFramework
{
public:
        class Producer {
        public:
                virtual int Produce(int data) = 0;
                        .....
        };

        Producer *myProducer; // The First Role
        Consumer *myConsumer; // The Second Role

        PCFramework(Producer *p, Consumer *c)
        {
                myProducer = p;
                myConsumer = c;
        }
};
class StackToProducer : public PCFramework::Producer
{
        Stack *intrinsic;
public:
        StackToProducer(Stack *s)
        {
                intrinsic = s;
        }
                .
                .....
};

PCFramework aPC(new StackToProducer(&aStack),
        new QueueToConsumer(&aQueue));
```

4. 결론 및 향후 연구

본 논문에서는 역할 객체를 이용하여, 간단하게 프레임워크에 기작성된 클래스를 적용하는 방법을 설명하였다. 이러한 방법은 컴포넌트 기반의 소프트웨어 개발시 적용될 수 있는 정도 이상의 동적 구성(Dynamic Composition)을 가능하게 한다. 즉 프레임워크 안에 다른 프레임워크가 참여하는 방식의 동적 구성도 가능하다. 향후 이러한 내용과 역할 객체를 위한 보다 견고한 언어적 구조에 대한 연구가 뒤따라야 할 것이다.

5. 참고 도서

[1] Clemems Szyperski, "Component Software, Beyond Object-Oriented Programming", pp.96-100, Addison-Wesley, 1997.

[2] Bent. B. Kristensen and Kasper Osterbye, "Roles: Conceptual Abstraction Theory & Practical Langauge Issues", Special Issue of TAPOS on Subjectivity in Object-Oriented Systems, 1996.

[3] Michael VanHilst and David Notkin, "Using Role Components to Implement Collaboration-Based Designs", OOPSLA '96, pp. 359-369, CA., USA..

[4] Yannis Smaragdaski and Don Batory, "Implementing Layered Designs with Mixin Layers", Proceedings of ECOOP'98, pp. 550-570, Springer Verlag, Lecture Notes in Computer Science.