C# 언어에서 컴포넌트 속성 가변성 구현 기법

김상욱[0], 김수동
숭실대학교 컴퓨터학과
swkim@otlab.ssu.ac.kr, sdkim@comp.ssu.ac.kr

Techniques to Implement Component Attribute Variability in C#

Sang Wook Kim[0], Soo Dong Kim
Dept. of Computing, Soongsil University

요 약

가변성은 컴포넌트의 재사용성을 향상시킬 수 있는 요소이다. 어떤 도메인의 여러 패밀리에서 사용될 수 있는 컴포넌트를 개발하기 위해 컴포넌트 가변성이 강조되고 있다. 개발된 컴포넌트가 다양한 유형의 가변성을 지원할수록 도메인 내에서 재사용성이 높아진다. 하지만, 컴포넌트가 너무 많은 가변성을 지원하도록 개발되면, 컴포넌트의 크기가 거대해지며, 개발 기간과 개발 비용이 증가하게 된다. 따라서 해당 도메인에 맞는 적절한 가변성을 지원하는 컴포넌트를 개발해야 한다. .NET의 C# 언어는 컴포넌트 개발 언어로 산업계에서 각광 받고 있다. 본 논문에서는 C#에서 제공하는 기본적인 장치를 활용해 속성 가변성을 지원하는 컴포넌트를 구현 기법을 제시한다. 클래스를 동적으로 생성할 수 있는 빌더(Builder) 클래스를 통해 속성 가변성을 지원한다. 그래서 여러 패밀리 멤버에서 재사용 할 수 있는 고품질의 컴포넌트 개발 방법을 제시한다.

1. 서론

객체단위의 소프트웨어 재사용의 한계를 극복하고자 컴포넌트 단위의 재사용이 요구되고 있다. 컴포넌트 단위의 재사용은 시스템 개발 시 복잡성을 감소시키고 재사용성을 증가시켜, 시스템 개발 비용과 기간을 단축 할 수 있는 효율적인 방법이다[1]. 따라서 컴포넌트의 재사용성을 높이기 위한 많은 연구가 수행 되고 있다. 그 중에서 가변성은 컴포넌트의 재사용성을 높이기 위한 기법으로 평가 되고 있다.

그러나 현재까지 컴포넌트 가변성에 대한 연구가 실제 프로젝트에 적용 할 수 있는 구체적인 기법을 제시하지 못하고 있다[1]. 이유는 업무에 따라 매우 다양한 가변성이 있어 한 컴포넌트에 가변성이 포함되도록 설계하기가 어렵기 때문이다. 그래서 컴포넌트에 존재하는 여러 유형의 가변성을 구현하기 위해서는 각각의 가변성 유형에 맞는 개발 방법이 필요하다.

본 논문에서는 C#에서 제공하는 기본적인 장치를 활용해 속성 가변성을 지원하는 컴포넌트를 구현한다. 제시된 기법은 빌더 클래스를 통해 가변성을 가지는 클래스를 동적으로 생성하는 기법이다. Required 인터페이스를 통해 가변성을 설정하여 가변성을 반영할 수 있다. 즉 재컴파일 작업 없이 속성 가변성을 반영할 수 있다.

본 논문에서는 C#으로 컴포넌트의 가변성 중 속성 가변성에 대한 구현 방법을 제시한다. 제2장에서는 C#에서 가변성 구성에 도움을 주는 여러 장치를 살펴본다. 제3장에는 가변성 설계 방법을 제시하고, 제4장에서는 구현 방법을 제시한다. 제5장에서 사례 연구를 한다.

2. 기반 연구

동적 생성자(Static Constructor)는 클래스의 객체가 메모리에 생성될 때 자동으로 한번만 호출되는 생성자이다[3]. 한번 호출되면 프로그램이 재실행되기 전까지는 호출되지 않는다. 매개변수를 사용할 수 없기 때문에 여러 개의 동적 생성자를 가질 수 없다. 직접 생성자를 호출할 수 없다. 비정적 멤버를 사용할 수 없다.

IL(Intermediate Language)는 .NET의 어떤 특정한 언어를 사용했는지에 상관없이, 중간 언어의 IL을 만든다[3][4]. 자바에서의 바이트 코드와 같은 개념이다. C#, VB, VC++ 등의 소스 코드는 서로 다르더라도 컴파일하면 모두 IL 코드 형태로 같은 형태로 나온다. 그러므로 모든 언어는 적절하게 정의된 가상적인 바이너리 공간 안에서 상호 작용이 가능하다. IL 코드의 문법은 어셈블러와 비슷하다.

Assembly는 .NET 프레임워크 어플리케이션의 빌딩 블록(Building Blocks)이다[5]. 배치(Deployment), 버전관리, 재사용, 보안관리 등의 기본 단위이다. 일반적으로, Assembly는 Assembly 메타데이터, 타입 메타데이터, MSIL 코드, 리소스로 이루어진다.

3. 속성 가변성 설계

속성 가변성은 여러 패밀리 멤버에서 동일한 기능을 수행하지만, 필요로 하는 속성의 종류, 타입 또는 개수가 다른 경우를 말한다[5]. 단, 속성의 타입이 호환 가능한 경우에는 동일한 속성으로 분류되어, 속성 가변성 유형에서 제외 된다.

속성 가변성에는 중요한 특징이 있다. 속성 가변성은 패밀리 멤버 간에 속성의 종류, 타입 또는 개수만 다른 것이 아니라 그 속성을 사용하는 멤버함수도 영향을 받아 멤버함수도 변경된다. 즉 속성 가변성은 로직 가변성을 포함하고 있다.

그림 1은 속성 가변성을 위한 클래스 설계를 나타낸다. P는 컴포넌트가 제공하는 서비스를 나타내는 Provide 인터페이스다. R은 가변성을 설정을 위한 Required 인터페이스다. 그림 1을 보면 빌더와 A 클래스가 있다. A 클래스는 속성 가변성을 가지는 클래스이다. 빌더 클래스를 통해 동적으로 생성된다. 빌더 클래스는 클래스를 동적으로 생성하는 클래스이다.

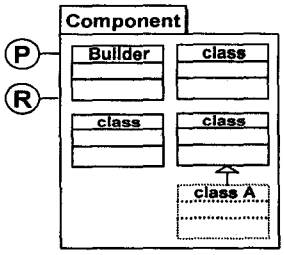

그림 1. 속성 가변성 클래스 설계

공통성 클래스와 가변성을 가지는 클래스로 분리해 설계하는 이유는 Unknown 가변성을 위해서이다. 그리고 속성 가변성은 멤버 함수에도 영향을 주기 때문에 멤버 함수도 변경될 수 있다.

그림 2는 Required 인터페이스를 통해 가변성을 설정 순서도를 나타낸다. 빌더 클래스에서 외부 파일을 읽어, 동적으로 클래스를 생성한다.

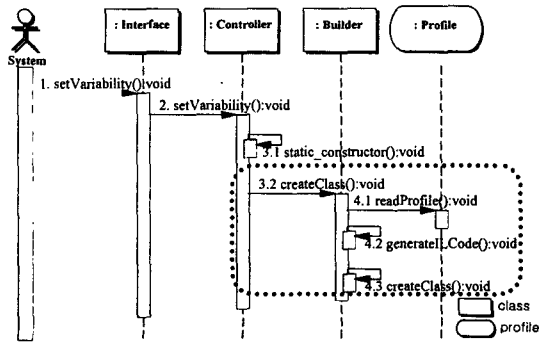

그림 2. 가변성 설정 순서도

가변성을 가지는 클래스의 정보는 외부 파일에 저장된다. 빌더 클래스는 가변성이 기록된 외부 파일(Pofile)을 통해 클래스를 동적으로 생성한다.

빌더 클래스를 통해 클래스를 동적으로 생성하는 장점은 속성 타입이 변경되거나 새로운 속성을 추가 해야 할 경우 외부 파일에 있는 클래스 정보를 변경하고 프로그램을 재실행하면 새로 수정된 내용이 적용된다는 것이다. 속성을 사용하는 멤버 함수도 변경이 가능하다. 코드를 수정하고 다시 컴파일 할 필요가 없기 때문에 컴포넌트가 유연해진다.

그림 3은 동적으로 생성된 클래스의 메소드를 호출하는 순서도를 나타낸다. 어떤 메소드 호출이 있을 때 빌더 클래스r로부터 생성된 클래스의 객체를 만들어 리턴한다. 그리고 리턴된 객체를 통해 메소드를 호출한다. 외부 파일로부터 읽은 가변성 내용에 맞게 객체를 리턴한다.

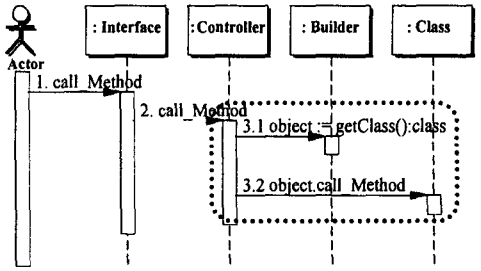

그림 3. 메소드 호출 순서도

4. 속성 가변성 구현 기법

빌더 클래스가 클래스를 동적으로 생성할 수 있도록 C#에서는 IL을 지원한다. 빌더 클래스가 외부 파일에 기록된 내용을 IL 코드로 변환해 클래스를 생성한다. IL을 사용하여 가변성을 지원하는 컴포넌트를 개발하면 간결하면서 효과적인 코드로 컴포넌트를 개발할 수 있다.

IL 코드는 두 가지 형태로 있다. 16진수 형태의 한 바이트 형태와 OpCode 형태가 있다. 예를 들어 한 바이트 형태의 함수 호출하는 IL 코드 0x28이고, OpCode는 call이다. 빌더 클래스에소는 OpCode 형태의 IL 코드를 사용하기 쉽다.

그림 4는 빌더 클래스의 일부분을 나타낸 코드이다. 클래스를 동적으로 생성하기 위해서는 Assembly 정의, 파일 정의 등 .NET에서 기본적으로 필요한 정보들을 정의해야 한다. 그리고 멤버 함수, 멤버 변수를 정의한다. ModuleBuilder는 동적으로 생성되는 클래스가 저장될 DLL 파일을 생성한다. TypeBuilder 클래스는 동적으로 생성할 클래스의 형태를 정의한다.

```
createClass()
{
  //클래스 정의
  AssemblyName.assemblyName     = new assemblyName();
  assemblyName.Name             = className;
  assemblyName.Version          = new Version("1.0.0.1");
  AppDomain curAppDomain        = Thread.GetDomain();
  AssemblyBuilder assembly      = curAppDomain.DefineDynamicAssembly
                                  (assemblyName, AssemblyBuilderAccess.Save);
  ModuleBuilder module          = assembly.DefineDynamicModule
                                  (className, className + ".dll")
  TypeBuilder typeBuilder       = module.DefineType(className + "." +
                                  className, TypeAttributes.Public)

  // 메소드 정의

  readProfile();
  GenerateILCode();
  createClass();
}
```

그림 4. 빌더 클래스 구현

외부 파일이 어떤 형태도 중요한 부분이다. 빌더 클래스가 외부 파일에 작성된 내용을 IL 코드로 변환하기 때문에 IL 코드로 변환하기 쉬운 형태가 좋다. 그리고 사람이 알아보기 쉽고, 사용하기 쉬운 형태가 되어야 한다. 잘못된 Profile 설계는 IL 코드 변환 작업이 힘들어지기 때문에 빌더 클래스 구현이 힘들어진다.

5. 사례 연구

A, B은행의 고객 정보를 관리 하는데 사용되는 속성은 A 은행은 ID, 이름, 전화번호, 주소이며, B은행은 ID, 이름, 전화번호, 직업, 회사전화번호, 결혼여부가 고객 정보 관리에 사용된다.

그림 5는 속성 가변성을 위한 클래스 설계이다. A, B 은행의 공통적인 부분은 Customer클래스에 구현된다. 그리고 A, B 은행의 가변성 부분은 외부 파일에 정의된다. 정의된 외부 파일로부터 CustomerA 클래스가 생성된다.

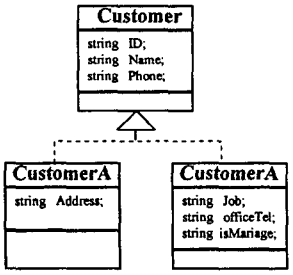

그림 5. 속성 가변성 클래스 설계

빌더 클래스로부터 생성되는 CustomerA 클래스는 상위 클래스인 Customer형태로 리턴된다. metaAPI를 통해 메소드를 찾아 호출한다. Customer형태로 리턴되는 이유는 CustomerA는 동적으로 생성되기 때문에 CustomerA 클래스를 프로그램 내에서 선언할 수 없기 때문이다.

그림 6는 A은행의 CustomerA 클래스의 IL 코드이다. IL 코드의 문법은 어셈블러와 비슷하다. CustomerA 클래스는 Address 변수와 getAddress, setAddress 함수로 정의된다. 빌더 클래스는 외부 파일에 기록된 속성 가변성 정보를 통해 CustomerA클래스를 생성한다. 그래서 Customer.dll과 CustomerA.dll이 생성되며, 이 두 파일이 하나인 것처럼 사용된다.

```
.class public auto ansi CustomerA extends Customer
{
.field private string address
.method public specialname rtspecialname
                                        instance void .ctor() cil managed
{
  .maxstack 2
  IL_0000: ldarg.0
  IL_0001: call  Customer::.ctor()
  IL_0006: ret
}
.method public instance string  getaddress() cil

                          managed
{
.maxstack 16
  IL_0000: ldarg.0
  IL_0001: ldfld
  IL_0006: ret
}
.method public instance void  setaddress
                                        (string var) cil  managed
{
.maxstack 16
  IL_0000: ldarg.0
  IL_0001: ldarg.1
  IL_0002: stfld
  IL_0007: ret
}
}
```

그림 6. CustomerA IL 코드

6. 결론

본 논문에서는 C#으로 속성 가변성 구현 기법에 대해 제안했다. 가변적인 부분을 외부 파일에 기록해 클래스를 동적으로 생성하는 기법이다. 제시된 기법은 속성 추가와 변경의 재컴파일 작업을 요하지 않는다. 외부 파일에 가변성을 정의하고, Required 인터페이스를 통해 설정하면 된다. 그래서 여러 패밀리에 적용하기 편하다. 그리고 가변성을 위한 오버헤드가 적어서 성능에도 문제가 없다.

참고문헌

[1] Grahn G., "Transition from Conventional to Component-based Development", in Int'l Workshop on Component-Based Software Engineering, pp.78-82, 1999.

[2] Atkinson, C., Component-Based Product Line Engineering with UML, Addison Wesley, 2002.

[3] C# Language Specification, Microsoft., 2001.

[4] Metadata Unmanaged API, Microsoft, 2002.

[5] Thuan L., .NET Framework Essentials 2Ed, O'Reilly, 2002

[6] 소동섭, 신석규, 김수동, "컴포넌트 가변성 유형 및 Scope에 대한 정형적 모델", 한국정보과학회논문지 소프트웨어 및 응용, Vol. 30, No. 05, pp. 414-429, 2003년 6월.