Type Object Class에 의한 Type Object 디자인 패턴의 런타임 클래스 참조 문제의 해결

김 윤 호*

Resolving the Runtime Class Reference Problem of the Type Object Design Pattern by Type Object Class

Yun-Ho Kim*

요 약

Type Object 디자인 패턴은 하나의 클래스가 수많은 하위 클래스를 갖거나, 그 하위 클래스의 개수를 소프트웨어 개발시에 예측할 수 없는 상황을 해결하고자 제시된 패턴이다. 그러나, 이 패턴은 적용력과 여러장점에도 불구하고 인스턴스를 생성하는 클래스와 그 인스턴스의 실제적 클래스가 분리되어 있고 또한 객체 레퍼런스에 의해서 서로 연관되어 있으므로 이에 대한 관리를 위한 메커니즘과 패턴의 이해에 있어서 많은 복잡성을 갖는 문제점이 있다. 본 논문에서는 이러한 문제를 해결하기 위한 Type Object Class의 설계와 구현을 제시한다. 즉, Type Object 패턴의 Type Class와 Object Class로부터 Type Object Class를 설정하고, 이를 런타임에 생성되고 사용되게 함으로써, 인스턴스들이 객체 지향 프로그래밍 언어에서 제공하는 고유의 클래스를 참조하게 되어 별도의 클래스 참조 메커니즘을 가질 필요가 없도록 하였다. 따라서, 별도의 클래스 참조 메커니즘을 개발하는 부담과 이 메커니즘의 동작으로 인한 실행 상의 성능 저하의 문제가 개선되는 효과가 있다.

ABSTRACT

The Type Object Design Pattern is proposed to provide a solution on the situation in the case that one class has too many subclasses or the number of subclasses are undefined. Although this pattern has many advantages in terms of applicability and dynamic object behavior, it has a weak point in runtime pattern operation that it has to build and maintain a class reference mechanism in runtime to reference the class (de facto 'object') of instances. To solve that problem, this paper addresses the solution of the runtime class reference problem of Type Object Pattern. it defines a new class of Type Object Class (TOC) from Type Class and Object Class in Type Object pattern and presents the methods of creating, compiling, and memory-loading the TOC. It depends on built-in class reference mechanism of object-oriented programming language, and is not necessary to fit with the additional mechanism. Consequently, we need not to set up the additional class reference mechanism and system performance is enhanced due to it.

키워드

Software Design Pattern, Object-Oriented Design, Object-Oriented Sofware System, Java Programming Language

* 안동대학교 전자정보산업학부 부교수

접수일자 : 2006. 9. 26

Ⅰ. 서 론

오늘날 보다 짧은 기간 내에, 보다 신뢰성 높은 소프트웨어 개발에 대한 요구는 나날이 증대되고 있다. 이러한 요구에 대한 해결책으로서 제시된 객체 지향 소프트웨어 설계에서 주요한 요소 중의 하나가 디자인 패턴이다 [1]. 이러한 디자인 패턴은 소프트웨어 설계 시에 빈번히 제기되는 특정한 상황에 대한 검증된 해결책이기도 하다 [2]. 따라서 적절한 디자인 패턴의 적용은 소프트웨어 개발 기간의 단축과 신뢰성의 제고에 기여하게 된다. 은행 업무, 상품 관리, 물품 제조 등 많은 애플리케이션 도메인에서 빈번히 제기되는 문제를 해결하기 위한 디자인 패턴으로서 Type Object 디자인 패턴이 제시되었다 [3]. 즉, 하나의 클래스가 수많은 하위 클래스를 갖거나, 그 하위 클래스의 개수를 소프트웨어 개발시에 예측할 수 없는 상황을 해결하고자 제시된 패턴이다.

그러나, 이 패턴은 적용력과 여러 장점에도 불구하고 인스턴스를 생성하는 클래스와 그 인스턴스의 실제적 클래스가 분리되어 있고 또한 참조에 의해서 서로 연관되어 있으므로 이에 대한 관리와 이해에 있어서 많은 복잡성을 갖는 문제점이 있다. 따라서, 본 논문에서는 Type Object 패턴에서의 Type Class와 Object Class로부터 TypeObjectClass를 설정하여, 런타임에서 이 클래스로부터 생성되는 인스턴스를 사용하도록 함으로써 런타임에서 클래스 참조의 문제를 해결하는 방법을 제시한다. 즉, 동적 코드 생성 기법 [4,5]을 사용하여 설정된 TypeObjectClass에 대한 소스파일을 생성하고, 컴파일 및 메모리에 적재하여 런타임에서 이 클래스의 사용이 가능하게 한다.

본 논문에서 제시하는 방식을 이용하면, 런타임에서의 클래스 참조는 객체 지향 언어에서 제공하는 클래스 참조 메커니즘에 의거하여 객체들이 동작한다. 따라서, Type Object 패턴에서와 같이 런타임에서 클래스 참조를 위한 추가적인 클래스 참조 루틴을 작성하지 않아도 되며, 추가된 클래스 참조 루틴의 동작으로 인한 실행시의 성능 저하의 문제도 해결할 수 있다. 또한 클래스와 인스턴스가 자연스럽게 대응되므로 패턴의 구조에 대한 이해가 용이하게 된는 효과가 있다.

Ⅱ. Type Object Pattern

이 절에서는 Johnson과 Woolf가 제안한 Type Object 디자인 패턴 [3]에 대한 내용을 고찰하고자 한다. Type Object 패턴은 클래스와 인스턴스를 분리하여 한 클래스의 인스턴스가 다른 클래스의 인스턴스들의 클래스의 역할을 하도록 구현하는 패턴이다. 이는 메타 클래스 [6], Object Types [7l], Power Type [8], MetaObject [9], Prototype [10] 등의 개념과 유사하다.

Type Object 패턴은 그림 1에서 보는 바와 같이 Object 객체들을 생성하기 위한 Object Class와 Object 객체들의 타입으로서의 Type Object 객체를 생성하기 위한 Type Class를 설정한다. 다음으로, Type Class의 인스턴스인 Type Object (그림에서 TypeObject1)가 Object Class의 인스턴스인 Object 객체들 (그림에서 Object1, Object2, Object3)의 클래스로서의 역할을 할 수 있게 하기 위하여, Object Class의 인스턴스들에 Type Object에 대한 레퍼런스 (그림에서 classRef 속성: 객체 참조 변수)를 갖게 하는 패턴이다.

이 패턴은 클래스 기반의 객체 지향 프로그래밍 언어가 기본적으로 런타임에서 동적으로 코드를 변경할 수 없는 제약을 극복하기 위한 한 해결책으로 제시된 것이다. 즉, 런타임에서 객체는 자유로이 생성이 가능한 성질을 이용하여 런타임에 동적으로 Type Object객체를 생성하고, Object 객체들의 참조에 의해 클래스의 역할을 하도록 한 것이 Type Object 패턴의 핵심이다.

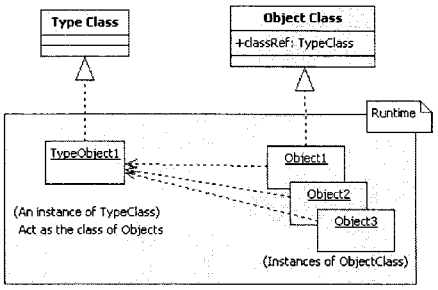

그림 1. Type Object Pattern 의 구조
Fig. 1. Structure of the Type Object pattern

그러므로, Type Object 패턴은 런타임에 생성된 Type Object 객체가 다른 객체들의 클래스의 역할을 할 수 있

도록 하기 위하여, 자체적인 클래스 타입 참조를 위한 메커니즘을 직접 제작하여야 하고, 실시간에 이에 대한 관리가 적절히 될 수 있도록 제작해야 한다.

Type Object 패턴을 적용하는 절차는 다음과 같다 [3].

1. 한 클래스로부터 인스턴스들에 공통적인 부분 (속성, 메소드)과 개별적인 부분을 분리한다.
2. 공통적인 부분들에 대하여는 Type Class로 설정하고, 공통적이지 않은 부분에 대하여는 Object을 생성하는 Object Class로 설정한다.
3. Object Class의 속성으로서 Type Class 클래스에 대한 레퍼런스를 가진 속성을 설정한다. 이것은 Object Class와 Type Class를 연결(참조)하는 역할을 한다.

실제로 Object 객체의 Type 즉, Object 객체의 가상적인 클래스로서 Type Object 객체의 역할이 수행되는 시점은 각각의 두 객체들이 모두 생성되는 런타임에서 이루어지도록 되어야 한다.

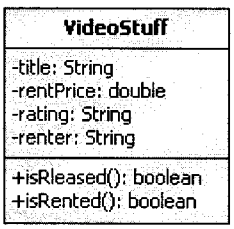

그림 2. 예제: VideoStuff Class
Fig. 2. Example: VideoStuff Class

그림 2에서 보인 VideoStuff 클래스를 예로하여 Type Object 패턴을 적용하는 과정을 보이면 다음과 같다. 그림2에서 보는 바와 같이 VideoStuff 클래스의 속성으로는 비디오 테이프의 영화 제목을 나타내는 title, 대여료를 나타내는 rentPrice, 영화 등급을 나타내는 rating, 그리고 비디오 테이프를 빌려간 대여자의 이름을 나타내는 renter가 있다. 또한, 비디오 테이프의 출시여부를 알려주는 isReleased() 메소드와 대여 여부를 알려주는 isRented() 메소드가 있다. VideoStuff 클래스에서 title, rentPrice, rating 등은 모든 VideoStuff 인스턴스들 (비디오 테이프들)에서 공통되는 부분들이며, isRented()와 renter는 인스턴스들에서 구별되는 부분들 (개별적인 비디오 테이프들의 상태들)이다.

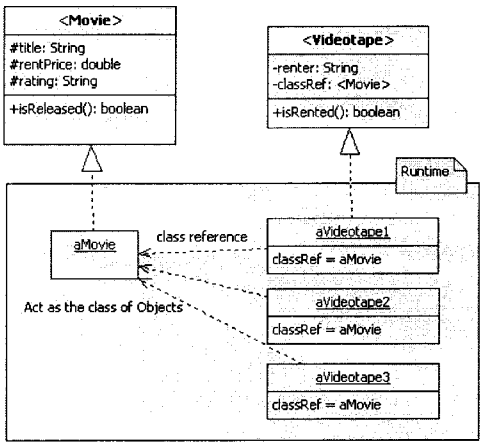

그림 3. VideoStuff에의 Type Object Pattern 적용
Fig. 3. Type Object pattern applied to VideoStuff

Type Object 패턴의 적용 과정은 그림 3에서 보는 바와 같이 공통되는 부분들을 가지는 Movie 클래스와 그렇지 않은 Videotape 클래스로 분리하여 클래스들을 설정한다. 즉, Movie 클래스에는 모든 비디오 테이프 인스턴스에서 공유되는 title, rentPrice, rating과 같은 속성과 isReleased() 메소드를 설정하고, Videotape 클래스에는 개별 비디오 테이프의 상태와 관련된 renter 속성과 isRented() 메소들을 설정한다. 또한, Videotape 클래스의 typeRef 속성에 Movie 클래스의 레퍼런스를 저장하여 참조할 수 있도록 함으로써, Videotape 클래스의 인스턴스가 Movie 클래스의 인스턴스를 Type Class로서 역할을 할 수 있도록 서로 연결시켜 준다. 실재로 Movie 객체가 Videotape 객체의 Type 즉, 가상적인 클래스로서 역할을 하는 것은 Movie 객체와 Videotape 객체가 생성되는 런타임에서 이루어지게 된다.

Ⅲ. Type Object Class의 설계 및 구현

본 절에서는 Type Object 디자인 패턴의 문제점을 해결하기 위한 Type Object Class의 설정과 그 구현에 대한 내용을 제시하고자 한다.

3.1. Type Object Class의 설계

Type Object 디자인 패턴은 Object 객체들을 생성하기

위한 Object Class와 Type Object 객체들을 생성하기 위한 Type Class를 설정하고, 런타임에서 Type Object 객체가 Object 객체들의 클래스 역할을 할 수 있도록 하는 것이 패턴의 핵심이다. 그러나, 이 패턴에서는 런타임에서 Type Object가 Object의 클래스 역할을 하도록 하기 위하여 Object에 속성으로서 Type Object에 대한 레퍼런스를 갖도록 한다. 따라서, 런타임에서 클래스에 대한 참조(속성, 메소드 등)에 대한 복잡한 처리를 직접 작성하고 관리 및 유지해 주어야 하는 것이 이 패턴의 가장 큰 문제점이다.

본 논문에서는 이 문제를 해결하기 위하여 Type Object 객체가 클래스 역할을 하도록 하는 것이 아니라, 런타임에서 실재적인 클래스인 Type Object Class를 생성하고 이 클래스에 의해 Object 객체가 생성되도록 함으로써 레퍼런스 설정과 관리의 문제를 해결하는 방법을 보이고자 한다. 런타임에서 실재 클래스를 생성하기 위하여 활성 코드 생성 기법 [4,5]을 이용한다. 런타임에서 Type Object Class가 생성되고 사용될 수 있도록 하는 절차를 간단히 요약하면 다음과 같다.

1. Type Class와 Object Class로부터 Type Object Class를 정의한다.
2. 런타임에서 Type Objec Class에 대한 소스 프로그램이 생성되게 한다.
3. 런타임에서 소스 프로그램을 컴파일되게 한다.
4. 런 타임에서 컴파일된 클래스를 로딩되게 한다.
5. 런타임에서 Type Object Class의 객체가 생성되게 한다.

Type Object 패턴에서 Object 객체의 클래스 역할을 하는 Type Object 클래스의 기능을 가지면서 Object 객체를 생성하는 기능을 가진 클래스를 갖도록 하기 위하여 본 논문에서는 Type Object Class를 설정한다. 이 클래스는 런타임에서 생성되고 로딩되어 이미 정의되어 있는 다른 클래스들처럼 동작하게 된다. 따라서, Type Object Class는 Type Object의 클래스 기능을 가져야 하므로 Type Object클래스의 모든 속성과 메소드를 가져야 한다. 특히, Type Object 패턴에서는 Type Object클래스의 속성들의 값이 모든 Object 객체들에서 불변의 값으로 공유하는 것이므로 static 변수들로 선언되어야 한다.

또한, Type Object Class는 Object 객체를 생성하여야 하므로 Object Class의 모든 속성과 메소드를 포함하여야 하며, Type Object 클래스에 있는 메소드는 static 변수를 이용하므로, static 메소드로 선언한다. Type Object Class를 설정하게 되면, Type Object에 대한 레퍼런스를 가지는 속성과 레퍼런스 관리 및 유지를 위한 속성 및 메소드는 필요가 없게 되므로 Type Object Class의 정의에서 제외시킨다.

결과적으로, Type Class의 속성들과 메소드들을 TcAttributes, TcMethods라 하고, Objec tClass의 속성들과 메소드들을 OcAttributes, OcMethods라 하면 Type Object Class의 정의는 이 클래스의 속성들 (TocAttributes)과 메소드들 (TocMethods)로 다음과 같이 표현된다.

[Type Object Class 정의 규칙]

- TocAttributes = (static TcAttributes) + OcAttritues
- TocMethods = (static TcMethods) + OcMethods
- TypeObjectClass = TocAttributes + TocMethods

2절에서 다룬 예제에 대하여 Type Object Class 정의 규칙을 적용한 결과를 보이면 아래의 그림 4와 같다. 여기서, 속성과 메소드에서의 밑줄은 static 변수 또는 메소드들을 나타낸다.

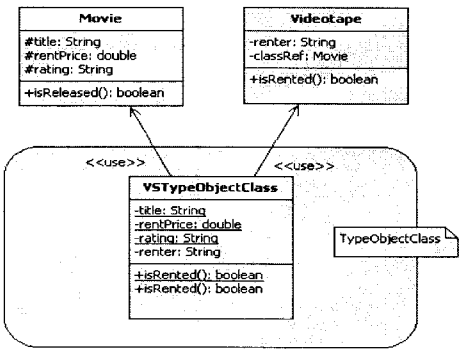

그림 4. Type Object Class 설정의 예
Fig. 4 An Example of TypeObjectClass Definition

그림4에서 보는 바와 같이TocAttributes는 TcAttributes (= title + rentPrice + rating)과 OcAttributes (= renter)로부터 다음과 같이 설정된다.

- TocAttributes = static (title + rentPrice + rating) + renter.

TocMethods는 TcMethods (= isReleased())와 OcMehods (= isRented())로부터 다음과 같이 설정된다.

· TocMethods = static isRelease() + isRented()

Videotape클래스에서 Movie 클래스에 대한 레퍼런스를 가진 classRef 속성은 Type Object Class의 설정으로 Movie 클래스 참조를 위한 클래스 참조값의 유지가 불필요하므로 제거한다.

3.2. Type Object Class의 구현

런타임에서 Type ObjectC lass의 클래스가 생성되게 하려면 먼저 클래스 소스파일이 만들어져야 한다. 따라서, 정의된 Type Object Class의 내용을 담은 소스 파일이 런타임에서 파일로 저장이 되게 해야 한다. 다음은 런타임에서 이 클래스에 대한 소스 파일을 생성하는 buildTOCSource() 메소드를 보인 것이다. 본 논문에서는 구현을 위한 언어로 Java 프로그래밍 언어를 사용한다.

```
void buildTOCSource(File fileName,
  String className) throws IOException {
 FileOutputStream out =
    new FileOutputString(fileName);
 String classHead =
    "public class " + className + " {";
 String attributes = getTocAttributes();
 String methods = getTocMethods();
 String classEnd = " }";
 String TocSrc =
    classHead + attributes +
    methods + classEnd;
 out.write(TocSrc.getBytes());
 out.flush();out.close();
}
```

런타임에서 생성된 자바 클래스 파일에 대한 컴파일은 Java 언어의 Runtime 클래스에서 제공하는 서비스를 이용하여 이루어진다. 즉, exec()의 매개변수로 자바 컴파일러와 컴파일할 소스파일을 넘겨주어 런타임에서 소스 파일이 컴파일되게 할 수 있다. 다음은 소스 파일을 컴파일하는 compileTOCSource() 메소드를 보인다.

```
void compileTOCSource(File fileName)
    throws IOException {
 String compiler = "javac";
 String classpath = ".";
 String[] cmd = {compiler, "-cp", classpath,
  FileName.getName()};
 Process process =
  Runtime.getRuntime().exec(cmd);
}
```

위에서 보인 compileTOCSource() 메소드에 의해 Type Object Class의 클래스 파일이 생성되게 된다. 이 클래스를 런타임에서 사용할 수 있게 하려면 생성된 클래스를 메모리에 로딩하여야 한다. 클래스의 로딩은 자바의 Reflection [10]을 이용하여 구현할 수 있다. 다음은 생성된 클래스를 로딩하는 loadTOCClass() 메소드를 보인다.

```
TOCType loadTOCClass(String className)
    throws IOException {
 File classFile = new File(className + ".class");
 byte[] buf =
       new byte[(int) classFile.length()];
 FileInputStream fin =
       new FileInputStream(classFile);
 fin.read(buf); in.close();
 Class toc =
  defineClass(className,buf,0,buf.length);
}
```

이상에서 보인 바와 같이 TypeObjectClass는 buildTOCSource(), compileTOCSource(), loadTOCClass() 메소드를 순차적으로 실행함으로써 런타임에서 사용이 가능하게 된다. 다음은 위의 과정을 총괄하는 createTOC() 메소드의 내용을 보인다.

```
public TOCType createTOC(File fileName,
          String className) {
 TOCType tocObj;
 try {
  buildTOCSource(fileName, className);
  compileTOCSource(fileName);
  tocobj = loadTOCClass(className);
```

```
    }
    catch (IOException e) {}
    return tocObj;
  }
```

이상에서 기술한 바와 같은 Type Object Class를 정의 및 소스 생성, 컴파일, 메모리에 로딩 등의 일을 TypeObjectClassCreator 클래스에서 맡도록 하였으며, 이 클래스의 내용을 간략하게 클래스 다이어그램으로 도식하면 그림 5와 같다.

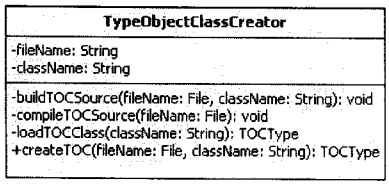

그림 5. TypeObjectClassCreator 클래스도
Fig. 5 TypeObjectClassCreator Class Diagram

그림 5에서 createTOC() 메소드는 소스파일을 생성하고 컴파일시킨 뒤 런타임에서 클래스가 사용될 수 있도록 클래스를 메모리에 로딩하는 전 과정을 순차적으로 수행하는 역할을 한다. 이 과정은 순차적이며 일괄적으로 수행되어야 하므로 하나의 통합적 성격의 메소드로 제공하는 것이 바람직하며, 필요시 자유로이 호출될 수 있도록 public 접근성을 갖도록 선언한다. createTOC() 메소드 내부에서 사용하는 buildTOCSource(), compileTOCSource(), loadTOCClass() 메소드들은 정보 은닉 (Information Hiding)을 제고하기 위해 private로 선언한다.

다음은 TypeObject 패턴에 본 논문에서 제시하는 Type Object Class를 적용한 패턴 구조를 도식한 것이다.

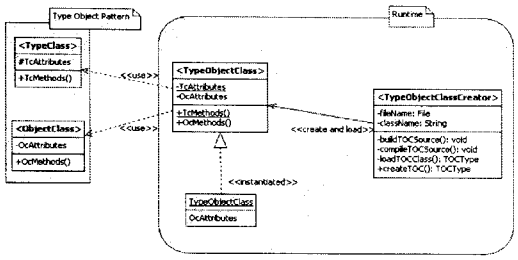

그림 6. TypeObjectClass를 적용한 Type Object 패턴
Fig. 6 Type Object Pattern with Type ObjectClass

그림 6에서 보는 바와 같이 Type Object 패턴에서의 클래스들인 Object Class와 Type Class로부터 Type Object Class을 정의하고, 이 정보에 기반하여 TypeObjectClassMaker는 클래스 Type Object Class를 런타임에서 생성하여 메모리에 로딩시키는 역할을 한다. Type Object 패턴을 사용하는 시스템에서 Type Object Class의 인스턴스를 생성하여 동작하도록 함으로써 Type Object 패턴에서의 런타임시에 객체 클래스를 클래스로 참조하기 위한 메커니즘이 필요가 없게 된다.

VI. 결 론

Type Object 패턴은 하나의 클래스가 수많은 하위 클래스를 갖거나 그 하위 클래스의 개수를 프로그램 제작시에는 예측할 수 없는 상황을 개선하기 위하여 제안된 패턴이다. 이 패턴은 적용력과 런타임에서 동적인 클래스와 객체 사이의 동작에 유연성을 가지는 반면, 실제로는 객체인 클래스를 참조하게 하는 문제가 있다. 즉, 클래스 참조를 위한 메커니즘을 이 패턴을 사용하는 시스템의 제작시에 자체적으로 개발하여 포함시켜야 하며, 이로 인한 코딩상의 복잡함과 더불어 실행시에 속도가 저하되어 성능이 떨어지게 된다.

따라서, 본 논문에서는 Type Object 패턴에서의 Type Class와 Object Class를 통합하여 Type Object Class를 설정하고 이 클래스를 런타임에서 소스 생성, 컴파일, 메모리 적재가 이루어지게 함으로써 런타임에서 Type Object Class의 인스턴스를 생성하여 사용하게 하였다. 이렇게 함으로써, 런타임에서 클래스 참조는 객체지향 프로그래밍언어에서 제공하는 고유한 메커니즘에 의해 이루어지므로, 이를 위한 별도의 메커니즘이 필요가 없게 된다.

결과적으로, 시스템 제작시에 클래스 참조와 관련된 메커니즘의 자체 보유를 위한 코딩의 부담이 줄어드는 효과와 동시에 실행시의 자체적 클래스 참조 메커니즘의 동작으로 인한 속도 저하의 문제도 해결되는 이점을 얻게 된다. 향후 Type Object Class의 설정을 편리하게 하기 위하여 사용자 편의성을 고려한 GUI 형태의 클래스 설정 인터페이스의 개발이 이루어지면 프로그램 제작시 뿐만 아니라 시스템 동작 중에도 TOC를 설정하여 사용할 수 있도록 하는 기능이 추가될 수 있을 것으로 기대된다.

참고문헌

[1] Coad, P., "Object-Oriented Patterns," CACM, vol. 35, no. 9, 1992.

[2] Gamma, Richard, Ralph, and Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[3] Johnson and Woolf, "Type Object," *Pattern Languages of Programming*, vol. *3*, pp.47-65, Addison-Wesley 1997.

[4] Hunter and Thomas, *The Pragmatic Programmer*, Addison-Wesley, 2000.

[5] Rettig and Fowler, "Reflection vs. Code Generator," *JavaWorld Magazine,* vol. 6, no. 11, 2001.

[6] Rumbaugh, Jacboson, and Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[7] Odell, J., "Object Types as Objects and Vice Versa," *Journal of Object-Oriented Programming*, vol.4, Issue 6, 45-48, 1992.

[8] Martin and Odell, *Object-Oriented Methods: A Foundation*, Prentice-Hall, 1998.

[9] Kiczales, Rivieres, and Bobrow, *The Art of the Metaobject Protocol*, MIT press, 1991.

[10] James Noble, "Prototype-Based Object System," *Pattern Languages of Programming*, vol. 4, 53-71, 2000.

저자소개

김 윤 호(Yun-Ho Kim)

1983. 2. 경북대학교 전자공학과 공학사
1993. 2. 경북대학교 컴퓨터공학과 공학석사
1997. 2. 경북대학교 컴퓨터공학과 공학박사
1997. 8. - 현재. 국립안동대학교 공과대학 전자정보산업학부 부교수
※관심분야 : 인터넷 컴퓨팅, OOA/D/P