특집

객체와 컴포넌트, 그리고 프레임웍

김 수 동[1]

목 차

1. 서 론

소프트웨어 품질과 개발 생산성 향상을 목표로 한 소프트웨어 공학의 역사가 40년에 이르고 있으나, 정보 산업 분야에서는 보다 효과적이며 실효성 있는 소프트웨어 개발 기술의 요구가 높아지고 있다. 소프트웨어 재사용 기술은 하드웨어, 건설, 기계 분야처럼 범용 부품을 개발하고 이를 재사용한 소프트웨어의 조립 생산을 목표로 하고 있다. 본 논문에서는 대표적인 재사용 기술로 80년대의 객체지향 프로그래밍, 90년대의 컴포넌트 기반 개발 (CBD) 기술, 2000년대의 프레임웍을 이용한 Product Line 공학에 대한 기술적 고찰을 한다. 각 기술의 핵심 구성 요소를 살펴보고 재사용을 효율적으로 지원하는지 장치들을 분석해 본다.

2. 구조적 개발 방식에서 객체지향 방식으로

70년대 까지 대표적인 개발 방식으로 알려진 절차적 프로그래밍은 COBOL, C등의 고급 언어를 사용하여 목표 시스템을 기능과 정보 저장소를 분리하여 개발하는 방식이다. 이와 함께 구조적 분석 기법이 널리 사용되어 왔으나, 소프트웨어 과제의 복잡화와 대형화로 이 기법의 한계성이 알려졌고, 이를 크게 보완한 새로운 개발 방식으로 객체지향 프로그래밍이 80년대부터 사용되어 왔다.

2.1 OOP 구성 요소

객체지향 프로그래밍은 객체가 실행의 기본이며 유일한 단위이다. 여러 객체들간 메시지를 통해 서비스를 지원하며 프로그램이 동작한다. 객체는 데이터와 이 데이터를 처리하는 함수들을 가진 작은 소프트웨어 모듈이다. 즉, 객체는 데이터와 함수 두 가지로 구성되는데 데이터는 그 객체의 상태를 나타내며, 함수는 그 객체가 수행할 수 있는 기능을 정의한다. 객체의 특징에는 캡슐화와 정보은폐가 있다. 캡슐화는 필요한 데이터와 함수를 하나의 단위로 묶는 것을 말한다[1].

객체는 내부의 데이터를 숨기고, 객체간에 허가된 메시지 교환을 통하여 정보를 교환한다. 따라서 각 객체는 다른 객체가 어떤 메시지를 보낼 수 있는가를 정의해야 하는데 이를 공용 인터페이스 (Public Interface)라 한다. 그러나 외부에 제공

1) 숭실대학교 컴퓨터학부 부교수

할 기능들이 어떤 알고리즘으로 구현된 것인지, 어떻게 수행되는지의 부분은 블랙박스로 노출시키지 않는다. 이것은 공용 인터페이스를 사용하지 않고, 데이터 부분의 변수들이 외부 객체나 프로그램에 의해서 직접 검색되거나 임의로 값이 수정되는 것을 피하기 위해서다.

클래스(Class)란 유사한 여러 객체들에게 공통적으로 필요로 하는 데이터와 이 데이터 위에서 수행되는 함수들을 정의하는 소프트웨어 단위이다. 모든 사물에 어떤 타입이나 종류가 있듯이 객체에도 종류가 있다. 예를 들면 한 회사에 수많은 직원들이 근무한다면 이들 모두가 '직원'이라는 타입에 속하며 직원으로서의 공통점을 가지고 있다. 이러한 객체의 집합을 소프트웨어적으로 표현하여 구현한 것을 클래스라 부른다.

상속은 새로운 클래스를 정의할 때, 처음부터 모든 것을 다 만들지 않고 기존의 클래스들의 특성을 상속 받아 필요한 특성만 추가하는 경제적인 방법이다. 여기서 특성이란 그 클래스가 가지고 있는 속성과 기능을 의미한다. 이렇게 생성된 서브 클래스는 슈퍼 클래스에 정의된 모든 변수들을 가지며 슈퍼 클래스의 함수(기능)들을 상속 받아 수행한다. 이외의 구성요소로 추상(Abstract) 클래스와 제네릭(Generic) 클래스가 있다.

2.2 OOP의 재사용 장치

객체는 절차적 프로그래밍과는 달리, 한 객체가 필요한 데이터와 기능을 한 단위로 가지며, 재사용에 적합한 기본 단위가 된다. 특히, 내부의 정보를 감추고 인터페이스를 통한 주변 객체와의 결합(Coupling)을 낮추어 객체지향 소프트웨어의 모듈성을 향상시킨다.

상속은 새로운 클래스를 정의할 때, 처음부터 모든 것을 다 정의하지 않고 기존의 클래스들의 속성을 상속 받아 추가로 필요한 속성만 확장하여 재사용하는 방법이다. (그림 1)은 Person 객체를 재사용하여 Student 객체를 만드는 Java 프로그램으로 Student는 Person의 모든 속성과 기능을 재사용하게 된다.

```
public class Student extends Person {
        private String grade;
        private String major;
        public String getGrade(){... };
}
```

(그림 1) 상속을 통한 수퍼 클래스의 재사용

추상 클래스는 그 서브 클래스들이 공통으로 가지는 속성과 함수들을 정의하고, 추가할 부분을 선언하기 위해 사용된다. 슈퍼 클래스에서 모든 함수들을 제공하는 것이 아니라, 서브클래스에서 필요한 구현을 하도록 강요할 수 있다. 따라서, 서브클래스는 추상 클래스에 정의된 속성과 함수 뿐 아니라, 인터페이스도 재사용하여 일관성 있는 프로그램 작성이 가능해 진다.

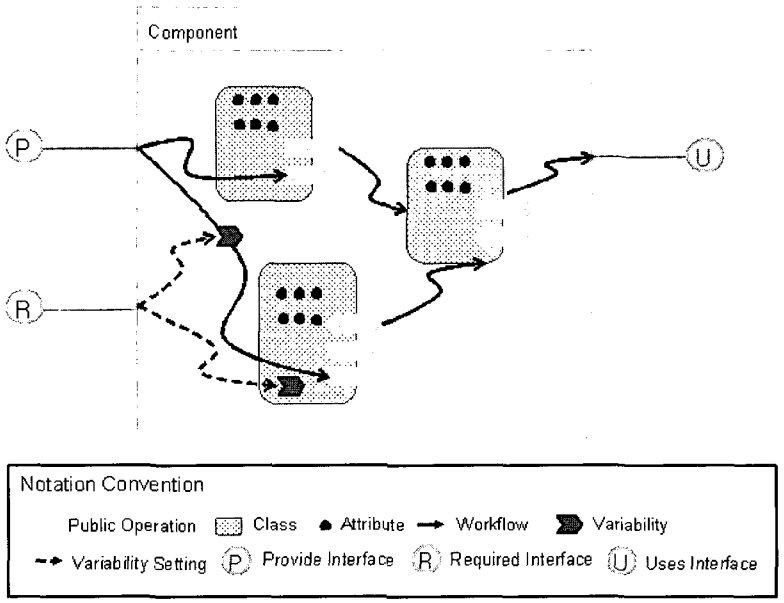
제네릭 클래스를 통해 데이터 타입을 미리 정하지 않고 제네릭 클래스를 만든 후, 필요한 데이터 타입만 지정하여 일반 클래스를 생성할 수 있어 객체지향의 재사용성을 높인다.

3. 객체에서 컴포넌트로

객체지향 기법의 보편화로 재사용에 대한 실효성과 기대감이 높아지고 있다. 객체보다 더 크고 효율적인 재사용 단위를 요구하게 되었고, 한 어플리케이션이 아니라 한 도메인에서 공통으로 사용할 수 있는 재사용 장치의 필요성이 컴포넌트 기반 개발 (CBD) 기술의 동기가 된다[2].

3.1 컴포넌트의 구성 요소

CBD 의 컴포넌트는 (그림 2) 처럼 관련된 여러



(그림 2) 컴포넌트의 구성 요소

클래스들로 구성된다. 대부분의 객체는 실행시 주변 객체와 메시지 교환을 통하여 시스템 기능(System Operation)을 수행하게 된다. 컴포넌트는 이런한 관련된 객체, 즉 클래스들을 미리 모델링하여 한 컴포넌트에 포함하게 된다.

컴포넌트가 제공하는 서비스나 기능은 인터페이스를 통하여 제공된다. 즉, 컴포넌트 외부에서는 인터페이스를 통하여 컴포넌트의 기능을 사용하며 메세지를 주고 받는다. 컴포넌트는 클래스의 인터페이스를 직접 사용할 수 있는 Facet도 제공하지만 컴포넌트 수준의 인터페이스인 Provide, Required 및 Uses 인터페이스를 제공한다. Provide 인터페이스는 컴포넌트가 가지고 있는 기능을 제공하기 위한 인터페이스로 외부의 사용자나 객체에서 이 인터페이스를 통하여 컴포넌트가 제공하는 서비스를 이용할 수가 있다. Required 인터페이스는 컴포넌트의 가변성을 설정하는 인터페이스다. Uses 인터페이스는 컴포넌트가 외부에 갖는 의존성을 표시하며 컴포넌트는 Uses 인터페이스를 통해 외부에 메시지를 보내고 외부의 기능을 사용한다. 객체는 인터페이스와 객체 구현이 하나의 단위이지만 컴포넌트는 이 둘을 분리하므로, 한 컴포넌트에 다른 인터페이스를 연결할 수 있는 유연성이 있다.

컴포넌트는 한 도메인 안의 여러 어플리케이션이 필요로 하는 공통 기능들을 모델링하며 이를 구현한다. 즉, 도메인의 표준이거나 공통적인 업무 로직, 데이터 및 워크플로를 제공한다. 고급 컴포넌트는 가변성을 제공하여 보다 많은 어플리케이션에서의 재사용을 도모한다. 가변성은 특정 어플리케이션에 맞게 컴포넌트가 가진 로직이나 워크플로우를 특화시킬 수 있는 장치이다.

컴포넌트의 인터페이스를 통해서 메시지가 전달되면, 컴포넌트는 그 메시지의 실행에 필요한 일련의 기능들을 수행하는데 이를 워크플로(Workflow)라 한다.

3.2 컴포넌트의 재사용 장치

컴포넌트는 관련된 여러 클래스들로 구성되므로 객체보다 더 큰 (Larger-Grained) 재사용 단위

를 제공한다. 객체지향에서는 관련된 클래스들을 개발자가 직접 찾아서 모델링해야 하지만, 컴포넌트는 이런 객체들의 식별과 클래스간의 관계가 이미 구현되어 있어서 재사용의 범위를 넓힌다.

컴포넌트는 한 도메인의 표준이거나 공통적인 기능을 제공하므로, 한 회사가 개발한 컴포넌트를 상용화하여 다른 회사가 사용할 수 있다. 이를 조직간(Inter-Organizational) 재사용이라고 하며 CBD 의 궁극적인 목표다[3].

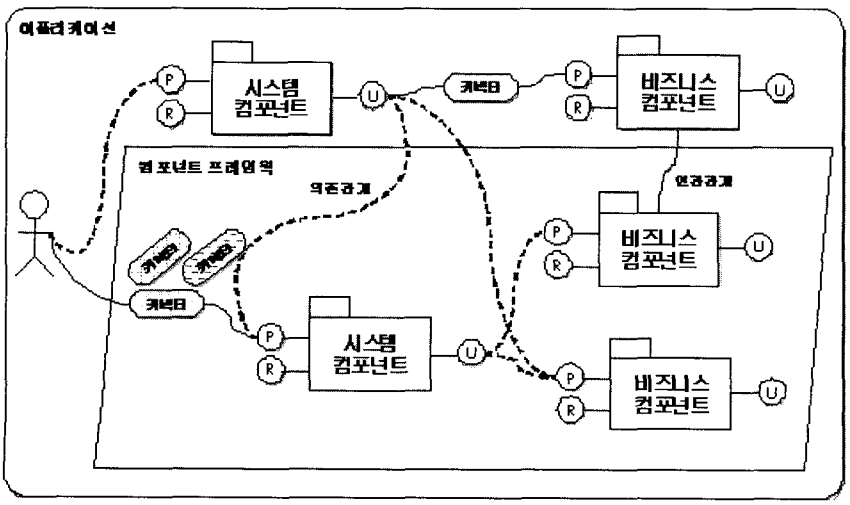
컴포넌트의 가변성 장치를 이용하면 컴포넌트 사용자 (Consumer)가 용도에 알맞게 기능을 특화할 수 있다. 객체는 고정적인 기능을 제공하지만, 컴포넌트는 가변적인 기능도 제공하므로 재사용성의 향상된다. 컴포넌트 내부의 가변성 구현 정도가 그 컴포넌트의 재사용성 및 적용성을 결정한다.

컴포넌트는 내부에 공통으로 사용할 워크플로를 가지고 있어서, 컴포넌트 사용자는 Sequence Diagram 등을 통한 메시지 흐름을 직접 구현할 필요가 없다. 즉, 컴포넌트는 워크플로의 재사용도 제공한다.

대부분의 컴포넌트는 소스 코드 형태가 아닌, 이진 코드 형태로 제공된다. 이는 사용자에게 컴포넌트 내부를 보지 않고 인터페이스 만을 통하여 컴포넌트 기능을 사용할 수 있게 하므로 보다 모듈성이 높은 소프트웨어 개발이 가능해 진다.

4. 컴포넌트에서 프레임웍으로

CBD컴포넌트 보다 더 효율적이고 큰 범위의 재사용성을 제공하는 프레임웍 기반의 Product Line 공학(PLE)이 각광을 받고 있다[4]. 컴포넌트는 전체 시스템에서 하나의 부품 역할을 하여 다른 컴포넌트와의 조립을 통해 기능성을 제공하게 되는데 컴포넌트가 다른 컴포넌트와 상호 작용할 때 불일치 문제가 발생하게 된다. 즉, PLE에서는 컴포넌트가 서로 잘 맞물려 작동되지 않는 것을 의미한다. 이러한 불일치의 문제를 해결하기 위해 도메인 별 패밀리 멤버간의 공통적인 특징(Feature)에 대해 어떠한 아키텍쳐를 기반으로 어떠한 컴포넌트가 존재하며 이들 사이의 상호 작용은 어떻게 이루어지는 지에 대한 기본 구조를 프레임웍(Framework) 형태로서 제공한다.



(그림 3) 컴포넌트 프레임웍의 구성요소

4.1 프레임웍의 구성 요소

프레임웍은 (그림 3)에서 처럼 아키텍쳐, 인터페이스, 컴포넌트, 컴포넌트간의 관계, 커넥터로 구성된다. 프레임웍을 기반으로 PLE 멤버에 종속적인 소수개의 컴포넌트나 객체들을 추가하여 어플리케이션을 만들게 된다.

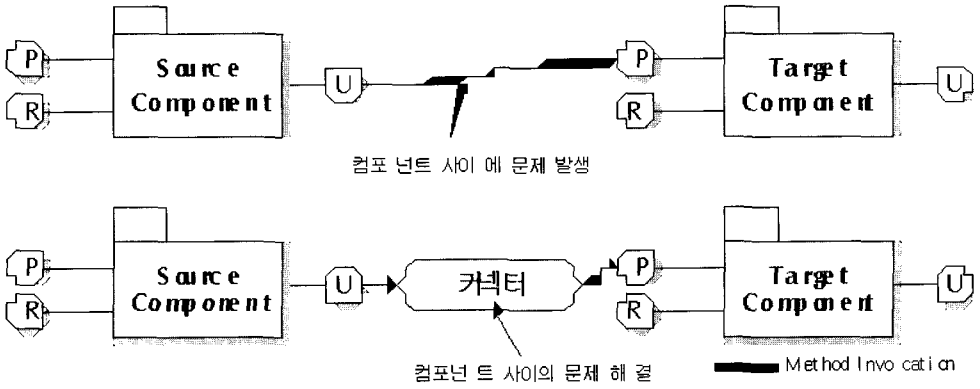
한 프레임웍은 어플리케이션이나 서브 시스템 개발에 필요한 여러 컴포넌트들을 모델링하여 이를 포함하고 있다. 따라서 컴포넌트 보다 더 큰 재사용 단위를 제공한다. 또한, 프레임웍은 Skeleton 아키텍쳐, 즉, 표준 범용 아키텍쳐를 가지고 있다. 컴포넌트 간의 관계에는 의존(Depends)과 연관(Associates)관계가 있다. 의존 관계는 컴포넌트들 사이의 메시지 호출 관계를 의미한다. 프레임웍 내에서 시스템 수준의 기능성을 제공하기 위해 여러 비즈니스 컴포넌트의 오퍼레이션을 사용하게 된다. 이처럼 컴포넌트 외부에 존재하는 Mediator 성격의 워크플로를 Macro 워크플로라 하며 의존 관계를 통해 나타난다. 연관 관계는 컴포넌트의 인스턴스들 사이의 영구적인 링크 관계를 의미한다. 의존관계는 컴포넌트들 사이의 약한 결합도를 나타내는 반면, 연관관계를 강한 결합도를 의미한다. 컴포넌트 간의 관계는 커넥터와는 달리 추가적인 기능을 제공하지 않는다.

컴포넌트들간의 상호작용은 컴포넌트가 가지고 있는 인터페이스를 사용한다. 커넥터는 호출하는 소스 컴포넌트의 Uses 인터페이스와 서비스를 제공하는 타켓 컴포넌트의 Provide 인터페이스 사이에 존재하게 된다. 커넥터는 컴포넌트들 사이에서 상호작용을 도와준다. (그림 4)와 같이 컴포넌트의 인터페이스 모양이나 그 의미가 변경된다면 다른 컴포넌트를 변경해야 하지만, 커넥터를 사용해서, 그 관계를 올바르게 연결 해준다. 커넥터는 이러한 역할을 수행하기 위해 종속적인 제약사항을 가지고 있으며, 컴포넌트들을 연결하기 위한 행위를 구현한다.

커넥터를 사용한 컴포넌트 프레임웍의 장점은 컴포넌트들 사이의 의존관계를 낮추어, 컴포넌트 변경시 컴포넌트의 재수정이 아닌 커넥터만 교체하면 된다. 상호작용을 모델링 하기 위해 커넥터를 사용하는 것은 대체성이 좋은 컴포넌트를 만들기 위한 컴포넌트 프레임웍 설계를 위한 강력한 기술이 된다.

4.2 프레임웍의 재사용 장치

프레임웍에 정의된 Skeleton아키텍쳐는 향후 여러 어플리케이션 개발에 재사용될 수 있다. 프레임웍은 여러 공통 어플리케이션에 적용되기 때



(그림 4) 커넥터를 사용한 상호작용

문에 프레임웍에서 대상 도메인에 적합하게 분류한 수직적인 계층은 대상 어플리케이션에 그대로 재사용될 수 있다. 특히 수평적인 계층은 기능적인 분류를 나타내기 때문에 대상 도메인에 대해 합당하게 분류된 Subsystem들은 여러 어플리케이션에서 유용하게 사용될 수 있다. 프레임웍의 재사용을 통해 어플리케이션을 새로이 만들고자 할 때, 이미 정의된 큰 틀을 대상 어플리케이션에 그대로 적용하여 그 틀을 기반으로 어플리케이션을 구축해 나가기 때문에 시간적인 측면이나 비용적인 측면에서 많은 장점을 얻을 수 있다.

프레임웍은 동일 도메인내의 여러 어플리케이션들의 공통성을 기반으로 만들어지며 가변성 또한 제공하므로 여러 어플리케이션에서 재사용될 수 있다. 프레임웍은 그 안에 어플리케이션들이 공통적으로 포함하고 있어야 하는 전체적인 계층 구조와 계층별 컴포넌트들의 역할과 컴포넌트들 사이의 상호작용 규칙이 정의되어 있다. 프레임웍은 인터페이스와 컴포넌트에 대해 명세서 수준으로 정의하고 있는, 어플리케이션을 만들기 위한 기본 템플릿으로 생각하면 된다. 어플리케이션에 적용시에는 비어있는 자리(Hot Spot)에 실제적인 값들, 즉, 구현 레벨의 컴포넌트, 커넥터, 인터페이스 등으로 대체하여 보다 간편하게 개발을 할 수 있게 된다. 프레임웍 수준의 가변성인 Macro 워크플로우의 가변성 또한, 인스턴스화 할 때 특화하여 여러 어플리케이션에서의 재사용을 극대화 할 수 있다.

커넥터는 컴포넌트들 사이의 의존 관계를 낮추어 컴포넌트들의 변경이 있을 때 사이에서 완충 작용을 하여 다른 컴포넌트의 변경없이 사용할 수 있게 해준다. 그러므로 컴포넌트를 수정하는 수고없이 컴포넌트의 재사용을 높일 수 있다. 만약 컴포넌트 인터페이스의 시그네쳐 하나 정도는 쉽게 변경할 수 있다고 생각하지만, 이 컴포넌트가 직접 개발한 컴포넌트가 아닌 블랙박스 형식의 COTS 컴포넌트라면 결국, 이 컴포넌트는 쓸모없게 된다.

또한 컴포넌트에서는 재사용장치로 가변성을 제공하지만, 패밀리 멤버들 사이에 그 가변성 종류가 너무 많고, 비즈니스 룰에 맞지 않아서 컴포넌트 내에 가변성을 적용하지 않았을 경우 커넥터를 사용해서 컴포넌트 외부에서 가변성을 적용할 수 있다.

5. 결 론

본 고에서는 재사용을 통하여 소프트웨어 위기를 해결하기 위한 세 가지 기술을 고찰하였다. 객체지향은 절차적 프로그램에 비하여 근본적으로 다른 패러다임을 제공하여 재사용성을 높였고, 컴포넌트는 여러 객체들을 모은 단위로서 공통성, 가변성, 인터페이스의 분리 장치들을 사용하여 재사용성과 소프트웨어 품질을 향상 시켰다. PLE의 프레임웍은 관련된 컴포넌트들을 묶은 매우 큰 재사용 단위로서 아키텍쳐, 커넥터, 어플리케이션 생성 장치들을 가지고 있어서 높은 재사용 기술로 기대되고 있다. 각 기술들은 상호 연동을 가지고 있어, 이들 기술들의 적절한 활용을 통하여 소프트웨어 개발 및 유지보수 기술이 한층 더 높아질 수 있을 것으로 예상된다.

참고문헌

[1] 김수동, 실무자를 위한 소프트웨어 공학, 홍능과학출판사, 1999.

[2] Heineman, G. T., "Practice of Software Engineering," Component-Based Software Engineering, Addison Wesley, 2000.

[3] Kim, Soo Dong, "Lessons Learned from

a Nation-wide CBD Promotion Project", Communications of the ACM, Vol.45, Issue.10, pp. 83-87, Oct. 2002.

[4] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wust, J. and Zettel, J., Component-Based Product Line Engineering with UML, Addison Wesley, 2002.

저자약력

김 수 동

1984년 Truman State University, 전산학 학사

1988년 University of Iowa, 전산학 석사

1991년 University of Iowa, 전사학 박사

1991~1993년 한국통신 연구개발단 선임연구원

1994~1995년 현대전자 소프트웨어 연구소 책임연구원

1995.9- 현재 숭실대학교 컴퓨터학부 부교수

연구분야 : 객체지향 방법론, 컴포넌트 공학, Product Line 공학, 소프트웨어 아키텍쳐

이 메 일 : sdkim@comp.ssu.ac.kr