객체지향 객체 모델의 컴포넌트 모델 전환 지침

유영란*, 김수동*
숭실대학교 컴퓨터학과
e-mail : yuandyu@selab.soongsil.ac.kr

Instructions for Transition from OO Object Model to Component-Based Model

Young-Ran Yoo*, Soo-Dong Kim*
*Dept. of Computing, Soongsil University

요 약

소프트웨어의 재사용성을 높일 수 있는 기법으로 객체보다 더 큰 재사용 단위인 컴포넌트 기반의 개발에 학계와 업계의 관심이 집중되고 있다. 객체지향 방식으로 구현된 모델들은 정보 은폐과 캡슐화를 지원함으로서 응집도 높은 객체들의 집합으로 컴포넌트를 식별하는 작업이 자연스러운 장점이 있다. 그러나 객체가 다른 객체들과 관계와 상속 등으로 연결되는 반면에, 컴포넌트는 컴포넌트들 사이의 인터페이스 호출에 의한 의존도만 나타나며 기본적으로 상호 독립적이다. 따라서 객체지향 모델을 컴포넌트 기반의 모델로 전환 시, 기존의 관계와 상속들을 컴포넌트의 인터페이스로 추출하여 제거하는 작업이 요구된다. 본 논문에서는 객체지향의 객체 모델을 컴포넌트 기반의 객체 모델로 전환 시 예상되는 문제점들을 해결하기 위한 실무적인 지침들을 제안하고자 한다.

1. 서 론

객체가 재사용의 단위로는 너무 작다는 인식이 확산되면서, 비즈니스의 재사용 단위로 컴포넌트에 학계와 업계의 관심이 집중되고 있다. 컴포넌트는 공개된 인터페이스를 통하여 비즈니스 서비스를 제공하는 하나 이상의 객체로 이루어진 블랙박스 형태의 재사용 단위이다.

기존의 객체지향 방식으로 모델링 된 시스템을 컴포넌트 기반의 시스템으로 전환하기 위해서는 분석 모델로부터 컴포넌트를 추출하는 작업이 가장 중요한 작업으로 추가된다. 이 때 객체지향 방식의 모델을 그대로 사용할 경우, 적절한 컴포넌트 추출을 어렵게 만드는 문제점들이 발생하게 된다. 많은 하부 클래스들을 가지는 수퍼클래스의 상속 문제와 전역적으로 사용되는 클래스의 컴포넌트 할당 등이 그 예이다.

본 논문에서는 객체 모델의 경우, 순수 객체지향으로 분석 및 설계된 모델이 컴포넌트 기반의 모델로의 전환 시 발생하는 문제점들과 그 원인을 밝히고, 컴포넌트를 추출하기 위한 모델로 정제하기 위한 지침들을 제안하고자 한다.

본 논문의 2 장에서는 관련 연구로서 컴포넌트의 특징과 컴포넌트 기반의 개발 프로세스에 대해서 살펴본다. 3 장에서는 기존의 객체기반의 객체 모델을 컴포넌트 기반의 객체 모델로 전환 시 부딪힐 수 있는 문제점들을 정리하고, 4 장에서 제시된 문제점들에 대한 수정 지침들을 제안한다. 그리고 마지막으로 5 장에서 결론과 향후 연구 방향을 제시하고자 한다.

2. 관련 연구

2.1. 컴포넌트 개발 방법론

컴포넌트는 관점에 따라서 다양하게 정의되고 있다0. 현재 상용중인 컴포넌트로는 사용자 인터페이스와 관련된 그 중 비즈니스 컴포넌트는 자치적인 (Autonomous) 비즈니스 개념이나 비즈니스 프로세스의 소프트웨어 구현물이며 보다 큰 비즈니스 시스템의 재사용 요소로서의 개념을 표현하고, 구현하고, 전개하기에 필요한 소프트웨어 산출물로 구현되어 있다고 정의되고 있다0.

컴포넌트 개발 방법론(CBD)으로는 Sterling 사의 CBD96 표준과 ICON Computing 사의 Catalysis, HP 의 Fusion, Compuware 사의 UNIFACE 등이 발표되고 있으나 아직 학계나 업계에서도 연구 중인 단계이다 0000. 이들 컴포넌트 개발 방법론이 기존의 OMT 등과 같은 객체지향 방법론과 가장 다른 점은 컴포넌트 추출 과정의 요구라고 할 수 있다. 제안되고 있는 대부분의 컴포넌트 개발 방법론들은 객체지향 모델링을 기본으로 하며, UML 을 기본 모델링 도구로 사용하고 있다.

이들 방법론들은 상위 수준의 Use Case 나 개념 수준의 클래스들로부터 후보 컴포넌트를 추출하도록 제안하고 있으며 동일한 Use Case 나 클래스가 서로 다른 컴포넌트에 동시에 할당되는 경우를 배제하고 있다0000.

2.2. 객체 모델

객체 모델은 객체지향 개념의 등장과 함께 제안된 가장 일반화 된 모델로서 대상 시스템의 정적인 자료 구조를 표현한다. 객체 모델이 기존의 구조적 방법론의 개체관계도(Entity Relation Diagram)와의 가장 큰 차이점은 정보와 정보를 처리하는 기능을 결합시킨 캡슐화와 정보 은폐, 그리고 상속(Inheritance)을 통한 추상화 혹은 일반화 과정을 들 수 있다.

캡슐화(Encapsulation)는 필요한 자료와 함수를 하나의 단위로 묶어서 정의하는 것으로서 그 단위가 객체가 된다0정보 은폐(Information Hiding)는 객체 내의 자료를 외부로부터 숨긴 채, 공개된 메소드들을 통해서만 허가된 정보를 접근하도록 하는 기법이다0정보 은폐를 통하여 객체 내부의 자료에 대한 무결성을 확보할 수 있다.

추상화(Abstraction)란 관련 있는 정보들을 일반화시켜서 수퍼클래스로 추출 함으로서 확장성과 적용성, 재사용성을 확보하는 기법이다. 하위 객체가 상위 객체에 대해서 “Is_A” 혹은 “Kinds_Of” 관계를 가지며 동일한 기능은 상위 객체에서 한번만 구현해주고, 서브 클래스들에서는 서로 상이한 부분들에 대해서만 작업해주도록 함으로서 재사용성과 생산성을 향상시키는 장점을 제공한다.

3. 객체지향 객체 모델의 문제점

객체지향 개발 방법과 모델들의 여러 장점에 기반을 두어 재사용의 단위를 객체에서 컴포넌트로 확장한 컴포넌트 기반의 개발 방법론에서는 객체지향에서 사용했던 여러 모델들을 그대로 사용한다. 그러나 객체가 다른 객체들과 관계와 상속 등으로 연결되는 반면에, 컴포넌트는 컴포넌트들 사이의 인터페이스 호출에 의한 의존도만 나타나며 기본적으로 상호 독립적이다.

따라서 객체지향으로 분석된 모델들을 그대로 컴포넌트 기반에 적용하기엔 어려운 부분들이 있다. 그 경우의 하나가 상속이다. 상속을 통하여 자료들을 추상화 시키고, 객체들간의 결합을 유도하여 재사용을 강조하는 객체지향 모델은 컴포넌트들간의 독립성을 강조하는 컴포넌트 기반의 모델들과 잘 맞지 않는다. 이러한 문제점들을 상속의 종류와 그 특징들을 통해서 살펴보도록 한다.

3.1. 자료 상속

자료 상속이란, 수퍼클래스가 서브클래스가 가지는 자료들의 많은 부분을 가지며, 수퍼클래스의 재사용이 자료에 비중을 두어 이루어지는 경우이다. 서브클래스들을 묶어서 자료를 처리할 필요가 있을 경우 수퍼클래스 형태의 업캐스팅(Up-Casting)을 통해서 서브클래스를 일괄 처리하도록 한다. 서브 클래스의 중복되는 자료들을 일반화 하는 작업의 일환으로 수퍼클래스를 추출한 경우로서 그림 1과 같은 경우가 그 예이다.

자료 상속의 경우, 서브클래스들끼리의 자료의 유사성이 높은 관계로, 슈퍼 및 서브클래스들을 동일한 컴포넌트에 할당 함으로서, 상속관계도 컴포넌트 내부로 끌어들일 수 있다.

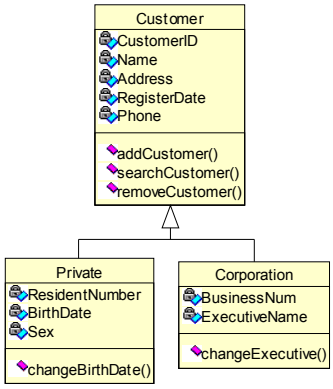

그림 1. 자료 상속의 예

3.2. 기능 상속

기능 상속이란, 수퍼클래스가 서브클래스가 수행하는 기능들을 대표하기 위해 추출된 경우이다. 서브클래스는 수퍼클래스의 기능들을 상속받아서 사용되며, 서브클래스들이 업캐스팅 되어 사용되는 사례가 거의 없다. 그림 2는 기능 상속의 예를 보여주고 있다.

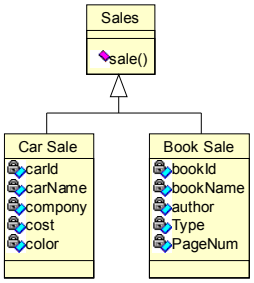

그림 2. 기능 상속의 예

자료 상속과 달리 기능 상속의 경우, 서브클래스들 간의 자료의 연관성이 적은 관계로, 서브클래스가 서로 다른 컴포넌트로 할당될 가능성이 존재하며, 그럴 경우 수퍼클래스와의 상속 관계는 컴포넌트들간의 결합도(Coupling)을 높이는 요인이 될 수 있다. 또한 수퍼클래스의 선언부를 어디에 할당하느냐에 따라 컴포넌트들간의 의존도도 영향을 받게 된다.

4. 컴포넌트 기반의 개발을 위한 객체 모델의 전환 지침

본 장에서는 컴포넌트 기반의 개발을 위하여 3 장에서 제시된 상속의 종류 중, 컴포넌트 추출에 영향을 미치는 기능 상속의 경우에 대하여, 그 유형들과 각 유형별 컴포넌트 할당 지침을 제시하고자 한다.

4.1. 순수 허상(Virtual) 기능 상속

수퍼클래스가 자료의 공유 없이 공통 기능만을 가지면서,그 구현 또한 서브클래스가 담당하는 경우로, 수퍼클래스는 인터페이스의 형태로 나타난다.

이 때는 먼저 서브클래스들을 중심으로 컴포넌트 추출을 하고, 서브클래스들이 서로 다른 컴포넌트에 할당되는 경우, 수퍼클래스를 없앤다. 이 수퍼클래스는 서로 다른 컴포넌트들의 인터페이스 형태로 표현될 수 있으며, 하부 클래스들간의 의존도가 약하므로 컴포넌트들간의 결합도도 낮아진다.

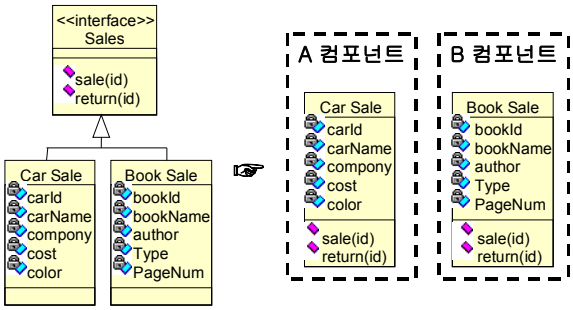

그림 3. 순수 허상 기능 상속의 예

4.2. 순수 실상(Concret) 기능 상속

수퍼클래스가 자료의 공유 없이 공통 기능만을 가지지만 구현을 포함하고 있으며, 수퍼클래스의 인스턴스 생성이 가능한 경우이다. 그러나 이 경우는 실제 모델로 나오기가 어렵다. 자체의 속성을 가지고 있지 않으면서 인스턴스도 가지지 않고 기능만을 가지는 경우, 내부 함수들이 C++의 MATH 와 같이 static 함수일 경우가 그 경우에 속한다.

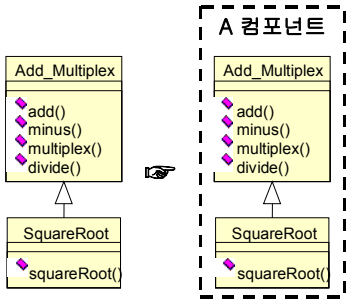

그림 4. 순수 실상 기능 상속

이 때는 수퍼클래스와 서브클래스가 동일한 컴포넌트에 할당할 수 있으며, 이 때 식별된 컴포넌트는 다른 컴포넌트들의 하부 계층에 위치하며 라이브러리와 유사한 형태의 기능들을 제공하게 된다.

4.3. 허상(Virtual) 기능 상속

수퍼클래스에 기능과 함께 일부 정보들을 포함하고 있으나, 수퍼클래스의 인스턴스 생성이 가능하지 않은 경우이다. 이 때, 수퍼클래스를 추상 클래스라고 한다.

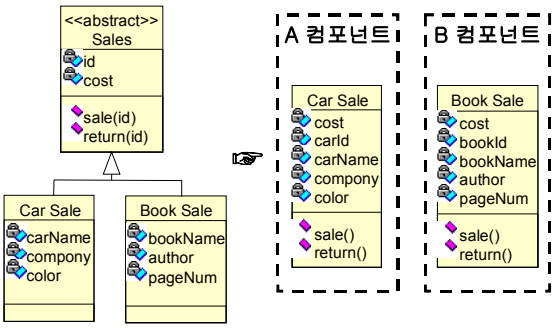

그림 5. 허상 기능 상속

컴포넌트 식별 단계 시, 상위의 허상 클래스는 제외하고 작업한다. 만약 서브클래스가 서로 다른 컴포넌트에 할당될 경우, 수퍼클래스를 제거하고, 수퍼클래스에 선언된 자료와 기능들을 서브클래스로 내린다. 서브클래스들이 동일한 컴포넌트에 할당될 경우, 수퍼클래스를 살린 채, 해당 컴포넌트에 할당시킨다.

4.4. 실상(Concret) 기능 상속

수퍼클래스가 일부 자료들을 공유하며, 인스턴스 생성 또한 가능한 경우이다. 이런 경우는 자료 상속과 혼동될 우려가 있는데 수퍼클래스의 추상화가 기능의 재사용에 맞추어져 있는지 자료의 재사용에 맞추어져 있는지를 판단하여 자료 상속인지 기능 상속인지를 결정하여야 한다.

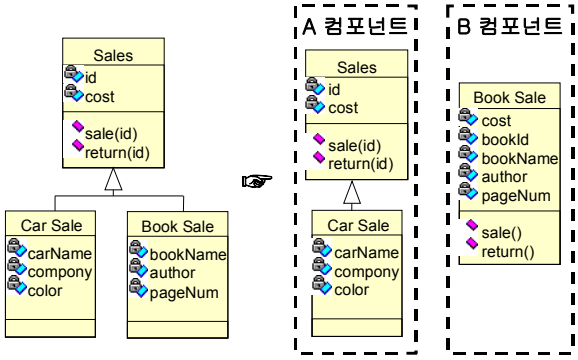

그림 6. 실상 기능 상속

먼저 상위의 실상 클래스를 포함하여 컴포넌트 식별 작업을 한 후, 서브클래스 일부가 수퍼클래스가 포함되지 않은 다른 컴포넌트로 할당된 경우, 수퍼클래스의 속성들과 기능을 내려받는다.

5. 결론 및 향후 연구 과제

본 논문에서는 객체지향 방식의 객체모델을 컴포너느 기반의 개발에 적용 시 예상되는 기능 상속에 관한 문제점들과 각 유형별 지침들을 제안했다. 이 지침들은 도메인의 성격에 따라 조금씩 조정이 될 수 있지만, 기본적으로 독립적인 컴포넌트를 추출하기 위하여 객체들간의 상속을 포함한 관계들을 조정할 필요가 있다.
향후에는 객체지향의 각 모델들을 검토하고, 컴포넌트 기반 모델로의 전환 시에 필요한 실무 지침들을 개발하고자 한다.

참고문헌

[1] Grady Booch, James Rumbaugh, and Ivar Jacobson, “The Unified Modeling Language User Guide”, Addison-Wesley, 1999.
[2] Kozaczynski, W. and Booch, G., “Component-Based Software Engineering”, IEEE Software, pp. 34-36, Sept./Oct. 1998.
[3] Sterling Software Inc., “The CBD Standard Version 2.1”, Sterling Software, July 1998.
[4] souza, D. F. and Wills, A. C., “Objects, Components, and Components with UML”, Addison-Wesley, 1998
[5] HP Company, “Engineering Process Summary: Fusion 2.0”, Hewlett-Packard Company, January 1998.
[6] Compuware Corp., “UNIFACE Development Methodology V7.2”, COMPUWARE Corp., 1998.
[7] 김수동, “실무자를 위한 소프트웨어 공학”, 에드텍, 1999.