기존 프로그램에서의 독립 컴포넌트의 추출

윤 석진[0] 신 규상
한국전자통신연구원
(sjyoon, gsshin)@etri.re.kr

Extracting Isolated Components from Legacy Object-Oriented Programs

Seok-Jin Yoon[0] Gyu-Sang Shin
Software Engineering Department, ETRI-Computer & Software Technology Laboratory

요 약

본 논문은 기존의 객체지향 방식으로 작성된 프로그램에서 독립(Isolated) 컴포넌트를 추출하기 위한 방법을 제안한다. 독립 컴포넌트는 별도의 컴포넌트가 필요 없이 독자적으로 이용 가능한 컴포넌트를 말한다. 기존 프로그램에서 추출되는 독립 컴포넌트는 다른 응용프로그램 개발에 쉽게 사용될 수 있다. 본 논문에서 제시하는 추출 방법은 기존의 객체지향 프로그램을 분석하여 클래스 정보를 추출하고, 클래스간의 의존 관계를 검사하여 상호의존성이 낮고 범용성이 높은 클래스 모듈을 선택하는 것이다. 대상 모듈의 범용성은 프로그램내에서 얼마나 많이 사용되는 가로 정의된다. 본 논문에서 제시하는 방법을 사용하여 컴포넌트를 자동으로 추출하는 시스템을 구축하고자 한다.

1. 서 론

기존의 객체지향 방식으로 구축된 시스템을 컴포넌트 기반의 시스템으로 전환하기 위해서는 소스코드의 클래스로부터 컴포넌트를 추출하는 작업이 필요하다. 이 때 재사용성이 가장 높은 클래스를 어떻게 선택하고 판별할 것인지가 가장 중요한 문제점이 된다. 객체지향 시스템은 일반적으로 클래스간의 밀접한 연관관계를 가지고 있다. 반면 컴포넌트 기반 시스템은 개별 컴포넌트의 독립성이 중요하다. 기존의 객체지향 시스템에서 일부분을 분리하여 컴포넌트화 시키는 작업은 컴포넌트화가 용이한 클래스를 선택하여 선택한 클래스를 시스템내의 다른 클래스와의 연관관계를 제거하는 과정이 포함되어야 한다.

본 논문에서는 객체지향 방식으로 작성된 프로그램을 컴포넌트 방식으로 재사용하기 위한 독립 컴포넌트 추출 방법을 제시하고자 한다. 본 논문에서 제시한 방법은 컴포넌트 개발 지원도구인 COBALT 에서의 컴포넌트 추출 시스템의 컴포넌트 추출 기능으로 사용된다.

본 논문의 2 장에서는 객체지향 시스템과 컴포넌트 기반 시스템에 대해서 설명하고, 3 장에서는 컴포넌트 추출을 위한 방법에 대해서 살펴본다. 그리고, 마지막으로 4 장에서는 결론과 향후 연구방향을 제시하고자 한다.

2. 객체지향과 컴포넌트

2.1 객체지향 시스템

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 된다. 정보 은폐(Information Hiding)는 객체 내의 자료를 외부로부터 숨긴 채, 공개된 메소드들을 통해서만 허가된 정보를 접근하도록 하는 기법이다. 정보 은폐를 통하여 객체 내부의 자료에 대한 무결성을 확보할 수 있다.

추상화(Abstraction)란 관련 있는 정보들을 일반화 시켜서 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 작업해 주도록 함으로서 재사용성과 생산성을 향상시키는 장점을 제공한다.

이와 같은 상속기능을 이용하여 일반적인 객체 지향 프로그램은 클래스 단위로 프레임워크를 확장해서 구성해나가는 특징이 있다. 이런 특징은 캡슐화에 대치되어 클래스간의 연관성을 더 높여주게 된다.

2.2 컴포넌트 기반 시스템

1990 년대에 국내외 IT 업계는 크게 두 종류의 중요한 패러다임의 진행을 경험하고 있다. 첫번째는 생성되는 응용 시스템의 종류와 관계된 새로운 컴퓨팅 모델로서 네트워크 컴퓨팅(network computing)의 출현이며, 두번

째는 이러한 응용 시스템이 생성되는 방법과 관련되는 것으로서 전통적인 개발방법과 아주 다른 방법인 컴포넌트 기반 개발(CBD: Component-Based Development) 방법의 출현이다. CBD는 낮은 비용으로 빠르게 어플리케이션을 개발하는 최선의 방법으로 간주되며, 이 개발 방법은 그 동안 산업계에서 장점으로 부각되고 있는 분산객체 기술(distributed object technology)과 자연스럽게 조화될 수 있다. 즉, 어플리케이션 개발자는 인터넷을 통해 해당 기업이나 조직의 network 상에 분산되어 있는 사용 가능한 바이너리 컴포넌트들의 인터페이스만을 간단히 정의하고 통합하여 응용 시스템을 쉽고(easier), 빠르고(faster), 저렴하게(cheaper) 개발할 수 있다.

컴포넌트는 관점에 따라서 다양하게 정의되고 있다. 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 있다.

컴포넌트의 중요한 특성중에 하나는 개별 컴포넌트들이 독립성이 강하고, 조립 가능하다는 것이다. 독립성을 얻기 위해서는 개별 요소간의 의존성 및 연관성이 적어야 한다. 또한 조립가능하기 위해서는 상호간의 연결을 위한 인터페이스가 잘 정의되어 있어야 하고, 인터페이스에 유연성을 갖고 있어야 한다. 이러한 특징은 객체지향 시스템으로서 표현하기 힘든 부분으로서 기존의 객체지향 시스템을 컴포넌트 기반 시스템으로 변환하기 위해서는 보다 더 독립성과 인터페이스를 재 정의 할 수 있도록 재구성되어야 한다.

3. 객체지향 프로그램에서의 독립 컴포넌트 추출

본 연구의 컴포넌트 추출의 기본 목표는 컴포넌트화 시켰을때 가장 재사용성이 높은 부분을 찾아서 컴포넌트화 시키는 것이다. 여기서 재사용성이 높다는 것은 다른 시스템에서의 활용가치가 높다고 말할 수 있다.

현재 본 연구에서의 컴포넌트 추출방법의 개략적인 절차는 다음과 같다.

1. 기존 코드에서 컴포넌트로 변환할 만한 부분 검색
2. 컴포넌트로 변환할 가치가 있는지 사용자의 결정
3. 변환할 컴포넌트를 EJB Style 로 변환

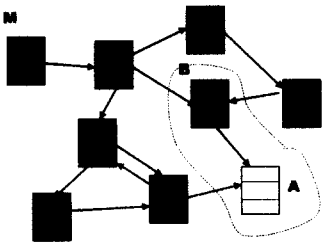
구조 분석에 의한 컴포넌트 추출방법은 소스코드의 모듈간의 관계 분석을 통해서 컴포넌트를 식별하는 것이다.

먼저 클래스간의 관계인 의존성(dependency)은 다음과 같이 정의할 수 있다.

- 메소드에서 다른 클래스의 인스턴스의 메소드를 호출
- 메소드 내에서 다른 클래스의 멤버 데이터를 사용 및 변경, 생성
- 클래스 내에서 다른 클래스를 멤버변수로 정의

일반적으로 독립 컴포넌트의 사용방법은 컴포넌트가 서비스 제공을 하고 이 서비스를 응용프로그램측에서 이용하는 스타일이다. 이와 같은 일반적인 스타일을 고려하면 임의의 특정한 클래스가 다른 클래스에 대한 의존성이 낮을수록 컴포넌트로서의 독립성이 높다고 할 수 있다.

<그림 1>은 이와 같은 의존성을 찾은 예이다.



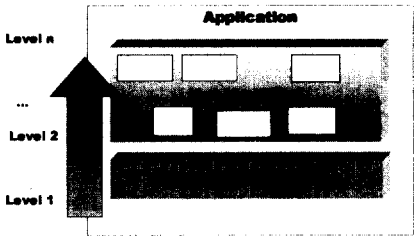
<그림 1> 클래스간의 호출 관계

다른 클래스로의 의존성이 없는 경우를 base core 클래스라고 정의한다. base core 클래스는 해당 응용프로그램의 제일 하위 레이어를 구성하는 부분으로서 Level 1 클래스라고 정의한다. 보다 상위로 올라갈수록 Level은 증가한다.

1. base core 클래스를 먼저 찾는다.
2. base core 클래스를 중심으로 두번째 클래스를 찾아서 클러스터링한다.
3. 클러스터링 한 결과를 base core 로 삼아서 Level n 까지 2 의 과정을 반복한다.

<그림 1>에서 클래스 A는 base core 클래스로서 선택되었다. 이 클래스 A를 사용하는 의존성 관계를 가진 클래스 B까지 포함하여 Level 2 컴포넌트가 만들어진다. 클래스 B를 컴포넌트화 시킬 경우 클래스 A는 반드시 필요하므로 클래스 A와 B를 함께 묶어주는 것은 합당하다. 그러나 Level 2 에서 핵심적인 역할을 하는 클래스는 클래스 A가 아니라 클래스 B라고 말할 수 있다. 클래스 B는 클래스 A가 가지고 있지 않은 응용 로직을 가지고 있기 때문이다.

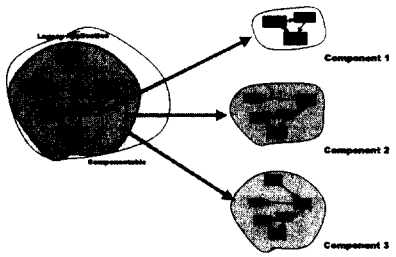
이러한 과정을 반복적으로 거치다 보면 최상위 레벨에 도달하게 되는데, 가장 최상위 Level 일 경우의 컴포넌트는 응용프로그램 자신이 된다 이러한 추출방법을 통해 얻어질 결과물은 <그림 2>와 같은 형태를 띄게 된다.



<그림 2> 독립 컴포넌트의 추출 과정

대상 응용프로그램이 계층구조(Layered)를 가지고 있다고 하면, 계층구조상에 하부계층에 속하는 부분부터 컴포넌트로 인식되어 컴포넌트로 추출되어진다. 사용자에게는 Level 1에 해당하는 클래스부터 우선순위를 두어 컴포넌트 후보로서 제시한다. 이때 사용자가 컴포넌트로 만들기 위해 선택한 클래스는 해당 상위 클래스들까지 모두 포함하여 컴포넌트로 만들어져야 한다.

이러한 추출 방식을 사용하면 <그림 3>과 같은 다양한 형태의 컴포넌트 후보가 만들어진다.



<그림 3>컴포넌트 후보의 추출 결과

각각의 컴포넌트 후보들은 독립적으로 사용될 수 있다. 하지만 좀 더 재사용할 만한 가치가 있는 컴포넌트들을 분리해낼 필요가 있다. 이를 위해 본 논문에서는 범용성(Mediocrity)이라는 척도를 제시한다. 범용성은 독립(Isolated) 컴포넌트의 품질을 평가하기 위한 척도로서 컴포넌트가 얼마나 범용적으로 다양한 분야에 사용될 수 있는지를 나타낸다. 응용 시스템에 사용되어진 하나의 컴포넌트(C)의 범용성은 다음과 같이 정의되어질 수 있다.

범용성(M_C) = 컴포넌트(C)를 사용하는 모듈의 개수(UM_C)/전체 모듈의 수(AM)

컴포넌트를 사용하는 모듈의 개수(UM_C)는 컴포넌트를 얼마나 많은 부분에서 사용하는지를 의미한다. 컴포넌트를 하나의 모듈에서만 사용하는 경우보다는 여러 모듈에서 사용한 경우가 범용성이 더 높다고 말할 수 있다. 사용하는 경우의 횟수(메소드 호출이나 참조의 빈도)를 계산하는 경우에는 컴포넌트의 형태에 따라 빈도수가 달라질 수 있으므로 객관적으로 사용할 지표로는 메소드 바디의 패턴을 분석하는 기술을 적용하지 않는 상태에서는 적용하기가 쉽지 않다. 반면에 모듈의 개수(UM_C)는 쉽게 적용할 수 있고 의미 있는 척도이다. 범용성의 척도를 이용하여 추출된 컴포넌트들을 범용성이 높은 순서대로 정렬할 수 있다. 사용자는 범용성 척도와 시스템에 대한 이해를 기반으로 적합한 컴포넌트들을 선택할 수 있다.

4. 결론

본 논문은 객체지향 프로그램에서 컴포넌트를 추출하기 위한 방법을 제안하였다. 각 클래스간의 사용관계를 이용하여 재사용 빈도를 측정하고 이를 이용해 범용성을 측정하여 재사용가치가 높은 컴포넌트를 추출하려 하였다. 여기에서 제시한 방법은 객체지향 시스템뿐만 아니라 구조적 프로그래밍으로 작성된 시스템에서도 유사하게 적용될 수 있다

향후 연구과제로는 클래스를 사용하는 형태별로 적합한 가중치를 부여하여 범용성의 계산 방식을 더욱 정교하게 만드는 일이다. 또한 다양한 사례 연구를 통해서 경험적인 실험결과를 얻어내어야 한다. 여기에 현재의 알고리즘을 이용하여 얻어낸 독립 컴포넌트들끼리의 연산방법을 이용하여 보다 추상화 되고 높은 수준의 비즈니스 로직을 추출하는 것이 고려될 수 있다

참고문헌

[1] Roger S. Pressman. "Software Engineering, A Practitioner's Approach", 3rd Ed. McGraw Hill, 1997

[2] 유영란, 김수동. "객체지향 객체 모델의 컴포넌트 모델 전환 지침", 한국정보처리학회 춘계학술발표 논문집, 2000.

[3] Grady Booch, James Rumbaugh, and Ivar Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley, 1999.

[4] Kozaczynski, W. and Booch, G., "Component-Based Software Engineering", IEEE Software, pp. 34-36, Sept./Oct. 1998.

[5] Sterling Software Inc., "The CBD Standard Version 2.1", Sterling Software, July 1998.

[6] Souza, D. F. and Wills, A. C., "Objects, Components, and Components with UML", Addison-Wesley, 1998