절차적 프로그램으로부터 객체기반 프로그램으로의 변환 방법

이정화, 김현수
금오공과대학교 컴퓨터공학부

The Method of Transformation from Procedural Program into Object Based Program

Jeong Hwa Lee, Hyeon Soo Kim
School of Comp. & Soft. Eng., Kumoh Nat'l University of Technology

요 약

기존의 절차 중심적으로 개발되어진 시스템들은 일반적으로 오래 전에 개발되었으며 규모가 크고 구조화되어 있지 않아서 이해하기 어렵고 분석에 드는 비용이 크다. 그러나 그런 시스템들은 이미 운영 환경에 관한 많은 중요한 정보들을 포함하고 있어서 시스템을 폐기하기 곤란하며 새로 개발하는 데에도 많은 비용과 시간이 요구된다. 이러한 절차 중심적 소프트웨어들이 제공하는 서비스를 계속 유지해 가면서 그 시스템을 현대화하기 위한 방안으로 객체 지향 구조로의 소프트웨어 재공학이 요구되고 있다. 본 논문에서는 기존의 절차 중심적 소프트웨어 시스템을 객체 지향 소프트웨어 시스템으로 변환하는 방법에 관하여 논의한다. 이를 위해 프로그램에 내재되어 있는 타입 가시성 정보와 자료 가시성 정보를 그래프로 표현하고 이 그래프를 이용하여 객체를 추출하는 방법을 제시한다. 또한, 추출된 객체들을 클래스로 표현하고 클래스들 간의 관계를 파악하며, 클래스들을 기반으로 기존의 코드를 객체 지향 중심의 코드로 변환하기 위한 방법들을 연구한다.

1. 서론

최근 들어 객체 지향 패러다임이 많은 소프트웨어 개발 현장에서 채택되어 지고 있다. 기존 시스템들은 절차 중심적으로 개발되어진 경우가 많아서 유지 보수 단계에서 많은 어려움을 겪고 있는 반면, 객체 지향 패러다임은 재사용성, 확장성 등의 특성을 내포하고 있어 유지 보수 단계를 용이하게 해 줄 수 있기 때문이다[2]. 기존의 시스템들은 일반적으로 오래 전에 개발되었으며 규모가 크고 구조화되어 있지 않아서 이해하기 어렵고 분석에 드는 비용이 크다. 그러나 그런 시스템들은 이미 운영 환경에 관한 많은 중요한 정보들을 포함하고 있어서 시스템을 폐기하기 곤란하며 새로 개발하는 데에도 많은 비용과 시간이 요구된다 이러한 절차 중심적 소프트웨어들이 제공하는 서비스를 계속 유지해 가면서 그 시스템을 현대화하기 위한 방안으로 객체 지향 구조로의 소프트웨어 재공학(reengineering)이 요구되고 있다[2]. 재공학은 소프트웨어 개발비용을 감소시키고 유지 보수를 용이하게 하기 위한 핵심 기술로서 인식되고 있다. 본 논문에서는 기존의 절차 중심적 소프트웨어 시스템을 객체 지향 소프트웨어 시스템으로 변환하는 방법에 관하여 논의한다 먼저 기존의 소스 코드로부터 객체 후보를 추출하기 위한 방법을 제안한다. 이를 위해 프로그램에 내재되어 있는 타입 가시성 정보와 자료 가시성 정보를 그래프로 표현하고 이 그래프를 이용하여 객체를 추출하는 방법을 제시한다. 다음으로, 추출된 객체들을 클래스로 표현하고 클래스들 간의 관계를 파악하며, 클래스들을 기반으로 기존의 코드를 객체 지향 중심의 코드로 변환하기 위한 방법들을 연구한다.

2. 관련 연구

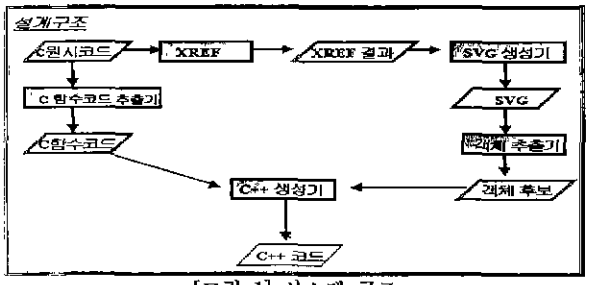
유지 보수자의 작업은 코드 내에서 타입과 데이터(전역변수, 정적 변수), 프로시저들의 관련 그룹인 객체가 파악된다면 상당히 쉬워진다는 것에 기초하여 프로그램 내에서 객체를 찾아내려는 연구가 있었다[3, 4]. 연구[4]에서는 전역변수 기반, 타입 기반과 반환 타입 기반(receiver based)을 통해 객체를 파악하였다. 그러나 연구[4]에서는 객체후보의 구성원간의 충돌문제 즉, 전역변수 타입이 사용자 정의 타입인 경우와 프로시저의 매개변수로 서로 다른 두 개의 사용자 정의 타입이 이용되는 경우 등에 대한 객체 정제 과정이 없으므로 파악된 객체의 질이 떨어진다 할 수 있다. 이것은 본 논문에서 보안될 것이다. 또 다른 연구로는 절차 중심 시스템을 캡슐(capsule= application-semantic data structure + manipulating procedure)들로 구성된 객체기반 시스템으로 변환한 객체 모델과 영역 지식으로부터 출발한 객체 모델을 비교 정제하여 최종 객체 모델을 생성해내는 COREM(Capsule Oriented Reverse Engineering Method)[5]이 있다. COREM에서는 객체와 유사한 캡슐로 이루어진 객체기반 시스템을 Scb(Capsule Based System)라 했으며 이것이 COSE(Capsule Oriented System Evaluation)에 의해 평가되어 SCL(Software Component Library)에 저장되는 과정과 또한, SCL에 저장된 재사용 가능한 컴포넌트(캡슐)들이 객체 지향 시스템 개발 방법을 통해 새로운 소프트웨어 시스템으로 생성되는 과정을 제안하고 있다.

3. 시스템 구조 및 도구들의 기능

이 절에서는 시스템의 구조와 그 구조의 단계별 기능을 설명한다.

3.1 시스템의 구조

본 논문의 전체적인 시스템 구조는 아래 [그림1]과 같다



[그림 1] 시스템 구조

3.2 시스템의 단계별 기능

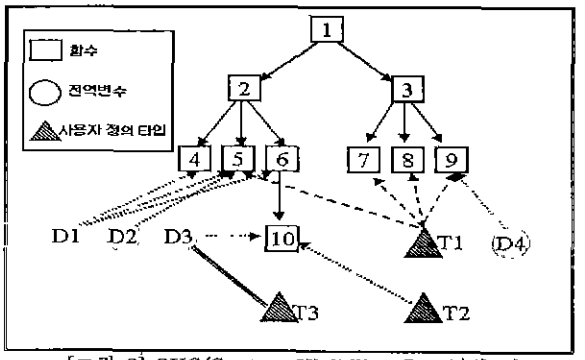
각 단계별 기능을 기술하면 다음과 같다.

(1) XREF

- 입력: C 원시코드(C 프로그램 코드)
- 출력: C 원시코드에서 추출한 화일 이름, Include정보, 사용자 정의 타입, 전역변수, 함수정보 등을 나타내는 텍스트 문서
- 기능 : 전역변수, 사용자정의 타입, 함수정보 등을 추출함

(2) SVG생성기

- 입력 XREF 결과문서
- 출력: SVG [그림2]
- 기능: XREF 결과문서에서 SVG(System Visibility Graph)를 생성한다. SVG는 소프트웨어 시스템의 구성 요소인 함수와 데이터 타입 및 전역 변수들 간의 정의 및 참조 관계 즉, 타입 가시성 정보와 자료 가시성 정보를 보여주는 그래프이다 [1]



[그림 2] SVG(System Visibility Graph)의 예

(3) 객체추출기

- 입력 SVG
- 출력 객체 후보 화일
- 기능 SVG를 탐색하면서 객체후보(C++Class) 생성규칙에 따라 전역 변수, 사용자 정의 타입, 함수들의 연관관계에 규칙을 *적용하여 그룹화한 객체후보를 만든다.*

(4) C함수 코드 추출기

- 입력 C 원시코드
- 출력 C 함수코드
- 기능: C 원시코드로부터 함수 부분만을 분리하여 각각 함수의 이름을 화일의 이름으로 갖는 화일들을 생성

(5) C++ 생성기

- 입력: 객체 후보, C 함수코드
- 출력: C++ 코드
- 기능: C 함수코드를 C++ 멤버함수로 변환하여 객체 후보 화일과 결합하여 완전한 C++ 코드를 만든다.

4. 객체후보 추출 및 정제

이 절에서는 3절에서 만들어진 SVG를 통해 객체후보(C++ Class)를 생성하기 위한 방법과 그 객체 후보의 질(quality) 높이기 위한 정제 과정을 설명한다.

4.1 객체 후보

복잡한 SVG에서 각 객체(함수, 사용자 정의 타입, 전역변수)들간에 관련 있는 것들을 그룹화한 것을 객체 후보(Object Candidate)라고 한다. 다시 말해 객체 후보는 사용자 정의 타입과 전역변수와 이를 조작하는 함수의 집합으로 정의되어질 수 있다. 따라서 객체 후보는 모듈화, 추상화, 정보은닉(Information Hiding)을 제공하는 C++의 클래스와 유사하게 된다. 객체 후보는 아래와 같이 정의될 수 있다.

$$OC = (O, T, \Delta)$$

$O \subset F$, $T \subset T$, $\Delta \subset D$. O는 함수의 집합이고, T는 사용자 정의 타입의 집합이며, Δ는 전역변수의 집합이다. 여기서 객체 후보는 전역변수들을 기반으로 하는 형태와 사용자 정의 타입을 기반으로 하는 형태가 있다.

4.2 전역변수 기반 객체 후보

전역 변수들과 그 변수들을 사용하는 함수들을 묶어 객체 후보를 파악한다 [그림2]에서 전역변수 기반 객체 후보는 다음과 같다.

$OC_{D1} = (O, T, \Delta) = (\{4, 5, 6\}, \phi, \{D1\})$

$OC_{D2} = (O, T, \Delta) = (\{5\}, \phi, \{D2\})$

$OC_{D3} = (O, T, \Delta) = (\{10\}, \{T3\}, \{D3\})$

$OC_{D4} = (O, T, \Delta) = (\{9\}, \phi, \{D4\})$

4.3 사용자 정의 타입 기반 객체 후보

사용자 정의 타입과 그것을 매개변수나 리턴 값의 타입으로 사용하는 함수들을 묶어 객체 후보를 파악한다. [그림2]에서 사용자 정의 타입기반 객체 후보는 다음과 같다.

$OC_{T1} = (O, T, \Delta) = (\{5, 7, 8, 9\}, \{T1\}, \phi)$

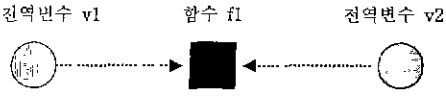
$OC_{T2} = (O, T, \Delta) = (\{10\}, \{T2\}, \phi)$

$OC_{T3} = (O, T, \Delta) = (\phi, \{T3\}, \{D3\})$

4.4 객체 후보의 정제(refinement)

각각의 전역변수와 사용자 정의 타입과 연관된 객체후보는 중복되어서는 안 된다. 그러나 시스템은 수많은 개체(함수, 사용자 정의 타입, 전역변수)들간의 무수한 관계가 존재하므로 실제적으로 객체 후보는 중복성을 가지게 되며 그 숫자가 커지게 되고 질(*quality*) 또한 떨어지게 된다[1]. 따라서 생성된 그룹에 대한 정제 과정이 필요하다.

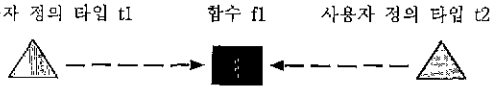
(1)전역 변수간의 응집 정제 방법



v1과 v2 : 전역 변수,
$M_d(v1)$, $M_d(v2)$ v1과 v2를 조작하는 함수의 집합,
$C = M_d(v1) \cap M_d(v2)$이고 $n(M_d(v1)) \leq n(M_d(v2))$일 때

if $1 \leq n(C) \leq 2$ 이고 $n(C) < n(M_d(v1)-C)$ 이라면
v1과 v2 각각을 기반으로 한 객체 후보 생성, f1과 같이 공통된 함수는 friend 함수로 선언
else *v1과 v2를 기반으로 한 병합된 하나의 객체 후보 생성*
[그림2]에서 D1과 D2와 함수 5는 병합되는 예가 된다.

(2)사용자 정의 타입간의 충돌 정제 방법



t1, t2 사용자 정의 타입
if t1이 t2의 서브 타입이라면
t1이 base class가 되고 t2는 t1으로부터 상속받는 subclass의 형태가 된다 그리고 공통 함수 f1은 friend 함수로 선언
else *t1, t2의 각 그룹이 형성되며 f1은 friend 함수*
[그림2]에서 T2와 T3이 서브 타입 관계가 아니라면 함수10은 friend 함수가 되며 각각의 그룹이 형성된다 이것은 T2, T3가 함수의 매개변수로 사용되는 경우 또는 두개 중 하나가 반환 타입으로 사용될 경우이다. T3이 함수10에 보여지지는 않지만 D3의 타입이 T3이므로 사실상 함수10은 T3과 연관된다

(3) 전역 변수와 사용자 정의 타입간의 충돌 정제 방법

만약 v1의 타입이 사용자 정의 타입이거나 v1이 전역 변수 충돌에서 다른 전역 변수들과 하나의 정제된 객체 후보를 형성하였다면 f1은 friend 함수가 되며 두 개의 객체 후보가 생성된다 그렇지 않다면 v1을 t1에 병합하여 하나의 객체 후보를 생성한다. [그림2]에서 함수 9에서 충돌 나는 T1과 D4는 하나의 객체 후보가 되며 함수 5에서 충돌나는 D1, D2와 T1은 두 개의 그룹이 된다. 그 이유는 전역변수 충돌에서 하나의 정제된 Group이 형성되었기 때문이다.

4.5 C원시 코드에서 C++ Class로 변환 결과

• C원시 코드(car.c)

```
#include <stdio.h>
typedef struct{
   int num_wheels;
   int range;
}Vehicle;

typedef struct{
   Vehicle v;
   int passengers;
}Car;

void comp(Vehicle *v, Car *c){
   if(v->range > (c->v).range) printf("Vehicle Range is big\n");
   else printf ("Car Range is big\n");
}

void compare(Car *c, Car *c1){
   if(c->passengers > c1->passengers) printf("C's passenger is big\n");
   else printf("C1's passenger is big\n");
}
void Car_Show(Car *c){
   printf("Wheels : %d\n",(c->v).num_wheels);
   printf("Range : %d\n", (c->v).range);
   printf("Passengers : %d\n", c->passengers);
}
```

• C++ class(car.c.cls)

```
#include <iostream.h>
class Car;
class Vehicle {
      int num_wheels;
      int range;
   public :
      Vehicle(int a, int b) {
         num_wheels = a;
         range = b;
      }
      int get_num_wheels(){ return num_wheels ; }
      int get_range(){ return range ; }
      friend void comp(Vehicle* a, Car* b);
};

class Car : public Vehicle {
      int passengers;
   public :
      Car(int a, int b, int c) : Vehicle(a, b) {
         passengers = c;
      }
      int get_passengers(){ return passengers ; }
      void Car_Show();
      friend void comp(Vehicle* a, Car* b);
      friend void compare(Car* a, Car* b);
};
```

Car_show()에서 매개변수가 없어지며(이 부분과 관련된 규칙은 생략) 사용자 정의 타입이 두 개 이상 매개변수로 사용되는 comp와 compare는 friend 함수가 되어 객체의 형태가 매개변수로 사용되었으며 Vehicle이 Car의 서브타입이므로 Car는 Vehicle로부터 상속받게 된다.

5. 결론 및 향후 연구 과제

본 논문에서는 C원시코드로부터 C++ class구조의 객체 지향 코드로 변환 방법을 정제 과정을 포함하여 제안하였다. 이 변환 방법은 전 과정이 거의 자동화된 것으로 프로그래머의 전문적 지식이나 판단이 개입될 부분을 많이 줄였다. 그러나 본 논문에서는 함수 내에서 사용되는 static 변수가 지역변수처럼 처리되는 단점이 있다. 앞으로 이 부분의 보완과 더 나은 정제작업 알고리즘이 이루어져야 할 것이다.

참고 문헌

[1] 김현수, "소프트웨어 유지 보수를 위한 프로그램 재구성 접근 방법", 한국과학기술원 박사학위논문, 1995.
[2] 박의진, 배두환, "객체 지향 재공학을 위한 객체 지향 코드 생성방법", 한국정보과학회 소프트웨어공학회지, pp.45-57,1999.3.
[3] H. Gall, R. Klosch, "Finding Objects in Procedural Program: An Alternative Approach", 2nd Working Conference on Reverse Engineering (WCRE '95). pp.208-16. IEEE Computer Society Press, July 1995.
[4] Panos E. Livadas and Theodore Johnson, "A New Approach to Finding Objects in Programs," Software Maintenance' Research and Practice. Vol. 6. pp 249-260, 1994.
[5] H. Gall, R. Klosch, "Capsule Oriented Reverse Engineering for Software Reuse," Proc of 4th European Software Engineering Conference, 1993