객체지향 프로그래밍에서의 Demeter 법칙의 정식화

황 석 형† 이 용 근†† 양 해 술†††

요 약

객체지향 프로그래밍에 있어서 클래스간의 불필요한 상호 의존관계를 줄이기 위한 프로그래밍 스타일에 관한 규칙으로써 Demeter의 법칙(The Law of Demeter)이 있다. 본 논문에서는 종래의 비형식적인 Demeter의 법칙을 실제의 프로그램에 적용, 평가하기 위해 클래스간의 관계로써 계승과 집약, 그리고 관련을 형식적으로 정의하고, 그것들을 이용하여 Demeter의 법칙을 정식화하였다. 또한 Demeter의 법칙을 만족하는지를 판정하는 알고리즘과 Demeter의 법칙을 위반한 프로그램에 대한 변환 알고리즘을 제안하였다.

Formulations of the Law of Demeter in the Object Oriented Programming

Suk Hyung HWANG†, Ryong Geun RHEE†† and Hae Sool YANG†††

ABSTRACT

In the last years, several articles have been devoted to the study of the Law of Demeter in the object oriented programming. The Law of Demeter is a style rule that aims at eliminating unnecessary coupling among classes. Although a large number of studies have been made on the informal definitions, little is known about the formulation of the Law of Demeter. In this article, we define three relationships among classes i.e. inheritance, aggregation and association, and formulate the Law of Demeter. We also propose the algorithms to decide whether a given program satisfies the law and to transform an unfulfilled program into a fulfilled one.

1. 서 론

최근 객체지향 패러다임에 대한 관심이 높아지고 있으며, 많은 객체지향 언어가 제안되고 있다. 또한 재사용성과 정보 은폐, 계승, 폴리모피즘을 이용한 소프트웨어의 구축 및 분석과 설계에 관한 객체지향 방법론의 등장 등 소프트웨어 공학에도 새로운 패러다임의 시대를 맞이하게 되었다[2].

그러나 객체지향 프로그래밍 언어로 작성한 프로그램에 대한 평가 및 복잡도에 관한 척도의 제안에 대한 연구는 아직까지 미흡한 실정이다. 객체지향 언어의 클래스의 품질평가에 관한 연구[8]에서는 클래스의 품질평가 척도로서 클래스를 모듈로 생각한 모듈강도와 클래스간의 계승관계에 의한 결합도라는 평가척도를 제안하고 있다. 그리고 제안척도를 C++ 프로그램에 적용한 결과, 대부분의 소프트웨어 개발자가 추상 데이타형과 계승성을 이해하지 못하고 개발에 임하고 있음을 알 수 있다.

또한, 클래스간의 상호의존관계에 주목하여, 메시지 전송에 제한을 가하는 것에 관한 연구[3, 4]가 수행되어 왔다. 이 연구에서는 클래스간의 결합도를 줄이기 위한 제한으로써, Demeter의 법칙이라는 지침을 제안하고 있다. 어떤 클래스가 이 법칙에 따르면, 그 클래스의 메소드는 제한된 클래스의 메소드만을 이용하도록 되어 클래스간의 결합도가 낮아지게 된다. 그러나 본 연구에서는 비형식적인 Demeter의 법칙의 정의와 그 법칙에 따르도록 프로그램을 변환하는 방법만을 제안한 것으로 형식

†정 회 원 : 日本오사카대학 정보공학과 박사과정
††종신회원 : 서울산업대학교 전자계산학과 강사
†††종신회원 : 강원대학교 전자계산학과 교수
논문접수 : 1994년 3월 26일, 심사완료 : 1994년 4월 30일

적인 정의가 수행되지 않고 있다.

따라서, 본 연구에서는 클래스간의 상호 의존관계에 주목하여 Demeter의 법칙을 정식화하고, 주어진 C++ 언어로 구현된 객체지향 프로그램에 대해 판정하는 알고리즘을 제안한다. 또한 위반 프로그램을 변환하는 몇가지 방법[3, 4]의 기본인 릴레이 메소드 추가방법의 알고리즘을 정식화하고 그 타당성을 검토하였다. 마지막으로 Demeter의 법칙을 만족시키기 위한 새로운 변환방법에 대해서도 제안하였다.

2. 정의

본 장에서는 프로그램상에 등장하는 클래스와 메소드, 그리고 클래스간의 관계 등에 대하여 정의한다. 우선, 프로그램 P의 클래스 집합을 C_P, 메소드 집합을 M_P로 한다.

2.1 클래스의 정의로부터 구할 수 있는 집합의 정의

클래스에는 데이타와 그것을 액세스하기 위한 메소드가 정의되어 있다. 클래스에 정의한 데이타를 데이타멤버라고 부르고, 클래스에는 다른 클래스의 인스턴스 변수도 정의할 수 있다. 또한, 클래스의 계승관계에 의해 각 클래스에는 수퍼 클래스가 정의된다. 이상의 사실에 기초하여 임의의 클래스 C에 대해서 다음과 같은 집합을 정의한다.

OB(C) : 클래스 C의 객체 집합
MD(C) : 클래스 C에 정의된 데이타 멤버의 클래스 집합
MF(C) : 클래스 C에 정의된 메소드의 집합

단, MF(C), MD(C)에는 계승된 메소드 및 데이타 멤버의 클래스는 포함하지 않는다.

계승에 관한 정의를 내리기 위하여, 클래스 C가 클래스 P를 직접 계승한다는 것을 P>>C 또는 $P \overset{1}{>>} C$로 나타내고, 클래스 P로부터 n회의 계승을 거쳐서 클래스 C가 얻어졌을 경우에는 $P \overset{n}{>>} C$로 표현한다. 또한, 클래스 C가 클래스 P를 0계층 이상에 거쳐 계승한 경우에는 $P \overset{*}{>>} C$로 나타낸다. 이것에 기초하여 클래스 C의 수퍼 클래스 집합 SC(C)를 다음과 같이 정의한다.

$$SC(C) \triangleq \{ P \in C_P \mid P \overset{*}{>>} C \}, \text{ where } C \in C_P$$

마지막으로, 임의의 메소드 m에 대해서 메소드 m이 정의된 클래스를 CM(m)으로 정의한다. 즉, $m \in MF(CM(m))$이다.

2.2 술어의 정의

임의의 두 클래스간의 메시지 전송 또는 한쪽 클래스에서 다른 쪽 클래스의 데이타 멤버를 참조하는 경우, 양 클래스간에는 상호 의존관계가 있다고 한다. 그러나 본 연구에서는 클래스간의 메시지 전송만을 생각하기로 한다. 실제의 프로그램에서는 메시지 전송과 같은 동작은 클래스에 정의한 메소드에 기술된다. 메소드 m에 관해서 다음과 같은 술어를 정의한다.

ref(m, i) : 메소드 m의 Body부에서 전역변수 i를 참조한다.
arg(m, a) : 메소드 m의 가인수 리스트에 객체 a가 있다.
contain(m, C) : 메소드 m의 Body부에 클래스 C의 객체를 생성하는 문을 포함한다.
send(m, f, C) : 메소드 m속에 클래스 C의 객체에 메시지 f를 전송하는 문을 포함한다.

이와 같은 술어는 정적으로 형이 부여되는 언어에서는 프로그램의 텍스트로부터 명확히 결정할 수 있다.

2.3 메소드에 관한 객체의 클래스

임의의 메소드 m에 대해서 그 메소드의 내부에 나타난 객체의 클래스의 집합을 다음과 같이 정의한다.

· 메소드 m에서 사용되고 있는 전역변수의 클래스의 집합 GV(m)

$$GV(m) \triangleq \{ A \in C_P \mid ref(m, i), i \in OB(A) \}$$

· 메소드 m의 가인수 리스트에서 사용되고 있는 객체의 클래스의 집합 AC(m)

$$AC(m) \triangleq \{ A \in C_P \mid arg(m, a), a \in OB(A) \}$$

· 메소드 m이 속해 있는 클래스에 정의된 데이타 멤버의 클래스의 집합 IV(m)

$$IV(m) \triangleq \{ A \in C_P \mid A \in MD(CM(m)) \}$$

· 메소드 m에서 새롭게 생성된 객체의 클래스의 집합 CRE(m)

$$CRE(m) \triangleq \{ A \in C_P \mid contain(m, A) \}$$

· 메소드 m의 공급자 클래스의 집합 SU(m)

$$SU(m) \triangleq \{ A \in C_P \mid send(m, f, A), f \in M_p \}$$

즉, 메소드 m의 공급자 클래스는 메소드 m의 Body부에서 호출된 메소드가 정의된 클래스이다.

2.4 클래스간의 관계

클래스간에는 다음과 같은 세가지 관계가 있다.

· 계승(inheritance)

$$i = \{(B, A) \mid A, B \in C_P \wedge A \overset{*}{>>} B\}$$

계승은 클래스간의 관계이며, 각 서브클래스는 수퍼 클래스의 속성과 메소드를 공유한다.

· 집약(part of)

$$p = \{ (A, B) \mid A, B \in C_P \wedge B \in MD(A) \}$$

집약은 "전체-부분" 또는 "a-part-of"로 표현되는 관계이다. 이것은 어떤 부품을 나타내는 클래스를 전체의 조립을 나타내는 클래스에 관련시키는 것이다.

· 관련(association)

$$a = \{ (A, B) \mid A, B \in C_P \wedge (B \in GV(m) \vee B \in AC(m) \vee B \in CRE(m)) \vee m \in MF(A) \}$$

관련은 어떤 클래스와 그 클래스의 메소드에서 참조하는 클래스와의 관계이다. 본 논문에서는 메소드에서 참조한 전역변수의 클래스와 메소드의 인수에 나타나고 Body부에서 참조한 클래스, 그리고 메소드에서 새로이 생성한 객체의 클래스라고 하는 세가지 경우만을 고려한다.

3. Demeter 법칙의 정식화

3.1 Strong/Weak version의 정식화

제안된 Demeter의 법칙[3]에서는 클래스간의 관계 및 메시지 전송에 대한 정의가 애매하기 때문에 실제로 주어진 프로그램에 적용하는 경우, 적용하는 사람에 의해 결과가 달라지는 경우가 있다. 여기에서는 2장의 정의를 이용하여 보다 명확히 Demeter의 법칙(Strong version)을 정의하기로 한다.

[정의 1] Demeter의 법칙(Strong version)

프로그램 P는

$$\forall C \in C_p [\forall m \in MF(C) \{ SU(m) \subset IV(m) \cup AC(m) \cup CRE(m) \cup GV(m) \}]$$

을 만족해야 한다.

즉, 임의의 클래스 C와 그 클래스에 속해 있는 메소드 m에 대해서, 메소드 m으로부터 메시지를 보낼 수 있는 클래스는 클래스 C에 정의된 데이타 멤버의 클래스, 메소드 m의 인수 객체의 클래스, 메소드 m에 의해 생성된 객체의 클래스, 메소드 m에 의해 사용된 전역변수의 클래스이어야 한다.

[정의1]을 조금 약화시켜서 수퍼 클래스에 정의한 데이타 멤버 클래스의 인스턴스에 메시지 전송을 허용하는 것도 생각할 수 있다(Weak version). 이와 같은 경우에는 다음과 같이 정의할 수 있다.

[정의 2] Demeter의 법칙(Weak version)

프로그램 P는,

$$\forall C \in C_p [\forall m \in MF(C) \{$$

$SU(m) \subset TIV(m) \cup AC(m) \cup CRE(m) \cup GV(m) \}$]
을 만족해야 한다.
단, $TIV(m) = \bigcup_{A \in SC(C)} MD(A)$

즉, 임의의 클래스 C와 그 클래스에 속해 있는 메소드 m에 대해서, 메소드 m으로부터 메시지 전송이 가능한 객체의 클래스는 클래스 C와 그 수퍼 클래스에 정의된 데이타 멤버의 클래스, 메소드 m의 인수 클래스, 메소드 m에 의해 생성된 객체의 클래스, 메소드 m에 의해 사용되고 있는 전역변수의 클래스이다. 즉, Weak version에서는 계승관계에 기초하여 수퍼 클래스에 정의된 데이타 멤버에도 메시지를 전송할 수 있다.

3.2 클래스의 분류

대부분의 클래스간의 메시지 전송은 클래스간의 관계에 따라 수행된다. 따라서, 메시지의 수신자 클래스와 송신자 클래스 사이에는 어떠한 관계가 있는지를 알아 둠으로써 Demeter의 법칙을 보다 명확히 이해할 수 있다. 이하에서는 클래스의 각 메소드에 대한 수신자 클래스를 분류한다[4].

어떤 클래스 C에 정의된 임의의 메소드 m의 Body부에 다른 클래스 D에 메시지 f를 전송하는 문이 있는 경우, 클래스 C는 메시지 f에 관해서 의뢰자 클래스가 되고 클래스 D는 메시지 f에 관해서 메소드 m의 공급자 클래스가 된다. 이와 같은 의뢰자-공급자 모델에 기초하여 임의의 메소드 m의 공급자 클래스 중에 아래의 3종의 클래스를 고려한다.

교우관계 클래스(AQ_C(m))

$$AQ\_C(m) \triangleq \{ A \in C_p \mid A \in SU(m) \text{ and } (A \notin SC(AC(m)) \text{ and } A \notin SC(IV(m)) \text{ and } A \notin SC(CM(m))) \}$$

상기의 식은 다음과 같이 나타낼 수 있다.

$$AQ\_C(m) = SU(m) - SC(AC(m)) - SC(IV(m)) - SC(CM(m))$$

즉, 교우관계 클래스는 인수 클래스, 데이타 멤버의 클래스, 그리고 메소드 m이 속해 있는 클래스도 아닌 공급자 클래스이며 메소드 m에 의해 생성된 객체의 클래스 및 전역변수 클래스의 인수턴스에 메시지를 보내는 경우 등이 해당한다.

· 우선 교우관계 클래스(PA_C(m))

$$PA\_C(m) = SC(CRE(m)) \cup SC(GV(m))$$

메소드 m이 동작함으로써 생성되는 객체의 클래스와 메소드의 Body부에서 이용하는 전역변수의 클래스는 교우관계 클래스 중에서 비교적 관계가 깊은 클래스로, 여기에서는 우선 교우관계 클래스로서 분류한다.

우선 공급자 클래스(PS_C(m))

$$PS\_C(m) \triangleq \{ B \in C_p \mid B \in SU(m) \text{ and } (B \in SC(IV(m)) \text{ or } B \in SC(AC(m)) \text{ or } B \in SC(CM(m)) \text{ or } B \in PA\_C(m)) \}$$

위의 식은 다음과 같이 나타낼 수도 있다.

$$PS\_C(m) = SU(m) \cap (PA\_C(m) \cup SC(IV(m)) \cup SC(AC(m)) \cup SC(CM(m)))$$

우선 공급자 클래스는 우선 교우관계 클래스, 메소드 m이 속해 있는 클래스에 정의된 데이타 멤버의 클래스, 메소드 m의 인수 클래스, 메소드 m이 정의된 클래스, 또는 그들 클래스의 수퍼 클래스라고 하는 조건을 만족하는 공급자 클래스이다.

3.3 Strict version의 정식화

앞절에서 정의한 클래스의 분류에 기초하여, 세번째의 Demeter의 법칙(Strict version)[4]을 정의한다.

[정의 3] Demeter의 법칙(Strict version)

프로그램 P는
$\forall C \in C_p$ [$\forall m \in MF(C)$ { $SU(m) = PS\_C(m)$ }]
을 만족해야 한다.

즉, 모든 메소드의 공급자 클래스는 우선 공급자

클래스이어야 한다는 것이다.

Demeter의 법칙(Strict version)에 따르도록 프로그래밍을 수행하면 클래스간의 상호 의존관계를 극소화할 수 있고 클래스의 기술을 이해하기 쉽다.

4. 판정 알고리즘

본 절에서는 주어진 프로그램에 대해서 그 프로그램이 Demeter의 법칙을 만족하는지를 판정하는 알고리즘을 제안하기로 한다. 먼저, 알고리즘을 기술하기 위하여 다음과 같은 클래스 계층 그래프 G를 정의한다.

[정의 4] 클래스 계층 그래프

G = (V, E)

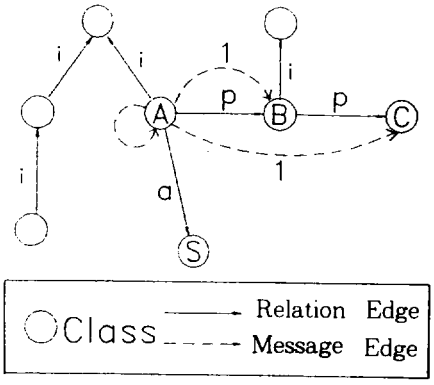
V : 클래스의 집합, E = RE ∪ ME

RE : 관계 에지의 집합, RE ⊆ V × V × R

ME : 메시지 에지의 집합, ME ⊆ V × V × W

R = {i, p, a}, i, p, a는 각각 계승, 집약, 관련을 나타냄

W : 양의 정수집합, 메시지 전송문의 수를 나타냄



(그림 1) 클래스 계층 그래프의 예
(Fig. 1) Example of Class Hierachical Graph

주어진 객체지향 프로그램은 클래스 계층 그래프 G로 변환할 수 있다. 클래스 계층그래프 G는 클래스를 정점으로 하고 RE와 ME라는 두 종류의 에지를 갖는다.

RE의 에지는 클래스간의 관계를 나타내고 클래스간의 관계인 계승(i), 집약(p), 관련(a) 중의 어느 하나가 레이블로 사용된다. 임의의 두 정점 v와 w 사이에 직접적인 계승관계 에지가 존재할 경우 ($w \gg v$), $(v, w, i) \in$ RE로 표시한다. 또한, n계층에 걸쳐서 계승한 경우 ($w \gg v$)에는 $(v, w, i^n) \in$ RE로 나타낸다. 마찬가지로 임의의 두정점 v, w가 집약 관계로 결합되어 있는 경우 ($w \in MD(v)$), $(v, w, p) \in$ RE로 표현한다. 관련 관계에 대해서도 마찬가지이다. 한편, ME의 에지는 메시지 전송을 나타낸다. 즉, $m \in MF(v)$일때, $send(m, f, w)$라면, $(v, w, W) \in$ ME가 된다. 단, W는 클래스 v의 메소드 속에 기술된 클래스 w로의 메시지 전송문의 총수를 의미한다. (그림 1)에 클래스 계층 그래프의 예를 나타내었다. 이상의 Demeter의 법칙의 정의와 클래스 계층 그래프 G에 기초하여, 판정 알고리즘을 제안한다.

주어진 프로그램을 상기의 클래스 계층 그래프로 변환하여 판정 알고리즘의 입력으로 한다. 알고리즘 Demeter에서는 먼저, 그래프상의 각 정점의 메소드 m에 대해 그 Body부의 메시지 문으로부터 공급자 클래스 $SU(m)$을 구한다. 그 다음에 클래스 계층 그래프 상에서 깊이우선 탐색을 수행하여 각 클래스 v의 수퍼 클래스의 집합 $SC(v)$를 구한다. 그 다음에는 모든 클래스 v에 대해 $(v, w, p) \in$ RE 또는 $(v, w, a) \in$ RE의 관계에 있는 클래스 w가 존재한다면, 그 수퍼 클래스의 집합 $SC(w)$ 전체와 $SC(v)$의 합집합을 클래스 v의 임시 우선 공급자 클래스 $TPS(v)$로 한다. 만일 모든 클래스 w에 대해서 $(v, w, p) \notin$ RE이며, $(v, w, a) \notin$ RE라면 클래스 v의 수퍼 클래스의 집합 $SC(v)$가 임시 우선 공급자 클래스 $TPS(v)$가 된다.

이상에서 구해진 집합 $SU(m)$와 $TPS(v)$에 기초하여, 정식화한 Demeter의 법칙을 이용하여 판정한다. 마지막에는 Demeter의 법칙을 위반한 메소드의 집합 VD $M(v)$와 그 위반메소드의 공급자 클래스의 집합 VS $C(m)$을 출력한다. 여기서, 모든 클래스에 대해 집합 VD_$M(v)$가 공집합인 경우 입력 프로그램은 Demeter의 법칙을 만족한다

고 하는 의미에서 적절한 스타일의 프로그램이라고 말할 수 있다. 다음에 판정 알고리즘 Demeter를 나타내었다.

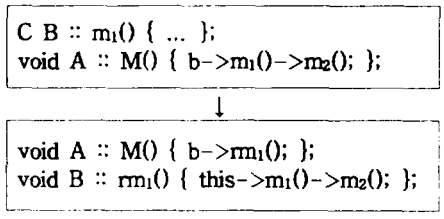
```
Demeter(G)
for each vertex v ∈ V[G] do
   for each method m of v do
        m의 Body부로부터 SU(m)을 구한다
        VS_C(m) ← 0
   VD_M(v) ← 0
   SC(v) ← 0
   TPS(v) ← 0
for each vertex v ∈ V[G] do
   color[v] ← WHITE
for each vertex v ∈ V[G] do
   if color[v] = WHITE then DFS(v)
for each vertex v ∈ V[G] do
   TPS(v) ← SC(v)
   for each vertex w ∈ V[G] do
      if (v, w, p) ∈ RE ∨ (v, w, a) ∈ RE then
         TPS(v) ← TPS(v) ∪ SC(w)
for each vertex v ∈ V[G] do
   for each method m of v do
      if SU(m) ∉ TPS(v) then
         VD_M(v) ← VD_M(v) ∪ {m}
         VS_C(m) ← SU(m) - TPS(v)
end_Demeter

DFS(x)
color[x] ← GRAY
SC(x) ← SC(x) ∪ {x}
for each (x, y, i) ∈ RE do
   if color[y] = WHITE then
        DFS(y)
    SC(x) ← SC(x) ∪ SC(y)
   color[x] ← BLACK
   end_DFS
```

5. 변환

Demeter의 법칙에 위반한 프로그램에 대해 Demeter의 법칙을 만족하도록 변환하는 몇가지 방법이 제안되고 있다[3, 4]. 여기에서는 이들 변환방법의 기초가 되는 릴레이 메소드 추가방법의 알고리즘을 살펴보며, 새로운 변환방법을 제안하기로 한다.

5.1 릴레이 메소드 추가방법[3]

몇개의 메시지를 나열하여 기술하는 경우 즉, 선행 메시지로부터 리턴된 객체에 다시 새로운 메시지를 송신하는 경우가 있다. 이와 같은 경우, 리턴된 객체가 의뢰자측의 우선공급자 클래스의 객체가 아닌 때에는 Demeter의 법칙을 위반하게 된다. 이와 같은 경우 의뢰자로부터 공급자로 메시지를 릴레이하는 새로운 메소드(릴레이 메소드라고 부름)를 클래스 계층그래프 상의 의뢰자 클래스로부터 공급자 클래스로의 관계에지로 구성되는 경로상의 각 클래스에 추가해 가는 변환방법이 있다. (그림 2)는 릴레이 메소드 추가방법의 예를 나타내고 있다.



(그림 2) 릴레이 메소드 추가방법
(Fig. 2) An Additional Way of Relay Method

(그림 2)는 (그림 1)의 클래스 계층을 기초로 하고 있다. 클래스 A로부터 클래스 B에 메시지 m_1을 전송하고 리턴되는 객체에 다시 메시지 m_2를 전송하는 메시지 전송이 수행되고 있다. 그러나, 위의 예에서는 클래스 C가 클래스 A의 메소드 M의 우선 공급자 클래스가 아니라면 Demeter의 법칙을 위반하게 된다.

메소드 M에 대해서 Demeter의 법칙을 만족하

도록 변환을 실시한다. 우선 새로운 메소드 rm_1을 클래스 B에 정의하고 메소드 M의 메시지 문을 릴레이한다. 그리고 메소드 M에서는 클래스 B에 메시지 rm_1을 전송하도록 변경한다. 만일 M에서 b $->$ $m_1()$ $->$ $m_2()$ $->$ $\cdots$ $->$ $m_n()$으로 되어 있는 경우에는 같은 방법으로 릴레이 메소드 rm_2 등을 추가해서 이상의 변환을 반복하면 모든 메소드는 Demeter의 법칙을 만족하게 된다.

이상의 변환을 알고리즘으로 나타내기 위하여 우선 클래스 계층 그래프상에서 나타나는 메시지 전송경로를 다음과 같이 정의한다.

[정의 5] 메시지 전송경로 mp

클래스 계층 그래프상에서 메시지 에지(ME)로 연결된 정점 A와 C사이의 각 정점간에 존재하는 관계에지(RE)에 따르는 경로를 메시지 전송경로라고 부르고, mp = $< v_1, v_2, \cdots, v_k >$로 나타낸다(A = v_1, C = v_k, $v_i = v_{i+1}$의 경우도 있을 수 있다). 여기서, 메시지 전송경로의 길이 | mp | 는 경로상에 있는 정점간의 관계 에지의 총수이다.

아래의 알고리즘 **RMI**는 릴레이 에지 메소드 추가방법을 실현하고 있다.

알고리즘 RMI

1. Demeter의 법칙에 위반한 메시지 전송문을 포함하는 메소드를 m_1이라 한다.
   C_1 :: $m_1()$ { $\cdots$ $c_2->m_2()->m_3()->$ $\cdots$ $->m_k()$; $\cdots$ };
   단, $i = 1, \cdots, k$에 대하여 $m_i \in MF(C_i)$.
   mp = $< C_1, C_2, \cdots, C_k >$는 메시지 전송경로라고 가정한다.

2. $c_2->m_2()->m_3()->\cdots->m_k()$에서 $->m_i()$는 Demeter의 법칙을 만족하지만 $->m_{i+1}()$은 위반하고 있는 메시지 전송(i가 최소인 것)을 찾아낸다.

3. C_i에 새로운 메소드 rm_i를 추가하고 다음과 같이 정의한다.
   $rm_i(){this->m_i()->m_{i+1}()->\cdots->m_k();}$

4. 메소드 m_1의 $c_2->m_2()->m_3()->\cdots->m_k()$ 부분을 $c_2->m_2()->m_3()->\cdots->m_{i-1}()->rm_i()$로 변환한다.

5. Demeter의 법칙에 위반하는 각 메시지 전송에 대해서 반복해서 이상의 변환을 수행한다.

상기 알고리즘에서는 먼저 주어진 프로그램의 클래스 계층 그래프 G에 대해서 Demeter의 법칙을 만족하는지를 알고리즘 Demeter로 판정한다. 판정 알고리즘으로부터 법칙위반으로 판명된 메소드 $m \in VD\text{-}M(C_i)$와 메소드 m에 정의된 메시지 전송문을 이용하여 판정 알고리즘으로 구한 임시 우선 공급자 클래스 중에서 메시지 전송경로상에서 클래스 C_i와 직접 관계하고 있는 클래스 C_j에 릴레이 메소드 rm_j를 추가한다.

5.2 변환방법의 정당성에 대한 검토

Demeter의 법칙을 위반한 프로그램은 알고리즘 RMI에 의해 법칙을 만족하도록 변환할 수 있음을 검토한다. 릴레이 메소드 추가법에 의한 변환에서는 다음과 같은 정리가 성립한다.

[정리 1] Demeter의 법칙을 위반한 프로그램은 알고리즘 RMI의 스텝 1의 조건을 만족하는 경우 RMI에 의해 법칙을 만족하도록 변환된다.

(증명) 법칙에 위반되는 메시지 전송을 포함하는 일련의 메시지 전송문 $c_2 -> m_2() -> m_3() -> \cdots -> m_k()$를 RMI가 법칙을 위반하지 않도록 변환한다는 것을 귀납법으로 보인다.

(1) $(C_1, C_2, R) \in RE$, 또는 $C_1 = C_2$이므로 $->m_2()$는 위반하고 있지 않다.
(2) $->m_j()$ (단, $j \leq i$)의 부분에서는 위반하

고 있지 않다고 한다. $->m_{i+1}()$이 위반하고 있다면 스텝 3, 4에 의해,

$c_2->m_2->\cdots->m_{i-1}()->rm_i()$

C_i :: $rm_i()\{$ this $->m_i()->m_{i+1}()$

$->\ \cdots\ ->m_k()\}$

로 변환된다.

변환전의 $c_2->m_2()->\ \cdots\ ->m_i()$가 위반하고 있지 않다는 것으로부터 변환후의 $c_2->m_2()->\ \cdots\ ->m_{i-1}()->rm_i()$도 위반하고 있지 않다($m_i$, $rm_i \in MF(C_i)$이므로). 또한, 변환후의 $->m_i()$는 같은 클래스로의 메시지 전송이므로 위반하고 있지 않다. 그리고 변환후의 $->m_{i+1}()$은 클래스 C_i에서 C_{i+1}로의 전송이므로 위반하고 있지 않다. 따라서, $->m_j()$ (단, $j \le i+1$)는 위반되지 않게 변환된다.

따라서, 이상의 변환을 Demeter의 법칙을 위반한 모든 메소드에 대해서 반복하여 수행하면 주어진 프로그램 P는 Demeter의 법칙을 만족하는 프로그램 P'로 변환된다.

Demeter의 법칙에 위반되는 프로그램에 대해서 법칙을 만족하도록 변환하는 릴레이 메소드 추가법을 알고리즘 RMI로써 정식화하였다. 그러나, 다음과 같은 두가지 경우, 릴레이 메소드 추가법으로는 변환할 수 없다는 것이 명확해 졌다. (그림 3)의

```
void C1 :: m1() { ... this->m2()->m3()->...->mk(); ... }
```

(a) Case 1

```
x1 = o->m1(); ... x2=x1->m2(); ... x3=x2->m3(); ...
```

(b) Case 2

(그림 3) 변환 알고리즘 문제점의 예
(Fig. 3) Example of Transformation Algorithm Problem

Case1에서 메소드 m_1 속의 메시지 전송문 $->m_3()$가 법칙을 위반하고, 클래스 C_2와 C_3사이에 관계 RE가 존재하지 않는 경우($(CM(m_2), CM(m_3), R) \notin RE$)가 있다. 이런 경우는 알고리즘 RMI에서와 같이 $CM(m_3)$에 릴레이 메소드를 추가하고 m_1의 메시지 전송문을 변경해도 위반은 해결되지 않는다. Case 2와 같이 일시적으로 지역변수를 이용하는 경우는 클래스간의 관계를 변화시키지 않는 한, 실행시에 메소드를 릴레이하는 객체를 특정할 수 없으므로 실현할 수 없다. 즉, 메소드 m_2로의 호출시에 리턴되는 객체를 그 클래스의 데이타 멤버에 저장($CM(m_3) \in MD(C_2)$)하는 등의 변환은 가능하지만 이것은 클래스간의 관계를 변경하는 것이 된다.

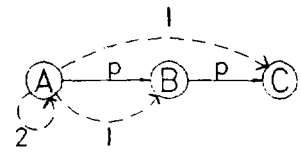
5.3 새로운 변환방법의 제안

지금까지 세가지 클래스 관계를 이용하여 클래스 계층을 논의하였지만, 특히 집약관계의 경우에는 클래스 구조를 재구성하는 편이 좋은 경우도 있다.

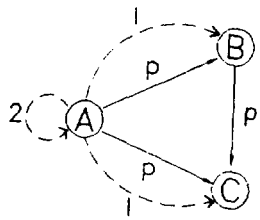
여기서는 법칙 위반이 발생한 클래스와 그 메시지 전송경로상의 클래스와의 클래스 계층을 변경해서 Demeter의 법칙을 만족하도록 변환하는 방법으로써 다음의 두가지 방법을 제안한다.

5.3.1 정보 이동(Moving Information)

집약관계의 클래스 사이에서 Demeter의 법칙을 위반한 경우, 의뢰자 클래스와 공급자 클래스를 관계짓는 방법을 생각할 수 있다. (그림 4)에는 집약관계의 클래스 계층구조를 변경한 예를 보인다.



(a) 정보 이동 전
(a) Before moving information

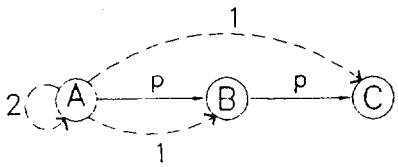


(b) 정보 이동 후
(b) After moving information

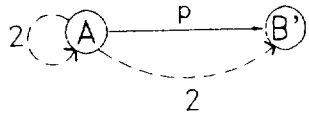
(그림 4) 정보 이동법의 적용 예
(Fig. 4) Example of Moving Information Method

5.3.2 새로운 클래스 생성(Creating New Class)

집약관계에서는 경우에 따라서 새로운 클래스를 생성하면 의미적으로 보다 명확히 되는 경우도 있다. 이러한 사고방식에 기초하여 집약관계의 두 클래스를 조합하여 새로운 클래스를 만드는 것도 생각할 수 있다. (그림 5)는 집약관계의 클래스간에 Demeter의 법칙에 위반된 메시지 전송이 있는 경우, 집약관계의 두 클래스를 합병해서 새로운 클래스를 생성한 예이다.



(a) 새로운 클래스 생성 전
(a) Before Creating New Class



(b) 새로운 클래스 성성 후
(b) After Creating New Class

(그림 5) 새로운 클래스 생성
(Fig. 5) Creating New Class

6. 결 론

본 연구에서는 클래스간의 상호 의존관계에 주목하여 클래스간의 관계와 메시지 전송에 대해서 정의하고 문헌 [3, 4]에서 제안된 Demeter의 법칙을 보다 구체적으로 정식화하였다. 또한, 이들 정의와 정식화에 기초하여 주어진 C++ 언어로 구현된 객체 지향 프로그램에 대해 Demeter의 법칙을 만족하는지 어떤지를 판정하는 알고리즘과 위반한 프로그램을 변환하는 몇 가지 방법의 메소드 추가방법의 알고리즘을 정식화하고 그 타당성을 기술하였다.

향후의 연구과제로서는 본 연구에서 제안한 정식화 기법을 다른 평가기준에 적용해 보는 것 등이 있다.

참 고 문 헌

[1] B. Henderson-Sellers, 'A Book of Object Oriented Knowledge : Object Oriented Analysis, Design, and Implementation : a new approach to software engineering', Prentice Hall, 1992.

[2] David E. Monarchi and Gretchen I. Puhr, "A Research Typology for Object-Oriented Analysis and Design," Communications of the ACM, Vol.35, No.9, September 1992.

[3] Karl J. Lieberherr, Ian Holland and Arthur J. Riel, "Object-oriented programming : An objective sense of style, " in Proc. Object-Oriented Programming Systems, Languages, and Applications Conf., ACM, New York, pp.323-334, 1988.

[4] Karl J. Lieberherr and Ian Holland, " Assuring Good Style for Object-Oriented Programs," IEEE Software, 6(5) 1989.

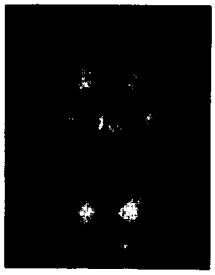
[5] M. Sakkinen, "Comments on the Law of Demeter and C++," SIGPlan Notices, pp.38-44, Dec. 1988.

[6] Rebecca Wirfs-Brock and Brian Wilerson, "Object-Oriented Design : A Responsibility-Driven Approack," Proc. OOPSLA '89, pp.71-75, 1989.

[7] Robert C. Sharble and Samuel S. Cohen, The Object-Oriented Brewery:A Comparison of Two Object-Oriented Development Methods, ACM SIGSOFT SOFTWARE ENGINEERING NOTES, Vol.18, No.2, Apr. 1983.

[8] 梁海述, 辻野嘉宏, 都倉信樹 : "オブジェクト指向言語のクラスの品質評價について," 情報處理學會ソフトウェア工學研究會, 1991. 2.

[9] 梁海述, 辻野嘉宏, 都倉信樹 : "オブジェクト指向言語におけるクラス間結合度," 電子情報通信学会論文誌, Vol. J75-D-I, No.3, 1992. 3.



황 석 형

1991년 강원대학교 자연과학대학 전자계산학과 졸업(이학사)
1994년 日本 오사카대학 정보공학과 소프트웨어공학 전공(공학석사)
1994년~현재 日本 오사카대학 정보공학과 박사과정
관심분야 : 소프트웨어공학(특히, 객체지향 분석과 설계 및 프로그래밍, User-Interface)

이 용 근

1988년 강원대학교 자연과학대학 전자계산학과 졸업(이학사)
1994년 강원대학교 전자계산학과 소프트웨어공학 전공(이학석사)
1989년~1992년 강원대학교 전자계산학과 조교
1994년~현재 한림전문대학, 서울산업대학교 전자계산학과 강사
관심분야 : 소프트웨어공학(특히, 소프트웨어 품질보증과 품질평가, 객체지향 프로그래밍, 객체지향 분석과 설계)

양 해 술

1975년 홍익대학교 공과 대학 졸업(학사)
1978년 성균관대학교 정보처리학과 정보처리 전공(석사)
1991년 日本 오사카대학 기초공학부 정보공학과 소프트웨어공학 전공(공학박사)
1975년~79년 육군중앙경리단 전자계산실 근무
1984년~92년 성균관대학교 경영대학원 강사
1986년~87년 日本 오사카대학 객원연구원
1994년~현재 한국산업표준원 이사
1994년~현재 한국정보과학회 학회지 편집부위원장
1994년~현재 한국정보처리응용학회 논문편집위원장
1980년~현재 강원대학교 전자계산학과 교수
관심분야 : 소프트웨어 공학(특히, S/W 품질보증과 평가, SA/SD, OOA/OOD/OOP, CASE), 소프트웨어 프로젝트관리.