영역 모델과 객체후보군의 유사도 측정에 관한 연구

박성옥, 노경주, 이문근
전북대학교 컴퓨터과학과

A Study on the Degree of Signature Similarity between Domain Model and Object Candidate Groups

Sung-ok Park, Kyoung-ju Noh, Moon-Kun Lee
Dept of Computer Science, Chonbuk National University

요 약

절차지향 소프트웨어를 객체지향 소프트웨어로 변환하는 여러 가지 방법이 존재한다. 프로그램을 변환하기 위하여 일반적으로 함수, 변수와 자료형들 간의 관계를 이용한다. 이들간의 관계성을 이용하면 결과로서 객체 후보가 생성된다. 생성된 객체 후보와 영역 전문가에 의하여 생성된 영역 모델을 비교하여 두 모델간의 유사성을 측정하여야 한다.

본 논문에서는 클래스의 시그너처(클래스 이름, 속성의 이름, 속성의 자료형, 메소드 이름, 메소드의 리턴 형, 메소드 파라미터의 자료형)을 이용하여 클래스와 객체 후보의 유사도를 측정하고, 측정된 유사도의 평균값을 이용하여 객체 후보군의 유사도를 측정한다. 기존의 연구 방법과는 다르게 n개의 클래스와 m개의 객체 후보사이의 구문적 측면의 유사도 측정뿐만이 아니라 의미적 측면의 유사도를 측정하는 방법을 제시하여 최적합 객체 후보군을 추출하도록 하였다.

1. 개요

소프트웨어는 제작된 후 지속적으로 유지·보수를 해야한다 개발된 시스템들은 사용자의 새로운 요구, 새로운 기술의 개발, 새로운 환경에 맞도록 수정 보완되어야 하며 이는 막대한 비용을 필요로 한다 그렇기 때문에 시스템을 재공학할 때 SW의 유지·보수 및 재사용적인 측면에서, 시스템을 기존의 방식들보다 유리한 객체지향 파라다임으로 재개발한다면 SW 생산성을 향상을 이룰 수 있다[1].

객체지향 파라다임으로 변환하기 위한 일반적인 방법으로서 함수, 자료형과 변수들간의 관계성을 이용하였다[2]. 이들간의 관계성을 이용하여 밀접하게 연관된 것들이 클러스터링되어 1개의 객체후보가 되고 객체 후보들이 모여 1개의 객체 후보군이 된다. 1개의 객체 후보는 최종적으로 1개의 클래스 후보가 되거나 상속 또는 포함 관계를 가지는 여러개의 클래스가 된다. 이러한 방법으로 추출된 클래스 후보들은 변환된 시스템의 프로토타입이 되거나 완성된 프로그램이 될 수 있다.

본 논문에서는 객체 후보군과 요구서(Requirement)로부터 영역 전문가에 의하여 생성된 영역 모델과의 유사성을 비교하는 방법에 대하여 논의한다 본 논문이 특히 잘 적용될 수 있는 경우는 여러개의 객체 후부군이 존재할 경우 어떠한 객체 후보군이 영역 모델과 가장 유사할 것이가를 판단할 경우이다

본 논문의 구성은 다음과 같다. 2장에서는 CORET 프로젝트에서 사용한 *osf*(Overall Similarity Factor)에 대하여 기술한다. 3장에서는 *DSS*(Degree of Signature Similarity)에 대하여 설명한다. 4장에서는 유사도 측정 결과를, 5장에서는 결론 및 향후 연구 과제에 대하여 기술한다.

본 연구는 한국과학재단 '97 핵심전문 연구과제(연구번호 · 971-0904-027-2)에서 지원 받았음

2. 관련연구

[3]에서는 *osf*(Object Similarity Factor)를 이용하여 유사도를 측정한다. *osf*는 시스템 요구서에 기반한 OMD(Object Model Design)의 클래스와 자료 흐름 분석에 의해 생성된 객체후보군 사이의 유사도를 클래스 시그너처에 기반 하여 수치로 표현한 것이다. 클래스의 시그너처는 클래스 이름(c), 속성의 이름(an), 속성의 형(at), 메소드의 리턴 형(or), 메소드 이름(on), 메소드의 파라미터 이름(opn), 메소드의 파라미터의 형(opt), 속성의 수(#a), 메소드의 수(#o)가 있다. *osf*는 1개의 클래스와 1개의 객체 후보사이의 유사도를 측정한 값이다.

*osf*계산의 세가지 단계를 거친다. 첫째, 각 클래스 시그너처에 해당하는 유사도(sf_c, sf_{an}, sf_{at}, sf_{or}, sf_{on}, sf_{opn}, sf_{opt})를 구한다. 이름에 관한 구문적 유사도 결정 시에는 이름에 대한 문자열을 부분 문자열로 나누어서 비교하는 퍼지 문자 비교(Fuzzy Text Comparison)방법을 사용하고, 타입에 관한 유사도는 전문가의 임의의 규칙에 의해 결정한다. 둘째, 각 클래스 시그너처에 대한 중요도나 가중치를 고려하여 가중치 벡터(w_c, w_{an}, w_{at}, w_{or}, w_{on}, w_{opn}, w_{opt})를 결정한다. 이때 각 가중치 벡터의 합은 0에서 1사이의 값을 갖는다. 마지막으로 위의 과정에서 도출된 유사도(similarity factor)와 가중치 벡터를 이용하여 전문가가 결정한 공식에 의해서 객체사이, 클래스 사이의 유사도가 결정된다.

*osf*의 측정은 프로그램의 의미적 측면을 반영하지 못하고 구문적 측면만을 기준으로 측정하고, m : n의

클래스와 객체를 비교하지 못하고 단지 1 : 1만의 비교만을 할 수 있다 이러한 단점을 본 논문에서는 해결하였다

3. 유사도 측정

유사도 측정의 목적은 추출된 객체 후보군들과 SW의 요구사항으로부터 영역 전문가에 의하여 구축된 영역모델과 유사도를 비교하여 적절한 객체후보군을 결정하는 것이다

먼저 영역 모델링의 결과와 객체 후보군과의 *DSS*를 구하는 방법에 대해서 기술하고 평균 유사도(ADSS : Average DSS)에 대하여 기술한다.

3.1 DSS

*DSS*는 [3]의 *osf*에 근거하고 있지만, 이를 위 비교에 적용하기 위하여 *osf*의 요소를 재정의하고 *n*개의 클래스와 *m*개의 객체들의 경우로 확장하였다 *DSS* 값을 구하는 공식은 다음과 같다:

$$DSS = sf_c * \left(\begin{array}{l} W_{an} * \left(\frac{\text{각 attribute의 최대 } sf_{an} \text{의 합}}{\text{총 attribute수}} \right) + \\ W_{at} * \left(\frac{\text{각 attribute의 최대 } sf_{at} \text{의 합}}{\text{총 attribute수}} \right) + \\ W_{mn} * \left(\frac{\text{각 method의 최대 } sf_{mn} \text{의 합}}{\text{총 method수}} \right) + \\ W_{mt} * \left(\frac{\text{각 method의 최대 } sf_{mt} \text{의 합}}{\text{총 method수}} \right) + \\ W_{mpt} * \left(\frac{\text{각 method의 최대 } sf_{mpt} \text{의 합}}{\text{총 method수}} \right) \end{array} \right)$$

여기에서, sf_c는 클래스 이름과 객체 이름과의 유사요인이며, *an*, *at*, *mn*, *mt*, *mpt*는 각각 속성의 이름, 속성의 자료형, 메소드 이름, 메소드의 리턴 형, 메소드의 파라미터 형 요인들을 의미하며, sf_{an}, sf_{at}, sf_{mn}, sf_{mpn} sf_{mpt}와 W_{an}, Wsf_{at}, W_{mn}, W_{mp}, W_{mpt}은 각각 요소들의 유사요인과 가중치 값을 의미한다.

각 유사요인에 대한 *sf*값은 각 클래스와 객체들간의 관련 요인의 구문적 측면과 의미적 측면의 유사성에 근거하여 영역전문가에 의하여 결정된다. 이들은 다음과 같은 4가지 조건을 만족하여야 한다 : 1) 가중치의 합은 1, 2) 각 객체후보에 대한 영역모델링의 클래스들과의 sf_c의 합은 $0 \le sf_c \le 1$, 3) 비교시 자료형은 이름에 종속, 4) 비교시 파라미터는 메소드에 종속된다는 조건들이다.

*osf*와 본 논문에서 제안한 방법과는 다음과 같은 차이점이 있다.

i) 구문적 의미적 유사성을 고려 : *osf*는 이름과 자료형에 의존한 구문적 측면만을 고려하므로 의미적 측면이 간과되기 쉽다 *DSS*의 공식에서는 클래스와 객체 후보 사이의 유사도는 속성과 메소드의 이름과 자료형에 따른 구문적 유사성과 전문가가 결정하는 의미적 측면에서의 sf_c를 이용하여 계산하게 된다. 이것은 전문가에게 의미적 유사도의 결정을 맡김으로써 의미적이거나 구문적 측면을 강조한 유사도가 결정 될 수 있다.

ii) 파라미터와 이름 시그네처의 차이 : *osf*는 메소드 유사도 측정 시 파라미터 이름에 대해 sf_{opn}이 계산된다. sf_{opn}은 파라미터 이름 생략이 많은 프로토타입단계에서 비교가 이루어지므로 완전히 같은 두 메소드에 대해 프로토타입에서 파라미터 이름이 존재하지 않을 경우에 *osf*는 1이 될 수 없다. *DSS*에서는 파라미터 이름에 관한 *sf*는 계산되지 않고 파라미터의 자료형에 대한 *sf*가 계산되는데 이러한 방법은 파라미터 자료형에 의해서 식별되는 객체지향언어의 메소드의 오버로딩(Overloading)등에서의 클래스 유사도를 구하는데 유용하다.

iii) *n*개의 클래스와 *m*개의 객체 후보의 비교 시 *osf*은 객체 후보나 클래스 비교 시 하나의 객체와 하나의 객체를 비교하는 일대일 관계에서 사용된다. 객체후보군에서 비교되는 하나의 객체와 하나의 객체사이에는 0과 1사이의 *osf*가 결정되므로 객체군의 *osf*의 합은 객체군의 객체후보의 수가 증가할수록 증가하게 된다 하지만 *DSS* 공식에 의해서 유사도 측정은 속성과 메소드에 대한 구문적 측면의 유사도와 영역 전문가에 의해 결정되는 의미적 측면의 sf_c에 의해서 결정된다. 하나의 클래스와 비교되는 모든 클래스와의 sf_c는 구조적 측면과 의미적 측면을 고려하여 영역 전문가에 의해서 총합이 0과 1사이의 값을 갖도록 결정된다. 따라서 하나의 객체후보와 *m*개의 클래스의 *DSS*의 합은 0과 1사이의 값으로, 하나의 클래스와 *n*개의 객체후보사이의 *DSS*의 합은 0과 1사이의 값으로 결정된다.

3.2 ADSS

*DSS*는 각 클래스가 각 결정요인에 대하여 얼마의 유사도가 있는지 종류별로 비례적 가중치에 따라 연산하고 이를 클래스와 객체의 단위적 유사요인에 따라 산출한 값이다. 따라서 영역 모델링의 *n*개의 클래스와 객체 후보군의 *m*개의 객체후보들에 대한 DSS는 $n \times m$ 행렬 *D*를 형성하며, *n*개의 클래스와 *m*개의 객체들간의 평균 *DSS*(ADSS : Average DSS)는 *D* 원소들의 합을 클래스의 수 *n* 또는 객체 후보의 수 *m*으로 나눈 값으로 다음과 같이 표현할 수 있다.

$$ADSS_{class} = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} d_{ij}}{\text{클래스 수}} \quad (d_{ij} \in D)$$

평균 유사도를 이용하여 1개의 객체 후보군과 영역 모델과의 유사도를 측정할 수 있다.

4. 유사도 측정 결과

[3]에서 사용한 예제의 결과 5개의 객체 후보군을 추출하였다. 각각의 객체 후보군에 대한 객체 후보의 수는 다음과 같다.

	객체후보의 수
후보군 1	1
후보군 2	2
후보군 3	3
후보군 4	4
후보군 5	5

사용한 예제의 영역 모델링 결과는 <그림 1>과 같이 3개의 클래스로 이루어졌다.

```
class_stack
    char from[20], to[20] ;
    int   dist, tos ;
public.
    void push(char *sfrom, char *sto; int sdist);
    void pop(char *sfrom, char *sto, int *sdist);
    int plus_tos(), minus_tos(), get_tos(), getdist();

class_FL
    char from[20], to[20], skip;
    int distance;
public.
  void setfrom_to_dis(char *sfrom, char *sto,int
sdistance),
  int getdistance(void), getskip(void);
  void setskip(void);
  int match(char *sfrom, char *sto);
  int find(char *sfrom, char *anywhere);

class_Manager
public.
    void isFlight(char *from, char *to, stack
        *btStack, Top &top, FL *flight, int fpos);
    void assertFlight(int *fpos);
```

<그림 1> 영역 모델링

DSS를 측정하기 위한 유사요인과 가중치는 다음과 같다.

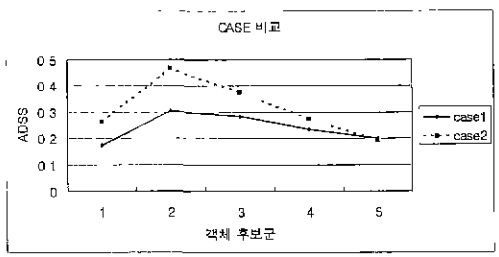
(W_{an}=0.2, W_{at}=0.2, W_{mn}=0.2, W_{mt}=0.2, W_{mpt}=0.2)

본 논문에서는 아래와 같은 2가지 경우에 대하여 유사도를 측정하였다.

CASE 1 : 영역 모델링의 클래스 기준으로한 *DSS*

CASE 2 : 객체 후보군의 객체 기준으로한 DSS

2가지 경우에 대한 객체 후보군의 *ADSS*는 <그림 2>과 같다.



<그림 2> Case 별 객체 후보군의 ADSS

위 결과를 비교 분석하여 보면 다음과 같은 사실을 알 수 있다:

1) CASE 별 각 객체 후보군의 평균 유사도 객체후보군에서 객체 후보의 수가 영역모델링의 클래스 수와 비슷할 때 높은 *DSS*의 값을 갖는 것으로 나타난다. 객체 후보군의 *DSS*의 총합을 영역 모델링의 클래스의 수로 나눈 경우(CASE 1), 객체 후보의 수로 나눈 경우(CASE 2) 평균 *DSS*의 값도 객체후보의 수가 영역모델링의 클래스 수와 비슷할 때 높은 값을 가지게 된다
2) CASE별 최고 유사도 객체군의 차이: 영역모델링의 클래스 수보다 하나 적은 객체후보군이 최대의 유사도를 지닌 것으로 나타난다. 이것은 객체 후보군의 구문적 유사도와 의미적 유사도를 동시에 고려한 *DSS*의 공식에 기인한다. 따라서 *DSS*의 공식에 의해서는 구문적인 측면뿐만 아니라 의미적 측면에서 영역 모델링과 비슷한 수준의 객체후보군이 가장 높은 유사도를 나타낸다고 할 수 있다

*DSS*의 방법은 클래스를 표현하는 클래스 시그네처에 기반으로 하여, 속성과 메소드의 이름과 자료형에 의한 구문적 측면과 기능이나 의미를 고려하여 영역 전문가가 결정하게되는 의미적 측면의 sf_c의해 유사도가 결정된다 이때 한 클래스와 비교되는 모든 객체 후보와의 sf_c의 합은 1이하의 값을 지니도록 결정되므로 n대 m의 객체비교 행렬 표현이 가능하다. 또 객체 후보군이 구조적 측면뿐만 아니라 의미적 측면에서 영역 모델링과 비슷한 수준을 지닐 때 클래스군과 객체군사이의 최대의 유사도를 나타내는 결과를 얻을 수 있다.

유사도를 측정한 후 가장 높은 유사도를 가진 것을 최적 객체 후보군으로 선택할 경우 사용자가 원하지 않은 객체 후보군일 수도 있다. 이는 영역 모델과 객체 후보군을 비교할 경우 객체 후보군의 객체 후보의 수와 영역 모델의 클래스의 수가 비슷할 경우 가장 높은 유사도를 가질 가능성이 많다. 그러나 사용자는 영역 모델의 클래스 수보다는 정제된 결과를 원할 수 있다. 따라서 최적 객체 후보군의 선택은 사용자가 유사도를 참조하여 최적 객체 후보군을 선택하도록 하는 것이 바람직하다.

5. 결론 및 향후 연구과제

소프트웨어의 유지·보수 비용을 줄이기 위하여 절차지향 소프트웨어를 객체지향 소프트웨어로 변환한다. 객체 지향 소프트웨어로 변환하기 위하여 절차지향 프로그램으로부터 객체 후보군을 추출하고 영역 모델링과 유사성을 측정하여 적절한 객체 후보군을 추출한다.

본 논문에서는 클래스의 시그너처(클래스 이름, 속성의 이름, 속성의 자료형, 메소드 이름, 메소드의 리턴 형, 메소드 파라미터의 자료형)을 이용하여 클래스와 객체 후보의 유사도를 측정하고, 측정된 유사도의 평균값을 이용하여 객체 후보군의 유사도를 측정한다. 기존의 연구 방법과는 다르게 n개의 클래스와 m개의 객체 후보사이의 구문적 측면의 유사도 측정뿐만이 아니라 의미적 측면의 유사도를 측정하는 방법을 제시하여 최적합 객체 후보군을 추출하도록 하였다.

[참고문헌]

[1] Robert S. Arnold, "Software Reengineering," IEEE Computer Society Press, 1994

[2] 박성옥, 이문근, "최적합 객체 선정을 위한 다중 객체 추출에 관한 연구," 한국소프트웨어공학 학술대회 발표 논문집, 제1권, 제1호, pp 90-100, 1999.

[3] Harald Gall and Johannes Weidl, "Binding Object Models to source Code An Approach to Object-Oriented Re-Architecturing," TUV-1841-87-14, 1998