Use Case 및 클래스의 가중치 분석에 의한 컴포넌트 추출 기법

(Component Extraction Method using Weight Analysis between Use Cases and Classes)

유 영 란 † 김 수 동 ††
(Young Ran Yoo) (Soo Dong Kim)

요 약 소프트웨어의 생산성과 유지보수 비용을 줄여줄 수 있는 기법으로 다양한 컴포넌트 기반의 개발 방법론이 제안되고 있다. 그러나 컴포넌트 기반의 시스템에서 재사용성과 독립성이 높은 컴포넌트의 식별은 가장 중요한 성공 요소 중의 하나임에도 불구하고, 대부분의 컴포넌트 기반 방법론들에서는 직관적이고 분석자의 경험에 의존적인 컴포넌트 식별 방법 만을 제공하고 있을 따름이다. 본 논문에서는 분석 단계의 산출물인 시스템의 기능 모델 Use Case 모델과 자료 모델인 클래스 모델에 기반 하여 체계적인 컴포넌트 식별 기법과 지침들을 제안한다. 먼저 클래스에 대한Use Case의 자료 접근값을 정의하고, 정의된 접근값을 기반으로 Use Case별로 접근되는 클래스의 가중치와 클래스별 동일 접근값을 가지는 Use Case들의 가중치를 계산한다. 두 가중치를 곱하여 최종적인 Use Case&클래스 가중치를 계산하여 후보 컴포넌트 식별의 기준으로 삼는다.

Abstract As the component has a better understanding as a reusable unit for software system, various component-based development (CBD) methodologies are proposed. Although good components that provide more reusability and undependability are one of the factors of success of component-based system, most of CBD methodologies only provide intuitive or heuristic approaches for component identification. A systematic method and instructions to identify components from the use case model and class model are proposed in this paper, those are artifacts of functional model and data structural model of analysis phase respectively First, access values of use case per class are defined, then the weight of class accessed per user case and the weight of use case that includes same access type per class, based on defined access value between use cases and classes. the two calculated weights are multiplied to make final weight of 'use case&class', then thses values can be the factor to identified candidate components.

1. 서 론

소프트웨어 업계에서 성공이란 수행속도가 더 빠르고, 성능이 뛰어나면서도 저렴한 소프트웨어를 만드는 것이다. 소프트웨어는 시장의 요구에 부응하여 현재의 요구사항 혹은 장래의 요구사항에 대한 서비스를 지원하면서도, 수행 시 더 적은 오류를 포함하고 생산 및 운영 비용이 저렴해야만 한다[11]. 이러한 성공 요소들의 관점에서 컴포넌트 기반의 개발(Component-Based Development: CBD)는 학계와 업계의 관심을 동시에 받고 있다. 그러한 추세가 반영되어 소프트웨어의 품질을 보장하고 재사용성을 확보하기 위한 여러 CBD 방법론들이 제안되었다[5, 6, 7, 8, 9, 16, 17, 18].

한편, 좋은 컴포넌트가 CBD 기반의 시스템의 성공과 성능을 좌우하는 핵심 요소라고 할 때, 컴포넌트의 식별과 추출은 가장 기초적이며 중요한 작업이라고 할 수 있다. 그러나 기존의 제안된 대부분의 CBD방법론들은 컴포넌트의 추출이나 식별보다는 컴포넌트의 명세나 모델링에 치우쳐져서 실제 중요한 컴포넌트 식별이 간과되고 있는

† 비 회 원 : TNG 정보기술 의료정보연구소 연구원
yuandyu@selab.soongsil.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
sdkim@computing.soongsil.ac.kr
논문접수 : 2000년 12월 19일
심사완료 : 2001년 6월 8일

실정이다.

따라서 본 논문에서는 분석 과정에서 산출된 Use Case 모델과 클래스 모델에 기반 하여 상호 가중치 계산에 근거한 컴포넌트 추출 기법을 제안한다. 본 논문은, 2장에서 관련 연구로서 대표적인 CBD 방법론인 Catalysis와 본 논문에서 제안된 기법의 이전 버전인 COMO 방법론의 컴포넌트 식별 작업과 장단점을 살펴본다. 3장에서는 Use Case와 클래스의 가중치 계산에 의한 컴포넌트 식별 기법을 제안하고 설명한다. 본 기법의 전제 조건들과 계층 모델을 설명하고, 구체적인 프로세스를 단계적으로 설명한다. 4장에서는 제안된 컴포넌트 패턴을 은행의 일부 업무에 적용한 사례 연구와 그 평가 결과를 서술하고, 마지막 장에서 본 논문의 결론과 향후 연구 방향을 제시하고자 한다.

2. 관련 연구

2.1 Catalysis

Desmond F. D'Souza 등에 의해서 제안된 Catalysis는 1991년에 시작된 OMT의 정형화 작업에 그 뿌리를 두고 있다. Catalysis는 비즈니스 모델로부터 소스 코드에까지 이르는 전 과정에서의 추적성을 보장하고, 불명확한 모델이나 문서를 최대한 명확하도록 만드는 정확성, 컴포넌트 기반의 개발, 코드뿐 만 아니라, 분석이나 디자인의 산출물들까지 확대된 재사용 개념, 확장성, 유연한 프로세스를 주요 특징으로 하고 있다.

Catalysis는 각각 협력(Collaboration), 타입(Type), 정제(Refinement)의 3 가지의 모델링 개념에 기초하여 구성되어 있다. 협력은 연관된 객체 들간의 상호연동으로 설명할 수 있는데, UML 1.3에서는 Use Case를 협력으로 실현된다. 협력은 특정 서비스를 제공해 주기 위하여 수행되는 작업들과 그 작업에 참여하는 객체들로서 이루어지는데, 협력은 큰 수준에서 점차 낮은 수준으로 세분화 될 수 있다. Catalysis는 협력 모델을 통해서 대상 시스템 혹은 도메인의 기능들을 큰 수준에서 작은 수준으로 명세 한다. Catalysis 방법론에서는 협력이 후보 컴포넌트의 기준이 되는 것이다 [5].

Catalysis는 모델 요소와 프로세스 패턴 등의 다양한 좋은 기법들을 사용하고 있지만, 그 내용들이 많은 경험을 요구하고, 특히 컴포넌트 식별 작업의 경우, 그 절차가 개념적이고, 분석자의 직관에 많이 의존된다고 할 수 있다.

2.2 COMO 방법론의 컴포넌트 추출 기법

2.2.1 Use Case/클래스 맵핑에 의한 클러스터링 기법

COMO 방법론에서는 컴포넌트 식별 작업을 두 가지로 제안하고 있다. Use Case 간의 관계 클러스터링을 통한 식별과 Use Case와 클래스 간의 관계 클러스터링을 통한 식별이 그것이다. 특히 COMO에서는 Use Case/클래스 행렬(Matrix)을 통한 클러스터링에 초점을 맞추어 컴포넌트 식별 작업을 설명하고 있다. Use Case/클래스 행렬은 Use Case와 클래스간의 관계를 접근하는 성격에 따라 생성(Create), 수정(Write), 삭제(Delete), 참조(Read)의 4가지로 나누어 정의한다. 두 가지 이상의 관계가 가능한 경우 우선 순위가 높은 관계로 표현된다. 우선 순위는 생성, 삭제, 수정, 참조의 순이다 (표 1 참조).

표 1 Use Case/클래스 행렬

	Class 1	Class 2	Class 3	Class 4
Use Case 1	C	R	C	R
Use Case 2	W	R	C	C
Use Case 3	R	C	R	R

Use Case/클래스 행렬이 생성된 후, 그림 1의 클러스터링 알고리즘에 의해 클러스터링 작업을 행렬에 적용한다. 표 1의 행렬이 클러스터링 된 결과가 표 2와 같다. 결과로 나온 행렬은 C와 D, W가 좌측 상단부터 우측 하단으로 정렬되게 된다. 변환된 행렬의 Use Case와 클래스들을 박

```
int init_C, init_R, end_C, end_R = 1;
int n = #of classes;
int m = #of use cases;
int disjPoint_C[n], disjPoint[m], index;
int i,j,k,x,y;
boolean disjFlag,

disjPointC[0] = 0;
disjPointR[0] = 0;
index = 0;

while((init_C < n && init_R < m) | (end_C<n | end_R<m)) {
  for ( i = end_C; i ≤ n; i++) {
    if (M(init_R, i) == 'C' | 'D' | 'W') {
      temp = Column(end_C);
      Column(end_C) = Column(i);
      Column(i) = temp,
      end_C++;
    }
  }
  for (j = init_C; j < end_C; j++) {
    for (k = end_R; k ≤ m; k++) {
      if (M(k, j) == 'C' | 'D' | 'W') {
        temp = Row(end_R);
        Row(end_R) = Row(k);
        Row(k) = temp;
        end_R++;
      }
    }
  }

disjFlag = True;
//disjoint point checking
for(y=init_R+1; y<end_R && disjFlag=True; y++) {
  for(x=end_C; x<=n && disjFlag=True; x++) {
    if(M(y,x) == 'C' | 'D' | 'W')
      disjFlag = False;
  }
}
if(disjFlag == True) {
  disjPointC[index] = end_C-1;
  disjPointR[index] = end_R-1;
  init_C = end_C;
  init_R = end_R;
  index++;
}
else {
  init_C = (init_C == n ? n : init_C+1);
  init_R = (init_R == m ? m : m(_R+1);
}
}
(Note)
    Init_C: Start location of column
    end_C: End location of column
    init_R: Start location of row
    end_R: End location of row
    C: "Create", D: "Delets", W: "Write"
    Column(end_C) : Column of current end_C
    Row(end_R): Row of current end_R
    Number of disjoint = number of index.
```

그림 1 클러스터링 알고리즘

표 2 클러스터된 Use Case/클래스 행렬

	Class 1	Class 2	Class 3	Class 4
Use Case 1	C	C	R	R
Use Case 2	W	C	C	R
Use Case 3	R	R	R	C

스 형태로 그룹 지어 후보 컴포넌트를 추출한다. 박스 형태의 그룹들은 서로 중복될 수 없다 [16].

COMO에서 제안된 기법에 의하여, 하나 이상의 클래스와 Use Case가 컴포넌트에 할당되며 한 클래스나 Use Case가 둘 이상의 컴포넌트에 동시에 할당될 수 없다.

2.2.2 문제점

COMO에서 제안된 기법은 Use Case와 클래스를 동시에 고려하여 컴포넌트를 식별하는 기법이란 점에서 장점을 가지지만, 실무에 적용 시, 몇 가지 문제점들을 가지고 있다.

첫째, 공통적인 성격이 강한 클래스의 경우 많은 Use Case들로부터 참조 됨으로써, 컴포넌트를 지나치게 크게 만들 우려가 있다. 물론 이 문제는 Use Case를 세분화 하여 《include》나 《extend》의 관계로 묶을 경우 어느 정도 해결된다. 그러나 클러스터링 기법이 모든 컴포넌트가 수직적(Vertical)으로 독립적인 관계를 전제로 하기 때문에 공통적인 부분들에 대해서는 약점을 가진다. 둘째, 생성과 수정, 삭제가 같은 비중으로 반영되기 때문에 미묘한 가중치를 반영하기 어렵다. 즉 동일한 클래스에 대해 생성, 수정, 삭제 관계를 가진다면, 무조건 같은 컴포넌트로 묶이게 된다. 셋째, 참조 만 하는 Use Case나 참조만 당하는 클래스의 경우 적절한 세부 컴포넌트 할당 지침이 요구된다. 현재는 클러스터링 알고리즘에 반영이 안되고, 직관에 의해 할당된다. 넷째, 시스템의 특성이 반영되지 않았다. 시스템 영역의 특성에 대한 고려 없이 일률적인 알고리즘 적용은 실무 적용 시, 많은 문제들을 야기시킬 수 있다. 따라서 시스템의 특성을 반영할 수 있는 인자(Factor)가 필요하다.

본 논문에서는 이상의 문제점들을 해결하기 위하여 Use Case/클래스 접근표를 기반으로 가중치를 반영하는 기법을 제안한다.

3. Use Case&클래스 가중치 산출 기법

Use Case&클래스의 가중치를 계산하기 위하여 Use Case/클래스 접근표부터 최종 결과물인 Use Case&클래스 가중치 테이블까지의 산출 과정은 그림 2의 흐름도와 같이 진행된다.

클래스 별로 Use Case의 접근값을 정의하고 접근값 표, V 행렬을 작성한다. V 행렬에 기반하여 Use Case 별 클래스 가중치 표, A 행렬과 클래스 동일 접근자 별 Use Case 가중치 표, B 행렬을 계산한 후, A 행렬의 값과 B 행렬의 값을 곱하여 Use Case&클래스 가중치, W 행렬을 계산한다.

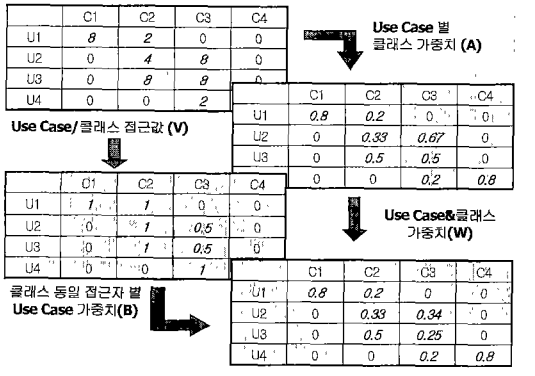

그림 2 Use Case&클래스 가중치 산출 흐름도

A 행렬(Use Case 별 클래스 가중치)의 값은 Use Case의 총 접근값으로 각 클래스의 접근값을 나누어서 해당 클래스에 대한 Use Case의 가중치를 계산한다 [3]. B 행렬(클래스 동일 접근자 별 Use Case 가중치)에서는 각 클래스의 동일 접근자끼리의 접근값의 가중치를 계산한다. 동일한 클래스에 대한 동일한 접근자가 많을 경우 각 접근자의 가중치는 동일한 접근자가 적은 경우에 비해 줄어들게 된다는 가정에 의해 산출된다. 마지막으로 W 행렬(Use Case&클래스 가중치)는 A 행렬과 B 행렬을 곱하여 최종적인 각 접근자에 대한 가중치를 계산한다. 결국 A 행렬의 자료가 B 행렬의 값에 의해 세분된다. 동일한 Use Case가 생성하는 두 클래스라 할지라도 B 행렬의 값에 의해 가중치가 달라지게 되는 것이다.

4. 컴포넌트 식별 프로세스

4.1 전제 조건

본 논문에서 제안한 프로세스는 분석 과정의 마지막 컴포넌트 식별 부분으로서, 그 이전 과정에서 작업 산출물로 Use Case 모델과 클래스 모델이 생성되었다고 가정한다. 관련 연구의 사례에서, 기존의 CBD 방법론들은 개념적인 Use Case나 혹은 상위 수준의 클래스를 후보 컴포넌트로 선정함으로써, 한 컴포넌트 내에 중복된 기능들이나 자료가 들어가는 것을 발견하여 제거해야 하는 부담이 있었다. 따라서 본 논문에서는 분석 과정에서 생성된 Use Case들과 클래스들을 모두 반영하여 서로 다른 컴포넌트에 중복해서 포함되는 경우를 피하도록 한다.

4.2 컴포넌트 계층 구조

본 작업에서는 기본적으로 컴포넌트 기반의 시스템이 공통적인 최하위 계층과 중간 업무 계층, 최상위 참조 계층으로 나누었다(그림 3). 물론 시스템은 그 이상의 계층

을 가질 수 있지만, 여러 컴포넌트들로부터 참조되어질 수 있는 공통적인 부분과 단순히 다른 컴포넌트들의 자료를 참조하여 가공만 하는 응용 컴포넌트를 일반적인 업무 컴포넌트들로부터 분리하였다.

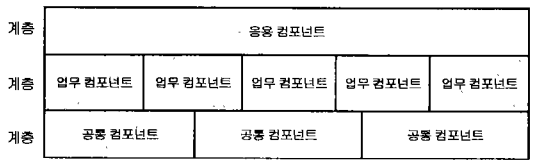

그림 3 컴포넌트 계층 구조

그림 3에서 보여지는 계층은 컴포넌트의 개념적인 계층으로 *공통 컴포넌트*는 다른 컴포넌트에 의해 공통으로 사용되는 확률이 높은 컴포넌트로서 다른 업무용 혹은 응용 컴포넌트로부터 의존관계를 가진다. *업무 컴포넌트*는 공통적인 성격이 약하면서 독립적인 자료 구조를 가지는 컴포넌트이다. *응용 컴포넌트*는 나름의 자료구조를 가지지 않으면서 적용되는 시스템의 특징과 요구에 의해 요청되는 기능들을 모은 컴포넌트이다. 응용 컴포넌트는 별도의 클러스터링을 통하여 분리할 필요가 없다. 왜냐하면 독립적인 자료구조를 가지지 않고, 시스템 의존적인 성격이 강하기 때문이다. 개발된 컴포넌트가 동일한 영역의 다른 시스템에 적용될 경우, 기본 컴포넌트와 업무 컴포넌트는 재사용성이 높으며 응용 컴포넌트는 시스템에 따라 많은 차이를 가질 수 있다.

제안된 기법은 전체 Use Case 모델과 클래스 모델로부터 응용 계층과 공통 계층에 할당될 구성 요소들을 제외한 후, 업무 계층을 중심으로 Use case&클래스의 가중치를 계산하도록 되어 있다. 이 기법은 나중에 공통 계층에도 적용될 수 있다. 그러나 본 논문에서는 업무 계층에 한하여 적용한 결과 만을 포함시켰다.

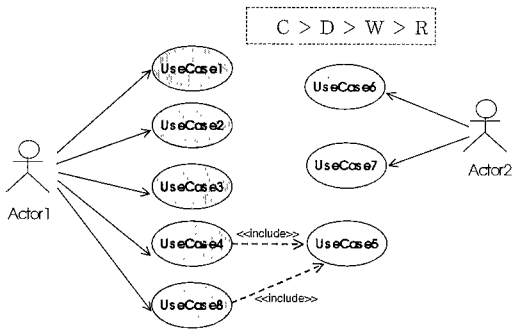

그림 4 Use Case 다이어그램의 예

4.2.1 Use Case/클래스 접근표 작성

분석 과정에서 그림 4과 같은 Use Case가 나왔다고 가정한다. 추출된 Use Case와 클래스에 대해서 접근권한을 표로 만든 것이 표 3의 Use Case/클래스 접근표와 같다. 생성의 경우 C(Create), 삭제는 D(Delete), 수정은 W(Write), 참조는 R(Read)로 약자를 정하고, 각 접근이 중복적인 경우에는 상위의 접근자를 취한다. 접근자의 순서는 다음과 같다.

표 3 Use Case/클래스 접근표의 예

클래스 / Use Case	Class1	Class2	Class3	Class4	Class5	Class6
UseCase1	C	R	W			R
UseCase2	R	R	C	C		
UseCase3	R	C				C
UseCase4	R			R	D	
UseCase5				R	C	
UseCase6			R	R		
UseCase7	R	R			R	R

4.2.2 응용 Use Case 식별

Use Case/클래스 접근표로부터 클래스들을 참조만 하는 응용 Use Case들을 식별하여 추출한다. 표 3의 예에서는 UseCase6과 UseCase7이 응용 Use Case에 해당한다. 이후 작업에서는 응용 Use Case가 제외된다.

표 4 응용 Use Case의 식별

클래스 / Use Case	Class1	Class2	Class3	Class4	Class5	Class6
UseCase1	C	R	W			R
UseCase2	R	R	C	C		
UseCase3	R	C				C
UseCase4	R			R	D	
UseCase5				R	C	
UseCase6			R	R		
UseCase7	R	R			R	R
UseCase8	R	R	W	D	D	

4.2.3 공통 Use Case와 클래스 식별

Use Case 다이어그램으로부터 공통적으로 사용되는 공통 Use Case들을 식별한다. 공통 Use Case에는 두 가지가 있는데 하나는 라이브러리 성격의 Use Case로서, 자료를 직접 접근하지는 않고 로직 만을 제공하는 경우이고, 다른 하나는 하나 이상의 다른 Use Case들로부터 재사용

되는 Use Case이다. 라이브러리 Use Case의 경우 접근하는 클래스를 하나도 가지지 않는다.

그림 4의 예에서는 라이브러리 Use Case는 가지지 않고, 후자의 공통 Use Case로서 Use Case5가 식별된다. 공통 Use Case인 Use Case5에 의해 생성되거나 삭제되거나 수정되는 클래스는 Class5이며, 기본 클래스인 Class5는 Use Case4에 의해 생성될 수 있으므로 비배타적 기본 클래스에 해당된다. 이후 작업에서는 공통 Use Case와 배타적 기본 클래스가 제외된다.

표 5 기본 클래스 및 Use Case의 식별

클래스 / Use Case	Class1	Class2	Class3	Class4	Class5	Class6
UseCase1	C	R	W			R
UseCase2	R	R	C	C		
UseCase3	R	C				C
UseCase4	R			R	D	
UseCase5				R	C	
UseCase8	R	R	W	D	D	

4.2.4 Use Case&클래스 가중치 계산

Use Case/클래스의 가중치 계산은 네 단계의 작업으로 나누어진다. 먼저 Use Case/클래스 접근표의 접근권한에 따라 접근값(Access Value)을 부여한다. 이 접근값은 시스템에 따라 가변적이기 때문에 각각의 접근자에 대한 접근값을 다음과 같이 변수화 한다.

c : C, d : D, w : W, r : R

각 변수는 적용 시스템에서 어떠한 접근자를 비중있게 볼 것인가에 따라 그 값들이 차등 지어진다. 만약 시스템에서 수정이나 삭제를 생성과 동일한 비중으로 다루고자 하는경우 동일한 값을 변수에 부여하면 된다. COMO에서 제안한 알고리즘은 생성과 수정 및 삭제가 동일한 변수를 가진 경우에 해당한다.

본 예제에서 c, d, w, r의 값을 각각 9, 6, 4, 1으로 가정했을 때, Use Case/클래스의 접근값는 표 6과 같다. 이 결과를 V 행렬로 정의한다. 각 Use Case/클래스 접근값 V_{ij}는 다음과 같이 표현할 수 있다.

V_{ij} = x, x ∈{c, d, w, r}, $0 \le i \le n$, $0 \le j \le m$

(n: Use Case의 수, m : 클래스의 수)

표 6 Use Case/클래스 접근값 (V 행렬)

클래스 / Use Case	Class1	Class2	Class3	Class4	Class5	Class6
UseCase1	9	1	4			1
UseCase2	1	1	9	9		
UseCase3	1	9				9
UseCase4	1			1	6	
UseCase8	1	1	4	6	6	

두 번째 작업은 Use Case 별 클래스 가중치를 계산한다. 각 클래스별 접근값을 Use Case별 전체 접근값으로 나눈다. Use Case 별 클래스 가중치를 A 행렬로 정의하고 다음과 같이 정의할 수 있다.

$$A_{ij} = V_{ij} \div \sum_{0 \le j \le m} V_{ij},\ 0 \le i \le n,\ 0 \le j \le m$$

(n: Use Case의 수, m : 클래스의 수)

수식으로 계산된 예가 표 9과 같다. 동일한 Use Case 내에서 각 클래스가 차지하는 비중이 표현된다. 값이 클수록 Use Case와 클래스의 친밀도는 크다고 할 수 있다. 그러나 A 행렬 값에서는 아직 미비점이 있다. 바로 한 Use Case의 동일한 접근자는 동일한 가중치를 가진다는 것이다. 이 문제를 다음의 작업들을 통해서 보정한다.

표 7 Use Case 별 클래스 가중치 (A 행렬)

클래스 / Use Case	Class1	Class2	Class3	Class4	Class5	Class6
UseCase1	0.59	0.07	0.27	0	0	0.07
UseCase2	0.05	0.05	0.45	0.45	0	0
UseCase3	0.06	0.47	0	0	0	0.47
UseCase4	0.13	0	0	0.13	0.74	0
UseCase8	0.06	0.06	0.22	0.13	0.33	0

세 번째 작업은 클래스 동일 접근자 별 Use Case의 가중치를 계산하는 것이다. 최종적인 가중치인 Use Case 별 클래스 가중치를 계산하기 전에, 클래스 별 Use Case의 가중치를 계산하는 이유는, 한 클래스에 대해서 동일한 접근자가 많은 경우, 각 Use Case의 비중은 낮아진다고 볼 수 있다. 즉 한 클래스에 대해서 생성 접근자를 가지는 Use Case가 단 하나 뿐이라면, 그 Use Case는 해당 클래스와의 응집도(Coherence)가 높다고 볼 수 있다. 그러나 반대로 한 클래스에 대해서 다수의 Use Case가 생성 접근자를 갖는다면 각각의 Use Case는 한 Use Case만 있는 경우에 비해 클래스와 높은 응집도를 가진다고 보기 어렵다. 따라서 동일한 접근자 별로, Use Case의 가중치를 계산하는 것이다.

클래스 별 Use Case의 가중치는 클래스 별로 동일한 접근자를 가지는 Use Case의 수로 1을 나눈 값이 된다. 수식으로 표현하면 다음과 같다.

1/(같은 클래스에 대한 동일한 전체 접근자 수)

Class1의 경우 생성 접근자를 갖는 Use Case는 1개이고, 조회 접근자를 갖는 Use Case는 4개이므로 생성 접근

자의 경우 1/1=1을, 조회 접근자의 경우 1/4=0.25를 가지게 된다. 이런 식으로 계산된 예가 표 8과 같다. 클래스 별 Use Case 가중치 다음 작업인 Use Case 별 클래스의 가중치 계산 시, 동일한 Use Case에 의한 동일한 접근자라도 클래스 별로 차별화 될 수 있도록 한다. 즉 해당 클래스에 대한 접근자의 가중치를 다시 세분화 하도록 한다.

본 예제에서 클래스 별 Use Case 가중치는 표 8과 같다. 이 결과를 B 행렬로 정의하고 다음과 같이 표현할 수 있다. B 행렬의 각 항목의 값은 0과 1 사이이며, 클래스별 전체 값은 관련된 접근자의 종류에 따라 1, 2, 3, 4의 값을 가질 수 있다.

$$B_{ij} = 1 \div \sum_{0 \le i \le n} (V_{ij}\text{와 동일한 접근자}),\ 0 \le i \le n,\ 0 \le j \le m$$

(n: Use Case의 수, m : 클래스의 수)

표 8 클래스의 동일 접근자 별 Use Case 가중치 (B 행렬)

Use Case \ 클래스	Class1	Class2	Class3	Class4	Class5	Class6
UseCase1	1	0.33	0.5			1
UseCase2	0.25	0.33	1	1		
UseCase3	0.25	1				1
UseCase4	0.25			1	0.5	
UseCase8	0.25	0.33	0.5	1	0.5	

마지막 작업은 Use Case 별 클래스의 가중치 A 행렬과 클래스 동일 접근자 별 Use Case의 가중치 B 행렬의 값을 곱한 값이다. Use Case&클래스 가중치를 W 행렬로 정의하고 다음과 같이 정의할 수 있다.

$$Wij = Aij * Bij,\ 0 \le i \le n,\ 0, \le j \le m$$

(**n: Use Case**의 수, m : 클래스의 수)

수식으로 계산된 예가 표 9과 같다. 클래스 별 Use Case의 가중치에 의해, UseCase8은 클래스 Class4와 Class5에 대해 동일한 삭제 접근자를 갖지만, 가중치는 달라진다. 즉 동일한 접근자를 갖는다고 동일한 가중치를 가지는 것이 아니라 그 접근이 해당 클래스에 얼마나 결정적이냐에 따라 값에 차이가 발생하게 되는 것이다.

표 9 Use Case 별 클래스 가중치 (W 행렬)

Use Case \ 클래스	Class1	Class2	Class3	Class4	Class5	Class6
UseCase1	0.6	0.02	0.13	0	0	0.07
UseCase2	0.01	0.02	0.45	0.45	0	0
UseCase3	0.01	0.47	0	0	0	0.47
UseCase4	0.03	0	0	0.13	0.38	0
UseCase8	0.01	0.02	0.11	0.33	0.17	0

4.2.5 업무 컴포넌트 식별

본 작업에서는 표 9의 가중치에 적용하여 후보 컴포넌트를 추출할 기준이 되는 적절한 *한계 가중치*(Limit of Weight: *LW*) 값을 찾는 것이다. 분석자가 임의의 한계 가중치를 설정하여 후보 컴포넌트를 추출한다. 한계 가중치는 업무의 특징이나 시스템의 규모에 따라 영향을 받게 된다. 한계 가중치 이상의 가중치를 가지는 Use Case들을 해당 클래스 별로 목록을 만든다.

한계 가중치 *LW*는 이론 적으로 0부터 1까지의 값을 가질 수 있다. 본 예와 다른 예들을 통하여 *LW*의 값과 식별된 컴포넌트와의 상관 관계를 살펴보면 표 10와 같다. *LW*의 값은 0부터 1사이 값 중, 표 9의 가능한 *LW* 값들을 적용하였다. 컴포넌트의 크기는 포함된 Use Case와 클래스 수를 더한 값으로 정의하였다.

표 10 한계 가중치 LW와 식별된 컴포넌트의 상관 관계

LW 값	컴포넌트 수	평균 컴포넌트 크기
0.01	1	11.0
0.02	1	11.0
0.03	1	11.0
0.06	1	11.0
0.11	2	5.5
0.13	2	5.5
0.17	2	5.5
0.33	3	3.67
0.38	4	2.75
0.45	3	2.67
0.47	2	2.25
0.6	1	2

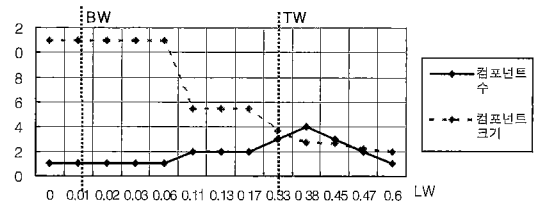

그림 5 LW와 컴포넌트 식별 상관 관계

표 10의 수치를 그래프로 표현하면 그림 5과 같다. 그림 7에서 *LW*가 어느 값 이하가 되면 컴포넌트가 하나로 뭉쳐서 식별되고 어느 값 이상이 되면 식별되는 컴포넌트의 수가 오히려 줄어들다가 결국 아예 식별이 불가능해짐을 알 수 있다. 그러한 최저값과 최고값을 각각 최저 가중치

(Bottom of Weight: BW)와 최고 가중치(Top of Weight: TW)로 정의하고 그래프에서 식별한 것이 그림 7에 표현되어 있다.

*LW*의 값이 작을수록, 식별된 컴포넌트의 수는 줄어들면서 대신 컴포넌트의 크기는 커지게 된다. 반대로 *LW*의 값이 클수록 식별되는 컴포넌트의 수는 늘어나면서 그 크기는 줄어들게 된다. 그리고 *LW*의 값이 BW보다 작은 경우, 모든 Use Case와 클래스들은 하나의 컴포넌트로 묶이며, 반대로, TW 이상이 되면 Use Case와 클래스 중 일부가 컴포넌트 식별에 제외되면서, 1에 가까워지면 컴포넌트 식별이 불가능해진다. BW는 클래스 별 최소 가중치의 최소값에 해당하고, TW의 값은 Use Case&클래스 가중치 값 중 클래스 별로 최대 가중치와 Use Case별 최대 가중치의 최소값에 해당한다. 수식으로 표현하면 다음과 같다.

$$BW \leq LW \leq TW$$

본 예에서 한계 가중치 *LW*의 값을 0.33으로 잡을 때, 다음과 같은 클래스 별 컴포넌트 목록이 생길 수 있다.

표 11 클래스 별 Use Case 목록 (LW = 0.33)

Class1	Class2	Class3	Class4	Class5	Class6
UseCase1	UseCase3	UseCase2	UseCase2	UseCase4	UseCase3
			UseCase8		

표 11의 목록을 동일한 Use Case를 갖는 클래스별로 모으면 그 결과가 후보 컴포넌트가 되는 것이다. 예에서 식별된 컴포넌트는 표 12과 같다. 예에서는 4개의 업무 컴포넌트가 식별되었다. 표 13는 식별된 컴포넌트와 각각 할당된 Use Case 및 클래스의 목록 표이다.

표 12 후보 컴포넌트

Class1	Class2	Class6	Class3	Class4	Class5
UseCase1	UseCase3	UseCase3	UseCase2	UseCase2	UseCase4
				UseCase8	

표 13 컴포넌트별 Use Case 및 클래스 목록

컴포넌트 명	클래스 목록	Use Case 목록
컴포넌트 A	Class1	UseCase1
컴포넌트 B	Class2,Class6	UseCase3
컴포넌트 C	Class3,Class4	UseCase2,UseCase8
컴포넌트 D	Class5	UseCase4

본 작업의 가장 어려운 부분은 적절한 한계 가중치를 발견하는 일이다. 그리고 적절하게 접근값과 한계 가중치 변수를 조정하며 적절한 컴포넌트를 발견할 때까지 반복적인 작업이 요구된다. 본 예에서는 응용 Use Case인 UseCase6은 컴포넌트C에 속한 클래스 Class3과 Class4만을 참조하기 때문에 지침 1에 의해, 컴포넌트 C에 할당시킨다. 그러나 UseCase7의 경우 컴포넌트A, B, D에 속한 클래스들을 모두 참조하므로 별도의 컴포넌트 E를 만들어서 할당 시킨다. 이때 식별된 컴포넌트 E는 응용 컴포넌트에 속한다.

4.3 접근값과 가중치의 상관 관계

4.3.1 접근값 정의 지침

본 논문에서 제안된 기법의 입력 인자값인 생성과 삭제, 수정, 참조의 각 접근값을 어떻게 정의할 것이냐에 대해서 다음과 같은 지침들을 제안한다. 먼저 각 접근값은 표 14와 같이 0부터 10까지의 값을 가지며, 각 접근값의 합이 20이 되도록 한다. 이는 적정 한계 가중치를 사용하기 위한 기반이 된다.

표 14 접근값 정의 예

접근값 종류	코드	0	1	2	3	4	5	6	7	8	9	10
생성	C									○		
삭제	D							○				
수정	W					○						
참조	R			○								
합계		20										

4.3.2 접근값의 편차와 가중치의 관계

다양한 접근값의 가능성에 대한 가중치의 영향을 살펴보기 위해서 A, B, C의 3가지 경우의 접근 값에 대한 가중치 영향을 도출하였다. 표 6의 Use Case/클래스 접근값 표에 의한 각각의 접근값 경우가 표 15와 같다.

표 15 3가지 유형 별 접근값

접근값 유형	c	d	w	r	합계
A	10	10	0	0	20
B	8	6	4	2	20
C	5	5	5	5	20

3가지 유형의 가능한 LW 별 추출되는 컴포넌트의 수와 컴포넌트의 크기는 그림 6와 같다. 그림 6에 나타난 자료들을 가중치 중심으로 정리하면 표 18와 같다. A와 같이 생성과 삭제로 접근값이 집중된 경우, BW와 TW의 값이 같이 증가하면서 최소 식별 컴포넌트의 수는 증가하였다.

C와 같이 모두 동일한 접근값을 가지는 경우, BW와 TW가 감소하면서 컴포넌트 식별 가능한 LW의 범위도 줄어든다. 그리고 식별되는 컴포넌트의 수 또한 줄어든다. 즉 컴포넌트의 입자도(Granularity)가 증가하게 된다.

그림 6의 자료를 기준으로, 접근값이 4.3.1절의 지침과 같이 주어질 경우, 가능한 LW의 범위가 0.05~0.5 사이가 됨을 알 수 있다. 그리고 접근자의 값이 적절하게 분산되어 있을수록 다양한 입자도의 컴포넌트 식별이 가능해짐을 알 수 있다.

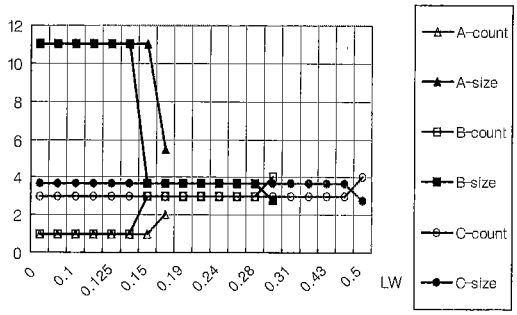

그림 6 유형 별 식별 컴포넌트 형태

표 16 3가지 유형 별 가중치 비교

유형	가중치 편차	BW	TW	LW 범위	최소 식별 컴포넌트	최대 식별 컴포넌트
A	5.77	0.25	0.5	0.25	3	4
B	0.25	0.025	0.3	0.275	1	4
C	0.5	0.05	0.167	0.117	1	2

4.4 Use Case/클래스 접근표 정제 지침

본 논문에서 제안된 기법은 잘 정의된 Use Case와 클래스를 전제로 하고 있다. 잘 정의되지 않고 정제되지 않은 Use Case와 클래스, 그리고 둘 간의 접근 관계는 좋은 컴포넌트 식별에 장애 요소가 된다. 예를 들어 어떤 클래스가 다수의 Use Case들로부터 생성 접근 관계에 있다면, A4 작업의 클래스의 동일 접근자별 Use Case 가중치 B 행렬의 값들이 낮아져서 생성 접근에 대한 가중치가 수정, 삭제, 혹은 참조 관계보다 낮아지는 부작용이 발생하게 된다. 이를 제거하기 위해서 B 행렬의 수식의 분자값을 증가시킬 경우, 앞서 정의한 접근값들에 영향을 준다. 따라서 다른 행렬에 영향을 주지 않고 부작용을 최소화 할 수 있는 방법은 Use Case/클래스 접근표를 정제하는 것이다. 표 17은 정제와 관련된 질문들과 해당 정제 지침 번호들이다.

표 17 Use Case/클래스 접근표 정제 지침

No.	질문	정제 지침 #
1	전혀 참조되지 않는 클래스가 존재하는가?	R1, R2
2	한번도 Create 되지 않는 클래스가 존재하는가?	R1, R2
3	한번도 Delete 되지 않는 클래스가 존재하는가?	R1, R2
4	한번도 Read 되지 않는 클래스가 존재하는가?	R1, R2
5	어떤 클래스도 참조하지 않는 Use Case가 존재하는가?	R2, R3
6	Create 하는 다수의 Use case를 가지는 클래스가 존재하는가?	R4, R5
7	Delete 하는 다수의 Use case를 가지는 클래스가 존재하는가?	R4, R5
8	다수의 클래스들을 Create 하는 Use Case가 존재하는가?	R5
9	다수의 클래스들을 Write 하는 Use Case가 존재하는가?	R5
10	다수의 클래스들을 Delete 하는 Use Case가 존재하는가?	R5
11	유사한 클래스들을 비슷한 접근값으로 접근하는 Use Case의 그룹이 존재하는가?	R5, R6

• R1: 누락된 Use Case를 식별한다.

해당 클래스를 접근하는 Use Case가 누락되지 않았는지 검사한다. 혹은 다른 Use Case에 문맥 상 포함되어 표현되지 않은 것은 아닌지 검사한다. 누락된 Use Case는 추가하고, 다른 Use Case에 포함된 경우는, 포함하는 Use Case가 둘 이상인 경우는 새로운 Use Case로 추출하고, 그렇지 않은 경우는 포함하는 Use Case의 접근자를 추가한다.

• R2: Use Case의 식별되지 않은 접근값을 찾는다.

Use Case의 접근자들을 살펴서 클래스에 대해 식별되지 않은 접근자가 없는지 검사한다.

• R3: 누락된 클래스를 추출한다.

도메인에서 누락된 클래스를 식별한다.

• R4: 클래스를 세분한다.

해당 클래스를 관계(Association), 상속 및 결합(Aggregation) 등의 관계를 이용하여 나눌 수 있는지 검토한다.

• R5: Use Case를 세분한다.

Use Case의 기능을 더 세분할 수 있는지 검토한다. 《include》 나 《extend》 의 스테레오타입(Stereo Type)을 이용하여 재사용되거나 확장되는 부분들을 추출한다.

• R6: 중복되는 부분들을 새로운 Use Case로 식별한다.

중복되는 작업이나 워크플로우가 없는지 검토한 후, 있을 경우, 새로운 Use Case로 추출하여 《include》 나 《extend》 의존관계로 연결한다.

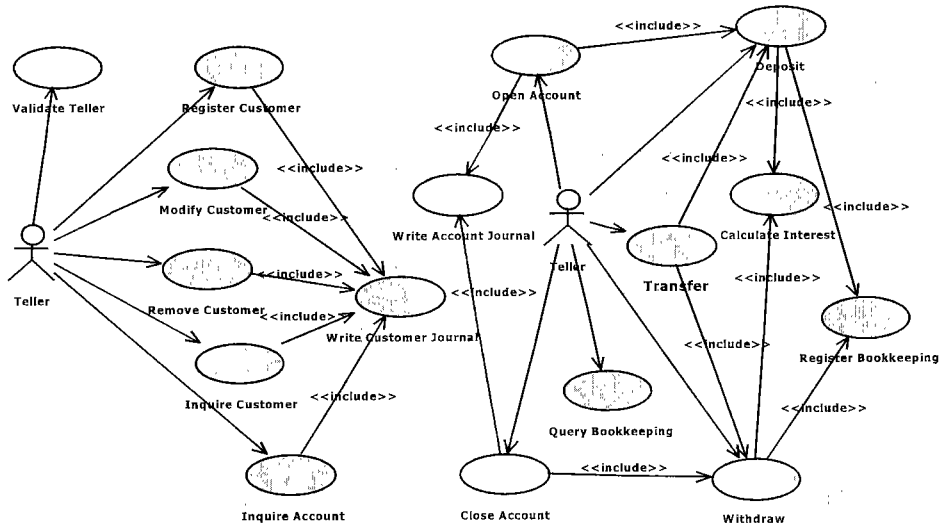

그림 7 Use Case 다이어그램

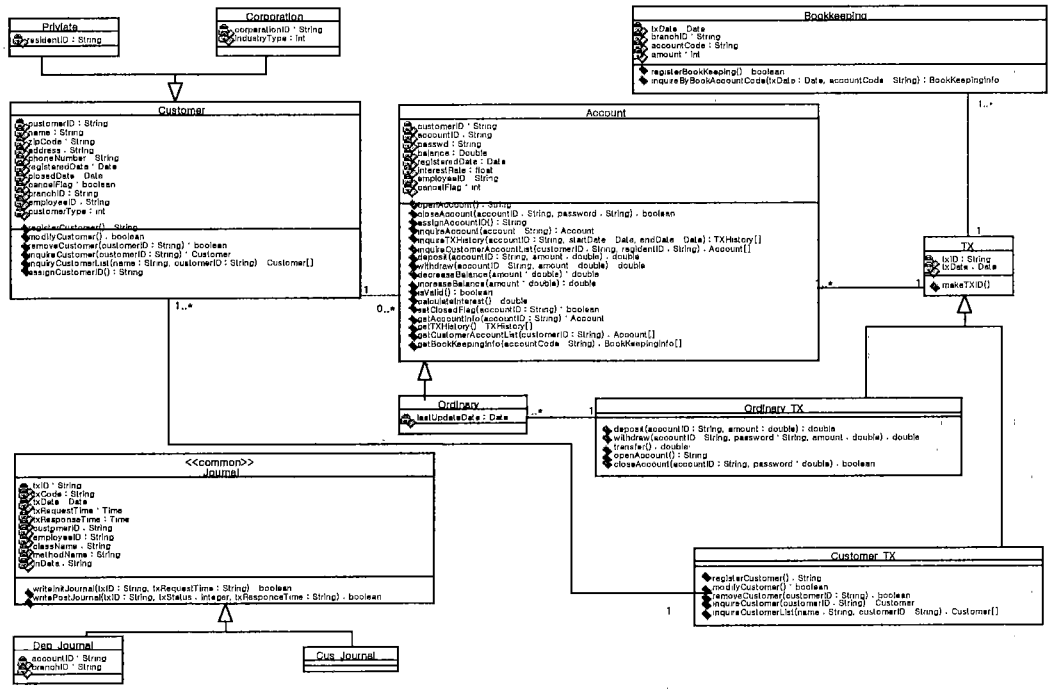

그림 8 클래스 다이어그램

5. 사례 연구

5.1 적용 분야 개론

제안된 기법의 사례 연구 시스템으로는 은행의 전체 업무 범위 중, 고객관리 및 수신관리 업무를 선택했다. 고객관리는 고객 등록부터 수정, 조회, 삭제까지의 업무를 포함하며, 수신관리는 입금, 출금, 이체 등의 업무를 관리한다. 각 업무는 각각의 저널(Journal)들에 의해 업무기록이 남게 되며 트랜잭션 관리가 공통적으로 포함된다. 분석 단계의 산출물로 Use Case 다이어그램과 클래스 다이어그램이 그림 7와 그림 8과 같이 생성되었다.

5.1.1 Use Case/클래스 접근표 작성

그림 7와 그림 8에서 식별된 Use Case들과 클래스들간의 접근자를 식별하여 표 18와 같이 Use Case/클래스 접근표로 작성하였다.

5.1.2 응용 Use Case 식별

표 18의 Use Case/클래스 접근표로부터 클래스들을 참조만 하는 응용 Use Case들을 식별하면 *Query Bookkeeping*과 *Calculate Interest* Use Case가 해당된다.

5.1.3 공통 Use Case와 클래스 식별

Use Case 다이어그램으로부터 공통적으로 사용되는 공통 Use Case들을 식별하면, *Register Bookkeeping*, *Register Customer Journal*, *Register Account Journal*의3 Use Case가 해당된다. 공통 Use Case로부터 식별된 기본 클래스는 *Bookkeeping*, *Cus_Journal*,

표 18 Use Case/클래스 접근표

Use Case \ 클래스	Private	Corporation	Account	Ordinary	Bookkeeping	Customer_Tx	Ordinary_Tx	Cus_Journal	Dep_Journal
Write Customer Journal								C	
Write Account Journal									C
Register Customer	C	C				C			
Modify Customer	W	W				C			
Remove Customer	W	W	R	R		C			
Inquire Customer	R	R				C			
Open Account			C	C			C		
Close Account			W	W			C		
Inquire Account			R	R			C		
Deposit				W			C		
Withdraw				W			C		
Transfer				W			C		
Calculate Interest				R					
Register Bookkeeping					C				
Query Bookkeeping					R				

표 19 식별된 응용 Use Case와 공통 Use Case 및 공통 클래스

Use Case \ 클래스	Private	Corporation	Account	Ordinary	Bookkeeping	Customer_Tx	Ordinary_Tx	Cus_Journal	Dep_Journal
Write Customer Journal								C	
Write Account Journal									C
Register Customer	C	C				C			
Modify Customer	W	W				C			
Remove Customer	W	W	R	R		C			
Inquire Customer	R	R				C			
Open Account			C	C			C		
Close Account			W	W			C		
Inquire Account			R	R			C		
Deposit				W			C		
Withdraw				W			C		
Transfer				W			C		
Calculate Interest				R					
Register Bookkeeping					C				
Query Bookkeeping					R				

*Dept_Journal*로서 모두 배타적 기본 클래스들이다.

A2의 응용 Use Case 식별과 A3의 공통 Use Case 및 클래스 식별 작업으로부터 식별되어 제외된 Use Case와 클래스는 표 19와 같이 나타난다.

5.1.4 Use Case/클래스 가중치 계산

본 사례 연구에서는 각 접근자에 대한 접근값을 다음과 같이 설정하였다. 정의된 접근값에 대한 Use Case/클래스 접근표는 표 20과 같다. Use Case 별 클래스 가중치는 4장에서 제안된 수식에 의해 계산하면 표 23과 같다. 클래스 동일 접근자 별Use Case 가중치는 표 22와 같다. 최종적인 Use Case&클래스 가중치는 표 20과 표 22의 접근값과 가중치로부터 계산하여 표 23와 같이 산출된다.

표 20 Use Case/클래스 접근값(V 행렬)

c = 9, d = 6, w = 4, r = 1

클래스 / Use Case	Private	Corporation	Account	Ordinary	Customer_Tx	Ordinary_Tx	합계
Register Customer	9	9			9		27
Modify Customer	4	4			9		17
Remove Customer	4	4	1	1	9		19
Inquire Customer	1	1			9		11
Open Account			9	9		9	27
Close Account			4	4		9	17
Inquire Account			1	1		9	11
Deposit				4		9	13
Withdraw				4		9	13
Transfer4						9	13

표 21 Use Case 별 클래스 가중치(A 행렬)

클래스 / Use Case	Private	Corporation	Account	Ordinary	Customer_Tx	Ordinary_Tx
Register Customer	0.33	0.33			0.34	
Modify Customer	0.24	0.24			0.52	
Remove Customer	0.21	0.21	0.05	0.05	0.48	
Inquire Customer	0.09	0.09			0.82	
Open Account			0.33	0.33		0.34
Close Account			0.24	0.24		0.52
Inquire Account			0.09	0.09		0.82
Deposit				0.31		0.69
Withdraw				0.31		0.69
Transfer				0.31		0.69

표 22 클래스 동일 접근자 별 Use Case의 가중치(B 행렬)

클래스 / Use Case	Private	Corporation	Account	Ordinary	Customer_Tx	Ordinary_Tx
Register Customer	1	1			0.25	
Modify Customer	0.5	0.5			0.25	
Remove Customer	0.5	0.5	0.5	0.5	0.25	
Inquire Customer	1	1			0.25	
Open Account			1	1		0.17
Close Account			1	0.25		0.17
Inquire Account			0.5	0.5		0.17
Deposit				0.25		0.17
Withdraw				0.25		0.17
Transfer				0.25		0.17

표 23 식별된 후보 컴포넌트(LW=0.08)

클래스 / Use Case	Private	Corporation	Account	Ordinary	Customer_Tx	Ordinary_Tx
Register Customer	0.33	0.33	0	0	0.08	0
Modify Customer	0.12	0.12	0	0	0.13	0
Remove Customer	0.11	0.22	0.03	0.03	0.12	0
Inquire Customer	0.09	0.09	0	0	0.20	0
Open Account	0	0	0.33	0.33	0	0.06
Close Account	0	0	024	0.06	0	0.09
Inquire Account	0	0	0.05	0.05	0	0.14
Deposit	0	0	0	0.08	0	0.12
Withdraw	0	0	0	0.08	0	0.12
Transfer	0	0	0	0.08	0	0.12

표 24 식별된 후보 컴포넌트(LW=0.08)

컴포넌트 A			컴포넌트 B		
Private	Corporation	Customer_Tx	Account	Ordinary	Ordinary_Tx
Register Customer	Register Customer	Register Customer	Open Account	Open Account	Close Account
		Modify Customer	Close Account	Deposit	Inquire Account
		Remove Customer		Withdraw	Deposit
		Inquire Customer		Transfer	Withdraw

표 25 다른 CBD 방법론과의 후보 컴포넌트 식별 방법 비교

비교 항목	Catalysis	CBD96	COMO	제안된 기법
컴포넌트의 식별 기준	개념 수준의 Use Case	상위 수준의 클래스	Use Case와 클래스	Use Case와 클래스
인터페이스	Use Case의 API	Use Case의 API	Use Case의 API	Use Case의 API
장점	이해가 쉽다. 작업이 용이하다.	이해가 쉽다.	클러스터링 흐름이 단순하다.	계층 별로 컴포넌트를 추출한다.시스템의 특성이 인자로 반영될 수 있다.
단점	컴포넌트 별로 자료 중복의 가능성이 있다. 공통 컴포넌트의 식별이 어렵다.	복잡한 업무인 경우 기능의 컴포넌트 할당이 쉽지 않다. 이후 공통 컴포넌트 식별 작업이 요구된다.	공통 컴포넌트들의 경우 식별이 어렵다. 모든 시스템을 동일하게 취급한다.	정확한 Use Case와 클래스 모델을 요구한다.작업 절차가 복잡하다.

5.1.5 업무 컴포넌트 식별

본 논문의 사례 연구에서는 한계 가중치를 0.25로 정하고 표 23의 가중치에 적용하여 후보 컴포넌트를 추출하였다. 이때 추출된 컴포넌트 목록은 표 24과 같다.

5.2 평가

개념 수준의 Use Case로부터 협력(Collaboration)을 추출하여 후보 컴포넌트 입장에서 모델링 하는 Catalysis나 상위 수준의 클래스를 먼저 식별하여 후보 컴포넌트로 설정한 후, 모델링을 하는 CBD96과 같은 CBD 방법론들과는 달리, 제안된 본 방법론에서는 클래스와 Use Case를 같이 다룸으로서 자료나 기능의 중복을 피하도록 하였다. 한편, 계층 분류 또한 컴포넌트 식별 단계에서 반영함으로써 이후에 생길 수 있는 재작업의 가능성을 줄였다. 자세한 비교 내용은 표 25와 같다.

6. 결 론

CBD 기반의 시스템의 성공 여부는 독립적이고 재사용성이 큰 컴포넌트를 식별하는 일일 것이다. 그러나 대

부분의 CBD 방법론들이 그 식별 과정이 직관적인 단점을 가지고 있다. 따라서 본 논문에서는 UML의 기본 산출물인 Use Case 다이어그램과 클래스 다이어그램을 기반으로 Use Case와 클래스 간의 친밀도의 가중치 분석에 의한 컴포넌트 식별 방법을 제안하였다. 제안된 기법은 시스템의 영역과는 상관없이 적용 가능하며, 컴포넌트 들 간에도 공통 컴포넌트와 응용 컴포넌트들을 분리함으로써 전체 시스템 구조 또한 명확하게 보여주고 있다. 객체지향에 익숙한 분석 및 설계자의 입장에서도 실무적으로 적용이 용이하고 이해가 쉬운 분명한 식별 지침들이 제공됨으로써, 분석자의 관점에 따른 상이한 컴포넌트 추출을 가능한 지양하도록 했다.

이후 연구과제로는 다양한 영역에 적용할 수 있는 영역별 적절한 접근값의 범위와 클래스 별 Use Case의 가중치에 대한 좀 더 상세한 연구가 필요하다. 또한 다양한 사례 연구를 통한 검증 작업을 통해서 변수들이 좀 더 정제될 수 있을 것이다.

참 조 문 헌

[1] Martin Fowler, *UML Distilled*, Addison-Wesley, 1999.

[2] Booch, Rumbaugh, and Jacobson, *The UML Modeling Language User Guide*, Addison-Wesley, 1999.

[3] James Martin, *Information Engineering, Book II: Planning and Analysis*, Prentice-Hall, 1990.

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns*, Addison Wesley, 1998.

[5] Souza, Desmond, *Objects, Components, and Frameworks with UML*, Addison Wesely, 1999.

[6] Mitchell I. Kramer, Patricia Seybold Group, *UNIFACE Seven: Delivering on the Promise of Component-Based Development and Deployment*, at URL: http://www.compuware.com/products/uniface/library/analysts/seybold.htm, Compuware Corporation, 1997.

[7] Compuware Corporation, *Uniface Development Methodology: Uniface 7.2*, Compuware Corporation, 1998.

[8] Sterling Software, *The CBD96 Standard Version 2.1 : Standards for Specifying & Delivering Software Components Using COOL:Gen* at http://www.cool.sterling.com/cbd/cbd96.htm, Stering Software Inc., 1998.

[9] Grant Larsen, *Designing Component-Based Framework Using Patterns in the UML*, Communications of the ACM, Vol. 42, No. 10, Oct. 1999.

[10] David C. Sharp, *Reducing Avionics Software Cost Through Component Based Product Line Development*, 2nd Software Architectures in Product Line Acquisitions Workshop, June 1998.

[11] Ivar Jacobson, Martin Griss, Patrik Jonsson, *Software Reuse*, Addison Wesley, 1997.

[12] Philippe Kruchten, *Architectural Blueprints - The 4 + 1 View Model of Software Architecture*, IEEE Software November 1995.

[13] Clemens Szyperski, *Component Software*, Addison Wesley, 1998.

[14] Capt Gary Haines, David Carney, John Foreman, *Component-Based Software Development / COTS Integration*, CMU Software Technology Review, October 1997.

[15] Peter Herzum, and Oliver Sims, *Business Component Factory A Comprehensive Overview of Component-Based Development for the Enterprise*, Wiley, 2000.

[16] Lee, Sang Duck, Yang, Young Jong, Cho, Eun Sook, Kim, Soo Dong, *COMO : A UML-Based Component Development Methodology*, Asia-Pacific Software Engineering Conference (APSEC 99), pp. 54 61, Dec. 1999.

[17] Yu, Young Ran, Kim, Dong Kwan, Kim, Soo Dong, *Connector Modeling Method for Component Extraction*, Asia-Pacific Software Engineering Conference(APSEC99), pp. 46 53, Dec. 1999.

[18] Cho, Eun Sook, Kim, Soo Dong, *A Comparative Study of CBD Methodology*, Workshop on Software Architecture and Components (WSAC '99), Dec. 1999.

[19] Nenade Medvidovic, David S. Rosenblum, *Domain of Concern in Software Architectures and Architecture Description Languages*, Proceedings of the 1997 USENIX Conference on Domain-Specific Language, October 1997.

[20] Nenade Medvidovic, Richard N. Taylor, and E James Whitehead, *Formal Modeling of Software Architectures at Multiple Levels of Abstraction*, Jr. Proceedings of the California Software Symposium 1996, pp. 28-40, Los Angeles, CA, April 1996.

[21] Len Bass, Paul Clements and Rick Kazman, *Software Architecture in Practice*, Addison-Wesley, 1997.

유 영 란

1993년 포항공과대학 전자계산학과 졸업. 1993년 ~ 1996년 현대전자 응용소프트웨어 개발팀 근무. 1996년 ~ 1998년 현대정보기술 의료사업부 근무. 2001년 숭실대학교 대학원 컴퓨터학과 석사. 2001년 ~ 현재 TNG 정보기술 의료정보연구소. 관심분야는 UML 기반의 모델링, CBD, 도메인 공학

김 수 동

1984년 미조리 주립대학교 전산학과 졸업(학사). 1988년 The University of Iowa, 전산학 석사. 1991년 ~ The University of Iowa, 전산학 박사. 1991년 ~ 1993년 한국통신 연구개발단 선임연구원. 1994년 현대전자 소프트웨어연구소 책임연구원. 1995년 ~ 현재 숭실대학교 컴퓨터학부 부교수. 관심분야는 객체지향 개발방법론, 분산객체 컴퓨팅, 컴포넌트 기반 개발방법론