객체지향 시스템으로부터 컴포넌트를 식별하기 위한 모델 기반의 정량적 재공학

이 은 주†

요 약

객체지향 기술은 단위가 되는 클래스가 지나치게 세밀하고 한정적이어서 재사용의 효용이 떨어진다. 컴포넌트는 객체보다 큰 단위로서 복잡도를 효율적으로 관리해주고 품질과 재사용성을 향상시킨다. 또한 MDA나 SOA와 같은 새로운 프레임워크가 등장하면서 컴포넌트 기술의 중요성은 더 커지게 되었다. 따라서 객체지향 시스템을 분석하여 새로운 환경에 적합한 컴포넌트로 재공학하는 기술이 필요하다. 본 논문에서는 객체지향 시스템으로부터 컴포넌트를 식별하기 위한 모델 기반의 정량적 재공학 방법을 제안한다. 본 방법에서는 이전 연구를 확장하여 시스템 모델과 프로세스를 상세히 정의하고 정형화하였다. 객체지향 시스템으로부터 시스템 모델을 구성하고 이 모델을 사용하여 정량적 방법으로 컴포넌트들을 추출하고 정제한다. 또한 지원 도구를 개발하여 현재 존재하는 객체지향 시스템에 적용하여 유효성을 확인한다.

키워드 : 재공학, 객체지향, 재사용, 컴포넌트

Model-Based Quantitative Reengineering for Identifying Components from Object-Oriented Systems

Eun Joo Lee†

ABSTRACT

Due to the classes in object-orientation, which are too detailed and specific, their reusability can be decreased. Components, considered to be more coarse-grained compared to objects, help maintain software complexity effectively and facilitate software reuse. Furthermore, component technology becomes essential by the appearance of the new frameworks, such as MDA, SOA, etc. Consequently, it is necessary to reengineer an existing object-oriented system into a component-based system suitable to those new environments. In this paper, we propose a model-based quantitative reengineering methodology to identify components from object-oriented systems. We expand system model and process, which are defined in our prior work, more formally and precisely. A system model, constructed from object-oriented system, is used to extract and refine components in quantitative ways. We develop a supporting tool and show effectiveness of the methodology through applying it to an existing object-oriented system.

Key Words : Reengineering, Object-oriented, Reuse, Component

1. 서 론

소프트웨어의 재사용은, 개발주기를 단축시켜주고 개발 시스템의 신뢰도를 높여준다. 객체지향 기술은 기대만큼의 재사용을 제공해주지 못하였다. 객체지향 기술은 초기에 캡슐화(encapsulation)와 상속(inheritance), 정보은닉(information hiding)을 통해 소프트웨어의 재사용을 촉진시키는 계기를 마련하였다. 그러나 재사용의 핵심인 효용적 측면에서 볼 때 [1, 2], 객체지향 기술의 단위가 되는 클래스들이 너무 세밀하고 한정적이어서, 재사용 자체는 용이하지만 재사용의 효용성은 떨어지게 된다 [3, 4].

이러한 객체지향 방식의 단점을 보완할 수 있는 컴포넌트 기술이 도입, 발전되었다. 일반적으로 객체보다 큰 단위로 여겨지는 컴포넌트는 구현이 아닌 조립(assembling) 방식으로 시스템을 구축하며 [5, 6], 이해나 유지보수가 어려운 대규모 시스템을 구조화하고 개발하는 데 보다 유용한 접근법을 제공해 준다 [3, 7]. 또한 시스템 수행환경이 단독(standalone)에서 분산시스템으로 변화하였고, MDA(Model Driven Architecture)나 SOA(Service Oriented Architecture) 기술이 등장하면서 컴포넌트에 대한 중요성은 더 커지게 되었다.

따라서 객체지향 기반의 레거시 시스템(legacy system)으로부터 컴포넌트를 추출하여 컴포넌트 기반 시스템으로 변

※본 논문은 정통부 및 정보통신연구진흥원의 정보통신선도기반기술개발사업의 연구결과로 수행되었습니다.
† 종신회원 : 경북대학교 컴퓨터공학과 전임강사
논문접수 : 2006년 8월 25일, 심사완료 : 2007년 1월 9일

환하는 재공학적인 접근법이 필요로 하게 되었다. 본 연구에서 '시스템'의 의미는 하드웨어는 제외하고 소프트웨어의 의미로 한정한다. 객체지향 시스템으로부터 결합도, 응집도 등의 메트릭을 활용하여 컴포넌트를 추출하는 연구들이 다수 수행되었다. 대부분의 경우 CBD(Component-Based Development) 기반이라 설계 정보를 필요로 하거나[3, 8-10], 레거시 시스템에서 재사용 컴포넌트를 추출하기 위하여 도메인에 대한 전문적인 지식을 필요로 하기도 한다[11].

도메인 전문가를 필수적으로 요하지 않으면서, 객체지향 모델로부터 여러 메트릭을 사용하여 재사용 컴포넌트로 구성된 시스템으로 변환하는 연구가 수행되었다[12]. 그러나 [12]에서 제안하는 방법론은 객체지향 시스템이 아니라 객체지향 모델로부터 컴포넌트 기반 시스템으로 변환하고 있다. 그리고 정의된 시스템 모델이 전체적인 프로세스에 충분히 활용되지 못하고 있으며, 프로세스와 메트릭에서 보완해야 할 부분이 존재한다. 또한 C++로 이루어진 비교적 작은 규모의 예제 시스템에서만 적용하고 있으므로, 실제 큰 규모의 시스템에 대한 적용성에 대한 검증이 필요하다.

본 논문에서는 객체지향 모델이 아니라 실제 객체지향 시스템으로부터 객체지향 모델 정보를 추출하고, 추출한 모델 정보와 메트릭을 이용하여 컴포넌트를 추출하는 재공학적 접근법을 제안한다. 이를 위하여 본 연구에서는, [12]에서 정의하고 있는 시스템 모델을 보다 세밀하게 정의하고, 정의한 시스템 모델을 메트릭 기술 및 프로세스 정의에 반영한다. 복잡도 메트릭에서 인터페이스 복잡도에 대한 고려를 추가하고, 이전 프로세스에서 미처 고려되지 않은 부분과 시스템으로부터 모델을 추출하는 단계를 포함하여 프로세스를 확장한다. 그리고 전체적인 재공학 프로세스를 지원해 주는 도구를 개발, JAVA로 구현된 실세계 시스템인 JBidWatcher[13]에 적용하여 방법론의 적용성을 평가한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구에서 이용되는 시스템 모델과 메트릭에 대해 기술하고, 3장에서는 확장된 재공학 프로세스를 제안한다. 4장에서는 실제 적용한 사례를 분석하고, 본 연구에 대한 평가를 기술한다. 5장에서는 결론 및 향후 과제에 관해 기술한다.

2. 시스템 모델과 메트릭

2.1 시스템 모델

[12]의 시스템 모델이 프로세스 전체를 시스템 모델을 이용하여 정형적으로 기술하기에는 완전하지 않으므로 보다 세밀하게 정의할 필요가 있다. 본 절에서는 시스템 모델을 다시 정의한다. 시스템 모델은 객체지향 시스템 모델과 컴포넌트 기반 시스템 모델로 나뉜다. 객체지향 시스템 모델을 아래와 같이 정의한다.

▪ 유일한 식별자들의 전체집합

*UID*는 유일한 식별자들의 전체집합이다.

▪ 자료형의 전체집합

*DTYPE*는 모든 자료형의 전체집합이다.

- $DTYPE = PTYPE \cup ETYPE \cup UC$

*PTYPE*은 *integer, real, bool, string*등과 같은 기본 자료형(data type)의 집합이다
*ETYPE*은 모든 *enumeration* 형의 전체집합이다.
*UC*는 클래스들의 전체집합이다.

▪ 부모 클래스 유형

*PARTYPE*는 부모 클래스의 유형 나타낸다

- $PARTYPE = \{abstract, concrete\}$
- *abstract*는 인터페이스 상속에 해당되는 추상 부모 클래스이고 *concrete*는 구현 상속에 해당되는 구체 부모 클래스이다.
- *partype*(c) $\in$PARTYPE$\cup\{\phi\}$ 클래스 *c*가 부모 클래스일 경우, 부모 클래스의 유형을, 부모 클래스가 아닐 경우는 널 값을 추출해 주는 함수이다.
- ϕ는 널값 *(NULL value)* 이다.

▪ 멤버에 대한 접근 제어자 유형

ACCTYPE 은 멤버에 대한 접근 제어자의 유형이다.

- $ACCTYPE = \{ public, private, protected\}$
- *a, c*$\in$*UC*에 대해 *a*가 *c*의 멤버 속성으로 쓰였을 때 *a*의 접근 제어자를 추출해 주는 함수 *aacctype(c, a)*를 아래와 같이 정의한다.
  - *aacctype(c, a)*$\in$*ACCTYPE*는 클래스 *c*가 멤버 속성으로 가지고 있는 멤버 속성 *a*의 접근 제어자를 추출해 주는 함수이다.

▪ 메소드의 결과형과 인자형의 전체집합

*SIG*는 메소드의 시그니처(*signature*), 즉 결과형과 인자형의 전체집합이다.

$SIG = \{\langle s_1 \times \ldots \times s_n, s\rangle \mid s_1, s_2, \ldots, s_n, s \in DTYPE \cup \{\phi\}\}$
$sorts(x) = \{ s_1, \ldots, s_n, s \}$, *where* $x = \langle s_1 \times \ldots \times s_n, s\rangle$

▪ 클래스 정의

*UC*는 클래스들의 전체집합이다. *c*$\in$*UC*인 *c*는 아래 튜플로 정의된다.

$c = \langle name, ATTRSET_c, MSET_c, PAR_c, CP_c, USE_c\rangle$

- *name* $\in$*UID* 은 클래스 c의 고유한 식별자이다.
- 클래스 속성의 집합

  $ATTRSET_c$은 클래스 *c*의 속성의 집합이며 아래 튜플의 집합으로 정의된다.

  $ATTRSET_c = \{\langle name, type, atype\rangle \mid name \in ID, type \in DTYPE, atype \in ACCTYPE,$ 여기서 *name*은 *c*의 멤버 변수의 고유 식별자, *type*은 *c*의 멤버 변수의 자료형, *atype*은 *c*의 멤버 변수의 접근 제어자*)*
  - $attrset(c) = \langle t_1, t_2, \ldots, t_m\rangle$, $t_1 \ldots t_m \in DTYPE$는 클래스 *c*의 속성의 자료형을 추출해 주는 함수이다.
- 클래스 멤버 메소드의 집합

  $MSET_c$은 클래스 c의 멤버 메소드들로서 아래 튜

플의 집합으로 정의된다.

$MSET_c$={<*name, sig, mtype, body*>| *name*∈*UID, sig*∈*SIG, mtype*∈*ACCTYPE,* 여기서 *name*은 메소드의 식별자, *sig*는 메소드의 시그니처, *mtype*은 메소드의 접근 제어자, *body*는 메소드의 실행 가능한 코드}

- *siglist*(*mid*) = <S_1, S_2, ..., S_n, S> (S_1, ...S_n, S ∈*DTYPE*)
- *siglist*(*mid*)은 메소드 *mid*∈*UID*의 시그니처의 각 타입(결과형, 인자 자료형)을 추출해 주는 함수이다.
- *macctype*(*mid*)∈*ACCTYPE* 은 메소드 mid∈*UID*의 접근 제어자 유형을 추출해 주는 함수이다.
- *mset*(c)⊂*UID*는 클래스 *c*의 멤버 메소드 식별자들을 추출해 주는 함수이다.

– 클래스 부모 클래스들의 집합

PAR_c은 클래스 c의 부모 클래스와 그 종류(추상, 구체) 의 집합이다.

- PAR_c = {<*name, ptype*> | *name*∈*UID, ptype* ∈*PARTYPE,* 여기서 *name*은 *c*의 부모 클래스의 식별자, *ptype*은 그 부모 클래스의 유형}
- *parent*(c)⊂*UC*는 클래스 c의 직접 부모 클래스들을 추출해 주는 함수이다.
- *partype*(p)*PARTYPE*는 부모 클래스인 p의 타입(*abstract, concrete*)을 추출해 주는 함수이다.
- *child*(c)⊂*UC*는 클래스 c의 직접적인 자식 클래스들을 추출해 주는 함수이다.

– 클래스 합성 멤버 클래스들의 집합

CP_c는 클래스 c의 합성 멤버 클래스들의집합이다.

- CP_c = { *d*| *d* ∈*UC,* 여기서 *d*는 c의 합성 멤버 클래스}
- *iscomposition*(x, y)는, 만일 클래스 x가 클래스 y를 합성 멤버 클래스로 가지고 있을 때 참이고 그렇지 않으면 거짓이다.
- *compose*(c)= { d | d ∈*UC, iscomposition*(c, d) }

– *compose*(c)는 c의 합성 멤버 클래스를 추출해 주는 함수이다.

- *composed*(c)={ d | d ∈*UC, iscomposition*(d, c) }

– *composed*(c)는 c를 합성 멤버 클래스로 가지고 있는 클래스를 추출해 주는 함수이다.

– 클래스 사용하는 클래스들의 집합

USE_c는 클래스 c가 사용하는 클래스의 집합이다.

- USE_c = { d | d∈*UC,* 여기서 d는 c가 사용하는 클래스}
- *use*(c)= {d| m_c∈*mset*(c)∧ ∃d∈*UC*∧ ∃m_d∈ *mset*(d)∧m_d∈*call*(m_c) } ∪{d|d∈*access*(c)}

– *use*(c)는 클래스 c가 사용하는 클래스들을 추출해 주는 함수로서, 멤버 메소드 사용과 멤버 속성 사용을 모두 포함한다.

– *call*(m)은 메소드 m이 호출하는 메소드를 추출해 주는 함수이다.

– *access*(c)은 클래스 c가 사용하는 클래스를 추출해 주는 함수로서, c 내에서 해당 클래스의 멤버 메소드에 대한 직접 호출이 없는 경우이다. 즉 해당 클래스의 속성을 읽고(get), 쓰는(set) 연산이 암묵적으로 발생하는 경우이다.

- 객체지향 시스템

객체지향 시스템 S는 상호작용하는 클래스들의 집합이다.

– S⊂*UC*

정의된 명세를 이용하여 모델링 된 객체지향 시스템은 재공학 프로세스를 통해 컴포넌트 기반 시스템으로 변환된다. 컴포넌트를 구성하고 있는 요소들을 1) 관련된 클래스들, 2) 목적 코드와 같은 실행 가능한 코드, 혹은 3) 이미지나 텍스트 파일 등 세 종류로 분류할 수 있다[14]. 본 논문에서는 컴포넌트를 관련된 클래스들의 집합으로 인터페이스를 통해 서비스를 외부로 제공해 주는 소프트웨어로서 외부 컴포넌트와의 결합력은 약하고 자체 응집력은 강한 모듈로 정의한다. 또한 컴포넌트간의 관계를 의존 관계만으로 한정 짓는다. 컴포넌트 기반 시스템에 대한 명세를 아래와 같이 정의한다.

- 컴포넌트 정의

*UCOMP*는 컴포넌트들의 전체집합이다. C∈*UCOMP* 인 C는 다음 튜플로 정의된다.

C = <*name*, $INTERFACES_C$, $CSET_C$, $COMPDEP_C$>

– *name* ∈*UID*

- 컴포넌트 인터페이스

$INTERFACES_C$는 C가 제공해 주는 모든 인터페이스집합을 나타낸다.

– $INTERFACES_C$={<$name_I$, *sig*> | $name_I$ ∈*UID*, *sig* ∈*SIG,* 여기서 $name_I$는 인터페이스의 식별자, *sig*는 인터페이스의 시그니처}

– *interfaces*(C)={ <S_1, S_2, ..., S_n, S> |S_1, ...S_n, S ∈ *DTYPE}*

interfaces(C)는 컴포넌트 C의 인터페이스시그니처의 집합을 추출해 주는 함수이다.

- 컴포넌트의 구성 클래스들의 집합

$CSET_C$ = {c | c ∈ *UC,* c는 C의 구성 클래스}

– *cset*(C)⊂*UC*은 컴포넌트 C의 구성 클래스들을 추출해 주는 함수이다.

- 컴포넌트가 의존하는 컴포넌트들의 집합

$COMPDEP_C$={b|a∈*cset*(C)∧b∈(*UCOMP*-{C})∧b∈use(a)}

– *compdep*(C)⊂*UCOMP*는 C가 의존하는 컴포넌트들의 집합이다.

- 컴포넌트 기반 시스템

변환된 시스템 S'는 컴포넌트들의 집합이다.

S' ⊂*UCOMP*

2.2 메트릭

본 논문에서 전체 시스템의 품질과 생성된 컴포넌트의 재사용성을 예측하기 위하여 응집도와 결합도를 이용한다. 높은 응집도와 낮은 결합도는 객체 지향 시스템뿐 아니라 컴포넌트 기반 시스템의 품질을 말해주는 척도가 되며 [15], 클래스의 재사용성[16, 17] 및 컴포넌트의 재사용성 [18]과 밀접한 관련이 있기 때문이다. 메트릭은 연결 강도(CS: Connectivity Strength), 응집도(COH: Cohesion), 복잡도(COX: Complexity) 세 가지가 기본적으로 정의되었고, 연결 강도와 응집도를 이용하여 시스템 품질값이 정의된다[12]. 아래 정의에 사용된 개념과 표기법은 2.1절에서 기술한 시스템 모델을 활용하였다.

[정의 1] 컴포넌트 연결강도

컴포넌트 연결 강도 CS(C_i, C_j)는 컴포넌트 C_i, C_j 사이에 얼마나 강하게 결합되어 있는지를 나타내는 척도이며, 클래스 연결 강도와 메소드 연결 강도를 이용하여 정의된다.

$$CS(C_i,C_j)=\sum_{c_u\in cset(C_i)}\sum_{c_v\in cset(C_j)}CS(c_u,c_v)$$

여기서

$$CS(c_u,c_v)=\sum_{m_x\in mset(c_u)}\sum_{m_y\in mset(c_v)}CS(m_x,m_y)+\sum_{m_x\in mset(c_v)}\sum_{m_y\in mset(c_u)}CS(m_x,m_y)$$

$$CS(m_x,m_y)=\begin{cases}|pri\_arg(m_y)|*w_{pri}+(|abs\_arg(m_y)|+1)\bullet w_{abs} & \text{if } m_x \text{ calls } m_y,\\ 0 & otherwise\end{cases}$$

w_{pri} , w_{abs} : 기본형과 사용자 정의형 인자에 관한 각각의 가중치로써 (w_{pri} + w_{abs} = 1.0, usually w_{abs} > w_{pri}) 이다.

$|pri\_arg(m_y)|$, $|abs\_arg(m_y)|$: 메소드 m_y에 대한 기본형과 사용자 정의형의 개수이며 사용자 정의형의 개수에서 1을 더한 이유는 항상 객체 자신에 대한 참조가 묵시적으로 this로 전달되기 때문이다.

[정의 2] 컴포넌트 응집도

응집도는 컴포넌트 내부 클래스 간의 관계 긴밀도를 보여주며 아래와 같이 정의된다.

$$COH(C_i)=\begin{cases}1 & if\ |C_i|=1\\ \dfrac{\sum_{c_u\in cset(C_i)}\sum_{c_v\in cset(C_i)}w_R(c_u,c_v)}{|C_i|*(|C_i|-1)} & otherwise\end{cases}$$

여기서

$$w_R=\begin{cases}w_u & c_u \text{ 가 } c_v\text{와 사용관계를 갖는경우}\\ w_g & c_u \text{ 가 } c_v\text{와 상속관계를 갖는경우}\\ w_c & c_u \text{ 가 } c_v\text{와 합성관계를 갖는경우}\\ 0 & otherwise\end{cases}$$

w_u, w_g, w_c : 각각 사용 관계, 상속 관계, 합성관계에서의 가중치.

($0 \le w_u, w_g, w_c \le 1.0$, $w_u + w_g + w_c = 1.0$)

$|C_i|$: 컴포넌트 C_i에 포함되어 있는 클래스 개수

S를 어떤 두 컴포넌트가 병합되기 전의 시스템, S'를 병합된 후의 시스템이라고 할 때, 시스템품질은 다음과 같이 정의한다.

[정의 3] 시스템 품질

시스템 품질: $Q(S')=w_1*\dfrac{COH(S')}{COH(S)}+w_2*\dfrac{CSS(S)}{CSS(S')}$

여기서 $CSS(S')=\sum_{C_i,C_j\in S'}CS(C_i,C_j)$, $COH(S')=\dfrac{\sum_{C\in S'}COH(C)}{|S'|}$

w_1, w_2 : 각각 COH와 CSS의 중요도에 관한 가중치.

$|S'|$: 시스템 S'에 속한 컴포넌트 개수

복잡도 메트릭의 경우[12]에서는 구성 클래스들의 복잡도만을 고려하고 있으나, 컴포넌트의 복잡도 산정 시 다수의 연구에서 인터페이스의 복잡도를 고려하고 있다[19- 21]. 또한 본 연구에서 컴포넌트는 관련된 클래스들의 집합으로 정의되며 관련된 클래스들이 하나의 컴포넌트를 구성할 때, 외부에서 사용하는 인터페이스도 생성된다. 따라서 컴포넌트의 복잡도를 측정함에 있어 컴포넌트를 구성하는 클래스들뿐 아니라 실제 컴포넌트의 서비스를 나타내는 인터페이스에 대한 복잡도가 고려되어야 한다. [12]에서는 컴포넌트 복잡도 정의에서 인터페이스의 복잡도를 고려하지 않고 있으므로 본 연구에서는 인터페이스가 포함하는 자료형의 개수 및 유형을 이용하여 아래 정의5와 같이 인터페이스 복잡도를 정의하였다. 그리고 인터페이스 복잡도와 클래스 복잡도를 이용하여 정의6과 같이 컴포넌트 복잡도를 정의하였다.

[정의 4] 클래스 복잡도

$$CCOX(c_i)=\sum_{a\in TYPE(c_i)}\begin{cases}w_{pri} & if\ a\ is\ primitive\ type\\ w_{abs} & if\ iscomposition(c_i,a)\ is\ false\\ CCOX(a)\bullet w_{abs} & otherwise\end{cases}$$

여기서

$$TYPE(c_i)=siglist(mset(c_i))\cup attrset(c_i)$$

[정의 5] 인터페이스 복잡도

$$ICOX(C)=\sum_{i\in ITYPE_c}\begin{cases}w_{pri} & if\ i\ is\ primitive\ type\\ w_{abs} & if\ i\ is\ user-defined\ type\end{cases}$$

여기서

$$ITYPE_c=\{siglist(i_m)\,|\,i_m\in iset(C)\}$$

$$iset(C)=\{m\,|\,m\in mset(c)\wedge c\in cset(C)\wedge macctype(m)\equiv public\wedge m\in mset(use(D))\wedge(C\neq D)\wedge(D\in UCOMP)\}$$

[정의 6] 컴포넌트 복잡도

$$COX(C_i) = \sum_{c_u \in cset(C_i)} \{C\,COX(c_u) + \sum_{c_v \in parent(c_u)} CCOX(c_v)\} + \sum ICOX(C_i)$$

프레임워크[22]을 사용한 연결 강도 및 응집도에 대한 검증은 [12]를 참고하고, 수정된 복잡도 메트릭 역시 프레임워크를 따르나, 그 증명은 지면 관계상 생략하기로 한다.

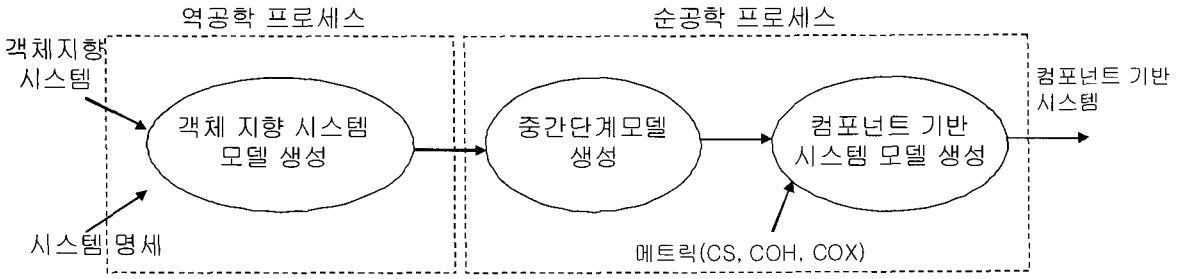
3. 모델 기반 재공학 프로세스

(그림 1)는 본 연구에서 적용한 재공학 프로세스이다. 프로세스는 크게 시스템 모델을 추출하는 역공학(reverse engineering) 프로세스와 추출된 객체 지향 시스템 모델을 컴포넌트 기반 시스템 모델로 변환하는 순공학(forward engineering) 프로세스로 구성된다. 순공학 프로세스는 다시 [12]의 기본 컴포넌트 개념을 확장한 중간 단계 모델 생성 단계와 중간 단계 모델로부터 클러스터링을 통하여 컴포넌트를 생성하는 컴포넌트 기반 시스템 모델 생성 단계로 구성된다.

모델에서 시작하는 [12]와 달리 본 연구에서는 역공학 프로세스를 추가하였고, 시스템 모델을 재정의하여 전체 과정에 보다 정확히 반영될 수 있도록 한다. 또한 [12]에서는 각 단계 정의가 세밀하지 않아 도구를 구현하여 실제 큰 규모의 시스템에 적용하기에는 부족한 부분이 많다. 본 연구에서는 [12]에서 고려되지 않은 부분을 포함하여 변환 프로세스를 확장하고, 전 프로세스는 기 정의한 시스템 모델로 기술하여, 이는 향후 도구 구현에 반영되어 정확성 및 효율을 기할 수 있도록 한다.

3.1 객체 지향 시스템 모델 생성

본 절에서는 [12]에서 제안한 프로세스에 포함되어 있지 않은, 객체지향 시스템으로부터 모델 정보를 추출하는 단계에 대하여 기술한다. 객체지향 시스템의 경우 시스템 모델을 생성하기 위해 필요한 정보는 대부분 코드로부터 자동으로 획득할 수 있다. 추출할 정보는, 3.1절에서 정의한 객체 지향 시스템 모델의 기본 단위인 클래스 c=<*name*, $ATTRSET_c$, $MSET_c$, PAR_c, CP_c, USE_c>이며, name, $ATTRSET_c$, $MSET_c$는 각각 클래스 c의 이름, 속성집합, 메소드집합이고, PAR_c, CP_c, USE_c는 각각 c와 상속, 합성, 사용 관계에 있는 클래스



(그림 1) 재공학 프로세스

```
public abstract class Employee{
    String name;
    public abstract void accept(Visitor visitor);
    public String getName(){   return name;}
}
public class HourlyEmployee extends Employee{
    private double rate;
    public HourlyEmployee(String name, double rate)
    {
        this.name = name;   this.rate = rate;
    }
    public double getRate(){        return this.rate; }
    public void accept(Visitor v)
    {
        v.visitHourlyEmployee(this);
    }
}
...
```

(그림 2) 클래스 예제

들의 집합이다.

예를 들어 다음 JAVA 클래스 *HourlyEmployee* (그림2)의 모델링 정보는 다음과 같은 방식으로 추출한다.

*name*은 클래스의 유일한 이름으로, 본 논문에서는 패키지의 이름을 계층 순서대로 클래스 이름 앞에 붙이는 방법을 선택하였고, 본 예제에서는 편의상 'c'라고 가정한다. $ATTRSET_c$ 은 클래스 c가 가지고 있는 속성의 이름, 타입, 접근제어자 유형을 포함한다. 그 중 자료형의 경우, 사용자 정의형과 기본형으로 나뉘어 이후에 이용된다. 사용자 정의형은 현 시스템에서 정의된 클래스 만을 가리킨다. 예를 들어 String이나 Vector과 같은 시스템 정의형은 기본형으로 간주한다. 예제의 $ATTRSET_c$ 은 다음과 같다.

- $ATTRSET_c$ = { <rate, double, private>}

$MSET_c$도 속성 집합과 마찬가지로 시그니처의 자료형을 중심으로 쉽게 추출 가능하며 본 예제는 다음과 같다.

- $MSET_c$ = {<getRate, sig_1, public, $body_1$>, <accept, sig_2, public, $body_2$>}
  - sig_1=<void, double>
  - sig_2=<Visitor, void>

sig_i의 가장 마지막 요소는 반환 자료형으로, sig_2의 경우에는 인자는 사용자 정의형인 Visitor이며 반환 자료형이 void임을 뜻한다. $body_i$는 해당 함수의 본체이다.

합성 멤버 클래스인 CP_c의 경우는 의미적인 부분이 있어 자동으로 추출하는 것에는 한계가 있으므로 몇 가지 휴리스틱을 이용하여 합성 관계의 후보를 결정한 후 사용자가 선택할 수 있도록 한다. 1)생성자 내에서 *new*로 생성하는 멤버 변수나 2)생성자 내에서 할당(assign)하는 멤버 변수, 3) 클래스 내에서 *new*로 미리 생성된 멤버 변수에 해당되는 *private* 클래스 4)클래스의 내부 클래스(inner class) 5)합성 멤버로 갖는 추상 부모 클래스의 자식 클래스 등의 경우 합성 멤버 클래스 CP_c의 후보로 둔다.

USE_c의 경우는 속성 사용과 메소드 사용으로 나뉘며 속성이나 메소드의 사용 빈도는 고려하지 않는다. 프로그램 정적 분석으로는 실제 실행 시의 사용 빈도를 예측할 수 없기 때문이다. 그리고 메소드 사용의 경우 사용하는 메소드가 생성자일 경우는 포함하되 사용되는 메소드가 객체 생성자일 경우는 제외한다. 본 예제에서 accept()의 내부에서 Visitor의 인스턴스인 v의 visitHourlyEmployee(this)를 호출하고 있으므로 USE_c는 다음과 같다.

- USE_c ={Visitor}

부모 클래스의 집합인 PAR_c도 코드상에서 직접적으로 획득할 수 있으며 다음과 같다.

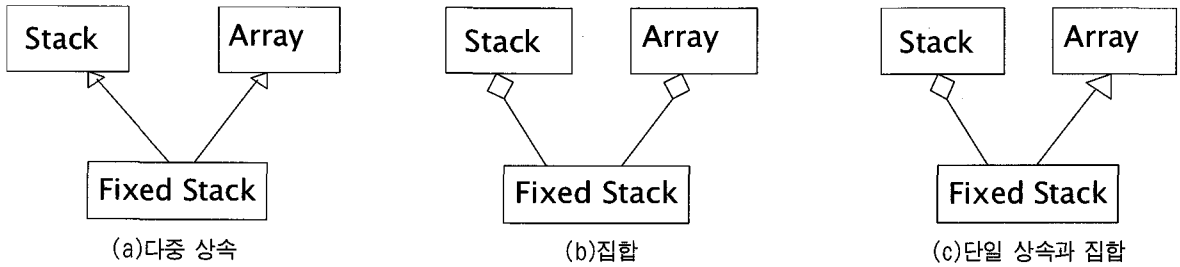
- PAR_c = { <Employee, abstract>}

단, [12]에서는 다중 상속(multiple inheritance)이 고려되고 있지 않으므로 이에 대한 가정이 필요하다. 다중 상속의 지원유무는 객체지향 언어에 의존적이며 그 사용에서 여러 문제점이 제기되고 있다[23, 24]. 다중 상속은 다이아몬드 형 상속 관계가 발생할 수 있고, 프로그래밍 언어에 의존적이며 시스템 간의 이식성 문제를 야기시키고, 클래스 사이의 IS-A 관계가 확실하지 않아 상위 클래스들의 자료구조와 연산 중의 일부만 상속받기 위한 형태가 되기 쉽다[23]. 또한 Willis등은 두 부모 클래스로부터 구조(structure)를 상속받는 경우 모순이 발생함을 이론적으로 증명하고, 따라서 자식 클래스는 두 개의 부모 클래스 중 한 클래스만의 인스턴스가 되어야 한다고 주장한다[24].

본 연구에서는 다중 상속이 있을 경우[23, 24]에서 제안한, 집합(aggregation) 관계를 활용한 방식을 이용하여, 부모 클래스들을 자식 클래스의 멤버로 두거나, 한클래스만 부모 클래스로 두고 한 클래스는 멤버로 두어 다중 상속을 제거하도록 한다(그림 3).

단, 본 논문에서의 다중 상속 제한은 구현 상속일 경우만 제한한다. 즉 한 구현 상속과 하나 이상의 인터페이스 상속이 포함된 다중 상속의 경우는 허용한다. 전기한 다중 상속의 문제점은 대부분 구현 상속의 문제점이며, 인터페이스는 객체에 대해 다양한 관점을 보여주는 추상화 도구로서, 한 객체가 다양한 관점을 가질 수가 있기 때문이다[23].

실제 JAVA 시스템은 다수의 인터페이스를 받아서 구현하는 경우는 많지만, 구현 다중 상속인 경우는 허용하지 않으므로 제안한 방식을 바로 적용할 수 있다. C++ 시스템과 같이 구현 다중 상속을 허용하는 경우에는 전기한 방식으로 코드에 대한 리팩토링(refactoring)을 거친 후에 본 프로세스를 적용하도록 한다. 시스템 모델은 XML 스키마로 정의되어 XML 문서로 저장된다. XML로 추출된 시스템 모델의 예를 (그림 10)에서 볼 수 있다.



(a)다중 상속 (b)집합 (c)단일 상속과 집합

(그림 3) 다중 상속의 변환

3.2 중간 단계 모델 생성

중간 단계 모델은 [12]의 기본 컴포넌트 생성에 해당한다. 기본 컴포넌트는 클러스터링을 이용한 시스템 정제 단계 이전에, 설계 정보를 이용하여 생성된 보다 큰 단위의 컴포넌트이다[12]. 합성 기본 컴포넌트와 상속 기본 컴포넌트가 있는데, 각각 클래스 사이의 합성 관계와 상속 관계로부터 생성되며, 결과적으로 결합도를 낮추고 응집도를 높이는 것을 기본 원칙으로 삼고 있다. 합성 기본 컴포넌트는, 합성 관계를 가지고 있는 클래스들을 하나의 컴포넌트로 만드는 것이다. 그리고 상속 기본 컴포넌트는 부모 클래스들과 자식 클래스들이 다른 컴포넌트에 분산되지 않도록 만드는 것인데, 부모 클래스의 유형에 따라 클래스의 이동과 복제가 수행된다. 부모 클래스가 추상 클래스(abstract class)인 경우는 해당 부모 클래스를 컴포넌트 내부로 복제하도록 하고, 구체 클래스(concrete class)인 경우, 즉 구현 상속인 경우는 컴포넌트 내부로 이동하도록 한다.

[12]에서는 기본 컴포넌트 생성 단계가 비교적 간략하게 정의되어 있어 실제 큰 규모의 시스템에 적용하기에 부족하다. 본 연구에서는 2장의 기본 컴포넌트 생성과 다음 단계로 진입하기 전에 수행되어야 할 시스템 정리 작업을 중간 단계 모델 생성과정으로 정의하고 과정의 일부를 시스템 모델을 사용하여 정형적으로 보인다. 시스템 정리 작업이란 기본 컴포넌트 생성 후, 생성된 컴포넌트의 중복을 줄이기 위하여 불필요한 컴포넌트를 제거하고 어떤 컴포넌트에도 속하지 않은 단독 클래스에 대해 컴포넌트를 생성하는 정리 단계이다.

기본 컴포넌트 생성 시에 발생하는 동작은 이동 및 복제이다. 이동은 어떤 클래스가 생성된 컴포넌트의 구성 클래스가 되는 것을 말하며, 복제는 한 클래스가 두 개 이상의 컴포넌트에 대해 구성 클래스가 되는 것을 말한다.

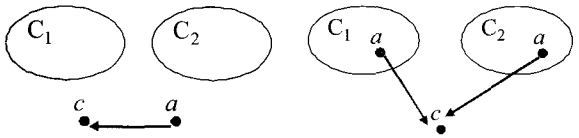
(1) 이동 및 복제 시 시스템 모델 변경

본 연구에서는 이동 및 복제 시 시스템 모델 변경을 다음과 같이 정의한다.

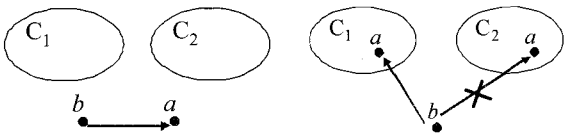
- 클래스의 이동
  클래스 c가 컴포넌트 C 내부로 이동될 때 시스템에 생기는 변화는 다음과 같다.
  - $CSET_C=CSET_C \cup \{c\}$ ($CSET_C$는 C의 내부 클래스들의 집합)
  - $S=S-\{c\}$
- 클래스의 복제
  클래스 c가 컴포넌트 C 내부로 복제될 때 시스템에 생기는 변화는 다음과 같다.
  - $CSET_C=CSET_C \cup \{c\}$

한 클래스가 두 개 이상의 컴포넌트 내로 복제될 경우, 해당 클래스에 대한 사용 관계에 변화가 생긴다. 이를테면 클래스 a가 컴포넌트 C_1과 C_2 내부로 복제되었을 때, a가 복제 이전에 외부 클래스 c를 사용했다면(그림 4(a)), 복제 후 C_1과 C_2 모두 c와 일종의 사용 관계를 가지게 된다 (그림 4(b)). 엄밀히 하면, 컴포넌트는 다른 컴포넌트와의 의존 관계만을 정의했으므로, C_1과 C_2는 c가 아니라 c를 컴포넌트화 한 C와 의존 관계를 가진다고 할 수 있다.

그러나 복제 이전에 다른 클래스 b가 a를 사용하고 (그림 5(a)), 그 사용 관계를 복제 후까지 유지하면 복제된 후에는 b가 C_1과 C_2 모두와 의존 관계를 가지게 된다. 이는 재공학 전·후 시스템의 실행 일관성을 저해하므로 C_1이나 C_2 중에 한 컴포넌트에만 의존 관계를 가지도록 한다 (그림 5(b)). 본 프로세스에서는 이 원칙을 기본 컴포넌트 생성 후, 생성



(그림 4) 복제 시 사용 및 의존 관계 변화 (1)



(그림 5) 복제 시 사용 및 의존 관계 변화 (2)

```
/* 부모 클래스를 갖지 않으며, 어떠한 클래스의 합성 멤버로도 포함되지 않고 합성
멤버 클래스를 갖는 클래스들의 집합 */
ROOT_COMP = { c | c ∈S, composed(c)=∅  ∧compose(c) ≠∅  ∧parent(c)= ∅ }
for all ci, ci ∈ROOT_COMP
      create composition basic component comp_ci ;
/* ci를 시작점으로 ci와 합성 관계에 있는 다른 클래스들을 깊이 우선 탐색법(DFS: Depth
First Search)으로 x에 반환해 준다 */
      while( x=DFS(ci) )
            if (child(x) or |composed(x)|≥2)
                  copy x to comp_ci ;
            else
                  move x to comp_ci ;
      end of while
end of for
```

(그림6) 합성 기본 컴포넌트 생성 알고리즘

된 기본 컴포넌트와 컴포넌트에 복제나 이동 되지 않은 클래스들을 컴포넌트화 하는 과정에서 적용하도록 한다.

본 연구의 전 프로세스는 시스템 모델을 이용하여 보다 체계적으로 표현이 가능하며, 본 절에서는 그 중 합성 기본 컴포넌트 생성 과정을 소개한다 (그림 6).

(2) 시스템정리 및 컴포넌트 기반 시스템으로 변환

기본 컴포넌트 생성 후, 다음 단계로 진행하기 전에 보다 효율적으로 수행하기 위하여 몇 가지 추가 작업이 필요하며, 아래와 같이 정리한다.

- 만일 한 클래스가 하나 이상의 컴포넌트로 복제되었다면, 클래스의 중복과 불필요한 다음 단계의 정제 프로세스를 줄이기 위해 시스템에서 제거한다.
- 단계1, 2에서 생성된 컴포넌트들이 사용하는 외부 클래스에 대해 만일 한 클래스가 오직 하나의 컴포넌트에 의해 사용되며 다른 클래스나 컴포넌트를 각각 사용, 의존하지 않을 때, 그 클래스를 컴포넌트 내부로 이동한다.
- 합성 및 상속 기본 컴포넌트 생성 과정을 거치면서, 다른 컴포넌트에 포함되는 컴포넌트가 생길 수 있다. 따라서 어떤 컴포넌트의 CSET이 다른 컴포넌트의 CSET

```
[가정]
-  기본 컴포넌트는 각각 name과 CSET을 가지고 있다.
-  COMP={ C | C는 이전 두 단계에서 생성된 기본 컴포넌트}
-  CLS={ c | c는 이전 두 단계를 거친 후 시스템에 남아 있는 클래스}

/* 불필요한 클래스 제거 */

COMP_CLS = ⋃_{i=1} cset(C_i), C_i ∈ COMP

CLS' = { c | c∈CLS ∧c ∉ COMP_CLS} ;

/* 단독 사용 클래스 이동 */
for all C_i ∈COMP
    for any d ∈CLS'∧d∈use(c)
        if ( ∃ C_k∈COMP-{C_i}:d∈cset(C_k))
            break ;
        endif
        add d onto C_i's CSET ;
        CLS'=CLS'-{d} ;
    end of for
end of for

/* 불필요한 컴포넌트 제거*/
for all C_i , C_j∈COMP
    if cset(C_i) ∈cset(C_j)
        COMP=COMP-{C_i} ;
    endif
end of for

/* 컴포넌트 화 */
for each c_i ∈CLS', create component C_i
    create C_i's name ;
    set C_i's CSET as {c_i} ;
    add C_i to COMP ;
end of for

/* 각 컴포넌트에 대해 COMPDEP 결정*/
for each C_i ∈COMP, create COMPDEP_i
    for each c_j ∈cset(C_i)  // 한 컴포넌트 C_i를 구성하는 클래스 각각에 대해
        for each u_k ∈use(c_j) // 클래스가 사용하는 클래스 각각에 대해
            TMPDEP_k = Ø ;
            for each D_i ∈COMP-{C_i} // 시스템의 다른 컴포넌트들에 대해
                if u_k ∈cset(D_i) // 사용하는 클래스가 속한 컴포넌트이면
                    TMPDEP_k=TMPDEP_k∪{D_i} ; //의존관계 후보에 추가
                endif
            end of for
            if |TMPDEP_k|>1 //한 클래스가 두개 이상의 컴포넌트에 있으면
                select one element D in TMPDEP_k ;
                TMPDEP_k={D} ; // 하나의 컴포넌트만 의존관계로
            endif
            COMPDEP_i = COMPDEP_i ∪TMPDEP_k ;
        end of for
    end of for
end of for
```

(그림 7) 시스템 정리 및 컴포넌트 기반 시스템으로 변환 과정

에 포함되는 경우, 포함되는 컴포넌트를 제거한다.

- 이전 단계에서 생성된 기본 컴포넌트에 포함되지 않은 각 클래스들도 하나의 컴포넌트로 간주하여 컴포넌트 기반 시스템을 생성한다. 이 때 본 절에서 기술된, 복제 시 컴포넌트 간의 의존 관계 변경 원칙을 적용한다.

(그림 7)은 정리 작업 전체를 보여준다.

3.3 컴포넌트 기반 시스템 모델 생성

본 절에서는 컴포넌트 기반 시스템 모델 생성에서 수행되는 정제 프로세스[12]와 정제 시에 변경되는 시스템 모델 정보에 관해 소개한다. 시스템 정제에서는 시스템의 품질을 향상시키는 방향으로 두 컴포넌트들 간의 클러스터링이 발생한다 (그림 8). 정제 프로세스는, 시스템 내의 모든 컴포넌트의 쌍에 대하여 가장 큰 연결 강도를 가지는 두 컴포넌트를 대상으로 수행된다. 해당 두 컴포넌트를 병합했을 때, 위의 (정의3)을 이용하여 구한 시스템 품질이 향상이 되었으면 병합하고, 그렇지 않으면 이들 쌍을 병합 후보군에서 제외한 다음, 다른 병합할 대상을 찾는다. 이러한 과정은 시스템이 더 이상 향상되지 않거나, 병합할 대상이 없을 때 중단된다.

두 컴포넌트가 클러스터링되면, 클러스터링된 컴포넌트의 CSET, INTERFACES, COMPDEP가 변경되고, 병합 되기 전의 두 컴포넌트 중 어느 하나에 의존 관계를 가지고 있던 컴포넌트의 COMPDEP이 변경되어야 한다. 본 절에서는 [12]에서 포함되지 않은 시스템 모델 정보 변경을 아래와 같이 정의한다.

- 두 컴포넌트의 병합

두 컴포넌트 C_1, C_2가 $C_{1\cup2}$로 병합될 때, 아래와같은 변경이 발생한다.

- $C_{1\cup2}$=<name, $CSET_{1\cup2}$, $INTERFACES_{1\cup2}$, $COMPDEP_{1\cup2}$>
  - *name* ∈UID
  - $CSET_{1\cup2}=CSET_1\cup CSET_2$
  - $INTERFACES_{1\cup2}=INTERFACES_1\cup INTERFACES_2$
  - $COMPDEP_{1\cup2}=COMPDEP_1\cup COMPDEP_2-\{C_1, C_2\}$

  (여기서 $CSET_1$은 C_1의 CSET을 말하며, 그 외의 항목에도 마찬가지로 적용된다.)
- $C_d\in\{ C \mid C\in COMP\wedge(C_1\in compdep(C)\vee C_2\in compdep(C))\}$ 에 대해

  $COMPDEP_d=COMPDEP_d\cup\{C_{1\cup2}\}-\{C_1, C_2\}$

정제 프로세스에서 많은 클래스들로 구성된 컴포넌트의 경우 외부 다른 컴포넌트들과 관련이 있는 경우가 많아 클러스터링이 더욱 많이 발생하는 경향이 있으므로, 복잡도 임계치를 두어 생성되는 컴포넌트의 규모를 적절하게 유지하도록 한다.

4. 사례 연구 및 평가

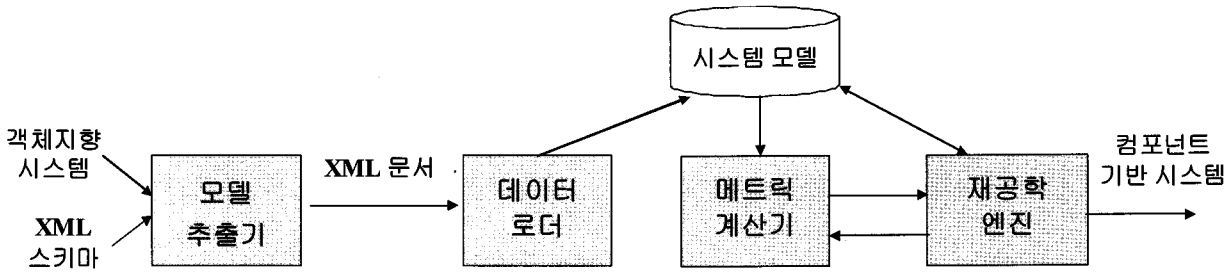
본 재공학 프로세스를 지원해주는 도구를 구현하여 실세계의 큰 규모의 시스템에 적용하고 그 결과를 분석한다. 아래 (그림 9)은 구현한 도구의 전체 구조를 보여준다.

코드를 읽어서 기 정의된 XML 스키마에 맞추어 시스템 모델을 각 클래스 마다 XML 문서로 추출해 주는 모델 추출기와, XML 문서 형태의 모델을 데이터베이스화 하는 데이터 로더, 저장된 시스템 모델 정보를 이용하여 정의된 메트릭을 계산해 주는 메트릭 계산기, 그리고 재공학 엔진으로 구성되어 있다. 재공학 엔진은 컴포넌트를 결정해 주는 부분, 즉 기본 컴포넌트 생성부터 컴포넌트 정제 단계까지 해서 한 컴포넌트에 각각 어떤 클래스들이 포함되는지를 결정한다. 재공학 엔진을 거쳐 최종적으로 컴포넌트 기반 시스템 모델이 생성된다. 모델 추출기는 JAVA로 구현된 프로그램으로부터 설계 정보를 추출해 주는 기존의 도구 [25]를 수정, 변경하여 이용하였다. 데이터 로더는 Java2 Standard

```
[가정]
- 가중치 wpri, wabs, wc, wg, wu, w1, w2와 복잡도 임계치 MAXCOX 는 미리 정의되어 있다.
- CANDIDATE(S')={ <Ci, Cj> | Ci, Cj ∈ S'∧COX(Ci) < MAXCOX∧COX(Cj ) < MAXCOX }
- Ci∪j는 Ci와 Cj를 병합한 컴포넌트를 가리킨다.

[클러스터링 프로세스]
while (CANDIDATE(S')  )
        S'=S ;
find <Cp, Cq> ∈ CANDIDATE(S') with the maximum CS(Cp, Cq) value.
        S' = S' - { Cp, Cq };
        S' = S'∪Cp∪q;
        CANDIDATE(S') = CANDIDATE(S') - { <Cp, Cq> };
        if  COX(Cp∪q) ≥ MAXCOX or Q(S') ≤ Q(S)
                break;
        else
                S=S';
                    for every <Ci, Cj> ∈ CANDIDATE(S'),
      if (Ci  == Cp or Ci  == Cq or Cj  == Cp or Cj == Cq )
                    Ci = Cp∪q or Cj = Cp∪q;
 end of while
```

(그림 8) 정제 프로세스



(그림 9) 재공학 도구 구조

```
- <CLS>
  - <TYPE>
      <name>QObject</name>
      <kind>concrete</kind>
    </TYPE>
  - <ATTRSET>
    - <attr>
        <fType>java.lang.String</fType>
        <fName>m_data</fName>
      </attr>
    + <attr>
    </ATTRSET>
  - <MSET>
    - <method>
        <signature>java.lang.String getData()</signature>
        <accessor>public</accessor>
      </method>
    + <method>
    - <method>
        <signature>void [init](java.lang.String, java.lang.String)</signature>
        <accessor>public</accessor>
      </method>
    </MSET>
    <PAR />
    <CP />
    <USE />
  </CLS>
```

(그림 10) 추출된 모델 정보(일부)

Edition(J2SE) 1.4를 이용하여 구현하였고 데이터베이스는 Oracle 9i Release 2를 이용하였다. 메트릭을 계산하는 메트릭 계산기는 Oracle 데이터베이스의PL/SQL로 구현하였으며, 데이터베이스의 테이블과 뷰, 함수와 내장형 프로시저를 이용하였다. 클래스 단위 메트릭의 경우 효율을 높이기 위해 연결 강도와 응집도는 클래스 정보 테이블과 뷰, 함수를 이용하여 자료 입력과 동시에 계산하고 복잡도는 입력을 마친 후 별도의 내장형 프로시저를 사용하여 계산한다. 합성 및 상속 기본 컴포넌트와 시스템 정리 단계 모두 PL/SQL을 이용하여 수행하도록 하고, 수행 과정에서 클래스 단위 메트릭, 뷰, 함수, 내장형 프로시저를 사용하였다. 정제 단계 역시 테이블과 PL/SQL을 이용하여 수행하였다. 다음 절에서는, 적용할 실제 시스템인 JBidWatcher에 대하여 설명한다.

4.1 JBidWatcher

JBidWatcher[13]는 오픈소스로서 JAVA로 구현된, 154개의 클래스로 구성된 경매(auction) 관리 시스템이다. ebay와 yahoo의 경매 사이트에 접속해서 사용자가 선택한 매물에 대한 입찰 관리, 매물정보, 비교 등을 수행해준다. 모델 추출기를 통하여 각 클래스에 대한 모델 정보가 XML 문서로 생성된다. (그림 10)은 추출된 모델 정보의 일부로서 클래스 ProxyClient에 대한 시스템 모델을 보여준다.

〈표 1〉 JBidWatcher의 합성, 상속, 기본 컴포넌트 개수

합성 관계	상속 관계	기본 컴포넌트 개수
47	75	57

〈표 2〉 JBidWatcher의 가중치 별 컴포넌트 생성 결과

가중치	w_1	w_2	*MAXCOX*(%)	최종 컴포넌트 개수
case1'	.5	.5	40	24
case2'	.5	.5	30	32
case3'	.5	.5	25	36
case4'	.4	.6	40	22
case5'	.4	.6	30	30
case6'	.4	.6	25	33

JBidWatcher의 합성 관계, 상속 관계, 이들 관계를 바탕으로 생성된 기본 컴포넌트의 개수는 아래 <표 1>과 같다.

기본 컴포넌트 생성 이후, 클러스터링을 통한 정제 단계에 들어간다. <표 2>는 w_1과 w_2, MAXCOX를 변경시킨 가중치

에 대해 생성된 최종 컴포넌트 개수를 보여준다. MAXCOX의 퍼센트의 의미는, 시스템을 구성하는 컴포넌트들의 복잡도의 총합인 시스템 복잡도에 대한 비율을 가리킨다. 가중치 w_{pri}, w_{abs}는 각각 0.4, 0.6, w_c, w_g, w_u는 각각 0.4, 0.3, 0.3로 동일하게 설정하였다. 이전 절에서 기술했듯이, 이들 가중치는 대소관계가 이미 설정되어 있고, 그 범위 내에서 값을 변경시키는 것이 실제 실험 결과에 거의 영향을 미치지 않아 제외하였다.

(그림 11)는 가중치 case2'의 정제 과정의 일부를 보여준다. 열 이름은 차례로 현재 프로세스 식별자(PROCESS_ID), 병합되는 컴포넌트 식별자 두 개(CLUSTERER, CLUSTEREE), 생성된 컴포넌트 식별자(NEW_ID)이고, 다음의 Q는 시스템 향상 수치이며 COX는 복잡도 임계치, ACCEPTANCE는 병합 여부를 Y/N으로 보여준다. TIMESTAMP는 수행 당시의 시간이며 SYS_CS와 SYS_COH가 각각 시스템 연결 강도와 시스템 응집도이다.

(그림 12)은 가중치 case2'의 정제 과정 중 후보 컴포넌트를 클러스터링한 경우만 추출한 것이다. 즉, 시스템의 품질 값인 Q가 1보다 크고 복잡도가 임계치보다 낮은 경우이다. 여기서, 시스템 품질 함수Q 값과 시스템 결합도(SYS_CS)와 시스템 응집도(SYS_COH)을 알 수 있다. 전체적으로 시스템 결합도는 낮아졌으며 시스템 응집도는 상승한 것을 알 수 있다.

JBidWatcher는 컴포넌트들간에 연결 강도가 고르게 분포되어 컴포넌트 병합이 하나의 컴포넌트에게 집중되지 않았고, 매 클러스터링 시에 줄어든 연결 강도가 차지하는 비중이 낮아 Q 값은 크게 상승하지 않았다.

다음은, 최종 생성된 컴포넌트에 대하여 설명한다. 컴포넌트를 보여주는 그림은 수행결과화면을 캡쳐한 것으로, 생성된 일부 컴포넌트와 해당 컴포넌트를 구성하는 클래스들에 대한 정보를 담고 있다. 해당 컴포넌트를 구성하는 클래스 명칭을 보면, 유사한 기능을 제공해 주는 클래스들로 구성

PROCESS_ID CLUSTERER CLUSTEREE NEW_ID ACCEPTANCE TIMESTAMP SYS_COH

P...	CL...	CLU...	NEW...	Q	COX	AC...	TIMES...	SYS_CS	SY...
408	10109	10184	10234	1,002467331339...	120,2	Y	15:16:34	2010	,61
408	10138	10172	10235	9,943548278295...	426,8	N	15:17:10	2009,4	,6
408	10141	10160	10236	9,967980116212...	101	N	15:17:44	2009,4	,61
408	10149	10160	10237	9,966935896773...	58,8	N	15:18:19	2009,4	,61
408	10162	10127	10238	9,929600772247...	50,8	N	15:18:54	2009,4	,6
408	10172	10162	10239	9,936071509097...	47,2	N	15:19:28	2009,4	,6
408	10176	10160	10240	9,879512475886...	27,4	N	15:20:02	2009,4	,6
408	10176	10218	10241	9,951266899073...	96	N	15:20:38	2009,4	,6
408	10182	10119	10242	9,921452436954...	123,2	N	15:21:13	2009,4	,6
408	10211	10153	10243	1,001458680324...	135	Y	15:21:48	2009,4	,61
408	10216	10172	10244	9,882572843195...	69,4	N	15:22:22	2004	,6
408	10216	10243	10245	9,997744685505...	200,6	N	15:22:56	1970,2	,6
408	10217	10144	10246	1,003092749950...	70,2	Y	15:23:30	2008,8	,62

(그림 11) JBidWatcher의 정제 프로세스(일부)

P...	CL...	CL...	NE...	Q	COX	A...	TIME...	SYS_CS	SYS_COH
408	10109	10184	10234	1,0024673...	120,2	Y	15:16:34	2010	,61
408	10211	10153	10243	1,0014586...	135	Y	15:21:48	2009,4	,61
408	10217	10144	10246	1,0030927...	70,2	Y	15:23:30	2008,8	,62
408	10229	10232	10249	1,0050043...	56,4	Y	15:25:11	2008,2	,62
408	10234	10243	10251	1,0083966...	254,6	Y	15:26:16	2005,2	,63
408	10216	10251	10252	1,0026734...	320,2	Y	15:26:51	1955,2	,62
408	10246	10182	10253	1,0001558...	139,8	Y	15:27:24	1954,6	,62
408	10195	10201	10256	1,0092675...	10,4	Y	15:28:54	1953,6	,63
408	10225	10112	10261	1,0088503...	414,4	Y	15:31:22	1943,8	,64
408	10253	10165	10265	1,0063681...	238,6	Y	15:33:18	1942,8	,65
408	10149	10223	10270	1,0046584...	100,6	Y	15:35:38	1941,6	,65
408	10162	10215	10275	1,0033537...	144,4	Y	15:37:55	1940,4	,66
408	10163	10249	10279	1,0051516...	105,6	Y	15:39:40	1939,2	,66
408	10212	10228	10284	1,0008839...	47	Y	15:41:47	1938	,66
408	10219	10284	10286	1,0080610...	211,6	Y	15:42:36	1936,8	,67
408	10270	10231	10290	1,0116178...	285,4	Y	15:44:11	1927,2	,68
408	10275	10290	10293	1,0097229...	429,8	Y	15:45:20	1926	,7
408	10279	10222	10299	1,0125492...	277,6	Y	15:47:13	1904	,71
408	10119	10154	10301	1,0096801...	59,8	Y	15:47:55	1902,4	,72
408	10138	10213	10307	1,0106636...	443,8	Y	15:49:59	1900,6	,74
408	10202	10299	10313	1,0024840...	297,4	Y	15:52:00	1896,6	,74
408	10252	10141	10331	1,0130111...	411,2	Y	15:57:16	1887,2	,75
408	10226	10261	10342	1,0035270...	431,4	Y	16:00:23	1883,6	,76
408	10218	10286	10358	1,0159099...	290,2	Y	16:04:31	1870,4	,78
408	10358	10160	10365	1,0007775...	300,2	Y	16:06:16	1865,6	,77
408	10230	10365	10374	1,0127724...	376,8	Y	16:08:12	1859,6	,79
408	10307	10143	10417	1,0016681...	446	Y	16:16:00	1837,6	,78

(그림 12) JBidWatcher 정제 프로세스 도중 컴포넌트의 병합

되어 있음을 알 수 있다. 주요컴포넌트들의 일부와 그들의 구성 클래스들을 보이고, 각 컴포넌트의 역할을 간단히 소개한다.

다음 (그림 13)은 각각 ebay 경매 서버(auction server)에 해당하는 컴포넌트들로서 매물에 대한 정보, 가격, 지속적 모니터링, 자동 입찰 등을 수행해 주며, 파서(parser) 컴포넌트(그림 14)를 이용하여 각 서버 페이지를 파싱한 후, 정해진 규칙에 맞게 변형시켜 보여준다. 마찬가지로 yahoo server 컴포넌트도 생성된다.

다음 (그림 15)는 HTTP Proxy 컴포넌트로서 실제 ebay와yahoo 서버에 HTTP로 접속할 때 행해지는 일들을 수행해준다.

(그림 16)은 경매 서버 관리자(auction server manager) 기능을 제공해 주는 컴포넌트이다. 그외 ConfigurationControl은 JBidWatcher 시스템 자체의 형상(configuration) 정보를 획득하고 관리해주는 컴포넌트이다. AuctionManager 컴포넌트는 경매 관리자(Auction Manager)의 전반적인 기능을 제공하는 컴포넌트로서 ebay와 yahoo 서버를 통해 매물을 검색, 추가, 삭제, 매물 회수 등을 수행한다. 그리고 논리적인 사용자 인터페이스(UI) 모델 (클래스명: AuctionsUIModel)에 대한 제어자(controller)의 역할을 수행한다. 실제 UI 구현 부분은 대부분 JAVA의 swing 패키지를 이용하였으므로 그 제어자만 존재하는 것이다. JBidWatch 컴포넌트는 주된 클래스(primary class)인 JBidWatch를 포함하는 컴포넌트로

23539	ebayServerRootCp	15018	ebayServer$JConfigEbayTab
23539	ebayServerRootCp	15019	ebayServer
23539	ebayServerRootCp	15020	AuctionServer
23539	ebayServerRootCp	15032	MessageQueue$Listener
23539	ebayServerRootCp	15040	SpecificAuction
23539	ebayServerRootCp	15054	ServerMenu
23539	ebayServerRootCp	15071	ebayServer$ebayServerMenu
23539	ebayServerRootCp	15076	AuctionInfo
23539	ebayServerRootCp	15079	JConfigTab
23539	ebayServerRootCp	15085	ebayServer$BadBidException
23539	ebayServerRootCp	15093	ebayServer$ebayAuction
23539	ebayServerRootCp	15149	XMLSerialize
23539	ebayServerRootCp	15154	CleanupHandler
23539	ebayServerRootCp	15160	XMLSerializeSimple

(그림 13) ebay server 컴포넌트

23679	JHTMLParserRootCpClusterCp	15031	JHTML$intPair
23679	JHTMLParserRootCpClusterCp	15040	SpecificAuction
23679	JHTMLParserRootCpClusterCp	15052	yahooServer$yahooAuction
23679	JHTMLParserRootCpClusterCp	15053	JHTMLParser
23679	JHTMLParserRootCpClusterCp	15076	AuctionInfo
23679	JHTMLParserRootCpClusterCp	15084	JDropListener
23679	JHTMLParserRootCpClusterCp	15093	ebayServer$ebayAuction
23679	JHTMLParserRootCpClusterCp	15096	JHTMLListener
23679	JHTMLParserRootCpClusterCp	15121	JDropHandler
23679	JHTMLParserRootCpClusterCp	15137	JHTML
23679	JHTMLParserRootCpClusterCp	15154	CleanupHandler
23679	JHTMLParserRootCpClusterCp	15164	JTableDrop

(그림 14) HTML Parser 컴포넌트

23642	ProxyClientRootInheritanceBasicCmp	15015	ProxyClient
23642	ProxyClientRootInheritanceBasicCmp	15048	JHTMLDialog
23642	ProxyClientRootInheritanceBasicCmp	15060	CookieJar
23642	ProxyClientRootInheritanceBasicCmp	15090	HTTPProxyClient
23642	ProxyClientRootInheritanceBasicCmp	15092	Base64
23642	ProxyClientRootInheritanceBasicCmp	15094	Base64$OutputStream
23642	ProxyClientRootInheritanceBasicCmp	15115	Base64$InputStream
23642	ProxyClientRootInheritanceBasicCmp	15135	Cookie
23642	ProxyClientRootInheritanceBasicCmp	15140	HTMLDump
23642	ProxyClientRootInheritanceBasicCmp	15141	JBidProxy

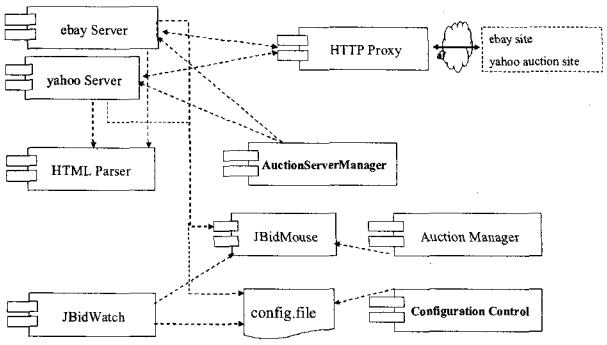
(그림 15) HTTP Proxy 컴포넌트

23645	JConfigFrameRootCpClusterCp	15017	SearchManager$URLSearcher
23645	JConfigFrameRootCpClusterCp	15032	MessageQueue$Listener
23645	JConfigFrameRootCpClusterCp	15035	EventLogger$EntryStatus
23645	JConfigFrameRootCpClusterCp	15043	TimerHandler$WakeupProcess
23645	JConfigFrameRootCpClusterCp	15047	JConfigFrame$JConfigWebserverTab
23645	JConfigFrameRootCpClusterCp	15049	UpdaterEntry
23645	JConfigFrameRootCpClusterCp	15062	AuctionServerManager
23645	JConfigFrameRootCpClusterCp	15066	DeletedManager
23645	JConfigFrameRootCpClusterCp	15078	JConfigStubTab
23645	JConfigFrameRootCpClusterCp	15079	JConfigTab
23645	JConfigFrameRootCpClusterCp	15087	JConfigFrame$JConfigMacBrowserTab
23645	JConfigFrameRootCpClusterCp	15088	SearchManagerInterface
23645	JConfigFrameRootCpClusterCp	15102	EventLogger
23645	JConfigFrameRootCpClusterCp	15105	JConfigFrame
23645	JConfigFrameRootCpClusterCp	15108	SearchManager$MyItemSearcher
23645	JConfigFrameRootCpClusterCp	15116	JConfigFrame$JConfigSecurityTab
23645	JConfigFrameRootCpClusterCp	15133	SearchManager$SellerSearcher
23645	JConfigFrameRootCpClusterCp	15146	AuctionServerManager$AuctionServerListEntry
23645	JConfigFrameRootCpClusterCp	15149	XMLSerialize
23645	JConfigFrameRootCpClusterCp	15151	SearchManager
23645	JConfigFrameRootCpClusterCp	15153	AuctionEntry
23645	JConfigFrameRootCpClusterCp	15160	XMLSerializeSimple

(그림 16) AuctionServerManager 컴포넌트

23673	CurrencyRootCpClusterCp	15055	ErrorManagement
23673	CurrencyRootCpClusterCp	15117	Currency$CurrencyTypeException
23673	CurrencyRootCpClusterCp	15138	Currency

(그림 17) Currency 컴포넌트



(그림 18) JBidWatcher 컴포넌트 다이어그램

서 사용자 인터페이스 정보를 이용하여 실제 시스템의 화면을 구성하고, 모든 쓰레드를 시작시킨다.

(그림 17)는 실제 대부분의 컴포넌트에서 이용되는 통화(通貨) 관련 컴포넌트이다. 본 실험에서는 Currency가 독립된 컴포넌트가 되었으나 복잡도 임계치를 높인 case1'에서는 다른 컴포넌트 내에 포함되었다. 이런 경우, 어느 하나의 컴포넌트에 포함되게 되면, Currency를 이용하는 수많은 컴포넌트들이 포함하고 있는 컴포넌트에 대하여 의존관계가 발생하게 되어 바람직하지 않다. 따라서 일종의 utility 컴포넌트처럼 따로 떨어져 있거나 규모가 작을 경우는 사용되는 컴포넌트 내로 복제할 수 있겠으나, 코드 중복은 기본적으로 유지보수성을 떨어뜨리므로 분리하는 것이 바람직하다. 이런 경우에는, 사용자가 개입하여 Currency 컴포넌트를 사용하는 독립된 컴포넌트로 분리해야 하도록 한다.

JBidMouse 컴포넌트는 마우스 제어 관련 기능을 제공해 주는 컴포넌트이다. Search 컴포넌트는 매물에 대한 검색을 수행하고 결과를 테이블로 표현해 주는 컴포넌트로서 여러 컴포넌트에서 사용하는 컴포넌트이다. (그림 18)은 주요 컴포넌트들을 중심으로 JBidWatcher 전체 구조를 보여주는 컴포넌트 다이어그램이다.

간단히 정리하면, ebayServer와 yahooServer는 HTTPProxy를 통하여 각 사이트에 접속하고 그 결과를 HTMLParser를 통하여 화면에 보여주며, AuctionServerManager에 의해 제어된다. ConfigurationControl은 파일 config.file로 저장된 형상정보를 제어하고 보여주며, JBidWatch는 주컴포넌트로서 사용자정보를 이용하여 첫 화면을 구성하고 쓰레드를 시작시킨다.

본 프로세스를 적용하여 생성된 컴포넌트들을 분석한 결과 대체로 기능적으로 관련 있는 클래스들로 이루어진 컴포넌트들이 추출되었다. 부가적으로, 이 컴포넌트들의 기능과 관계를 분석하여 전체 시스템에 대한 큰 그림을 얻을 수 있었다. 본 사례의 경우, 대략적으로 MVC(Model-View-Controller)구조를 반영함을 추측할 수있었다.

그리고 본 사례 연구를 통하여 정제 단계 시에 전체 시스템의 응집도는 높아지고 결합도가 낮아짐으로써(그림 12), 전체 시스템의 품질이 향상됨을 정량적으로 보였다. 또한 소프트웨어의 재사용성은 다수의 연구에서 응집도와 결합도, 복잡도, 이해도, 적응성, 이식성 등을 정의하고 이들로부터 재사용성을 추정하고 있는데 [2, 16-18, 26], 본 연구에서는

생성된 컴포넌트의 응집도가 증가하고 결합도가 감소하는 방향으로 재공학이 수행됨을 보임으로써 컴포넌트의 재사용성이 향상되었음을 간접적으로 보였다.

4.2 비교 및 평가

본 절에서는 본 연구와 공통점이 있는 관련 연구들에 대한 소개 및 비교를 통하여 본 연구의 적용성에 대하여 설명하고, 연구의 기대 효과를 기술한다.

객체지향 시스템으로부터 결합도, 응집도 등의 메트릭을 활용하여 컴포넌트를 추출하는 연구들이 다수 수행되었다. Price등은 재사용성 측정 메트릭을 사용하여 사용자의 관점에서 재사용성이 높은 컴포넌트를 식별, 생성하는 방법론을 제안하였다[11]. 우선 설계자가 주관적으로 객체 지향 코드를 분석하여 클래스들을 재사용성 수준에 따라 미래에 사용될 가능성이 높은 일반(general) 클래스 레벨 0에서 현 프로그램에만 적용 가능한 특수(special) 클래스까지 수준을 나눈다. 일반, 특수 클래스의 개념에 더하여 소프트웨어 설계자는 미래에 구현할 시스템에서 함께 사용될 가능성이 높은 클래스들의 계층(hierarchy)들 간에 '연관(Related)' 관계를 맺는다. 그리고 재사용성 있는 컴포넌트를 서로 연관된(Related) 클래스들의 그룹으로 간주한다. 도출된 일반, 특수 클래스와 그들간의 연관 관계를 바탕으로 재사용성을 측정하기 위해 여덟 가지 결합도 유형을 제안하였으며, 이 결합도 유형을 이용하여 향후 재사용을 증진시킨다. 각 결합도 유형은 컴포넌트 내의 일반 클래스와 특수 클래스들 간에 연관 관계의 유, 무에 따라 여덟 가지로 분류되었으며 바람직하지 않은 결합도에 대해 바람직한 형태로 변환 되도록 하는 리팩토링 기법을 제시하였다. 예를 들어, 결합도 개수(Coupling Count) 유형1은 모든 일반 클래스들에 대해 그들과 연관된 일반 클래스들간의 결합도 개수로, 이러한 결합도는 바람직하다. 그리고 유형5는 모든 특수 클래스들에 대해 그들과 연관된 일반 클래스와의 결합도 개수이며, 재사용성을 향상시키기 위해서는 유형1로 변경되어야 한다. Price등의 방법은 다른 관련 연구와는 달리 별다른 설계 정보를 요구하지 않고 프로그램으로부터 재사용 컴포넌트를 얻는다는 점에서 본 연구와 유사하지만 다음과 같은 차이가 있다. Price등의 방법에서는 일반, 특수 클래스 및 연관된 관계를 초반에 설정하는 것이 중요하다. 이를 위해서는 소프트웨어 설계자들이 어플리케이션 도메인과 시스템 종류에 대해 확고한 지식을 가지고 있어야 한다. 따라서 실제 규모가 큰 시스템에는 바로 적용되기가 어렵고, 역공학 등을 통해 시스템의 설계 패턴 같은 정보를 미리 얻는 사전 작업이 필요하다. 즉, 사용자의 개입도가 낮은 본 연구에 비하여 사용자의 개입도가 아주 높으며 크고 복잡한 시스템에 바로 적용되기 어렵다는 단점이 있다.

그 외, 대부분의 컴포넌트 추출 연구는 본 연구와 달리 CBD에 기반하여 여러 설계 정보를 활용하고 있다. Lee등은 컴포넌트의 결합도(coupling)와 응집도(cohesion), 의존, 인터페이스, 규모, 아키텍처를 고려한 컴포넌트 식별 방법을 제안하였다[8]. 사전작업으로 각 서브 시스템에 할당되는 클래스들과 유스 케이스들을 식별하고, 시퀀스 다이어그램을 통한 서브 시스템들 사이의 의존 관계 결정한 후, 의존 관계의 복잡도를 낮추는 방향으로 서브 시스템을 재조직화한다. 그 후, 응집도를 측정하기 위해 유스 케이스(use case) 다이어그램에서 주요 기능을 식별하고, 시퀀스(sequence)다이어그램과 콜레보레이션(collaboration) 다이어그램을 이용하여 각 기능에 해당되는 클래스를 중요도에 따라 키 엔티티 클래스부터 가장 낮은 수준의 키 클래스까지 다섯 가지 수준으로 분류한다. Lee등은 결합도를 객체 간의 메시지 개수에 기반한 상호작용 결합도와 클래스 사이의 정적 관계를 반영하는 정적 결합도로 분류하고, 최종 결합도는 상호작용 결합도와 정적 결합도의 곱으로 구한다. 클래스간의 결합도와 응집도를 기반으로 하여 도출된 클래스 관계 그래프에 대해 클러스터링 알고리즘을 수행하여 후보 컴포넌트들을 생성해 낸다. 이후몇 가지 가이드라인을 통해 잘 정의된 컴포넌트를 식별한다. [8]의 방법은 컴포넌트 생성시 클래스들 간의 결합도와 응집도를 주요 지표로 삼는다는 점에서 본 연구와 유사하지만, 본 연구가 재공학적 접근법이라면 [8]은 CBD중 이용되는 기법으로, 설계자의 적극적인 개입을 요하며, 분석, 설계 정보인 유스 케이스, 시퀀스, 콜레보레이션 다이어그램 등을 필요로 한다.

Jain은 비즈니스 도메인을 표현하는 객체 모델로부터 컴포넌트를 식별하는 방법을 제안하였다 [3]. 클래스 간의 연관 관계로부터 파악된 클래스 간의 정적 관계치와, 유스 케이스 다이어그램과 시퀀스 다이어그램에서 이끌어 낸 클래스 간의 동적 관계치를 이용한다. 정적 관계치는 클래스 사이의 연관(association) 관계를 이용하고, 동적 관계치는 유스 케이스들과 그들에 할당된 클래스들을 이용하여 구한다. 두 관계치를 이용하여 클래스 간의 클러스터링을 통해 컴포넌트를 생성한다. 클러스터링 과정에 임계치를 두어 클래스의 병합을 중단한다. 품질 개선을 위해 하나 이상의 서브 클래스를 포함하는 컴포넌트의 외부에 존재하는 수퍼 클래스를 컴포넌트 내부로 복제하도록 한다. 또한 클러스터링 알고리즘으로부터 나온 결과를 정제하기 위해 설계자나 시스템에 의해 각각 수동적, 자동적으로 휴리스틱들을 적용하도록 한다. 그러나 컴포넌트의 인터페이스 생성에 관한 언급이 없고, 한 수퍼 클래스의 각 서브 클래스들이 모두 다른 컴포넌트에 산재해 있을 경우에는 비효율적이다. 또한 프로세스를 수행 시, 유스 케이스 다이어그램이나 시퀀스 다이어그램 같은 분석 모델을 필요로 하고 있다.

본 연구의 접근법과 유사한[11]은 사용자의 개입도가 본 연구에 비해 상당히 높고 규모가 큰 프로그램에 바로 적용되기 어렵다. 그 외 대부분의 컴포넌트 추출 기법은 분석 및 설계 모델을 이용하여 본 연구보다 다양한 정보를 활용할 수 있으며, 사용자의 개입이 본 연구보다 높은 편이다. 그러나 재공학적인 접근법이 아니며, 기본적으로 시스템 분석 및 설계 문서가 존재한다고 가정하고 있으므로 본 연구와는 근본적으로 차이가 있다고 할 수 있다. 본 연구 결과는 설계 정보가 없는 레거시 시스템에 대하여 적용이 될 수 있으며, 시스템에 대한 사용자의 지식은 추출 프로세스의

처음부터 적용되는 것이 아니고 생성된 결과에 일부 적용된다. 따라서 본 연구는 문서화가 잘 되어 있지 않고 해당 시스템의 개발자가 존재하지 않는 상황에서 적용성이 높다고 할 수 있다. 또한 소프트웨어의 규모가 크더라도 초반에 자동적으로 컴포넌트 추출을 수행해 주므로 [11]에 비해 소프트웨어의 규모나 복잡도에 덜 좌우된다.

본 연구의 기대 효과는 다음과 같다.

첫째, 클래스 관계와 같은 의미적인 요소와 메트릭을 통한 정량적인 요소를 이용하여, 객체 지향 시스템이나 객체 지향 설계로부터 컴포넌트 기반 시스템으로 변환하는 전체적인 방법론을 제공하고 있다. 시스템 모델만 도출 된다면 클래스 다이어그램이나 시퀀스 다이어그램과 같은 분석, 설계 정보를 필요로 하지 않으므로 다른 CBD 기반 컴포넌트 추출 기법과 차이가 있다. 또한이러한 재공학 과정을 자동화함으로써 소프트웨어 개발 시 효율을 높일 수 있다.

둘째, 객체 지향 시스템에 대해 컴포넌트 기반 시스템으로 재공학하지 않더라도, 상호 관련 있는 클래스들을 함께 연관지어 볼 수 있는 관점을 제공하고 있다. 이는 시스템에 대한 이해도를 높일 수 있으며 유지보수에도 영향을 줄 것이다. 4.1절의 JBidWatcher 사례 적용 결과를 분석함에 있어, 설계 문서나 각 클래스에 대한 설명과 같은 시스템에 대한 구체적인 정보를 가지고 있지 않았으나, 프로세스를 통해 자동으로 관련성이 높은 클래스들을 한 단위의 컴포넌트로 제시해 주어 분석 과정이 비교적 용이하였다. 재공학 전의 154개의 클래스로 구성된 프로그램에 대해, 소스 코드 정보만을 가지고 시스템을 이해하기는 어려울 것이다. 또한 이것을 전체 클래스 다이어그램으로 보인다고 하더라도, 상속 및 합성 관계 이외에 수많은 클래스들 사이의 사용 관계가 존재하므로 이해가 쉽지 않다. 그러나 사례에서 보여지듯, 상호 연관된 클래스들을 컴포넌트라는 보다 큰 기능적 단위로 파악할 수 있어 전체 시스템을 이해하는 데 도움을 준다. 그리고, 위에서 언급한 바와 같이, 방대한 크기의 시스템을 재사용이 가능한 부분으로 분해하는 작업을 자동화하므로 시스템 분석에 필요한 비용을 상당 부분 줄일 수 있을 것이다.

셋째, 본 기법에서 사용된 클래스 클러스터링 기법은 여러 분야에서 응용 될 수 있다. 이를테면 MDA에서 PIM(Platform Independent Model)으로부터 PSM(Platform Specific Model)으로 변환할 때 컴포넌트 간의 통신량을 줄이기 위해 컴포넌트의 개수를 가급적 줄이기를 권장하는데[27], 이런 경우 상호 관련 있는 클래스들을 한 컴포넌트로 만들 때 본 기법을 활용할 수 있다.

넷째, 시스템에 대한 동적 분석을 추가하여 연결 강도 메트릭을 보완 하고 적절한 배치 알고리즘을 이용하면, 컴포넌트들을 향후 분산 환경으로의 배치 단위로 활용할 수 있다. 서론에서 언급한 바와 같이 컴포넌트는 객체보다 분산 환경에 더욱 적합한 단위이다. 그리고 본 기법의 적용 결과 생성된 컴포넌트들은 상호 의존 관계만 가지며, 현재 정의된 연결 강도는 클래스 사이의 호출 관계와 한 번의 호출 시 교환되는 자료의 양만 고려한다. 따라서 연결 강도를 정의할 때에 호출 회수를 추가한다면 전체 교환되는 통신량이 추정 가능하므로 배치 시에 활용할 수있다.

5. 결론 및 향후 과제

본 논문에서는 객체지향 시스템으로부터 모델 정보를 추출하고, 추출한 정보를 이용하여 정량적으로 클래스들을 관련지어 컴포넌트를 생성하는 기법을 제안하고 실제 시스템을 대상으로 실험을 하여 그 적용성을 검증하였다. 본 기법은 [12]의 방법론에 기반하고 있지만, 적용 대상이 객체지향 모델이 아니라 객체지향 시스템으로 프로세스를 확장시켰다. 또한 [12]에서 정의된 프로세스는 실제 큰 규모의 시스템에 적용하기에는 프로세스의 정의가 상세하지 않아 보다 체계적인 정의가 필요하였다. 따라서 본 방법론에서는 보다 세밀하게 정의된 시스템 모델을 이용하여 전체 프로세스를 수행하기 위하여 매 프로세스 단계마다 체계적인 알고리즘을 정의하고, 복잡도 메트릭에서 인터페이스 복잡도에 대한 고려를 추가하여 새로이 정의하였다. 유효성을 검증하기 위하여 전체적인 재공학 프로세스를 지원해 주는 도구를 개발하여 150여 개의 클래스로 이루어진, JAVA로 구현된 실세계 시스템인 JBidWatcher에 적용하였다. 실험 결과, 정제단계가 진행되면서 생성된 시스템의 응집도는 높아지고 결합도는 낮아져 전반적인 시스템의 품질은 높아졌고, 생성된 컴포넌트에 대한 분석을 수행한 결과 기능적으로 응집력있는 컴포넌트들이 생성되었다. 부수적으로 시스템의 전체적인 구조 파악을 용이하게 하여 시스템에 대한 이해도를 높일 수 있었다.

향후 연구에서는 시스템 분석에 초점을 맞추고자 한다. 시스템 모델과 메트릭 정의가 모두 시스템에 대한 단순 정적 분석으로부터 도출되었으나, 내부 구조에 대한 단순 정적 분석만으로는 실제 구조를 의미적으로 정확하게 반영하고 파악하는데 한계가 있다. 동적 분석을 수행하여 메소드의 호출 관계뿐 아니라 호출 횟수 및 유형, 메소드간의 병렬성(concurrency)등의 정보를 추출할 수 있다. 이렇게 추출된 정보를 기존의 정적 모델에 추가하고 연결 강도 메트릭을 호출 횟수를 이용하여 수정한다면 보다 정제된 결과가 나올 수 있을 것이다.

참 고 문 헌

[1] H. Mili, F. Mili, and A. Mili, "Reusing Software: Issues and Research Directions," IEEE Transactions on Software Engineering, Vol. 21, No. 6, pp. 528-562, 1995.

[2] H. Washizaki, H. Yamamoto, and Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components," In Proceedings of International Software Metrics Symposium, pp. 211-223, 2003.

[3] H. Jain, "Business Component Identification-A Formal Approach,"In Proceedings of Fifth IEEE International Enterprise Distributed Object Computing Conference, pp. 183-187, 2001.

[4] D. F. D'Souza and A. C. Wills, 'Object, Components, and Frameworks with UML: The CATALYSIS Approach,' 1st Ed., Addision Wesley, Harlow, England, 1998.
[5] C. Szyperski, 'Componet Software', 1st Ed., Addison Wesley, 1997.
[6] C. Atkinson et Al, 'Component-based Product Line Engineering with UML', 1st Ed., Addison Wesley, 2001.
[7] K. Bergner, A. Rausch, M. Sihling, and A. Vilbig, "Putting the Parts Together-Concepts, Description Techniques, and Development Process for Componentware", In Proceedings of 33th Annual Hawaii International Conference on System, Vol. 8, 2000.
[8] J. K. Lee, S. J. Jung, S. D. Kim, W. H. Jang and D. H. Han, "Component Identification Method with Coupling and Cohesion,"In Proceedings of Asia-Pacific Software Engineering Conference, pp. 79-86, 2001.
[9] 유영란, 김수동, "Use Case 및 클래스의 가중치 분석에 의한 컴포넌트 추출기법," 한국정보과학회 논문지B: 소프트웨어 및 응용, Vol. 28, No. 8, pp. 537-550, 2001.
[10] 최미숙, 윤용익, 박재년, "RUP기반의 컴포넌트 식별방법에 관한 연구," 한국정보처리학회논문지D, Vol. 9-D, No. 01, pp.91-102, 2002.
[11] M. W. Price, D. M. Needham and S. A. Demurijan, "Producing Reusable Object-Oriented Components: A Domain-and-Organization-Specific Perspective,"In Proceedings of Symposium on Software Reusability, pp. 41-50, 2001.
[12] 이은주, 신우창, 이병정, 우치수, "객체지향 모델로부터 정적 메트릭을 이용하여 컴포넌트 기반 시스템으로 변환하는 기법," 한국정보과학회 논문지B: 소프트웨어 및 응용, Vol. 31, No. 6, pp.728-740, 2004.
[13] http://www.jbidwatcher.com/
[14] E. Braude, 'Software Design', John Wiley & Sons, 2003.
[15] T. Vernazza, G. Granatella, G. Succi, L. Benedicenti, and M. Mintchev, "Defining Metrics for Software Components," In Proceedings of International Conference on Information Systems, Analysis and Synthesis, 2000.
[16] S.R. Chidamber, and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design,"IEEE Transaction on Software Engineering, Vol. 20, No. 6, pp. 476-493, 1994.
[17] J. M. Bieman and B. K. Kang, "Cohesion and Reuse in an Object-Oriented System," In Proceedings of ACM Symposium on Software Reliability, pp. 259-262, 1995.
[18] P. Vitharana, H. Jain, and Fatemeh, "Strategy-Based Design of Reusable Business Components,"IEEE Transaction on Systems, Man, and Cybernetics, Part C, Vol. 34, No. 4, pp. 460-474, 2004.
[19] E.S. Cho, M. S. Kim, and S. D. Kim, "Component Metrics to Measure Component Quality," In Proceedings of Asia-Pacific Software Engineering Conference, pp. 419-426, 2001.
[20] N. S. Gill and P. S. Grover, "Few Important Considerations for Deriving Interface Complexity Metric for Component-Based Systems,"ACM SIGSOFT Software Engineering Notes, Vol. 29, Issue 2, 2004.
[21] V. L. Narasimhan and B. Hendradjaya, "A New Suite of Metrics for the Integration of Software Components," In Proceedings of the Workshop on Object Systems and Software Architectures, 2004.
[22] L. C. Briand, S. Morasca, and V. R. Basili, "Property-Based Software Engineering Measurement,"IEEE Transaction on Software Engineering, Vol. 22, No. 1, pp. 68-86, 1996.
[23] 김태균, 'K교수의 객체지향 이야기', 초판, 도서출판 배움터, 2004.
[24] C. P. Willis, "Analysis of Inheritance and Multiple Inheritance," Software Engineering Journal, July, 1996.
[25] Chanjin Park, Yoohoon Kang, Chisu Wu, and Kwangkeun Yi, "A Static Reference Flow Analysis to Understand Design Pattern Behavior," In Proceeding of the Working Conference on Reverse Engineering, pp.300-301, 2004.
[26] G. Caldiera and V. R. Basili, "Identifying and Qualifying Reusable Software Components,"IEEE Computer, Vol. 24, No. 2, pp. 61-70, 1991.
[27] A. Kleppe, J. Warmer, and W. Bast, MDA explained: the Model Driven Architecture:practice and Promise, 1stEd,, Addision Wesley, Harlow, England, 2003.

이 은 주
e-mail: ejlee@knu.ac.kr
1997년 서울대학교 계산통계학과(학사)
1999년 서울대학교 전산과학과(석사)
2005년 서울대학교 기컴퓨터공학부(박사)
2005년 3월~2005년 10월 서울대학교 공과대학 BK 박사 후 연구원
2005년 11월~2006년 2월 삼성종합기술원 전문연구원
2006년 3월~현재 경북대학교 컴퓨터공학과 전임강사
관심분야: 소프트웨어 재공학, 메트릭, 웹서비스 기술, 웹응용 테스팅