공통성과 가변성 분석 기반의 컴포넌트 모델링 기법

(Commonality and Variability Analysis-based Component Modeling Technique)

김 수 동 † 조 은 숙 †† 류 성 열 †

(Soo Dong Kim)(Eun Sook Cho)(Sung Yul Rhew)

요 약 컴포넌트 기반의 소프트웨어 개발이 소프트웨어의 복잡성, 비용, 그리고 품질을 해결하기 위한 새로운 대안으로 소개되고 있다. COM, Enterprise JavaBeans, CORBA 컴포넌트 모델등과 같은 다양한 컴포넌트 아키텍쳐들이 소개되고 있으며 컴포넌트 기반의 소프트웨어 개발 방법론과 여러 CASE 도구들이 이를 지원하고 있다[1,2,3,4]. 그러나 현재 컴포넌트를 구현할 수 있는 기술은 제시되어 있지만 컴포넌트를 모델링하는 기법들에 대한 연구는 미약한 상태이다. 본 논문에서는 도메인 분석에서 공통성과 가변성 추출 및 클러스터링 기법을 이용한 컴포넌트를 분석하는 기법을 제시한다. 즉 컴포넌트 추출 기법, 컴포넌트의 핫스팟(또는 가변성) 표현 기법, 컴포넌트 요구사항 정의 기법 등을 제시한다. 컴포넌트 개발에 있어서 이러한 모델링 기법을 적용함으로써 컴포넌트를 효율적으로 개발할 수 있을 뿐만 아니라 재사용성이 높은 고품질의 컴포넌트 개발을 지원할 수 있다.

Abstract In order to address problems of complexity, high cost, and low quality of software, component-based software development(CBSD) was introduced. Various component architectures such as COM, Enterprise JavaBeans, and CORBA component model have been introduced, and CBSD methodologies and various CASE tools support component development[1,2,3,4]. Component implementation technologies are being proposed. However, modeling techniques for building software components have not been studied enough. In this paper, we propose a component analysis technique through commonality and variability identification, and clustering technique. That is, component identification technique, representation technique of component hot spot(or variability), and definition technique of component requirement specification are described. We believe that efficient components as well as high quality components can be developed through applying proposed identification techniques to component development.

1. 서 론

최근 몇 년 동안 소프트웨어 개발 형태가 급속도로 복잡해지고 있다. 클라이언트/서버 형태, 다양한 플랫폼의 지원, 그리고 복잡하고 다양한 사용자 인터페이스를 지원하는 어플리케이션의 요구 사항으로 인해 개발자들은 어플리케이션 개발에 있어서 새로운 접근을 시도하고 있다. 객체지향 기술, 어플리케이션 프레임워크, 혹은 파워 빌더나 델파이와 같은 4GL 등이 그러한 예가 된다. 이러한 새로운 접근은 어플리케이션 개발이나 유지보수에 있어서 개발 시간의 단축과 비용의 절감 효과를 가져왔으나 소프트웨어의 품질은 고객의 기대에 미치지 못하는 경우가 많이 발생되고 있을 뿐만 아니라 새로운 개념과 개념을 적용한 개발 기법이나 도구 등을 요구함으로 인해 새로운 기술을 효율적으로 적용하기에는 많은 시간과 노력이 소요될 뿐만 아니라 소프트웨어의 복잡도를 증가시키는 역효과를 가져오기도 하였다.

컴포넌트 기반의 소프트웨어 개발이 이러한 소프트웨

† 종신회원 : 숭실대학교 컴퓨터학부 교수
sdkim@computing.soongsil.ac.kr
syrhew@computing.songsil.ac.kr
†† 비 회 원 : 동덕여자대학교 정보학부 교수
escho@dongduk.ac.kr
논문접수 : 1999년 9월 17일
심사완료 : 2000년 6월 19일

어의 복잡성, 비용, 그리고 품질을 해결하기 위한 대안으로 소개되었다. Cox는 소프트웨어 컴포넌트를 전자 공학이나 기계 공학과 같은 하드웨어 공학에서의 집적 회로(IC)에 비유하여 소프트웨어 IC라고 정의하였다[5]. 이미 다른 공학 분야에서는 컴포넌트를 이용한 제품의 효율적인 개발이 보편화되어 있다. 전자 공학의 경우, 컴포넌트들은 스위치, IC, 전원 공급기 등과 같이 미리 만들어져서 시험이 완료된 구축 블록이다. 그런 빌딩 블록들에 대해서는 ISO와 같은 표준들이 마련되어 있다[6,7]. 이처럼 소프트웨어 컴포넌트들도 새로운 소프트웨어를 개발할 때 빌딩 블록으로 사용될 수 있다. 따라서 소프트웨어를 처음부터 전부 개발함으로써 생기는 비용이나 위험 요소들을 감소시킬 수 있게 된다.

최근에 COM, EJB, CORBA 컴포넌트 모델과 같은 컴포넌트 개발 기술들이 소개되고 있다. 컴포넌트 개발 기술들에는 컴포넌트 아키텍처, 컴포넌트 모델링 도구, 컴포넌트 개발이나 컴포넌트 기반의 소프트웨어 개발 기법에 이르기까지 다양하다[8,9]. 컴포넌트 관련 기술들 가운데 컴포넌트 개발이나 컴포넌트 기반의 소프트웨어 개발 기법에 관한 연구는 현재 많이 진행되어 있지 않은 상태이다. 따라서 본 논문에서는 재사용성이 높은 컴포넌트를 개발하기 위한 기법으로 컴포넌트를 모델링하는 기법을 제시한다. 특히 공통성 분석과 가변성 분석을 통해 컴포넌트 내에 포함될 기능들과 어플리케이션마다 변경될 수 있는 가변성들을 정의하는 기법들을 제시할 뿐만 아니라, 사용 예들과 클래스들을 클러스터링하는 알고리즘을 적용하여 컴포넌트를 식별하는 기법들을 구체적으로 제시한다.

본 논문의 구성은 제2장에서 기존의 컴포넌트 모델링 기법들을 소개하고 각각의 기법이 갖는 한계점을 살펴본다. 제3장에서는 공통성과 가변성 분석을 통한 컴포넌트 모델링 기법을 제시한다. 컴포넌트 모델링을 위한 업무들과 각각의 업무 내에서 수행해야 할 지침들을 구체적으로 제시한다. 제4장에서는 사례 연구를 통해 본 논문에서 제시한 기법의 실무적인 유용성을 살펴보고, 다른 모델링 기법들과의 비교를 통해 제안한 기법의 적용성을 살펴본다. 5장에서 결론 및 향후 연구과제를 제시한다.

2. 관련 연구

2.1 CBD with COOL:Spex

스털링 소프트웨어(Sterling Software) 사에서 개발한 컴포넌트 모델링 기법으로서 이 기법은 COOL:Spex이라는 모델링 도구를 이용해서 모델링이 가능하다[10, 11]. 이 기법은 Catalysis 방법론을 기반으로 한 기법으로서 그림1에 표현된 것처럼 컴포넌트 모델링을 위한 주요 단계를 요구사항 정의, 분석, 행위 명세, 아키텍처 모델링으로 구분하고 있다.

이 방법론은 요구사항 정의 단계부터 시작한다. 여기서는 사용 예(Use Case) 모델링과 비즈니스 타입 모델링 업무를 수행하는데 이 업무는 어플리케이션과 컴포넌트 요구 사항들을 식별하기 위한 것이다. 분석 단계에서는 도메인을 이해하기 위해 도메인 모델링 업무를 수행한다. 이 업무는 아키텍처 개발과 행위 명세 이전에 수행되어야 한다. 다음으로 행위 명세 단계를 거쳐 컴포넌트들을 모델링한 후 아키텍처 모델링 단계를 수행한다. 여기에 있는 컴포넌트 모델링 업무들은 반복적으로 수행된다.

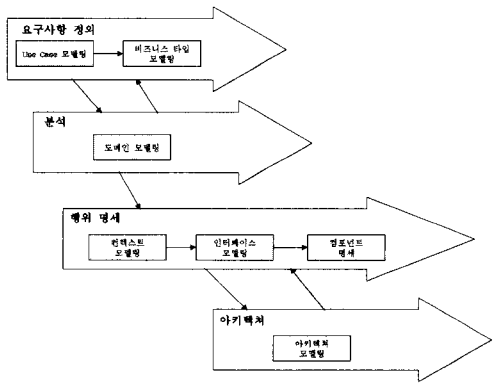

그림 1 CBD의 컴포넌트 모델링

CBD의 한계점은 컴포넌트를 모델링하기 위한 업무들이 정의되어 있으나 업무에 대한 지침이 구체적으로 명시되어 있지 않으며 특히 도메인 모델링과 사용 예 모델링에서 컴포넌트를 추출하기 위한 기법이 제시되어 있지 않다. 그리고 행위 명세 단계의 인터페이스 모델링 업무에서 인터페이스를 어떻게 정의할 것인지에 대한 지침이 빈약하다.

2.2 SCIPIO

새로운 시스템은 정보 기술 전략과 정보 기술 요구사항들을 컴포넌트 기반의 소프트웨어 개발로 자연스럽게 모델링할 수 있는 개발 프로세스와 개발 기법을 요구하고 있다. 이러한 요구사항을 만족시키기 위해 SCIPIO 컨소시엄에서 SCIPIO 방법론을 개발하였다. SCIPIO 방법론은 비즈니스 요구사항을 컴포넌트 기반의 소프트웨어 개발로 연결하는 방법론이다[12].

SCIPIO 방법론은 단지 컴포넌트 개발에만 초점을 두

기보다는 컴포넌트들을 기반으로 해서 어플리케이션을 개발하는 방법론에 초점을 두고 있다. 그러나 이 방법론에서 새로운 컴포넌트를 개발하는 업무들이 포함되어 있다. 여기에서는 컴포넌트 모델링과 관련된 업무로 컴포넌트 요구사항 명세서를 작성하는 분석 단계와 컴포넌트를 추출하고 인터페이스를 정의하는 설계 단계, 그리고 컴포넌트를 개발해서 시험하는 구현단계가 있다.

컴포넌트를 추출하는 업무는 기존에 이미 구축된 유사 어플리케이션들의 분석을 통해 이루어진다. 기존에 구축된 어플리케이션 요구사항 명세서들을 기반으로 각각의 어플리케이션의 기능과 다른 여러 가지 기술 요구사항과 제약사항들을 기술한 컴포넌트 요구사항 명세서들을 산출한다. 설계 단계에서는 분석 단계에서 작성한 컴포넌트 요구사항 명세서를 기반으로 컴포넌트를 식별하고, 식별된 컴포넌트들에 대해 인터페이스를 정의하고 서로 연동하는 컴포넌트들 간의 상호 흐름을 정의한다. 그리고 컴포넌트에 대한 설계 명세서를 작성한다. 컴포넌트 구현 단계에서는 컴포넌트 설계 명세서를 기반으로 컴포넌트를 개발하는 업무와 개발된 컴포넌트에 대해 단위 시험하는 업무로 구성된다.

SCIPIO의 한계점은 업무들간의 입· 출력물이 어떻게 반영되는지 구체적으로 명시되어 있지 않다. 그리고 컴포넌트 개발과 관련된 구체적인 모델링 지침이 충분히 기술되어 있지 못함으로써 컴포넌트 추출이나 설계에 대한 지침이 부족하다.

2.3 Catalysis

Catalysis는 UML을 기반으로 컴포넌트 기반의 소프트웨어 개발을 지원하는 방법론이다[13,14,15]. 이 방법론은 Fusion과 Syntropy와 같은 2세대 방법론들을 확장시켰는데 프레임워크 기반의 개발 형태를 지원하고 분석에서부터 구현까지 이르는 모델링 기법들을 정의하며 컴포넌트들을 명세화하는 기법 등을 제시하고 있다. Catalysis 방법론에서는 소프트웨어 개발 프로세스를 여러 다양한 프로젝트에 적용하기 위해 단일 프로세스보다는 프로세스 패턴(Process Pattern)을 제공하고 있다.

Catalysis의 컴포넌트 기반의 개발 프로세스는 상호 연동할 수 있는 컴포넌트들을 가지고 어플리케이션을 개발하는 데 초점을 두고 있다. Catalysis 개발 프로세스는 분석, 설계, 구현, 그리고 시험 단계로 구성되어 있다. 여기에 해당하는 개발 프로세스가 그림2에 제시되어 있다.

Catalysis의 한계점으로는 우선 업무들 가운데 컴포넌트를 추출하는 방법이 구체적으로 제공되어 있지 않다. 둘째, 컴포넌트 타입 모델에서 클래스들과 오퍼레이션들을 분리시킴으로 인해서 컴포넌트 인터페이스와 클래스의 오퍼레이션의 구분이 불분명하다.

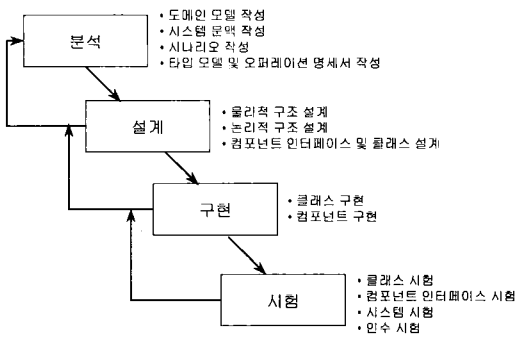

그림 2 컴포넌트 및 어플리케이션 개발 프로세스

이외에 기타 관련 컴포넌트 모델링 기법으로 Uniface 7.2[19]와 Fusion 2.0[20]이 있다. Uniface 7.2는 3차원 패러다임인 컴포넌트 기반 개발, 컴포넌트 기반 디플로이먼트(Deployment) 및 컴포넌트 기반 보급으로 구성되어 있다. 따라서 생산성이 높고 변경에 빠르게 대처할 수 있다. UNIFACE 개발 방법론은 개발 단계의 업무들 간의 흐름이 제시되어 있고, 업무들에서 사용되는 입.출력 물에 대한 정의는 체계적으로 제공되고 있으나 특정 업무 내에서 수행해야 할 지침들은 구체적으로 정의되지 않다. 따라서 개발자들이 실무에 적용하는데 어려움이 따른다. Fusion 2.0 방법론은 Fusion, OMT, Objectory, Booch 및 기타 여러 주요 방법론들의 기법들을 반영함으로써 개발자들이 작은 규모이든지 대형의 원격에서 팀별로 개발하든지 간에 효율적으로 컴포넌트나 소프트웨어를 개발할 수 있도록 하고있다. 이 방법론은 활동, 기법, 산출물 및 팀 기반의 프로젝트 계획을 포함한 전체 생명주기를 지원하고, 사용 예 중심의 접근법을 사용한다. 또한 OLE/COM과 OMG의 CORBA 아키텍쳐를 이용한 시스템 아키텍쳐와 컴포넌트 정의를 위한 기법들을 제시한다. 그러나 컴포넌트 명세서를 개발하는 기법은 구체적으로 정의되어 있으나, 컴포넌트 추출이나 컴포넌트 커스터마이즈 기법은 구체적으로 기술되어 있지 않다. 또한 공통성과 가변성이 단지 사용 예 모델링을 이용해서만 표현되어 있다. 이러한 정보만으로는 컴포넌트들을 분석하고 설계하기에 어려움을 초래한다.

3. 컴포넌트 모델링 프로세스

이 분석의 목표는 여러 유사한 어플리케이션들이 속하는 문제 도메인을 정확히 파악해서 컴포넌트에 관한

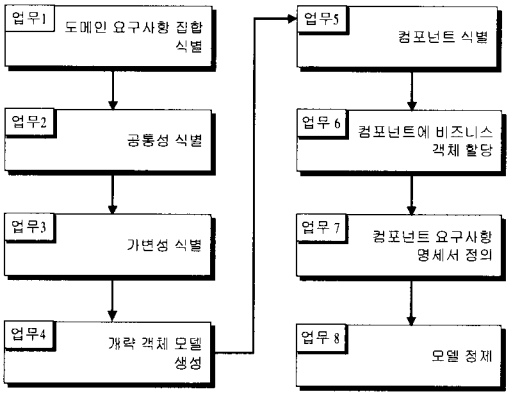

그림 3 컴포넌트 모델링 업무 흐름

요구사항들을 추출하는 것이다. 특히 비즈니스 컴포넌트는 그 도메인에서 여러 어플리케이션들이 공통적으로 사용하는 기능들을 재사용할 수 있는 단위로 구성되어야 하기 때문에 도메인 분석이 매우 중요한 부분을 차지한다. 컴포넌트 모델링에서 수행되는 업무들에 대한 정의와 흐름이 그림3에 표현되어 있다.

3.1 도메인 요구사항 집합 식별(업무 1)

이 업무의 목적은 재사용성이 높은 컴포넌트들을 개발하기 위해 특정 도메인에 대한 어플리케이션 요구사항 명세서들을 수집함으로써 도메인에 대한 요구사항 집합을 추출하기 위한 것이다. 따라서 도메인 요구사항 집합을 추출하기 위해서는 우선적으로 특정 도메인에 대한 어플리케이션 요구사항 명세서들을 수집해야 한다.

그러나 어플리케이션 요구사항 명세서들을 수집했을 경우, 수집된 어플리케이션 요구사항 명세서들이 미흡하거나 불완전할 수 있다. 이를 보완하기 위해 추가적으로 해당 도메인에 대해 가상의 어플리케이션 요구사항 명세서를 작성한다. 가상의 어플리케이션 요구사항 명세서를 작성하기 위해서는 도메인 분석을 통해 이루어지게 되는데, 도메인 분석은 도메인 전문가와 인터뷰를 하거나 어플리케이션 개발 사이트를 방문하거나 어플리케이션 시스템을 시뮬레이션 해 본다.

추출된 기능적 요구사항 명세서 집합을 보면 어플리케이션마다 용어들이 불일치하는 경우가 존재한다. 따라서 표1에 나타난 것처럼 도메인에 적합한 표준 용어들을 식별하여 용어 사전(Term Dictionary)을 작성한 후, 요구사항 명세서 집합을 정제하는 과정이 필요하다. 수집된 어플리케이션 요구사항 명세서들을 가지고 각각의 어플리케이션 요구사항 명세서에서 기능적 요구사항 명세서들을 추출한다. 컴포넌트는 특정 도메인에 속하는

표 1 용어사전

용어명	용어설명	범주	별명
고객	고객은 이름, 나이, 주소, 전화번호를 가지며, 상품을 주문하게 된다.	엔티티	사용자
이름	이름은 8자리 문자로 표현된다.	속성	성명
...	...	...	...

표 2 기능사전

기능번호	기능이름	기능설명	사용속성
F1	SearchCustomer( )	이 기능은 고객의 주민등록 번호를 가지고 특정 고객을 찾는 기능이다.	customerId
...	...	...	...

여러 어플리케이션들 간의 공통된 기능들에 초점을 두기 때문이다.

또한 정제된 요구사항을 기반으로 표2에 나타난 것처럼 도메인에 대한 기능 사전(Function Dictionary)을 작성하게 되는데, 이는 이후 사용 예 모델링과 컴포넌트 추출 업무에 사용하기 위함이다.

이처럼 도메인에 대한 기능들을 식별하면 각각의 어플리케이션 별로 기술된 기능 요구 사항들을 기능 사전에 정의된 기능들 단위로 정규화 시킨다.

3.2 공통성 식별(업무 2)

이 업무의 목적은 도메인 요구사항 집합에서 공통된 기능들을 추출하는 것이다. 이 업무에서는 공통성을 추출하여 이를 표현하기 위한 수단으로 UML의 사용 예 다이어그램과 사용 예 명세서를 사용한다. 우선 그림4에 표현된 것처럼 어플리케이션들 간에 중첩된 공통 기능들을 식별한다. 식별된 공통 기능들에 대한 요구사항 명세서를 도메인 요구사항 집합으로부터 식별한다.

식별된 공통 기능 요구사항 집합으로부터 행위자를 식별한다. 행위자를 식별하는 과정은 Jacobson의 방법을 적용한다[16,17]. 또한 공통 기능들을 기반으로 사용 예들을 식별하는데, 서로 응집도가 높은 기능을 그룹화하여 사용 예 단위로 추출한다. 여기서 응집도가 높은 기능들은 의미적으로 기능 유사성이 높은 기능들을 기반으로 추출하고 또한 추가적으로 기능 사전에서 같은 속성 또는 데이타를 가지고 처리하는 기능들을 기반으로 추출한다.

사용 예들을 식별하면 사용 예들 간에 중첩된 기능들

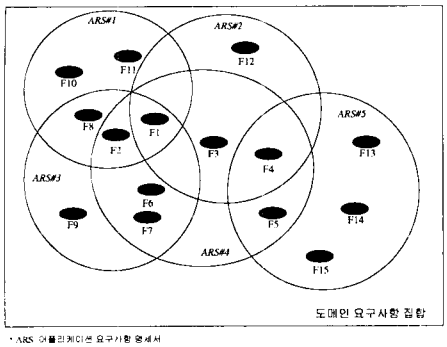

그림 4 공통기능 추출

이 존재할 수 있다. 이와 같은 여러 사용 예들 사이에 중첩되어 있는 기능들은 하나의 독립된 사용 예로 정의한다. 그리고 기존에 중첩되었던 사용 예들과 <<uses>> 구문으로 사용 예들을 연결시켜준다. 식별된 사용 예들과 행위자들을 가지고 공통된 사용 예들에 대해 사용 예 다이어그램을 작성한다. 사용 예 다이어그램을 작성한 후, 각각의 사용 예 별로 사용 예 명세서를 작성한다.

3.3 가변성 식별(업무 3)

이 업무의 목적은 특정 도메인에 대해 어플리케이션마다 변경될 수 있는 부분들을 식별하는 것이다. 어플리케이션마다 변경될 수 있는 있는 부분을 가변성(Variability)[18]이라고 하는데, 본 논문에서는 이 가변성의 부분을 세가지 부분인 속성, 로직, 워크플로우로 나누어서 추출한다. 여기서 로직이라 함은 기능은 동일하지만 어플리케이션 마다 기능의 구현이 달라 질 수 있는 부분을 의미한다. 워크플로우는 한 사용 예 내에서 객체들 혹은 객체와 컴포넌트들 간의 메시지 흐름을 말한다.

```
구조적 영어 형태의 Use Case 포맷
  Main Flow
    1. 오퍼레이션 이름 1
    2. 오퍼레이션 이름 2 - E2
    3. 오퍼레이션 이름 3 -A3, V3
    4. ...
  Alternative Flows
    A3.1: 오퍼레이션 이름1
    A3.2: 오퍼레이션 이름2
    ...
  Exception Flows
    E2: throw 예외명
    E3: throw 예외명
  Variant Flows
    V3.1: 오퍼레이션 이름
```

그림 5 구조적 영어 기반의 사용 예 명세서

표 3 공통성/가변성 테이블

사용예 이름 \ ARS번호		ARS#1	...	ARS#n	공통성	가변성
사용예 1	속성	카드명,카드번호,유효기간	...	카드명,카드번호,비밀번호,유효기간	카드명,카드번호,유효기간	비밀번호
	로직	1,2,3	...	1,2v2.1, 3	1, 3	2v2.1
	워크플로우	1→2→3	...	1→2v2.1→3	1→2(2v2.1)→3	없음
...	...	...	...	...	...	...
공통성 정도	속성	3/3	...	4/3		
	로직	3/2	...	3/2		
	워크플로우	1	...	1		
가변성 정도	속성	0	...	1		
	로직	0	...	1		
	워크플로우	0	...	0		

이러한 가변성들을 추출하기 위해서 사용 예 명세서를 참조해야 하는데, 사용 예 명세서는 텍스트 중심으로 기술되어 있기 때문에 이를 구조적 영어 형태로 재구조화해야 용이하게 추출할 수 있다. 즉 사용 예 명세서의 메시지 흐름을 오퍼레이션 단위로 표현하고, 각각의 오퍼레이션에서 변경이 발생하는 부분을 V 번호로 표현하고, 선택 흐름과 변경 흐름을 A 번호, E 번호로 표기한다(그림 5 참조)

속성은 각각의 어플리케이션의 사용 예 단위로 기술된 사용 예 명세서에서 추출한다. 로직과 워크 플로우는 구조적 영어 형태의 사용 예 명세서에서 찾은 후 기술한다. 기술된 내용들을 기반으로 공통성 부분에는 여러 어플리케이션들 간에 공통된 속성,로직,워크플로우를 추출하여 기술하고, 가변성 부분에는 어플리케이션마다 변경될 수 있는 속성, 로직, 워크플로우를 추출하여 기술한다. 세가지의 가변성을 추출하면서 표3에 제시한 "공

표 4 공통성 가변성 측정 메트릭

척도의 유형	공통성	가변성
속성	ARS의 속성 개수/사용 예의 공통 속성 개수	ARS의 속성 개수/사용 예의 가변 속성 개수
로직	ARS의 로직 개수/사용 예의 공통 로직 개수	ARS의 로직 개수/사용 예의 가변 로직 개수
워크 플로우	ARS의 워크플로우 개수/사용 예의 공통 워크플로우 개수	ARS의 워크플로우 개수/사용 예의 가변 워크플로우 개수

통성/가변성 테이블"을 이용하여 표현한다.

또한 각각의 어플리케이션에 있는 속성, 로직, 메시지 흐름이 해당 도메인 상에서 공통성이나 가변성의 정도가 얼마가 되는지를 표4에 표현된 메트릭을 이용하여 공통성 정도와 가변성 정도 부분에 기술한다.

다음으로는 공통성/가변성 테이블에 있는 각각의 가변성에 대해 가변성이 미리 결정될 수 있는지 혹은 예측 불가능한지를 판단하여 값(Predictable 또는 Non-Predictable로 표시)을 결정한다. 그리고 가변성에 대해 가변성의 범위를 선정한다. 예를 들어 전자 상거래에서 결제는 범위는 카드 결제 또는 현금 결제로 선정할 수 있다. 또한 가변성에 대해 컴포넌트에서 디폴트로 제공할 수 있는 값을 결정하는데, 이는 공통성/가변성 테이블의 공통성 정도를 가지고 결정한다. 즉 공통성 정도의 값이 가장 낮은 값을 디폴트 값으로 결정한다. 이처럼 가변성에 대한 번호, 이름, 타입, 범위, 디폴트 값, 결정 값 등을 표 5에 제시된 가변성 목록에 기술한다.

표 5 가변성 목록

번호	가변성 이름	가변성 타입	범위	디폴트 값	결정 값
V1	Compute Payment ()	로직	{카드결제, 현금결제}	카드 결제	Non-Predictable
...	...	...	...	...	...

3.4 개략 객체 모델 생성(업무4)

지금까지 특정 도메인 내에 있는 여러 어플리케이션들 사이에 공통적인 기능들을 식별하여 공통성과 가변성들을 추출하는 기법을 설명하였다. 이러한 공통된 기능들에서는 특정 기능을 수행하는 데 필요한 정보들이 포함되어야 한다. 이 업무에서는 지금까지 추출한 공통 사용 예들에 대해 도메인에서 필요한 클래스들을 추출하여 개략적인 클래스 다이어그램을 작성한다.

우선 공통된 기능 요구사항 명세서 집합, 용어 사전, 그리고 도메인 지식을 기반으로 공통 사용 예들에 대해 후보 클래스들을 추출한다. 후보 클래스들을 추출하는 기법은 Rumbaugh의 OMT 방법[13]을 적용한다. 추출된 후보 클래스들을 정제하고, 정제된 클래스들의 속성들과 관계들을 식별하여 개략적인 클래스 다이어그램을 작성한다. 그리고 기능 사전과 도메인 지식을 기반으로 추출한 클래스에 해당하는 오퍼레이션들을 추가한다.

3.5 컴포넌트 식별(업무 5)

이 업무의 목적은 지금까지 추출한 사용 예들과 클래스들을 기반으로 컴포넌트를 식별하는 것이다. 본 논문에서는 컴포넌트를 식별하기 위한 기법으로 사용 예들을 클러스터링하는 기법과 사용 예와 클래스들을 클러스터링하는 기법을 제시한다.

표 6 사용 예/클래스 매트릭스

사용 예 \ 클래스	C1	C2	C3	C4
사용 예 1	C	R	C	R
사용 예 2	W	R	C	C
사용 예 3	R	C	R	R

첫째, 사용 예들을 클러스터링한다는 것은 사용 예들 간의 관계를 고려하여 서로 응집력이 높은 사용 예들을 클러스터링하는 것을 의미한다. 사용 예들 간의 관계에는 <<uses>> 관계와 <<extends>> 관계가 존재한다. <<uses>> 관계는 여러 사용 예들 간에 공통적으로 사용하는 사용 예이기 때문에 사용 예들 간의 클러스터링에서는 일단 고려하지 않는다. <<extends>> 관계는 한 사용 예에서 가변적으로 발생하는 부분을 별개의 사용 예로 처리했기 때문에 <<extends>> 관계에 있는 사용 예들은 하나의 사용 예로 클러스터링한다.

다음으로는 사용 예들과 클래스들 간의 관계를 기반으로 서로 관련성이 높은 사용 예와 클래스들을 클러스터링 한다. 클러스터링을 하기 위해서는 표 6처럼 우선적으로 사용 예와 클래스들 간의 관계를 표현하는 매트릭스(Matrix)를 작성하고, 이 매트릭스에 해당 사용 예와 클래스 간의 관계 값들을 부여하게 된다. 본 논문에

```
int init_C, init_R, end_C, end_R = 1;
int n = # of classes;
int m = # of use cases;
while(init_C < n && init_R < m) {
  for ( l = end_C; l ≤ n; l++) {
      if (M(init_R, l) == 'C' | 'D' | 'W') {
          temp = Column(end_C);
          Column(end_C) = Column(l);
          Column(l) = 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++;
          }
      }
  }
  if (init_C == n ?: n : init_C++);
  if (init_R == m ?: m : init_R++);
}

(참고)
      init_C: 컬럼의 시작 위치
      end_C: 컬럼의 끝 위치
      init_R: 행의 시작 위치
      end_R: 행의 끝 위치
      C: "Create", D: "Delete", W: "Write"
      Column(end_C) : 현재 end_C의 열
      Row(end_R): 현재 end_R의 행
```

그림 6 사용 예/클래스 클러스터링 알고리즘

표 7 클러스터링 알고리즘 적용 결과

클래스 / 사용 예	C1	C3	C4	C2
사용 예 1	C	C	R	R
사용 예 2	W	C	C	R
사용 예 3	R	R	R	C

서는 관계 유형을 4가지인 생성(Create), 삭제(Delete), 쓰기(Write), 읽기(Read)로 구분하여 유형들에 대해 우선 순위를 부여한다.

우선 순위는 생성>삭제>쓰기>읽기 순으로 부여된다. 따라서 한 사용 예에서 클래스가 생성도 되고 쓰기도 되면 가장 우선 순위가 높은 생성으로 값을 할당 한다.

다음으로는 이처럼 할당되어 정의된 매트릭스를 가지고 그림 6에 제시된 클러스터링 알고리즘을 통해 서로 관련 있는 사용 예들을 클러스터링 한다. 이와 같은 클러스터링 알고리즘을 적용하면 표 7처럼 서로 관련 있는 사용 예들과 클래스들이 그룹화되어 표현된다.

클러스터링 알고리즘을 적용한 결과를 보면 사용 예1과 사용 예 2가 클래스 C1, C3, C4를 생성하기 때문에 이 두개의 사용 예는 한 컴포넌트로 클러스터링 되며, 사용 예 3은 단지 클래스 C2만을 생성하기 때문에 사용 예 3은 별개의 컴포넌트로 구성된다.

3.6 컴포넌트에 비즈니스 객체 할당(업무 6)

이 업무는 이전 업무에서 추출된 컴포넌트들에 포함되어야 할 클래스들을 할당한다. 각각의 컴포넌트 별로 포함되어야 할 클래스들은 이전 업무에서 클러스터링 된 클래스들을 해당 컴포넌트에 할당한다. 그러나 한 클래스가 여러 컴포넌트에서 필요할 경우가 발생한다. 이러한 경우에는 클래스의 성격을 고려한다. 만일 클래스가 일시적인 데이타와 오퍼레이션을 갖는 클래스이면(Transient 클래스), 해당 클래스를 복제하여 여러 컴포넌트들에 할당시킨다. 그러나 클래스가 영구적인 데이타를 갖는 경우는 그림 7에 표현된 것처럼 그 클래스를 생성하는 컴포넌트에 할당시키고, 그 클래스를 참조하는 컴포넌트와 종속성(Dependency) 관계로 표현한다.

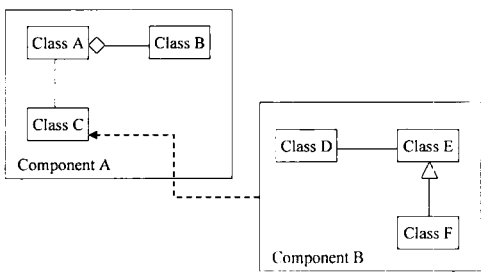

그림 7 클래스와 컴포넌트와의 종속성 관계

3.7 컴포넌트 요구사항 명세서 정의(업무 7)

지금까지 분석한 결과들을 기반으로 컴포넌트 설계를 위한 정보로 사용될 컴포넌트 요구사항 명세서를 작성한다. 그림 8에 기술된 것처럼 컴포넌트 요구사항 명세서에는 컴포넌트의 이름, 개요, 컴포넌트 내에서의 작업 흐름, 컴포넌트 내에 포함되어 있는 클래스들, 사용 예들, 그리고 서로 관계를 맺고 있는 컴포넌트, 컴포넌트의 공통성과 가변성에 대한 부분들이 기술된다.

```
<<컴포넌트 요구사항 명세서>>
1. 컴포넌트 이름:
2. 개요:
3. 메시지 흐름
    3.1 사용 예 1의 메시지 흐름:
    3.2 사용 예 2의 메시지 흐름:
4. 클래스 이름:
5. 참조 사용 예 이름:
6. 관련 컴포넌트:
7. 공통성/가변성 테이블:
8. 가변성 목록:
```

그림 8 컴포넌트 요구사항 명세서

3.8 모델 정제(업무 8)

이 업무의 목적은 지금까지 수행한 컴포넌트 모델링 과정의 모든 업무들의 결과물들을 검토하여 정제하는 것이다. 검토 및 정제과정에서 산출물들에 대해 산출물의 완전성과 산출물들 간의 일관성 등을 검토한다. 우선적으로 어플리케이션들 간의 공통된 기능적 요구 사항들은 모두 사용 예 다이어그램에 반영되어야 한다. 즉 공통된 기능적 요구 사항들은 적어도 한 개의 사용 예에는 반영되어야 한다. 추출된 각각의 사용 예에 대해서 사용 예 명세서가 정의되어야 한다. 또한 각각의 추출된 컴포넌트에 대해서, 컴포넌트 요구사항 명세서가 정의되어야 한다. 만일 한 클래스가 두개 이상의 컴포넌트에 의해 참조된다면, 그 클래스는 한 컴포넌트에 할당이 되어야 하고, 참조되는 다른 컴포넌트들과는 종속성 관계를 맺어야 한다. 그리고, 컴포넌트 요구사항 명세서의 '클래스 이름' 부분에는 반드시 하나 이상의 클래스가 명시되어야 하고, '참조 사용 예 이름' 부분에는 하나 이상의 사용 예가 명시되어야 한다.

4. 실험 및 평가

이 장에서는 본 논문에서 제안한 컴포넌트 모델링 기법을 전자 상거래 도메인에 적용한 실무 예제를 통해 제안한 모델링 기법의 실무 적용성과 효율성을 살펴보

고 기존의 컴포넌트 모델링 기법들과의 차이점을 살펴본다.

4.1 사례 연구

인터넷과 웹 기술의 급속한 발전으로 웹 어플리케이션 유형들 가운데 급성장하고 있는 분야로 전자 상거래 어플리케이션을 들 수 있다. 이를 바탕으로 본 논문에서는 사례 연구로 전자 상거래 도메인을 선택하여 이 도메인에서 재사용할 수 있는 기능들을 식별하여 컴포넌트 단위로 모델링 하였다.

우선 도메인 분석 단계의 첫번째 업무인 도메인 요구사항 집합 수집 업무를 통해 전자 상거래 도메인 상에서 개발된 여러 전자 상거래 시스템들의 요구 사항 명세서들을 수집하였다. 이러한 요구사항 명세 집합들과 도메인 지식을 바탕으로 전자 상거래 도메인에서 사용하는 용어들을 정립하여 표 8에 제시된 것처럼 용어 사전을 작성하였다.

표 8 전자 상거래용 용어 사전

용어명	용어설명	범주	별명
주문	주문에는 고객이 상품을 주문한 주문 날짜, 주문 번호, 주문 수량, 주문 금액, 주문 품목 들이 포함되어 있다.	엔티티	구매
결제	고객이 상품을 주문하여 구매하기 위한 것으로서, 결제 형태는 크게 카드 결제와 현금 결제로 나누어져 있다.	엔티티	지불
...	...	...	...

이러한 용어 사전을 기반으로 해당 도메인에 대한 요구사항 명세서들을 정제한다. 정제된 요구사항 명세서 집합들과 도메인 지식으로 해당 도메인의 주요 기능들을 식별하여 기능 사전을 작성하였다.

표 9 전자 상거래용 기능 사전

기능 번호	기능명	기능 설명	범주	사용속성
F1	주문 하기	결제 방법과 배달지 정보를 입력하도록 요청한 후, 입력된 정보를 가지고 결제 방법을 확인한 후 결제 처리를 한다.	주문 관리	결제방법, 배달지
F2	주문 조회	고객이 주문한 내역을 조회한다.	주문 관리	사용자ID, 비밀번호, 주문번호, 주문일자
...	...	...	...	...

이러한 기능 사전들을 기반으로 전자 상거래 어플리케이션 요구사항 명세서의 기능적 요구 사항들을 정규화 시켰다.

다음으로는 두번째 업무인 공통성 추출 업무로서 이 업무에서는 그림 9에 표현된 것처럼 정규화 된 기능들을 바탕으로 전자 상거래 어플리케이션 요구사항 명세서들을 통합하여 그들 간에 공통된 기능들을 추출하였다. 수집한 모든 어플리케이션 요구사항 명세서에 모두 공통적으로 존재하는 기능도 있고, 일부는 몇몇 어플리케이션 요구사항 명세서에서만 공유되는 기능이 있다.

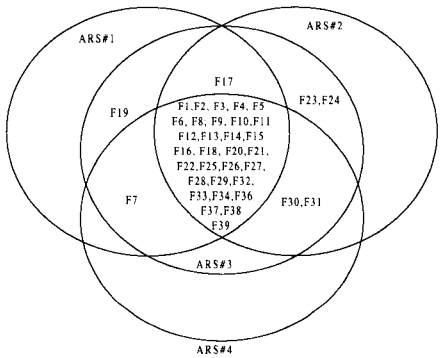

그림 9 애플리케이션들 간의 공통 기능 추출

이처럼 추출된 공통 기능들을 가지고 사용 예 모델링을 하였다. 모델링 한 결과 중의 하나인 사용 예 다이어그램이 그림 10에 표현되어 있다.

그림 10 전자 상거래 도메인의 사용 예 모델링

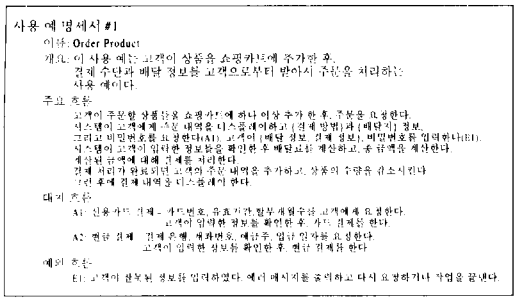
사용 예 명세서 #1
이름: Order Product
개요: 이 사용 예는 고객이 상품을 쇼핑카트에 추가한 후, 결제 수단과 배달 정보를 고객으로부터 받아서 주문을 처리하는 사용 예이다.
주요 흐름
고객이 주문할 상품들을 쇼핑카트에 하나 이상 추가 한 후, 주문을 요청한다.
시스템이 고객에게 주문 내역을 디스플레이하고 {결제 방법}과 {배달지} 질문.
그리고 비밀번호를 요청한다(A1). 고객이 {배달 정보}, {결제 정보}, {비밀번호}를 입력한다(E1).
시스템이 고객이 입력한 정보들을 확인한 후 배달료를 계산하고, 총 금액을 계산한다.
계산된 금액에 대해 결제를 처리한다.
결제 처리가 완료되면 고객의 주문 내역을 추가하고, 상품의 수량을 감소시킨다
(각) 후에 결제 내역을 디스플레이 한다.
대치 흐름
A1: 신용카드 결제 - 카드번호, 유효기간, 할부개월수를 고객에게 요청한다.
고객이 입력한 정보를 확인한 후, 카드 결제를 한다.
A2: 현금 결제 - 결제은행, 계좌번호, 예금주, 입금 일자를 요청한다.
고객이 입력한 정보를 확인한 후, 현금 결제를 한다.
예외 흐름
E1: 고객이 잘못된 정보를 입력하였다. 에러 메시지를 출력하고 다시 요청하거나 작업을 끝낸다.

그림 11 'Order Product' 사용 예 명세서

추출된 사용 예들에 대해 각각의 사용 예 단위로 사용 예 명세서를 작성하였다(그림 11 참조).

세번째 업무인 가변성 추출 업무에서는 이전 업무에서 추출한 공통성들을 기반으로 각각의 공통성에서 어플리케이션 마다 변경될 수 있는 가변성들을 식별하여 공통성/가변성 테이블을 작성하였다. 공통성/가변성 테이블을 작성하기 이전에 가변성 식별을 용이하도록 하기 위해 이전 업무에서 작성한 사용 예 명세서를 오퍼레이션 단위로 기술하였다. 'Order Product' 사용 예 명세서를 오퍼레이션 형태로 구조적 영어로 표현한 예가 그림 12에 제시되어 있다.

가변성 목록에는 속성에 대한 가변성, 로직에 대한 가변성, 그리고 워크플로우에 대한 가변성이 있는데, 일례로 전자 상거래 도메인에서는 로직에 대한 가변성은 어플리케이션마다 지불하는 로직이나 배달 처리하는 로직

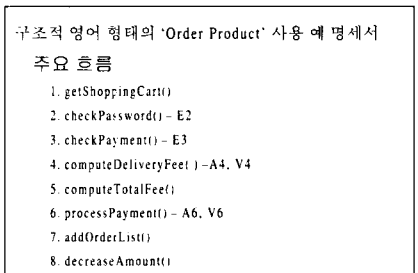
구조적 영어 형태의 'Order Product' 사용 예 명세서
주요 흐름
1. getShoppingCart()
2. checkPassword() - E2
3. checkPayment() - E3
4. computeDeliveryFee( ) -A4. V4
5. computeTotalFee()
6. processPayment() - A6. V6
7. addOrderList()
8. decreaseAmount()
9. deleteShoppingCart()
대치 흐름
A4.1: computeDeliveryFee(Ship)
A4.2: computeDeliveryFee(Airline)
A4.3: computeDeliveryFee(Land)
A6.1: processPayment(CreditCard)
A6.2: processPayment(Online)
예외 흐름
E2: throw PasswordException()
E3: throw PaymentException()

그림 12 구조적 영어 기반의 사용 예 명세서

표 10 전자 상거래 도메인의 공통성/가변성 테이블

		ARS#1	ARS#2	ARS#3	공통성	가변성
Order Product	속성	회원 ID, 비밀번호, 쇼핑카드 내역(물품명, 제조원 단가, 소계, 총합계액), 주문일자, 주문번호, {카드 결제, 현금결제}, {배달 정보}	회원 ID, 비밀번호, 쇼핑카드 내역(물품명, 제조원 단가, 소계, 총합계액), 주문일자, 주문번호, {카드 결제, 현금결제}, {배달 정보}	회원 ID, 비밀번호, 쇼핑카드 내역(물품명, 제조원 단가, 소계, 총합계액), 주문일자, 주문번호, {카드 결제, 현금결제}, {배달 정보}	회원 ID, 비밀번호, 쇼핑카드 내역(물품명, 제조원 단가, 소계, 총합계액), 주문일자, 주문번호, {카드결제, 현금결제}, {배달정보}	변화정보도
	로직	1,2,3,4,5,6,7,8,9	1,2,3,4,v4.1, 4v4.2, 4v4.3 5,6,7,8,9	1,2,3,4,5,6,v6.1, 6,v6.2 7,8,9	1,2,3,5,7,8,9	4v4.6v6
	워크플로우	1→2→3→4→5→6→7→8→9	1→2→3→4→4v4.1|4v4.2,4v4.3→5→6→7→8→9	1→2→3→4→5→6→6v6.1|6v6.2→7→8→9	1→2→3→4→5→6→7→8→9	None

표 11 전자 상거래 도메인의 가변성 목록

번호	가변성 이름	가변성 타입	범위	디폴트 값	결정 값
V1	process Payment	로직	{CreditCard, Online}	CreditCard	Non-Predictable
V2	compute Delivery Fee()	로직	{Ship,Airline, Land}	Land	Non-Predictable
...	...	...	...	...	...

이 가변적인 부분으로 추출되었다. 추출된 공통성/가변성에 대한 목록이 표 10에 제시되어 있다.

또한 추출된 가변성들에 대한 가변성 목록이 표 11에 제시되어 있다.

네번째 업무인 개략적인 객체 모델 생성에서는 전자 상거래 도메인에 대한 정적인 측면을 모델링하는 업무로서, 이 도메인의 클래스들을 추출하여 개략적인 클래스 다이어그램으로 모델링 하였다. 이 도메인에 대해 추출한 클래스들에는 고객, 주문, 배달, 상품, 지불 등과 같은 클래스들이 포함되어 있고, 각각의 클래스가 지니는 속성들과 클래스들 간의 관계들이 개략 클래스 다이어그램으로 그림 13에 표현되어 있다.

다섯번째 업무인 컴포넌트 추출에서는 지금까지 모델링 한 사용 예들과 클래스들을 기반으로 전자 상거래

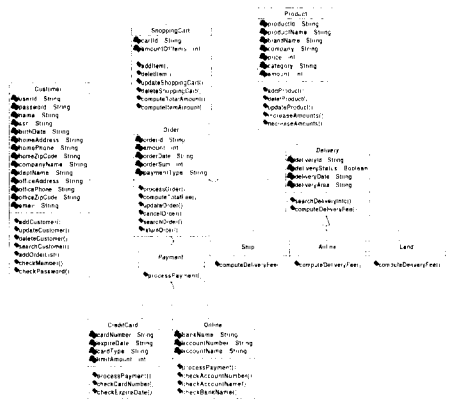

그림 13 전자 상거래 도메인의 개략 클래스 다이어그램

표 12 전자 상거래 도메인의 사용 예/클래스 매트릭스

	customer	product	Line item	order	payment	delivery
joinMembership	C	N	N	N	N	N
searchMemberInfo	R	N	N	N	N	N
updateMemberInfo	W	N	N	N	N	N
deleteMemberInfo	D	N	N	N	N	N
orderProduct	R	R	R	C	C	C
searchProductInfo	N	R	R	N	N	N
searchDeliveryInfo	R	N	N	R	R	R
addProductInfo	N	C	C	N	N	N
updateProductInfo	N	W	W	N	N	N
...	...	...	...	...	...	...

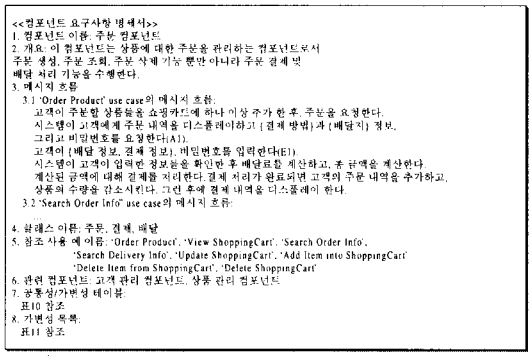

그림 15 컴포넌트 요구사항 명세서

도메인의 컴포넌트들을 추출하는 과정이다. 컴포넌트를 추출하기 위해 표 12처럼 사용 예와 클래스들 간의 매트릭스를 작성하여 본 논문에서 제시한 클러스터링 알고리즘을 적용하여 관련 사용 예와 클래스들을 클러스터링하여 컴포넌트를 추출하였다. 사례 연구로 적용한 전자 상거래 도메인에서는 '고객 관리', '상품 관리', '주문 관리' 3의 컴포넌트로 추출하게 되었다.

여섯번째 업무에서는 추출된 컴포넌트들에 포함되어야 할 클래스들을 할당하는 업무를 수행하였는데, 추출된 클래스들을 3개의 컴포넌트들로 할당한 결과가 그림 14에 표현되어 있다. 특히 주문에서는 상품이나 고객에 대한 정보를 참조하기 때문에 주문 컴포넌트는 고객 컴포넌트와 상품 컴포넌트와 종속 관계를 맺고 있다.

일곱번째 업무에서는 추출된 컴포넌트들에 대해서 컴포넌트 요구사항 명세서를 작성하는 업무로서, 본 논문에서 제시한 컴포넌트 요구사항 명세서를 기반으로 추출된 3개의 컴포넌트들에 대해서 컴포넌트 요구사항 명세서를 작성하였다(그림 15 참조).

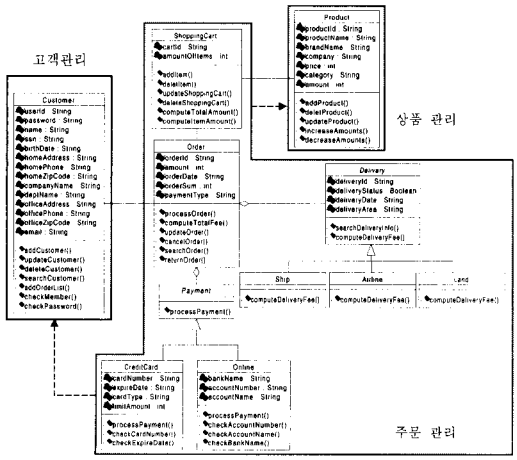

그림 14 전자 상거래 도메인의 컴포넌트 추출

4.2 평가

이 절에서는 본 논문에서 제안한 컴포넌트 모델링 기법과 기존의 컴포넌트 모델링 기법들과의 차이점을 살펴본다.

표 13 기존 기법들과의 차이점

모델링 기법 / 비교항목	CBD	SCIPIO	Catalysis	제안된 기법
업무 정의	O	O	O	O
지침 정의	O	O	O	O
공통성 추출	O	X	O	O
가변성 식별	X	X	X	O
컴포넌트 식별	X	X	X	O
모델링 언어	UML기반	UML기반	UML기반	UML기반

본 논문에서 제안한 모델링 기법은 기존의 기법들에 비해 컴포넌트 모델링에 있어서 구체적인 지침을 제공하고 있다. 특히 도메인 분석에서 공통성 추출이나 가변성 추출과 같은 업무는 재사용성이 높으면서 효율적인 컴포넌트를 추출하기 위한 기반을 마련해 준다.

또한 기존의 기법들에서는 컴포넌트 추출에 대한 지침이 텍스트 중심으로 기술되어 있고 구체적인 기법이 미약하였으나, 제안한 기법에서는 컴포넌트 추출에 있어서 매트릭스와 클러스터링 알고리즘을 적용함으로써 보다 정형화되고 체계적인 컴포넌트 추출을 제공한다. 이처럼 기존의 기법들과 본 논문에서 제안한 기법들과의 차이점이 표 13에 제시되어 있다.

6. 결론 및 향후 연구과제

본 논문에서는 컴포넌트 모델링 프로세스를 제안하였

고 컴포넌트 모델링 기법의 구체적인 지침들을 제시하였다. 또한 사례 연구를 통해 본 논문에서 제안한 컴포넌트 모델링 기법의 적용성을 살펴 보았다. 본 논문에서 제시한 모델링 기법은 기존의 기법들에 비해 보다 정형화하고 체계적인 지침들을 제시하였기 때문에 고품질의 컴포넌트를 효율적으로 개발할 수 있도록 한다. 향후 연구과제로는 특정 컴포넌트 아키텍처를 기반으로 한 컴포넌트 설계 및 구현 기법들을 제시함으로써 컴포넌트 개발의 전체 개발 프로세스를 정의하는 것이다.

참 고 문 헌

[1] Rogerson D., Inside COM, Microsoft Press, 1997.

[2] Microsoft Crop., The Component Object Model Specification, Microsoft Press, 1995.

[3] Object Management Group, CORBA Components, December 1998.

[4] Sun Microsystems, Enterprise JavaBeans Specification, at URL: http://www.javasoft.com, 1999.

[5] Cox B., Object-Oriented Programming: An Evolutionary Approach, Addison-Wesley, 1986.

[6] Pfister C., Component Software: A Case Study using BlackBox Components, Oberon Microsystems, Inc., June 1997.

[7] Szyperski C., Component Software: Beyound Object- Oriented Programming, Addison Wesley Longman, Reading, Mass., 1998.

[8] Kozaczynski Wojtek and Booch G., Component-Based Software Engineering, IEEE Software, pp.34-36, Sept./Oct. 1998.

[9] Brown A. W. and Wallnau K. C., The Current State of CBSE, IEEE Software, pp.37-46, Sept./Oct. 1998.

[10] Short K., Component Based Development and Object Modeling, Sterling Software, 1997.

[11] Harmon P., Visual Modeling Tools, CASE vendors, and component methods, Component Strategies, 1999.

[12] Veryad R., SCPIO: Aims, Principles and Structure, SCPIO Consortium, April 1998.

[13] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-Oriented Modeling and Design, Prentice-Hall, 1991.

[14] Rational Software Corp., Unified Modeling Language(UML) Summary, 1997.

[15] D'souza D. F. and Wills A. C., Objects, Components, and Components with UML, Addison-Wesley, 1998.

[16] Jacobson, I., et al. Object-Oriented Software Engineering, New York: ACM Press, 1992.

[17] Booch G., Rumbaugh J., and Jacobson I., The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[18] James O. Coplien, Daniel Hoffman, and David Weiss, Commonality and Variability in Software Engineering, IEEE Software, pp.37-45, November 1998.

[19] Compuware Corp., UNIFACE Development Methodology V7.2, COMPUWARE Corp., 1998.

[20] HP Company, Engineering Process Summary: Fusion 2.0, Hewlett-Packard Company, January 1998.

조 은 숙

1993년 2월 동의대학교 전산통계학과 졸업(이학사). 1996년 2월 숭실대학교 대학원 컴퓨터학과 졸업(공학석사). 2000년 2월 숭실대학교 대학원 컴퓨터학과 졸업(공학박사). 2000년 9월 ~ 현재 동덕여자대학교 정보학부 교수. 관심분야는 객체지향 개발 방법론, 컴포넌트 모델링 기법, 컴포넌트 기반의 소프트웨어 공학, 분산 컴퓨팅

김 수 동

정보과학회논문지 : 소프트웨어 및 응용 제 27 권 제 3 호 참조

류 성 열

1997년 2월 아주대학교 컴퓨터공학부(공학박사). 1997년 3월 ~ 1998년 3월 George Mason University 교환 교수. 1981년 3월 ~ 현재 숭실대학교 컴퓨터학부 교수. 1998년 3월 ~ 현재 숭실대학교 정보과학대학원 원장. 1998년 3월 ~ 현재 숭실대학교 전자계산원 원장. 관심분야는 리엔지니어링, 분산객체 컴퓨팅, 소프트웨어 재사용