레거시 시스템을 현대화하기 위한 유스케이스 기반의 컴포넌트 추출 방법

(A Use-case based Component Mining Approach for the Modernization of Legacy Systems)

김 현 수 † 채 흥 석 †† 김 철 홍 †††

(Hyeon Soo Kim) (Heung Seok Chae) (Chul Hong Kim)

요 약 레거시 시스템은 입증된 안정성과 신뢰성을 갖고 있을 뿐만 아니라 그것의 개발 과정에 많은 투자와 수년간의 축적된 경험과 지식이 투입되었기 때문에 다양한 조직의 핵심 비즈니스 응용 시스템을 오랫동안 지원해왔다. 그런데 웹을 기반으로 한 e-비즈니스 환경의 출현으로 이러한 핵심 비즈니스는 웹 기반의 환경에서 동작할 필요가 강하게 대두되었다. 이것은 새로운 비즈니스 환경에서 경쟁력이 되기 때문이다. 따라서 여러 조직들은 새로운 e-비즈니스 응용 시스템에서 재사용하기 위해 레거시 시스템에 묻혀 있는 비즈니스 가치를 찾아야할 필요를 느끼게 되었다. 본 논문에서는 특정 비즈니스 서비스를 수행하는 컴포넌트를 추출하기 위한 체계적인 접근 방법을 제안한다. 이 컴포넌트들은 레거시 시스템의 자산들로 구성되며 새로운 플랫폼으로 도입될 것이다. 컴포넌트 추출 과정은 여러 개의 작업들로 이루어진다. 먼저, 비즈니스 프로세스를 실현하고 있는 유스케이스가 파악된다. 다음으로, 유사한 기능성을 갖는 유스케이스를 통합하기 위해 파악된 유스케이스별로 설계 모델을 구축한다. 세 번째 단계에서는 설계 모델을 바탕으로 컴포넌트 후보를 도출하고, 컴포넌트 후보들 간에 공유되는 공유 요소들을 파악하고 컴포넌트 후보들을 수정한다. 또한 비즈니스 컴포넌트를 J2EE/EJB 환경에 도입하기 위하여 세 개의 보다 작은 규모의 컴포넌트들로 세분한다. 마지막으로, 컴포넌트가 제공하는 기능에 대한 인터페이스를 정의한다.

키워드 : 레거시 시스템 현대화, 컴포넌트 추출, 엔터프라이즈 자바 빈즈

Abstract Due to not only proven stability and reliability but a significant investment and years of accumulated ,experience and knowledge, legacy systems have supported the core business applications of a number of organizations over many years. While the emergence of Web-based e-business environments requires externalizing core business processes to the Web. This is a competitive advantage in the new economy. Consequently, organizations now need to mine the business value buried in the legacy systems for reuse in new e-business applications. In this paper we suggest a systematic approach to mining components that perform specific business services and that consist of the legacy system's assets to be leveraged on the modern platform. The proposed activities are divided into several tasks. First, use cases that realize the business processes are captured. Secondly, a design model is constructed for each identified use case in order to integrate the use cases with the similar functionalities. Thirdly, we identify component candidates from the design model and then adjust the component candidates by considering common elements among the candidate components. And also business components are divided into three more fine-grained components to deploy them onto J2EE/EJB environments. Finally, we define the interfaces of components which provide functionalities of the components as operations.

Key words : Legacy systems modernization, Component mining, Enterprise JavaBeans (EJB)

† 종신회원 : 충남대학교 전기정보통신공학부 교수
hskim401@cnu.ac.kr
†† 비 회 원 : 부산대학교 컴퓨터공학과 교수
hschae@pusan.ac.kr
††† 비 회 원 : 한국전자통신연구원 소프트웨어공학연구팀 연구원
kch@etri.re.kr
논문접수 : 2004년 1월 29일
심사완료 : 2005년 5월 26일

1. 서 론

현재 많은 수의 은행, 증권회사, 보험사 및 일반 기업들에서 COBOL 언어로 작성된 레거시 시스템을 회사의 중요한 업무 처리 과정에서 활용하고 있다. 그런데 이러한 COBOL 레거시 시스템들은 대부분 오래 전에 개발

되어서 그것들을 유지보수하기 위한 노력이 상당히 요구된다. 그렇지만 현실적으로 COBOL 시스템을 잘 다룰 줄 아는 개발자의 수가 계속해서 줄어들고 있는 실정이기 때문에 기존의 COBOL 시스템에 새로운 기능을 추가하거나 기존의 일부 기능을 향상시키기란 거의 불가능한 상황이다[1]. 그렇지만 이러한 레거시 응용 시스템은 그것들의 개발 및 운용에 많은 투자와 수년간 축적된 지식과 경험이 투입되었고 또한 아직까지 조직에서 중요한 업무를 수행하기 때문에 그것들은 여전히 중요한 가치를 지닌다고 볼 수 있다[2,3].

한편, 인터넷의 도입은 모든 산업의 모습을 근본적으로 바꾸어 놓았다. 제품 중심의 서비스에서 편리성을 지닌 부가가치 서비스 및 대고객 서비스가 최고의 가치를 지닌 고객 중심의 서비스로 바뀌어 가고 있다[4]. 따라서 인터넷 환경으로의 연결은 오늘날 기업 경쟁체재에서 경쟁력을 확보하기 위한 필수적인 요소로 인식되고 있다. 경쟁력을 확보하고 유지하기 위해 기업이 취할 수 있는 가장 적절한 해결책은 기존의 레거시 시스템들과 인터넷을 기반으로 한 새로운 프로세스들을 통합하는 것이다[5,6]. 그러나 기업들이 갖고 있는 레거시 시스템의 대부분은 단일체(monolithic) 구조이고, 동종 플랫폼을 목표로 개발되었기 때문에 분산 환경, 이기종 아키텍처, 컴포넌트 기반 등과 같은 특성을 지닌 요즈음의 기술 시스템에는 잘 맞지 않는다[1].

레거시 시스템 현대화 전략은 레거시 시스템의 자산을 새로운 플랫폼으로 이주하여 새로운 비즈니스 모델을 지원하게 함으로써 시스템의 기능을 연장하고 확장하게 하는 접근 방법을 말한다. 오늘날 레거시 시스템 현대화 방법은 비즈니스 관련 정보와 지식을 독립된 조각으로 즉, 컴포넌트로 나누는 것이다. 이러한 컴포넌트들은 기본적으로 특정 비즈니스 서비스를 수행하는 객체들의 모임으로서, 명확하게 정의된 APIs(Application Program Interfaces)를 갖으며 새로운 표준 프로토콜을 통해 접근 가능하다[1,7]. 레거시 시스템 내의 특정 기능을 수행하는 요소들을 독립적인 컴포넌트들로 변환함으로써 조직 내의 다른 시스템 또는 외부 시스템들이 직접 접근하고 쉽게 접근할 수 있도록 레거시 시스템을 개방할 수 있다. 이런 변환을 통해 레거시 시스템 내에 내재되어 있는 비즈니스 관련 정보와 지식이 기업의 경쟁력이 된다[1].

이 논문에서는 레거시 시스템 현대화 전략의 일환으로 레거시 시스템을 J2EE/EJB와 같은 분산 컴포넌트 환경으로 이주하기 위한 방안에 관해 논의한다. 레거시 시스템 현대화 과정은 먼저, 비즈니스 레거시 시스템으로부터 독립적으로 접근 가능한 컴포넌트들을 추출하기 위한 과정과 계속해서, 추출된 레거시 컴포넌트를 J2EE/EJB 환경에서 동작하는 새로운 형태의 컴포넌트로 변환하는 과정으로 이루어진다. 이 논문에서는 레거시 시스템으로부터 독립적으로 접근 가능한 컴포넌트들을 추출하는 방법에 초점을 두고 기술한다. 레거시 시스템으로부터 컴포넌트를 추출하기 위한 방법으로 기존에 수행된 연구에는 서브시스템을 컴포넌트로 추출하는 방법[8], 특정 기능과 연관된 요소를 컴포넌트로 추출하는 방법[3,9], 작업흐름을 컴포넌트로 추출하는 방법[10,11] 등과 같이 여러 유형의 시도가 있었지만 기존 연구들은 컴포넌트의 독립성, 전체 시스템을 의미 있는 요소로 분할하여 컴포넌트화하는 측면에서 취약점을 지니고 있었다. 따라서 본 연구에서는 이러한 문제점을 해결하고자 레거시 시스템에서 발견되는 유스케이스를 기반으로 컴포넌트를 추출하는 방법을 제시한다. 이를 위한 구체적인 컴포넌트 추출 활동은 몇 개의 세부적인 작업으로 이루어진다. 첫 번째 작업은 레거시 시스템의 화면 흐름 정보를 이용하여 비즈니스 프로세스를 실현하고 있는 유스케이스를 파악하는 것이다. 다음으로, 유사한 비즈니스 기능성을 갖는 유스케이스를 통합하여 보다 큰 규모의 컴포넌트를 구성하기 위하여 파악된 유스케이스별로 설계 모델을 구축한다. 설계 모델은 유스케이스의 수행에 관련된 시스템 요소들로 표현된다. 세 번째로 컴포넌트를 생성하기 위한 기초 작업으로서 유스케이스의 실행에 관련되는 시스템 요소들로 이루어진 컴포넌트 후보들을 파악한다. 계속해서, 여러 컴포넌트 후보들 간에 공유되는 공유 요소들을 파악하고 이들을 바탕으로 컴포넌트 후보들을 조정하고 최종적으로 컴포넌트를 확정한다. 또한 추출된 비즈니스 컴포넌트를 J2EE/EJB 환경으로 배치하기 위하여 그것을 사용자 인터페이스 컴포넌트, 비즈니스 로직 컴포넌트, 데이타 처리 컴포넌트 등과 같이 세 개의 작은 규모의 컴포넌트들로 분할한다. 이 논문에서 제안한 방법을 적용하면 레거시 시스템에 내재되어 있는 비즈니스 프로세스들은 관리하기 쉬운 크기의 컴포넌트들로 자연스럽게 나뉘어져, 다중계층(multi-tier) 클라이언트 서버 환경 또는 웹기반 환경에 쉽게 배포될 수 있을 것이다. 마지막으로, 컴포넌트가 제공하는 기능을 바탕으로 컴포넌트의 인터페이스를 정의한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 연구에 대해 간략하게 살펴보고, 3장에서 레거시 시스템으로부터 비즈니스 컴포넌트를 추출하기 위한 과정을 살펴본다. 4장에서는 본 논문에서 제안한 방법을 실제 시스템에 적용시켜 본 사례 연구에 대해 기술하고, 5장에서 결론을 맺는다.

2. 기존의 컴포넌트 마이닝 방법들

레거시 시스템으로부터 컴포넌트를 추출하기 위해 지금까지 연구되고 개발된 방법은 크게 세 가지 유형으로 분류할 수 있다.

첫 번째로 서브시스템을 하나의 컴포넌트로 인식하는 방안이다[8]. 서브시스템을 찾는 방법은 먼저, 시스템의 데이타베이스 뷰(view)를 만드는 것이다. 즉, 데이타베이스들과 프로그램들과의 연관 관계를 표현한다. 다음으로, 두 프로그램이 공통의 파일 집합을 사용하거나, 두 파일이 공통의 프로그램 집합에 의해 사용될 때의 관계를 표현한다. 마지막은 두 번째 과정에서 파악된 프로그램들과 데이타베이스들과의 연관관계를 바탕으로 관련된 시스템 구성 요소들을 클러스터링 하는 과정이다. 두 개의 원소를 포함한 클러스터부터 시작한다. 이 클러스터를 연관 관계를 바탕으로 보다 큰 클러스터를 구성하는 방식으로 병합한다. 이런 반복적인 과정은 클러스터들이 비유사성을 가질 때까지 계속된다. 여기서 클러스터는 서브시스템을 나타낸다. 이러한 반복적인 방법을 적용하여 시스템은 계층적 구조의 서브시스템들로 표현될 수 있는데, 이러한 각각의 서브시스템들은 시스템을 웹 기반의 분산 환경으로 이주시킬 때, 하나의 컴포넌트로 사상(mapping)된다. 이 방법은 시스템의 종속성(dependency) 정보만을 사용하여 컴포넌트를 인식하는 방법이다.

두 번째 방법은 특성(features)을 기반으로 한 컴포넌트 추출 방법이다. 이 방법은 한국전자통신연구원[9]이나 MineIT[3]에서 적용하는 기법으로서 특정 기능과 관련된 시스템 구성 요소들을 클러스터링 하여 하나의 컴포넌트로 인식하는 방법이다. 즉, 어떤 특성과 관련된 기능을 수행하는 컴포넌트를 추출하기 위하여 재공학자는 그 특성과 관련된 특성 평가 요소에 높은 가중치를 주고 그것을 바탕으로 시스템 구성 요소들을 검색한다. 예를 들어, 재공학자는 제어 흐름, 수학 연산, 입/출력, 조건절 등과 같은 다양한 코드 평가 파라미터의 민감도를 조정할 수 있다. 검색 후 찾아진 시스템 구성 요소를 기준 요소로 설정하고 그 요소에 영향을 주거나 혹은 그 요소로부터 영향을 받는 시스템 요소들을 시스템 종속성 정보를 적용하여 찾은 다음 그들을 클러스터링 하여 하나의 컴포넌트로 사상하는 방법을 사용한다. 이 방법을 적용할 경우, 레거시 시스템에서 재사용 가능한 컴포넌트를 쉽게 찾을 수 있다는 이점은 있지만 레거시 시스템 전체를 재사용하기 위한 측면을 고려하지 않는 문제점이 있다. 또한 계산이 많은 컴포넌트, 또는 입출력 중심의 컴포넌트 등과 같이 어떤 특성을 반영한 컴포넌트는 쉽게 찾을 수 있지만 그것이 어떤 하나의 목적을 지닌 의미 있는 컴포넌트라고 하기는 어렵다. 의미 있는 컴포넌트를 찾기 위해서는 보다 많은 재공학자의 개입을 필요로 한다.

세 번째 방법은 시스템 내에서 작업 흐름을 나타내는 트랜잭션을 기반으로 컴포넌트를 인식하고 추출하는 방안이다[10,11]. 트랜잭션은 하나의 서비스를 수행하기 위한 일련의 작업 활동을 의미한다. 사용자가 시스템에 어떤 서비스를 요청하면 그 서비스를 수행하기 위하여 시스템의 여러 구성 요소들이 정해진 순서에 의해 차례로 각자의 기능을 수행하게 된다. 시스템 구성 요소들이 정해진 순서에 의해 수행되는 것을 작업 흐름이라 하며 하나의 작업 흐름에 관여하는 시스템 구성 요소들을 클러스터링 하여 하나의 컴포넌트로 사상하는 방법이 여기에 해당한다. 이 방법은 컴포넌트를 추출하기 위해 시스템이 제공하는 서비스 측면을 주로 고려한 방법이라 할 수 있다. 이 유형을 따르는 기존의 방법들은 하나의 트랜잭션을 수행하기 위한 작업 흐름들을 하나의 컴포넌트로 구성하기 때문에 세밀한 컴포넌트를 추출할 수 있지만 그 크기가 너무 작아 분산 환경에서 의미 있는 컴포넌트가 되기 위해서는 추가적인 노력이 필요하다.

우리의 방법은 시스템의 유스케이스 관점에서 컴포넌트를 인식하는 방법이다. 우리의 방법은 위의 세 가지 유형 중 세 번째 유형과 매우 유사한 방법이다. 그런데 우리는 작업 흐름을 유스케이스로 인식하고 있다. 어떤 목적을 위해 수행되는 작업의 순서를 시스템이 사용자/액터에게 제공하는 하나의 서비스 즉, 유스케이스로 인식한다. 세 번째 방법과의 차이는 하나의 작업 흐름으로 하나의 컴포넌트를 구성하지 않고, 공통의 목적을 갖는 여러 작업 흐름들을 분석하여 하나의 컴포넌트를 구성한다는 것이다. 우리의 방법은 앞의 두 가지 유형에 비해 컴포넌트를 추출하기가 다소 복잡한 측면은 있으나 레거시 시스템으로부터 의미 있고 독립적인 기능을 수행하는 컴포넌트를 추출한다는 장점이 있다. 추출된 컴포넌트가 시스템이 사용자에게 제공하는 서비스를 독립적으로 표현하고 있으므로 웹 기반의 분산 환경에 적용할 경우 각각의 서비스를 독립적으로 접근할 수 있으므로 시스템의 병행성을 높일 수 있어서 시스템의 활용도를 더욱 높여 줄 수 있다는 장점이 있다. 또한, 두 번째 방법과는 달리 시스템 전체를 여러 컴포넌트들로 분할함으로써 시스템 전체 차원에서의 재사용성을 높일 수 있다는 이점도 갖는다.

3. 컴포넌트 추출 과정

전형적인 레거시 시스템은 수백 내지 수천 개의 프로그램들로 구성된다. 이들 프로그램들은 다양한 비즈니스 프로세스 흐름의 연결에 묶여 함께 동작한다. 레거시 시스템으로부터 컴포넌트를 추출하고, 이렇게 추출된 컴포넌트를 기반으로 시스템을 재구성하기 위해서는 이들

프로그램들을 독립된 하나의 비즈니스 기능(서비스)을 제공하는 프로그램들의 집합으로 분리해야 한다. 컴포넌트 파악 과정은 레거시 시스템에 실현되어 있는 비즈니스 기능들의 영역을 파악하여 독립된 하나의 비즈니스 기능을 수행하는 프로그램들의 집합을 컴포넌트 후보로 인식하는 과정이다. 이 장에서는 레거시 시스템에 내재되어 있는 비즈니스 컴포넌트들을 추출하기 위한 세부 과정에 대해 기술한다. 컴포넌트 추출 과정은 다음과 같이 네 개의 주요 작업으로 나눠진다.

단계 1: 유스케이스 파악

레거시 시스템으로부터 비즈니스 기능들의 영역을 파악하기 위해서 유스케이스를 파악한다. 유스케이스는 본질적으로 기능적인 관점을 나타낸다. 우리의 접근 방법은 기능을 실현하는 비즈니스 프로세스(또는 작업 흐름)를 추출함으로써 비즈니스 기능을 파악한다는 개념이다. 일반적으로 하나의 비즈니스 프로세스는 여러 유스케이스들이 결집되어 형성된다. 유스케이스를 파악하기 위한 세부적인 단계는 다음과 같다.

(1) 중심 화면을 찾는다.

중심(또는 메인) 화면은 응용 시스템이 시작할 때 처음 열리는 화면을 말한다. 예를 들어, 웹 기반 응용 시스템의 경우 홈 윈도우가 이에 해당한다. 대부분의 상호작용 시스템의 경우 중심 화면에는 특정 기능을 수행하기 위해 선택할 수 있는 메뉴가 존재한다.

(2) 화면 흐름을 추적한다.

다음으로 메인 화면의 메뉴로부터 선택되어진 각 기능의 시작점을 나타내는 초기 화면을 찾는다. 이 화면에서 시작하여 연관된 화면들을 계속해서 추적한다. 이 과정에서 우리는 서브 화면도 역시 파악한다. 서브 화면은 자신을 내포하고 있는 화면의 동적인 내용 및 정보들을 포함하고 있다. 이 모든 화면들을 가지고 화면 흐름을 형성할 수 있다. 그렇지만 때때로 어떤 기능은 순차적인 화면 흐름보다는 단 하나의 화면을 통해 기능을 수행하기도 한다.

(3) 화면 항해 경로를 정의한다.

어떤 기능의 수행과 관련된 화면들이 수집되면 연관된 화면들 간의 관계를 통해 화면 항해 경로를 정의할 수 있다. 화면 항해 경로는 화면들 간의 유효한 전이를 정의하며, 화면들과 화면들 간의 전이를 유발하는 사용자 행동들로 표현된다. 그것은 각 기능의 수행에 대해 하나의 성공 경로와 여러 개의 대체 경로들을 포함한다. 물론 오류 상황에 대한 경로들도 역시 포함한다.

(4) 화면 항해 경로를 유스케이스로 대응시킨다.

어떤 기능과 관련된 하나의 화면 항해 경로를 하나의 유스케이스에 대응시킬 수 있다. 유스케이스는 하나의 성공 시나리오, 몇 개의 대체 시나리오, 또한 몇 개의 예외적인 시나리오 등과 같이 여러 개의 시나리오들로 구성되고, 이들 시나리오들은 순차적인 행동들로 표현되기 때문이다. 사용자들에 의해 수행되는 순차적인 행동들은 화면 항해 경로에서 화면들 간의 전이를 유발하는 사용자 행동에 대응될 수 있다. 따라서 화면 항해 경로는 대응되는 비즈니스 기능을 수행하는 유스케이스로 인식된다.

예를 들어 현금자동지급기(ATM: Automated Teller Machine) 시스템의 기능들을 살펴보자. ATM 시스템에는 메인 화면으로부터 선택될 수 있는 다양한 서비스가 존재하고 이들 서비스와 연관된 많은 화면들이 존재한다. 결과적으로 ATM 시스템에 대한 화면 항해 경로들은 그림 1과 같다. 이 그림을 통해 우리는 CashInquiry, CashWithdrawal, CreditWithdrawal 등과 같은 6 개의 유스케이스들을 파악할 수 있다. CreditWithdrawal는 비록 계좌에 잔고가 없더라도 그 계좌로부터 돈을 인출할 수 있는 서비스이고 CreditTransfer는 잔고가 없는 계좌로부터 타 계좌로 돈을 이체할 수 있는 서비스이다. 다시 말해 이 기능들은 고객의 신용을 기반으로 한 일종의 대부 서비스이다.

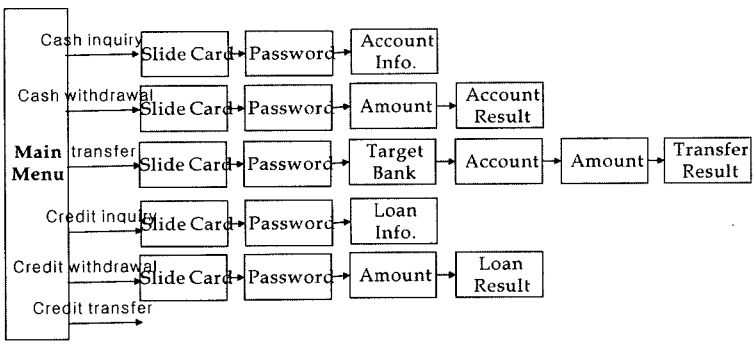


그림 1 ATM 시스템에서의 화면 항해 경로

컴포넌트 도출 기법을 보다 명확하게 설명하기 위하여 도출된 유스케이스 집합을 뜻하는 UC를 정의하도록 한다.

정의 1. UC는 레거시 코드로부터 도출된 유스케이스 집합이다.

예를 들어, ATM 시스템의 경우에는 UC = { CashInquiry, CashWithdrawal, Transfer, CreditInquiry, CreditWithdrawal, CreditTransfer }이 된다.

단계 2: 설계 모델 파악

단계 1에 따라서 레거시 코드로부터 화면 흐름을 분석하여 도출된 각 유스케이스에 대해서 설계 모델을 구축한다. 설계 모델은 유스케이스의 기능을 실현(realization)하기 위하여 필요한 시스템의 요소와 그들 간의 관계를 보여 준다.

정의 2. 시스템 요소(system element)는 레거시 시스템을 구성하는 기본 요소로서 Map Set 요소, 실체 요소, 사용자 인터페이스 요소, 데이타 접근 요소, 비즈니스 로직 요소로 분류된다.

- ME는 Map Set 요소의 집합이다. Map set은 프로그램에서 사용되는 화면을 정의하는 Map들의 집합을 뜻한다.
- EE는 실체 요소(Entity elements)의 집합으로서 영속적인 정보를 나타내며, 파일 또는 데이타베이스로 구현된다. 레거시 코드에서 각 파일 및 데이타베이스의 테이블은 하나의 실체 요소로 분석된다.
- UE는 사용자 인터페이스 요소(User interface elements)의 집합이다. 사용자 인터페이스 요소는 Map Set을 조작함으로써 사용자 화면을 생성하거나 사용자가 입력한 내용을 조회하는 서브프로그램(Paragraph)를 말한다. 예를 들어, CICS 프로그램에서 SEND MAP 명령어 또는 RECEIVE MAP 명령어를 가진 서브프로그램은 사용자 인터페이스 요소로 간주된다.
- DE는 데이타 접근 요소(Data access elements)의 집합이다. 데이타 접근 요소는 실체 요소를 조작하는 서브프로그램을 뜻한다. 예를 들어, READ 명령어, WRITE 명령어, EXEC SQL 등의 명령어를 이용해서 파일 또는 데이타베이스를 조작하는 서브프로그램은 데이타 접근 요소에 해당된다.
- BE는 비즈니스 로직 요소(Business logic elements)의 집합이다. 비즈니스 로직 요소는 시스템의 비즈니스 로직을 제공하기 위하여 다양한 연산과 계산을 포함하는 서브프로그램을 뜻한다. 예를 들어, "월 이자 계산", "월별 입금액 변화율 계산" 등은 비즈니스 로직 요소에 해당된다.

본 논문에서는 각 유스케이스별로 위와 같은 유형의 시스템 요소를 파악한다. 즉, 각 유스케이스에 대하여 유스케이스의 기능을 제공하는 데 관련된 Map Set 요소, 실체 요소, 사용자 인터페이스 요소, 데이타 접근 요소, 비즈니스 로직 요소를 도출한다. 설계 모델은 이들 시스템 요소들과 그들 간의 관계를 표현한 것으로서 레거시 코드로부터 컴포넌트를 도출하는 기반으로 사용된다.

정의 3. 유스케이스 uc(uc ∈UC)에 대한 설계 모델은 DMuc 로 표현되며, 유스케이스 uc의 실현과 관련된 시스템 요소(se) 간의 관계로 정의된다. DM_{uc}은 무방향 그래프 G = (N, E)로서 다음과 같이 정의된다.

- $N = ME_{uc} \cup EE_{uc} \cup UE_{uc} \cup DE_{uc} \cup BE_{uc}$. ME_{uc}, EE_{uc}, UE_{uc}, DE_{uc}, BE_{uc}는 각각 유스케이스 uc의 실현에 관련된 Map Set 요소의 집합, 실체 요소의 집합, 사용자 인터페이스 요소의 집합, 데이타 접근 요소의 집합, 비즈니스 로직 요소의 집합을 뜻한다.
- $E = \{ (se_i, se_j) \mid$ 시스템 요소 se_i가 se_j를 사용하거나 se_j가 se_i를 사용한다.$\}$

시스템 요소 간의 "사용"은 관련된 시스템 요소의 유형에 따라서 달라진다. 사용자 인터페이스 요소와 Map Set 요소인 경우에는 "사용"은 사용자 인터페이스 요소가 Map Set을 이용해서 사용자 화면을 조작하는 것을 뜻한다. 그리고 데이타 접근 요소와 실체 요소 간의 "사용"은 실체 요소가 나타내는 파일 또는 데이타베이스 테이블을 데이타 접근 요소가 접근하는 것을 뜻한다.

그림 2는 CashInquiry 유스케이스와 CreditInquiry 유스케이스에 대한 설계 모델을 보여 준다. 그림에서 사각형은 해당 유스케이스와 관련된 시스템 요소를 나타내며 시스템 요소의 유형은 << >>로 표현되었다.

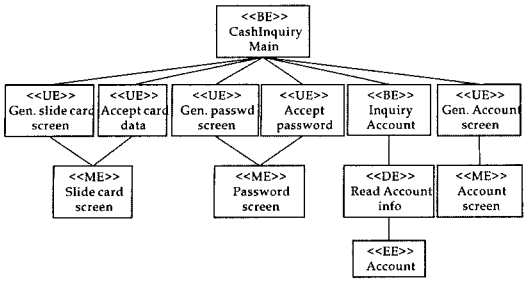
단계 3: 컴포넌트 구축

이전 단계에서 파악된 유스케이스별 설계 모델을 바탕으로 유사한 비즈니스 기능을 제공하는 관련된 유스케이스의 시스템 요소들을 통합함으로써 컴포넌트는 구축된다. 컴포넌트를 구축하는 세부 절차는 다음과 같이 요약된다.

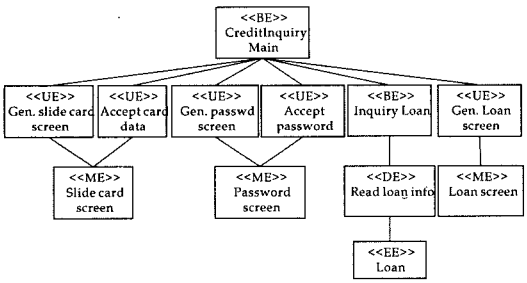
1. 후보 컴포넌트 파악: 관련된 유스케이스의 시스템 요소들을 클러스터링 함으로써 후보 컴포넌트를 파악한다.
2. 공유 요소 관리: 두 개 이상의 후보 컴포넌트에서 공통적으로 존재하는 공유 요소를 파악하고 공유 요소의 특성에 따라서 개별 컴포넌트화, 분할, 복제를 수행한다.
3. 컴포넌트 세분화: 파악된 컴포넌트를 J2EE/EJB와 같은 다중 계층의 분산 환경에 적합하도록 보다 세분화된 컴포넌트를 도출한다.

단계 3.1: 후보 컴포넌트 파악

실체 요소를 기준으로 관련된 유스케이스의 시스템



(a) CashInquiry



(b) CreditInquiry

그림 2 CashInquiry 유스케이스와 CreditInquiry 유스케이스의 설계 모델

요소를 클러스터링 함으로써 후보 컴포넌트를 도출한다. 기존의 대부분의 레거시 시스템은 비즈니스 서비스를 제공하는 데 초점을 두고 있다. 그리고 비즈니스 서비스에서 가장 중요한 요소는 비즈니스 서비스를 통하여 관리되는 영속적인 정보 즉 실체 요소이다. 따라서 각 유스케이스에서 사용되는 실체 요소를 기준으로 유사한 비즈니스 서비스를 제공하는 유스케이스인 지를 판단할 수 있다. 그러므로 본 논문에서는 실체 요소를 기준으로 관련된 유스케이스를 판단한다. 즉, 동일한 실체 요소를 공유하는 유스케이스는 관련된 비즈니스 서비스를 제공하는 것으로 간주하여 이들 유스케이스를 구성하는 시스템 요소들을 하나의 컴포넌트에 포함시킨다. 설계 모델을 바탕으로 후보 컴포넌트를 도출하는 자세한 절차는 다음과 같다.

1. 각 유스케이스별로 주요 실체 요소(primary entity elements)를 파악한다. 실체 요소 중에서 도메인의 실제 정보를 나타내는 실체 요소를 주요 실체 요소라고 정의한다. 예를 들어, 그림 3은 ATM 시스템에서 파악된 실체 요소를 보여 준다.

이 그림의 5개의 실체 요소 중에서 Account, Loan, Customer 실체 요소는 입/출금 도메인의 실제 정보로서 계좌, 대출, 고객 정보를 각각 나타내므로 주요 실체 요소라고 볼 수 있다. 반면에, Account-Customer 실체 요소와 Loan-Customer 실체 요소는 다른 두 실체 요소 간의 관계를 나타내는 실체 요소이므로 주요 실체 요소로 판단되지 않는다. 본 논문에서는 PEE_{uc}를 유스케이스 uc를 구성하는 주요 실체 요소라고 정의한다. 예를 들어, $PEE_{CashInquiry}$ = $PEE_{CashWithdrawal}$ = $PEE_{Transfer}$ = {Account, Customer}, $PEE_{CreditInquiry}$ = $PEE_{CreditWithdrawal}$ = {Loan, Customer}, $PEE_{CreditTransfer}$ = {Account, Loan, Customer}이다.

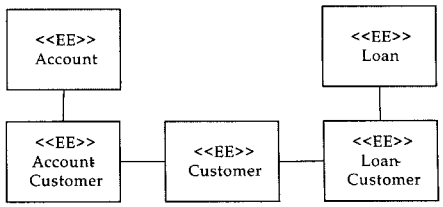


그림 3 ATM 시스템의 실체 요소

2. 우선 각 유스케이스별로 각각 컴포넌트를 정의한다. SC를 유스케이스 집합 UC로부터 도출된 컴포넌트의 집합이라고 하고 C_{uc}를 유스케이스 uc에 대해서 도출된 컴포넌트라고 하면, SC = { C_{uc} | uc ∈ UC } 이고 C_{uc} = { se | se ∈ $N(DM_{uc})$ }이다. 여기서 $N(DM_{uc})$는 유스케이스 uc에 대한 설계 모델을 구성하는 시스템 요소를 뜻한다.
3. 주요 실체 요소를 동일하게 공유하는 컴포넌트를 병합한다. 즉, 이 과정은 다음과 같이 정의될 수 있다. 만일 PEE_{uci} = PEE_{ucj} 이면 => SC = SC - {C_{uci}, C_{ucj}} ∪ {C_{uci_ucj}}, C_{uci_ucj} = C_{uci} ∪ C_{ucj}.

설명된 절차를 따르면 ATM 시스템에 대해서는 세 개의 컴포넌트가 도출된다. CashInquiry, CashWithdrawal, CashTransfer 유스케이스는 모두 {Account, Customer}를 주요 실체 요소로서 가지므로, 이 세 개의 유스케이스를 클러스터링하여 CashService라는 컴포넌트를 도출할 수 있다. 그리고 CreditInquiry 유스케이스와 CreditWithdrawal 유스케이스의 주요 실체 요소는 {Loan, Customer}이므로 CreditService라는 컴포넌트에 해당될 수 있다. 마지막으로 CreditTransfer 유스케이스의 주요 실체 요소는 {Account, Loan, Customer}이므로 CreditCashService라는 별도의 컴포넌트를 구성한다.

단계 3.2: 공유 요소 관리

단계 3.1을 통하여 구축된 컴포넌트에는 공통적으로 포함된 시스템 요소가 존재할 수 있다. 동일한 요소의 중복을 지양하고 시스템 요소의 재사용성을 높이기 위하여 여러 컴포넌트에서 공유되는 시스템 요소에 대한 평가와 적절한 관리가 필요하다.

공유 요소를 파악하는 작업은 높은 재사용성을 지닌 시스템 요소들을 찾기 위해 수행된다. 기본적으로 재사용성이 높은 컴포넌트는 하나의 시스템 내에서도 여러

기능들 간에 많이 공유되기 때문이다. 이런 컴포넌트들은 에러 보고서작성이나 트랜잭션 로그 기록 또는 날짜 계산 루틴 같은 공통 시스템 기능을 수행한다. 이런 유틸리티 컴포넌트들은 보통 비즈니스 컴포넌트보다 낮은 추상화 수준에서 동작한다. 처리의 중복을 피하고 시스템 행동의 일관성(consistency)을 보장하기 위해 이런 유틸리티 컴포넌트들은 비즈니스 컴포넌트들로부터 분리되어 시스템 전체에 재사용 가능한 유틸리티 라이브러리로 표준화 될 필요가 있다.

정의 4. CDM_{1_2}는 두 설계 모델 DM_1과 DM_2의 공통 설계 모델로서 무방향 그래프 G = (N, E)로서 다음과 같이 정의된다.

- $N = N(DM_1) \cap N(DM_2)$
- $E = \{(se_i, se_j) \mid (se_i, se_j) \in E(DM_1) \cap E(DM_2)\}$. $E(DM_i)$는 설계 모델 DM_i를 구성하는 에지의 집합이다.

예를 들어, ATM 시스템에서 $CDM_{CashInquiry_CreditInquiry}$는 설계 모델 $DM_{CashInquiry}$(그림 2(a))와 설계 모델 $DM_{CreditInquiry}$(그림 2(b))의 공통 설계 모델이다. 그리고 공통 설계 모델을 구성하는 설계 요소는 $N(CDM_{CashInquiry\_CreditInquiry})$ = {Gen.SlideCardScreen, AcceptCardData, SlideCardScreen, Gen.PasswordScreen, AcceptPassword, PasswordScreen}이다. 이런 공유 요소들은 Cardvalidation과 PasswordValidation 등과 같은 사용자 인증(User Authentication) 작업과 관련된 요소들이다. 따라서 UserAuthentication이라는 공유 컴포넌트를 만들 수 있고, 이 컴포넌트는 이런 공유 요소들로 구성된다.

파악된 공유 요소에 대해서는 평가를 통하여 그 공유 요소가 별도의 독립된 컴포넌트로 구성될 수 있는가의 여부를 판단한다. 공유 요소의 평가를 통해 첫째, 공유 요소를 독립된 별도의 컴포넌트로 도출 하거나 둘째, 공유 요소를 분할하거나 셋째, 공유 요소를 복제하는 등의 세 가지 가능한 결정 중의 하나를 내릴 수 있다. 이런 결정을 내리기 위해 적용할 수 있는 평가 기준으로 얼마나 높은 fan-in 값을 갖는지, 얼마나 낮은 fan-out 값을 갖는지, 얼마나 높은 응집도를 갖는 지와 얼마나 낮은 결합도를 갖는 지를 기준으로 사용할 수 있다. 아울러 기능에 대한 평가 기준으로 그 요소가 어떤 특정 기능에 대해 높은 점수를 갖는가를 평가한다. 그 요소가 어떤 특정 기능에 대해 높은 점수를 갖는다면 그것은 별도의 독립된 컴포넌트로 구성될 가능성이 있을 것이다. 이 기능에 의한 평가 점수에 따라 컴포넌트를 유사한 컴포넌트들의 그룹으로 분류할 수도 있다.

공유 요소가 독립성과 재사용성 측면에서 높은 점수를 획득하였을 경우에는 공유 요소를 컴포넌트화 한다. 즉, 공유 요소로 구성된 공유 컴포넌트를 독립적으로 캡슐화 한다. 공유 컴포넌트를 추출하고 나면, 단계 3.1에서 파악된 각각의 컴포넌트 후보들을 컴포넌트로 구성한다. 이 때, 각 컴포넌트 후보들의 구성 요소에서 공유 컴포넌트의 구성 요소들을 제외한다. 예를 들어, ATM 예제에서 UserAuthentication 공유 컴포넌트가 높은 점수를 획득하였다면 그것을 독립된 컴포넌트로 구성한다. 다음으로, 두 컴포넌트 CashService와 CreditService가 UserAuthentication 이라는 공유 컴포넌트에 포함되는 공유 요소를 제외하고 나머지 관련된 구성 요소들을 가지고 각각의 컴포넌트로 구성된다.

평가 결과 공유 요소를 분할하여야 할 경우에는 먼저 공유 요소를 분할한다. 공유 요소를 분할하게 되는 경우는, 공유 요소가 명백하게 몇 개의 독립적인 제어 쓰레드를 가짐으로 인해 공유 요소에 대한 평가 결과가 높은 점수를 획득하지 못하였을 경우에 해당한다. 이 경우에 공유 요소는 독립적인 제어 쓰레드에 따라 분리된다. 분할의 경우 간단하게는 서브프로그램 단위로 분할될 수 있고, 좀 더 복잡한 경우에는 하나의 서브프로그램을 프로그램 슬라이싱 기법을 적용하여 보다 작은 단위의 서브프로그램(슬라이스)으로 분할하여야 할 경우도 있다. 공유 요소를 분할하고 나면, 단계 3.1에서 파악된 각각의 컴포넌트 후보들에서 공유 요소들을 제외한 요소들을 가지고 컴포넌트로 구성한다. 계속해서, 분할된 공유 요소들은 적절하게 각각의 컴포넌트들에 편입시킨다.

평가 결과 공유 요소를 복제하여야 할 경우는, 공유 요소가 매우 낮은 평가 점수를 획득한 경우이다. 즉, 공유 요소가 응집도도 낮으면서 다른 컴포넌트의 요소와 매우 높은 결합도를 가진다. 이 경우에는 기존에 파악된 컴포넌트 후보들을 컴포넌트로 구성하는 과정에서 필요한 공유 요소들을 복제한다. 즉, 이 과정에서 공유 요소들은 그 요소를 공유하는 원래의 컴포넌트 후보들 모두의 구성 요소가 될 수 있도록 각각 복제된다.

단계 3.3: 컴포넌트 세분화

이 단계에서는 추출된 비즈니스 컴포넌트를 궁극적으로 J2EE/EJB 환경과 같은 다중 계층 분산 환경으로 배포하기 위하여 세 개의 보다 작은 규모의 컴포넌트들로 분할한다. 첫 번째 부분은 시스템 요소의 타입이 UE, ME인 시스템 요소들로 구성되는 사용자 인터페이스 컴포넌트이다. 궁극적으로 사용자 인터페이스 컴포넌트로부터 JSP나 Servlet 등이 유도된다. 두 번째는 시스템 요소의 타입이 BE인 시스템 요소들로 구성되는 비즈니스 로직 컴포넌트이다. 그것은 세션 타입의 EJB로 대응될 수 있다. 마지막은 시스템 요소의 타입이 DE, EE인 시스템 요소들로 구성되는 데이타 처리 컴포넌트이다. 이것은 엔티티 타입의 EJB에 대응된다.

단계 4: 컴포넌트 인터페이스 정의

인터페이스는 컴포넌트에 의해서 구현된 기능을 외부에서 접근할 수 있는 방법을 제공한다. 인터페이스를 통하여 접근되는 컴포넌트의 기능은 비즈니스 로직과 데이타 접근 로직이다. 예를 들어, J2EE 환경에서 세션 타입의 EJB와 엔티티 타입의 EJB의 리모트 인터페이스가 컴포넌트 인터페이스에 해당된다. 따라서 논문에서 제시한 컴포넌트 도출 기법에 따라서 정의된 각 컴포넌트를 구성하는 비즈니스 로직과 데이타 접근 로직에 대응되는 인터페이스를 정의하면 된다. 구체적으로 말하면, 컴포넌트에 포함된 비즈니스 로직 요소(BE)와 데이타 접근 로직 요소(DE)가 제공하는 기능을 포함하도록 컴포넌트의 인터페이스를 결정한다. 예를 들어, 그림 4는 ATM 시스템을 대상으로 컴포넌트 세분화를 수행하기 이전의 컴포넌트에 대해서 도출된 인터페이스를 보여주는 컴포넌트 다이어그램이다. 여기서 CashService라는 비즈니스 컴포넌트에 대해, *queryAccount( )*와 *withdrawCash( ), transferMoney( )*라는 세 개의 오퍼레이션을 갖는 ICashService라는 이름의 인터페이스를 정의할 수 있다.

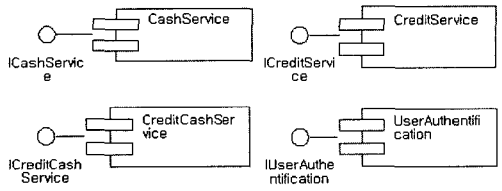


그림 4 ATM 시스템으로부터 도출된 컴포넌트 및 인터페이스

4. 사례 연구

이 논문에서 제안한 우리의 방법은 레거시 시스템을 현대화하기 위한 프로세스를 개발하는 것을 목표로 진행되는 프로젝트의 일부분으로 실현되었다. MaRMI-RE (Magic and Robust Methodology Integrated-ReEngineering)로 명명된 이 프로젝트는 한국전자통신연구원과 국내의 몇몇 대학과 기업이 공동으로 참여하여 수행하였다.

우리는 논문에서 제안한 접근 방법을 적당한 크기의 파일럿 프로젝트에 적용하여 보았고, 지금 현재 훨씬 더 큰 규모의 실제 프로젝트에 적용하고 있는 중이다. 이 절에서는 파일럿 프로젝트에 적용한 사례 연구의 결과에 대하여 기술한다. 이 프로젝트의 목표는 국내의 한 방적 회사에서 운용 중인 레거시 시스템인 원가 계산 시스템(Cost Accounting System)을 현대화하는 것이다. 이 원가 계산 시스템은 IBM 메인프레임에서 동작 중이며, 대략 60,000 라인의 COBOL 코드로 작성되었으며 61개의 소스 코드 파일들에 나뉘어져 있다. 이 시스템은 충분한 도큐먼트를 갖고 있지 않는 단일체 구조 시스템이다.

그림 5는 이 시스템에 대한 개략적인 계층 구조를 보여준다. 이 시스템은 대부분의 레거시 시스템이 그러하듯이 전형적인 IPO(Input-Processing-Output) 구조로 구성되어 있다. 여기에는 '데이타 등록', '데이타 대체', '원가 계산', '데이타 및 결과 조회', '데이타 및 결과 발행' 등과 같은 5개의 서브시스템들이 존재한다. '데이타 등록' 서브시스템은 생산품의 가격을 결정하기 위해 요구되는 기본적인 데이타를 등록하는 7개의 프로그램들로 구성되고, '원가 계산' 서브시스템은 생산품의 가격 계산과 관련된 6개의 프로그램들로 구성된다. 서브시스템 '데이타 및 결과 조회'는 등록된 데이타나 계산 결과를 조회하기 위한 8개의 프로그램으로 구성된다. 이런 형태로 나머지 서브시스템들도 구성되며, 전체 61개의 소스 코드 파일 중 25개의 소스 코드 파일들이 서브프로그램들, 맵 세트들, 카피 북들의 형태로 존재한다.

우리의 방법은 레거시 시스템을 J2EE/EJB 환경에서 그들의 서비스를 제공하는 컴포넌트들의 집합으로 변환하는 것이다. 이 방법을 따라 그림 6에서 보는 바와 같은 14개의 비즈니스 컴포넌트들을 생성하였다. 이들은 계속해서 23개의 JSP와 14개의 세션 타입의 EJB, 10개

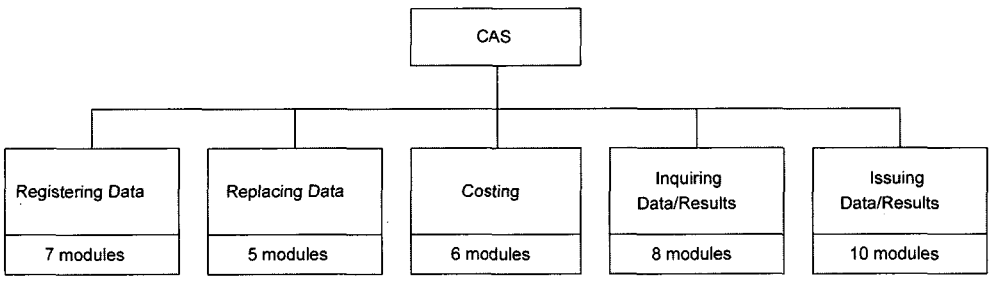


그림 5 원가계산 시스템의 계층적 구조

의 엔티티 타입의 EJB로 나뉘어졌다.

예를 들어, ByProductControl 컴포넌트의 도출 과정을 소개한다. 우선, CAS 시스템을 분석하여 화면 항해 경로를 구축하였다. 그림 7은 화면 항해 경로로서 ByProductControl 컴포넌트와 관련된 부분만을 보여준다. MainMenu -> ProductCost -> RegisterByproduct -> ReplaceByproduct 경로로부터 CostingByproduct 유스케이스를 도출하였고, MainMenu -> ProductCost -> InquiryByproduct 경로로부터 ByproductCostInquiry 유스케이스를 도출하였다.

두 번째로 도출된 각 유스케이스로부터 설계 모델을 구축하였다. 그림 8은 각 유스케이스 별 설계 모델을 보여 준다. 그림에서 볼 수 있듯이, 설계 모델은 각 유스케이스의 구현에 관련된 여러 유형의 설계 요소와 그들 간의 관계를 보여 주고 있다. 그리고 두 설계 모델은 Byproduct라는 주요 실체 요소를 공유하고 있다. 따라서 두 유스케이스의 설계 모델은 하나의 ByProductControl 컴포넌트로서 도출되었다. 그리고 ByProductControl 컴포넌트의 인터페이스로서 iByProductControl을 정의하였으며 이 인터페이스에는 ByProductControl 컴포넌트가 제공하는 두 개의 기능에 대응되는 computeBPCost()와 inquireBPCost()가 연산으로 정의되었다. ElectricPowerRatesControl, OilRatesControl, RawMaterialControl 등의 다른 컴포넌트들도 마찬가지 방식으로 구축되었다.

본 논문에서는 레거시 코드로부터 설계 모델을 구성할 때 그 역할에 따라서 UE, DE, BE, EE, ME 등으로 분류하고 있고, EE, ME 유형은 레거시 코드 상에서 명시적으로 구분이 되는 요소이지만, UE, DE, BE 유형은

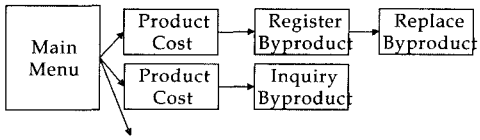
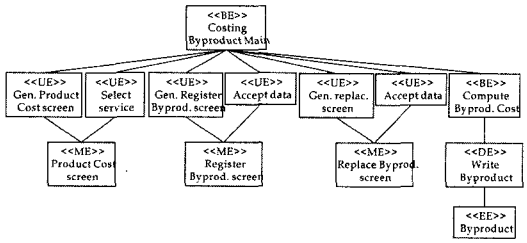
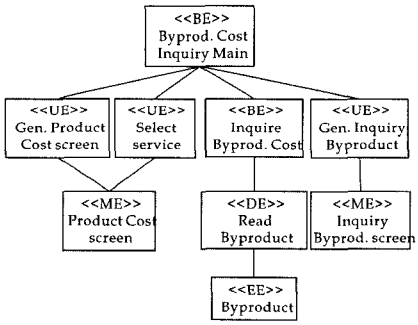


그림 7 ByProductControl 컴포넌트와 관련된 화면 항해 경로



(a) CostingByproduct의 설계 모델



(b) ByproductCostInquiry의 설계 모델

그림 8 CostingByproduct와 ByproductCostInquiry 유스케이스의 설계 모델

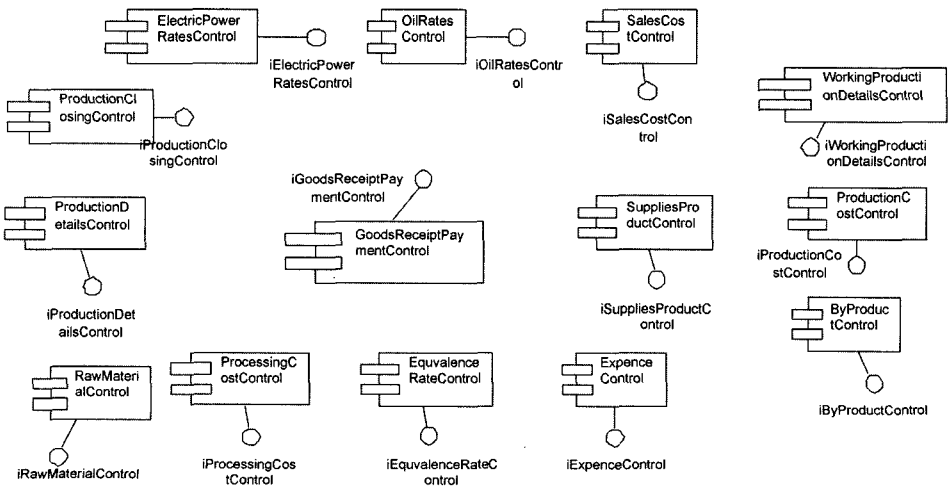


그림 6 원가계산 시스템으로부터 파악된 비즈니스 컴포넌트들

레거시 코드의 내용에 따라서 결정된다. 즉, 패러그래프(서브프로그램)가 실제로 제공하는 로직의 유형이 무엇인가에 의해서 UE, DE, BE로 결정이 된다. 본 논문에서는 패러그래프가 UE, DE, BE 유형 중에서 오직 하나에 해당되는 것을 가정하고 있다. 즉, 하나의 패러그래프가 사용자 인터페이스 로직(UE), 데이타 접근 로직(DE), 또는 비즈니스 로직(BE) 중의 하나를 제공하는 것을 가정하고 있다.

사례 연구를 수행하는 과정에서 레거시 COBOL 코드의 패러그래프가 비교적 세 가지 유형의 로직(UE, BE, DE)으로 잘 분할되어 있음을 확인하였다. 하지만, 유지보수가 체계적으로 수행되지 않을 경우에는 하나의 패러그래프에 2개 이상의 로직 유형이 혼재할 가능성도 있다. 이럴 경우에 본 논문에서 제시하는 컴포넌트 추출 방법의 효율성을 높이기 위해서는 설계 모델을 구성하기에 앞서 패러그래프의 로직을 분석하여 세 가지 유형의 로직을 분리해서 별도의 모듈로 나누는 재구조화 작업이 선행되어야 한다.

5. 결 론

이 논문에서 기술된 우리의 방법은 레거시 시스템의 현대화에 대한 필요성이 동기가 되었다. 레거시 시스템의 현대화는 COBOL 등의 언어로 작성되고 메인 프레임에서 동작하며 단일체 구조를 갖는 다양한 레거시 시스템을 J2EE/EJB 환경과 같이 새로운 분산 플랫폼으로 이주하는 과정을 말한다. 레거시 시스템을 현대화하기 위해서는 레거시 시스템에 묻혀 있는 비즈니스 컴포넌트를 추출하기 위한 작업이 선행되어야 한다. 이를 위해 본 연구에서는 레거시 시스템으로부터 어떤 목적을 위해 수행되는 작업의 흐름을 시스템이 사용자에게 제공하는 하나의 서비스 즉, 유스케이스로 인식하고, 이러한 유스케이스의 수행과 관련된 시스템 요소를 파악한 다음 이들 요소를 중심으로 컴포넌트를 추출하는 방법을 제시하였다. 이런 방법을 사용함으로써 레거시 시스템으로부터 의미 있고 독립적인 기능을 수행하는 컴포넌트를 추출할 수 있었다.

한편 본 논문에서 제시한 방법은 먼저 화면 흐름을 바탕으로 유스케이스를 도출하고 유스케이스와 관련된 레거시 코드들을 분석하는 방식으로 컴포넌트를 추출하고 있다. 따라서 레거시 시스템으로부터 충분한 화면 정보를 도출할 수 있어야 한다. 그러므로 제안한 방법은 배치(batch) 성격의 레거시 시스템 보다는 화면을 통해 사용자와 상호작용 하는 인터액티브(interactive) 성격의 레거시 시스템에 보다 적합한 방법이라 말할 수 있다.

현재 우리의 방법은 MaRMI-RE라는 이름의 레거시 현대화 프로세스의 일부분으로서 실현되었다[12]. 또한 현대화 프로세스를 지원하기 위해 프로그램 분석 도구, 설계 정보 파악 도구, 코드 변환 도구(COBOL2Java) 등과 같은 다양한 도구들이 개발되었다. 게다가 대규모 소프트웨어에 대한 우리 방법의 응용가능성을 확신하기 위하여 보다 많은 사례 연구들이 계획되고 있고 진행중이다. 우리는 이 방법이 다양한 형태의 레거시 응용 시스템으로부터 컴포넌트를 추출하기 위한 방법의 기초를 제공할 것이라고 확신한다.

참 고 문 헌

[1] L. Erlikh, "Leveraging legacy systems in modern architectures," *White Paper from Relativity Technologies*, 2001.

[2] M. Battaglia, G. Savoia and J. Favaro, "RENAISSANCE: A Method to Migrate from Legacy to Immortal Software Systems," *Proc. of CSMR'98*, pp.197-200, 1998.

[3] Intercomp, "MineIT: Automated extraction of business rules from legacy COBOL applications to Java applications and objects," *White Paper from Intercomp*, 2001.

[4] SEEC, "A next-generation architecture for financial services on the internet," *White Paper from SEEC*, 2000.

[5] Intercomp, "WebIT: Web-Enabling of Legacy Applications," *White Paper from Intercomp*, 2001.

[6] W. M. Ulrich, *Legacy Systems: Transformation Strategies*, Prentice Hall PTR, 2002.

[7] D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall PTR, 2001.

[8] M. A. Serrano, D. L. Carver, and C. M. Oca, "Reengineering legacy systems for distributed environments," *The Journal of Systems and Software*, **Vol. 64**, No. 1, pp.37-55, 2002.

[9] 송문섭, 박창순, "레거시 시스템 이해 도구를 이용한 비즈니스 로직 추출 기법", 한국정보과학회 제29회 추계학술발표회 논문집, 2002.

[10] 한무희, 김현수, 김철홍, "레거시 시스템으로부터 컴포넌트를 추출하기 위한 방법", 한국정보과학회 소프트웨어공학회지, 제16권, 제1호, pp.91-102, 2003.

[11] Micro Focus, "Componentization of legacy assets: A rapid, low-risk method of creating reusable components from legacy CICS applications," *White Paper from Micro Focus*, 2002.

[12] H. S. Kim and C. H. Kim, "A use-case driven approach to component mining for legacy modernization," *Proc. of the IASTED Int'l Conf. on Software Engineering*, pp.303-308, 2004.

김 현 수

1988년 서울대학교 계산통계학과(학사) 1991년 한국과학기술원 전산학과(석사) 1995년 한국과학기술원 전산학과(박사) 1995년~1995년 한국전자통신연구원 Post Doc. 1996년~2001년 금오공과대학교 컴퓨터공학부 조교수. 1999년~2000년 Colorado State Univ. 방문 연구교수. 2001년~현재 충남대학교 전기정보통신공학부 컴퓨터전공 부교수. 관심분야는 Software Engineering(Software Maintenance, Software Reengineering, Software Testing), Object Oriented Software Engineering, Component-based Software Engineering

채 흥 석

1994년 서울대 원자핵공학 학사. 1996년 한국과학기술원 전산학 석사. 2000년 한국과학기술원 전산학 박사. 2000년~2003년 ㈜ 동양시스템즈 기술연구소 선임연구원. 2003년~2004년 한국과학기술원 전산학과 초빙교수. 2004년~현재 부산대학교 컴퓨터 공학과 전임강사. 관심분야는 객체지향 방법론, 소프트웨어 테스팅, 소프트웨어 메트릭, 소프트웨어 유지보수

김 철 홍

충남대학교 문리과대에서 학사, 성균관대 대학원에서 정보공학 석사학위를 취득하고, 한국전자통신연구원 임베디드S/W연구단 S/W공학연구팀에 책임연구원으로 재직중이다. 현재 임베디드 S/W 재사용 프로세스 및 체계 연구를 수행중이다. 관심분야는 재공학, 재사용, 시스템 개발 방법론, 테스팅 등이다.