유즈 케이스를 적용한 시스템 기능 분해
(A Use Case Driven Approach to Systemetic Functional Decomposition)

윤 청 [†] 김 응 모 [††] 배 두 환 [†††]
(Cheong Youn) (Ung Mo Kim) (Doo Hwan Bae)

요 약 기능 분해는 복잡한 시스템을 이해하기 위해 광범위하게 사용되는 시스템 모델링 기술이다. 기능 분해는 문제 영역을 기능별로 분해하는 데 그 기반을 두고 있으며, 이는 시스템의 기능에 대한 식별을 전제로 한다. 일반적으로 시스템의 기능에 대한 식별은, 분석가에 의해 어떠한 조직적인 지침없이 비정형적으로 수행되는 것이 관례였다. 따라서 이러한 기법을 이용하면 시스템을 분할하거나 시스템의 기능을 올바르게 식별하기가 매우 어렵다. 본 논문은 이러한 기능 분석에 대해 use case을 이용한 기법을 제안하고자 한다. 본 기법의 장점은 크게 두 가지로 요약할 수 있다. 첫째, 시스템의 분할과 기능에 대한 식별이 전통적인 기법보다 더 용이하다. 둘째, 시스템의 요구사항과 구현이 사용자에 의해 쉽게 검증될 수 있다. 본 기법은 하향식으로 이루어져, 구조적 분석과 같이 보편화된 기능 분석 기법들과 자연스럽게 병합될 수 있다. 본 논문은 이를 위해 use case의 식별, 그리고 이를 이용한 기능 분해를 단계적 과정과 가이드라인을 통해 설명하고, 이를 특정 애플리케이션에 적용하여 그 유용성을 입증한다.

Abstract Functional decomposition is a widely used system modeling technique for understanding complex systems. It is based on functional partitioning of the problem domain and requires identification of a system's functionalities. The identification of a system's functionalities is performed informally by system analysts without any systematic guidelines. The approaches which employ this technique have, therefore, difficulties in partitioning of the system and identification of functionalities. This paper proposes a use case driven functional decomposition approach. This approach has two major advantages. Firstly, the identification of a system's functionalities is easier than the classical approaches. Secondly, the system requirements and implementation can be easily validated with the users of the system. It also matches well with common functional decomposition approaches like Structured Analysis as it is also a top-down approach. The approach is presented along with an example of its application. We also provide engineering steps, heuristics, and guidelines.

1. 서 론

시스템 모델링은 오랜 기간동안 여러 응용 분야에서 사용되어 왔다. 엔지니어와 기술자들은 시스템을 설계하기 이전 우선 전체 시스템에 대한 문제를 이해하기 위해 모델을 구축한다. 이러한 시스템 모델링은 소프트웨어 시스템을 체계적으로 구축하기 위해서도 요구된다. 개발자들은 복잡한 소프트웨어 시스템에 대한 서로 다른 관점을 추상화(abstraction)하기 위한 모델을 정립하고, 이를 요구 사항 검증을 위해 사용하며, 시스템의 구현을 위해 점진적으로 세부 사항을 추가해나간다.

기능 분해(functional decomposition)는 소프트웨어 시스템을 모델링하기 위해 광범위하게 사용되는 접근 기법이다. 설계자는 우선 시스템을 구성하는 기능들을 식별함으로서 그 시스템에 대한 모델링을 시작한다. 그리고 전체 시스템에 대한 이해를 단순화하기 위해 이러한 기능들을 세부 기능들로 분할(partition)하면서, 이들

† 종신회원 : 충남대학교 컴퓨터과학과 교수
cyoun@iclab.chungnam.ac.kr
†† 종신회원 : 성균관대학교 전기전자 및 컴퓨터공학부 교수
umkim@yurim.skku.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
bae@salmosa.kaist.ac.kr
논문접수 : 1998년 6월 1일
심사완료 : 1998년 10월 16일

서로간의 인터페이스를 정의한다. 이러한 문제 영역(problem space)에 대한 분할 접근 기법은 구조적 분석(structured analysis)과 같은 전형적인 기능 분해 기법의 중요한 단면을 나타낸다[1, 2].

기능 분해에 대한 초기의 연구는 기능 분해의 도구로서 자료흐름도(data flow diagram)를 사용하였던 구조적 분석에서 주로 이루어졌다[1, 2]. 자료흐름도는 시스템 설계상의 문제들을 기술하기 위한 표기 기법으로서 소프트웨어 공학 분야에서 최초로 사용되었다[3, 4, 5]. 전형적으로 시스템의 완성된 모델은 여러 계층의 자료흐름도들로 구성된다. 각 계층의 자료흐름도에서는, 이에 상응하는 상위 계층 자료흐름도에서 정의된 복잡한 기능에 대한 분해를 표현할 수 있다. 또 다른 시스템 모델링 기법으로서는 DSSD(Data Structured Systems Development)[11, 12, 13, 14], JSD(Jackson System Development)[15]와 같은 데이타 구조(data structure) 기반의 기법들과, OOADA(Object-Oriented Analysis and Design)[8]나 OMT(Object Modeling Technique)[9]와 같은 객체지향 기반의 기법들이 제안되었다. 객체지향 기법은 에플리케이션 도메인으로 부터 객체들을 식별하여 여기에다 연산(operation)들을 병합하는데 역점을 두는 반면, 구조적 분석 기법은 시스템의 기능을 분해하는데 역점을 둔다.

시스템을 어떻게 분할해 나갈 것인가의 문제는 시스템을 개발하는 대부분의 사람들에게 어렵게 느껴지는 문제이다. 기존의 분할에 사용할 수 있는 정보가 있는 경우 이를 활용할 수도 있다. 회사 업무를 위한 시스템을 만들 때 회사의 조직(인사부, 자재부 등)은 기능 중심으로 이미 분류되어 있는 경우가 많고 이를 활용하여 시스템을 분할해 나갈 수도 있다. 만약 기존의 정보가 없이 새로운 시스템을 만들어 나갈 경우 분할은 더욱 어려운 문제로 느껴진다.

본 논문에서는 유즈 케이스를 적용하여 기능 분해를 수행하는 기법을 제안하고자 한다. 유즈 케이스를 이용한 시스템 분석 기법은 Jacobson이 객체지향 분석에 처음 적용하여 많은 호응을 얻었고 현재 객체지향 분야에서 많이 활용되고 있다[7]. 유즈 케이스 기법의 장점은 시스템이 무엇보다도 사용자의 관점(view point of users)에서 이해된다는 것이다. 여기서 사용자 관점은 시스템과 사용자간의 상호교류에 기반을 둔 것이다.

이 논문에서 제안하고 있는 기법의 주요 장점은 유즈 케이스를 이용하면 (초보자라도) 시스템의 기능들을 분해하기가 쉽다는 것이다. 이와 같은 사실은 전형적인 구조적 분석에서는 시스템의 기능들을 식별하는 분명한 지침(guidelines)이 없다는 것과 대조된다. 본 기법은 또한 하향식(top down)으로서 구조적 분석과 같은 전형적인 기능 분해 기법과 자연스럽게 병합될 수 있다. 이 점은 매우 중요한 장점으로서, 최근에 제안된 사건 분할(event partition) 기법은 하향식도 상향식(bottom up)도 아닌 혼합식(mixed)으로 기능 분해 기법에 적용하기가 어렵다는 것을 들 수 있다[6]. 또 다른 장점은 유즈 케이스를 이용하여 설계된 시스템은 고객(customer)들에 의해 쉽게 검증된다는 것이다. 이는 일반적으로 고객들이 시스템에 대해 이해하려고 할 때, 시스템이 사용자에게 제공할 수 있는 기능(user visible functions)들이 우선 고려된다는 점이다.

이에 따라 본 논문에서는 유즈 케이스를 기반으로 기능을 분석하는 방법에 대한 지침(guidelines)과 과정을 단계별로 보이고자 한다. 이 지침에 따라 개발자는 시스템의 구성하는 전체적인 기능들을 쉽게 분해할 수 있다. 또한 제안한 지침을 자판기(vending machine) 시스템에 적용하여 그 유용성을 설명하고자 한다. 본 논문의 구성은 다음과 같다. 제 2 장은 제안하고 있는 기능 분석 방법의 제안 동기에 대하여 설명한다. 제 3 장은 전형적인 기능 분석 기법인 구조적 분석에 대하여 설명한다. 제 4 장은 유즈 케이스를 이용한 기능 분석을 위한 가이드라인과 지침을 단계별로 설명한다. 제 5 장은 제 4 장에서 제안한 지침을 자판기 시스템에 적용한 사례를 보이고 있다. 제 6 장에서는 제안한 기법과 다른 방법들과의 비교 분석을 통해 제안한 기법의 장점을 설명하고 결론을 맺는다.

2. 제안 동기

구조적 분석과 같은 전형적인 기능 분해 기법은 시스템을 여러 개의 부시스템(sub-system)들로 분할하기 쉬운 단순한 형태의 에플리케이션에 잘 적용된다. 그러나 대규모 에플리케이션, 기능에 대한 식별이나 이들 기능에 대한 분할이 명확하지 않은 시스템 등에서는 많은 문제점을 내포하고 있다. 이는 근본적으로 하향식(top-down) 접근방법에서 기인하는 것으로, 하부를 정확히 모르고 출발하는 것이 가장 큰 한계점으로 지적되고 있다. 그러므로 이 기법은 시스템에 대한 주 기능(major function)들이 우선적으로 식별되지 않고서는 대규모의 복잡한 에플리케이션에 적용하기에는 한계점을 가지고 있다.

이를 해결하기 위해 Yourdon은 사건 분할 기법(event partition scheme)을 제안하였다[6]. 이 기법에서는 사건(event)을 기반으로, 시스템의 동작(behavior)

을 표현하는 자료흐름도(data flow diagram)를 이용하여 시스템에 대한 모델링을 수행한다. 사건 분할 기법은 구조적 분석과 같은 전통적인 기능 분할 기법이 적용되지 않는 에플리케이션들의 경우 시스템이 반응하는 사건을 고려한 자료흐름도를 생성함으로서 분할의 문제를 해결하려 하고 있다. 그러나 이 기법의 문제점은 하나의 자료흐름도로 표현이 불가능할 만큼 많은 양의 버블(bubble)들이 생성되어, 자료흐름도의 대한 이해도가 복잡해진다는 것이다. 이를 해결하기 위한 방안으로, 버블들을 집단화(aggregation)하기 위해 상위(upward)와 하위(downward) 레벨링을 수행하여 상위 레벨의 자료흐름도로 전환하는 방안이 제안되었다[6]. 이 기법으로 자료흐름도의 복잡도를 단순화할 수는 있지만, 이 기법은 하향식도 상향식(bottom-up)도 아닌 혼합식(mixed) 기법의 특성을 가지고 있다.

본 논문이 제안하는 유즈 케이스 적용 기법에서는 시스템을 우선 블랙박스(black box)로 간주하여 시스템과 사용자간의 역할에 의한 상호교류를 식별한다. 이 상호교류들로 부터 시스템에 대한 유즈 케이스가 식별된다. 그런 다음 이 유즈 케이스를 기능으로 간주하여 시스템을 모델링한다. 여기서 하위 레벨들은 각 유즈 케이스의 기능을 세분화 해 나간다. 즉 시스템의 가장 상위 레벨(topmost level)에서 시작하여 점차적으로 하위 레벨로 모델링을 해 나간다. 이와 같이 기능 분석에서 유즈 케이스를 적용했을 때 다음과 같은 장점이 있다.

첫째, 유즈 케이스는 우선 시스템을 블랙박스로 바라보아 표현한다. 이는 시스템의 동작을 추상화(abstraction)함으로서 사용자로부터 시스템의 내부적인 모습을 감출 수 있다.

둘째, 유즈 케이스는 본질적으로 기능적(functional)인 특성과 가장 밀접하다고 인식되고 있다. 이는 유즈 케이스가 자연스럽게 기능 분할과 통합(integration)할 수 있음을 시사한다. 시스템 모델링에서 사용하는 대부분의 접근 기법들은 서로 다른 유형의 방법론(methodology)들을 통합하는데 어려움을 가지고 있다. 예를 들어 객체지향적인 해법을 얻기 위해 기능 요구사항들을 이용하는 것이 쉬운 일이 아니다. 이는 기능지향적 사고 기법에서 객체지향의 기법으로 전환이 있다는 것을 의미하며, 이는 관점 또는 논리의 비약으로 말미암아 소프트웨어 제품에 심각한 오류가 발생할 가능성이 있다는 것이다. Bearard[16]는 많은 에플리케이션을 개발해 본 경험으로부터 대부분의 조직체들이 기능지향의 전위기(front-end)를 객체지향 전위기와 통합하는데 많은 어려움이 있었다고 밝히고 있다.

셋째, 유즈 케이스는 사용자의 역할에 대한 고찰을 통하여 시스템의 요구사항(requirements)을 가장 자연스럽게 반영하므로 시스템의 기능을 식별하기가 매우 용이하다. 이에 대해 Jacobson은 "모든 유즈 케이스 설명서는 시스템에 대한 기능들을 정의할 수 있다" 고 언급하고 있다[7]. 현재 많은 소프트웨어 개발 업체들이 고급 수준의 유즈 케이스로 표현된 사용자 요구사항을 출발점으로 하여 에플리케이션을 개발하는 것은 보편적인 일이다.

3. 유즈 케이스의 적용

구조적 분석에서는 시스템의 기능들이 계층적으로 표현된다. 이 기법은 하향식 기법이므로 복잡한 각 기능을 부기능(subfunction)들로 분할한다. 각 부기능은 하위 레벨(low level)의 부기능들로 계속해서 분할되어진다. 이러한 과정은 더 이상의 분할이 필요없는 원시기능(primitive function)들에 도달할 때까지 반복해서 수행된다. 구조적 분석은 여러 개의 계층적으로 상호 연결된 자료흐름도들로 구성된다. 자료흐름도에서의 시스템의 기능들은 이러한 기능들을 수행하는 프로세스(process)들을 나타내는 버블(bubble)들에 의해 표현된다.

본 논문에서는 기능 분해를 위한 접근 기법의 방안으로 Jacobson이 제안한 유즈 케이스의 개념을 이용한다[7]. 유즈 케이스는 시스템의 행위자(actor)에 의해 수행되는 서로 밀접하게 관련있는 트랜잭션(transaction)들의 순서이다. 여기서 행위자는 사람이나 혹은 다른 어떤 시스템이 각기 맡게되는 고유의 역할(role)을 내포하고 있다. 예를 들면, 비행기 시스템에서의 행위자는 승객, 조종사, 정비사 등이 될 수 있다. Jacobson은 이러한 유즈 케이스의 개념을 Objectory라고 하는 객체지향 시스템 패러다임에 적용하였다. 유즈 케이스는 시스템과 상호 교류하는 외부 요소(external elements)들을 식별함으로서, 시스템의 기능들을 완전하게 기술할 수 있다. 그리고 Booch의 OOADA[8]와 Rumbaugh의 OMT[10] 등 다른 객체지향 방법들 역시 시스템을 구성하는 클래스, 객체, 객체들간의 관계, 동작 등을 식별하는데 유즈 케이스 개념을 이용한다. 이와는 달리 본 논문의 접근 기법은 유즈 케이스를 이용하여 구조적 분석에 기반을 둔 시스템의 기능 모델을 개발하는데 초점을 두고자 한다. 다음 절은 유즈 케이스 기법을 이용하여 기능 분해를 하기위해 필요로하는 지침과 과정들을 설명한다. 우선 유즈 케이스의 식별 방법에 관한 것이 다루어지고, 다음으로 자료흐름도에 기반을 둔 기능 분해 모델의 개

발에 관한 내용이 다루어진다.

3.1 유즈 케이스의 식별

유즈 케이스를 이용한 기능 분할에 적용될 수 있는 순서와 가이드라인은 다음과 같다.

● **유즈 케이스의 규명**

1. 문제기술 준비
2. 시스템을 사용하는 행위자(actor)가 누구인지 규명한다.
3. 각 행위자에 대하여 유즈 케이스를 찾아내고 목록을 만든다.
4. 각 유즈 케이스에 대하여 기본적인 활동만 구체적으로 기술한다.
5. 각 유즈 케이스에 대하여 예외적인 경우와 선택적인 경우를 찾아내어 구체적으로 기술한다.

3.1.1 문제설명서 준비

기능 분석을 위한 유즈 케이스 적용 기법의 첫 번째 단계는 문제설명서(problem description statement) 작성이다. 문제설명서는 "시스템이 무엇(what)을 수행하는가" 에 대한 기술이다. 이 설명서는 시스템에 대한 여러 유형의 사용자들과의 인터뷰를 통해 얻어지는 것이 일반적이다. 이 설명서는 결국 개발하고자 하는 시스템의 요구 사항(requirements analysis)이므로 일관성이 있도록 같은 레벨의 추상화를 유지해야 한다.

3.1.2 행위자들의 식별

유즈 케이스는 시스템의 행위자들을 식별하는 것을 전제로 한다. 즉 문제설명서로 부터 시스템을 사용하는 행위자들의 유형을 식별해야 한다. 이는 시스템의 구축을 위해 직접 관련이 있는 **주(primary) 행위자**와 시스템의 작동을 도와주는 **보조(secondary) 행위자**로 분류할 수 있다. 이를 위해 시스템과 상호교류하는 외부 개체(entity)들을 식별하여, 이를 능동적(active) 개체와 비능동적(nonactive) 개체로 구별한다. 여기서 능동적 개체는 사용자의 서비스 요청, 시스템을 관리하는 관리자, 그리고 시스템에 대한 입력 디바이스(device) 등과 같이 시스템의 서비스를 개시할 수 있는 개체들이다. 이러한 능동적 개체들이 바로 행위자이며, 이들은 배경도에서 시스템과 연결된 화살표로서 식별될 수 있다.

3.1.3 각 행위자에 대한 유즈 케이스의 식별

각 행위자는 목표를 가지고 시스템과 서로 일련의 상호동작을 수행한다. 예를 들면, 은행의 고객은 현금인출기를 사용하여 자기 구좌로 부터 현금 인출을 수행한다. 이렇게 고객과 현금인출기와의 일련의 상호작용이 바로 하나의 유즈 케이스로 식별될 수 있다. 각 유즈 케이스는 여러 가지의 시나리오들을 가질 수 있다. 시나리오는 기본적인 시나리오, 예외적인 시나리오 등으로 나눌 수 있다. 다른 유즈 케이스로는 고객이 자기 구좌의 잔액 확인에 대한 요청도 될 수 있다. 각 행위자에 대해서 그 행위자가 참여하는 유즈 케이스들을 식별할 수 있을 것이다.

3.1.4 기본적인 경우에 대한 설명서

유즈 케이스가 비록 시스템과 행위자간의 일련의 상호교류를 정의하지만 예외적인 조건으로 인해 예기치 않았던 상황이 발생할 수도 있다. 예를 들어 고객이 구좌로 부터 현금을 인출하는 유즈 케이스에서 만약 그 구좌의 잔액이 부족한 경우라면 예기치 않는 상황이라 볼 수 있다. 이와 같이 유즈 케이스 중에서 정상적인 상호작용을 기본적인 경우(Basic Course)라고 하며, 비정상적인 교류를 예외적인 경우(Alternative Course)라고 한다. 일단 각 행위자에 대해서 유즈 케이스를 식별한 후에는 각 유즈 케이스의 기본적인 경우에 대한 자세한 설명서를 작성해야 한다. 이러한 작업은 상대적으로 어렵지 않으며, 시스템으로 부터 긍정적(positive)인 반응을 수행하는 유즈 케이스에 대한 조건들만을 식별하면 된다.

3.1.5 예외적인 경우에 대한 설명서

다음으로는 각 유즈 케이스에 대해 예외적인 경우를 식별해야 한다. 이미 기본적인 경우가 작성되었기 때문에, 이를 참고로 시스템으로 부터 부정적(negative)인 반응을 나타내는 상호 작용을 고려하면 된다. 따라서 이러한 예외적인 경우에 대한 시스템과 행위자간의 상호교류가 예외적인 경우를 형성한다.

3.2 배경도의 개발

이러한 방법으로 개발된 최상위 레벨(topmost level)의 자료흐름도는 시스템 주변의 환경(environment)인 시스템 배경을 나타내므로, 이를 배경도(Context Diagram)라 부른다. 배경도에서 시스템은 외부 요소들(external elements)과의 인터페이스를 나타내는 블랙박스(black box) 프로 세스로 표현한다. 따라서 배경도에는 단지 하나의 프로세스(버블)만이 존재한다. 여기서 터미날(terminal)이라고 명칭되는 외부 요소들은 직사각형으로 표기된다. 또한 시스템과 외부 요소들간의 화살표로 표기된 연결은 시스템의 데이타 흐름(입/출력)를 나타내며, 점선으로 표기된 연결은 시스템의 제어 흐름(control flow)를 나타낸다.

배경도를 만드는 순서와 가이드라인은 다음과 같다.

● **배경도를 만드는 순서**

a) 시스템에 대한 프로세스를 그리고 이름을 정한다.

b) 각 행위자에 대하여 외부객체를 그린다.

c) 각 유즈 케이스에 대한 기술 내용을 점검하고 시스템과 행위자 사이에 주고 받는 정보와 사건을 규명한다.

d) 유즈 케이스에 의해 규명된 정보들을 외부객체와 프로세스 사이의 데이타 흐름과 제어 흐름으로 나타낸다.

e) 연관된 데이타 흐름과 제어 흐름들을 묶어 통합된 흐름을 배경도에 나타낸다.

3.2.1 시스템 버블의 작성

구조적 분석 기법에서 배경도는 시스템 자체를 나타내는 하나의 버블과 이 시스템과 교류하는 외부 요소들로 구성된다. 시스템을 나타내는 하나의 버블이 그려지고 이에 대한 이름이 명시된다.

3.2.2 각 행위자에 대한 외부객체(terminator)의 작성

배경도를 만들 때 시스템의 외적 요소들을 표현할 필요가 있다. 이에 따라 유즈 케이스의 행위자들은 시스템과 교류하는 외적 요소들을 나타내기 때문에 이를 배경도에 외부객체로 명시한다.

3.2.3 정보 항목과 사건의 식별

다음 단계로는 각 유즈 케이스의 설명서를 조사한 후, 시스템과 행위자간에 전달되는 정보 항목(information item)들이나 사건(event)들을 식별한다.

3.2.4 데이타 흐름과 제어 흐름의 표현

일단 위에 기술한 요소들이 식별된 후, 버블과 이에 관련 있는 외부객체간의 데이타 흐름과 제어 흐름을 명시한다. 이 과정에서 유즈 케이스에서 식별된 정보 항목은 데이타 흐름이나 제어 흐름으로 표시하고, 사건은 제어 흐름으로 표시한다.

3.2.5 데이타 흐름과 제어 흐름의 집단화

배경도에 많은 데이타 흐름과 제어 흐름이 나타날 수 있으므로, 서로 관련 있는 데이타와 제어 흐름들을 합쳐서 집단화(aggregate)된 흐름으로 표시할 수 있다. 이 과정에서 같은 형(type)과 같은 외적 요소를 가진 흐름들을 합쳐야 한다.

3.3 Diagram Zero의 개발

배경도에 나타나는 단일 프로세스는 Diagram Zero라고 하는 하위 레벨의 자료흐름도(daigram zero)에서 주요 기능들로 분해된다. 배경도가 시스템의 블랙박스를 나타내는 반면, Diagram Zero는 시스템의 내부적인 면을 기능들로 분할하여 나타낸다. 구조적 분석의 일관성 규칙(consistency rule)은 자료흐름도의 순수 입력 및 출력이 상위 레벨의 자료흐름도의 부모(parent) 프로세스(즉, 시스템)의 순수 입력 및 출력과 같아야 한다는 것을 의미하며, 이를 레벨 균형(level balancing)이라고도 한다. Diagram Zero의 각 프로세스에 대해서는 이 프로세스에 대한 기능 분해를 나타내는 하위 레벨의 자료흐름도가 만들어진다. 그러므로 한 프로세스에 대한 하위 레벨의 자료흐름도는 이 프로세스를 구성하는 서브프로세스(sub-process)들간의 정보흐름을 나타낸다.

Diagram Zero를 만드는 순서와 가이드라인은 다음과 같다.

● 레벨 0 자료흐름도 개발 순서

a) 각 유즈 케이스에 대하여 레벨 0 자료흐름도에 프로세스를 하나씩 그리고, 여러 유즈 케이스에 존재하는 공통기능이 있으면 이에 대한 프로세스를 그릴 수 있다.

b) 각 유즈 케이스에 대한 구체적인 기술 내용을 살펴보아 데이타 흐름과 제어 흐름을 추가한다.

c) 다음으로 각 유즈 케이스를 구체적으로 살펴보고 서로 다른 유즈 케이스를 수행하는 동안 기억되어야 할 데이타를 찾아내고 자료저장소를 만들어 자료흐름도에 추가한다.

d) 다음, 필요한 정보에 따라 프로세스들을 자료저장소에 연결한다.

e) 각 행위자에 대한 처리가 비동시적으로 일어나면 일반적으로 프로세스 사이에 직접적인 교류가 일어나지 않으며 자료저장소를 통하여 정보를 교류한다.

f) 레벨 0 자료흐름도를 배경도와 비교하여 모순이 없도록 검사한다.

3.3.1 각 유즈 케이스에 대한 버블의 작성

구조적 분석 기법에서 Diagram Zero는 시스템의 주요 기능들을 정의하고 이들 사이의 정보흐름을 규명하여 시스템의 내부적인 면을 표현한다. 시스템의 각 주요 기능에 대해서 하나의 버블이 있으며, 기능 사이의 상호 교류는 데이타 흐름과 제어 흐름을 사용하여 나타낸다. 이러한 흐름들은 두 기능들 사이 혹은 한 기능과 자료저장소(datastore) 사이에 존재한다. 제안한 기법에서 Diagram Zero는 각 기본적인 유즈 케이스에 대해 하나의 버블로 표시한다. 따라서 Diagram Zero는 최소한 유즈 케이스들의 숫자만큼 버블들을 가질 수 있다. 이 때 유즈 케이스에 이름을 명시한 후 버블에 대한 이름을 명시하는데, 이는 각 버블과 이에 상응하는 유즈 케이스의 연관 작업을 단순화하기 위함이다.

3.3.2 각 공통 기능에 대한 버블의 추가

유즈 케이스들에서 발견되는 각 공통 기능에 대해 버블을 추가할 수 있다. 이러한 공통 기능은 유즈 케이스에 대한 문제설명서가 개발되는 과정에서 식별될 수 있

다. 여기서 다른 유즈 케이스들에서 발견되는 공통적인 기능은 하나의 프로세스로 추상화될 수 있다.

3.3.3 데이타 흐름과 제어 흐름의 추가

유즈 케이스에 대한 문제설명서를 조사함으로서, 데이타 흐름과 제어 흐름을 추가한다. 이 과정은 배경도에서의 3.2.4와 3.2.5의 과정과 같다.

3.3.4 자료저장소의 추가

유즈 케이스에 대한 문제설명서를 조사함으로서, 장시간 저장되어야 하는 데이타를 위해 자료저장소를 추가한다.

3.3.5 데이타 저장과 버블과의 연결

일단 자료저장소가 식별되면, 요구되는 정보의 유형에 따라 버블들을 적절한 자료저장소로 연결한다. 이러한 과정을 거친 후에 Diagram Zero가 완성된다. 여기서 버블들은 비동기적(asynchronous) 트랜잭션들을 나타내기 때문에, 버블들끼리 직접 연결되지 않는 경우가 대부분이다. 이들을 동기화하기 위해서는 버블들은 자료저장소를 통해서 서로 연결되어야 한다.

3.3.6 일관성 검증

마지막 단계로서 배경도에서와 같이 일관성에 대한 검증을 한다. 이를 위해서는 균형(balancing), 프로세스 추적(tracing) 등과 같은 기법들을 이용할 수 있다.

3.4 하위 레벨 자료흐름도의 개발

자료흐름도의 각 프로세스에 대한 하위레벨 자료흐름도를 만드는 순서와 가이드라인은 다음과 같다.

- **자료흐름도의 각 프로세스에 대하여**
  - a) 앞의 두 번째 단계에서 한 것처럼 아래 단계의 자료흐름도를 개발한다.
  - b) 앞의 경우와 달리 각 유즈 케이스의 기본적인 활동, 각 예외적인 경우와 선택적인 경우에 대하여 프로세스를 만들 수 있다.
  - c) 프로세스가 복잡하면 계속 더 낮은 단계로 확장시킨다.
  - d) 모든 원시 프로세스에 대하여 유즈 케이스 및 응용분야에 대한 전문지식을 활용하여 미니 명세서를 만든다.

3.4.1 하위 레벨 자료흐름도의 작성

Diagram Zero의 각 버블에 대해 하위 레벨 자료흐름도를 작성한다. 이 과정에서 기본적인 경우와 예외적인 경우에 대해 각각 하나의 버블로 표현한다. 데이타 흐름, 제어 흐름과 자료저장소에 대한 표현은 Diagram Zero에서의 과정과 동일하다. 그러나 이 과정에서는 프로세스들간의 연결이 가능한데, 이는 이들 프로세스들이 하나의 유즈 케이스에 속하기 때문이다. 즉, 동일한 유즈 케이스에 대해 기본적인 경우와 예외적인 경우로 표현된다.

3.4.2 버블의 하위 레벨 자료흐름도로의 확장

만약 하위 레벨 자료흐름도의 버블들이 복잡하면, 이들은 유즈 케이스 설명서에 따라 하위 레벨 자료흐름도로 더 확장될 수 있다. 그러나 이 과정은 Diagram Zero로 부터 하위 레벨 자료흐름도를 작성하는 기법과 다를 수가 있는데, 이 경우는 유즈 케이스의 기능에 상관되기 때문이다. 그러므로 유즈 케이스 설명서를 조사해서 필요한 부기능들과 이들간의 정보 흐름을 결정할 수 있다.

3.4.3 프로세스 명세서의 추가

모든 원시 프로세스(primitive process)들에 대해서 유즈 케이스 설명서로 부터 이들 프로세스에 대한 명세서를 만들 수 있다.

4. 유즈 케이스 기법의 적용

이 장에서는 앞서 설명한 기능 분해를 위한 유즈 케이스 적용 기법을 [그림 1]과 같은 자판기(vending machine) 시스템에 적용함으로서 이 기법의 장점을 보이고자 한다.

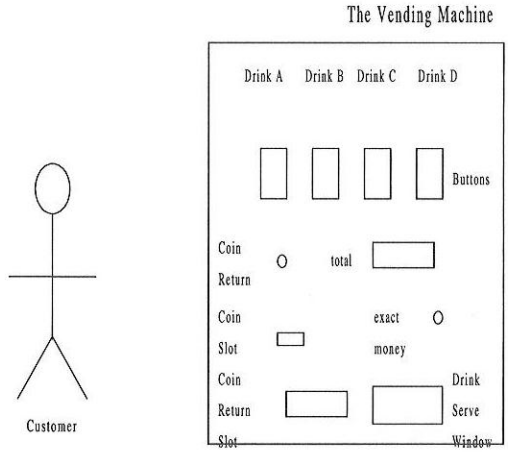

그림 1 자판기 시스템과 사용자

4.1 유즈 케이스의 식별

4.1.1 문제설명서의 준비

다음은 단순한 형태의 자판기 시스템에 대한 문제설명서의 일부를 나타낸 것이다.

자판기는 고객들에게 음료수를 제공하는 자동화 시스템이다. 여러 종류의 음료수가 준비되어야 하지만, 한

고객에게 한 종류의 음료수가 제공된다. 음료수마다 서로 가격이 다를 수 있으며, 이 가격은 시스템에 진열되어 표시되어야 한다. 시스템은 동전 형태로 돈을 받을 수 있어야 한다. 시스템은 현재까지 투입된 동전들의 총 값을 계산할 수 있어야 한다.

시스템은 여러 다른 기능들을 수행할 수 있는 관리자에 의해 유지되어야 한다. 관리자는 음료수의 가격을 변경할 수도 있으며, 모든 트랜잭션에 대한 기록을 보고서 형태로 출력할 수 있어야 한다.

4.1.2 행위자들의 식별

위의 문제설명서로 부터 자판기 시스템의 주 행위자(primary actor)는 시스템으로 부터 음료수를 제공받는 '고객(customer)' 이라는 것을 식별할 수 있다. 그러나 이 시스템을 관리하는 또 다른 행위자인 관리자(operator)가 있어야 한다.

여기서 '관리자'는 시스템을 보조하는 역할로 간주할 수 있으므로 보조 행위자(secondary actor)가 된다. 위의 두 행위자들로 부터 각 행위자에 대해 유즈 케이스를 식별할 수 있다. 이는 시스템과 사용자간의 교류에 의해 기능들을 기술하는 문제설명서를 분석함으로서 가능하다. 본 자판기 시스템에 대해서는 기능들이 다음과 같이 표현될 수 있다.

고객은 음료수를 얻기 위해서는 시스템에 표시된 음료수의 가격 이상의 동전들을 투입한다. 일단 동전이 투입되면 고객이 원하는 음료수를 선택됨에 따라 음료수가 전달된다. 음료수가 전달된 후에는 고객이 동전 반환 버튼을 누를 때 시스템은 잔돈도 함께 반환해야 한다.

4.1.3 각 행위자에 대한 유즈 케이스의 식별

한 고객에게 하나의 유즈 케이스가 존재한다. 고객의 주요 역할은 시스템으로부터 제공(dispense)되는 음료수를 얻는 것이다. 따라서 이에 대한 유즈 케이스는 고객이 동전를 투입하여 음료수에 대해 선택하는 것이다. 시스템은 음료수를 제공하며 고객의 요청에 따라 잔돈을 반환하는 것이다. 그러므로 고객에 대한 유즈 케이스는 ***음료수 분배(dispense drink)*** 라고 명시할 수 있다. 관리자는 하루의 거래 실적을 나타내는 보고서를 출력하며, 또한 음료수의 가격을 원하는 가격으로 변경할 수 있다. 여기서 두 종류의 유즈 케이스가 식별될 수 있으며, 이는 ***매일 보고서(get daily report)*** 와 ***가격 변경(change price)***으로 각각 명시할 수 있다. ***음료수 분배***의 경우에서는 ***동전 투입량 부족(amount is less)***, ***음료수 부족(drink is not available)***, 그리고 ***고객의 거래 취소(customer cancels the transaction)***와 같은 예외 경우들을 식별할 수 있다.

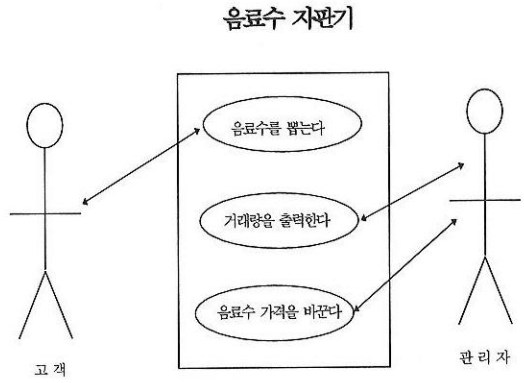

그림 2 행위자 입장에서 본 자판기에 필요한 기능들

그림 2는 행위자 입장에서 본 자판기의 기능을 보여주고 있다.

4.1.4 기본적인 경우에 대한 기술

위에서 식별한 각 유즈 케이스에 대해서, 우선 기본적인 경우에 대해 자세한 기술을 하고자 한다. 예를 들면, ***음료수 분배***에 대한 기본적인 과정은 다음과 같이 기술될 수 있다.

- **"음료수를 뽑는다"**

-이 기능은 고객이 자판기에 첫 번째 동전을 넣었을 때 시작된다. 자판기는 금액 표시란에 입력된 금액(총액)을 보여준다.

-고객이 자판기에 동전을 더 넣으면, 자판기는 금액을 증가시키고 금액 표시란에 총액을 보여준다.

-표시된 총 금액이 고객이 원하는 음료수의 가격과 같거나 더 많을 때, 고객은 음료수 버튼을 누름으로써 음료수를 선택할 수 있다.

-고객이 원하는 음료수에 대한 버튼을 누를 때, 자판기는 음료수를 제공한다.

-음료수가 제공된 후, 자판기는 현재 총액에서 음료수의 가격을 빼고 수정된 총액을 보여준다.

-고객은 잔돈 회수 버튼을 누르고 자판기는 금액 표시란에 보여진 총액과 같은 금액을 되돌려준다.

-마지막으로 자판기는 거래를 기록하고, 고객에게 제공된 음료수의 총 양을 증가시키고, 고객에게 지출된 거스름돈의 양만큼 거스름돈의 총액을 감소시킨다.

4.1.5 예외적인 경우의 기술

위에 기술한 ***음료수 분배*** 유즈 케이스에 대한 예외적인 경우는 다음과 같다.

- **"동전 투입량 부족"**

-고객이 음료수의 선택을 하지만, 투입된 동전이 음료수 가격보다 적어서 음료수는 제공이 안된다. 이 때 고객은 동전 반환 버튼을 누르면 동전이 반환된다.

● **"음료수 부족"**

-고객이 음료수 선택을 하지만 음료수가 모자라서 음료수 제공이 안된다. 이 때 고객이 동전 반환 버튼을 누르면 동전이 반환된다.

4.2 배경도의 개발

본 절에서는 위에서 작성된 유즈 케이스 설명서를 이용하여 기능 분해 모델을 개발하고자 한다. 이는 시스템에 대한 유즈 케이스의 관점을 기능적인 관점으로 변환하는 것이다. 3.2.1에서 3.2.5까지 설명된 지침에 따라 [그림 3]와 같은 배경도를 개발할 수 있다. 여기서 배경도를 단순화하기 위해 일부 데이타 흐름과 제어 흐름들은 하나로 합쳐서 집단화된 이름을 명기할 수 있다. 예를 들어, 데이타 흐름 ***음료수 선택(drink request)***은 기본적으로 동전(coin)과 선택(choice)라는 두 개의 데이타 흐름들로 구성된다. 이렇게 집단화된 데이타나 제어 흐름은 데이타 사전에 세분화된 요소 흐름(component flows)들로 정의된다.

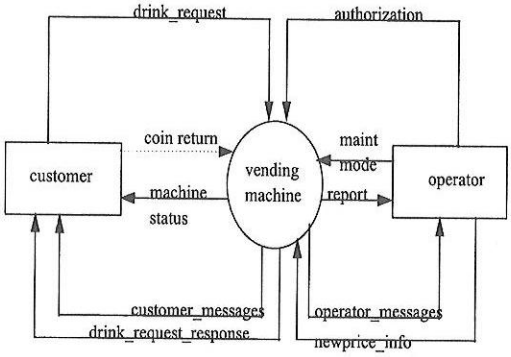

그림 3 자판기 배경도

4.3 Diagram Zero의 개발

3.3.1에서 3.3.6까지 설명된 지침에 따라 [그림 4]과 같은 자판기 Diagram Zero를 개발할 수 있다. 여기서 각 유즈 케이스는 하나의 버블로 표현된다. 특히 ***변경 시스템 모드(change system mode)***라는 유즈 케이스는 ***가격 변경(change price)***과 ***매일 보고서(get daily report)***라는 유즈 케이스의 공통적인 behavior를 표현하고 있다. 이들 유즈 케이스에서 관리자는 이러한 기능들을 수행하기 위해서 '운영(operation)'에서 '관리(maintenance)'로 시스템 모드를 변경할 필요가 있다. 이러한 이유로 Diagram Zero에서 ***변경 시스템 모드(change system mode)***에 대한 버블을 도입하고 있다. 또한 유즈 케이스들에 대한 문제설명서를 조사함으로서 데이타 흐름과 제어 흐름들이 첨부될 수 있다.

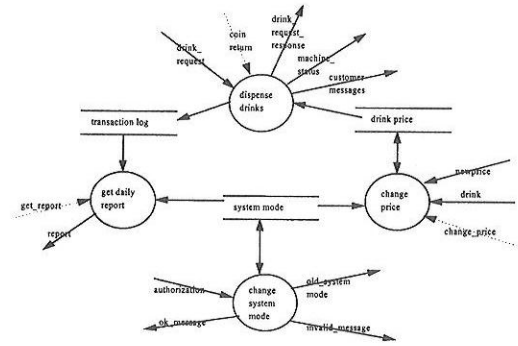

그림 4 자판기 Diagram Zero

4.4 하위 레벨 데이타흐름도 개발

Diagram Zero의 각 버블에 대해서 하위 레벨 자료흐름도를 개발하고자 한다. 각 버블은 유즈 케이스를 나타내므로 이에 대한 하위 레벨 자료흐름도를 개발할 수 있다. 예를 들면, Diagram Zero의 ***음료수 분배(dispense drinks)*** 버블에 대해서는 [그림 5]와 같은 자료흐름도를 얻을 수 있다. [그림 5]에서와 같이 각 유즈 케이스에 대해 정상적인 경우를 나타내는 하나의 버블과 각 예외적인 경우에 대해 하나의 버블이 있을 수 있다. 이러한 유즈 케이스들은 분석가로 하여금 해당 에플리케이션에서 필요로 하는 기능을 쉽게 분해할 수 있도록 도와준다.

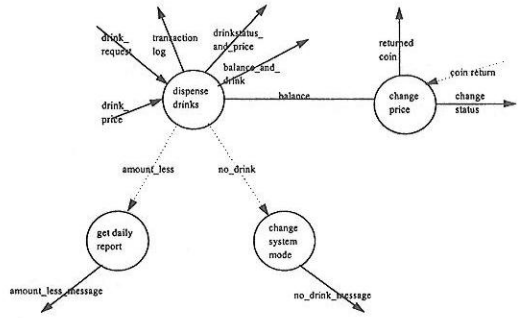

그림 5 음료수 분배(dispense drinks)에 대한 자료흐름도

4.5 버블의 저급 레벨 DFD로의 전환

만약 유즈 케이스의 정상적인 경우이든 예외적인 경우이든 더 이상의 기능 분할이 요구될 만큼 복잡하다면, 이를 조사하여 분해한다.

4.6 프로세스 명세서의 추가

각 원시 프로세스에 대해 명세서를 추가해야 한다. 이 경우 프로세스 명세는 프로세스의 논리적인 기능에 대한 비정형적인 설명서로 간주할 수 있다.

5. 결 론

본 논문은 유즈 케이스를 적용하여 시스템의 기능을 분해하는 기법을 제안하였다. 이 기법의 장점으로는 시스템이 사용자의 관점에서 이해되어 점진적으로 세분화될 수 있다는 것이다. 이는 기본적으로 시스템과 외부 사용자간의 상호교류에 기초한 것으로서, 유즈 케이스로부터 시스템의 기능과 동작을 모델링하는 기법이다. 이 기법에서 유즈 케이스는 자료흐름도에서 프로세스들을 식별하기 위해 사용된다. 본 논문에서 제안하고 있는 유즈 케이스 적용 기법은 기능 분석 모델링에 필요한 경험적 법칙(heuristic rule)과 지침을 포함하고 있으며, 이들은 잘 정립된 개념과 단계에 기반을 둔 것이다.

제안한 기법의 주요 장점은 유즈 케이스를 이용하면 개발자들이 시스템의 기능들을 분해하기가 쉽다는 것이다. 이와 같은 사실은 전형적인 구조적 분석에서는 아직까지 시스템의 기능들을 식별하는 분명한 지침(guide-lines)이 없다는 것과 대조된다. 둘째, 본 기법은 사용자와 개발자간의 상호 의사소통 능력을 향상 시킬 수 있다. 일반적으로 사용자는 개발 대상이 되는 시스템에 대해서 고객 관점에서 더 잘 이해하고 있다. 이러한 사실이 전통적인 구조적 분석 기법에서 적용되고 활용되기가 상대적으로 어려운 면을 가지고 있다. 이는 요구사항 분석단계에서 사용자가 원하지도 않고 또한 이해하기도 힘든 불필요한 시스템의 내부 사항이나 작동 방법들이 표출될 수 있기 때문이다. 셋째, 본 기법은 하향식(top down) 접근 방법을 도입으로서 구조적 분석과 같은 전형적인 기능 분해 기법과 자연스럽게 병합될 수 있다. 이 사실은 매우 중요한 장점으로 인식될 수 있다. 반면 최근에 제안된 사건 분할 기법은 하향식도 상향식도 아닌 혼합식으로 기능 분해 기법에 적용하기가 어렵다는 점을 감안할 수 있다. 넷째, 유즈 케이스를 이용하여 설계된 시스템은 고객(customer)들에 의해 쉽게 검증될 수 있다는 점이다. 이는 일반적으로 고객들이 시스템에 대한 모델을 이해할 때, 시스템이 사용자에게 제공할 수 있는 기능들이 모델에 반영되어야 한다고 믿기 때문이다. 본 기법은 전형적인 구조적 분석을 이용하여 만드는 시스템 모델을 사용자 관점에서 이해하기 쉽지 않다는 점을 극복하려는 시도라 볼 수 있다.

참 고 문 헌

[1] C. Gane and T. Sarson, *Structured Systems Analysis: Tools and Techniques*, Englewood Cliffs, New Jersey, Prentice Hall, 1978.

[2] T. DeMarco, *Structured Analysis and Systems Specification*. Englewood Cliffs, New Jersey, Prentice Hall, 1979.

[3] W. Stevens, G. Meyers and L. Constantine, "Structured Design", IBM Systems Journal, May, 1974.

[4] E. Yourdon and L. Constantine, *Structured Design*, New York: YOURDON Press, 1975.

[5] G. Meyers, *Reliable Software Through Composite Design*. New York: Petrocelli/Charter, 1975.

[6] E. Yourdon, *Modern Structured Analysis*, YOURDON Press, 1989.

[7] I. Jacobson, et al., Object-*Oriented Software Engineering: A 유즈 케이스 Driven Approach*, Addison Wesley, 1992.

[8] G. Booch, *Object-Oriented Analysis and Design with Applications*, second ed., The Benjamin/Cummings Publishing Company Inc., 1994.

[9] J. Rumbaugh, et al., *Object-Oriented Modeling And Design*, Prentice Hall, 1991.

[10] J. Rumbaugh, *"OMT: The development process", in the Journal of Object-Oriented Programming*, May 1995, pp. 8-16.

[11] J. D. Warnier, *Logical Construction of Programs*, Van Nostrand Reinhold, 1974.

[12] J. D. Warnier, *Logical Construction of Systems*, Van Nostrand Reinhold, 1981.

[13] K. T. Orr, *Structured Systems Development*, Yourdon Press, New York, 1977.

[14] K. T. Orr, *Structured Requirements Definition*, Ken Orr & Associates, 1981.

[15] M. A. Jackson, *System Development*, Prentice-Hall, 1983.

[16] E. Berard, "Be careful with 유즈 케이스s", The Object Agency, Inc., 1996.

윤 청

1979년 서울대학교 물리학과 졸업(학사). 1983년 Sagamon State University Computer Science 졸업(석사). 1988 Northwestern University Computer Science 졸업(박사). 1983 ~ 1985 Wayne State College 전임강사. 1985 ~ 1987 Nothwestern University 전임강사. 1988 ~ 1993 Bell Communication Research 선임연구원(MTS). 1993 ~ 현재 충남대학교 컴퓨터과학과 부교수. 관심분야는 소프트웨어 공학, CALS, 객체지향 모델링 및 설계

김 응 모

1981년 2월 성균관대 수학과 학사. 1986년 5월 Old Dominion University 컴퓨터과학과 석사. 1990년 2월 Northwestern University 컴퓨터과학과 박사. 1998년 ~ 현재 성균관대학교 전기 전자 및 컴퓨터공학부 부교수

배 두 환

1980년 서울대학교 조선공학과 학사. 1987년 위스콘신-밀워키대학 전산학 석사. 1992년 플로리다 대학 전산학 박사. 1992년 ~ 1994년 플로리다 대학 전산학과 조교수. 1995년 ~ 1996년 한국과학기술원 정보 및 통신공학과 조교수. 1996년 ~ 현재 한국과학기술원 전산학과 조교수.