공 학 석 사 학 위 논 문

UML을 적용한 Use Case Diagram 추출 기법

이 논문을 공학석사 학위논문으로 제출함

2005년 2월

부 경 대 학 교 대 학 원

전자공학과

김 학 철

김학철의 공학석사 학위논문을 인준함

2004년 11월 24일

주 심 이학박사 윤 성 대 

위 원 공학박사 조 우 현 

위 원 공학박사 김 종 진 

목 차

<그 림 차 례>

<표 차 례>

Use Case Diagram Extraction Technique which Applies UML

Hak-Chul Kim

Abstract

We have to carry out clear requirements analysis for the successful development of software. The UML gives the ways to grasp user or customer requirements and decide the boundary of business systems from use case modeling. We present how to extract use case diagram from the requirements specification systematically by applying the standardized rules as a part of the study for use case modeling. We modify requirements specification by applying R_{RS}(Rule for Requirements Specification), R_A(Rule for Actor), R_U(Rule for Use Case) and R_R(Rule for Relationship) to the modified requirements specification separately and then become to make out use case diagram in the end. By Applying the rules presented to the requirements specification for school work management in this thesis, we can reduce the existing difficulties of extracting use case diagram based on the narrative descriptions without any standardized rules.

제 1 장 서 론

객체지향 방법론은 현실세계의 객체란 개념에 초점이 맞추어 있고, 현재는 여러 가지 모델링 기법 및 방법론들이 소개, 연구되고 있다. 객체지향 방법론은 객체 추출, 추상화, 상세화, 단순화된 실제 세계의 객체들을 표현함으로써 단순성과 재사용성이 우수하다. 현실 세계에 대한 것을 객체지향 방법론을 적용하기 위해서는 분석, 설계 과정을 필수적으로 거치게 되며, 이에 필요한 산출물들이 만들어진다. 이러한 산출물들은 그 프로젝트 관계자들이 서로 이해할 수 있어야 하고, 프로젝트 진행에 있어서 표준적인 표기법이 필요하다. 이것이 UML이고, UML은 소프트웨어 시스템의 시각화, 기술화, 구축화에 필요한 산출물을 문서화하는데 사용되는 모델링 언어이다. Rational Software와 그 협력 회사들에 의해 개발되었으며, OMT, Booch, OOSE/Jacobson에의 모델링 언어의 장점을 계승 및 발전하고 있다. OMG의 공식 표준으로 자리 잡은 UML은 4개의 Model과 8개의 Diagram으로 나누어진다. 4개의 Model로는 Use Case Model, 설계 모델, 구현 모델, 분산/배치 모델이 있고, 8개의 Diagram은 Use Case Diagram, Class Diagram, State Diagram, Activity Diagram, Sequence Diagram, Collaboration Diagram, Component Diagram, Deployment Diagram으로 구성된다.[1][2][6][31]

대부분의 회사들이 표준으로 계정된 UML을 가지고 그들의 개발 프로세스에 적용하고 있는데 가장 많이 사용되고 있는 개발 프로세스가 Rational Unified Process(RUP)이다. RUP는 유스케이스 중심(Use Case Driven), 아키텍처 중심(Architecture Centric), 반복과 점진(Iterative and Incremental) 개발이라는 3가지 핵심 요소를 가진다.[2][7][34]

이 중 Use Case Driven의 의미는 시스템 개발에 있어 Use Case Model 이후의 Model 들이 Use Case Model로부터 시작된다는 것이다. Use Case Model은 분석 초기 단계에서 사용자의 요구사항을 모델링하는 도구로서, 구축 대상 시스템이 최종 사용자에게 제공하는 서비스들을 사용자의 용어를 대상으로 기능별로 표현한다. 이것은 복잡한 표기법 없이 Actor와 Use Case 만의 관계를 통해 Use Case Diagram을 산출한다는 것을 의미한다. 그리고 간결하

게 시스템의 기능을 파악할 수 있도록 지원하지만, 서술적인 묘사로 다른 모델들에 비해 모호하다는 단점을 가지고 있다.[24][29] 따라서 막상 요구사항을 분석하여 Actor와 Use Case 및 관계를 추출하여 Use Case Diagram을 작성할 때 어려움을 가지게 된다.

본 논문에서는 사용자의 요구사항을 반영한 요구사항 기술서로부터 Use Case 작성에 필요한 Actor, Use Case, 관계 추출에 필요한 몇 가지 규칙(R_{RS}, R_A, R_U, R_R)을 제시하고 그것을 사용함으로써 좀 더 쉽게 Use Case Diagram을 작성할 수 있도록 제안하고자 한다.

제 2 장 UML (Unified Modeling Language)

2.1 UML과 모델링

시대가 변화할수록 소프트웨어 개발은 규모면에서나 구현면에서 어려워지고 있다. 컴퓨터 언어 개발자와 도구 개발자들, 그리고 방법론의 대가들이 최선을 다하여 노력하고 있지만 사실은 그렇지 못한 것이 현실이다. 비교적 작은 시스템일지라도 내부는 매우 복잡한 양상을 띠고 있는 경우가 많다. 따라서 모델이 필요하며, 모델은 현실세계의 축소판이며 소프트웨어가 본질적으로 가지고 있는 복잡성을 이해하는데 도움을 준다.

UML은 소프트웨어 개발에 참여하는 모든 사람들이 구축하려고 하는 소프트웨어에 대한 모델을 만들 수 있도록 도와준다. 소프트웨어 모델은 시스템을 가시화하고, 그 구조와 행동을 명세화하며, 시스템을 구축하고, 그 과정에서 내려진 결정들을 문서화 하도록 도와준다. 그러면 좋은 소프트웨어 모델의 몇 가지 조건에 대해서 한번 살펴보자.[2][5][23]

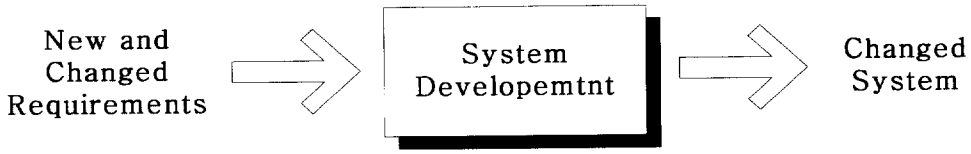
첫째로 좋은 모델은 현재의 일부 문제 혹은 전체를 해결하기 위해 프로젝트 팀이 추구하려는 갑작스러운 계획에 잘 적응할 수 있어야 한다. 필수적인 개념에 치중하고 부수적인 것들은 무시해버린 모델일수록 훨씬 더 나은 버전을 제시해주면서 오래 지속될 수 있어야 한다.

둘째로 좋은 모델은 언제든지 서로 다른 검토자들에게 상이한 수준의 내용을 제공해야한다. 프로젝트 이해 당사자들은 다양한 관점을 가지고 개발 시스템을 바라본다. 따라서 좋은 모델은 임원용 요약 수준에서부터 아주 상세한 수준의 내용까지 다양한 검토 수준을 제공해야 한다.

셋째로 좋은 모델은 실세계와 연결되어 있어야 한다. 모델을 엄격하게 개발하고 꾸준하게 개선시켜야만 프로젝트 팀이 기대하는 시스템과 흡사한 행동을 하는 그런 시스템을 생산할 수 있다.

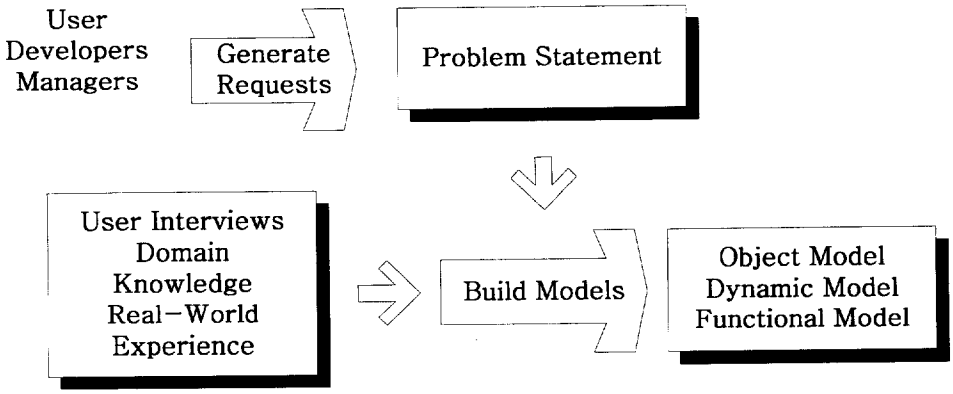
넷째로 좋은 모델은 시스템의 필수적인 요소들을 파악하는 다른 모델들과 잘 어울릴 수 있어야 한다.

UML의 발전 과정을 살펴보면, 1989년경 다양한 객체지향 모델링 기법들이 등장해서 나름대로 위세를 떨치며, 몇 년간 사람들은 "방법론 전쟁"이라는 소용돌이 속에서 살게 되었다. 1990년대 중반, 전쟁의 포연이 사라지자 객체 지향 사회에서도 괄목할 만한 사건이 있었는데, Grady Booch, Jim Rumbaugh, Ivar Jacobson이다. 그러면 이들이 제시한 방법들에 대해서 한번 살펴보자.



[그림 2.1] Jacobson의 시스템 개발 관점

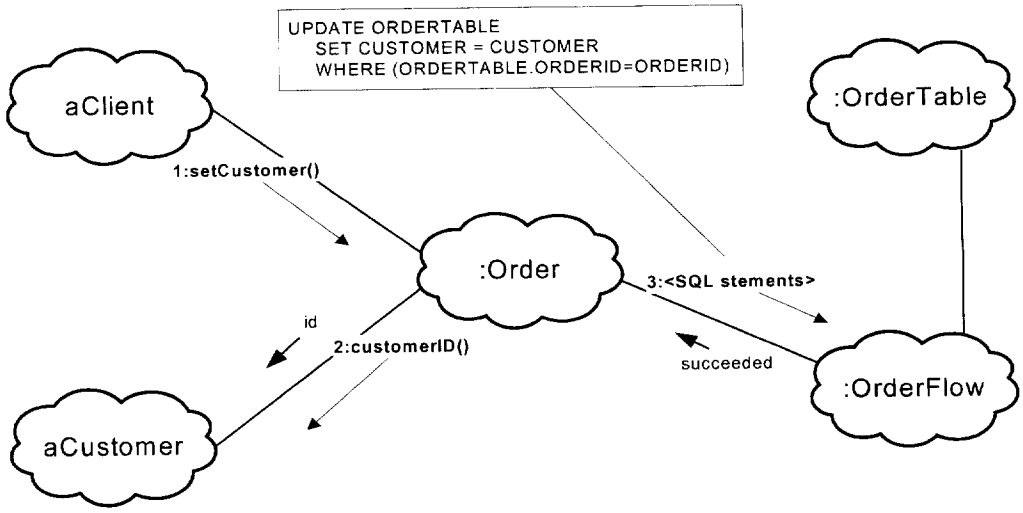
Ivar Jacobson은 Objectory 방법을 주창했는데, [그림 2.1]에 잘 나타나 있다. 사용자의 끊임없이 변화하는 새로운 요구사항에 대한 시스템 개발의 관점에서 변화된 시스템을 바라보는 방법론이다.[3][12]



[그림 2.2] Jim Rumbaugh의 분석 단계 요소

Jim Rumbaugh의 Object Modeling Technique(OMT)는 데이터-중심 시스템을 대상으로 꾸준한 노력을 기울인 결과로 탄생했다. 이것은 문제영역을 모델링하는데 가장 큰 장점을 가진다. [그림 2.2]는 Jim Rumbaugh의 분석 단계 요소에 대한 그림이다. 이것은 OMT 방법의 첫 단계인 분석 단계 주요 요소들을 보여주고 있는데, 사용자와 개발자들의 요구사항이 발생하면 그것을 문

제 영역으로 도출시키고, 사용자 인터뷰와 도메인, 지식이나 경험 등을 취합하여 모델로 세우고 이것을 Object Model, Dynamic Model, Functional Model 요소로 만든다.



[그림 2.3] Booch의 세부 설계

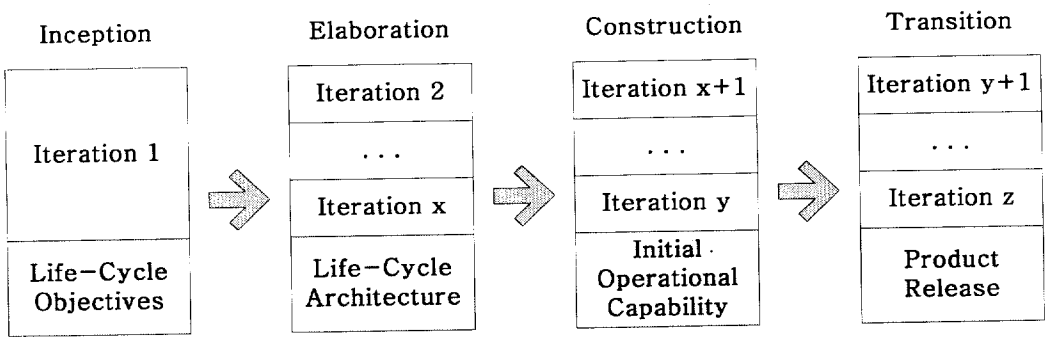
Grady Booch 방법의 장점은 세부 설계와 코딩이 특징이다. [그림 2.3]은 고객의 id로 접근하여 SQL문을 실행하는 SQL 트랙잭션에 대한 전형적인 Booch 방법이다.

이러한 세 가지 방법들이 서로의 장점을 융합하여 나온 것이 UML이다. UML은 Rational Software와 그 협력 회사들에 의해 개발되었으며, OMT, Booch, OOSE/Jacobson에서 발견되는 모델링 언어의 장점을 계승 및 발전하고 있다.

2.2 UML과 개발공정

UML은 개발공정과 무관하게 독립적으로 설계되었다. 그리고 UML 설계자들은 UML과 같이 사용하기에 적합한 공정에 대해서 Use Case Driven, Architecture Centric, 반복과 점진(Iterative and Incremental)의 세 가지 주요한 특징을 권고하고 있다.

소프트웨어 시스템의 수명은 단계별 주기로 표현할 수 있다. 각 주기는 새로운 버전을 고객에게 제공하는 것으로 종료된다. Unified Process에서 각 주기는 4단계로 나누어진다. [그림 2.4]는 이러한 단계를 잘 보여주고 있다. 단계란 단순하게는 두 주요 일정 사이 기간을 말하는데, 각 단계는 관리자들이 개발을 계속할 것인지 말 것인지 결정하는 주요한 의사결정 시점이기도 하다.[2][12] 그러면 각 단계들에 대해 좀 더 살펴보자



[그림 2.4] 단계와 주요 일정

도입 단계(Inception Phase)의 주요 목표는 제안 시스템의 타당성을 입증하는 것이고 수행되는 작업을 살펴보면, 시스템 범위 설정(포함되는 것과 포함되지 않는 것은 무엇인가?)과 6 기본 모델[그림 2.5] 초안을 통해 Architecture 후보 수립, 그리고 위험을 식별하고 언제, 어떻게 공략할지 여부에 대한 계획 수립, 프로젝트를 추진할 가치가 있다는 것을 입증하기 위한 비용, 투입 공수, 일정 그리고 프로젝트 품질 등에 기초한 사업타당성 분석이다. 도입 단계 말 주요 일정은 수명주기목표(Life-Cycle Objectives)를 설정하는 것이다. 프로젝트가 이 지점에 도달했다는 징표로는, 주요 이해 당사자들이 제안 시스템의 범위에 동의했다. 후보 아키텍처가 명백하게 핵심적인 고수준 요구 사항을 다

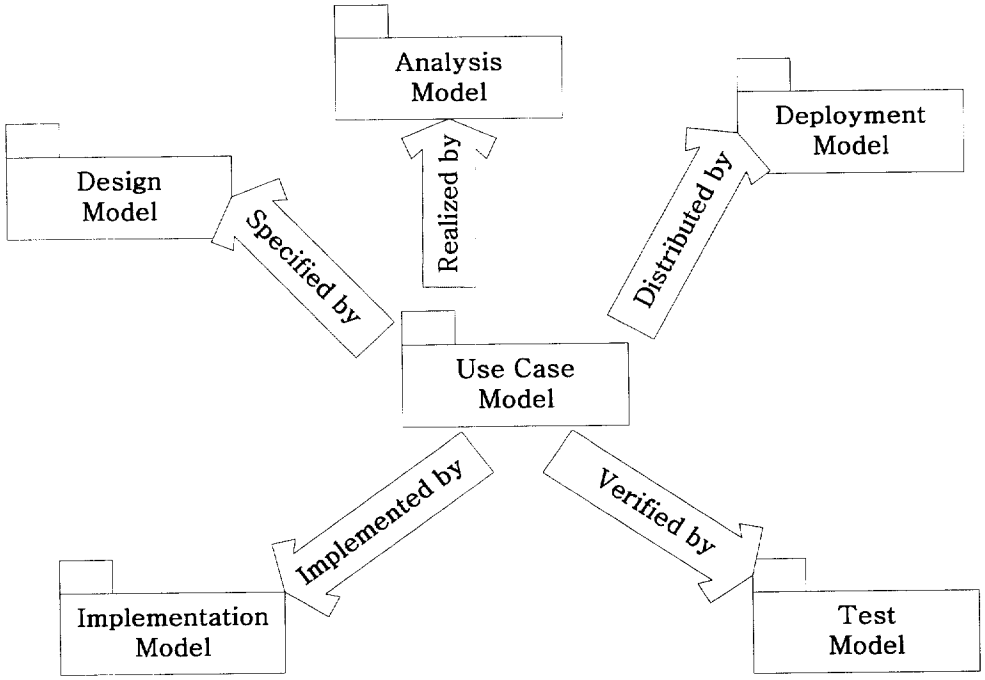
루고 있다. 사업 타당성 분석 결과가 개발을 계속해도 좋다는 청신호를 보내기에 충분하다.

정련 단계(Elaboration Phase)의 주요 목표는 주어진 자금, 일정 그리고 개발 프로젝트가 당면한 다른 제약들에도 불구하고 새로운 시스템을 구축할 수 있다는 가능성을 보여주는 것이다. 정련 단계에서의 작업 과정을 살펴보면, 기능 요구 사항의 상당부분 파악, 아키텍처 후보를 완전한 아키텍처 기준선으로 확장하고, 시스템 아키텍처를 설명하는 내부 배포판으로 활용하고, 프로젝트 위험성을 지속적으로 관리하고, 프로젝트 사업 타당성을 최종 확정하고 프로젝트 계획을 수립하여 다음 구축단계를 진행하기에 충분할 정도로 상세한 내용 반영한다. 정련 단계 말 주요 일정은 수명주기 아키텍처 (Life-Cycle Architecture) 확립이다. 프로젝트가 이 지점에 도달했다는 징표로는 새로운 시스템에 대한 기능 요구사항 대부분에 쓰임새 모델로 파악되었다. 아키텍처 기준선이 작고, 효율적인 시스템으로서 개발을 계속하기에 무리 없는 확고한 기반을 제공한다. 타당성 분석이 승인되었고 구축 단계를 수행할 프로젝트 계획 초안이 마련되었다.

구축 단계(Construction Phase)의 주요 목표는 사용자 환경에서 성공적으로 실행되는 시스템을 구축하는 것이다. 구축단계에서는 시스템을 반복적, 점진적으로 구축한다. 한편으로 시스템의 실행 가능성을 실행 상태에서 시험을 통해 확인한다. 구축 단계 말 주요 일정은 초기 실행 능력 (Initial Operational Capability)의 입증이다. 프로젝트가 이 지점에 도달했다는 징표로는 베타 시험을 통해 고객들이 일부 완전하게 동작하는 실행 시스템을 확인하는 것이다.

전이 단계(Transition Phase)의 주요 목표는 완벽하게 시스템을 고객에게 인도하는 것이다. 전이 단계에서는 이전 단계에서 식별되지 않았던 문제점들을 보완하기 위해 오류를 찾아내서 시정한다. 전이 단계 말 주요 일정은 제품 배포(Product Release)이다.

[그림 2.5]와 같이 Unified Process에는 4단계를 가로지르는 5 워크플로가 있다. 각 워크플로는 다양한 프로젝트 작업자들이 수행하는 일련의 활동으로 구성된다.[34]



[그림 2.5] Unified Process 6 기본 모델

요구사항(Requirement) 워크플로의 주요 활동은 Use Case Model을 만드는 것이다. Use Case Model은 모델링하려는 시스템의 기능 요구사항을 파악한다. 이 모델은 프로젝트 이행 당사자들이 시스템 기능과 지켜야할 조건들에 합의하도록 만든다. [그림 2.5]의 Unified Process 6 기본 모델에서 Use Case Model이 다른 모델의 기초가 되는 과정을 보여주고 있다. 그러면 그 각각의 모델에 대해 살펴보자.

분석(Analysis) 워크플로의 주요 활동은 분석 모델을 만드는 것이다. 분석 모델은 Use Case Model로 파악한 기능 요구 사항을 정제하고 구조화 시키는 것을 도와준다. 이 모델은 Use Case 실현을 포함하는데, 이것은 Use Case를 설계와 구현 작업에 적합하도록 더욱 발전시킨 것이다. 분석 모델은 시스템

아키텍처에서 설계 View와 프로세스 View에 참여한다.

설계(Design) 워크플로의 주요 활동은 설계 모델을 만드는 것이다. 설계 모델은 Use Case Model과 분석 모델을 재료로 삼아 Use Case를 물리적으로 실현하는 것을 설명한다. 설계 모델은 구현모델을 추상화한 것이다. 설계 모델은 시스템 아키텍처에서 설계 View의 중심에 위치한다.

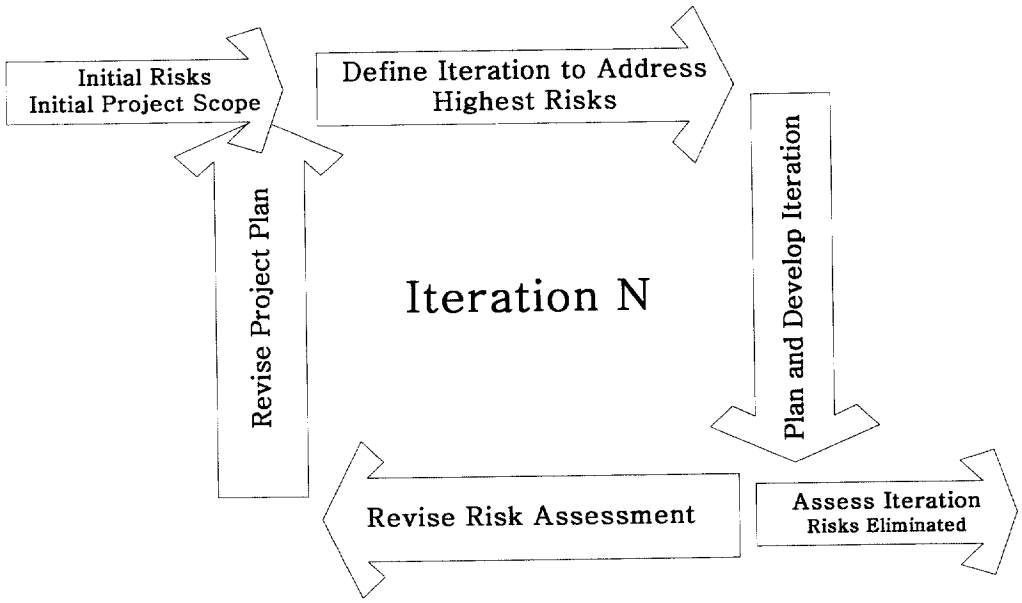
구현(Implementation) 워크플로의 주요 활동은 구현 모델을 만드는 것이다. 구현 모델은 어떻게 설계 모델 요소들을 패키지로 묶어서 소프트웨어 컴포넌트로 만들 것인지를 설명한다. 구현 모델은 시스템 아키텍처에서 구현 View의 중심에 위치한다.

시험(Test) 워크플로의 주요 활동은 시험 모델을 만드는 것이다. 시험 모델은 구현 모델을 구성하는 실행 컴포넌트에 대해서 어떻게 통합(Integration) 시험과 시스템 시험을 실시할 것인지 설명한다. 시험 모델은 통합 시험, 시스템 시험 이전에 각 개발자들이 단위 시험을 어떻게 실시할지도 설명한다.

Unified Process 단계는 반복이라는 기법으로 더 작게 쪼개어진다. 반복(Iteration)이란 쉽게 생각하면, 프로젝트 시간 단축으로 잘게 쪼갠 미니-프로젝트라고 볼 수도 있다. 각 반복은 시스템의 증분(Increment)을 만들어 낸다. 증분이란 이전 반복에서 만들어낸 배포판 위에 추가하거나 개선된 기능을 더한 시스템 개정판을 말한다.

[그림 2.6]은 소프트웨어 개발에 있어서 반복과 점진적 접근법의 핵심을 보여준다. 먼저 초기의 위험 요소와 프로젝트 영역이 들어오면, 큰 위험요소를 정의하고, 개발과 계획을 반복하고, 위험을 재평가 개정하고, 프로젝트 계획을 다시 세우는 이러한 반복이 계속된다.

반복과 점진적 접근법을 사용하면 우선 프로젝트에 존재하는 위험을 평가하면서 개발공정을 시작한다. 개발 프로젝트에서 흔히 볼 수 있는 위험으로는 요구사항, 개발 팀의 수준, 기술 그리고 정책 등과 같은 것들이 포함된다.



[그림 2.6] 반복과 점진 개발

2.3 UML의 구성

UML은 4개의 Model과 8개의 Diagram으로 구성된다. 이렇게 다양한 Diagram을 그리는 이유는 여러 가지 시각으로 우리가 만들고자 하는 시스템에 대하여 구현하기 전에 미리 만들어 보는 것이다. 시스템의 정적인 면을 나타내는 Class Diagram이 있고, 동적인 면을 나타내는 Collaboration Diagram, Sequence Diagram, State Diagram, Activity Diagram, Deployment Diagram, Component Diagram으로 구성되어져 있다. 그 외에 다른 모든 다이어그램 작성의 기반이 되는 Use Case Diagram이 존재한다.

Use Case Diagram은 시스템 구축 초기에 필요한 시스템 요구 분석의 결과물로써, 사용자와 개발자간의 상호 대화 수단으로 사용될 수 있다. Actor와 Use Case를 사용하여, 시스템 외부의 작용과 시스템 내부적인 기능을 도식화함으로써, 시스템이 최종적으로 갖추어야할 기능들을 묘사한다. 또한, Use Case Diagram은 요구 분석이후 설계와 분석, 구현에서의 기초가 되는 Diagram이 될 수 있다.

Class Diagram은 시스템의 정적인 구조를 나타내는 Diagram으로 시스템 내부에 존재하는 클래스들을 선별하여 나타내고 각 클래스들의 속성(Attribute)과 행위(Behavior)를 기입하고 이들 클래스들 사이의 여러 가지 관계(Relationship)를 표현한다.

Sequence Diagram은 시스템의 동적인 면을 나타내는 대표적인 Diagram으로 메시지의 흐름을 나타낸다. Collaboration Diagram과 동일한 내용을 나타내지만, 관점이 서로 다르다. Sequence Diagram은 상호작용의 시간적인 순서에 중심을 두고 다이어그램을 표현하며, 시간의 진행은 수직선상에서 위에서 아래로 진행된다.

Collaboration Diagram은 Sequence Diagram과 마찬가지로 메시지의 흐름을 나타낸다. 하지만 Collaboration Diagram은 객체와 객체들의 역할 사이의 관계에 중심을 두고 Diagram을 표현한다.

상태 다이어그램(State Diagram)은 객체들의 상태의 전이를 나타낸다. 시스템에서 사용되는 오브젝트의 어떤 상태가 필요로 하며, 그 상태의 변화에 따라 어떠한 행위를 하는지에 대해서 표현하게 된다. 객체가 수없이 많기 때문에 특별히 관심을 가져야 할 객체에 관하여 그리고 특정 조건에 만족하는 기간 동안의 상태를 표시한다.

Activity Diagram은 UML에서 가장 늦게 추가된 다이어그램으로, 객체들의 행위의 전이를 나타낸다. Flowchart가 UML에 접목되었다면 가장 쉽게 이해될 것이다. 시스템 내부에 존재하는 여러 가지 행위들 그리고 각 행위의 분기, 분기되는 조건 등을 모두 포함하게 된다.

Component Diagram은 소프트웨어의 물리적 단위(exe, dll등)의 구성과 연결 상태를 나타낸다. 컴파일의 단위를 어떻게 가져 갈 것인지에 대한 표현이라고도 할 수 있다.

Deployment Diagram은 프로세스 간, 하드웨어 간 장비와 이들 간의 연결과 같은 시스템의 하드웨어와 소프트웨어의 물리적 구조를 보여준다. 구현환경을 어떻게 배치할 것이지, 노드에 배치하고 각 노드에는 어떠한 컴포넌트를 배치시킬 것인지 등을 나타낸다.

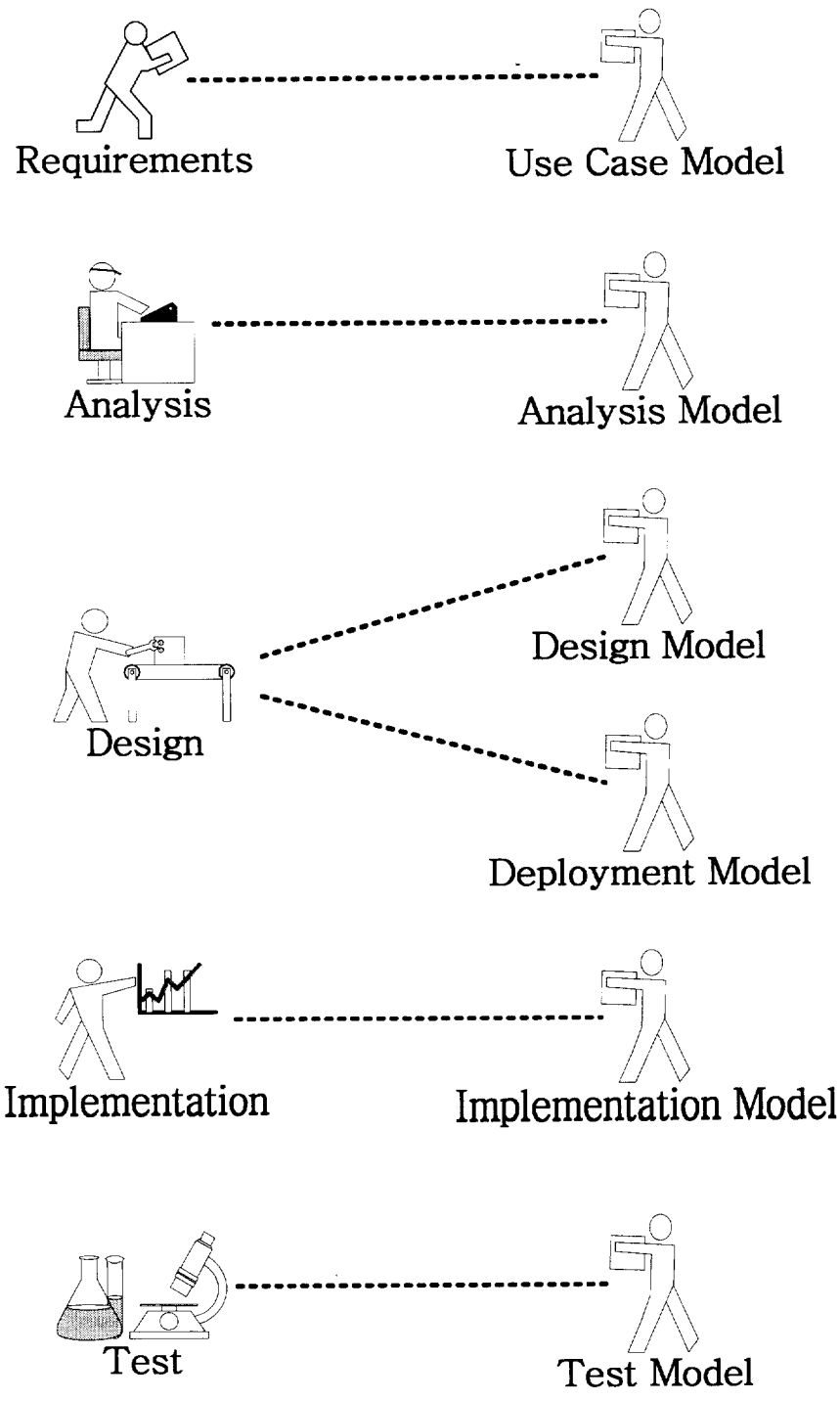
2.4 UML과 Rational Unified Process

소프트웨어 개발 방법론은 대개 원리적으로 모델링 언어와 공정으로 구성된다.[4] 모델링 언어는 그래픽 위주의 표기법이고 방법론은 이를 이용해서 설계를 한다. UML이 바로 모델링 언어에 해당하고 공정(process)은 방법론을 도와서 설계 수행 중에 취할 단계를 알려준다. 여기서는 Booch, Rambaugh, Jacobson이 만들어낸 통합 공정 RUP를 살펴보고자 한다.

[그림 2.7]에서 보듯이 Unified Process는 요구사항부터 테스트에 이르기까지 일련의 작업흐름이다. 작업흐름은 Use Case 모델로부터 테스트 모델에 이르기까지의 범위를 포함하는 모델을 개발한다.[18] 각 단계별 내용을 간단히 살펴보면 다음과 같다.

요구사항 파악(Requirement)은 'true requirements'를 발견하고, 사용자들, 고객들, 그리고 개발자들을 위해 알맞은 방법으로 표현하는데 그 목적이 있다.[15] 요구사항 기술서로부터 첫 번째 변형되어 만들어진 것이 요구사항 모델로써 Use Case Model, 인터페이스 기술서, 문제영역 모델로 구성된다. Use Case Model은 Actor와 Use Case를 사용한다. 이들 개념은 시스템의 외부에 존재하는 것(actors)과 시스템에 의해 수행되어질 것(Use Cases)을 정의하는 것을 돕는다.[16] 결과물로는 시스템에 요구하는 기능을 적어놓은 일반적인 문서가 있으며 UML에서 Use Case Diagram, 간단한 Class Diagram, Activity Diagram으로 표현되어진다.

분석(Analysis)은 실제 풀어야 할 문제에 대한 분석단계로 실세계의 존재물에 해당하는 모델들(클래스, 객체, 상호작용)에 관한 분석이 이루어진다. 분석 모델은 Use Case Model로부터 정형화된다. 각 Use Case는 오로지 3가지 타입(Behavior, information, Presentation)의 객체로 나누어진다.[16] UML에서 Class Diagram, Sequence Diagram, Collaboration Diagram, State Diagram, Activity Diagram이 결과물로 획득된다.



[그림 2.7] The Unified Process

설계(Design)는 분석단계의 결과물에 기술적인 부분을 첨가하여 확장하는 것이다. 기술적인 확장이란 어떻게 구현할 것인지에 초점을 두고 어떻게 동작하고 어떤 제약이 있어야 하는 지에 관하여 생각하는 것을 의미한다. 디자인 모델은 주요한 입력으로써 분석 모델을 사용하여 생성되지만, 객체 요청 브로커, GUI 구축 Kit, 데이터베이스 관리 시스템등과 같은 선택된 구현환경을 적용시킨다.[15] 결과물로는 UML에서 Class Diagram, Sequence Diagram, Collaboration Diagram, State Diagram, Activity Diagram, Component Diagram, Deployment Diagram이 있다.

구현(Implementation)은 실제로 소스 코드를 생성하는 단계이다. 다이어그램에서 특정 언어의 구문으로 옮겨 적는 과정 그리고 컴파일하고 링킹하고 다시 디버깅하는 작업이 포함되어 있다. 구현 모델은 ActiveX Component, JavaBeans, 뿐만 아니라 더 많은 여러 종류의 컴포넌트들과 같은 모든 실행가능한 것을 포함하는 컴포넌트로 구성된다.[15] 결과물로 어떤 다이어그램을 만드는 일은 드물지만 대신 설계 단계를 정정하는 것이 필요하다.

테스트(Test)는 시스템 개발의 마지막 단계로써 테스팅의 상태, 결과를 기술한다. 테스트의 목적은 코드에서의 에러를 발견하는 일이다. 여기서 에러의 발견은 프로그램에서의 실수가 아니고 성공이라고 보아도 좋다. 테스트 결과의 에러가 문서로 남게 되고 이것이 다음 버전에서 고쳐질 수 있기 때문이다.

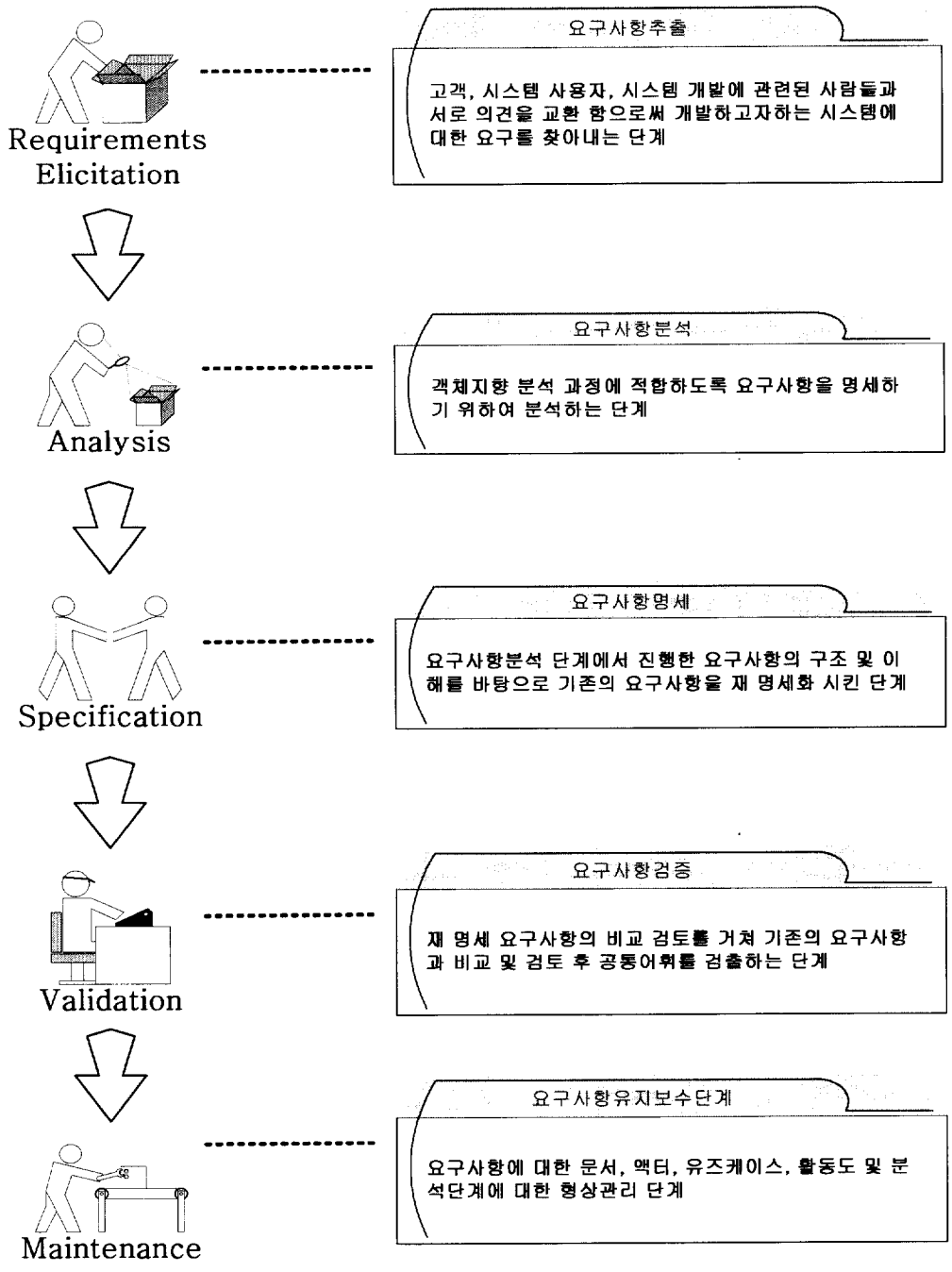
결국, RUP는 Use Case Driven, Architecture Centric, 반복 점증 개발(Iteration and Incremental)이라는 핵심 키 요소를 가진다. Use Case 중심이란 시스템 행위를 변경하고자 할 때, Actor와 Use Case를 적절하게 리모델링하며, 전체 시스템 아키텍처는 사용자가 시스템을 이용해서 실행하고자 하는 것에 의해 통제되어진다. 우리는 모든 모델을 통해 추적가능하고, 새로운 요구사항으로부터 시스템의 수정이 가능하다.[17] 다시 말하면, Use Case 중심의 의미는 시스템 개발에 있어 Use Case Model 이후의 모델들이 Use Case Model로부터 시작된다는 것이다. Use Case Model의 내용에 의해서 분석모델이 만들어지고, Use Case Model에 의해서 설계모델이 만들어지고, Use Case

에 의해서 구현 모델이 만들어지고, 또한 테스트 모델이 만들어지게 된다. 왜 Use Case로부터 모든 것이 파생되어야 하는가? 소프트웨어 시스템은 사용자에게 어떤 서비스를 제공하기 위해 존재한다. 바로 Use Case Model이 이러한 사용자의 요구사항을 반영하는 것이기 때문에, 시스템 구축까지 사용자의 요구사항을 반영하기 위해서는 Use Case 중심이 되어야 한다는 것은 당연한 것이 될 것이다.

2.5 UML과 요구 사항 파악

소프트웨어를 개발에 있어서 요구 사항 분석 단계가 갖는 가장 큰 목표는 포함될 기능, 제약요소, 기타 관련 사항들을 파악하여 분명하게 명세화하는 것이다. 그러나 소프트웨어가 사용되는 분야가 복잡해지고 대형화되면서 소프트웨어 개발에 있어서 사용자의 요구분석과 올바른 서술이 점차 복잡하고 어려워지게 되었으며, 그에 따라 요구사항에 관계되는 모든 활동에 대한 좀 더 체계적이고 공학적인 접근이 필요하게 되었다. 요구 사항 공학은 요구사항 분석 및 서술뿐만 아니라 이들의 추출, 관리, 검증, 유지 등을 포함하여 요구사항에 관계되는 모든 활동과 원칙들을 의미하며, 현재 요구공학에 관한 연구가 활발히 진행되고 있다. 요구공학을 좀 더 구체적으로 정의한다면 시스템 개발에 관련된 고객들의 요구사항에 대한 추출, 명세, 분석, 관리를 일관성 있게 수행하기 위한 체계적이고 반복적인 활동 및 프로세스의 집합이라고 볼 수 있다.

요구사항공정은 소프트웨어공학 단계에서 사용자가 원하는 시스템을 찾아내고, 설계에 들어가기 전에 문서화하는 모든 공정을 포함한다. 보통 요구공학에서 사용되는 공정은 [그림 2.8]과 같이 요구 사항 추출(Elicitation), 분석(Analysis), 명세(Specification), 검증(Validation), 유지보수(Maintenance)의 단계로 나눌 수 있으며, 본 절에서는 UML 기반 요구공학 프로세스로서 요구사항 추출, 요구사항 분석, 요구사항 명세, 요구사항 검증, 요구사항 변경관리의 다섯 단계로 나누어 각 프로세스가 갖는 특징을 살펴보기로 한다.


Requirements Elicitation
요구사항추출
고객, 시스템 사용자, 시스템 개발에 관련된 사람들과 서로 의견을 교환 함으로써 개발하고자하는 시스템에 대한 요구를 찾아내는 단계
Analysis
요구사항분석
객체지향 분석 과정에 적합하도록 요구사항을 명세하기 위하여 분석하는 단계
Specification
요구사항명세
요구사항분석 단계에서 진행한 요구사항의 구조 및 이해를 바탕으로 기존의 요구사항을 재 명세화 시킨 단계
Validation
요구사항검증
재 명세 요구사항의 비교 검토를 거쳐 기존의 요구사항과 비교 및 검토 후 공통어휘를 검출하는 단계
Maintenance
요구사항유지보수단계
요구사항에 대한 문서, 액터, 유즈케이스, 활동도 및 분석단계에 대한 형상관리 단계

[그림 2.8] 요구 공학 공정

요구사항 추출은 고객, 시스템 사용자, 그리고 시스템 개발에 관련된 사람들과 서로 의견을 교환함으로써 실제로 개발하고자 하는 시스템에 대한 요구들을 찾아내는 공정으로 시스템의 감추어진 정보를 관련된 모든 사람에게 분명하고 정확한 문장으로 얻어내는 과정이다. 요구사항 획득, 요구사항 식별 및 분류, 공통어휘 추출 단계를 거쳐 산출물을 작성하게 되는데 면담, 워크샵, CRC 카드, 프로토타이핑, 시뮬레이션, 시나리오 등을 사용하여 획득된 요구사항을 요구사항 명세 요건에 따라 분류한 요구사항 식별 모델을 통해 분류하여 작성하고, 이를 기반으로 요구사항을 추출한다. 그리고 그 다음으로 시스템 개발의 전 과정에서 사용될 요구사항의 공통 어휘를 추출하여 실세계의 객체와 업무와 관계된 객체를 선택하여 정의, 명세하고 요구사항 명세에 사용한다. 이에 따른 산출물로 요구사항 식별 모델에 의한 요구사항 명세 및 분류표 혹은 DB와 요구사항의 공통어휘 명세가 있다.

요구사항 분석은 객체지향 분석과정에 적합하도록 요구사항을 재명세하기 위하여 분석하는 단계로서, 사용자의 요구를 이해하고 문제 해결의 여러 제약조건들을 정리하여 시스템을 어떻게(How) 구현할 것인가 보다는 무슨(What) 시스템을 구현할 것인가를 분석하게 된다. 요구사항 추출단계를 통해 요구사항 획득 후 분류한 요구사항과 추출한 공통어휘를 기반으로 시스템의 영역을 정의하고, UML 도구인 Use Case와 Actor를 사용하여 요구사항을 구조화하고, Activity Diagram을 통해 시스템을 다시 이해한다. 즉, 영역분석, Actor 추출, Use Case 추출, Use Case 클러스터링, Use Case 우선순위 결정, 활동도 작성 과정을 통해 산출물을 획득하게 된다.

요구사항에서 시스템의 경계의 정의 및 제한, 시스템에 부과되는 제약사항을 식별함으로써 영역을 분석한 후, 요구사항에서 외부 인터페이스를 찾는다. 사용자 인터페이스, 하드웨어 인터페이스, 소프트웨어 인터페이스, 통신 인터페이스 등의 인터페이스는 일반적으로 Use Case Diagram에서 Actor로 존재하게 된다. 추출한 Actor가 시스템에 어떤 기능을 요구하는지를 중심으로 Use Case를 추출하고, 추출된 Use Case 간의 공통 범위와 기능 식별 및 관리를 위하여 서로 관련된 Use Case를 큰 기능군으로 클러스터링 한다. 그리고 관련된 Use Case 안에서 업무들 간의 중요도, 위험요소와 복잡성의 포함정도, 새롭거나 개발에 검증되어 있지 않은 기술의 적용 유무에 따라 요구사항 등급

을 부여한다. 마지막으로 추출된 Use Case 별로 활동도를 그려 목표 시스템의 작업흐름을 모형화하고 시스템을 보다 상세하게 이해한다. 산출물로는 후보 Actor와 Use Case에 대한 명세 또는 이에 대한 다이어그램, 활동도 명세 혹은 다이어그램이 있다.

요구사항 명세는 요구사항 분석단계에서 진행한 요구사항의 구조화와 획득한 재 이해를 기반으로 기존 요구사항을 재명세한다. 클러스터링한 Use Case 기능 군을 큰 단위로 하여, 요구사항 ID와 각 요구사항의 제목, 상태정보를 기술하고 각 요구사항을 재명세한 후 요구사항 간의 추적성을 설정한다. 이 단계에서는 재명세된 요구사항 명세 혹은 DB를 산출한다.

요구사항 검증은 재명세 요구사항과 공통어휘를 검증하게 된다. 재명세한 요구사항을 기존 요구사항과 비교, 검토하고 추출된 공통어휘를 검증한다.

요구사항 관리는 요구사항 문서와 후보 Actor, Use Case, 활동도 분석 결과 등의 형상에 대한 관리가 필수적으로 시행되어야 하고, 분석단계의 형상관리 계획도 수립되어야 한다. 이와 더불어 각 요구사항 명세가 후보 Use Case, Actor와 어떻게 연관, 추적되어지는지를 기술하여야 하며, 분석단계의 검증과 검정계획으로 요구공학 프로세스의 형상과 어떻게 연관할 것인지에 대한 계획을 수립하여야 한다. 또한 소프트웨어 산출물이 완전히 사용자의 요구사항에 부합하도록 검증과 검정 또는 다른 관리 활동을 진행함으로써 품질 보증이 이루어지도록 한다.

요구사항의 변경관리는 요구사항의 변경은 식별자인 요구사항 고유의 ID와 요구사항의 상태를 나타내는 상태정보로 관리되어지는데, 고객으로부터의 요구사항에 대한 변경요구가 발생하면 타형상의 영향분석 이후 요구사항 간의 연관관계와 타 형상과의 연계를 설정하기 위해 추적 테이블에 관련된 요구사항 간의 정보를 저장한다. 또한 이전 버전 혹은 기준선으로 형상을 복귀하기 위하여 변경자, 변경일자, 변경사유 등을 내력 테이블에 저장하며, 관련된 모든 참여자 혹은 내부적 인지 규칙에 의해 전달하여 신속한 대응과 검토가 가능하도록 한다.

제 3 장 Use Case Diagram 추출과정 및 규칙

3.1 요구사항 기술서

고객의 요구사항은 업무 시스템 개발에서부터 테스트 과정까지 영향을 미치게 된다. 따라서 프로젝트의 성공을 위해서는 체계적이고 명확한 요구분석이 이루어져야하며, UML에서는 Use Case Modeling을 통해 사용자 혹은 고객의 요구사항을 파악하고 업무 시스템의 범위를 결정하는 방법을 제공하고 있다. 요구사항 추출은 고객이 이해할 수 있는 용어로 쓰여진 요구사항 정의문서를 작성 할 수 있도록 하는데, 요구사항 정의서(Requirements definition)는 제안된 시스템에 대해서 고객이 기대하는 모든 기능에 대한 완전한 리스트이다. 이것이 고객이 원하는 것이 무엇인지 알 수 있도록 고객과 개발자간의 이해를 도와주고 보통 고객과 개발자가 함께 공동으로 작성한다. 반면에 요구사항 기술서(명세서)는 시스템 개발을 위해 적절한 기술 용어를 이용하여 요구사항 정의서를 재작성 한 것이다.[10] 구축할 업무시스템의 영역을 문서로써 기술한 것을 「요구사항 기술서(Requirements Specification)」라 하며, 작성된 요구사항 기술서는 Use Case Modeling 과정에서 가장 중요한 기초 자료가 된다.[12] 따라서 요구사항 기술서에는 구축되어야 할 업무 시스템의 영역이 명확히 정의되어야 하며, 그 내용 또한 사용자의 검증을 거친 것이어야 한다. 또한, 요구사항 기술서는 가능한 한 자세하게 작성하는 것이 좋으며, 업무 영역의 구체적인 부분까지도 기록하고 사용자로부터 확인받는 것이 바람직하다.

3.2 Use Case Model

Unified Object Modeling 접근법에서, Use Case Model을 구축하는 것이 초기 단계에 포함된다. 이 모델에서 중요한 것은 새롭게 혹은 기존 시스템을 기반으로 개발되거나 새로운 시스템에 대한 사용자의 요구사항을 파악하는 것이다.[3] 따라서 도메인 전문가의 도움으로 요구사항 기술서가 작성되면 시스템

을 개발하는 동안 시스템이 제공해야할 기능 또는 서비스가 Use Case Model에 기록된다. Use Case Modeling의 목적을 간단히 살펴보면 다음과 같다.

첫째, 시스템의 기능적 요구사항을 결정하고 설명하여 고객과 개발자간의 합의를 도출한다.

둘째, 시스템이 무엇을 할 것인가에 대한 명백하고 지속적인 설명을 통해 개발 과정에서 모든 개발자들이 요구사항에 대해 의사소통하고, 요구된 기능성을 제공해주는 상세한 설계를 위한 기반을 제공한다.

셋째, 시스템을 검증하기 위한 테스트를 수행하는 기초 역할을 담당한다.

넷째, 시스템 내에서 기능상의 요구사항이 실제 클래스와 동작을 하는지에 대한 추적 능력을 제공함으로써 Use Case Model을 수정함으로써 시스템을 간단하게 바꾸고 확장시킬 수 있으며, 그 후에 Use Case는 시스템 설계와 구현에 영향을 미친다.

이러한 목적을 가진 Use Case Model은 여러 사람들이 관심을 가진다. 먼저, 고객 입장에서는 Use Case Model이 시스템의 기능을 정의하는데 사용되므로 관심을 가지는데 특히, 고객이 Use Case Modeling 과정에서 실질적인 역할을 할 때 도움이 되고 모델이 고객의 희망에 의해 세부적으로 적용된다. 따라서 Use Case는 고객이나 사용자의 언어로 설명된다. 다음으로 개발자는 시스템이 무엇을 해야 되는지 이해하고, 미래의 작업(다른 모델과 구조설계, 구현)을 위한 기초로 이들을 사용하기 위해 Use Case Model에 관심을 갖는다. 그리고 시스템 통합과 테스트 팀에게는 시스템이 Use Case에 정의된 기능을 수행하는지 확실하게 테스트하기 위해서 Use Case Model이 필요하다. 마지막으로 마케팅, 영업, 지원, 보고서작업 팀 등 시스템의 기능에 연결된 어떤 사람들이라도 Use Case Model에 관심을 가질 수 있다.

Use Case Model을 만드는데 실제 필요한 작업은 시스템의 정의, Actor와 Use Case 찾기, Use Case 그리기, Use Case 간의 관계 정의, 그리고 최종적으로 모델을 검증하는 과정이다. 이는 고객과 Actor로 표현되는 사람들 간의 토론을 포함하여 매우 상호 교류적인 형태로서, Use Case Model은 Actor와 Use Case, 이들 간의 관계를 나타내는 Use Case Diagram으로 구성된다. 이들 다이어그램은 개괄적인 것을 설명하고, Use Case의 실제 설명은 텍스트로 이루어진다. 그러나 많은 사람들은 대부분의 저서에서 설명하는 설명을 가지

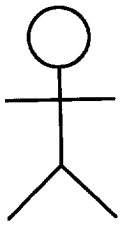
고는 Use Case Diagram을 작성하는데 어려움을 갖는다. 본 장에서는 Actor, Use Case 및 관계를 추출하기 편리하도록 요구사항 기술서의 전제조건을 두었으며, 변경된 요구사항 기술서로부터 Use Case Diagram의 요소를 추출하였다.

3.3 Actor

Actor는 사용자의 종류로서 시스템의 외부에 존재하면서 시스템과 직접 상호 작용하는 누구 혹은 어떤 것이다. '시스템과의 상호작용 한다'는 것은 Actor가 시스템으로부터 메시지를 받거나 시스템에게 메시지를 보내는, 혹은 시스템과 정보 교환하는 것을 말한다.[6] 뒤에 언급하겠지만 상호작용을 한다는 의미에는 'Use Case'를 수행한다는 의미를 포함하고 있으며 Actor는 보통 요구사항 기술서에 '명사형'으로 표현되어진다. 예를 들면 사람들, 다른 소프트웨어, 하드웨어 디바이스, 데이터 저장장치, 네트워크 등을 Actor로 볼 수 있다. Actor를 고려할 때 중요한 것은 사람이나 직위보다는 '역할'을 먼저 생각해야 한다는 것이다.[7] 여기서 '역할'은 사용자와 시스템 사이의 관계를 구성하게 되는데 특유의 필요, 흥미, 기대, 행위, 책임 같은 것의 집합에 의해 정의되며[8] 이것은 Actor와 사용자(users)를 구별해야 한다는 점과 의미를 같이한다. 사용자(users)는 시스템을 사용하는 실제 사람인데, Actor는 사용자 (user)가 수행하는 어떤 역할을 나타낸다.[9] 면담이나 요구사항 기술서등의 방법을 이용하여 사용자의 요구사항을 파악하여 Actor를 추출할 때 유용하게 쓸 수 있는 몇 가지 질문을 살펴보면, 어떤 요구내용을 누가 관심 있어 하는가? 이 작업이 누구를 유익하게 하는가? 시스템이 어디에서 사용될 것인가, 그리고 그 곳에 누가 있을 것인가? 누가 시스템에 정보를 넣을 것인가? 누가 이 정보를 사용할 것인가? 누가 이 정보를 편집하고 삭제할 것인가? 누가 시스템을 지원하고 관리하는가? 시스템이 외부 자원을 사용하는가? 여러 사람이 동일한 역할을 하는가? 한 사람이 여러 가지 역할을 하는가? 시스템이 기존 시스템과 상호작용 하는가?

여기까지 기술된 내용을 가지고 Actor를 추출하게 되는데 추출된 Actor 목

록 중에서 시스템의 주요 기능을 사용하는 Actor를 주 Actor, 주요 기능을 보조하는 Actor를 보조 Actor라 한다. 또, 시스템과 상호작용 하는 형태에 따라 시스템에 입력정보를 주는 능동 Actor, 시스템으로부터 정보만을 받는 수동 Actor, 시스템에 정보를 주고, 받는 Actor로 구분할 수 있다. UML에서 Actor는 [그림 3.1]과 같이 나타낸다.



[그림 3.1] Actor

3.3.1 요구사항 기술서의 적용규칙

요구사항 기술서로부터 Actor와 Use Case 및 관계를 추출하기 위해서는 요구사항 기술서의 각 문장을 분석하는 과정이 필요하다. 따라서 문장의 의미가 변하지 않는 범위 안에서 분석하기 편리하도록 문장의 형태를 변경하도록 한다. 요구사항 기술서의 관련 규칙을 보면 다음과 같다.

○ Rules for Requirements Specification : R_{RS}

[R_{RS1}] : 요구사항 기술서에서 수동태적인 문장은 능동태적인 문장으로 변형한다.

면담이나 요구사항 정의서를 통해 사용자의 요구사항을 파악한 후 개발자의 입장에서 요구사항 기술서를 작성하게 된다. 이때 시스템을 사용할 업무담당자가 및 사용자를 중심으로 실제 사용하고 있는 용어를 이용하여 기술하고, 수동태적인 문장 보다는 보통 평서문의 형태인 능동태적인 문장으로 변형하는 것이 합리적이다. 예를 들어 "~ 된다."는 "~ 한다."로, "B는 A에 의해 변경 된다."는 "A는 B를 변경한다."로 변형을 가한다.

[R_{RS2}] : 동작 주체인 주어가 생략된 부분이나 지시어로 기술된 부분은 의미를 파악하여 동작주체인 주어를 찾아서 기술한다.

역할(Actor)와 Use Case를 추출하기 위해서는 동작의 주체 및 동작행위를 아는 것이 관건이므로 주어가 생략된 경우 문장을 읽고 문장의 주어를 유추하여 기록한다.

[R_{RS3}] : 요구사항 기술서의 문장의 형태를 다음과 같이 변경한다.

문장형태 : 주어+목적어(직접목적어+간접목적어)+서술어

동작주체와 행위가 핵심 내용이므로, 동작주체는 주어로 문장의 앞부분에 기술하고 동작행위는 서술어로 문장의 뒤에 기술하는 형태로 변경한다.

[R_{RS4}] : 동일한 기능을 담당하는 역할의 명칭은 통일한다.

요구사항 기술서를 작성할 때 보통 개발하고자 하는 시스템과 관련된 여러 명의 담당자들과 면담을 하게 된다. 이때 각 담당자마다 같은 기능을 수행하는 것에 대한 명칭을 다르게 표현할 수 있다. 같은 기능을 수행하는 역할이므로 한 가지 명칭으로 통일해 준다.

3.3.2 후보 Actor 추출 규칙

다음에서 제시한 규칙을 이용하여 요구사항 기술서로부터 Actor 후보 목록을 작성하여 Actor 후보를 추출한다. Actor 후보를 추출하기 위한 규칙을 R_A (Rule for Actor)라 정의한다. Actor 후보 추출을 위한 규칙은 다음과 같다.

○ Rules for Actor Specification : R_A

[R_{A1}] : 각 문장에서 주어나 간접 목적어가 사람인 경우 후보 Actor로 본다.

시스템에 정보를 입력하거나, 시스템으로부터 정보를 얻을 수 있는 역할을 담당하는 시스템 혹은 사용자를 UML 언어에서 Actor로 본다. 따라서 어떤 행위의 주체를 지시하는 주어나 간접목적어에 해당하는 '사람'은 일단 Actor 후보로 본다.

[R_{A2}] : 한 문장에서 주어로서 사람은 아니지만 역할을 가지는 의도적 동작 주체(명사, 고유명사로 표현되는)를 Actor 후보로 본다.

[R_{A3}] : 주어가 동작의 상태를 나타내는 경우 목표를 가지고 있어 상태에 능동적인 역할을 할 수 있는 주체는 잠재적 Actor로 본다.

요구사항 기술서에는 직접 나타나지 않더라도 시스템의 상태에 변화를 줄 수 있는 Actor를 잠재적 Actor로 보고 Actor의 후보로 본다.

[R_{A4}] : 동작 주체의 행위에 의해 결과를 얻거나 이익을 얻는 대상을 후보 Actor로 본다.

동작을 수행하는 주체는 아니지만 다른 동작 주체의 행위에 의한 결과를 입력받거나, 그 결과로 자신의 상태에 변화를 가져올 수 있는 대상은 후보 Actor로 본다.

[R_{A5}] : 동작의 주체이지만 의도적이지 않은 즉, 우연한 것이나 의미 없는 동작을 수행하는 경우는 후보 Actor로 보지 않는다.

'의도적이지 않은'에는 목적을 가지지 않아도 수행되거나 목적과 관계없는 동작이 수행되었을 경우를 의미하므로 Actor 후보에서 제외시킨다.

[R_{A6}] : 동작의 주체 중 이벤트를 발생시키지 않는 주어는 Actor 후보에서 제외시킨다.

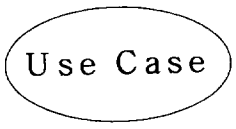
동작을 수행하는 주체이기는 하지만 동작 자체가 어떤 결과를 나타내는 것이 아니라 단순한 상태설명인 경우에 해당한다면 후보 Actor에서 제외시킨다.

3.4 Use Case

Actor는 업무 수행을 위해 어떤 목적을 가지고 시스템을 사용한다. 이러한 Actor의 목적을 위해 제공해 주어야 하는 시스템의 기능이 바로 Use Case이다. 따라서 Use Case는 특정 Actor에 대하여 관측 가능한 결과를 산출하는 일련의 이벤트 흐름으로 언제나 Actor에게 어떤 가치를 제공해야 한다. 요구사항과 Use Case 간의 관계를 보면 Use Case는 행위의 단위를 기술한 것, 요구사항은 행위에 적용된 법칙을 기술한 것, Use Case는 하나 혹은 그 이상의 기능적 요구사항을 만족할 수 있으며, 기능적 요구사항은 하나 혹은 그 이상의 Use Case에 의해 만족되어 질 수 있을 것이다. UML에서 Use Case는 항상 Actor에 의해 시작되며[19] Use Case를 초기화하는 Actor의 입장에서 보통 서술형으로 명명된다. 그러므로 요구사항 기술서에서 Actor와 시스템간의 상호작용을 찾아 서술어를 중심으로 하여 Use Case를 추출한다. 이와 같은 방법을 통해 시스템을 파악하는 방법은 최종적으로 시스템을 사용할 사람의 입장에서 시스템을 파악한다는 중요한 의미를 가진다. Use Case를 추출할 때

유용하게 사용 될 질문들을 살펴보면, 각 Actor의 업무(임무)는 무엇인가? 이 시스템에서 어떤 Actor라도 정보를 작성, 저장, 변경, 제거 또는 읽을 수 있는가? 어떤 Use Case가 정보를 생성, 저장, 변경, 제거 또는 읽을 수 있는가? 어떤 Actor가 외부변화에 대한 사항들을 시스템에 알릴 필요가 있는가? 시스템 내 특정 현상에 대해 어떤 Actor에 알릴 필요가 있는가? 어떤 Use Case가 시스템을 지원하고 유지할 것인가? 모든 기능적 요구사항이 Use Case에 의해 수행되어질 수 있는가?

UML에서 Use Case는 [그림 3.2]와 같이 타원으로 표기한다.



[그림 3.2] Use Case

(1) 후보 Use Case 추출 규칙

다음에서 제시한 규칙을 이용하여 3단계 과정을 거쳐 후보 Actor로부터 Use Case를 추출한다. 후보 Use Case를 추출하기 위한 규칙은 다음과 같다.

○ Rules for Use Case Specification : R_U

[R_{U1}] : Actor 후보와 관련된 문장에서 동사구를 분석하여 Actor 후보가 시스템에 대하여 수행할 수 있는 활동들의 순서를 찾는다. Actor 후보가 수행하는 이벤트에 따라 임시 Use Case를 추출한다.

[R_{U2}] : 후보 Actor의 활동에 따른 Use Case 중 시스템에 관한 활동이 아니라 단순히 동작의 내용이나 상태를 기술한 경우 Actor 후보와 임시 Use Case를 제거한다.

[R_{U3}] : [R_{U2}]에 의해 정리된 나머지 Actor와 그 Actor의 활동을 기술 한

후 각 활동을 요약하여 Use Case를 작성한다.

[R_{U4}] : 하나의 Actor 혹은 여러 Actor에서 같은 행위를 수행하는 Use Case가 중복되는 경우는 하나만 작성한다.

[R_{U5}] : 정의된 Use Case 중 동일 기능을 수행하는 Use Case가 구체화되어 있는 경우는 인터페이스를 고려하여 일반화 할 수 있다.

[R_{U6}] : 정의된 Use Case 중 하나의 Use Case가 세분화되어 있어 포괄적 의미의 Use Case과 세분화된 Use Case가 공존한다면 포괄적 의미를 가지는 Use Case로 통합할 수 있다.

[R_{U7}] : 정의된 Use Case 중 각 Use Case에 내용이 공통적으로 사용되는 부분이 있다면 공통된 내용만을 추출하여 별도의 Use Case를 작성한다.

[R_{U8}] : 정의된 Use Case 중 다른 Use Case를 바탕으로 새로운 요구사항을 추가하여 작성된 새로운 Use Case를 추출하여 작성한다.

[R_{U9}] : Use Case 중 공통된 기능과 속성을 가지지만 더 많은 기능을 포함하고 있는 Use Case를 추출하여 작성한다.

3.5 관계 (Relationship)

Actor와 Use Case를 추출한 후 Actor와 Use Case 간의 연관성을 기술한다. 그 연관성으로는 '연관관계', '확장관계', '포함관계', '일반화관계' 로 구분할 수 있으며 이들 관계는 'Actor와 Actor 간의 관계', 'Actor와 Use Case 간의 관계', 'Use Case와 Use Case 간의 관계'를 표현 할 때 사용된다.

3.5.1 Actor와 Use Case와의 관계

Actor와 Use Case와의 관계의 연관관계(Association)는 관련된 요소 간에 한 방향으로 또는 양방향으로 모두 진행 가능한 연관성을 나타낸다. UML에서 요소간의 연관관계는 실선으로 표현하며, Actor에서 Use Case로 또는 Use Case에서 Actor로의 경우처럼 단지 한 방향으로만 진행할 경우 상호작용의 초기화를 담당하는 요소에서 상대 요소로 화살표를 추가한다.

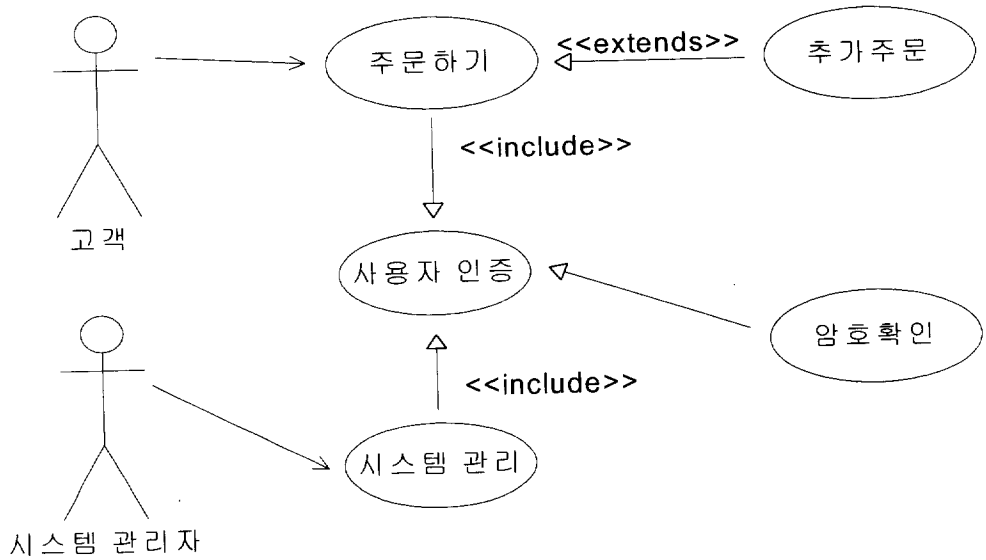
3.5.2 Use Case와 Use Case와의 관계

Use Case와 Use Case와의 관계는 포함관계(Include), 확장관계(Extension), 일반화관계(Generalization)로 나눌 수 있는데, 먼저 포함관계(Include)를 보면, 여러 개의 Use Case들이 동일한 기능을 공유하는 경우가 있다. 그 기능을 필요로 하는 Use Case마다 공통된 내용을 반복해서 기록하기보다는 공통 기능을 별도의 독립된 Use Case로 생성하고 이를 필요로 하는 Use Case들과 연관성을 형성하도록 한다. 다시 말하면, 다른 Use Case 행위를 명시적으로 수용하는 것으로 포함된 Use Case는 그것을 포함한 기본 Use Case의 일부로만 인스턴스를 만들 수 있고 독자적으로는 만들 수 없는 특징을 가진다. 결국, 공통의 기능을 재사용 하는 방법이며, UML 에서는 공통된 Use Case 방향으로 색칠 안 된 삼각형 화살표에 스테레오 타입을 이용하여 <<include>>로 표현한다.

확장관계(Extension)는 기존의 Use Case에 새로운 요구사항을 추가로 표현하고자 할 때 기존의 Use Case를 수정하는 것 보다 기존의 Use Case를 그대로 두고 새로운 요구사항을 별도의 Use Case로 작성하여 기존 Use Case를 참조하도록 하는 연관성을 '확장관계'로 본다. 다시 말해 기본 Use Case에 행동을 밀어 넣어 Use Case가 늘어난 것으로 어떤 특이한 조건에서만 수행되는 부 흐름을 별도로 모델링 할 수 있다. 포함관계에서는 공통 기능을 표현하는 Use Case가 없는 경우 포함관계 연관성을 가지는 다른 Use Case를 독립적으로 설명하기 어렵지만, 확장관계에서는 기존 Use Case만으로도 독립적으로 설명이 가능하다. 확장관계 역시 포함관계와 마찬가지로 색칠 안 된 삼각형

화살표에 스테레오 타입을 설정하여 <<Extend>>로 표현한다. 확장관계는 보통 선택적 행위나 특정 상황에서만 작동하는 행위, Actor의 선택을 바탕으로 작동하는 몇 가지 상이한 플로우들과 같은 상황에서 사용된다.

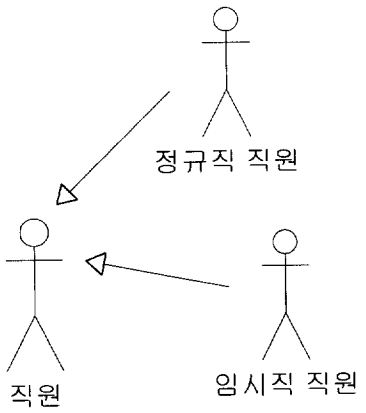
일반화관계(Generalization)는 하나의 Use Case가 다른 Use Case와 유사하나 그 이상의 더 많은 일을 수행 할 때 사용한다. 객체 지향 언어에서 흔히 볼 수 있는 상속(inheritance)의 의미와 동일한 것으로 일반적인(general)것과 이 일반적인 것에서 특수화된(specific)것 사이의 관계를 나타낼 때 사용한다. 구체화된 Use Case(child)는 일반화된 Use Case(parent)로부터 기능과 의미를 상속받고, 자신만의 기능을 추가·재정의 하여 완전한 Use Case를 생성한다. UML에서 일반화 관계는 구체화된 Use Case에서 일반화된 Use Case 방향으로 색칠 안 된 삼각형 화살표를 이용하여 표현한다. 다음의 [그림 3.3]은 Actor와 Use Case 또는 Use Case와 Use Case 간에서 나타날 수 있는 연관관계, 포함관계, 확장관계, 일반화 관계를 표현하였다.



[그림 3.3] 연관, 포함, 확장, 일반화 관계 표현

3.5.3 Actor들 간의 관계

Actor들 간의 관계인 일반화관계(Generalization)는 객체지향의 상속의 의미와 같으며 일반화된 Actor의 모든 특성을 특수한 Actor가 모두 가지게 된다. [그림 3.4]에서 보는 것처럼 Actor 간의 일반화 관계는 Use Case 간의 일반화와 마찬가지로 구체화된 Actor에서 일반화된 Actor 방향으로 색칠 안 된 삼각형을 이용하여 표현된다.



[그림 3.4] Actor 간의 일반화 관계

(1) 관련성 추출 규칙

다음에서 제시한 규칙을 이용하여 최종적으로 추출된 Actor와 Use Case 간의 관계를 설정한다. 관련성 추출을 위한 관련성 추출 규칙은 다음과 같다.

[R_{R1}] : Actor 후보 중 공통 속성과 기능을 갖는 경우 공통 속성과 기능만을 갖는 Actor와 그 공통 속성에 새로운 속성을 재 정의한 나머지 Actor들로 구분한다. 이때, 공통 속성과 기능만을 갖는 Actor를 상위레벨에 놓고 나머지 Actor들은 하위 Actor로 보고 하위 Actor에서 상위 Actor로 일반화 관계로 정의한다.

[R_{R2}] : Use Case와 Actor와의 관계를 분석한다. 이때, Use Case를 초기화

하는 Actor 쪽에서 Use Case 방향으로 열린 화살표를 그리고, Actor가 Use Case에 의해 결과를 받는다면 Use Case에서 Actor 방향으로 열린 화살표를 그린다.

[R_{R3}] : [R_{U3}] ~ [R_{U6}]의 내용은 이벤트 흐름이 두 개 이상의 Use Case에 걸쳐 진행되는 경우에 해당되므로 Use Case와 Use Case 간의 관계로서 표현한다.

[R_{R4}] : [R_{U7}]에 의해 추출된 내용대로 Use Case를 작성하고 기존 Use Case 방향에서 공통 Use Case 방향으로 포함관계를 정의한다.

[R_{R5}] : [R_{U8}]에 의해 추출된 내용대로 Use Case를 작성하고 기존 Use Case 방향에서 기존 Use Case에 요구사항이 추가된 Use Case 방향으로 확장관계를 정의한다.

[R_{R6}] : [R_{U9}]에 의해 추출된 내용대로 Use Case를 작성하고 기존 Use Case 방향에서 더 많은 기능을 포함하는 Use Case 방향으로 일반화 관계를 정의한다.

3.6 유스 케이스 다이어그램(Use Case Diagram)

Use Case Diagram은 시스템에서 찾아낸 Actor, Use Case, 그들의 상호 동작 등의 전체 혹은 일부분을 시각적으로 볼 수 있게 한다. 각 시스템은 일반적으로 메인 Use Case Diagram을 갖고 있고, 그것은 시스템 경계(Actor)와 시스템에서 제공하는 주요기능(Use Case)들의 그림이다. 필요에 따라 다른 Use Case Diagram들도 만들어 질 수 있다.

제 4 장 적용사례

지금까지 요구사항 기술서를 통해 Actor 추출, Use Case 추출, 관련성을 통해 학사 정보 시스템의 구축 사례에 적용하여 Use Case Diagram을 작성하도록 한다.

4.1 요구사항 기술서 관련 규칙 적용

요구사항을 파악하기 위해 "학사정보관리"의 요구사항 기술서의 각 문장에 "문장번호"를 부여하고 문장번호 순으로 R_{RS} 규칙 적용 여부를 분석한다.

○ 학사 정보 관리

학사정보관리는 학사관리, 수업관리, 수강관리, 사용자 관리 등의 업무 자동화를 목표로 한다.

사용자관리는 학생, 교수, 수업담당자, 학사 담당자가 학사정보관리의 사용자로서 시스템에 로그인하고 자신의 암호를 변경하고, 시스템의 사용을 완료한 후에 로그아웃을 하는 기능을 포함한다. 시스템 사용자(학생, 교수, 수업담당자, 학사 담당자)가 학사정보관리에 접속하면 로그인 화면이 표시된다. 아이디와 암호를 입력하여 로그인에 성공하면, 각 사용자 별로 자신이 이용할 수 있는 기능이 표시된다. 로그인에 성공하면, 각 사용자 유형별로 자신이 이용할 수 있는 기능이 표시된 메뉴 화면을 보여준다.

학사관리는 학사 담당자가 학생 정보 및 교수 정보를 등록, 조회, 검색, 수정, 삭제할 수 있는 관리 기능을 포함한다. 학사 관리 기능은 학사 담당자가 대학에 새 학생을 등록하거나, 기존 학생 정보를 조회하거나, 수정하거나, 삭

제하는 기능을 포함한다.

수업관리는 수업담당자가 이용하는 기능으로서 강좌 및 강의 정보를 관리하는 기능을 포함한다. 대학 정보 시스템에서는 교수가 특정학기에 실제로 담당하는 수업을 강의라고 하고, 이름, 번호 학점 등과 같이 개설된 강의에 공통적인 정보는 강좌로 구분한다. 즉 특정 학기에 실제로 개설되는 강좌가 강의다. 강좌 관리 기능은 수업 담당자가 새 강좌를 등록, 기존 강좌 정보를 조회, 수정, 삭제하는 기능을 포함한다.

수강관리는 수강 신청, 성적 등록, 성적 조회, 출석부 조회 기능으로 구성된다. 이 기능 중에서 수강 신청과 성적 조회는 학생이 이용할 수 있는 기능이며, 성적 등록과 출석부 조회는 교수에게 제공되는 기능이다. 학생은 수강 신청 기능을 이용하여 개설된 강의에 대한 수강 신청을 할 수 있고, 현재 수강신청된 강의를 조회하고, 수강신청을 취소할 수 도 있다. 수강 신청 등록 화면은 수강 신청 메인 화면에서 강의 신청을 누르면 전환되는 화면으로, 개설된 강의를 검색하고 강의 이름을 선택하여, 해당 강의에 대한 수강을 신청할 수 있다. 교수는 성적등록 기능을 이용하여 자신이 담당한 강의를 수강한 학생들의 성적을 입력하고 수정할 수 있다.

4.1.1 요구사항 기술서 정규화

(1) 학사정보관리 시스템은 학사관리, 수업관리, 수강관리, 사용자 관리 등의 업무 자동화를 목표로 한다.

(2) 사용자관리는 학생, 교수, 수업담당자, 학사 담당자가 대학 정보 시스템의 사용자로서 시스템에 로그인하고 자신의 암호를 변경하고, 시스템의 사용을 완료한 후에 로그아웃을 하는 기능을 포함한다.

(3) 시스템 사용자(학생, 교수, 수업담당자, 학사 담당자)가 대학 정보 시스템에 접속하면 로그인 화면이 표시된다. 아이디와 암호를 입력하여 로그인에 성공하면, 각 사용자 별로 자신이 이용할 수 있는 기능이 표시된다.

(4) 학사 관리는 학사 담당자가 학생 정보 및 교수 정보를 등록, 조회, 검색, 수정, 삭제 할 수 있는 관리 기능을 포함한다.

(5) 학사 관리 기능은 학사 담당자가 대학에 새 학생을 등록하거나, 기존 학생 정보를 조회하거나, 수정하거나, 삭제하는 기능을 포함한다.

(6) 수업관리는 수업담당자가 이용하는 기능으로서 강좌 및 강의 정보를 관리하는 기능을 포함한다. 대학 정보 시스템에서는 교수가 특정학기에 실제로 담당하는 수업을 강의라고 하고, 이름, 번호 학점 등과 같이 개설된 강의에 공통적인 정보는 강좌로 구분한다. 즉 특정 학기에 실제로 개설되는 강좌가 강의다.

(7) 강좌 관리 기능은 수업 담당자가 새 강좌를 등록, 기존 강좌 정보를 조회, 수정, 삭제하는 기능을 포함한다.

(8) 수강관리는 수강 신청, 성적 등록, 성적 조회, 출석부 조회 기능으로 구성된다. 이 기능 중에서 수강 신청과 성적 조회는 학생이 이용할 수 있는 기능이며, 성적 등록과 출석부 조회는 교수에게 제공되는 기능이다.

(9) 학생은 수강 신청 기능을 이용하여 개설된 강의에 대한 수강 신청을 할 수 있고, 현재 수강 신청된 강의를 조회하고, 수강신청을 취소할 수 도 있다.

(10) 수강 신청 등록 화면은 수강 신청 메인 화면에서 강의 신청을 누르면 전환되는 화면으로, 개설된 강의를 검색하고 강의 이름을 선택하여, 해당 강의에 대한 수강을 신청할 수 있다.

(11) 교수는 성적등록 기능을 이용하여 자신이 담당한 강의를 수강한 학생들의 성적을 입력하고 수정할 수 있다.

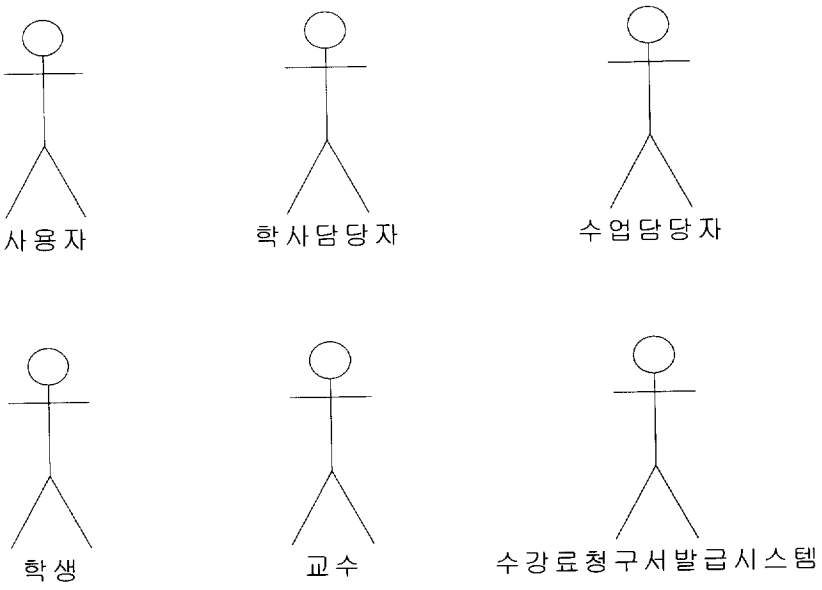
4.1.2 후보 Actor 제안

위에 제안된 요구사항 기술서를 바탕으로 Actor 추출 규칙(R_{A1}~R_{A6})을 적용하여 [표 4.1]과 같은 후보 Actor를 추출할 수 있는데, 후보 Actor의 오른쪽에는 후보 Actor의 역할들이 기술되어 있다.

[표 4.1] 후보 Actor 목록 작성 결과

후보 Actor	역할사항
사용자	사용자 Actor는 개발될 시스템의 기능을 이용하는 사용자를 의미한다. 예를 들면, 대학 정보 시스템의 경우, 이 시스템이 개발되었을 때 이용하는 사용자로는 교수, 학생, 학사 담당자, 수업담당자가 있다.
교 수	성적등록 기능을 이용하여 자신이 담당한 강의를 수강한 학생들의 성적을 입력하고 수정할 수 있다.
학 생	수강 신청 기능을 이용하여 개설된 강의에 대한 수강 신청을 할 수 있고, 현재 수강 신청된 강의를 조회하고, 수강 신청을 취소할 수도 있다.
학사담당자	학생 정보 및 교수 정보를 등록, 조회, 검색, 수정, 삭제할 수 있는 관리 기능을 포함한다.
수업담당자	수강 신청, 성적 등록, 성적 조회, 출석부 조회 기능으로 구성된다.
수강료청구서발급시스템 (System Actor)	개발될 시스템과 연동 되는 다른 시스템에 해당한다.

[그림 4.1]은 [표 4.1]의 후보 Actor 목록 작성 결과에 대한 Actor들을 UML 표기법으로 나타낸 것이다.



[그림 4.1] Actor 추출 결과

4.1.3 Use Case 추출

Actor 추출 규칙[R_{A1}~R_{A6}]을 적용한 결과 [표 4.1]과 [그림 4.1] 같이 Actor들을 찾을 수 있다. [표 4.1]에서 나타난 Actor들을 대상으로 Use Case를 생성(R_{U1}~R_{U9})한다.

[표 4.2] Use Case 후보 목록 작성 결과

후보 Actor	활 동	Use Case
사용자	사용자로는 교수, 학생, 학사 담당자, 수업담당자가 있다.	로그인 로그아웃 암호변경
교 수	자신이 담당한 강의를 수강한 학생들의 성적을 입력하고 수정할 수 있다.	출석부조회 성적등록
학 생	개설된 강의에 대한 수강 신청, 조회, 취소할 수도 있다.	성적조회 수강신청
학사담당자	학생 정보 및 교수 정보를 등록, 조회, 검색, 수정, 삭제 할 수 있다.	교수관리 학생관리
수업담당자	수강 신청, 성적 등록, 성적 조회, 출석부 조회 기능으로 구성된다.	강좌관리 강의관리
수강료청구서발급시스템 (System Actor)	개발될 시스템과 연동 되는 다른 시스템에 해당한다.	수강료청구서 발급

[표 4.2]는 후보 Actor와 그 Actor들의 역할 사항에 대한 내용과 Use Case들을 보여주고 있다.

[그림 4.2] Use Case 추출 결과

[그림 4.2]는 [표 4.2]에서 나타난 후보 Actor와 그 Actor들의 역할 사항에 대한 내용과 Use Case들을 표기형식으로 보여주고 있다.

4.1.4 관계 추출

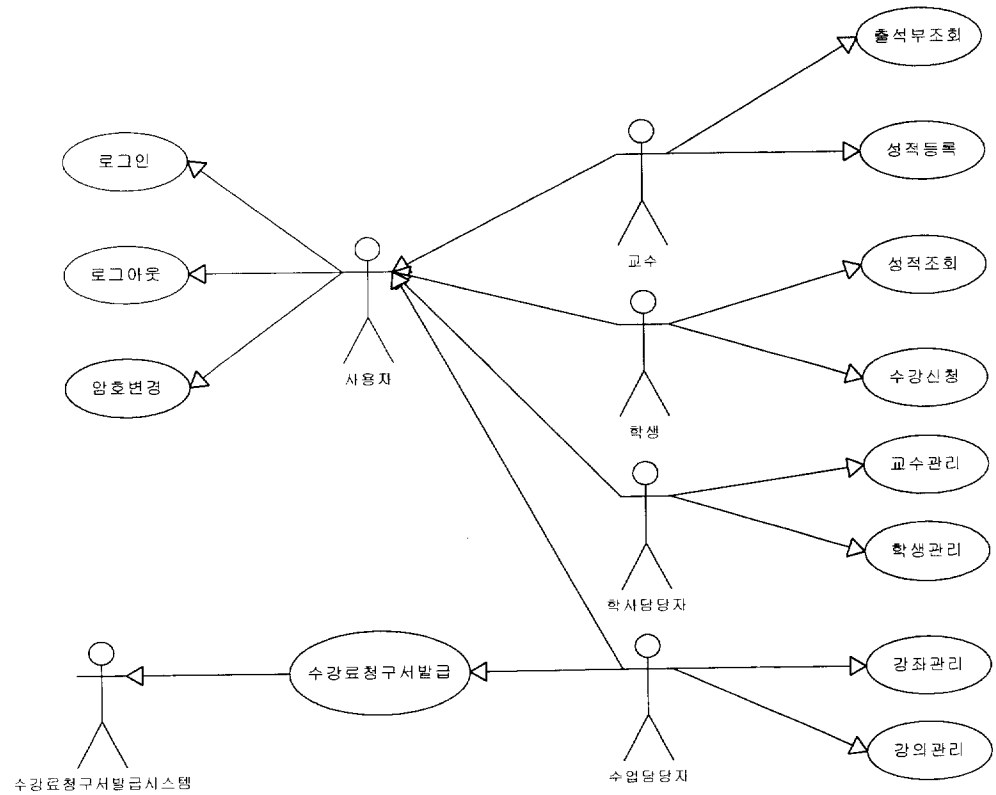
Actor와 Use Case를 추출한 뒤 Actor 간, Actor와 Use Case 간 또는 Use Case와 Use Case 간의 관계를 추출하여 Diagram을 완성하게 된다. 추출된 Actor와 Use Case에 규칙 R_{R1}~R_{R6}을 적용하여 [표 4.3]과 같은 결과를 나타낼 수 있다. 사용자는 로그인, 로그아웃, 암호변경 기능이 있고, 교수는 출석부조회, 성적등록을 할 수 있으며, 학생은 성적조회, 수강신청을 할 수 있고, 학사담당자는 교수관리, 학생관리 기능이 있고, 수업담당자는 강좌관리, 강의관리 기능이 있고, 수강료청구서발급시스템은 수강료청구서를 발급할 수 있다.

[표 4.3] 관계 추출 작성 양식

추출된 Actor	Use Case
사용자	로그인, 로그아웃, 암호변경
교 수	출석부조회, 성적등록
학 생	성적조회, 수강신청
학사담당자	교수관리, 학생관리
수업담당자	강좌관리, 강의관리
수강료청구서발급시스템 (System Actor)	수강료청구서발급

4.2 Use Case Diagram

Actor 추출, Use Case 추출, 관련성 추출을 거친 최종적인 Actor와 Use Case, 관련성이 정해지고, Actor와 Use Case에 관련성을 정리하면 [그림 4.3]과 같은 Use Case Diagram을 작성할 수 있다. 사용자는 로그인과 로그아웃, 암호변경을 할 수 있고, 사용자 그룹으로는 교수와 학생, 학사담당자, 수업담당자가 있다. 교수는 출석부조회와 성적등록을 할 수 있으며, 학생은 성적조회와 수강신청을 할 수 있다. 학사담당자는 교수관리, 학생관리 기능이 있으며, 수업담당자는 강좌관리, 강의관리를 할 수 있다.



[그림 4.3] Use Case Diagram

제 5 장 결 론

Use Case Model은 복잡한 표기법 없이 Actor와 Use Case 간의 관계를 보여줌으로써 Use Case Diagram을 산출하여 간결하게 시스템의 기능을 파악할 수 있도록 지원한다. 그러나 UML의 분석 모델이나 설계모델에 비하여 정형화 정도가 낮고 정해진 규칙의 부재로 모델링이 모호하다는 단점을 가지고 있다.

따라서 인터뷰 결과 또는 이미 작성된 요구사항 기술서의 내용에 대해서 몇몇 유용한 질문들을 사용하여 Actor와 Use Case 및 관계를 추출하는데 필요한 규칙을 제안하고, 이 규칙을 적용하여 Use Case Diagram의 추출 과정을 단계별로 제시하였다. 그리고 Use Case Diagram의 추출과정의 단계별 결과를 각 요소들의 선택 여부 및 다음 단계의 기초를 제공하였다.

본 논문에서는 Use Case Diagram을 추출하는 과정을 단계별로 제시하여 각 단계의 핵심행위와 단계별 연관성을 알 수 있게 하였고, 초보자들도 각 과정에서 제안한 규칙에 따라 순차적으로 Use Case Diagram을 작성할 수 있도록 하였다. 그리고 요구사항 기술서로부터 Actor, Use Case, 관계를 추출함에 있어서 문장을 분석 단위로 취급하여 각 문장마다 요구사항 기술서 관련규칙을 적용함으로써 Actor와 Use Case를 쉽게 추출할 수 있고, Actor 추출을 적용한 결과를 정리하고 각 후보 Actor별 항목을 통해 객관적으로 Actor를 추출할 수 있다. 또한 관계 추출 과정을 통해 Actor 추출 단계와 Use Case 추출 단계의 결과를 이용하여 관련성을 쉽게 추출할 수 있다.

향후과제로 Actor를 추출함에 있어서 역할의 범위 설정과 기준 및 Use Case 추출에 대한 세분화와 정밀도와 시나리오에 대한 Use Case Diagram의 패키지 관련성에 대한 Activity Diagram의 작성 기법 및 연관성 등에 대한 연구가 필요하다.

참 고 문 헌

[1] Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley Pub. Co., 1999.

[2] Kendall Scott, *UML Explained*, Addison Wesley Pub. Co., 2001.

[3] Ivar Jacobson, *Object-Oriected Software Enginnering*, Addison Wesley Pub. Co., 1992.

[4] Hans-Erik Eriksson, Magnus Penker, *UML Toolkit*, Wiley Computer Publishing, 1998.

[5] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar vergaard, *Object-Oriented Software Engineering - A Use Case Driven Approach*, Addison-Wesley, 1998.

[6] Doug Rosenberg, Kendall Scott, *Applying Use Case Driven Object Modeling with UML*, Addison Wesley Pub. Co., 2001.

[7] Martin Fowler, Kendall Scott, *UML Distilled Second Edition : A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, 2000.

[8] Wirfs-Book Rebecca, WilkerSon Brain, Wiener Lauren, *Desibing Object-Oriented Software,* Prentice Hall, 1990.

[10] Joseph Schmuller, *Teach Yourself UML in 24 Hours*, SAMS Pub. Co., 1999.

[12] Alistair Cockburn, *Writing Effective Use Cases*, Addison-Wesley Pub.

Co., 2000.

[13] Kurt Bittner, Ian Spence, *Use Case Modeling*, Addison-Wesley Pub. Co., 2003.

[14] Craig Larman, *Applying UML and Patterns*, Prentice Hall PTR, Pub. Co., 2003.

[15] Ivar Jacobson, Grady Booch, James Rumgaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[16] Doug Rosenberg, Kendall Scott, *Use Case Driven Object Modeling with UML : A Practical Approach*, Addison-Wesley, 1999.

[17] Geri Schneider, Jason P.Winters, *Applying Use Cases : A Practical Guide*, Addison-Wesley, 1998.

[18] Leon Starr, 김인기 역, *UML 설계와 응용*, 정보문화사, 2003.

[19] Stephen R. Schach, 유해영 역, *UML과 C++ 중심의 구조적 객체 지향 소프트웨어공학*, 이한출판사, 2001.

[20] 장옥배, 유철중, 이병걸, 김지홍, 양해술, 김병기, *소프트웨어공학*, 도서출판 한산, 2001.

[21] (주)한국정보감라단, 이우용, 고영국, 박태희, 김준수, *UML과 객체 지향 시스템 분석 설계*, 도서출판 그린, 2003.

[22] 류형규, 이순천, 류시원, 신성호, *UML 기반 객체지향 클라이언트 / 서버 구축*, 홍릉과학출판사, 2000.

[23] 정문재, 소프트웨어공학, 도서출판 상조사, 2000.

[24] 김민선, 김수동, "전사적 프로젝트 Use Case 모델링을 위한 실무 지침,"

한국정보처리학회 추계학술대회, VOL.6, NO.2, pp.1~4, 1999.

[25] 박화규, 백종명, "UML 기반의 ERP 개발방법론," 한국정보처리학회, VOL.6, NO.5, pp.27~37, 1999.9.

[26] 허진선, 최시원, 김수동, "Use-Case Based OO Project Scheduling," 한국정보과학회, VOL.30, NO.4, pp.293~30, 2003.4.

[27] 김재성, "컴포넌트 개발과정에서 UML 표기법과 컴포넌트 모델링," 한국정보처리학회논문지 D , VOL.8-D, NO.6, pp.747~752, 2001.12.

[28] 김은미, "객체 지향 설계 명세서에 대한 설계 검증 방법," 한국정보처리학회논문지 , VOL.6, NO.6, pp.1520~1531, 1999.6.

[29] 권순각, 김태석, "UML을 이용한 소프트웨어 개발," 한국멀티미디어학회, VOL.5, NO.4, pp.80~86, 2000.12.

[30] 정유진, 장종표, 김병기, "객체 지향 요구공학 프로세스," 한국정보처리학회 춘계학술대회논문집 , VOL.6, NO.1, pp.492~495, 1999.

[31] 강문설, 김태희, "객체 지향 소프트웨어 개발 방법론의 표준화 : UML(Unified Modeling Language)," 한국정보처리학회논문지 , VOL.5, NO.5, pp.64~73, 1998.9.

[32] 염근혁, "객체 지향 소프트웨어 개발에서 요구사항에 대한 검증방법," 한국정보과학회논문지, VOL.25, NO.5, pp.802~811, 1998.5.

[33] 장안나, 염근혁, "객체 지향 소프트웨어의 신뢰성 향상을 위한 요구 검증," 한국정보과학회논문지 , VOL.11, NO.4, pp.23~34, 1998.12.

[34] 안성욱, 이남용, 류성열, "UML의 "4+1" 뷰를 이용한 지식 관리 시스템의 개념적 모델," 한국전자거래(CALS/EC)학회지, VOL.5, NO.1, pp.123~134, 2000.6.