Introduction to OOAD using UML tools

- OBJECT-ORIENTED ANALYSIS AND DESIGN
- ITERATIVE DEVELOPMENT AND THE UNIFIED PROCESS
- CASE STUDY: THE NEXTGEN POS SYSTEM
- INCEPTION(개념화)
- 사용사례 모델: 요구사항을 문맥으로 쓰기
- IDENTIFYING OTHER REQUIMENTS
- FROM INCEPTION TO ELEBORATION
- Importance of UML & Diagram Components

Class B
TEAM 1

OBJECT-ORIENTED ANALYSIS AND DESIGN

●분석과 설계를 비교하고, 객체지향 분석과 설계를 정의

-OOAD에 UML과 패턴을 적용하기

객체설계를 잘 한다는 것은 어떤 의미일까?
Java, C++, Smalltalk, 그리고 C# 등의 객체 기술과 언어들을 사용하여 잘 설계되고, 견고하며 그리고 유지보수가 가능한 소프트웨어를 만드는 것이 가장 큰 목표가 된다.

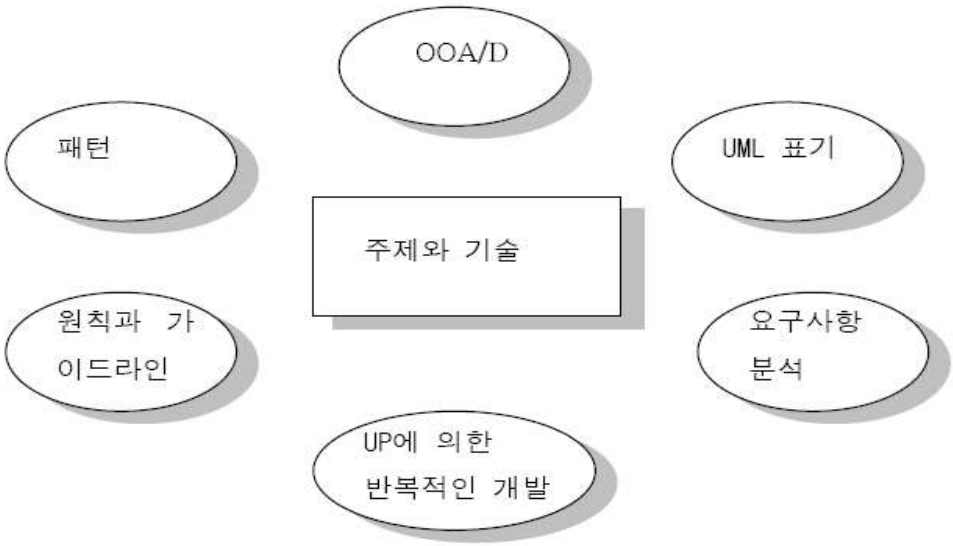
미국속담 "망치를 가지고 있다고 해서 목수가 되는 것은 아니다."는 객체기술 측면에서 보면 적절한 비유가 된다. 객체 시스템을 만들기 위하여 객체지향 언어를 습득하는 것은 필요한 첫 발자국임에는 틀림없지만 그것으로는 매우 불충분하다. 어떻게 "객체적으로 생각"할 것인가 가 가장 중요하다.

UML은 표준화된 다이어그램 표기법에 해당한다. 표기법을 배우는 것이 매우 유용하지만 이에 더하여 꼭 알아야 할 객체 지향 개념-어떻게 객체적으로 생각할 것인가와 어떻게 객체 지향적 시스템을 설계할 것인가 를 익히는 것이 더 중요하다.
즉, UML은 OOAD도 방법론도 아니고 단지 표기법일 뿐이다.
문법적으로 정확하게 UML 다이어그램을 그리는 것이나 UML CASE 도구를 사용하는 것은 별로 도움이 안 된다. 그보다는 훌륭한 설계를 고안하고 계속적으로 개선시킬 수 있는 능력이 더 중요하고 가치있고 어려운 기술이다.

그러나 OOA/D와 "소프트웨어 설계도"를 위한 언어가 필요한데 이는 생각을 표현하고 다른 사람들과 의사소통을 하는 데에 사용되어질 수 있다. 그러므로 OOAD를 활용하기 위하여 UML을 어떻게 적용할 것인가가 중요하며, 자주 사용되는 UML 표기법을 이용한다. 어떻게 책임을 객체 클래스에 할당해야 하는가? 어떻게 객체들이 상호작용 해야 하는가? 어떤 클래스들이 무엇을 해야 하는가? 이는 시스템을 설계하는데 있어 매우 중요한 질문들이다. 시스템 설계문제의 가장 좋은 해결방법은 좋은 연습을 해보는 것, 시행착오를 반복하며 발견적인 해결방법을 찾는 휴리스틱스 방법을 사용하는 것, 그리고 패턴을 사용하는 것 등을 들 수 있다. 패턴은 디자인 원칙을 요약해놓은 문제-해결 공식을 말하며 이 각 패턴은 이름을 갖는다.

또한 OOAD(또는 모든 소프트웨어 설계들)를 하는데 있어서 요구사항 분석이 반드시 선행되어져야 한다. 이 요구사항 분석에는 사용사례를 작성하는 것이 포함된다.

결국, 더 좋은 객체 설계를 위해 원리와 패턴을 적용하고, Unified Process를 기반으로 하여 분석과 설계에서 공통적인 활동을 수행하며, UML 표기법을 사용하여 자주 사용되는 다이어그램을 생성한다.



-책임 할당

OOAD에는 많은 활동과 산출물들이 가능하고, 또한 수많은 원리와 가이드라인들도 포함된다. 여기에 제시되는 많은 주제 중에서 단 하나의 실제적인 기술을 뽑아내라면 그것은 무엇이 될까?

OOA/D에서 가장 중요하고, 기본적인 능력은 소프트웨어 컴포넌트에 책임을 제대로 할당하는 것이다.

"책임 할당"하는 일은 UML 다이어그램을 그리거나 프로그램을 하거나 간에 반드시 수행되어야 하고, 이 활동은 소프트웨어 컴포넌트의 견고성, 유지보수성, 그리고 재사용성에 막대한 영향을 미치기 때문이다.
실제 프로젝트에서, 개발자가 시간이 부족하여 분석, 설계의 활동 없이 곧바로 구현으로 들어가는 경우에서 조차도 이 책임 할당을 하는 활동은 필요하다.

객체 설계와 책임 할당에 대한 9개의 기본 원리가 제시되고 적용된다. 이들은 GRASP 패턴 이라고 불리 운다.

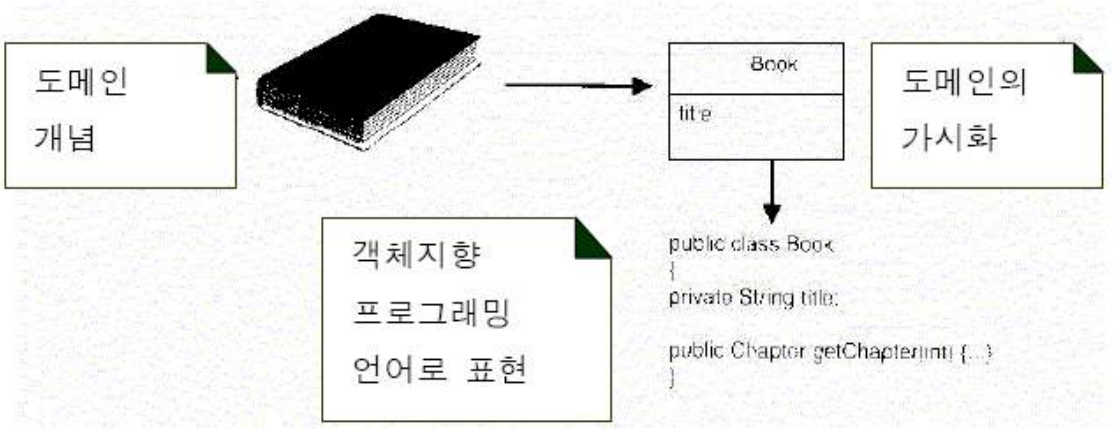
-분석과 설계란 무엇인가?

분석(Analysis)은 문제와 요구사항에 대한 해결책 보다는 문제와 요구사항에 대한 조사를 더 강조한다.
예를 들어 새로운 전산화된 도서 정보 시스템을 원할 때, 어떻게 이것이 사용될 것인가?
"분석"은 광범위한 용어로 좀더 정확하게는, 요구사항 분석(요구사항에 대한 조사)혹은 객체분석(도메인 객체에 대한 조사)으로 표현되어질 수 있다.
설계(Design)는 요구사항에 대한 구현 보다는 요구사항을 만족시키기 위한 개념적인 해결책을 더 강조한다. 예를 들면 데이터베이스 스키마와 소프트웨어 객체 등을 기술하는 활동이다. 궁극적으로 설계는 구현되어질 수 있다.
분석과 마찬가지로, 설계는 좀 더 정확하게 객체 설계 혹은 데이터베이스 설계로 표현되어 질 수 있다.
분석은 적절한 일을 하는 단계로, 설계는 일을 제대로 하는 단계로 요약 되어진다.

-객체 지향 분석과 설계는 무엇인가?

객체 지향 분석시기에는 문제 도메인에서의 객체-또는 개념-들을 발견하고 기술하는 것에 중점을 둔다. 예를 들어서 도서 정보 시스템의 경우, *Book*, *Library*, 그리고 *Patron*같은 개념들을 찾아 기술하게 된다.

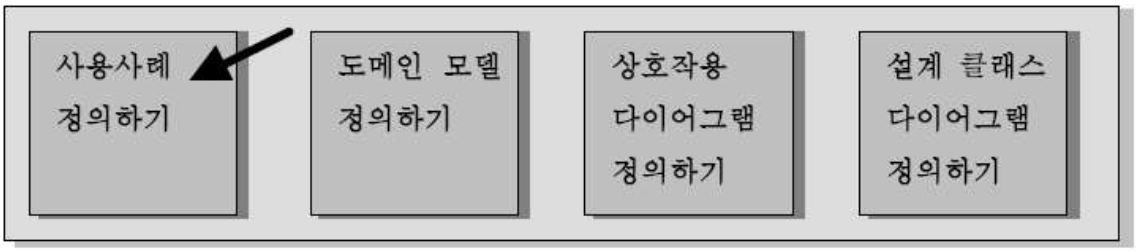
객체 지향 설계시기에는 소프트웨어 객체들을 정의하고 그 객체들이 책임을 수행하기 위해서 어떻게 상호작용 하는 가에 중점을 둔다. 예를 들어서 도서 정보 시스템의 경우 *Book* 소프트웨어 객체는 *title* 속성과 getChapter 메소드를 가질 수 있다
마지막으로, 구현시기에 또는 객체 지향 프로그래밍을 하는 동안, 설계 객체들이 Java 에서 *Book* 클래스를 만드는 것처럼 구현되어진다.



-예제

▶ 사용사례 정의하기

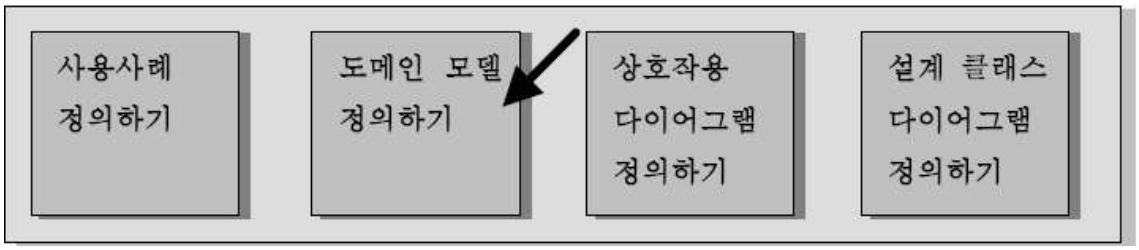
요구사항 분석은 관련된 도메인 프로세스를 기술하는 활동도 포함한다. 이것들은 사용사례로서 작성될 수 있다.



사용사례는 객체지향적인 산출물은 아니다. 사용사례는 단순하게 쓰여진 이야기 형식이다. 그러나 이 사용사례는 요구사항 분석에 있어서 가장 널리 쓰이는 도구이며 Unified Process에서 중요한 부분을 차지한다.

▶ 도메인 모델 정의하기

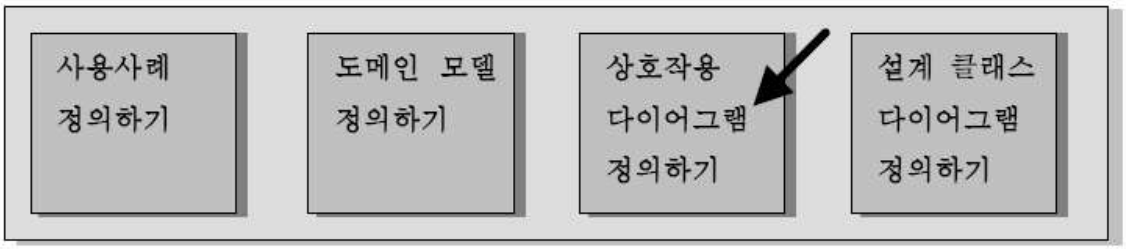
객체지향 분석에서는 객체를 분류함으로써 도메인을 기술하게 된다. 도메인을 분해하는 것은 중요하다고 생각되는 개념, 속성, 연관(association)을 식별하는 것을 포함한다.



그 결과로 도메인 모델이 생성되어지며 도메인 개념 또는 객체를 보여주는 일련의 다이어그램들로 표현되어진다.

▶ 상호작용 다이어그램 정의하기

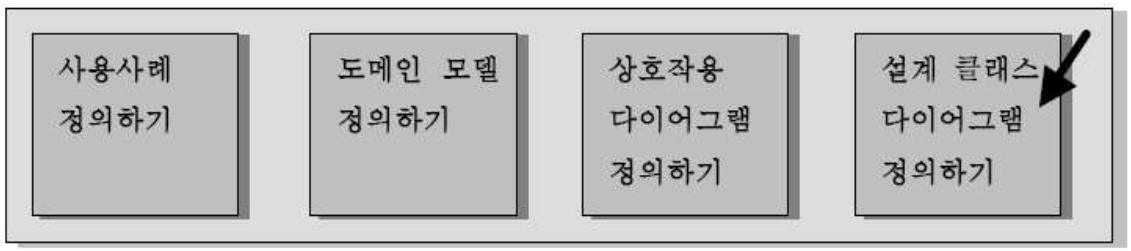
객체지향 설계는 소프트웨어 객체들과 이 객체들의 협조를 정의하게 된다. 이 협조를 제시하여주는 일반적인 표기법은 상호작용 다이어그램이다. 이는 소프트웨어 객체들간 의 메시지의 흐름과 이에 따른 메소드들의 호출을 보여준다.



소프트웨어 객체 설계와 프로그램은 실세계의 도메인에서 아이디어를 얻지만 실세계에 대한 직접적인 모델이나 시뮬레이션을 하는 것은 아니다.

▶ 설계 클래스 다이어어램 정의하기

상호작용 다이어그램에서 보여주듯이 협력하는 객체들의 동적인 관점 뿐 아니라, 설계 클래스 다이어그램을 이용하여 클래스 정의의 정적인 관점을 생성하는 것도 유익하다.
이것은 클래스의 속성과 메소드를 보여준다.



-UML

Unified Modeling Language(UML)은 비즈니스 모델링과 다른 비-소프트웨어 시스템을 위한 언어일 뿐 아니라, 소프트웨어 시스템의 산출물을 명세화하고, 가시화하고, 만들고, 문서화하기위한 언어이다
UML은 객체지향 모델링을 위한 표준 다이어그램 표기법으로 정착되고 있다. 1994년 Grady Booch와 Jim Rumbaugh는 그들의 널리 알려졌던 두 가지의 방법- Booch방법론과 OMT(Object Modeling Technique)방법론-들로부터 다이어그램 표기법을 통합하려는 노력을 시작했다. 나중에 Objectory 방법의 창시자인 Ivar Jacobson이 합류하여, 이들 그룹은 세 명의 아미고(amigo)라고 알려지게 되었다.

UML에 공헌한 많은 사람들 가운데 세분화 작업의 리더인 Cris Kobryn을 꼽을 수 있다.
UML은 1997년에 OMG(Object Manage Group, 산업체 표준 기구)로부터 표준으로 채택되었고 새로운 OMG UML 버전으로 계속 정련되어왔다.

UML은 주로 OOAD 과정 동안에 적용 되어지며 OOA/D 과정 전에 요구분석 단계가 선행되어진다.

ITERATIVE DEVELOPMENT AND THE UNIFIED PROCESS

●반복적이고 적응적인 프로세스를 정의, Unified Process의 기본 개념을 정의한다.

-PreView

반복적인 개발은 소프트웨어 개발하는 데에 어떤 기술을 요하는 접근 방법이며, 이러한 반복적인 개발 방법에 의거한 OOAD의 과정을 제시한다. Unified Process는 OOAD를 이용한 프로젝트를 위한 샘플 성격의 반복적인 프로세스이다.

소프트웨어 개발 프로세스는 소프트웨어를 만들고, 배치하고, 그리고 가능하다면 유지보수하는 것에 대한 접근 방법을 기술한다. Unified Process는 객체지향 시스템을 만들기 위한 소프트웨어 개발 프로세스로 널리 알려져 왔다.
Unified Process(UP)는 반복적인 생명주기나 위험 위주의 개발(risk-driven development)같은 일반적으로 잘 실행되는 사항들을 응집력 있고 문서화된 형태로 기술 할 수 있게 한다.

UP는 반복적인 프로세스이다. 반복적인 개발은 아주 중요한 Practice이며 UP는 OOAD를 어떻게 수행하고 어떻게 학습할 것인가에 대해서 묘사 하게 된다. 또한 UP는 요구사항 분석과 OOAD를 수행하는 프로세스로서 사용된다. 그리고 UP는 일반적인 활동과 최적의 Practice를 가지고 있다.

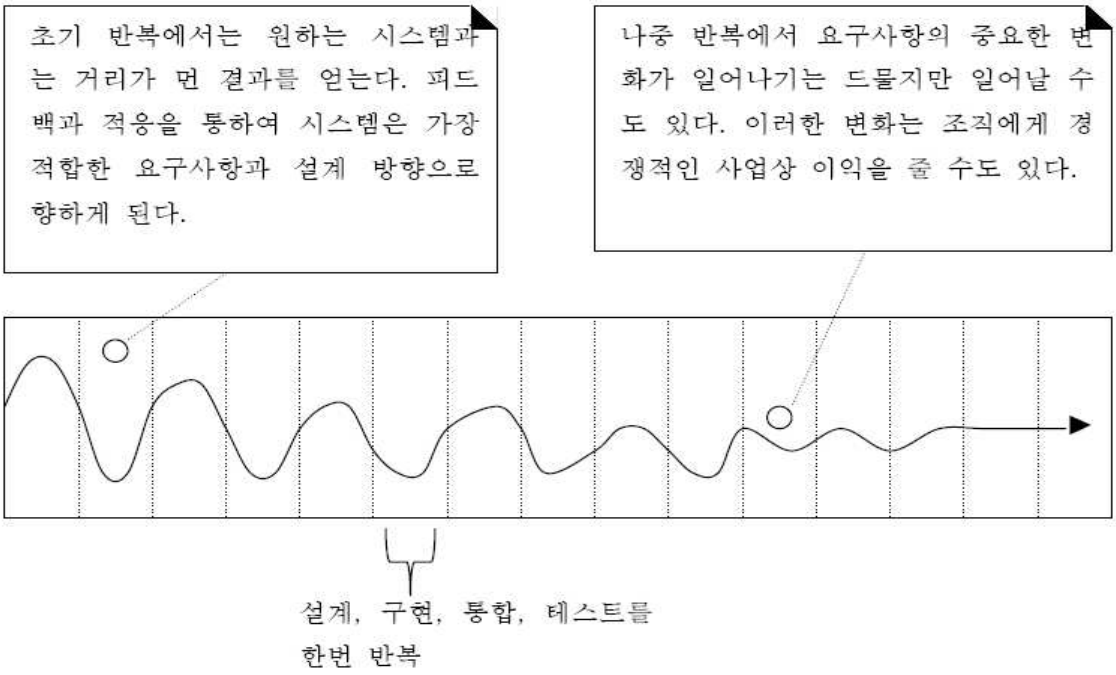
-가장 중요한 UP 아이디어: 반복적인 개발

UP는 여러 가지 좋은 Practice를 제시하고 있으며 가장 중요한 것은 반복적인 개발이다. 이러한 접근방법에서 개발은 여러 개의 짧고 고정된 기간의 반복 이라고 불리는 작은-프로젝트로 나뉘어 진다. 이들 각각의 작은-프로젝트들은 테스트와 통합이 이루어지고 실행 가능한 시스템이 된다. 각각의 반복 내부에서 자체적으로 요구사항 분석, 설계,구현 그리고 테스트 활동이 이루어진다.

반복 생명주기는 여러 번 반복을 수행하고 주기적인 피드백을 하며 원하는 시스템이 되도록 초점을 맞

추어 연속적으로 시스템을 확장하고 정련 시키는데 기초를 두고 있다. 시스템은 반복과 반복을 통하여 점진적으로 구체화 된다. 그러므로 이러한 접근 방법을 반복적이고 점진적인 개발 이라고 부른다. 초기의 반복적 프로세스 아이디어는 나선형 개발과 진화적 개발이다.

프로그래밍하기 전에 상세하게 모든 부분의 설계를 완성하게 하기 위해서 코드를 작성하는 것도 아니며 한번에 오랜 기간동안 설계에 집중하는 것이 아니다. 대략적 이고 빠른 UML 표기로서 가시적인 모델링을 통한 설계로서 '약간'의 선견이 이루어진다. 각각의 반복 작업의 결과는 실행 가능하지만 완전한 시스템이 만들어지는 것이 아니다. 즉, 생산라인 쪽으로 넘겨줄 준비가 된 것이 아니다. 시스템은 여러 번 반복개발이 이루어져야 생산 배포하기에 알맞다.



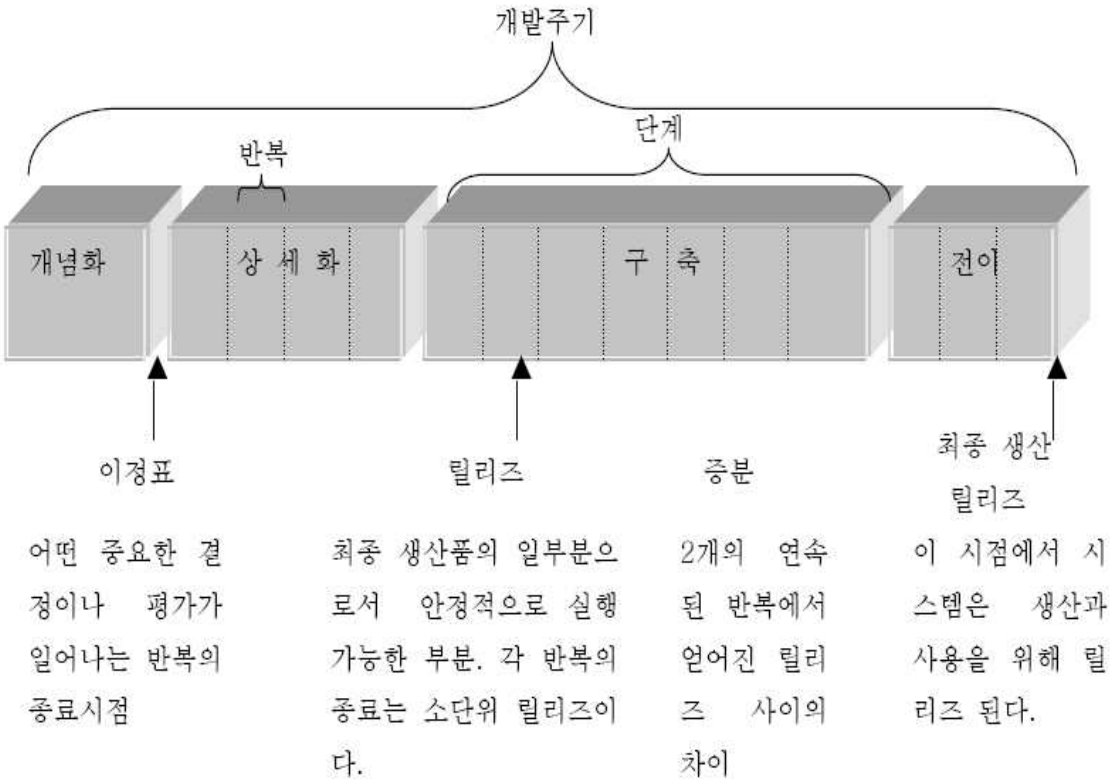
▶ 반복적인 개발의 이점

①위험요소(기술, 요구사항, 목적, 사용성 등에 관한 위험요소)를 나중보다는 초기에 감소 시킨다.
②초기부터 가시성 있는 진행을 한다.
③초기의 피드백, 사용자와의 계약, 적응과 제 삼자의 실제적인 요구에 적합하도록 하는 정제된 시스템으로 유도 등을 사용한다.
④복잡성을 조절한다. 팀은 정체된 분석이나 매우 길고 복잡한 단계에 의해서 압도당하지 않는다.
⑤반복 내에서 학습된 것이 반복에 의한 반복 개발 프로세스 자체를 개선시키기 위해서 사용되어질 수 있다.

▶ UP 의 스케줄 지향 용어

맨 처음에 요구사항을 모두 정의하고 그런 다음 대부분 모든 것을 설계하는 기존의 폭포수(Waterfall)

모델이나 순차적 생명주기와는 다르다.
개념화는 요구사항 단계가 아니다. 충분한 조사로서 계속 수행할 것인지 정지할 것인지 결정할 수 있는 적합한 단계이다.
상세화는 요구사항 단계도 아니며 설계 단계도 아니다. 핵심 아키텍처가 반복적으로 이루어
지고 상층부 위험요소 문제가 완화 되는 단계이다.



위 그림은 UP의 스케줄 지향 용어를 보여준다. 한 개발 사이클이 많은 반복으로 구성된다.

▶ Discipline

한 반복작업 동안 대부분 또는 모든 Discipline을 수행해 나아간다. 그러나 이 Discipline에 따른 상대적인 노력은 결국 변하게 된다. 초기 반복은 당연히 요구사항과 설계에 상대적으로 중점을 두는 경향이 있고 나중 반복은 피드백과 적응 프로세스를 통하여 분석과 핵심 설계를 안정화 시키듯이 중점이 덜하다.

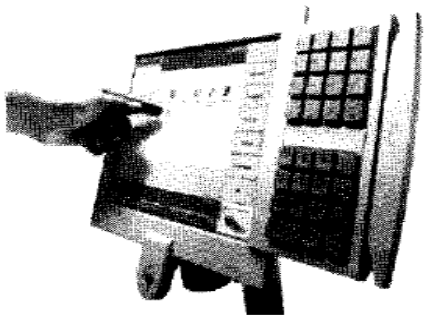
Discipline과 UP 단계(개념화, 상세화,…)와의 관계를 보면 각 단계에 따라 상대적으로 노력이 변한다. 이것은 고정된 것이 아니라 한 예를 보여주고 있는 것이다. 예를 들어 상세화 단계에서 반복은 요구사항과 설계 작업에 상대적으로 높은 가중치를 두는 경향이 있으며 구현은 약간만 한다. 구축 단계에서는 구현이 상대적으로 높고 요구사항 분석이 낮다.

CASE STUDY: THE NEXTGEN POS SYSTEM

●NextGen POS 시스템

NextGen POS(point-of-sail: 현금 출납기) 시스템이다. 문제 도매인이 분명함으로 우리는 해결하기위한 요구사항과 설계 문제를 알 수 있어서 매우 흥미가 있다. 실제적인 문제는 POS 시스템을 객체를 사용하여 조직적으로 구성하는 것이다.

POS 시스템은 판매한 것을 기록하고 지불을 계산해주는 전산화된 제품이다. 이 시스템은 실제로 점포에서 사용되고 있다. 또한 이 시스템은 컴퓨터와 바 코드 센서 그리고 시스템을 운영하는 소프트웨어 등과 같은 컴포넌트들로 구성된다. 이 시스템은 3분기 세금계산이나 재고품 관리 등과 같은 다양한 서비스 어플리케이션 들과 인터페이스 한다. 이러한 시스템은 상대적인 결함허용이 요구 된다. 즉, 원거리 서비스(재고품 시스템과 같은) 가 일시적으로 불가능할 때에도 판매 사항을 처리해야 하고 최소한 수표지불을 취급할 수 있어야 한다(그래서 사업이 장애를 받지 않아야 한다).



POS 시스템은 점차적으로 다중의 클라이언트 쪽의 터미널 및 다양한 인터페이스를 제공해야한다. 예를 들어 웹 브라우저 터미널이나 Java 스윙 그래픽 사용자 인터페이스를 가진 퍼스널 컴퓨터, 터치 스크린 입력, 무선 PDA 등과 같은 것을 제공해야 한다.

더욱이 우리는 상업적인 POS 시스템을 구축하여 직업상 서로 다른 위치에 있는 소비자에게 팔 수도 있다. POS 시스템 구매자는 시스템을 사용하여 어떻게 판매를 기록을 하고 새로운 품목을 추가하는지를 알기 위해서 시나리오로 제공되는 안내서에 따라 실행하기를 원한다.

그러므로 우리는 이러한 고객에 맞는 유연성을 제공하는 메커니즘이 필요하다.
반복적인 개발전략을 사용하여 우리는 요구사항, 객체지향 분석, 설계, 구현 들을 계속 수행한다.

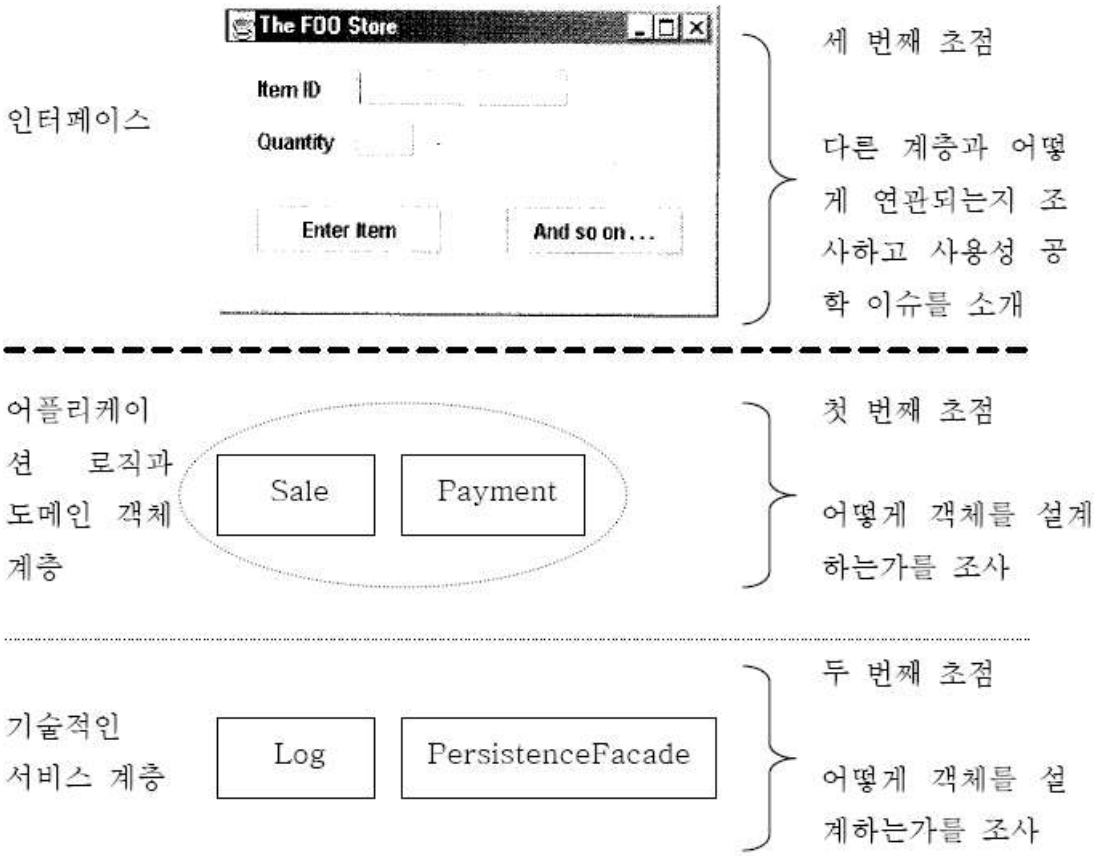
OOAD는 일반적으로 어플리케이션 로직과 기술적 서비스 계층을 모델링하는 것과 가장 관련이 있다.

NextGen 사례연구는 문제 도매인 객체들에게 어플리케이션의 요구사항을 수행하도록 하는 책임을 할당하는 것을 강조하고 있다. 객체지향 설계는 또한 데이터베이스와 접속하는 기술적인 서비스 서브시스템을 구축하는데 적용된다.

이러한 설계 접근방식에서 UI 계층에는 거의 책임이 할당되지 않는다. 그래서 '얇은(thin)' 계층으로 불린다. 윈도우를 구성하는 데에 어플리케이션 로직이나 프로세스를 수행하는 코드를 작성하지 않는다. 다른 계층에서 업무요청이 일어난다.

●반복적인 학습과 개발

NextGen POS 시스템을 다중 반복으로 개발하도록 OOAD가 적용된다; 첫번째 반복에서는 몇 개의 핵심 기능에 초점이 맞추어진다.



다음의 반복에서는 시스템의 기능성을 확장한다. 반복적인 개발이 결합되어, 분석과 설계 주제, UML 표기, Pattern 등의 제시가 반복적으로 그리고 점진적으로 소개된다.
첫번째 반복에서는 분석과 설계 주제의 핵심 내용과 기호들이 제시된다. 두 번째 반복에서는 새로운 아이디어로서 UML 표기와 Pattern을 확장한다. 그리고 세 번째 반복에서도 마찬가지 방법으로 확장한다.

INCEPTION(개념화)

●개념화 단계의 정의

개념화 단계의 목적은 모든 요구사항을 정의하는 것이 아니며 신뢰 있는 측정을 하는 것도 아니고 프로젝트 계획을 하는 것도 아니다. 너무 단순화 시킨 위험요소에서, 아이디어는 전체 목적에 타당하고 정당한 평가와 잠재적인 새로운 시스템의 적합성을 형태를 갖추도록 충분한 조사를 하며 더 깊게 조사(상세화 단계의 목적) 할 가치가 있는지 결정하는 것이다.
그러므로 개념화 단계는 개념화 단계는 프로젝트의 진지한 조사(상세화 단계에서 수행)가 아니라 조사할 가치가 있는지 어떤지를 결정하는 것이다.

반복 개발에서 중요한 점은 개념화 단계에서 이 산출물들은 단지 부분적으로만 완성되어지고 다음 반복에서 더욱 상세화 된다는 것이며 실제적인 변수가 더 필요하다고 생각되지 않는 한 새로운 것 조차도 만들어 지지 않는다는 것이다. 개념화 단계에서 조사와 산출물의내용은 가벼워야만 한다.

●요구사항 이해하기

모든 요구사항이 똑같이 만들어지지 않는다.
요구사항은 시스템과-더 넓게는 프로젝트와-일치해야만 하는 능력과 조건들이다 요구사항 작업의 초기 문제는 의뢰인과 개발자에게 명확하게 말할 수 있도록 하기 위해서 실제로 필요한 것이 무엇인가를 찾고 커뮤니케이션하며 기록하는 것이다.

-요구사항 타입

UP에서 요구사항은 FURPS+ 모델에 다라 범주가 정해지며 다음의 의미를 갖는다.

①기능성-특징, 수행능력, 보안
②사용성-인간적 요소, 협조, 문서
③신뢰성-실패 빈도, 회복성, 예견성
④수행성-시간반응, 효율, 정확성, 유용성
⑤지지성-적응, 유지, 국제화, 형태

이러한 요구사항의 몇 가지는 모아놓은 것을 질적 속성, 질적 요구사항 이라 한다. 이것들은 사용성, 신뢰성, 수행성과 안정성을 포함한다. 보통, 요구사항들은 기능적 요구사항(행위적)과 비-기능적 요구사항(그 밖의 것들)으로 분류 된다.

사용사례 모델: 요구사항을 문맥으로 쓰기

● 목적과 이야기

고객과 시스템 사용자는 (UP에서 필요하다고 알려진) 목적을 가지고 있으며 컴퓨터 시스템이 오일 저장량을 측정하고 판매를 기록하는 여러 가지 일들을 해주기를 원한다. 이러한 목표와 시스템 요구사항을 획득하는 여러 가지 방법이 있다. 좋은 방법은 간단하고 보기 쉬워야 하는데 이것은 고객과 시스템 사용자에게 요구사항을 알기 쉽게 해주며 그들이 요구사항정의나 평가에 도움을 주기 때문이다. 이러한 방법이 목표에 벗어나는 위험요소를 감소시킨다.
사용사례는 요구사항을 간단하게 해주고 모든 제 삼자가 이해할 수 있는 메커니즘이다. 사용사례는 목표에 맞게 시스템을 사용하는 이야기들이다.
사용사례의 간단한 예

> **-판매 처리하기:** 고객은 구입할 물품을 가지고 계산대로 온다. 출납원은 구입하는 각각의 물품을 기록하기 위해서 POS 시스템을 사용한다. 이 시스템은 구입하는 전체 금액과 각각의 항목을 상세하게 표현한다. 고객은 시스템이 확인과 기록을 하는 지불 접수처로 들어간다.. 시스템은 상품목록을 정리한다. 고객은 시스템으로부터 영수증을 받고 물건을 가지고 떠난다.

사용사례는 보통 이보다 더 자세한 것을 필요로 한다. 다양한 제 삼자의 목표를 맞추는데 도움을 주도록 하기 위해서 시스템 사용을 이야기식 으로 써서 기능 요구사항을 조사하고 기록하는 것은 필수적이다. 즉 이것은 사용하는 사례(case of use)이다. 필요한 것이 무엇인지 조사하고 결정하는 것은 어렵더라도 이것은 어려운 아이디어가 아니다. 세부사항이 유용한 수준으로 주어지도록 사용사례를 집중적으로 적는다.
사용사례에 관한 많은 자료가 있으나 단순한 아이디어를 복잡한 계층으로 애매모호하게 전개하여 위험요소를 불러 일으키는 일이 자주 일어난다. 이것은 사용사례를 수행하는 초심자에게 잘못 인식시켜서 이야기식 으로 사용사례를 적기 보다는 사용사례 다이어그램, 사용사례 관계, 사용사례 패키지와 선택적인 속성 들과 같은 부수적인 이슈에 관심을 갖게 한다. 이것들은 사용사례의 강점은 요구에 따라 복잡성과 형식성 으로 이리저리 알아보는 능력 이라고 말한다.

● 사용사례와 값 추가하기

첫번째, 몇 가지 비형식적으로 용어 정의를 한다; 행위자는 사람(역할이 주어지는), 컴퓨터 시스템이나 조직 등과 같이 행위를 하는 어떤 것이다; 예를 들어 출납원은 행위자 이다. 시나리오는 행위자와 시스템 사이의 행위와 상호작용을 명백하게 열거한 것이다; 이것은 또한 사용사례 실체라고도 불린다. 이것은 시스템을 사용하는 하나의 특별한 이야기이며 사용사례를 표현하는 하나의 길이다; 예를 들어 수표를 가지고 성공적으로 물건을 사는 시나리오가 있고 신용카드 처리 거부로 인하여 물건을 사는데 실패하는 시나리오가 있다.

-반품 취급

성공적인 시나리오: 고객은 반품할 물품을 가지고 계산대로 온다.
출납원은 반품된 물건을 기록하기 위해서 POS 시스템을 사용한다…

선택적 시나리오: 신용카드 인증이 거절되면 고객에게 알리고 다른 지불 방법을 요구한다.
시스템에서 물품 확인을 찾을 수 없으면 출납원에게 알리고 안내 목록의 확인코드를 제시한다.
시스템이 외부 계산 시스템과 통신 접속을 실패하면, …

사용사례 실체들의 모임으로서 이들 각각의 실체들은 시스템이 특별한 행위에 대하여 관찰할 수 있는 결과 값의 처리를 수행하는 일련의 행위이다.
“관찰할 수 있는 결과 값”이라는 표현은 미묘하지만 중요한데 이것은 시스템의 행위가 사용자에게 값을 제공하는 것을 강조 해야만 하는 태도에 중점을 두고 있기 때문이다.

> 사용사례 작업에서 중요한 태도는 시스템 요구사항만 생각하여 특징이나 기능에 대하여 상세한 목록을 작성하는데 초점이 있는 것이 아니라 “사용자에게 관찰할 수 있는 값을 제공하거나 이들의 목적에 부합 되도록 하기 위해서 시스템을 어떻게 사용해야 하는가”라는 질문에 초점을 두고 있다.

● 사용사례와 기능 요구사항

-사용사례는 요구사항이다

사용사례는 시스템이 무엇을 수행할 것인지를 지시하는 요구사항
이다. FURPS+ 요구사항 타입에서는 기능과 행위를 강조하는데 이것은 다른 타입이 사용사례와 밀접한 관계가 있을 때 이들 타입에도 사용될 수 있다. UP와 가장 현대적인 방법론에서사용사례는 요구사항 조사와 정의를 하는데 추천되는 중심 메커니즘이다. 사용사례는 시스템이 어떻게 행위를 할 것인지에 대한 내용을 정의한다.
명확히 말해서 사용사례는 요구사항이다 (비록 요구사항 전체는 아닐지라도). 요구사항을 “시스템이 수행하여야 하는…”기능이나 특징들의 목록이라고 생각하는 사람들이 있다. 그러나 그렇지 않다. 사용사례의 중심 생각은 상세한 특징들의 목록을 작성하는 구식 스타일을 지양하는 것이며 오히려 기능 요구사항에 대한 사용사례를 쓰는 것이다.
사용사례는 다이어그램이 아니라 문서이며, 사용사례 모델링은 그림을 그리는 것이 아니라 글을 쓰는 행위인 것이다. 그러나 UML은 사용사례와 행위자 그리고 이들 사이의 관계를 보이기 위해서 사용사례 다이어그램을 정의한다.

● 사용사례 타입과 형식

-블랙박스 사용사례와 시스템 책임

블랙박스 사용사례가 가장 일반적으로 사용된다; 이것은 시스템, 시스템의 컴포넌트, 또는 설계 등의 내부작동은 묘사하지 않는다. 오히려 시스템이 가져야 할 책임을 묘사하는데 이책임은 객체지향 생각-소프트웨어 요소는 책임을 가지고 있고 책임을 가지고 있는 다른 원소와 협력한다는 생각-에 있어서 보통 일체화 시키는 상징적인 주제이다.
블랙박스 사용사례로 시스템의 책임을 정의 함므로서 어떻게 설계를 할 것인가에 대한 결정 없이 시스템이 무엇을 할 것인가(기능요구사항)를 구체화 시키는 것이 가능하다. 실제로, "분석"과 "설계"의 정의는 "무엇"과 "어떻게"로서 비유된다. 이것은 소프트웨어 개발에 있어서 중요한 개념을 제공한다: 요구사항 분석 시기에는 "어떻게"라는 결정을 피하고 블랙박스처럼 시스템의 외부 행위를 명세화 한다. 이 후 설계 시기에 이 명세화에 적합하도록 해결책을 찾는다.

블랙박스 스타일	블랙박스 스타일이 아님
시스템은 판매사항을 기록한다.	시스템은 데이터베이스에 또는 그 이와의 것에 판매사항을 기록한다. … 시스템은 판매사항에 대하여 SQL INSERT 문을 만든다. …

-유형별 사용사례 형식

사용사례는 필요에 따라 서로 다른 형식으로 작성된다. 사용사례는 블랙박스와 화이트박스의 가시적인 유형 이외에 다양한 정도의 형식으로 쓰여진다.

①개요 형식(brief formality)-보통 주요 성공 시나리오로 한 문단으로 명료하게 요약한 형태. 앞에서 제시한 판매 처리하기의 예가 여기에 속한다.
②정리 형식(causal formality)-비 형식적인 문단 형태. 다양한 시나리오로 구성되는 여러 문단 형태. 앞에서 제시한 반품 취급의 예가 여기에 속한다.
③갖춘 형식(fully dressed formality)-가장 상세한 형태. 모든 단계와 변형이 상세하게 쓰여지고 전제조건과 책임 같은 부분을 지지하고 있다.

● 각 부문별 설명하기

-머리말 요소

많은 머리말 요소를 선택하여 쓸 수 있다. 이 요소는 주요 성공적인 시나리오에 앞서서 읽어야 할 중요한 것으로 시작 위치에 놓는다. 외부의 중요한 자료는 사용사례 끝에 위치 시킨다.

①초기 행위자:

목적에 맞게 시스템 서비스를 수행하는 중요한 행위자.
중요한 것: 제 삼자와 관심사
이 목록은 언뜻 보아서 나타나는 것 이상으로 중요하며 실제적이다. 이것은 시스템이 무엇을 해야 하는지를 제시하며 범위를 설정하게 해준다.

[시스템]은 계약의 행위적인 부분을 상세히 알려주는 사용사례로서 제 삼자 사이의 계약을 조정한다. 사용사례는 계약의 행위 뿐만 아니라 제 삼자의 관심사를 만족시키는 것과 관련된 행위 모두를 포함하고 있다.

②제 삼자와 관심사:

- 출납원: 정확하고 빠른 기록이 요구되며 지불 오차가 없어야 한다. 현금 오차는 자신의 봉급에서 공제한다.
- 판매원: 판매 수수료가 새롭게 바뀌기를 원한다.
- …

-전제조건과 성공보장(결론조건:Postconditions)

전제조건은 사용사례에서 시나리오를 시작하기 전에 항상 참 이어야 하는 상태를 말한다.
전제조건은 사용사례 안에서 검사되는 것이 아니라 참 이라고 가정되는 조건이다. 전제조건은 로그 인이나 또는 “출납원의 신분이 확인되고 승인되었음”과 같이 성공적으로 수행된 또 다른 사용사례의 시나리오를 포함한다. 참 이어야 하는 조건들이 잇는 것이지 “시스템은 파워가 있다.”와 같이 실제적인 값을 쓰는 것이 아니라는 것에 유의 하자. 전제조건은 사용사례를 읽는 사람이 주의를 기울이고 있다고 사용사례 작성자가 생각하는 가정들과 상당히 상통한다.
①전제조건: 출납원은 신분이 확인되어져야 하고 승인 되어져야 한다.
성공보장(결론조건): 판매는 기록 된다. 세금은 올바르게 계산된다. 회계와 재고품은 새롭게 정리된다. 수수료가 기록된다. 영수증이 만들어진다.
②주요 성공 시나리오와 단계 (또는 기본 흐름)
이것은 “기본 흐름” 이라고도 말한다. 이것은 제 삼자의 관심사를 만족하는 전형적인 흐름을 기술한다. 이것은 어떤 조건이나 분기를 포함하지 않는다. 이것은 매우 이해하기 쉽고 확장할 수 있도록 일관성 있게 되어 있으며 확장 부분에 대한 모든 조건은 뒤로 미루어 취급한다.
시나리오는 3가지 종류의 단계로 기록된다.

1. 행위자들 사이의 상호작용
2. 타당성(보통 시스템에 의한 타당성)
3. 시스템에 의한 상태변화(예를 들어 어떤 것을 기록하고 수정하기)

사용사례 중에서 한 단계가 항상 이러한 분류에 속하는 것은 아니다. 단지 시나리오를 시작하는 트리거 이벤트 역할을 할 뿐이다.
행위자의 이름을 구별하기 쉽게 하기 위해서 대문자로 시작하는 것이 보통이다.

주요 성공 시나리오:
1. 고객은 구매할 상품이나 서비스를 가지고 POS 계산대로 온다.
2. 출납원은 새로운 판매를 시작한다.
3. 출납원은 물품 확인작업에 들어간다.
4. …
출납원이 다른 지시를 하기 전까지 3-4 단계를 반복한다.
5. …

-확장(선택적 흐름)

확장은 매우 중요하다. 확장은 그 밖의 모든 시나리오 또는 성공이나 실패로 분기되는 것을 표현한다. 갖춘 형식의 예에서 확장 부분이 주요 시나리오 부분 보다 상당히 더 길고 더 복잡하다는 것을 관찰하라. 이러한 것이 보통이며 권고되어 진다. 확장은 또한 “선택적 흐름”으로 불리어 진다.
사용사례에서 주요 성공 시나리오와 확장의 합성으로 제 삼자의 모든 관심을 “거의” 만족시켜야 한다. 이점은 매우 중요한데 왜냐하면 비기능 요구사항이 사용사례 보다는 오히려 보충사례 명세화로서 표현되듯이 관심도 표현되기 때문이다.
확장 시나리오는 주요 시나리오부터의 분기이며, 주요 시나리오에 관련하여 기술된다. 예를들어 주요 시나리오 단계3에서 시스템에 올바르게 들어갈 수 없거나 확인 불명으로 인하여 물품 확인 불가로 되어질 수 있다. 이 확장은 “3a”라벨로 표기한다. 이것은 먼저 조건을 만족하여야 하고 그런 다음 반응 하여야 한다. 단계3에서 또 다른 선택적 확장은 “3b”라는 라벨이 붙이고 이런 형태로 계속한다.

확장
3a. 확인 불가:
1. 시스템은 오류 신호를 내고 등록을 거절한다.
3b. 같은 항목 영역이 여러 개 있고 야채버거 5꾸러미처럼 단일 항목 꾸러미가 있다.
출납원은 항목 영역을 확인하고 개수를 입력한다.
1. 출납원은 항목 영역을 확인하고 개수를 입력한다.

-특이 요구사항

비기능 요구사항, 특징적 속성 또는 제약조건이 사용사례와 특별히 관련이 있을 때 사용사례와 함께 이것들을 기록한다. 이것은 수행성, 신뢰성과 사용성 같은 특징 등을 포함하며 명령이 내려지고 그럴듯해 보이는 설계 제약조건(주로 I/O 장치) 등도 포함한다.

특이 요구사항
- 큰 평면 패널 모니터에 터치 스크린으로 주어지는 UI. 글씨가 1m 근방에서 잘 보여야만 한다.
- 신용카드 인증이 30초에 90% 반응한다.
- 재고품 시스템과 같은 원거리 서비스 접속에 실패할 때 다소간 견고한 복구를 원한다.

- 국제화된 언어가 디스플레이 되어야 한다.
- 3과 7단계에서 사용 가능한 비즈니스 규칙이 삽입될 수 있다.

사용사례와 함께 이러한 것들을 기록하는 것은 UP의 내용이며 사용사례를 첫번째 작성할 때 당연한 것이다. 그러나, 이러한 요구사항 들은 전체 아키텍처 분석 시기에 고려되어야 할 사항이므로 많은 사람들은 보충사항 명세화에서 모든 비기능 요구사항 등을 궁극적으로 합병해야 한다고 생각한다.

-기술과 데이터 변동 목록

무엇이 아니라, 어떻게 하여야 하는가에 있어서 기술적 변동이 일어난다. 이것을 사용사례에 기록하는 것이 중요하다. 일반적인 예가 입출력을 고려하여 제 삼자가 제기하는 기술적 제약 조건이다. 예를 들어 제 삼자는 “POS 시스템은 카드 리더나 키보드를 사용하여 신용카드 계정 관리가 이루어져야 한다.”라고 의견을 제시할 수 있다. 이러한 것들은 초기 설계 결정이나 제약조건의 예이다; 일반적으로, 때 이른 설계 결정은 피하는 것이 필수적이다.
그러나 입출력 기술에 관련해서는 때이른 설계 결정이 불가피하다.
물품 확인을 위해서 UPC나 EAN 등을 사용하여 바코드 기호를 해독하는 것과 같이 데이터 체계에서 여러 가지 기술 변동을 이해하는 것이 필요하다.
여기에서의 목록은 이러한 기술 변동을 기록하는 것이다. 이것은 또한 특별한 단계에서 파악될 수 있는 데이터에 있어서의 변동을 기록하는 것이다.

기술과 데이터 변동 목록
3a. 물품 확인은 바코드 레저 스캐너(바코드가 있는 경우) 또는 키보드를 사용한다.
3b. 물품 확인은 UPC, EAN, JAN, 또는 SKU 코딩 체계를 사용할 수 있다.
7a. 신용카드 정보는 카드 판독기나 키보드를 사용한다.
7b. 신용카드 지불 서명은 영수증 용지에 받는다. 그러나 2년 내에 많은 고객들이 전자서명
을 사용할 것이라고 예견된다.

● 사용사례의 목적과 범위

-기본적인 비즈니스 프로세스를 위한 사용사례

EBP 사용사례
컴퓨터 응용을 위한 요구사항 분석을 하기 위해서는 **기본적인 비즈니스 프로세스(EBS: Elementary Business Process)** 수준의 사용사례에 초점을 둔다.

EBP는 비즈니스 프로세스 공학 분야의 용어이다.
다음과 같이 정의 된다: 비즈니스 이벤트를 고려하여 한 사람에 의해서 한 장소에서 일정한 시간에 수행된 업무로서 알맞은 비즈니스 이벤트를 삽입하고 일관된 상태로 데이터를 남겨두는 업무를 말한다. 예를 들어 신용카드나 주문가격 승인하기 등의 업무를 말한다.

이것은 문자 그대로 취급될 수 있다: 만일 두 사람이 요구되어지거나 한 사람이 다각적으로 검토한다면 EPB로서 사용사례는 적합하지 못한 것인가? EBP의 정의로만 볼 때는 적합하지 못하지만 아마도 그렇지는 않을 것이다. 이것은 “재고 품목 지우기”나 “문서 인쇄하기”와 같이 하나를 취급하는 작은 단계는 아니다. 오히려 주요 성공 시나리오는 5 내지 10 단계를 가지고 있다. 단일 회기 동안에 수행되는 업무처럼“공급자의 계약 협의”와 여러 날이 걸리는 많은 회기를 다루지 않는다. 아마도 10분에서 한시간 이내로 시간이 걸릴 것이다. UP의 정의에 의하면 이것은 관찰되는 많은 비즈니스 변수를 삽입하는 것이 강조되고 시스템과 데이터가 안정적이고 일관된 상태로 유지시키는 결과를 가져온다.
보통 사용사례에서 범하기 쉬운 실수는 너무 낮은 수준에서 많은 사용사례를 정의하는 것이다.즉, 하나의 단계로서 부분 기능 이나 EBP 범위 안에서 부분 업무를 정의 하는 것이다.

-사용사례와 목적

행위자는 자신을 만족시켜줄 어플리케이션 사용하고자 하는 목적(요구)을 가지고 있다. 결과적으로 EBP 수준의 사용사례는 시스템 사용자나 초기 행위자의 목적을 충족시키는 것을 초점으로 하는 사용자 목적-수준의 사용사례로 불려진다.

1. 사용자의 목적을 찾는다.
2. 각각에 대하여 사용사례를 정의 한다.

이것은 사용사례 작성자를 위해서 강조하고자 하는 것을 약간 변형시킨 것이다. “사용사례가 무엇인가”라고 질문하기 보다는 “너희 목적이 무엇인가”라는 질문에서 출발한다.
사실, 사용사례의 이름은 사용자의 목적을 강조하기 위해서 그 목적이 반영된 이름이어야 한다-목적: 판매 사항을 파악하며 판매를 수행한다; 사용사례: 판매 처리하기.
이와 같이 이름의 유사성으로 인하여 EBP 가이드라인은 목적과 사용사례가 만족할만한 수준에 있는지 어떤지를 결정하는데 똑같이 적용될 수 있다.

요구사항 워크숍에 우리들이 함께 참여하고 있다고 생각하자. 우리는 각자 질문 할 수 있 다: “당신은 무엇을 하십니까”(대략적인 사용사례 질문) “당신의 목적은 무엇 입니까” 첫 번째 질문은 현재의 해결책과 절차 그리고 그들과 관련된 복잡성을 더 반영하는 것 같 다. 두 번째 질문은 새롭고 개선될 해결책을 보이기 위한 목적 계층(“목적을 위한 목적이 무 엇인가”)을 이끌어 조사를 결합시키고 비즈니스 변수를 삽입하는 데 초점이 있으며 제삼 자가 시스템으로부터 서비스 받기를 원하는 것을 토의를 통해 얻고자 하는 것이다.

● 초기 행위자, 목적 및 사용사례 찾아내기

사용사례는 초기 행위자의 사용자의 목적을 만족시켜야 한다. 그러므로 기본적인 절차는 다음과 같다:.
1. 시스템 경계(boundary)를 설정한다. 시스템 단위가 소프트웨어 어플리케이션인지 하드웨어와 어플리케이션인지, 그리고 시스템을 사용하는 사람을 추가시킬 것인지 조직 전체를 대상으로 할 것인지 등에 관한 시스템 경계를 설정한다.
2. 초기 행위자를 확인한다. 이들 행위자는 시스템 서비스 사용을 통하여 수행되어지는 사용자 목적을 가지고 있다.
3. 각각의 사용자 목적을 확인한다. EBP 가이드라인을 만족하는 가장 높은 사용자 목적 수준까지 목적을 끌어 올린다.
4. 사용자 목적을 만족 시키는 사용사례를 정의한다. 목적에 따라서 사용사례의 이름을 정한다. 보통, 사용자 목적-수준 사용사례는 사용자 목적과 일대 일이 될 것이다.

-단계 1: 시스템 경계 설정하기

POS 시스템 사용사례에서 시스템 자체는 설계상의 시스템이다. 이 시스템 외부의 모든 것들 예를 들어 출납원, 지불인증 서비스 등과 같은 것들은 시스템 경계 밖에 놓인다.
시스템 경계가 명확하지 않다면 시스템 외부-외부 초기 행위자와 보조 행위자-가 무엇 인지를 정의 함으로서 명확해 진다. 일단 외부 행위자가 확인 되면 시스템 경계는 명확해 진다.
예를 들어, 시스템 경계 안에서 지불인증에 대한 완전한 책임이 이루어질 수 있을까? 그렇지 않다. 외부의 지불인증 서비스라고 하는 행위자가 있어야 한다.

-단계 2와 3: 초기 행위자와 목적 찾기

사용자의 목적에 앞서서 초기 행위자를 확인하는 것은 인위적이다; 요구사항 워크숍에서 사람들은 목적과 행위자를 함께 생각한다. 때때로 목적은 행위자를 나타내게 하고 행위자는 목적을 나타내게 한다.

-행위자와 목적을 찾기 위하여 고안된 질문

분명한 초기 행위자와 사용자 목적에 덧붙여서 다음의 질문은 실수할 수 있는 그 밖의 다른 것들을 확인하는데 도움을 준다.
누가 시스템을 시작하고 종료하는가? 누가 시스템을 관리하는가? 누가 사용자와 보안을 관리하는가? 시스템이 시간 이벤트에 반응하여 어떤것을 하기 때문에 “시간”은 행위자인가?
시스템이 고장 났을 때 재 시작 시키는 모니터링 프로세스가 있는가?
시스템 활동과 수행에 대하여 누가 평가하는가? 어떻게 소프트웨어 업데이트를 취급하는가?
누가 로그를 평가 하는가? 원격으로 로그가 복구 되는가?

-초기 행위자와 보조 행위자 (Primary and Supporting Actors)

초기 행위자는 시스템 서비스 사용을 통하여 수행되어지는 사용자 목적을 가지고 있음을 상기하자. 이들은 자신의 목적을 만족 시키기 위해 시스템을 사용한다. 이것은 설계상에서 시스템에 서비스를 제공하는 보조 행위자와 구별된다. 지금은 보조 행위자를 찾는 것이 아니라 초기 행위자를 찾는데 초점을 두고 있다.

-행위자: 목적 목록

초기 행위자와 이들의 사용자 목적을 사용자-목적 목록으로 기록하라. UP 산출물의 견지에서 이 목록은 가시화된 산출물(다음 장에서 기술할 것임) 부분에 속한다.

행위자	목적		행위자	목적
출납원	판매 프로세스 대여 프로세스 반품 취급 현금 입금 현금 출금 …		시스템 관리자	사용자 삽입 사용자 수정 사용자 지우기 보안 관리 시스템 테이블 관리 …
관리자	시작 종료		판매 활동 시스템	판매와 데이터 수행 능력 분석
…	…		…	…

판매활동 시스템은 네트웍에 연결되어 있는 각각의 POS 시스템으로부터 판매 데이터를 자주 요청하는 원거리 어플리케이션이다.

-프로젝트 계획의 범위

이 목록은 간단해 보이지만 작성하는 데 있어서 어떤 것 보다도 현실성이 있다.
“로그인” 목적에 EBP 규칙을 적용하는 앞의 예를 생각하자. 워크숍에서 이 목록을 작성하는 동안에 출납원은 사용자 목적 가운데 하나로서 “로그인”을 제시할 수 있다. 시스템 분석가는 더 깊이 숙고하여 로그인의 낮은 수준의 메커니즘을 넘어서 사용자의 “신분을 확인하고 증명하는” 수준까지 그 목적 수준을 끌어올린다.
그러나 분석가는 이것이 EBP 가이드라인을 넘어선다는 것을 깨달고 사용자 목적에 포함시키지 않는다. 예를 들어 경험 있는 분석가가 “사용자 인증은 EBP가 아니므로 사용자 목적에 포함되지 않는다”는 것을 알고 있는 것처럼 이들이 과거의 경험이나 연구로부터 발견적 방법을 가지고 있기 때문에 현실성은 이와는 좀 다를 수 있다.

-시스템 경계에 관련되는 초기 행위자와 사용자 목적

판매 처리하기의 사용사례에서 왜 초기 행위자가 고객이 아니라 출납원일까? 왜 고객은 행위자-목적 목록에 나타나지 않을까? 이에 대한 대답은 그림 6.1에서와 같이 시스템 설계상에서 시스템의 경계에 의존하여 생각하기 때문이다. 사업체나 계산대 서비스를 하나의 시스템으로 생각하여 본다면 초기 행위자는 고객이 될 것이며 이때 고객의 목적은 물건을 사거나 서비스를 받는 것이 된다. 그러나 POS 시스템만을 시스템 경계로 생각한다면 초기 행위자는 출납원이 될 것이며 출납원의 목적은 고객의 판매 처리하기를 제공받는 것이 된다.

-이벤트 분석에 의한 행위자와 목적

행위자, 목적과 사용사례를 찾기 위한 또 다른 접근 방법은 외부 이벤트를 확인하는 것이다.
그것들이 무엇이며, 어디서 왜 왔는가? 이벤트 집단들이 같은 EBP-수준의 목적이나 사용사례에 속할 때가 종종 있다.

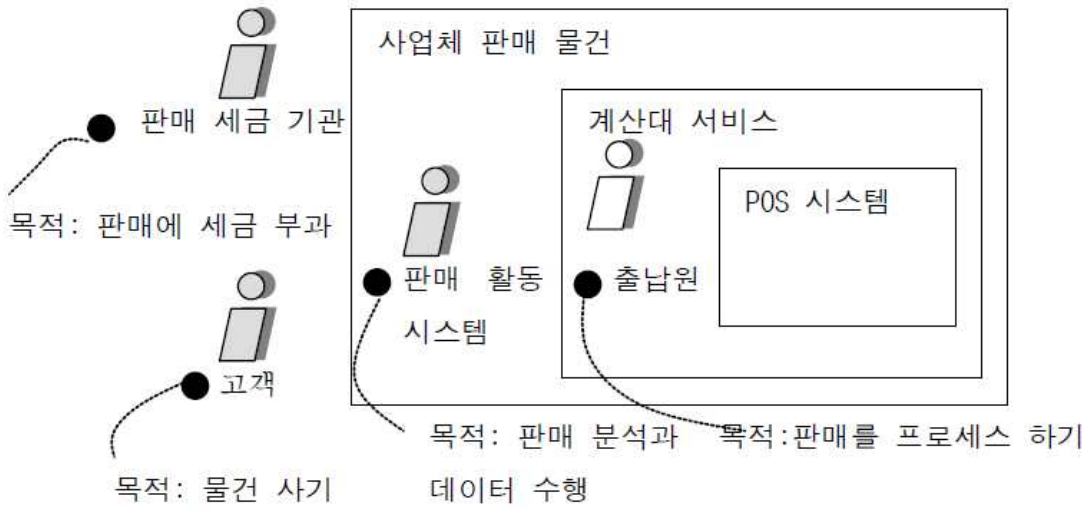


그림 6.1 시스템 경계를 달리 했을 때의 초기 행위자와 목적

외부 이벤트	행위자로부터	목적
판매 품목명 입력	출납원	판매를 처리하기
지불하기	출납원 또는 고객	판매를 처리하기

-단계 4: 사용사례 정의하기

일반적으로 각각의 사용자 목적에 대하여 하나씩 EBP- 수준의 사용사례를 정의 한다. 사용자 목적이 드러나게 사용사례의 이름을 정한다.
예를 들어, 목적: 판매를 처리한다, 사용사례: 판매 처리하기.

각각의 목적에 대하여 단 하나의 사용사례를 정의하는 예외적인 경우는 CRUD(create, retrieve, update, delete)각각의 서로 다른 목적을 <X> 관리하기로 하나의 CRUD 사용사례로 만드는 것이다. 예를 들어, 목적 "사용자 편집", "사용자 삭제" 등과 같은 목적들은 사용자 관리하기 라는 하나의 사용사례로 만족 된다.

"사용사례 정의하기"는 몇 분 안 걸리게 단지 이름만 적는 것에서 몇 주가 걸리는 갖춘 형식 사용사례를 작성하기까지 여러 가지 수준의 노력이 필요하다. 이장 뒷부분의 UP 프로세스 부분에서 - 언제 얼마나 많이 작업하는지-반복 개발과 UP의 내용에 이 작업이 포함된다.

● 행위자

행위자는 행위를 하는 어떤 것으로서 다른 시스템의 서비스를 요청할 때 논의되는 시스템 (SuD: System under Discussion) 자체를 포함한다. 초기 행위자와 시스템을 지지하는 행위자는 사용사례 문맥의 행위 단계에 나타날 것이다. 행위자는 사람 뿐만 아니라 조직, 소프트웨어와 기계로서 역할을 한다. SuD에 관련된 외부 행위자는 세가지 종류가 있다.

초기 행위자(Primary actor)-SuD 서비스 사용을 통하여 사용자 목적을 이행한다. 예를들어 현금 출납원이 여기에 속한다.

①왜 확인을 하는가? 사용사례를 이끄는 사용자 목적을 알기 위함이다.
지지 행위자(Supporting actor)-SuD 에 서비스(예를 들어 정보)를 제공한다. 자동화된 지불인증 서비스는 이것의 한 예가 된다. 가끔 컴퓨터 시스템이 여기에 속하며 조직이나 사람이 될 수도 있다.
②왜 확인을 하는가? 외부 인터페이스와 프로토콜을 명확하게 하기 위함이다.
보이지 않는 행위자(Offstage actor)-사용사례의 행위에 관심이 있으나 초기 행위자나 지지 행위자가 아니다; 예를 들어 정부의 세무서가 여기에 속한다.
③왜 확인을 하는가? 모든 필요한 관심을 확인하고 만족하는지를 확인하기 위함이다. 보이지 않는 행위자는 명확하게 이름이 정해지지 않는 한 때때로 미묘하고 놓치기 쉽다.

● 사용사례 다이어그램

UML 은 사용사례와 행위자의 이름을 보여주고 이들 사이의 관계를 보여주는 사용사례 다이어그램 기호를 제공한다.

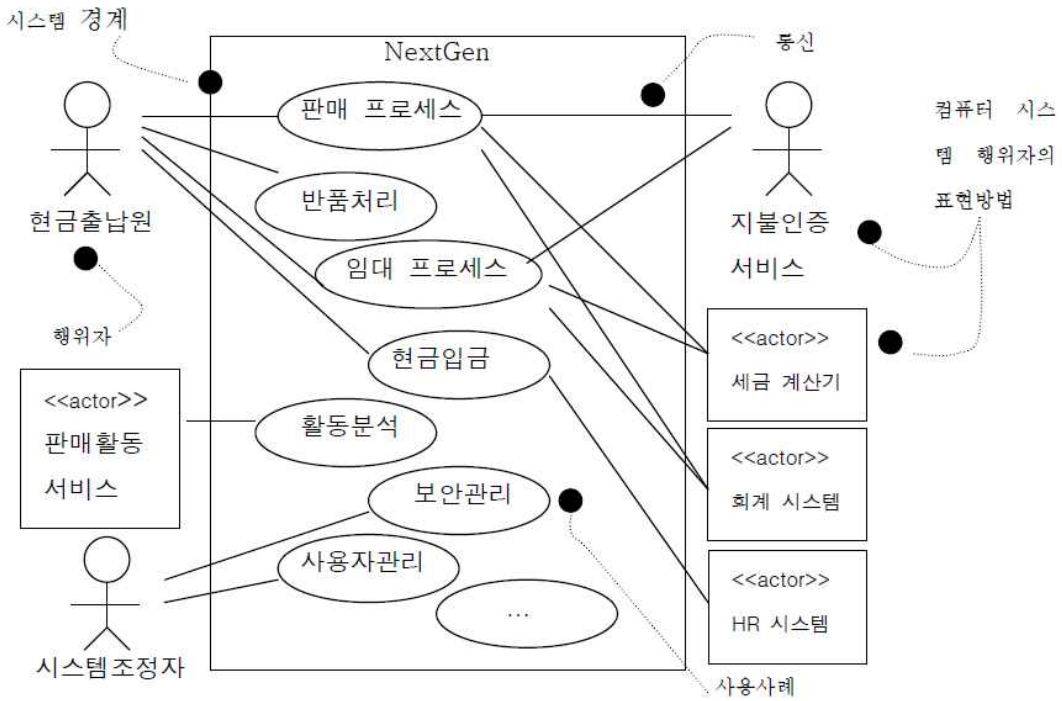


그림 6.2 사용사례 문맥 다이어그램의 일부분

사용사례 모델링을 하는 초심자는 텍스트로 쓰기보다는 사용사례 다이어그램과 사용사례 관계에 몰두하게 된다. Anderson, Fowler, Cockburn 과 그 밖의 세계적인 사용사례 전문가들은 사용사례 다이어그램과 사용사례 관계를 중요시하지 않고 대신에 쓰는 것에 초점을 두고 있다. 이러한 주의점을 가지고 단순한 사용사례 다이어그램은 시스템의 간결하고 가시적인 문맥 다이어그램을 보여주며 외부 행위자와 이들이 어떻게 시스템을 사용하는가를 예시해준다는 것을 알아야 한다.

이것은 좋은 문맥 다이어그램을 만들어준다. 즉, 시스템의 경계를 보여주고 시스템 밖에는 무엇이 놓이며 어떻게 시스템을 사용하는가를 보여준다. 사용사례 다이어그램은 시스템과 시스템의 행위자들의 행위를 요약해주는 의사소통 도구로서 역할을 한다.

-다이어그램 그리기 제시

<<actor>> 이 기호는 UML 에서 스테레오타입(stereotype)이라 부른다. 이것은 몇가지 방법으로 요소들의 특징을 기술하는 메커니즘이다.

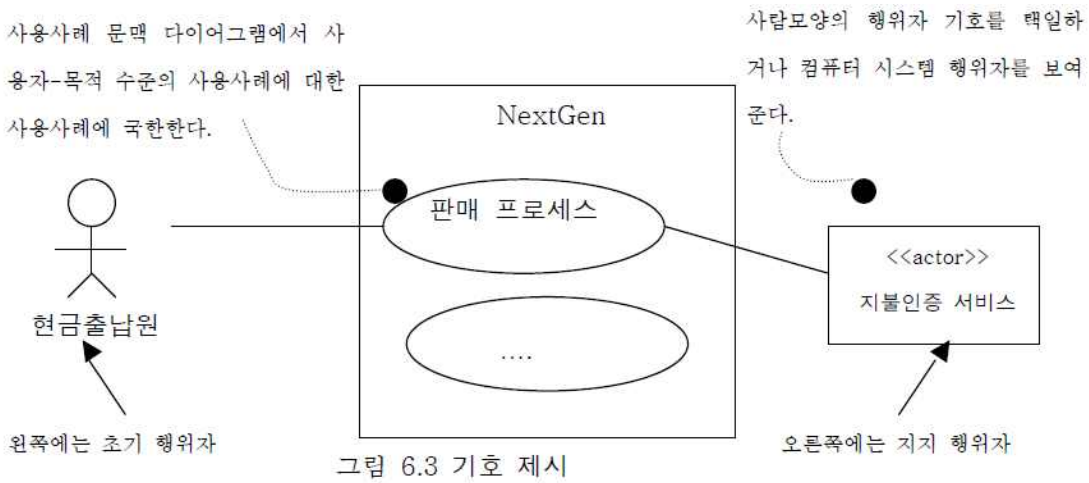


그림 6.3 기호 제시

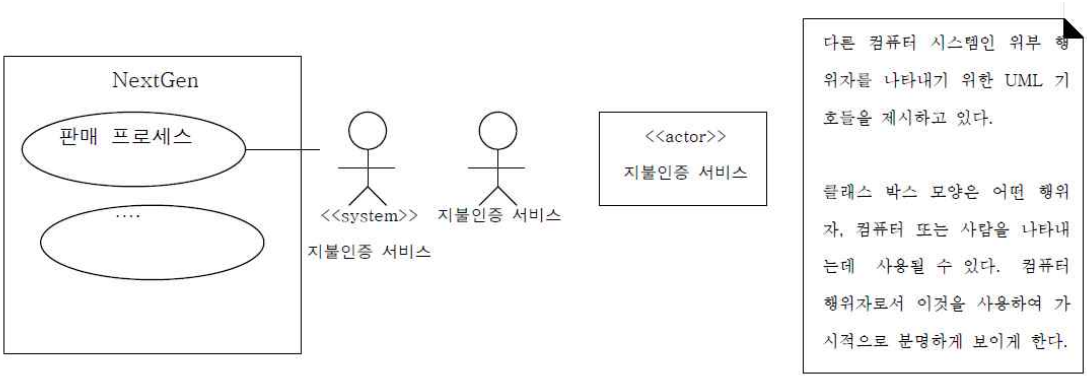


그림 6.4 행위자 표시 기호

-지나친 다이어그램 그리기의 주의점

사용사례 작업에서 가장 중요한 것은 다이어그램이나 사용사례 관계에 초점을 두고 있는 것이 아니라 문맥으로 쓰는 것이라는 것을 반복하여 강조한다. 만일 어떤 조직이 문맥으로 쓰는 것에 초점을 두지 않고 사용사례 다이어그램을 그리고 사용사례 관계를 논의하는데 많은 시간을 소비한다면 상대적인 노력이 잘못 이끌어진 것이다.

IDENTIFYING OTHER REQUIREMENTS

●NextGen 예제: 보충사항 명세화

보충사항 명세화

개정 내역

버전	날짜	묘사	작성자
개념화 도입	2031년 1월 10일	첫번째 초안. 상세화 시기에 처음으로 개량 되어짐	Craig Larman

-소개

이 문서는 사용사례에서 놓친 모든 NextGen POS의 요구사항의 저장소이다.

-기능성

(많은 사용사례에서 기능성이 나온다.)

-입출력 기능과 오류 취급

일관성 있는 저장으로 모든 오류를 저장한다.

-사용할 수 있는 비즈니스 규칙

다양한 시나리오에서 여러 가지 (정의되어질) 사용사례의 요점은 이러한 요점이나 이벤트에서 실행하는 임의의 규칙을 가지고 시스템의 기능을 최적화 하는 능력을 제시한다.

-보안

모든 사용은 사용자의 인증을 요구한다.

-사용성(인적요소)

고객은 POS의 대형 모니터를 볼 것이다. 그러므로

① 모니터의 내용이 1미터에서 쉽게 보여야 한다.

② 색맹에게 영향을 주는 형태의 색깔들을 피하라.

③ 고객은 빨리 처리되기를 바라고 적합하지 못한 처리를 인식하므로 빠르기, 편안함 그리고 오류 없는 처리들은 판매 프로세스에서 가장 우선한다.

④ 출납원은 컴퓨터의 화면뿐만 아니라 물품들을 바라본다. 그러므로 신호나 경고등이 그래픽을 통해서 뿐만 아니라 소리로도 전달되어야만 한다.

-신뢰성(회복성)

만약 외부의 서비스(지불 인증 시스템, 회계 시스템,…)들을 이용하는 것에 실패하면 판매를 계속 수행하기 위해 자체 해결을(예를 들어. 저장과 진행) 하도록 시도한다. 여기에서 더 많은 분석이 요구된다.

-수행성

인적 요소에서 언급했던 것처럼 고객은 매우 빠르게 판매처리를 수행하기를 원한다. 한가지 잠재적인 애로사항은 외부 지불 인증이다. 우리의 목적은 1분 안에, 1분의 90% 안에 인증을 받는 것이다.

-지지성(적응성)

NextGen POS의 여러 고객들은 판매 프로세스 동안 하나의 비즈니스 규칙과 처리 요건들을 가지고 있다. 그러므로 시나리오(예를 들어, 새로운 판매가 시작될 때, 새로운 물품 목록이 더해질 때)에서 정의된 여러 가지 요점들에서 사용할 수 있는 비즈니스 규칙이 사용되어질 것이다.

-Configurability(구성성)

여러 고객들은 그들의 POS시스템이 위해서 다양한 기능의 클라이언트와 여러 물리 계층과 같은 네트워크 구성이 이루어지기를 바란다. 게다가 자신들의 변화하는 비즈니스와 수행 요구들을 반영할 수 있도록 이러한 구성들을 수정하는 능력을 갖추기 원한다. 그러므로 그 시스템은 이러한 필요 사항들을 반영할 수 있도록 구성되어질 것이다. 유연성의 정도와 범위를 발견하기 위해서 더 많은 분석이 이 분야에서 필요하고 이것을 이루는 노력이 필요하다.
구현 제약 조건 NextGen 지도부는 자바 기술 솔루션이 개발의 편리성과 더불어 장기간 도입되어 지지될 것으로 예측하며 이 솔루션을 주장한다.

-구입 컴포넌트

세금 계산기. 다른 나라에서도 사용할 수 있는 계산기들을 지원해야 한다.

-인터페이스

주목할 하드웨어와 인터페이스

① 바코드 레이저 스캐너(이것들은 보통 특수한 키보드에 부착되고 스캐너에 의한 입력은 타자를 친 것처럼 소프트웨어에서 인식된다.)
② 터치 스크린 모니터(이것은 보통의 모니터처럼 운영 체제와 마우스 이벤트 같이 만지는 손짓에 의해 인식된다.)
③ 영수증 프린터
④ 신용/데빗 카드 리더
⑤ 서명 리더

소프트웨어 인터페이스

대부분의 외부의 합작 시스템들(세금 계산기, 회계, 재고조사,…)을 사용하기 위해 다양한 시스템과 다양한 인터페이스를 플러그 인 할 필요가 있다.

-도메인(비즈니스) 규칙

도메인(비즈니스) 규칙

ID	규칙	변환 가능성	소스
규칙1	신용카드 지불은 서명이 요구된다.	구매자의 "서명"은 계속 요구될 것이다. 그러나 대부분의 고객들은 2년 안에 디지털 캡쳐 장치에서 서명 캡쳐하기를 원한다. 그리고 5년 안에 우리는 지금 미국 법에의 해 지지되는 새로운 유일한 디지털 코드 "서명"의 수용을 기대한다.	사실상 모든 신용 인증 회사들의 정책
규칙2	세금 규칙. 판매는 부과된 세금을 요구한다. 현재의 세부 사항을 알기 위하여 정부의 법령을 보라.	높음. 세법은 매년 정부의 기준에 의해서 바뀐다.	법
규칙3	신용지불 파기는 현금이 아니라 구매자의 신용 예금 계좌에서 신용카드로만 변제 가능하다.	낮음	신용 인증 회사 정책
규칙4	구매자 할인 규칙. 예: 고용인-20%할인 단골고객-10%할인 노인-15%할인	높음. 각 소매인은 서로 다른 규칙들을 사용한다.	소매인 정책
규칙5	판매 (거래 등급) 할인 규칙들. 세금이 포함된 총액 사용하기. 예: 미화 100달러 이상 10%할인 매주 월요일에는 5%할인 금일 10am-3pm은 모든 판매에서 10% 할인 . 금일 9am-10am 에는 제한적 50%할인	높음. 각 소매인은 서로 다른 규칙을 사용하며 이들은 매일 혹은 매 시간마다 규칙을 바꿀 수 있다.	소매인 정책.
규칙6	제품(재고 목록 등급) 할인 규칙들. 예: 금주에는 트랙터들을 10%할인 2개의 야채 버거를 사면 1개 공짜	높음. 각 소매인은 서로 다른 규칙들을 사용하며 이들은 매일 혹은 매 시간마다 규칙을 바꿀 수 있다.	소매인 정책.

-법적 문제

만약 오픈 소스 컴포넌트의 라이선스 제약조건들이 오픈 소스 소프트웨어가 포함된 제품들을 재판매를 할 수 있도록 허락하는 문제가 해결된다면 우리는 이러한 오픈 소스 컴포넌트를 추천한다.
모든 세금 규칙은 법에 의해 판매 시기에 적용되어야만 한다. 이 규칙들은 자주 바뀔 수 있음을 주의하라.

-관심 있는 도메인 정보(가격)

도메인 규칙 영역에서 기술된 가격 규칙들 이외에 제품들은 원래 가격이 정해져 있으며 영구히 정찰가로 판매되는 것이 있다는 것을 주의 하라. 한 제품의 가격이(더 할인하기 전에) 제시되면 영속적인 정찰가를 가진다. 가격 인하가 있을 지라도 회계와 세금의 이유로 인하여 조직들은 원래가격을 유지한다.

-신용카드와 데빗카드 취급

전자 신용카드와 데빗카드 지불이 지불 인증 서비스에 의해 인증되면 지불인증 서비스는 구매자가 아니라 판매인에게 지불할 책임이 있다. 결과적으로 각각의 지불에 대하여 판매인은 인증 서비스로부터 자신들이 받을 수 있는 계좌로 소유권을 얻은 돈을 기록할 필요가 있다. 보통 밤을 기초로 하여 인증서비스는 매일 전체 갚을 돈에 대하여 매번 약간의 처리 비용을 부과하면서 구매자의 계좌에서 전자자금을 보내는 일을 수행한다.

-판매 세금 계산

판매 세금 계산들은 매우 복잡 할 수 있고 정부 차원에서 법령에 따라 규칙적으로 변화 한다. 그러므로 제 삼자의 계산 소프트웨어(여러 가지 사용할만한 소프트웨어가 있다.)에 세금 계산을 위임하는 것은 타당하다. 세금은 도시, 지역 ,주 그리고 나라 전체의 소유가 된다. 여러 가지 물품들은 무조건적으로 면세될 수 있다. 또는 소비자나 지정 수납자(예를 들어, 농부나 아이)에 따라서 면세될 수 있다.

-물품 확인: UPCs, EANs, SKUs, 바코드, 바코드 판독기

NextGen POS는 다양한 물품 확인기구를 지원할 필요가 있다. UPCs(Universal Product Codes),EANs(European Article Numbering) 그리고 SKUs(Stock Keeping Units)는 상품화된 있는 제품을 확인하는 일반적인 시스템들이다. Japanese Article Numbers(JANs)는 일종의 EAN 버전이다.
SKUs는 완전히 소매 상인들에 의해 임의로 정의된 확인 시스템이다.
그러나 UPCs와 EANs는 표준화되고 규정된 컴포넌트를 가지고 있다.

●시스템의 질적 속성

몇 가지 요구 사항들은 시스템의 질적 속성으로 불린다. 이것들은 유용성, 신뢰성 등을 포함한다. 질적 속성은 반드시 높은 질이 아닌 시스템의 특성들을 언급하는 것을 주의하라. 예를 들어, 지지성의 질은 그 제품이 장기적인 목적을 제공하도록 의도되지 않는다면 마땅히 낮은 상태로 선택 될 것이다.

질적 속성에는 두 가지 타입이 있다.
1. 실행에서 관찰되는 형태(기능성, 유용성, 신뢰성, 수행성, …)
2. 실행에서 관찰되지 않는 형태(지지성, 검사성, …)
다른 질적 속성들이 사용 사례를 구체화 시키는 것과 관련 있는 것처럼 기능성은 사용 사례에서 구체화 된다. (예를 들어, 판매 프로세스 사용 사례에서 수행 특성들).

그 밖에 시스템에 영향을 미치는 FURPS+ 질적 속성은 보충 사항 명세화에서 기술 된다. 기능성이 보통의 경우에는 타당한 사용에서 정당한 질적 속성을 뜻하지만 “질적 속성”이라는 용어는 때때로 “기능성 보다는 시스템의 질”을 내포하는 의미를 가지고 있다. 여기에서 용어의 의미는 이와 같이 사용된다. 이것은 비기능적인 요구 사항들과 정확하게 같은 것이 아니다. 이것은 기능성(예를 들어, 패키징 과 라이선스 받기)이외에도 모든 것을 포함하는 넓은 용어이다.
우리가 아키텍트의 입장에 서 있으면 시스템에 미치는 질적 속성(그러므로 사람이 기록한 보충사항 명세화)에 특별히 흥미가 있다. 왜냐하면-32장에서 소개되어 지듯이-아키텍처 분석과 설계는 기능 요구사항의 문맥에 있는 질적 속성의 확인과 해결책에 관련이 있다. 예를 들어, 질적 속성들의 하나가 원격 서비스들이 실패할 때 NextGen 시스템은 반드시 결함 허용을 버려야 한다고 가정해 보자. 아키텍처의 관점에서 그것은 큰 규모의 설계 결정에 지배적인 영향을 줄 것이다.
질적 속성은 상호 의존성을 가지며 거래와 관련된다. POS에 있는 간단한 예로서 “매우 신뢰할 수 있는(결함허용)”과 “테스트하기 쉬운”이라는 말은 약간 상반된다. 왜냐하면 분산 시스템은 실패할 수 있는 많은 미묘한 방법이 있기 때문이다.

●도메인(비즈니스)규칙

도메인 규칙은[Ross97, Gk00] 어떻게 도메인이나 비즈니스가 운영되는지를 제시한다. 이러한 규칙은 어플리케이션의 요구 사항이 자주 도메인 규칙에 의해 존재하지만 어떤 한 어플리케이션의 요구 사항이 아니다. 회사의 정책, 물리적인 법칙 그리고 정부의 법 등이 일반적인 도메인 규칙이다.
도메인 규칙은 보통 비즈니스 규칙 이라고 불린다. 그러나 일기 시뮬레이션이나 군 병참학과 같은 어떤 소프트웨어 어플리케이션은 비즈니스가 아닌 문제인 것처럼 이 용어는 제한적이다. 일기 시뮬레이션은 물리적인 법칙과 관계성에 관계된 어플리케이션 요구 사항에 영향을 주는 “도메인 규칙”을 가진다.
요구사항에 영향을 주는 이러한 도메인 규칙은 보통 사용사례를 현실화 하고 애매모호한 사용사례 문맥을 명확하게 해주기 때문에 도메인 규칙을 확인하고 기록하는 것이 필요하다.
예를 들어, NextGen POS에서 판매 프로세스 사용사례를 서명 없이 신용카드 지불을 허락하는 것으로 양자 택일하여 써야 하는지를 누군가가 요구하면 여기에는 어떤 신용 카드 인증회사에 의해 승인 되지 않는다는 것을 분명히 해주는 비즈니스 규칙(RULE1)이 적용된다.

-주 의

규칙은 어플리케이션 요구 사항이 아니다. 시스템 특징을 규칙으로서 기록하지 말아야 한다. 규칙은 도메인이 어떻게 작동하는지에 대한 행위와 제약 조건을 기술한다.

-관심 있는 도메인 정보

주제 설정 개발 팀을 위해 더 깊은 통찰력과 전후관계를 제공하기위해 새로운 소프트웨어시스템(판매와 회계, 땅속의 기름/물/가스흐름의 지구 물리학,…)과 관련 있는 도메인에 대하여 여러 가지 설명을 쓰는 것(또는 URLs를 제공하는 것)은 가치가 있다. 이것은 중요한문학 또는 전문가, 공식, 법칙 또는 다른 참고 자료 등을 가리키는 것도 포함된다. 예를 들어, UPC의 애매성, EAN 코딩 체계, 그리고 바 코드 기호론 등은 어느 정도 NextGen 팀이 이해 시켜야 한다.

● NextGen 예제: (부분적인) 비전

비 전

수정내역

버전	날짜	묘사	작성자
초안	2031년 1월 10일	첫번째 초안. 상세화 시기에 처음으로 개량 되어짐	Craig Larman

-소개

다양한 고객, 비즈니스 규칙, 다중 터미널과 사용자 인터페이스 메커니즘과 다양한 제 삼자 지지 시스템으로서 통합 등을 지원할 수 있는 유연성을 지닌 차세대의 결함허용 포스(POS) 어플리케이션인 NextGen POS를 계획한다.

-입장 정하기 (Positioning)

다양한 비즈니스 규칙과 다양한 네트워크 설계(예를 들어, 클라이언트이든 아니든; 2,3,4 계층의 아키텍처)면에서 볼 때 현존하는 POS제품들은 고객의 사업에 적합하지 않다. 더욱이 이들 제품은 터미널과 비즈니스가 증가함에 따라 잘 적용할 수 없다. 그리고 네트워크 연결 실패에 동적으로 적응하여 온라인이나 오프라인 어느 모드에서나 다 작동하는 제품이 없다. 어떤 것도 쉽게 많은 제 삼의 시스템을 합병시키지는 못한다. 어느 것도 모바일 PDAs와 같은 새로운 터미널 기술들을 지원할 수 없다. 이와 같이 업무 처리에 유연하지 못하여 시장성에서 불만족스럽다. 이에 따라 POS 시스템의 개선이 요구된다.

-문제제기

전통적인 POS 시스템들은 불변적이고 결함허용이 안되며 제 삼의 시스템과 통합하는 것이 어렵다. 이에 따라 소프트웨어와 일치하지 않는 프로세스를 개선 시켜야 하고 정확하고 적기에 맞는 회계와 평가 및 계획을 지원할 수 있도록 고안된 데이터 등의 적절한 판매 프로세스에 관한 문제가 야기된다. 이러한 문제는 출납원과 점원 관리자들 그리고 시스템 관리자들과 법인의 경영자들에게 영향을 준다.

-제품 특성

시스템이 누구를 위한 것인지, 시스템의 현저한 특징은 무엇인지 그리고 경쟁사와 본 시스템의 차이점은 무엇인지를 간결하게 요약한다.

-사용자 수준의 목적

사용자들(그리고 외부의 시스템들)은 다음의 목표들을 수행하는 시스템을 필요로 한다.

① 출납원. 판매 프로세스, 반품 처리, 현금 입금, 현금 출납
② 시스템 관리자. 사용자 관리, 보안 관리, 시스템 테이블 관리
③ 관리자. 시작하기, 종료하기
④ 판매 활동 시스템: 판매 데이터 분석
⑤ 사용자 환경

높은 수준의 목적	우선 순위	문제점과 관심사	현재의 해결책
빠르고, 견고하고 통합된 판매 프로세스	높음	시스템의 로드가 증가함에 따라 속도 감소. 만약 컴포넌트 접속이 실패하면 판매 프로세스 능력을 상실한다. 현재의 회계와 재고품 조사, 그리고 HR 시스템 등과 통합되지 못하여 회계 시스템과 다른 시스템에서 오는 최신의, 정확한 정보의 결핍. 독특한 비즈니스 요구 사항에 비즈니스 규칙을 최적화 시키는 능력 부족. 새로운 단말장치와 사용자 인터페이스 타입 등을 추가 시키기 곤란함.(예를 들어, 모바일 PDAs).	현재의 POS 제품들은 기본적인 판매 프로세스를 제공하지만 이러한 문제점들을 참조하지 못한다.
...	...	...	...

-제품 개요(제품의 전망)

NextGen POS는 보통 점포에서 사용된다. 만약 이동 단말기를 사용한다면 단말기는 건물 내부 또는 외부의 장소로서 네트워크 가까이에 놓일 것 이다. 이것은 사용자들에게 서비스를 제공 할 것이며 그림 비전 - 1 과 같이 다른 시스템들과 협력할 것이다.

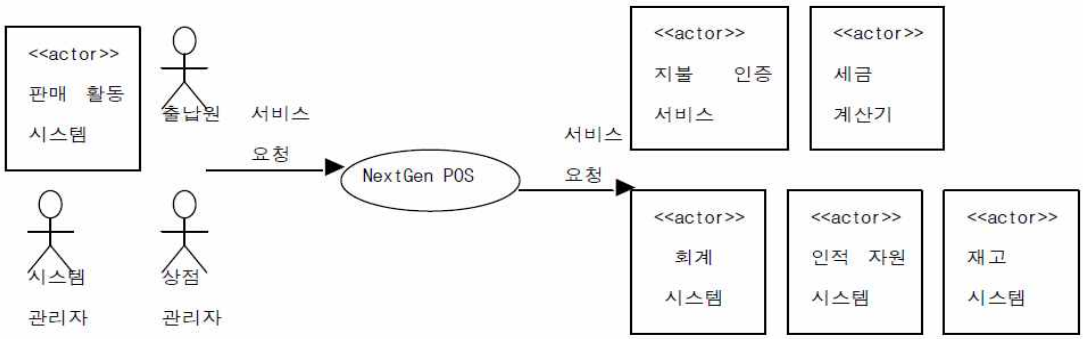


그림 비전 - 1. NextGen POS 시스템 컨텍스트 다이어그램

-제품의 이점 요약

지지하는 특징	제 삼자의 이점
기능적으로 시스템은 모든 보통의 서비스로서 판매 사항 기록, 지불 인증, 반환 취급 등을 포함한 판매 조직의 요구 사항들을 제공할 것이다.	자동화된, 빠른 판매 서비스
실패에 대한 자동 탐지, 가능하지 않은 서비스들에 대해 자체적 오프라인 처리로 대체	외부의 컴포넌트가 작동하지 않을 때 계속되는 판매 처리과정
판매 처리 동안 다양한 시나리오 시점에서 사용 가능한 비즈니스 규칙들	유연한 비즈니스 로직 구성
산업 표준 프로토콜을 사용하여 제 삼의 시스템과	적시에, 정확한 판매, 회계 그리고 재고

-시스템 특징의 요약

① 판매 사항 기록
② 지불 인증(신용, 데빗, 수표)
③ 사용자, 보안, 코드와 상수 테이블 등에 따라 시스템 관리
④ 외부 컴포넌트들이 실패할 때 자동 오프라인 판매처리
⑤ 재고, 회계, 인적 자원, 세금 계산기 그리고 지불 인증 서비스 등을 포함한 제 삼의 시스템들과 함께 산업 표준에 기초를 둔 실시간 처리
⑥ 판매 과정에서 일정한 공통점으로 최적화된 사용 가능한 비즈니스 규칙의 정의와 실행

●신뢰할 수 있는 명세화: 모순어법

쓰여진 요구 사항들은 실제 요구사항들을 이해하고 잘 정의해서 착각을 일으킬 수 있다. 그리고 (일찍) 신뢰성 있게 평가하고 프로젝트를 계획하데 사용 되어질 수 있다는 착각을 불러일으킬 수 있다. 이 착각은 비 - 소프트웨어 개발자에게는 더 사실적이다; 프로그래머들은 이것이 얼마나 믿을 수 없는 것인지를 아픈 경험을 통해 알고 있다. 이것은 괴테에 의해 쓰여진 시작 인용에 대한 동기 부여의 일부분이다.

실제 문제들은 사용자들과 제 삼자들에 의해 정의된 승인 검사를 통과하는 소프트웨어를 만드는 것이며 그들의 진정한 목적을(그들이 평가하거나 그 소프트웨어와 함께 작동할 때까지 자주 발견되지 않는) 만족 시키는 소프트웨어를 만드는 것이다.

비전과 보충사항 명세화를 작성하는 것은 원하는 것에 대한 대략적인 첫 번째 결과를 분명히 하는 연습으로서, 제품에 대한 동기부여로서, 그리고 큰 아이디어들에 대한 저장소로서 가치가 있다. 그러나 이것들은 - 어떤 요구 사항들의 산출물도 아니며 - 믿을만한 명세화가 아니다. 다만 코드를 쓰고, 테스트하고, 피드백을 얻고, 사용자와 고객이 함께 계속적으로 밀접한 협력을 하고, 그리고 적응하는 것이 진정한 목적이 된다.

이것은 분석하는 것과 생각하는 것을 버리고 코드로 돌진하라는 요청이 아니라 쓰여진 요구 사항들을 가볍고 지속적으로 - 실제로 날마다 - 취급하고 사용자들을 관여 시키라는 제안이다.

●개념화 시기에는 UML을 적게 사용하는가

개념화의 목적은 일반적인 비전을 세우기 위해서 충분한 정보를 모으고, 또한 앞으로 작업을 진행하는 것이 실현 가능한지와, 프로젝트가 상세화 단계에서 진지한 조사를 할 가치가 있는지를 결정하기 위하여 충분한 정보를 모으는 것이다. 따라서 간단한 UML 사용 사례 다이어그램들을 넘어서서 많은 다이어그램을 작성하는 것은 도움이 되지 않는다. 원문의 형태에 있는 요구 사항들의 10%와 기본적인 범위를 이해하는데 개념화의 초점이 있는 것이다. Practice에서, 이러한 요구사항을 프리젠테이션 하기 위해서 대부분의 UML 다이어그램은 다음 단계인 상세화 단계에서 사용될 것이다.

FROM INCEPTION TO ELEBORATION

●체크포인트: 개념화에서 무슨 일이 이루어지는가

NextGen POS 프로젝트에서 개념화 단계는 단지 일주일 정도 지속된다. 생성된 산출물들은 간단하고 미완성이며 빠른 단계이고 가볍게 조사하는 단계이다.

프로젝트는 요구사항 단계가 아니며 기본적으로 적합성, 위험요소, 그리고 범위를 결정하는 짧은 단계이고, 프로젝트가 상세화 단계에서 더 진지하게 조사할 가치가 있는지를 결정하는 단계이다. 개념화에서 타당하게 일어날 수 있는 모든 활동들이 완료되는 것이 아니다; 이것의 상세화는 요구사항-지향 산출물을 강조한다. 상세화 단계에서 몇 가지 활동과 산출물들은 다음을 포함한다:

① 간단한 요구사항 워크숍
② 대부분의 행위자, 목적, 그리고 사용사례 이름
③ 대부분 개요 형식으로 쓰여진 사용사례; 범위와 복잡성의 이해를 개선시키기 위해서 사용사례의 10-20%가 개요형식으로 쓰여진다.
④ 가장 영향을 주고 위험 요소가 있는 질적 요구사항 확인
⑤ 비전과 요구사항 명세화가 작성된 초기 버전
⑥ 위험요소 목록
⑦ 기술적으로 개념이 입증된 프로토타입과 특이 요구사항의 기술적 적합성을 알기 위한
⑧ 그 밖의 조사(자바 스윙이 터치 스크린에서 적합하게 작동하는가
⑨ 기능적 요구사항의 비전을 명료하게 하기위한 사용자 인터페이스 지향 프로토타입
⑩ 어떤 컴포넌트를 구매하고/만들고/재사용할 것인지를 추천, 상세화에서 재고되어짐
-예를 들어, 세금계산 패키지는 구매할 것을 추천
⑪ 높은-수준의 아키텍처 후보와 컴포넌트 제시
- 이것은 상세한 아키텍처의 표현이 아니며 마지막이 아니고 옳게 설정된 것이 아니다. 오히려 상세화 단계에 있어서 조사 시점에서 사용할 간단한 고찰인 것이다. 예를 들어, “서버 어플리케이션이 아니라 Java 클라이언트쪽 어플리케이션, 데이터베이스에 대하여는 Oracle, …” 상세화 단계에서 이것은 가치 있다는 것이 입증될 수 있으며 잘못된 생각으로 거절될 수도 있다.
⑫ 첫번째 반복 계획
⑬ 툴 목록

●개념화 들어가기

개념화는 팀원이 진지하게 조사하고 핵심 아키텍처를 구현(프로그램과 테스트)하고 대부분의 요구사항을 명료하게 하며 높은-위험요소에 관한 이슈를 다루는 일련의 초기 반복이다.
UP에서 “위험요소”는 비즈니스의 가치를 포함한다. 그러므로 초기 작업은 중요하다고 생각되어지는 구현 시나리오를 포함할 수 있으나 특히 기술적으로 위험요소를 다루는 것은 아니다.
상세화는 반복으로 구성된다. 각 반복은 종료 날짜가 고정된 정해진 시간이 있다; 만약 팀이 날짜를 맞추지 못할 것 같으면 요구사항을 다음 업무목록의 뒤에 놓아서 현재의 반복을 안정성 있고 검사된 릴리즈를 가지고 정해진 시간 안에 종료할 수 있게 한다.
상세화는 설계 단계가 아니며 구축 단계에서 구현을 위한 준비로서 모델을 충분히 개발하는 단계이다.

상세화를 한 문장으로 표현:
핵심 아키텍처를 만들고, 높은-위험요소를 해결하며, 대부분의 요구사항을 정의하고, 전체 스케줄과 자원을 추정한다.

상세화에서 명시될 중심 아이디어와 최선의 작업내용은 다음을 포함한다.
① 짧은 시간이 정해진 위험요소 위주의 반복을 시행하라.
② 초기에 프로그래밍을 시작하라 .
③ 아키텍처의 핵심과 위험요소 부분을 적응적으로 설계하고, 구현하고 테스트하라.
④ 초기에 자주 현실적으로 테스트하라.
⑤ 테스트, 사용자, 그리고 개발자로부터의 피드백에 기초를 두고 개작한다.
⑥ 일련의 워크숍을 통하여 한 단계의 상세화 반복에서 대부분의 사용사례와 그 밖의 요구 사항들을 자세하게 써라.

●상세화에서 아키텍처 면에서 중요한 것은 무엇인가

초기 반복에서 핵심 아키텍처를 만들고 입증한다. NextGen POS 프로젝트에서 보듯이 이 단계에서는 다음 내용을 포함한다:

① “넓고 얕은”설계와 구현이 필요하다; Grady Booch가 말한 “이음매를 가진 설계”를 한다.
- 즉, 분리된 프로세스, 계층, 패키지, 그리고 서브시스템 등을 확인하고 이것들의 높은-수준의 책임과 인터페이스를 확인한다. 이것들을 연결하고 인터페이스를 분명하게 하기 위해서 부분적으로 구현한다. 모듈은 대부분 “그루터기가 된” 코드를 포함할 수 있다.

② 내부-모듈인 지역 인터페이스와 외부 인터페이스(이것은 가장 상세한 매개변수와 리턴 값을 포함한다)를 개량한다.
- 예를 들어 감싸진 객체에 대한 인터페이스가 제 삼자의 회계 시스템에 접속한다.
- 인터페이스의 첫번째 버전은 완전한 것이 아니다. 테스트, “침입”, 그리고 인터페이스의 개량을 강조하고자 하는 초기 의도는 안정된 인터페이스에 의존하여 나중에 여러 팀이 분산 작업을 하는 것을 지원하기 위함이다.

③ 현재의 컴포넌트들을 통합한다.
- 예를 들어 세금계산기.
  많은 중요한 컴포넌트를 사용하여 설계, 구현, 그리고 테스트를 이끌어갈, 끝에 끝을 붙인 시나리오를 제공한다.
- 예를 들어 신용카드 지불의 확장 시나리오를 사용하는 판매 프로세스의 주요 성공 시나리오.

상세화 단계에서 테스팅은 피드백을 얻고 개량하고 핵심 아키텍처가 견고하도록 입증하기 때문에 중요하다. NextGen 프로젝트에 대한 초기 테스트는 다음을 포함한다:

① 판매 프로세스에 대한 사용자 인터페이스의 사용성 테스트.
② 신용 인증과 같은 원거리 서비스가 실패했을 때 회복성 테스트.
③ 원거리 세금계산의 로드와 같은 원거리 서비스의 높은 로드에 대한 테스트.

●반복 1 요구사항과 강조되는 것: 기본적인 OOAD 기술

상세화 단계의 반복 1에서는 객체에 책임을 할당하는 것과 같이 객체 시스템을 만드는데 사용되어지는 기본적이고 일반적인 OOAD 기술을 강조한다. 물론, 많은 다른 기술-데이터베이스 설계, 사용성 공학, 그리고 UP 설계와 같은 기술-과 단계들은 소프트웨어를 만드는데 필요하나 OOAD 소개와 UP에서 이것들은 범위를 벗어난다.

-반복 1 요구사항

NextGen POS 어플리케이션의 첫번째 반복의 요구사항은 다음과 같다:

① 판매 프로세스 사용사례의 기본적인 주요 시나리오 제공: 아이템 들어가기와 현금 지불 받기.
② 각 반복의 초기 요구를 지원하는데 필요한 시작 사용사례 제공.
③ 공상적이거나 복잡한 것을 취급하지 않고 간단하고 만족스러운 시나리오와 이것을 지원하는 설계와 구현을 제공.
④ 세금 계산기나 생산 데이터베이스와 같은 오부 서비스와의 협력은 없다.
⑤ 복잡한 가격 규칙은 적용되지 않는다.

UI를 지원하는 설계와 구현은 이루어져야 하지만 완전히 이루어지는 것은 아니다. 이것을 기초로 하여 다음의 반복이 이루어진다.

-반복을 통하여 똑 같은 사용사례에 대한 점진적 개발

판매 프로세스 사용사례에서 모든 요구사항이 반복 1에서 취급되는 것이 아니다. 여러 반복에서 똑 같은 사용사례를 가지고 변하는 시나리오와 특징에 대해 작업하며 궁극적으로 기능적으로 요구되는 모든 것들을 취급하기위해 점차적으로 시스템을 확장한다. 한편 짧고 간단한 사용사례는 한 번의 반복에서 완성되어질 수 있다.

UML이 무엇이며 왜 중요한가?

UML은 소프트웨어 시스템이나 업무 모델링 그리고 기타 비 소프트웨어 시스템 등을 나타내는 가공물을 구체화하고, 시각화하고, 구축하고, 문서화하기 위해 만들어진 언어이다. UML은 Rational Software와 그 협력회사에 의해 개발되었다. 업무 처리과정에서 그 업무의 범위와 규모가 커짐에 따른 시스템의 복잡성을 처리할 필요성을 느끼게 되었는데, 특히 물리적인 시스템의 분산, 동시성, 반복성, 보안, 결점 보완, 시스템들의 부하에 대한 균등화와 같은 반복해서 발생하는 구조적 문제에 대한 프로세스가 필요하게 되었다. 또한 웹의 발전에 따라 시스템을 만들기는 쉬워졌으나 이러한 구조적 문제는 더욱 악화되었기에 이러한 모든 필요성에 의해 UML은 만들어졌다.

●모델링의 중요성

시스템의 복잡성이 증가함에 따라, 강력한 소프트웨어 시스템을 만들기 위해 구축하고 개선하기에 앞서 모델을 만드는 것이 건물을 만들기 위한 청사진을 만드는 것과 같이 핵심적인 요소가 되었다. 잘 만들어진 모델은 프로젝트 팀 간의 통신수단으로써, 구조적인 문제를 해결하기 위한 수단으로써 핵심적인 것이다.

- 모델링 언어가 반드시 포함해야 하는 것

모델 요소(Model Elements) : 기본적 모델링 개념과 의미
표기(Notation) : 모델 요소의 시각적인 그림
가이드 라인(Guide Line) : 관용적인 사용 방법

시스템의 복잡성이 증가함에 따라 객체 지향 시스템과 컴포넌트 기반 시스템을 구축하기 위한 시각적 모델링 언어를 선택하는 것이 필연적이다.

●UML의 목적

사용자에게 즉시 사용가능하고 직관적인 시각적 모델링 언어를 제공함으로써 사용자는 의미 있는 모델들을 개발하고 서로 교환할 수 있어야 한다.
핵심적인 개념을 확장할 수 있는 확장성과 특수화 방법을 제공한다.
특정 개발 프로세스와 언어에 종속되지 않아야 한다.
모델링 언어를 이해하기 위한 공식적인 기초를 제공한다.
협동(Collaboration), 프레임 워크, 패턴, 컴포넌트 같은 고수준의 개발 개념을 제공한다.

●프로그래밍 언어

UML은 비쥬얼 모델링 언어지 비쥬얼 프로그래밍 언어가 아니다. 하나 어떤 의미에서는 시각적이고 의미적인 비쥬얼적 모델링 언어가 제공하는 모든 지원을 가지고 있다. UML은 실제 코드로의 지향을 위해 사용될 수 있다. UML은 객체 언어와 밀접하게 묶여 사용이 가능하고 이는 최고의 결과를 낼 수 있다.

●툴(Tools)

UML은 툴들의 상호 운용성을 위한 툴 인터페이스, 저장소, 실행시간 모델등과 같은 방법을 제공하는 것이 아니라, 의미적 메타 모델(Meta-model)을 제공한다.

UML의 구성

●UML과 방법론의 차이

방법론이란 말 그대로 어떠한 작업을 할 때 이러저러한 절차를 가지고 작업을 하면 된다고 하는 것을 이론적으로 정립을 해놓은 것이다. UML은 이러한 방법론을 적용할 때의 결과물을 나타내기 위한 도구이다. 예를 들면, 모든 소프트웨어를 설계할 때 어떠한 표준적인 규칙을 가지고 설계도를 그려야 하는데 이 때 표준이 되는 것이 UML이다. 각자 다양한 방법론을 자기의 프로젝트에 적용하더라고 UML을 공통적으로 적용할 수 있다.

●UML의 구성

UML은 8가지 다이어그램으로 나타난다.

- 시스템의 정적인 면을 나타냄

Class Diagram

- 시스템의 동적인 면을 나타냄

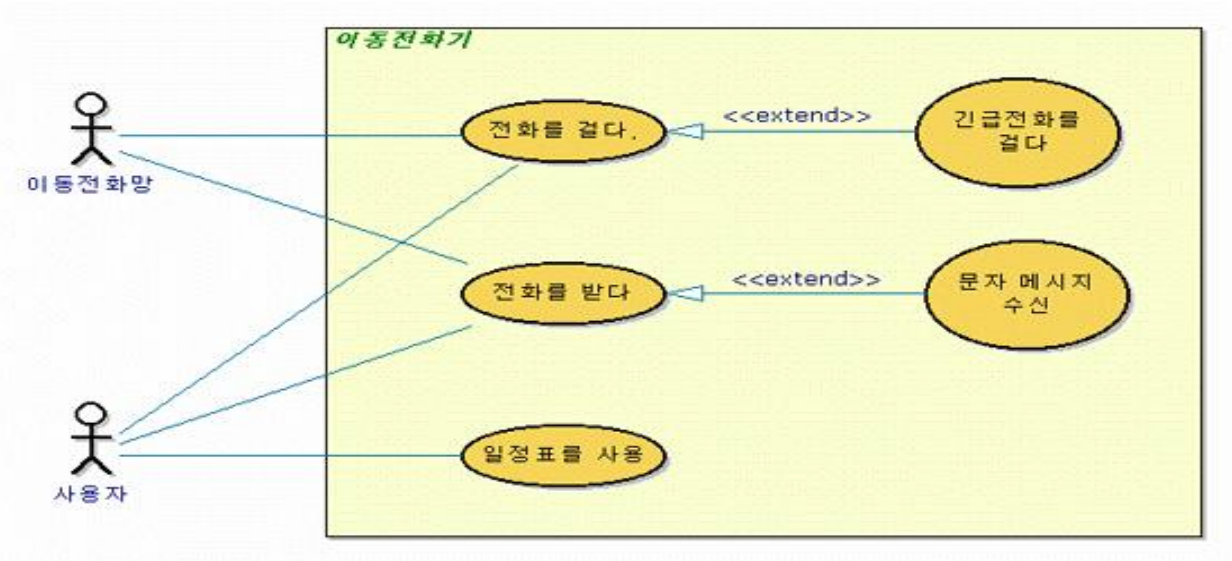
Collaboration Diagram
Sequence Diagram
Statechart Diagram
Activity Diagram
Deployment Diagram
Component Diagram

- 기타

Usecase Diagram

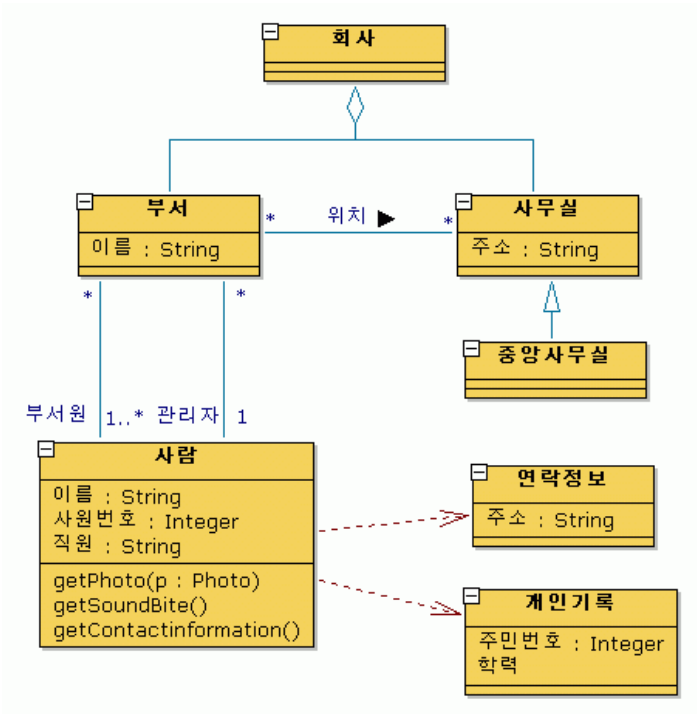
- 유스케이스 다이어그램(Usecase Diagram)

Usecase란 컴퓨터 시스템과 사용자가 상호작용을 하는 하나의 경우이다.
예를 들어 보험처리 프로그램의 경우 "고객이 보험증권에 sign한다", "보험 판매원이 판매 통계량을 종합한다"등이 usecase가 된다.
useacase 다이어그램은 시스템 구축 초기에 이 시스템이 어떤 일을 하는지에 대해 사용자 입장에서 이해할 수 있을 정도로 기술을 해야 된다. 이러한 usecase 다이어그램은 사용자와의 대화수단으로써 앞으로 구축해 나갈 때 밑바탕이 되는 것이다. (밑 그림- next page)



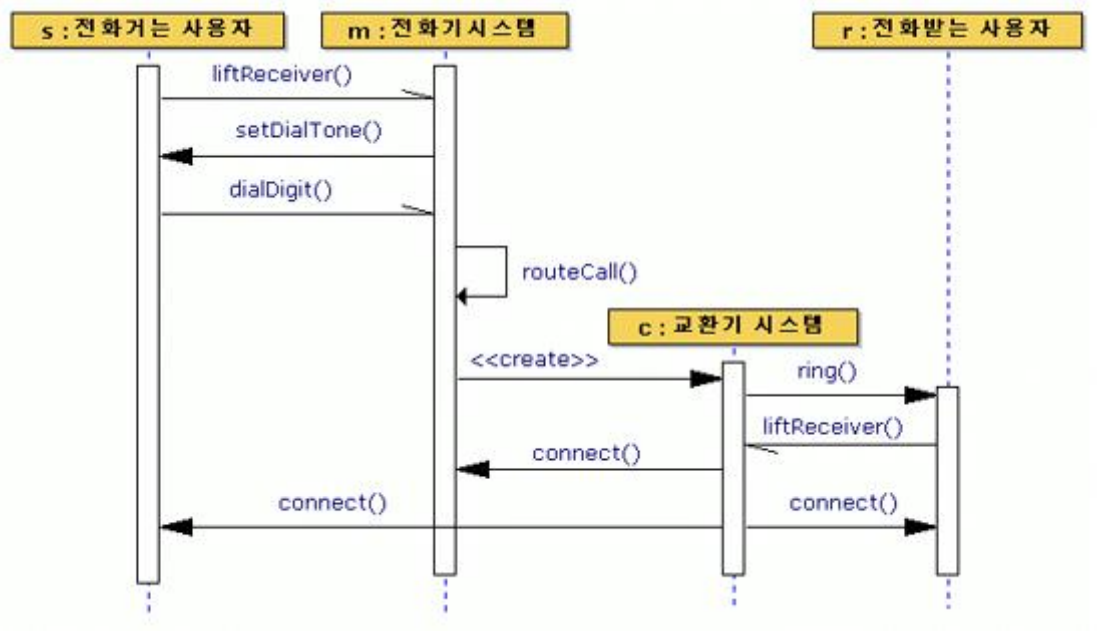
- 클래스 다이어그램(Class Diagram)

시스템 내부에 존재하는 클래스들을 선별하여 나타내고 각 클래스들의 속성과 행위를 기입한다.
클래스들 사이에 여러 가지 관계(Relationship)를 가질 수 있다.
연관 관계(Association) : 클래스와 클래스가 어떤 연관을 가지고 있음을 나타낸다.
복합 연관(Composition), 집합 연관(Aggregation)
이외에 상속관계(Generalization), 의존관계(Dependency)가 나타날 수 있다.
추상화의 단계가 높은 경우 대강의 속성과 행위, 관계를 기입해도 충분하다. 이 단계에서 너무 상세한 내용을 찾고 기입하다 보면 클래스 다이어그램 내부의 구현 단계에서 이루어져야 할 일이 이루어지는 오류를 범하게 된다. 이러한 오류는 실제 구현 단계에서 커다란 위험의 요소를 내재하게 된다.



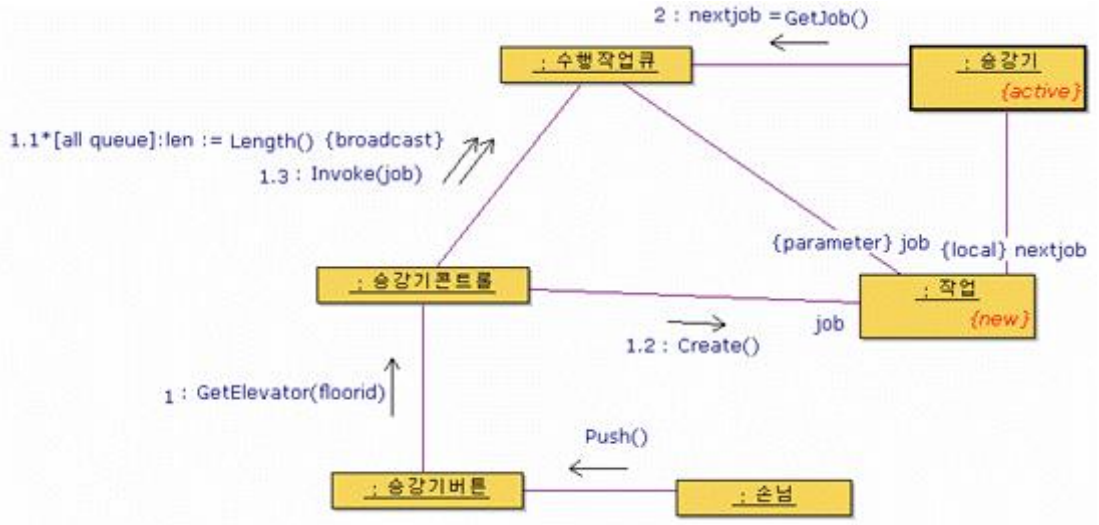
- 시퀀스 다이어그램(Sequence Diagram)

콜레버레이션 다이어그램과 함께 시스템의 동적인 면을 나타내는 대표적인 다이어그램.
시스템 실행 시 생성되고 소멸되는 객체를 표기하고 객체들 사이에 주고 받는 메시지를 나타내게 된다.
콜레버레이션 다이어그램 또한 메시지의 흐름을 나타내지만, 시퀀스 다이어그램만의 특징이라면 횡축을 시간축으로 삼아 시간의 흐름을 나타내 메시지의 순서에 역점을 두고 있다는 것이다.



- 콜레버레이션 다이어그램(Collaboration Diagram)

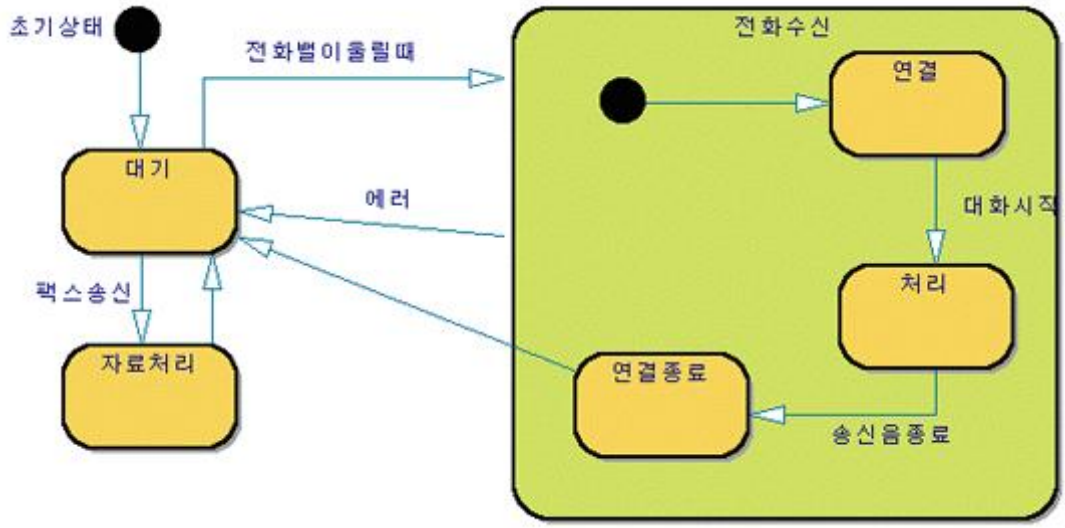
시퀀스 다이어그램과 함께 메시지의 흐름을 나타내지만 객체와 객체들 사이의 관계 또한 표기한다.
클래스 다이어그램과 거의 동일한 오브젝트 다이어그램을 그리기보다는 특별히 클래스 다이어그램과 차이점이 되는 부분을 콜레버레이션 다이어그램에 기입하는 것이 좋다.



- 상태 다이어그램(Statechart Diagram)

한 객체의 상태 변화를 다이어그램으로 나타낸다.

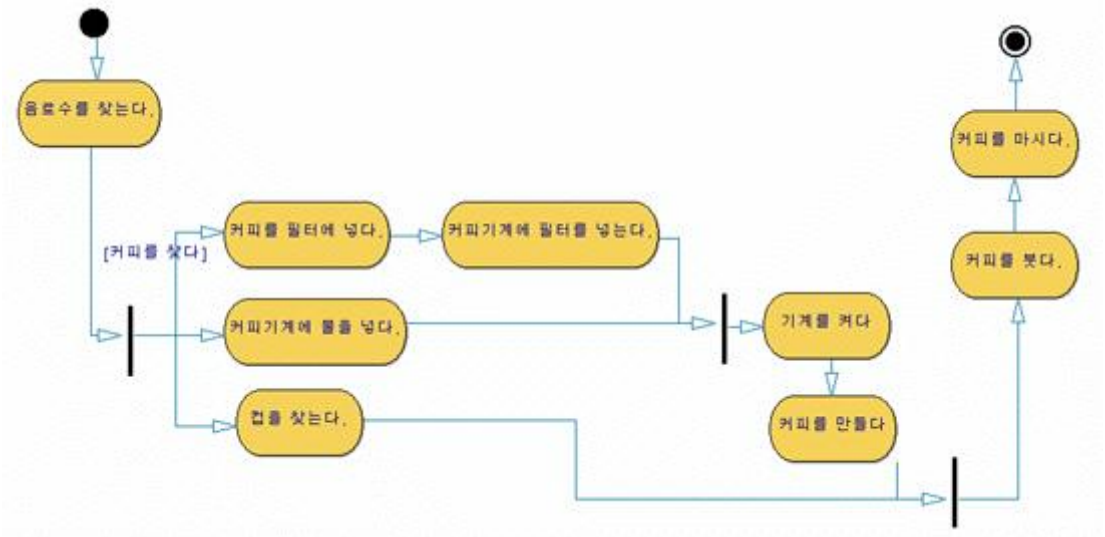
시스템 실행 시 무수한 객체의 상태 전부를 모아 다이어그램으로 나타내는 것은 불가능하므로, 특별히 관심을 가져야 할 객체에 관한, 특정 조건에 만족하는 기간 동안의 상태를 표시해야 한다.



- 액티비티 다이어그램(Activity Diagram)

플로우 챠트가 UML에 접목되어 시스템 내부에 존재하는 여러 가지 행위들, 각 행위의 분기, 분기되는 조건 등을 포함한다.

기존 플로우 챠트와 다른 점은 어떤 행위에 따라 객체의 상태를 표기할 수 있다는 것인데 이것을 제외하곤 기존 플로우 챠트와 표기법과 의미만 약간 다를 뿐이다.

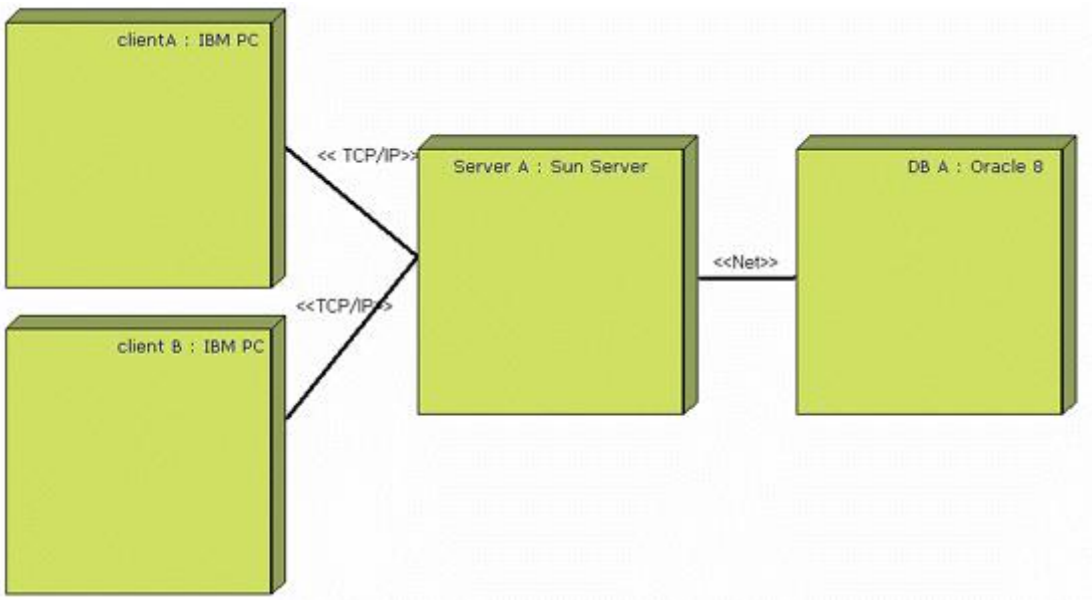


-디플로이먼트 다이어그램(Deployment Diagram) 과 컴포넌트 다이어그램(Component Diagram)

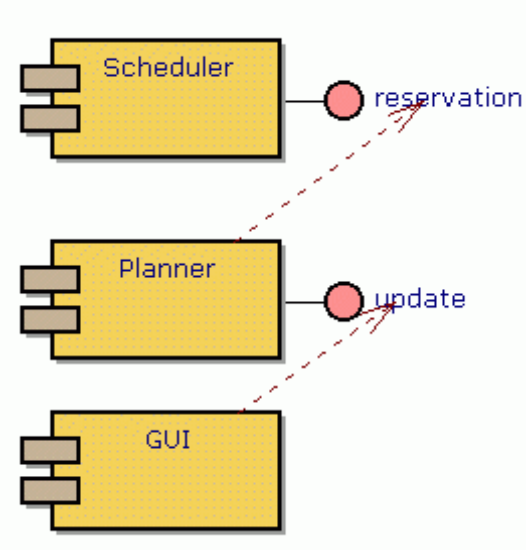
시스템의 물리적인 부분의 구성을 나타낸다.

디플로이먼트 다이어그램은 실제 하드웨어적인 배치와 연결상태를 나타낸다.

컴포넌트 다이어그램은 소프트웨어의 물리적 단위(exe,dll등 library)의 구성과 연결상태를 나타내게 된다.



디플로이먼트 다이어그램(Deployment Diagram)



컴포넌트 다이어그램(Component Diagram)

- 유스케이스 다이어그램 (UseCase Diagram)

프로젝트가 무엇인지 모르고 프로젝트를 수행할 수는 없다.
프로젝트가 무엇인지에 대해서 알아보는 것을 요구 분석(Requirement Analysis)라 하는데, 이를 위한 다이어그램이 바로 유스케이스 다이어그램이다. 즉, 프로젝트 수행 시 가장 먼저 나오는 다이어그램이 유스케이스 다이어그램이고, 다른 다이어그램의 배경이 되는 중요한 다이어그램이다.

표기(Notation)과 의미(Semantics)

유스케이스 (Usecase)

타원으로 표시하고 이름을 타원 안에 명시한다.

유스케이스의 의미

말 그대로 쓰임새를 나타내며, 한 프로젝트의 결과물이 작동하여 사용되는 쓰임새를 분류해 나타낸다.
집이 사용되는 예를 들어보자. 집은 식사를 위한 장소로 사용될 수 있고, 휴식을 위한 장소, 수면을 취하기 위한 장소 등등 여러 가지 용도의 사용 예를 들 수 있다. 이러한 여러 사용 예들이 집의 구조를 결정하는 사항이 될 것이다.

액터 (Actor)

액터를 스틱맨으로 표시하고 하단에 액터의 이름을 명시

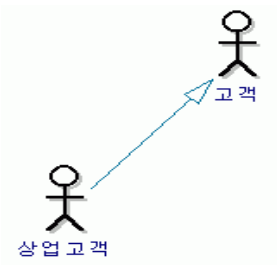
스테레오 타입(Stereotype)을 'Actor'로 가지는 클래스 표기

액터의 의미

액터는 구축해야 할 시스템과 상호 교류하는 추상적인 역할을 수행하는 어떤 사람이나 어떤 것이 될 수 있다. 예를 들어 입출금 ATM기를 보면 입출금을 하는 손님의 경우 하나의 액터가 될 수 있고, ATM기가 입출금 처리를 위해 연결하는 은행의 주 전산망도 하나의 액터가 될 수 있다.

액터들 간의 관계 (Relationship)

일반화(Generalization) 관계의 표기



일반화 관계의 의미

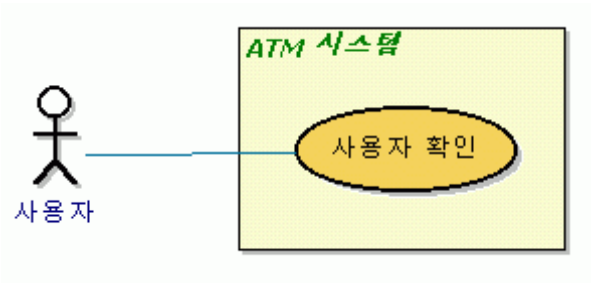
일반화 관계는 객체 지향의 상속의 의미와 유사하다. 일반화된 액터의 모든 특성을 특수한 액터가 모두 가지게 된다.

위 그림과 같이 고객 액터의 모든 특성을 상업 고객이 모두 포함하게 된다.

액터와 유스케이스, 유스케이스와 유스케이스 사이의 관계

(여기서 설명하는 개념은 모두 UML 1.1을 대상으로 설명한 것이다)

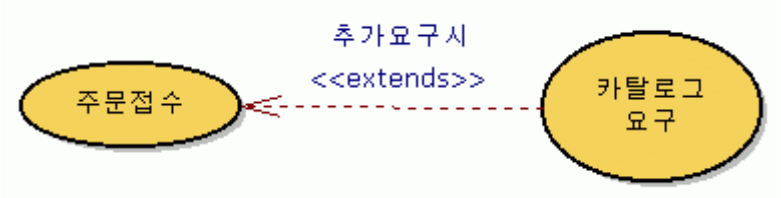
통신(Communicates) 관계



통신 관계의 의미

위 그림에서와 같이 현금 자동 출납기의 시스템에서 그 사용자와 사용자 확인의 유스케이스는 상호작용을 하게 된다. 이를 관계로 표시한 것이 통신 관계이다. 즉 관계로 묶인 두 개체가 상호 작용을 하는 것이 통신 관계이다. (UML 1.3 RTF의 경우 통신관계는 연관(Association) 관계로 대체된다)

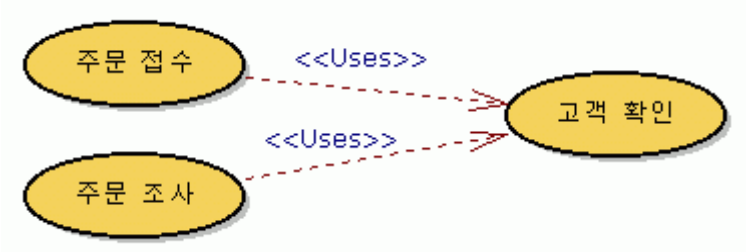
확장(Extends) 관계



확장 관계의 의미

확장 관계는 유스케이스가 어떠한 조건을 만족할 경우 확장할 수 있는 확장 시점(Extends Point)을 가지고 그때의 연관된 유스케이스를 포함하는 관계이다. 위 그림과 같이 추가 요구 시라는 확장 시점에서 카탈로그 요구의 유스케이스가 주문접수의 유스케이스에 포함된다.

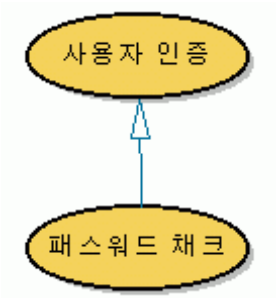
사용 (Uses) 관계



사용 관계의 의미

사용관계는 특정한 유스케이스가 다른 유스케이스를 포함하고 있는 경우를 나타낸다. 위 그림에서 고객 확인의 유스케이스가 주문 접수와 주문 조사의 유스케이스를 모두 포함하게 되는 경우이다. (UML 1.3 RTF에서는 Uses의 관계가 include의 관계로 이름이 바뀌었다)

일반화 (Generalization) 관계



일반화 관계의 의미

액터 사이의 일반화 관계와 동일하게 객체 지향의 상속의 개념과 유사하다.

액터와 유스케이스의 추출법

정확한 액터와 유스케이스를 추출하기 위해서는 여러번의 반복이 필요하지만 처음으로 추출하려는 사람은 다음 지표를 통해 추출해보는 것도 좋다.

액터의 추출법

시스템의 주기능을 사용하는 사람은 누구인가?
누가 시스템으로부터 업무 지원을 받는가?
누가 시스템을 운영, 유지 보수하는가?
시스템과 정보를 교환하는 외부 시스템은 무엇인가?
시스템이 내어놓은 결과물에 누가 관심을 가지는가?

유스케이스 추출법

Actor가 요구하는 시스템의 주요 기능이 무엇인가?
Actor가 시스템의 어떤 정보를 수정, 조회, 삭제, 저장하는가?
시스템이 Actor에게 주는 어떠한 Event가 있는가? Actor가 시스템에 어떠한 Event가 있는가?
시스템의 입력과 출력으로 무엇이 필요한가? 그리고 입력과 출력이 어디에서 오고 어디로 가는가?
시스템의 구현에서 가장 문제가 되는 점은 무엇인가?

시나리오

유스케이스 다이어그램을 그리면서 빠뜨려서는 안될 내용이 시나리오이다.
유스케이스 다이어그램을 완성했다면 유스케이스 다이어그램의 명세가 필요하게 된다. 즉 무엇을 해야 하고 어떻게 해야 하는가에 대한 부연 설명이 필요한 것이다. 유스케이스의 순서에 의해 배열이 가능하고 이러한 순서를 일반적인 자연어 문장으로 표현하되 외부인이 보아도 알기 쉬운 정도로 쉽게 기술해야 한다.

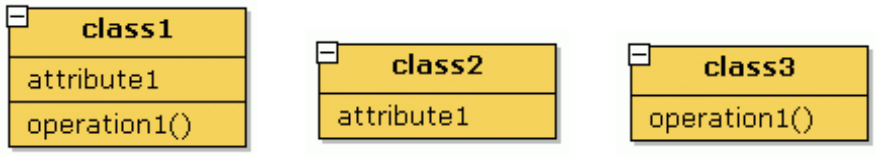
마무리

유스케이스 다이어그램을 잘 그리기 위해 다음 단계로 넘어가는 걸 주저하지 말라. 프로젝트의 성공적인 수행을 위해 반복적 개발을 통한 오류 수정 과정이 필요하고 이는 유스케이스 다이어그램의 수정 또한 포함한다.

어느 정도 유스케이스 다이어그램이 완성되면 다음 다이어그램을 진행하라.

- 클래스 다이어그램(1)

클래스



클래스의 표기는 위 그림과 같다. 제일 좌측에 있는 Attribute와 Operation이 축약되지 않은 표기이고 나머지는 축약된 표기다.

Attribute와 Operation

추상화 단계에 따라 표기 방법이 달라질 수 있다.

예를 들어 구현 단계에 근접한 클래스 다이어그램을 도시하려면 구현하기 위한 언어에 밀접한 형태의 Attribute와 Operation으로 나타내야 하지만, 추상화 단계가 높을 경우 대략적인 의미 전달을 할 수 있을 정도의 표기도 괜찮다.

Attribute UML 1.1 표준 형식은 다음과 같다.

visibility name : type-expression = initial-value { property-string

Operation의 UML 1.1 표준 형식은 다음과 같다.

visibility name (parameter-list) : return-type-expression { property-string }

Visibility를 private는 '-', protect는 "#', public은 "+"로 표기한다.

클래스와 클래스의 연관 관계(Association Relationship)

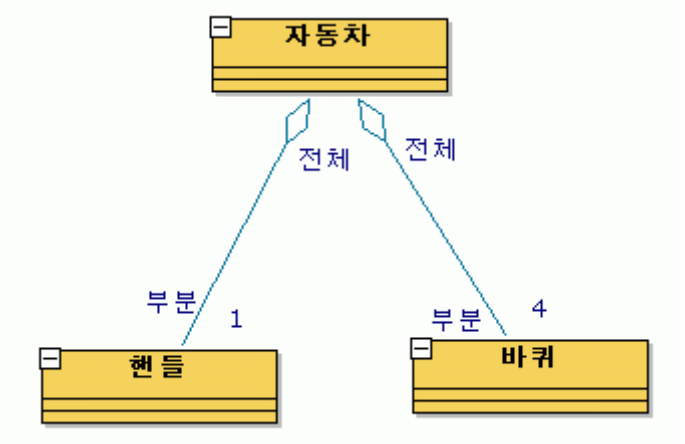
연관 관계의 표기는 실선으로 하게 된다.

예를 들어 회사와 사원은 어떤 식으로든지 연관을 가지고 있는데 이를 표현하기 위해 연관 관계를 사용한다.

장식(Adornments): 표기의 확장을 위해 사용됨

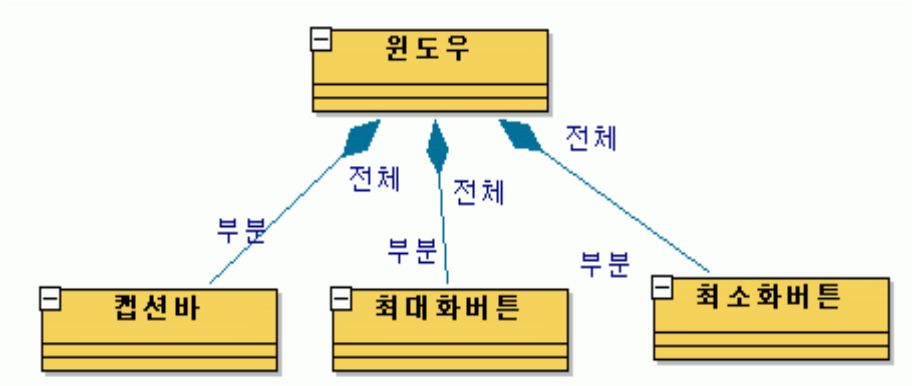
- 연관의 이름 : 어떤 연관인지 명시적으로 나타냄
- 다중성(Multiplicity) : 연관된 상대의 수를 표시
- 역할 이름(RoleName) : 연관을 맺은 상태에서 상대 클래스에서 사용되는 역할의 이름

클래스와 클래스의 집합 연관 관계(Aggregation Relationship)



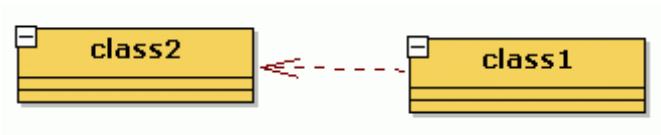
속이 빈 마름모 머리를 가진 실선으로 표기. 집합 연관 관계는 연관 관계의 일종으로 연관 관계에서 쓰이는 모든 장식들을 다 사용할 수 있다. 클래스와 클래스의 관계가 부분과 전체의 관계를 가질 때 표시할 수 있다. 예를 들어 자동차와 바퀴는 전체와 부분의 관계가 될 수 있다.

클래스와 클래스의 복합 연관 관계(Composition Relationship)



속이 찬 마름모 머리를 가진 실선으로 표기. 부분 클래스가 전체 클래스와 같은 생명 시간을 가진다(전체 클래스 객체가 소멸될 때 부분 클래스 객체 또한 소멸)는 차이점을 빼고는 집합 연관 관계와 유사하다.

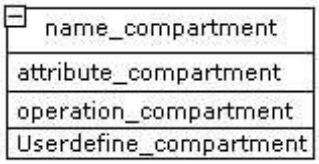
클래스와 클래스의 의존 관계(Dependency Relationship)



열려진 머리의 화살표를 가진 점선으로 표기하며, 한 클래스의 변화가 다른 클래스의 영향을 미칠 때 사용한다.

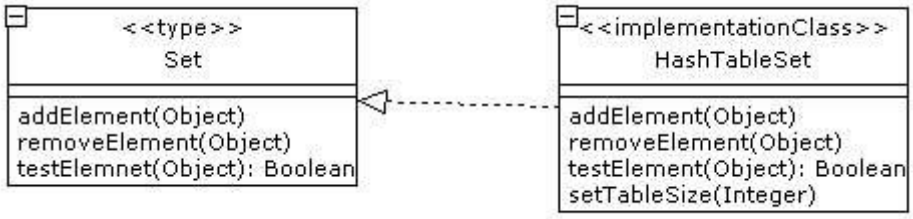
- 클래스 다이어그램(2)

클래스에서 사용자 정의 구역(User-defined compartment)



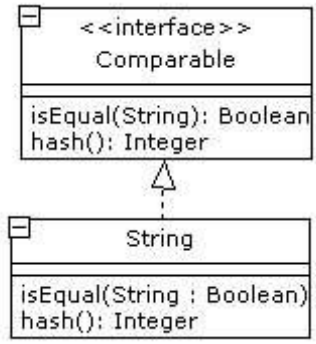
UML에서 미리 정의하는 부분 외에 사용자가 정의하여 작성할 수 있는 새로운 구역을 첨가할 수 있다.

Type and Implementation Class



Type의 경우 스테레오 타입으로 'type'을 가지며, 객체가 가지는 Specification만을 표시한다. Implementation class의 경우 'Implementation Class'를 가지며, 실제 물리적인 언어에 바인딩 되게 표현한다.

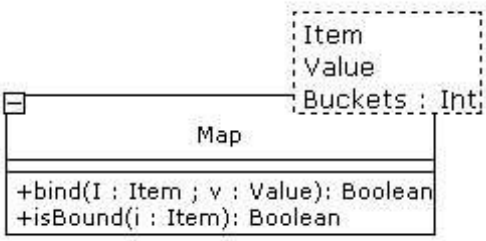
Interface



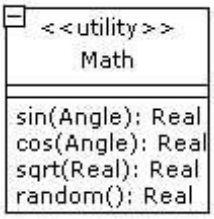
인터페이스 클래스의 경우 스테레오 타입을 'interface'로 가지며 객체지향 언어인 java에서 사용되는 인터페이스의 의미와 동일하게 클래스의 행위만을 확정한다. 이런 인터페이스는 구현을 가지지 않으므로 추상적인 operation을 가지게 된다.

Parameterized Class (Template Class)

객체 지향 언어 C++에서 사용되는 Template와 동일하다.



Utility



스테레오 타입으로 'utility'를 가지며 일반적인 클래스의 의미가 아닌 프로그램의 편리를 위해 만들어진 클래스이다. 프로그래밍을 하면 반드시 전역으로 만들어야 할 프로시져나 변수들이 존재한다. 이를 기능적으로 분리하기 위해 Utility Class를 사용한다. Utility Class 내부에 존재하는 attribute나 operation의 경우 전역 변수나 프로시져로 인식하면 된다.

MetaClass

스테레오 타입 'metaclass'를 가지는 클래스이며, metaclass의 인스턴스가 클래스가 되는 클래스를 의미

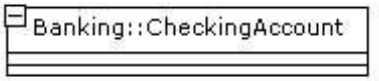
Enumeration

스테레오 타입 'enumeration'을 가지는 클래스이며, 프로그램 언어에서 사용되는 enumeration type과 유사하다. Enumeration Class의 인스턴스는 반드시 사용자가 정의한 특정 문자의 집합이어야 한다. 이러한 문자는 상대적인 순서를 지닌다.

Stereotype

스테레오 타입 'stereotype'을 가지는 클래스이며, 사용자 정의 스테레오 타입을 만들기 위해 사용되는 클래스이다.

ClassPathname



클래스를 표기함에 있어 UML에서 패키지(Package)를 같이 붙여 클래스의 범위를 지정할 수 있는데, 패키지는 UML에서 Namespace의 역할을 한다. 패키지 속에 패키지가 포함될 수 있으므로 패키지 path를 다 적용하여 클래스의 Pathname을 표기하기도 한다.

-후기-

UML Tool 로 OOAD 를 수행하는 과정을 설계하라는 과제를 받고, 처음엔 어떤 UML Tool 을 쓸 것 인지, 종류엔 무엇이 있는지, 어떻게 해야 할지 어리둥절했다. 팀원과 조사를 하면서, OOAD를 수행하는 과정은 어떻게 되는지 과제를 하면서 조금씩 배워나가지만 아직도 어려운 부분이 많다. 앞으로 세 번 정도는 더 할 수 있을 것 같은데 이번학기가 끝날 무렵 에는 자세하게 알 수 있도록 더 깊이 알아볼 수 있는 시간이 되었으면 한다.

-By Team1.