

과정명

OO- 226 Object- Oriented Analysis and Design Using UML

단원1 : OOSD Process와 객체지향 방법론

1 모듈 : 소프트웨어 개발과정

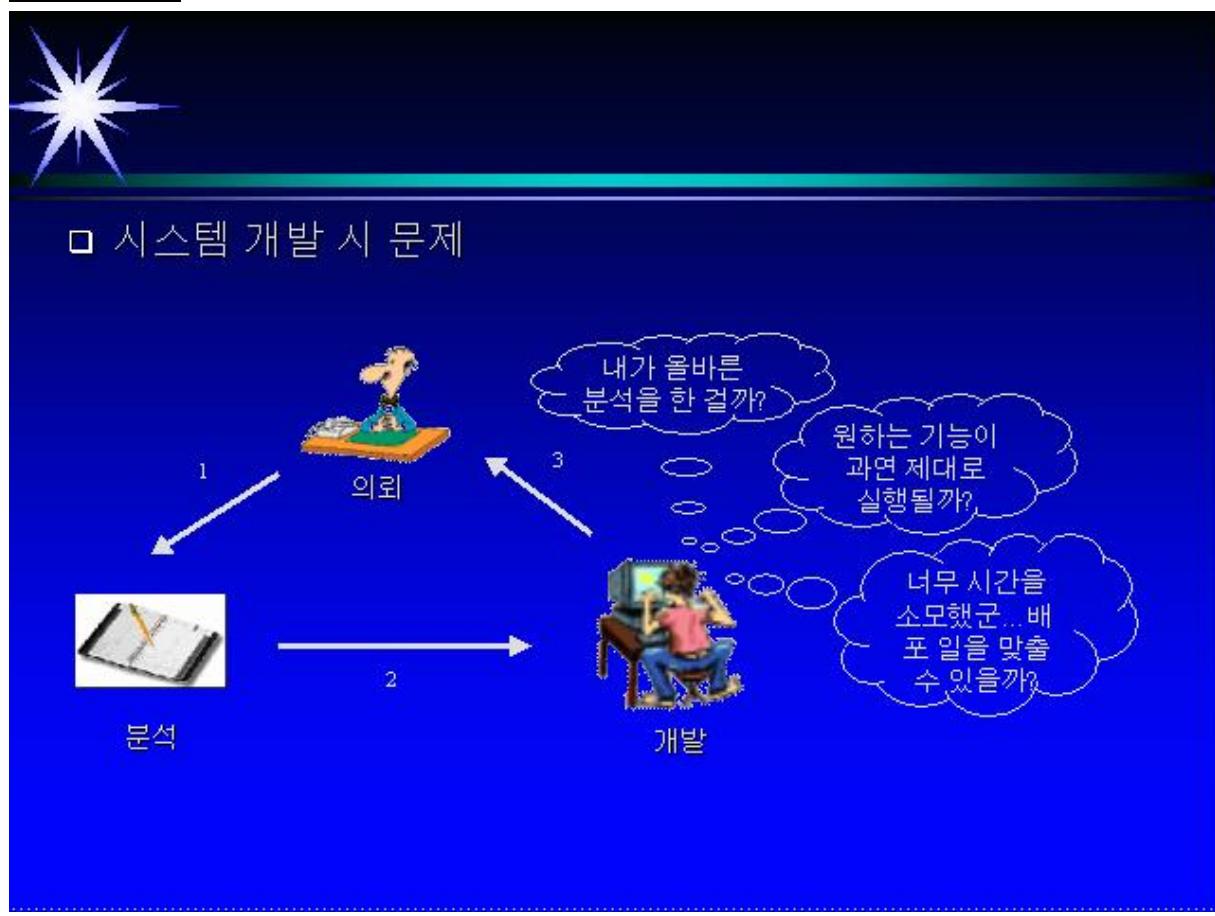
담당강사 : 전은수

■ 생각해봅시다 ■

소프트웨어를 작성하는 일은 어렵습니다.

막상 개발을 시작하려고 하면 어떤 과정을 거쳐 무슨 일을 해야 할 것인가와 같은 기본적인 문제부터 막히게 됩니다. 그렇기에 프로그래머라면 누구나 한번쯤 제한된 자원을 가지고 정해진 기한을 맞추기 위해 밤을 새워본 기억이 있을 것입니다. 그러나 프로그램을 배포(**release**)할 때 이미 유행이 지나버릴 위험, 예산 초과의 위험, 목적한 작업도 제대로 수행하지 못하는 소프트웨어를 구축하는 위험, 게다가 시간에 쫓겨 변변한 테스트도 거치지 않은 버그 투성이의 제품을 내 놓을 수도 있는 위험은 항상 내재해 있습니다.
그렇다면 이런 장애물을 극복하는 해법은 없을까요?

애니메이션



■ 학습하기 ■

1. OOSD Process에 대한 이해

1) OOSD Process의 필요성

소프트웨어 산업은 그 태동으로부터 반세기 동안 기하급수적으로 성장, 변경되어 왔습니다. 프로그래밍 기술은 언어(**language**), 운영체제(**OS:Operating System**), 네트워킹, 커뮤니케이션 프로토콜, 컴포넌트 기반 **API**, 어플리케이션 서버 소프트웨어 등 엄청난 변화를 겪어 왔습니다. 결국 소프트웨어 개발 프로세스는 ‘기술(**Technology**)’에 따라 변화되었다고 할 수 있습니다. 그러므로 만약 여러분들이 객체지향기술을 사용하여 소프트웨어를 개발하고자 한다면 객체 지향 소프트웨어 개발 과정(**OOSD Process: Object- Oriented Software Development Process**)을 사용해야 합니다.

또한 소프트웨어 프로젝트는 그것이 조직화 되는 법과 관리되는 법에 관한 무수한 변화를 겪어 왔습니다. 개발하려는 소프트웨어가 무엇을 필요로 하는지 누가 결정합니까? 소프트웨어 솔루션은 어떻게 만들어질까요? 개발 라이프사이클(**lifecycle**)은 어떻게 관리됩니까? 그 해답이 바로 **OOSD Process**에 있습니다.

참고하세요

▪ 표준화된 OOSD Process는 존재하는가.

표준화된 단 하나의 **OOSD Process**는 없습니다. 그것은 단지 프로젝트를 성공적으로 이끌기 위한, 여러 방법론을 통해 유추해낸 하나의 가능한 접근일 뿐입니다.

Key Point

▪ OOSD Process의 필요성

개발하려는 소프트웨어가 무엇을 필요로 하는지 누가 결정합니까? 소프트웨어 솔루션은 어떻게 만들어질까요? 개발 라이프사이클(**lifecycle**)은 어떻게 관리됩니까?
그 해답이 바로 **OOSD Process**에 있습니다.

(1) 소프트웨어 방법론(**Methoddology**)이란

사전적인 의미로 방법론이란, 경험에 의해 얻어진 방법, 규칙, 가정들의 집합입니다.

소프트웨어 개발에서 방법론은 개발과정의 가장 고수준으로 얘기되어집니다.

소프트웨어 산업 역사 전반에 걸쳐 많은 방법론이 개발되어져 왔습니다.

OO방법론은 OOSD Process를 통해 객체지향 개념을 통합시켰습니다.

근래의 많은 방법론은 **Inception, Elaboration, Construction, Transition** 이렇게 4단계

로 구성되어집니다.

이들 단계는 다시 세부적인 워크플로우(**Workflow**)로 구성되어지고 이 워크플로우는 특별한 **Activities**(동적행위)로 이루어집니다. **Activities**는 **Worker**(워커)와 **Artifacts**(산출물)를 포함합니다.

Worker는 **activity**를 수행할 사람을 뜻하고 **Artifacts**는 **activity**에 의해 산출된 유형의 정보 조각, 예를 들어 다이어그램들이나 문서, 소프트웨어 코드 등을 말합니다.

Artifacts는 대개 툴을 통해서 쉽게 얻을 수 있도록 간구 되어 왔고 그 중 **UML(Unified Modeling Language)**은 소프트웨어를 모델링하기 위한 가장 강력한 툴 중의 하나입니다.

참고하세요

■ **workflow** 동의어

이 과정에서는 **workflow**란 용어를 사용하지만 **OMG(Object Management Group)**에서 는 **workflow**란 말 대신에 **discipline**이라는 새 용어를 권고하고 있습니다.

Key Point

■ 방법론의 요소

Worker는 **activity**를 수행할 사람을 뜻하고 **Artifacts**는 **activity**에 의해 산출된 유형의 정보 조각, 예를 들어 다이어그램들이나 문서, 소프트웨어 코드 등을 말합니다.

Artifacts는 대개 툴을 통해서 쉽게 얻을 수 있도록 간구 되어 왔고 그 중 **UML(Unified Modeling Language)**은 소프트웨어를 모델링하기 위한 가장 강력한 툴 중의 하나입니다.

(2) OOSD 계층 피라미드

다음 그림은 **OOSD** 계층 피라미드입니다.

Methodology는 방법론을 말하고 **Phase**는 **OOSD Process**의 4단계 **Inception, Elaboration, Construction, Transition**을 말합니다.

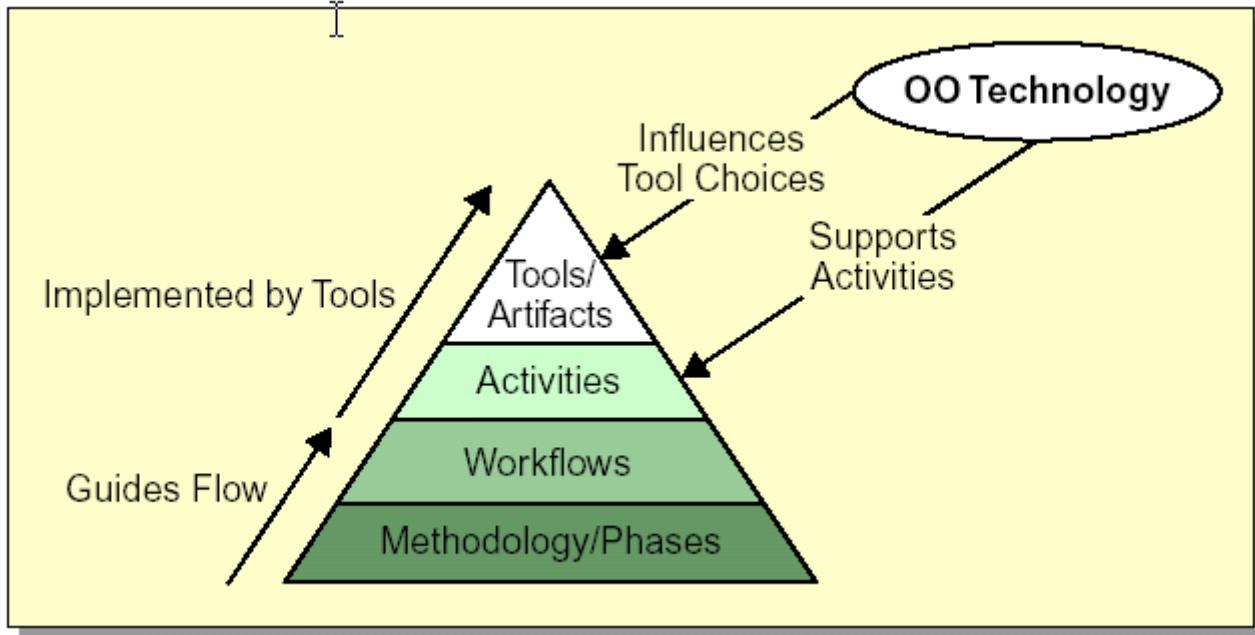
Workflow란 각 방법론을 바탕으로 한 개발 단계에서 행해지는 일련의 절차를 말하고

Activity는 구체적인 동적행위를, **Artifact**는 그 행위로 얻어지는 산출물을 일컫습니다.

이 산출물들은 일반적으로 **Tool**을 통해 쉽게 얻어 낼 수 있습니다.

이미지

■ **OOSD** 계층 피라미드



보충

n 방법론과 Phase의 차이

개발 방법론이란 프로젝트의 규모와 성격에 맞게 정형화될 수 있는 ‘틀’을 제시하는 여러 가지 사안을 의미합니다.

따라서 제시하는 사람 혹은 팀에 따라, 프로젝트의 규모와 성격에 따라 천차만별인 내용으로 구성 될 수 있습니다.

그러나 수많은 방법론들이 대부분 같은 단계를 뽑아 목적했던 소프트웨어를 구축해내게 되는데 그 단계를 바로 **Phase**라고 하는 것입니다.

예를 들어, 집에서 회사까지 가는 길은 여러 가지가 있고 교통 수단도 여러 가지가 있습니다. 어떤 사람은 버스 전용 차로를 달리는 버스를 타고 가면 가장 빠르다고 할 것이고, 어떤 사람은 택시를 타다가 막히는 구간에서만 버스를 타면 더 효율적이라고 할 것이고, 어떤 사람은 지하철을 타라고 할 것입니다.

그것이 바로 회사에 도달하기 위한 ‘방법’입니다.

그런데 어떤 방법을 택했던지 아침에 집에서 나와야 하는 단계를 거쳐야 하고 선택한 교통 수단에 탑승해야 하며, 회사 앞에서 내려야 하는 단계는 같습니다.

소프트웨어 구축도 마찬 가지로 선택한 방법론 안에서 계단을 끊고 올라가듯 일정하게 거쳐야 하는 단계가 있습니다.

이 방법론은 **Methodology**, 단계를 **Phase**라고 하는 것입니다.

소프트웨어 **Phase**의 일반적인 4단계는 **Inception**, **Elaboration**, **Construction**, **Transition**이라고 했습니다. **Inception**은 도입, **Elaboration**은 전개, **Construction**은 구축, **Transition**은 전이 단계를 말합니다. 이들 단계에 대한 자세한 사항은 본 단원 [모듈4. 객체지향 방법론과 올바른 선택]에서 자세히 다룹니다.

2) OOSD Process의 Workflow

① Requirements Gathering (요구사항 수집)

비즈니스 오너와 시스템 사용자들과의 인터뷰를 통해서 시스템의 요구 사항을 결정짓는 단계

② Requirements Analysis (요구사항 분석)

시스템 요구 사항을 분석, 정제, 모델링하는 단계

③ Architecture (아키텍쳐 수립)

시스템의 **high-level** 구조를 모델링하여 프로젝트의 리스크(**risk**)를 진단하고 그 것을 줄일 수 있는 방법을 간구하는 단계

④ Design (설계)

시스템 요구사항을 만족하는 솔루션 모델을 생성하는 단계

⑤ Implementation (구현)

솔루션 모델에 정의되어 있는 소프트웨어 컴포넌트를 생성하는 단계

⑥ Testing (테스트)

시스템이 요구사항을 만족시키는지를 검증하는 단계

⑦ Deployment (배치)

시스템을 배치하는 단계

3) 객체지향 기술의 장점

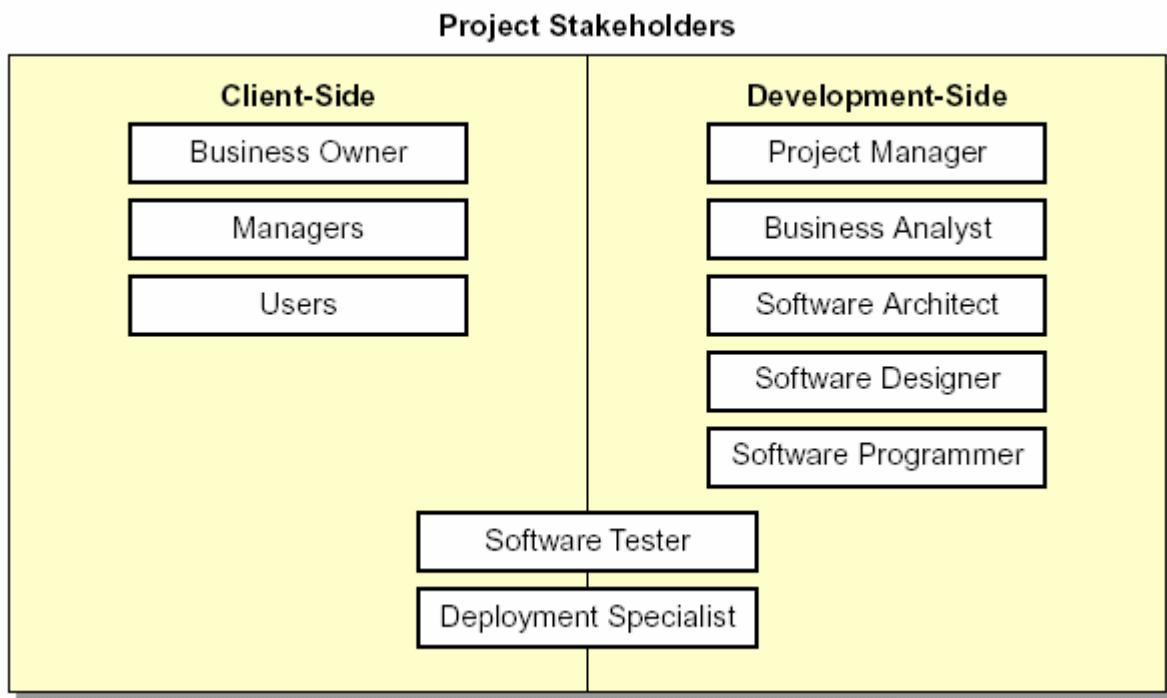
	Procedural Paradigm	OO Paradigm
유기적 구조	계층적인 수행 구문들로 이루어짐	네트워크를 통한 객체간의 협력 구조 를 가짐
소프트웨어 변경 능력	한번 만들어진 소프트웨어를 변경하는 것이 어려움	소프트웨어는 견고하면서도 변경에 유연함
재사용	copy-and-paste 이외의 재사용이 어려움	데이터나 기능을 가진 객체의 재사용성이 탁월
특수한 경우의 구성	종종 if 나 switch 구문으로 해결	Polymorphism 이라고 하는 다형성 을 표현할 수 있는 기능을 가짐
기능의 분리	기능별로 분리되기 어려움	모듈기능의 코드가 기능의 분리를 도와줌

4) 소프트웨어 개발팀의 Job Roles

다음 그림은 프로젝트 참가자들을 그룹화 시킨 것입니다.

[이미지]

▶ Project Stakeholders



| Business Owner

클라이언트쪽 프로젝트 리더를 말합니다. 이들은 프로젝트에 관한 최종적인 결정권자입니다.

| Users

시스템 사용자를 전부 일컫습니다. 기업내부 시스템이라면 일반적으로 사원들이 될 것이고 웹 기반 시스템이라면 모든 인터넷 사용자가 될 것입니다.

| Managers

내부 사용자의 관리자를 말합니다.

| Project Manager

소프트웨어 개발 프로젝트의 관리자를 말합니다. 이들이 항상 기술적인 관리자라고는 할 수 없습니다. 그들은 프로젝트에 대한 예산을 세우고 자원을 산정하고 스케줄을 관리하는 임무를 맡습니다. 종종 클라이언트에게 프로젝트를 설명하고 컨설팅하는 역할을 하기도 합니다.

| Business Analysis

클라이언트쪽 **stakeholder**들을 통해 수집된 정보를 가지고 시스템이 요구하는 기능적 요소가 무엇인지를 분석합니다.

| Software Architect

시스템의 구조를 정의합니다. 프로젝트 수행 시에 발생할 수 있는 위험 요소들을 분석하

고 그것을 줄일 수 있는 방법을 모색하여 시스템의 비기능적 요구 사항을 맞출 수 있도록 분석하는 사람들을 말합니다.

I Software Designer

아키텍쳐 프레임워크 내에서 시스템의 기능적 요구 사항을 만족시킬 수 있는 솔루션 모델을 생성하는 사람들을 말합니다.

I Software Programmer

소프트웨어 솔루션을 구현하는 사람들입니다. 작은 개발팀에서는 **Software Designer**가 **Software Programmer**의 역할을 동시에 하기도 합니다.

참고하세요

■ **Software Programmer** 동의어

Software Programmer는 **Software Developer**와 동의어입니다.

I Software Tester

시스템이 원래 목적했던 기능적 요구 사항이나 비기능적 요구 사항 모두를 만족시키는지를 검증하는 사람입니다. 시스템의 품질보증을 위한 시험을 담당합니다.

참고하세요

■ **Software Tester** 동의어

Software Tester는 **QA Specialist**와 **Test Engineer**와 동의어입니다.

I Deployment Specialist

생산 플랫폼위에 구현물을 배치하는 사람을 말합니다. 이 역할을 다른 여러 **Job role**을 포함합니다 : **System Administrator**, **Network Administrator**, **Script Writer** 등등.

이 역할은 프로젝트의 구축단계에서는 개발팀 내에서 수행되지만 시스템이 생산 플랫폼으로 갔을 때는 클라이언트 조직 내에서 수행하게 됩니다.

Key Point

■ **Project stakeholder**란

프로젝트에 참가하는 모든 사람들을 **project stakeholder**라고 합니다.

여기에는 일반 사용자들도 포함됩니다

보충

■ **Project stakeholders**

프로젝트에 참가하는 모든 구성원들은 서로 다른 역할을 하게 되거나 한 사람이 여러 역할을 맡을 수도 있습니다. 또한 여러 명이 한 역할을 공유할 수도 있습니다.

따라서 각각의 역할에 따른 책임과 임무는 프로젝트 계획에 정확히 정의되어 있어야 합니다.

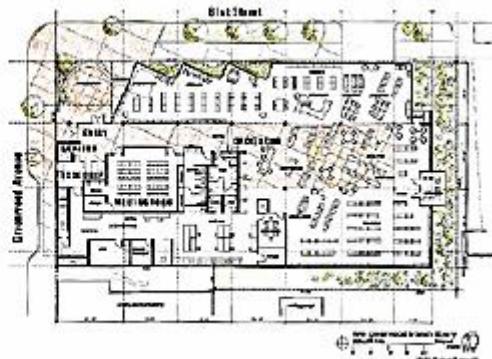
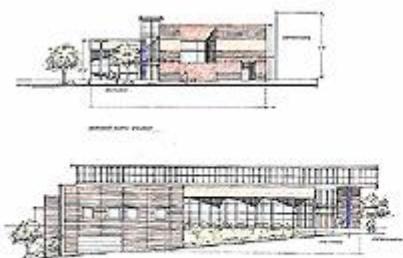
2. 소프트웨어 모델링의 장점

1) 모델이란?

“A Model is a simplification of reality.” (Booch UML User Guide page6)

[이미지]

n 모델의 여러 가지 뷰



(Buffalo Design © 2002. Images used with permission.)

- | 모델이란 어떤 엔티티 또는 시스템의 ‘추상적 개념화’ 입니다.

모델은 그것의 본질로 추상화 되어집니다. 또한 모델은 여러 다른 사물들 (예를 들어, 빌딩이나 컴퓨터 네트워크 같은 물리적 사물들이나 소프트웨어와 같은 관념적 사물들까지도)로 표현 될 수도 있습니다.

- | 모델은 어떻게 바라보느냐에 따라 다양한 뷰(view)가 나올 수 있습니다.

건축가는 집을 지을 때 평면도, 정면도, 투시도와 같은 다양한 관점에서의 그림을 그립니다.

소프트웨어도 마찬가지로 이러한 여러 관점에서의 그림을 그리게 됩니다.

2) 왜 소프트웨어를 모델하는가?

“We build models so that we can better understand the system we are developing.”
(Booch UML User Guide page6)

모델링은 다음과 같은 것을 가능하게 합니다.

I 새 시스템이나 기존 시스템의 가시화

모델은 우리가 어떤 시스템을 이해하는데 좀 더 쉬운 방법으로 가시화 할 수 있습니다.

예를 들어, 기존 시스템이 어떤 기능을 가지고 있는지를 설명하기 위해 우리는 유즈케이스 다이어그램을 그릴 수 있는데, 이것이 바로 시스템을 사용자의 비즈니스적 입장에서 바라보는 그림이라 할 수 있습니다.

I 프로젝트 stakeholder들간의 의사소통을 원활히 함

어떤 모델은 비즈니스 오너와 도메인 전문가 그리고 다른 클라이언트 참가자들과의 이해를 돋는 데 필수적입니다. 또 어떤 모델은 개발팀내에서 사용하는데 더 유용할 수도 있습니다.

I 각 OOSD 워크플로우에서 만들어지는 결정의 문서화

개발팀의 크기에 따라 모든 팀원들이 같은 워크플로우를 진행할 수 있는 것은 아닙니다. 각 역할 구성원들이 다른 작업을 수행하면서 이루어지는 모든 프로젝트에 관한 결정은 문

서화 되어야만 이후에 다른 구성원들에게 전달되어 사용될 수 있을 것입니다.

| 시스템의 정적, 동적 요소들을 특화시킴

디자인 워크플로우와 구현 워크플로우를 수행하는 동안에 소프트웨어 시스템의 정적, 동적 요소들은 요구사항 모델로부터 특화되어야만 합니다. 여기서 정적 요소란 시스템의 구조를 말하고 동적 요소란 시스템의 동작을 말합니다. 솔루션 모델은 소프트웨어 시스템의 완전한 개념적 표현입니다. **UML**은 이 솔루션 모델을 볼 수 있도록 다양한 다이어그램을 제공합니다.

3) 모델의 종류

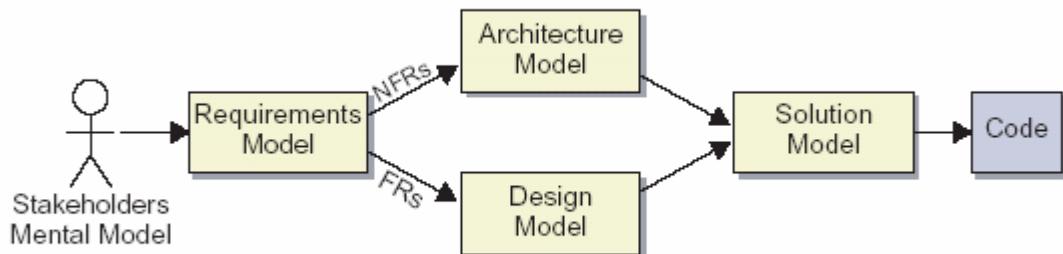
다음 그림은 프로젝트 진행 시에 나오는 몇 가지 모델을 보여줍니다.

또한 최초 **stakeholder**의 **Mental Model**이 프로젝트가 진행되는 동안에 어떻게 변화, 진화되어 가는지를 간략히 도식화 하고 있습니다.

- | **stakeholders Mental Model** : 프로젝트 참가자들의 구상 모델
- | **Requirements Model** : 시스템이 최종적으로 가져야 할 모든 요구 사항 모델
- | **Architecture Model** : 시스템의 비기능적 요구 사항들에 관한 모델
- | **Design Model** : 시스템의 기능적 요구 사항들에 관한 모델
- | **Solution Model** : 시스템의 요구 사항을 만족시키기 위해서 설계된 총체적인 통합 모델

[이미지]

n 소프트웨어 개발 단계에 다른 모델의 종류



이러한 모델들은 저장하고 기록하는 것이 중요합니다.

SRS document과 같은 산출물은 문서이고 유즈케이스 다이어그램 같은 산출물은 그림입니다. 이를 문서와 그림을 그리는 여러 가지 방법이 있는데, 이 과정에서는 **UML(Unified Modeling Language)**을 통해 여러 산출물들이 어떻게 만들어 지는지를 살펴 볼 것입니다.

4) UML이란?

“The Unified Modeling Language(UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.” (UML v1.4 page xix)

UML이란 프로젝트에 참여하는 모든 **Stakeholder**들이 이해하고 동의 할 수 있는 방법으로 개발 과정을 조직화하는 수단입니다.

예를 들어, 어떤 건축가가 그려 놓은 설계도면으로 다른 건축가가 집을 지어도 원래 계획했던 집의 형태대로 완성되어 질 수 있는 것은 두 건축가 사이에 그림을 이해할 수 있는 능력과, 그림이 표현하고자 하는 의미가 동일하기 때문인 것입니다.

이처럼 시스템 개발 시에도 프로젝트 참가자 전원이 이해 할 수 있는 같은 의미의 표현 수단이 존재해야 하는데 그런 표현 수단은 다양하게 발전되어 왔고 **1990년** 중반 이후로 **UML**이라는 표현 수단이 가장 주목 받기 시작했습니다.

1997년에는 **UML** 컨소시엄이 **UML** 버전 **1.0**을 발표하고, 이어 **OMG(Object Management Group)**에서 표준 모델링 언어로 채택하게 됩니다.

UML을 사용하면 모델은 다음과 같이 구성됩니다

- | **Elements (요소)**
- | **Diagrams (그림)**
- | **Views (관점)**

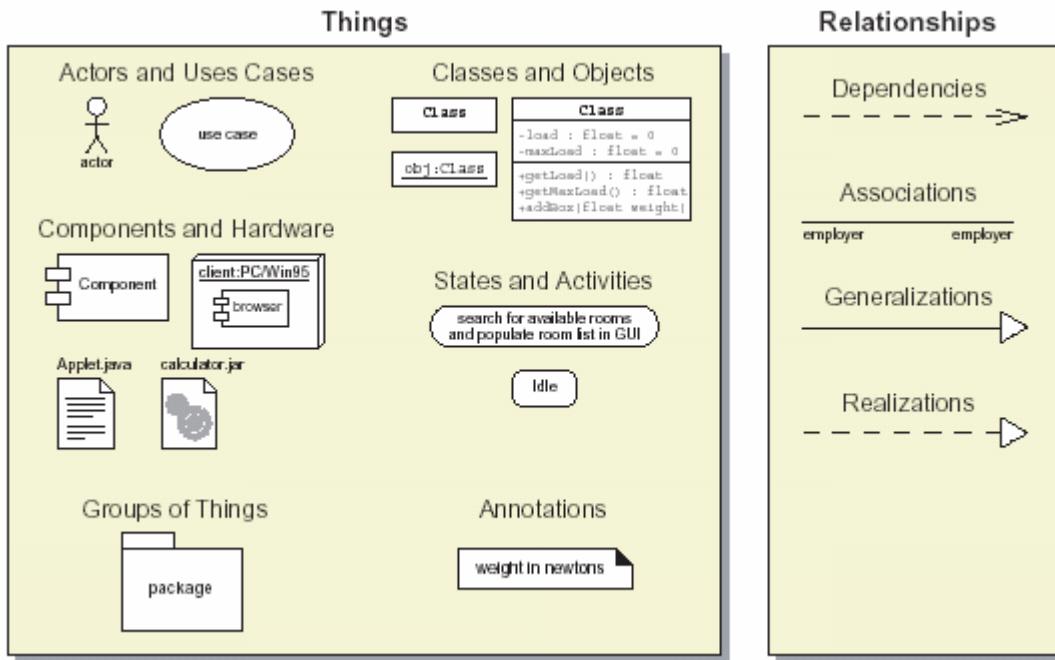
(1) UML Elements (UML 요소)

UML 요소는 크게 사물(**Things**)과 관계(**relationships**)로 나눌 수 있습니다.

- | **Actor** : 시스템 사용자나 다른 시스템
- | **use case** : 시스템의 기능
- | **class** : 객체지향 프로그램의 단위
- | **object** : 프로그램 수행시의 구체적인 메모리 정보
- | **components** : 독립적으로 재사용 가능한 모듈
- | **state** : 프로그램이나 구성 요소들의 상태
- | **Activities** : 프로그램의 동적 행위
- | **dependencies** : 의존도
- | **Associations** : 연관성
- | **Generalizations** : 상속
- | **Realizations** : 구현

[이미지]

n UML 요소



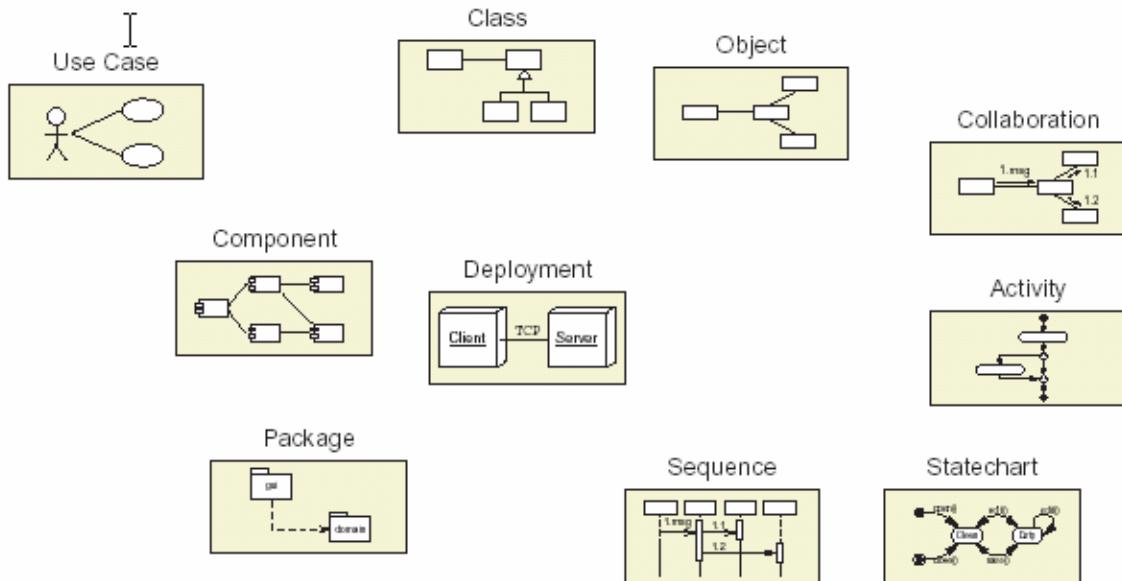
(2) UML Diagrams

UML 요소를 통해 프로젝트를 표현하기 위한 큰 그림, 즉 다이어그램은 다음과 같은 종류가 있습니다.

- | **Use Case diagram**
- | **Class diagram**
- | **Object diagram**
- | **Collaboration diagram**
- | **Sequence diagram**
- | **Activity diagram**
- | **Component diagram**
- | **Deployment diagram**
- | **Package diagram**
- | **Statechart diagram**

[이미지]

n UML 다이어그램



① Use Case Diagram

use case란 사용자의 입장에서 본 시스템의 행동을 말합니다. **시스템의 기능적 요소들을 표현한다고 생각하시면 되겠습니다.**

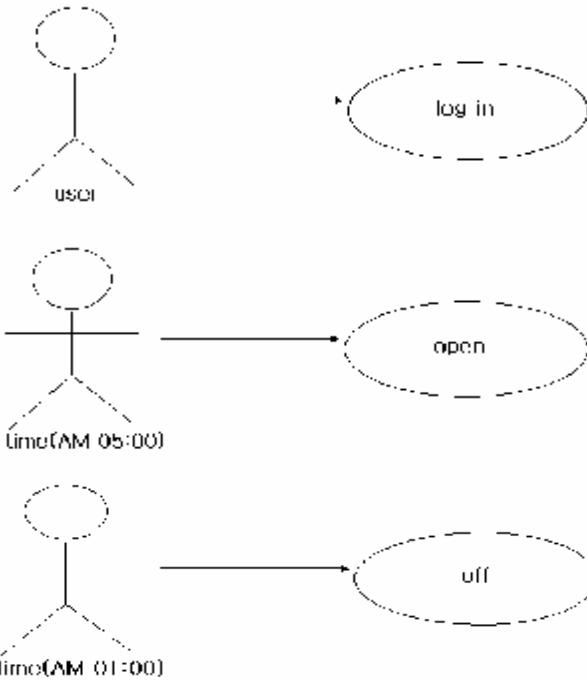
Use Case Diagram은 UML 요소 중 **actor**와 **use case**를 사용해 그려지는 그림입니다.

예를 들어 여러분들이 어떤 웹 기반 시스템에 접속하기 위해서 로그인을 원한다면 여러분들이 사용자, 즉 **actor**가 되고 ‘로그인’이 시스템의 행동, 즉 **use case**가 되는 것입니다.

actor는 반드시 사람만을 의미 하지는 않습니다. 어떤 경우엔 다른 시스템이 될 수도 있고 아니면 시간이 될 수도 있습니다.

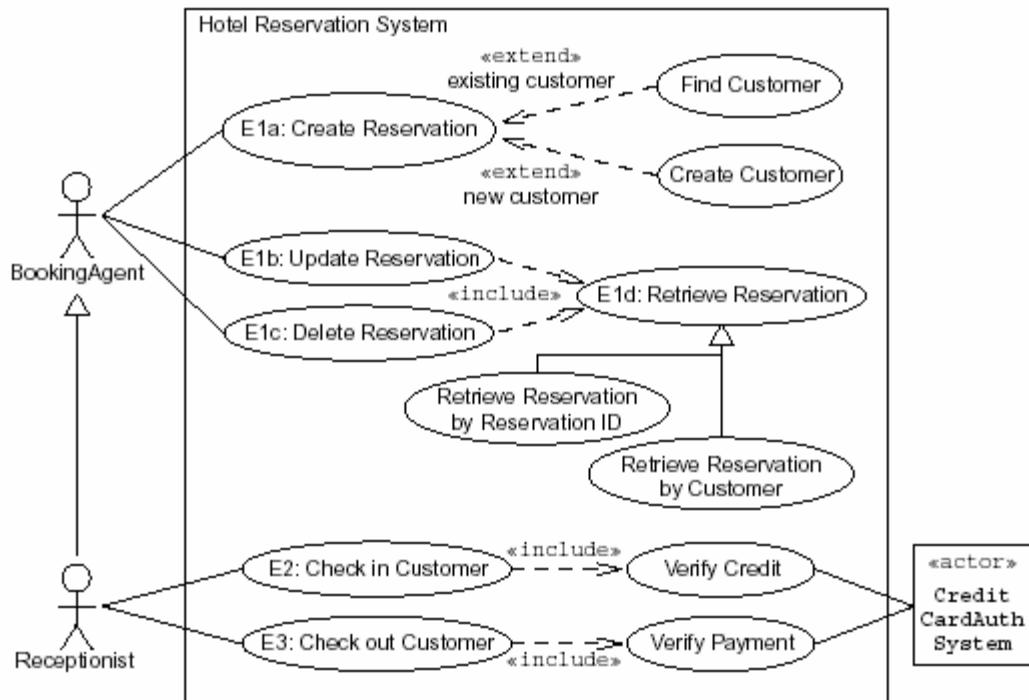
이미지

n 간단한 Use Case Diagram



이미지

n 상세한 Use case diagram



② Class Diagram

class란 객체지향 프로그램의 단위입니다.

객체지향 기술을 사용해서 프로젝트를 수행할 때 최종적으로 얻어지는 코드의 단위라는 것입니다.

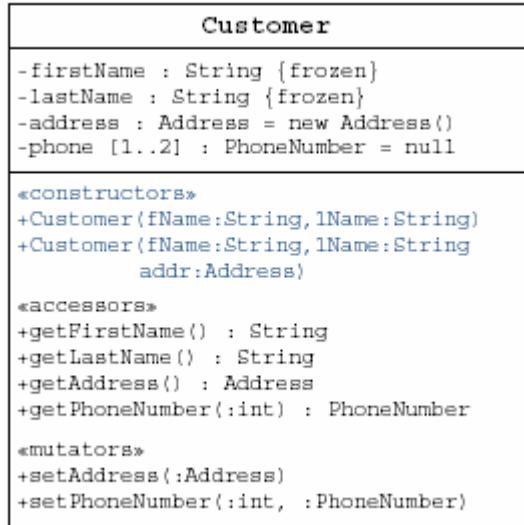
클래스는 변수(**attributes**)와 메소드(**method**)로 이루어집니다.

Class Diagram은 어떤 클래스의 이름과 변수, 메소드를 표현해 놓은 그

립인 것입니다.

[이미지]

n 클래스 다이어그램



③ Object Diagram

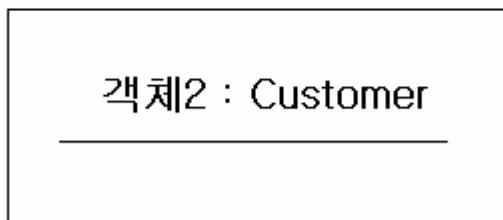
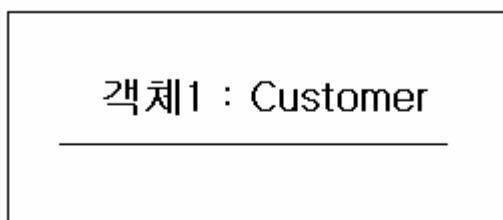
Object(객체)란 시스템이 수행 시에 클래스를 통한 구체적인 정보를 메모리에 저장해 놓고 사용하게 되는데 바로 그 메모리 정보를 일컫습니다. 하나의 클래스를 통해서 다양한 객체가 생길 수 있습니다.

예를 들어 이름변수를 가진 클래스를 통해서 생긴 객체1은 ‘전은수’라는 이름 변수 값을 갖고 객체2는 ‘홍길동’이라는 이름 변수값을 갖는 것입니다.

이때 객체1과 객체2는 서로 다릅니다. 객체는 유일무이하다라는 특성을 가지고 있습니다. 객체1과 객체2를 표현하는 그림을 **Object Diagram**이라고 합니다.

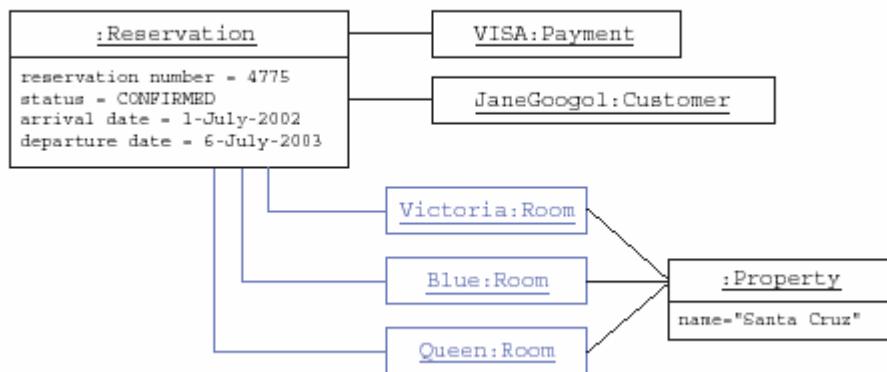
[이미지]

n Object Diagram 형식



[이미지]

▣ Object Diagram



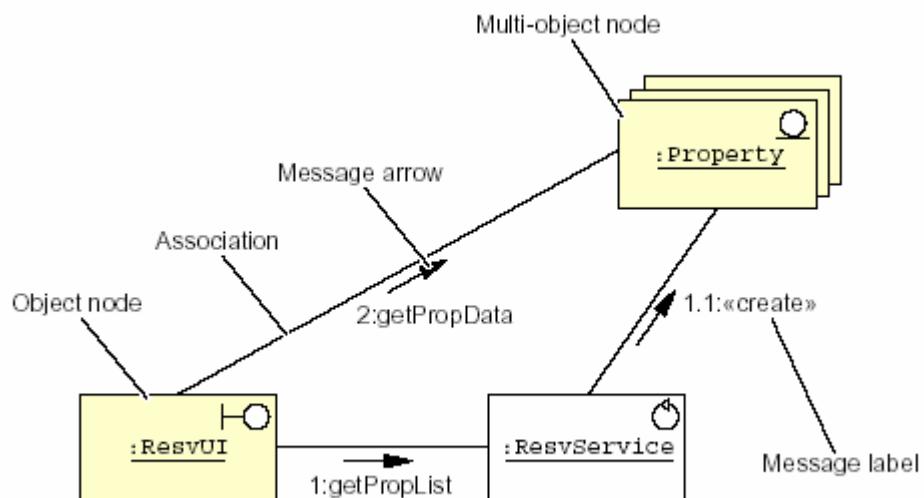
④ Collaboration Diagram

Collaboration이란 ‘협력’이란 뜻입니다. **Collaboration Diagram**은 객체들간의 상관 관계 또는 협력 작업을 그림으로 그려 놓은 것을 말합니다.

객체들간의 관계는 실선으로 표현하며 화살표가 관계의 방향성을 지정합니다. 화살표에는 숫자를 매겨서 관계의 시간상 흐름을 표현하고 선의 이름으로 그 관계를 더욱 명확히 설명할 수 있습니다.

[이미지]

▣ Collaboration Diagram



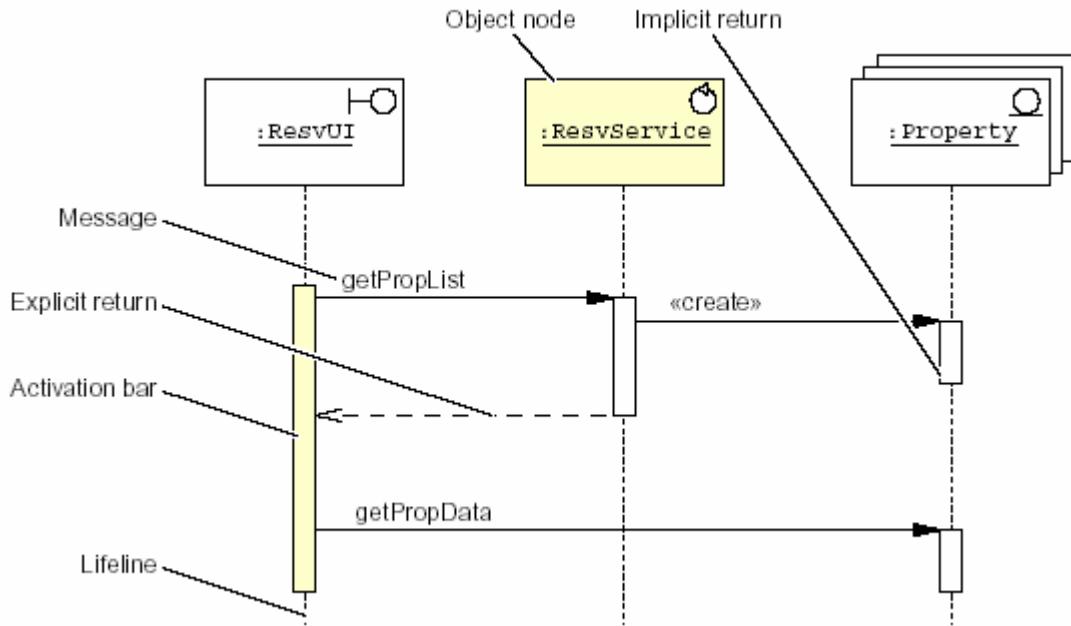
⑤ Sequence Diagram

Collaboration Diagram이 객체간의 협력관계를 표현한 그림이라고 합니다. 그런데 **Collaboration Diagram**을 보실 때 객체간의 협력관계를 시간의 흐름으로 일목요연하게 보시기 어려우셨을 것입니다.

Sequence Diagram은 객체들끼리 주고 받는 메시지의 순서를 시간의 흐름에 따라 보여주는 그림입니다.

[이미지]

n Sequence Diagram

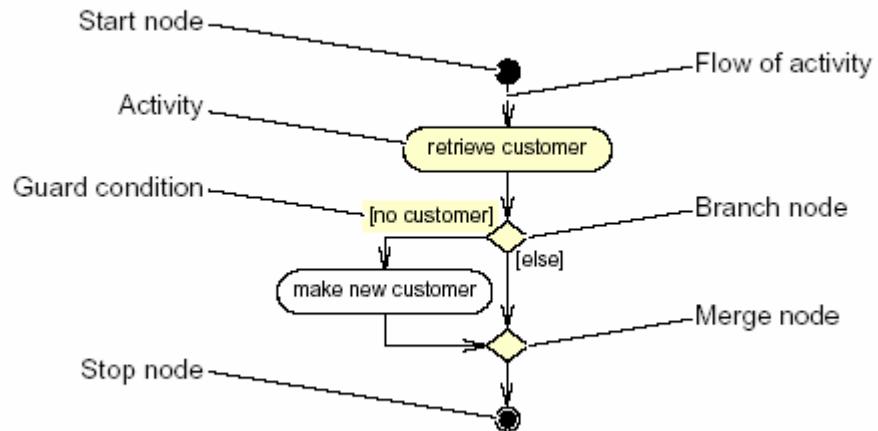


⑥ Activity Diagram

앞서 시스템의 기능적인 그림을 **use case diagram**이라고 했습니다. 그런데 **use case diagram**은 단지 어떤 기능을 한다는 것을 최대한 간략히 표현한 그림입니다. 따라서 그 기능을 수행하기 위해서는 어떤 일련의 동작들이 일어나고 그에 따라 시스템의 상태가 어떻게 바뀌는지에 대해서는 또 다른 그림으로 설명해야 할 것입니다. 바로 그런 이유에서 **Activity Diagram**을 그립니다.

[이미지]

n Activity Diagram



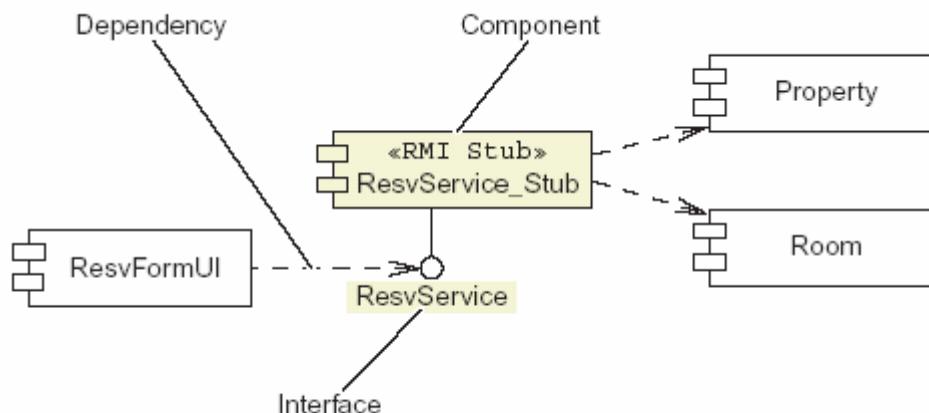
⑦ Component Diagram

컴포넌트(Component)란 독립적으로 재사용 가능한 모듈을 말합니다.

근래의 컴퓨팅 패러다임은 컴포넌트 기반으로 치닫고 있습니다. 특히 객체지향 기술은 컴포넌트 기반으로 개발할 수 있는 근간이 된다 할 수 있습니다. 여러 컴포넌트를 통해 하나의 시스템을 이루는 그림을 **Component Diagram**이라 합니다.

[이미지]

n Component Diagram



[보충]

n 컴포넌트란

객체 지향 개념이 발전 확산되면서 소프트웨어 자체를 재사용 가능하면서도 생산성이나 보수성이 높게 개발하기 위한 연구 개발도 계속되었습니다. 이들 중 소프트웨어를 하드웨어처럼 부품을 조립하듯 구축하면 유연성이거나 확장성이 높은 소프트웨어가 단기간에 저 비용으로 구축 될 수 있겠다는 것이 컴포넌트웨어(**componentware**)입니다. 이것은 근래 소프트웨어공학의 꿈입니다. 이를 위해서 프로그램의 단위를 컴포넌트화하고 이것을 모델화하여 표현한 것이 **component diagram**입니다.

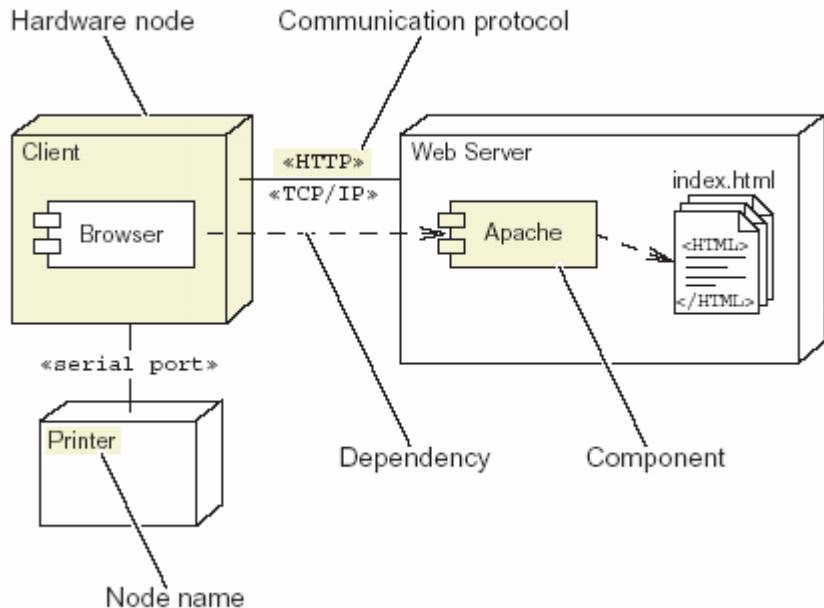
⑧ Deployment Diagram

Deployment Diagram은 컴퓨터를 기반으로 하는 시스템의 물리적인 구조를 나타낸 그림입니다.

직육면체는 하드웨어, 즉 컴퓨터 시스템을 나타내고 하드웨어 사이의 연결 관계는 실선으로 표현하며 각 하드웨어 위에 어떤 컴포넌트가 배치 되어 있는지를 보여줍니다.

[이미지]

▫ deployment diagram

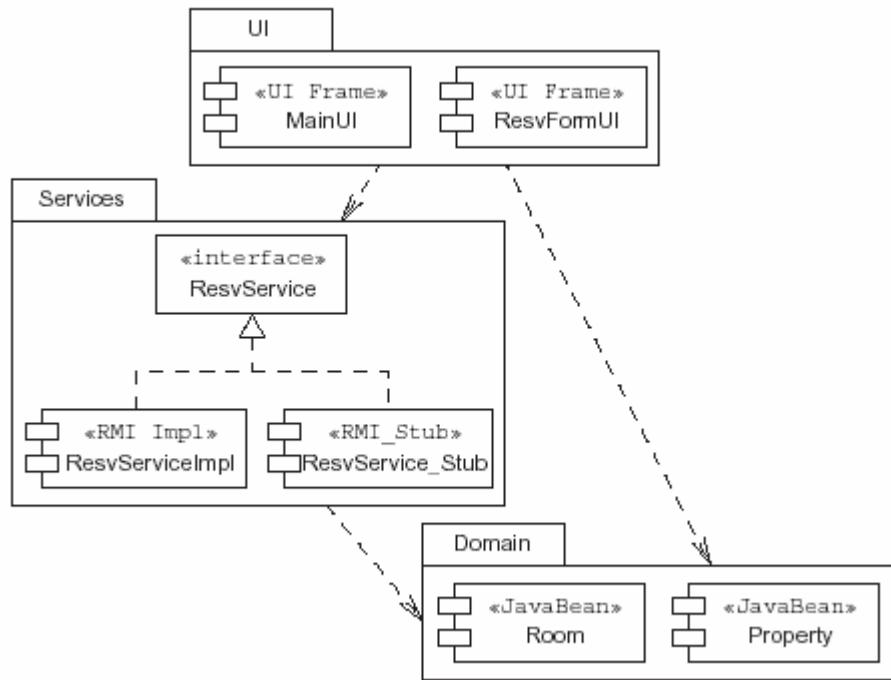


⑨ Package Diagram

한 프로젝트를 진행할 때 다양한 종류의 많은 다이어그램들이 작성되게 됩니다. 이런 다이어그램들은 연관성 있는 단위들로 그룹화해서 뮤어 관리하면 훨씬 유용하게 됩니다. 이런 그룹화 그림을 **Package Diagram**이라고 합니다.

[이미지]

▫ Package diagram



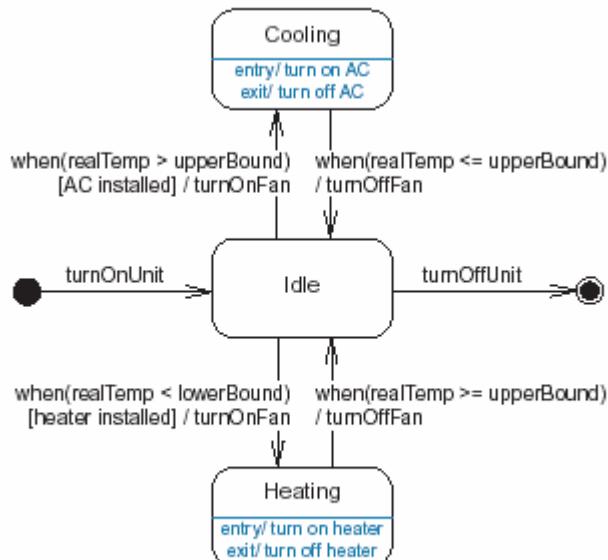
⑩ Statechart Diagram

객체는 시간에 따라 다른 상태에 놓일 수 있습니다.

예를 들어, 지하철역의 턴 스틸은 사람들이 패스포트를 집어 넣으면 통과 가능 상태가 되고 사람이 통과한 후엔 도로 잠김 상태가 됩니다. 또한 잘못된 패스포트를 집어 넣으면 에러 상태가 되면서 통과 할 수 없음을 알려주게 되고 에러가 풀리면 다시 잠김 상태로 돌아가 대기하게 됩니다. 이것처럼 어떤 시스템에서 활동하고 있는 객체가 상황에 따라 자신의 상태를 변경할 수 있다면 이를 그림으로 표현하여 객체의 상태를 좀 더 쉽게 이해 할 수 있도록 해주는 것이 **Statechart Diagram**입니다.

[이미지]

n Statechart Diagram

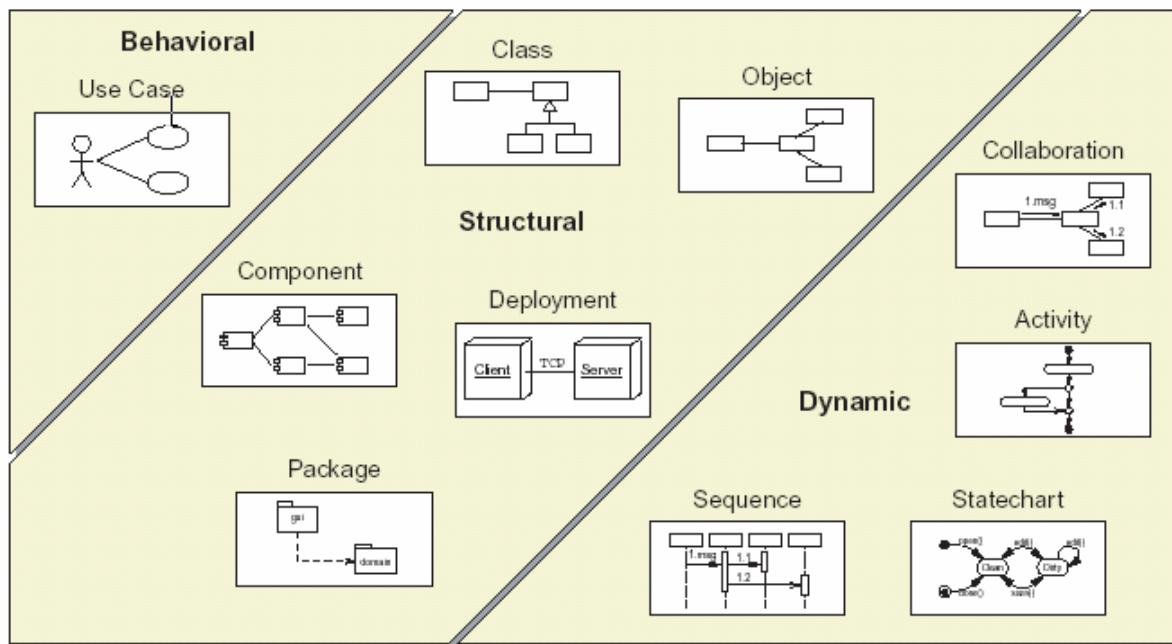


(3) Views

뷰(Views)란 여러 가지 다이어그램들을 보는 시각에 따라 어떻게 분류하는가 하는 것입니다. **Use Case Diagram**은 시스템의 기능적인 것을 표현한다고 하였으므로 행위 중심의 뷰(**Behavioral view**)입니다. 프로그램 코드가 어떻게 이루어 질 것인지를 보여 주는 **Class Diagram**이나 실행 시 메모리 상의 구체적인 정보를 표현하는 **Object Diagram**, 시스템을 구성하고 있는 컴포넌트들을 볼 수 있는 **Component Diagram**과 시스템이 어떤 하드웨어로 구성되어 있는지를 보여주는 **Deployment Diagram**, 마지막으로 여러 다이어그램들이 서로 잘 그룹화 되어져서 보여지는 **Package Diagram**은 모두 구조 중심적인 시각으로 그려진 것들입니다. 이들을 **Structural view**라고 합니다. 그에 비해 객체들간의 관계를 보여 주는 **Collaboration Diagram**, 시간상의 흐름을 보여주는 **Sequence Diagram**, 구체적인 동적 행위를 순차적으로 보여 주는 **Activity Diagram**, 객체의 상태변화를 보여주는 **Statechart Diagram**은 모두 동적인 관계를 묘사한 것입니다. 이것은 **Dynamic view**라고 합니다.

[이미지]

n 뷰에 따라 구분된 UML Diagram



5) UML에 대한 오해

실지로 많은 개발자들로부터 **UML**은 상당한 오해를 받고 있습니다.

어떤 이들은 **UML**을 프로그램 랭귀지라고도 하고 어떤 이는 개발 방법론 중 하나라고도 합니다.

다음 표는 우리가 흔히 범할 수 있는 **UML**에 대한 오해를 바로 잡아 줍니다.

오해	실제
UML은 수행적 모델을 생성합니다	모델은 수행 할 수 있는 무엇인가가 아닙니다. 오로지 프로그램 코드만이 수행할 수 있습니다. UML은 단지 그 코드가 쉽게 이해되도록 보여주는 표준화된 그림입니다.
UML은 프로그래밍 랭귀지입니다.	UML은 절대 프로그래밍 랭귀지가 아닙니다. 굳이 얘기하자면 그것은 비주얼한 모델링 랭귀지입니다. 모델하는데 필요한 언어라는 것입니다. 실제로 UML 을 통해서 모델화된 소프트웨어는 여러 가지 객체지향 랭귀지를 통해 구현될 수 있습니다.
UML은 방법론입니다.	UML은 그냥 여러 그림들일 뿐입니다. 여러분들은 개발팀내에서 채택한 특별한 방법론을 통해 개발과정을 수행할 것이고 작업 중에 발생하는 여러 결정을 텍스트 문서나 그림으로 그려두려 할 것입니다. UML은 바로 그때 사용되는 표준화된 그림일 뿐입니다.

6) UML Tools

UML은 그 자체가 도구입니다.

UML Tool은 모델링을 위해 다음의 것들을 돋습니다.

- | 컴퓨터로 **UML**다이어그램을 그릴 수 있도록 지원합니다.
- | 다이어그램을 통해 가지게 되는 의미를 지원합니다.
- | 특별한 방법론을 지원할 수 있습니다.
- | **UML** 다이어그램을 통해 원시 코드를 얻을 수 있습니다.
- | 프로젝트를 위한 모든 다이어그램들을 체계화합니다.
- | 특별한 패턴이나 플랫폼에서 사용되는 요소들을 자동 모델화하는 기능도 있습니다.

보충

■ UML Tools

래쇼날사의 로즈나 투게더 소프트사의 투게더라는 툴을 많이 들어 보셨을 것입니다.

이것들이 바로 **UML tool**입니다.

과정명

OO- 226 Object- Oriented Analysis and Design Using UML

단원1 : OOSD Process와 객체지향 방법론

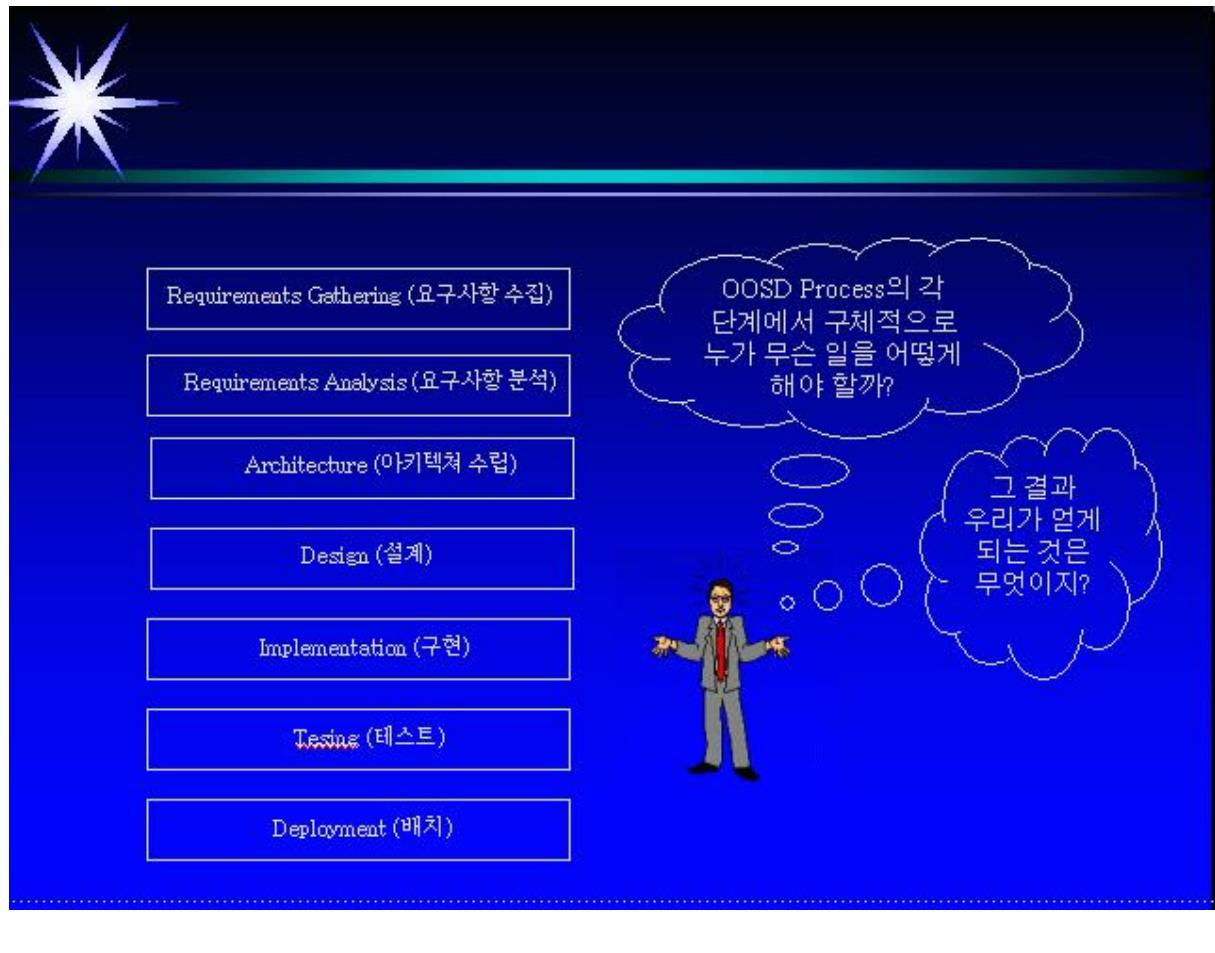
2 모듈 : OOSD Process 의 Workflow

담당강사 : 전은수

■ 생각해봅시다 ■

전 시간에 우리는 소프트웨어를 개발하고자 할 때 특별한 **Process**를 따르는 이유에 대해서 알아보고 간단히 그 **Process**의 **Workflow**에 대해서도 보았습니다. 그렇다면 **OOSD Process**의 **Workflow**의 목적과 그것을 수행하는 **Job Roles**는 무엇이라고 생각됩니까? 이번 시간에는 **OOSD Process**의 **Workflow**에 대해서 자세히 알아보겠습니다.

[애니메이션]



■ 학습하기 ■

1. 요구사항 수집 단계의 목적과 산출물

1) OOSD Process의 Workflow

① Requirements Gathering (요구사항 수집)

비즈니스 오너와 시스템 사용자들과의 인터뷰를 통해서 시스템의 요구 사항을 결정짓는 단계

② Requirements Analysis (요구사항 분석)

시스템 요구 사항을 분석, 정제, 모델링하는 단계

③ Architecture (아키텍쳐 수립)

시스템의 **high-level** 구조를 모델링하여 프로젝트의 리스크(**risk**)를 진단하고 그 것을 줄일 수 있는 방법을 간구하는 단계

④ Design (설계)

시스템 요구사항을 만족하는 솔루션 모델을 생성하는 단계

⑤ Implementation (구현)

솔루션 모델에 정의되어 있는 소프트웨어 컴포넌트를 생성하는 단계

⑥ Testing (테스트)

시스템이 요구사항을 만족시키는지를 검증하는 단계

⑦ Deployment (배치)

시스템을 배치하는 단계

2) 요구사항 수집 단계의 목적과 Job Roles

시스템의 요구 사항은 기본적으로 두개의 카테고리로 나뉘어질 수 있습니다.

하나는 기능적 요구 사항(**Functional Requirements : FRs**)이고 다른 하나는 비기능적 요구 사항(**Non- Functional Requirements : NFRs**)입니다. 기능적 요구 사항은 시스템을 사용하는 사용자의 입장에서 보는 시스템의 행동입니다. 이것은 **use case**로 가시화 될 수 있습니다.

비기능적 요구 사항은 시스템의 서비스의 질을 묘사합니다. 예를 들어 응답시간으로 계산된 퍼포먼스나 다수의 동시 사용자의 요구를 처리하는 작업량 등과 같은 것은 비기능적 요구 사항에 포함됩니다.

요구 사항 수집 단계의 목적은 비즈니스 오너와 시스템의 최종 사용자와의 인터뷰를 통해서 시스템이 무엇을 해야 하는지를 결정하는 것입니다.

다음 테이블은 요구 사항 수집 단계를 정리한 것입니다.

Workflow	목적	설명
요구 사항 수집	시스템이 무엇을 해야 하는지를 결정하기 위함	<p>결정 할 내용:</p> <ul style="list-style-type: none"> 시스템을 누가 사용할 것인지에 대해 시스템이 지원해야 하는 행동에 대해(Use Case) 각 Use Case에 대한 자세한 기능적 요구 사항 비기능적 요구 사항

요구 사항 수집 단계에서의 Activity는 주로 **business analyst**와 **architect**가 수행합니다.

심화학습

n 비기능적 요구 사항

비기능적 요구 사항은 시스템의 서비스의 질을 묘사합니다.

서비스의 질이란, **scalability**(확장성), **performances**(속도), **availability**(유용성), **reliability**(신뢰성) 등등을 포함합니다.

이러한 요소들의 결핍은 프로젝트를 완전히 실패하게 만들진 않지만 자연되거나 잠재 사용자를 수용하지 못하는 결과를 낳기도 합니다.

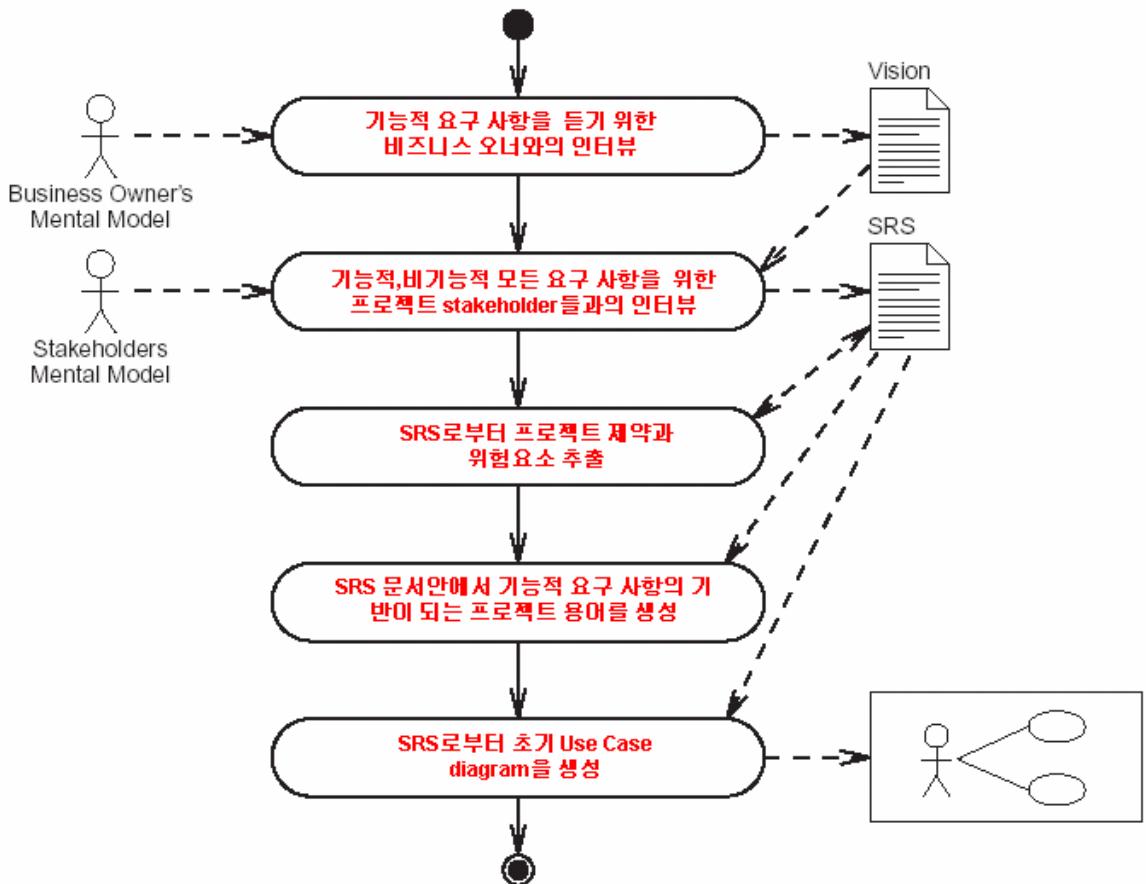
이 부분의 내용에 대해서는 [단원5. 비기능적 요구 사항 구축]의 전 모듈에서 자세히 다루게 됩니다.

3) 요구 사항 수집 단계의 Activities와 Artifacts

다음 그림은 요구 사항 수집 단계에서의 **activity**와 **artifacts**를 보여 줍니다.

이미지

n 요구사항 수집 단계의 Activities와 Artifacts



I 기능적 요구 사항을 듣기 위한 비즈니스 오너와의 인터뷰

소프트웨어 프로젝트의 크기와 범위를 결정하기 위해서 아키텍트(**architect**)와 비즈니스 분석가(**business analyst**)는 시스템의 기능적인 요구에 대한 사항을 비즈니스 오너와의 인터뷰를 통해 파악해야 합니다.

이때의 산출물(**artifacts**)이 **Vision Document**입니다.

I 기능적, 비기능적인 모든 요구 사항을 위한 프로젝트 참가자(stakeholder)들과의 인터뷰

비즈니스 오너는 다른 클라이언트쪽 참가자들과는 구별됩니다.

개발팀내의 프로젝트 참가자들과의 인터뷰를 통해서 다음과 같은 것들을 얻어야 합니다

1. **use case**들을 수정, 보완하고 추가합니다.
2. 각 **use case**를 위한 자세한 기능적 요구 사항을 구별해 냅니다.
3. 비기능적 요구 사항을 추출합니다.
4. **SRS** 문서에 기록합니다.

I SRS로부터 프로젝트 제약과 위험 요소 추출

프로젝트 제약은 구현 기술의 보유 상황이나 배포될 시점에서의 시장성등을 포함

하고 위험 요소는 기능적 요구 사항이나 비기능적 요구 사항 뿐만 아니라 프로젝트 제약 사항까지 곳곳에 산재해 있습니다.

이것은 **SRS**에 기록되어야 합니다.

I SRS 문서 안에서 기능적 요구 사항의 기반이 되는 프로젝트 용어를 생성

프로젝트 영역(이하 도메인 : **domain**)에서 특별히 사용되는 용어를 **SRS**에 부록으로 기재합니다.

I SRS로부터 초기 Use Case diagram을 생성

얻어진 **use case**를 단순한 **Use Case Diagram**으로 그려냅니다.

이것도 **SRS**에 포함시킵니다.

참고하세요

1. **SRS (System Requirements Specification)** : 시스템 요구 사항 명세서 – 프로젝트에 관한 모든 내용 즉, 참가자, 역할, 기능적 요구 사항, 비기능적 요구 사항, 위험 요소, 제약 사항, 용어 등을 총망라해서 정리 해 놓은 문서
2. **domain** : 프로젝트 영역

심화학습

1. SRS (System Requirements Specification) : 시스템 요구 사항 명세서의 특징

i. SRS는 정확해야 합니다.

이것은 문서의 유효성을 의미하기도 합니다. 문서가 어떤 프로젝트에 관한 것이고 언제 어떻게 작성된 것인지 정확해야 합니다.

ii. SRS는 명확해야 합니다.

이것은 문서가 프로젝트 참가자 전원에게 이해 될 수 있어야 함을 말합니다. 특정인에게 국한 되거나 모호한 표현은 문서의 의미 전달을 어렵게 합니다.

iii. SRS는 완전해야 합니다.

모든 내용에 대해서 전부 문서화 되어 있어야 함을 의미합니다. 그러나 모든 내용이 초반부터 완전히 문서화되어 지는 것은 어렵습니다. 하지만 가능한 한 모든 내용을 다룰 수 있도록 구조화합니다.

iv. SRS는 일관되어야 합니다.

문서 안에서 요구되어 지는 내용이 서로 상충되어서는 안됩니다. 설사 충돌하는 요구 사항이 있다면 그것은 논리적이거나 일시적이어야 합니다.

v. SRS의 내용에는 순위가 있습니다.

문서 안에서 요구 되어 지는 내용은 순위가 있습니다. 순위는 중요성 혹은 안정성으로 따질 수 있습니다. 이 순위가 높은 요구 사항이 먼저 개발 되어야 하며 계속해서 검증 되어야 합니다.

vi. SRS는 증명 가능해야 합니다.

시스템이 요구 하는 영역이 실행 혹은 구현 가능해야 함을 의미합니다.

vii. **SRS**는 수정 가능해야 합니다.

요구 사항은 계속 변화할 수 있습니다. 문서는 이것을 수용할 수 있어야 합니다.

viii. **SRS**는 추적 가능해야 합니다.

각 요구 사항은 그 근원과 원리를 추적할 수 있어야 합니다.

2. 요구사항 분석 단계의 목적과 산출물

1) 요구사항 분석 단계의 목적과 Job Roles

이 단계에서는 요구 사항 모델의 두 가지 뷔가 있습니다.

| 정밀한 **Use Case Diagram**

| **Domain Model** (**problem space**의 **key abstraction**을 통한 **Class Diagram**)

요구 사항 수집 단계에서 추출해낸 초기 **Use Case diagram**은 아주 단순화된 표현이기 때문에 세부적인 내용을 위해서는 요구 사항을 분석하여야만 합니다.

problem space는 클라이언트들이 그들의 비즈니스를 일컫는 용어입니다.

예를 들어 호텔 예약 시스템의 **problem space**에서는 고객, 룸, 호텔 부대 시설, 지불 방법, 결재 수단 등등의 용어를 사용합니다. 이러한 용어는 **key abstraction**과정에서 추출되어 클래스로 될 수 있습니다. 이것을 가시화한 것을 **Domain Model**이라고 합니다.

요구 사항 분석 단계의 목적은 현재의 비즈니스 과정을 모델하는 것입니다.

Workflow	목적	설명
요구 사항 분석	현재의 비즈니스 과정 을 모델하기 위함	결정 할 내용: 정교한 Use Case Diagram 의 내용 problem domain 안에서 무엇 을 key abstraction 할 것인 지

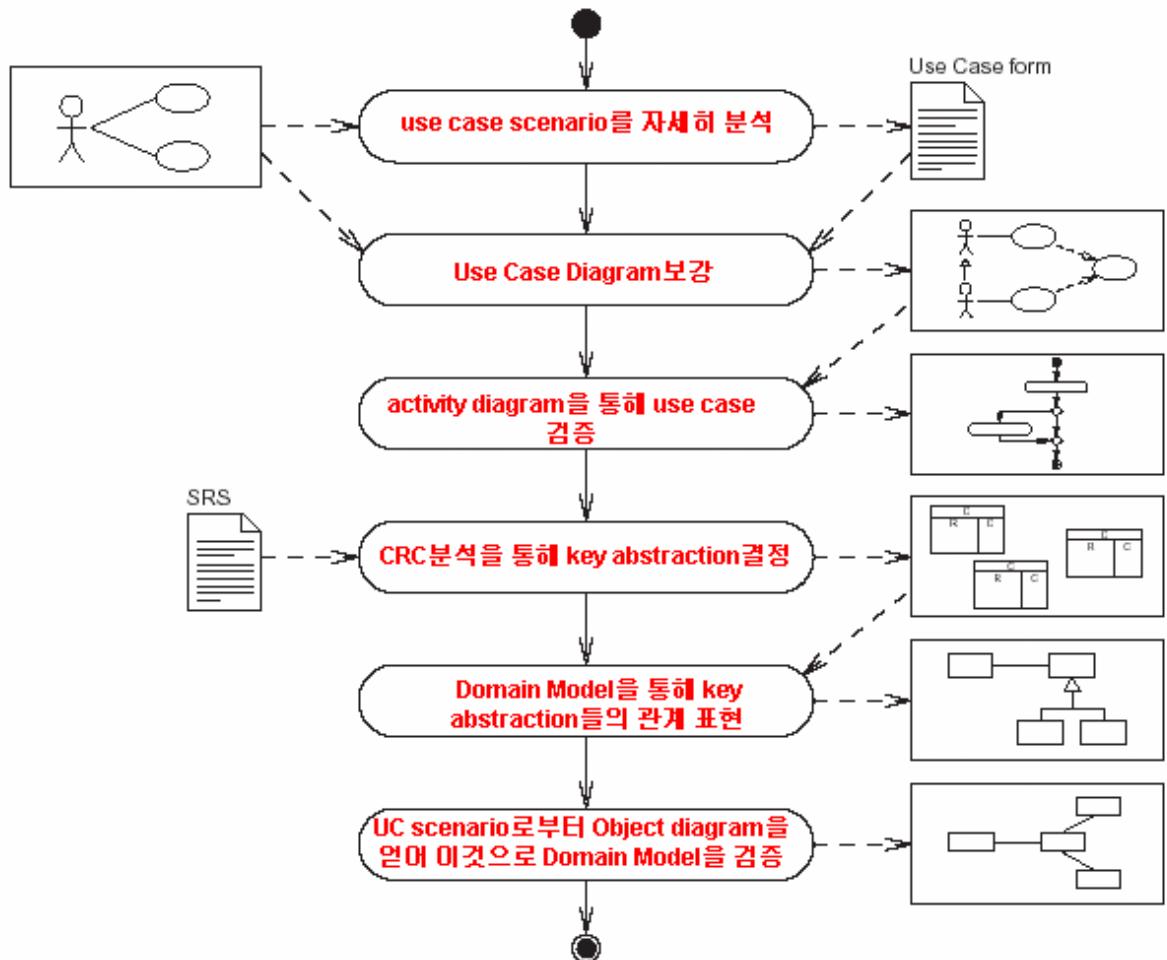
요구 사항 수집 단계에서와 마찬가지로 요구 사항 분석 단계에서의 **Activity**는 주로 **business analyst**와 **architect**가 수행합니다.

2) 요구 사항 분석 단계의 Activities와 Artifacts

다음 그림은 요구 사항 분석 단계에서의 **activity**와 **artifacts**를 보여 줍니다.

[이미지]

n 요구 사항 분석 단계의 Activities와 Artifacts



I use case scenario를 자세히 분석

Use case scenario란 use case를 아주 세밀히 관찰한 내용을 써 놓은 문서를 말합니다. 이런 시나리오들은 일반적으로 공통된 패턴을 가지며 작성되기 때문에 보기 편한 형태로 재 작성한 것이 Use Case Form입니다.

I Use Case diagram 보강

정밀한 Use Case Diagram을 만드는 작업을 말합니다. 한 actor와 다른 actor와의 관계, 한 use case와 다른 use case와의 관계를 세밀히 표현합니다.

예를 들어, **log-on use case**가 다른 유즈케이스와 함께 작업 해야 한다면 그것을 모두 표현하는 것이 바로 이 단계입니다.

I Activity diagram을 사용하여 use case를 검증

모델을 검증하는 것은 아주 중요하고 필요한 작업입니다.

검증되지 않으면 나중에 되돌리기 어렵기 때문입니다. **use case**를 검증하기 위해선 **Use Case Form**을 보면서 다른 그림을 그려보는 것이 최선의 방법입니다.

Use Case Form은 한 **use case**를 실행하기 위한 세밀한 정보가 들어 있는 문서이며 이를 통해 **use case**를 수행하기 위한 동작(activity)를 자세히 유출해 낼 수 있을 것입니다. 바로 이 그림을 **Activity diagram**이라고 하며 **activity diagram**을 그리면서 이 **use case**가 실현 가능한지를 검증해 볼 수 있습니다.

I CRC분석을 통해 key abstraction을 결정

CRC란 **Class Responsibility Collaboration**의 약자입니다.

class를 얻어 내기 위해서는 요구 사항 수집 단계에서 작성해 놓은 **SRS(System Requirements Specification)**를 가지고 **key abstraction**하는 과정을 거쳐야 합니다. 간략히 말하면 **SRS** 문서에서 명사만 추출해 내어 이것이 서로 어떤 관계가 있는지, 실제로 구체적인 값을 갖게 될 것인지 아닌지, 등등의 분석을 하게 되는데 이러한 분석을 **CRC분석**이라고 합니다. 이 분석 단계를 거치면 우리가 추출해놓았던 명사들은 서로 포함하고 있기도 하고 삭제되기도 합니다. 이때 최종적으로 남아 있게 되는 명사 중에 독립적인 수행 단위가 될만한 것들(어떤 값을 포함하고 있거나 수행 로직을 가져야 할 만한 것)을 클래스로 도출하게 되는 것입니다.

I Domain Model을 통해 Key abstraction들의 관계 표현

위 **activity**를 통해 **class**가 도출되어졌으면 그것을 그림으로 그려봅니다. 클래스들은 서로 연관성을 가지면서 그려질 수도 있을 것입니다. 이 그림을 **Domain Model**이라고 합니다.

I Use Case scenario로부터 Object diagram을 얻어 이것으로 Domain Model을 검증

Domain Model도 또한 검증되어야만 합니다.

이미 작성되어 있던 **Use Case scenario**를 통해 **Domain Model**과 상관없는 다른 그림을 그려봅니다.

그런데 **Use Case scenario**는 어떤 **use case**를 실행하기 위한 세부적인 내용을 담고 있는 것이기 때문에 상당히 구체적일 수 밖에 없습니다. 예를 들어, <고객이 호텔방을 예약한다>는 **use case**는 시나리오 상에서 <전은수라는 고객이 호텔에서 해당 날짜에 투숙 가능한 2인용 디럭스룸 하나를 예약한다>로 상당히 구체화 되어집니다. 이런 시나리오를 바탕으로 그림을 그리게 되면 구체적인 정보들(객체)간의

관계가 표현됩니다. 이것이 **Object Diagram**입니다. 이것은 명사 추출만으로 얻은 **Domain Model**을 검증하는데 아주 유용합니다.

3. 아키텍쳐 구축 단계의 목적과 산출물

1) 아키텍쳐 구축 단계의 목적과 Job Roles

아키텍쳐 워크플로우는 다른 단계보다 훨씬 복잡합니다. 그러나 이장에서는 아키텍쳐 워크플로우가 디자인 워크플로우에 미치는 영향 정도만 설명하도록 할 것입니다. 아키텍쳐 구축에 대한 심화 학습은 [단원 5. 비기능적 요구 사항 구축]의 전 모듈에서 다루게 됩니다.

디자이너 입장에서 봤을 때 아키텍쳐 모델은 시스템의 뼈대가 되는 구조를 말합니다. 디자이너들은 이 뼈대에 살(**Component**)을 붙이는 작업을 하게 되는 것입니다.

아키텍쳐 구축 단계의 목적은 프로젝트의 리스크를 산정하고 그것을 감소 시킬 수 있는 시스템 구조를 모델하기 위함입니다.

Workflow	목적	설명
아키텍쳐 구축	비기능적 요구 사항을 만족시키기 위한 시스템의 고수준 구조를 모델하기 위함	소프트웨어 솔루션의 최상위 레벨 구조를 개발. 아키텍쳐 모델을 지원할 기술 선정 비기능적 요구 사항을 만족시키기 위한 아키텍쳐 패턴을 가지고 정밀한 아키텍쳐 구축

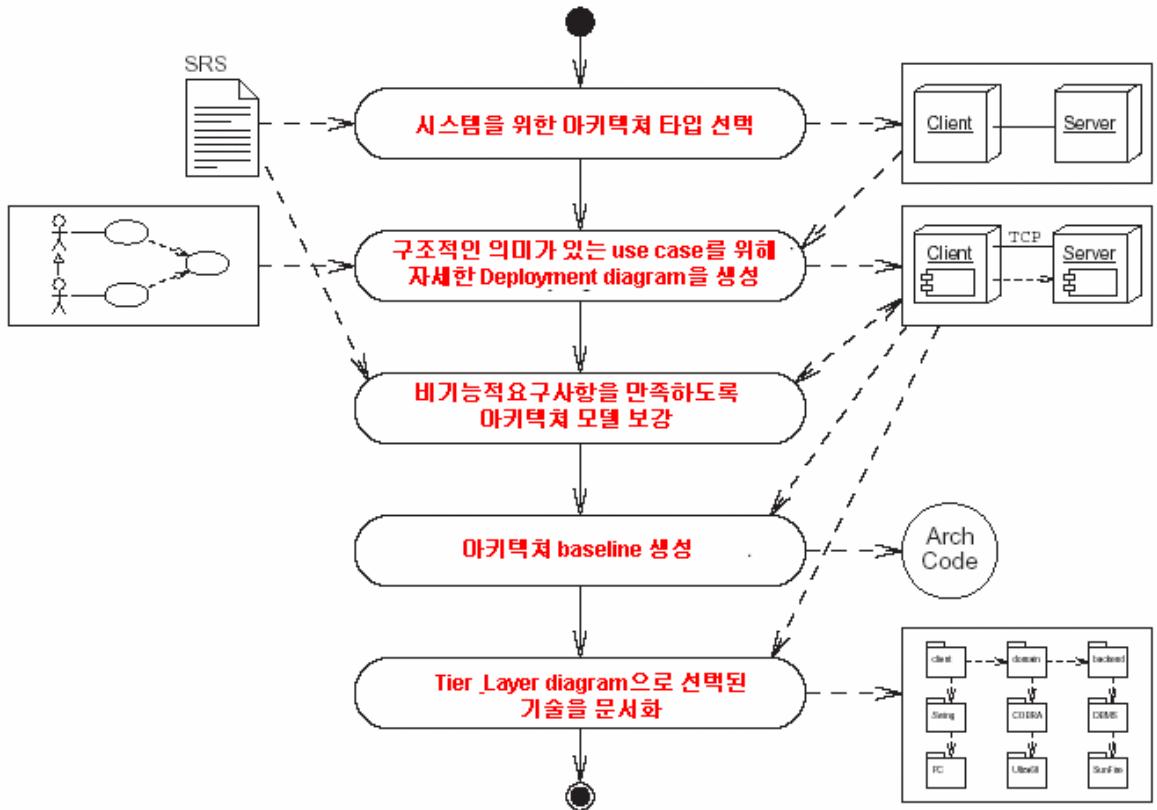
아키텍쳐 구축 단계에서의 Activity는 architect가 수행합니다.

2) 아키텍쳐 구축 단계의 Activities와 Artifacts

다음 그림은 아키텍쳐 구축 단계에서의 **activity**와 **artifacts**를 보여 줍니다.

[이미지]

n 아키텍쳐 구축 단계의 Activities와 Artifacts



I 시스템을 위한 아키텍쳐 타입 선택

아키텍트는 비기능적 요구 사항과 고수준 제약사항을 만족시키기 위한 최선의 시스템 아키텍쳐 타입을 선택해야 합니다. 아키텍쳐 타입이란 **standalone**, **client/server**, **App- centric**, **n- tier**, **Web- centric n- tier**, **Enterprise n- tier** 같은 추상적인 아키텍쳐의 가장 작은 단위입니다.

선택된 아키텍쳐 타입을 기반으로 하여 여러 하드웨어 노드에 분산되어 있는 컴포넌트를 보여주는 **Deployment Diagram**이 작성될 수 있습니다.

I 구조적인 의미가 있는 use case를 위해 자세한 Deployment Diagram 생성

아키텍트는 구조적인 의미가 있는 **use case**를 지원하기 위해 필요한 주요 컴포넌트를 보여 주는 보다 정밀한 **Deployment Diagram**을 그려야 합니다.

이 다이어그램은 컴포넌트 간의 의존도도 간략히 보여 줄 수 있습니다.

I 비기능적 요구 사항을 만족하도록 아키텍쳐 모델 보강

아키텍트는 비기능적 요구 사항을 만족 시키기 위하여 견고한 하드웨어 구조를 가진 고수준 **아키텍쳐 패턴**을 적용할 수 있습니다. 아키텍쳐 패턴에 대한 얘기는 단원 5에서 다루어 집니다.

I 아키텍쳐 baseline 생성

아키텍트는 소프트웨어 진화의 원형이 되는 아키텍쳐 베이스라인을 생성하고 테스트하여 **Construction(구축)** 단계의 시작점(**start point**)이 되도록 해야합니다. 이 과정을 거치면 아키텍쳐에 관한 최소 코드가 만들어지게 됩니다.

I Tier & Layer Diagram으로 선택된 기술을 문서화

시스템에 적용될 기술을 총망라한 **Tier & Layer Diagram**을 그려 문서화합니다.

4. 디자인 단계의 목적과 산출물

1) 디자인 단계의 목적과 Job Roles

디자인 워크플로우의 최고의 목적은 솔루션 모델을 개발하기 위함입니다. 솔루션 모델을 통해 원하는 시스템을 구축할 수 있습니다.

디자인 모델은 **use case**의 **activity flow**를 만족시키는 소프트웨어 컴포넌트의 추상적인 모델입니다. 디자인 모델안에서 컴포넌트는 직접적으로 코드로 구현될 수 없습니다. **use case**를 위한 디자인 모델은 **activity**의 강력 분석을 통해 얻어집니다.

디자인 모델은 아키텍쳐 모델과 결합되어 솔루션 모델이 됩니다.

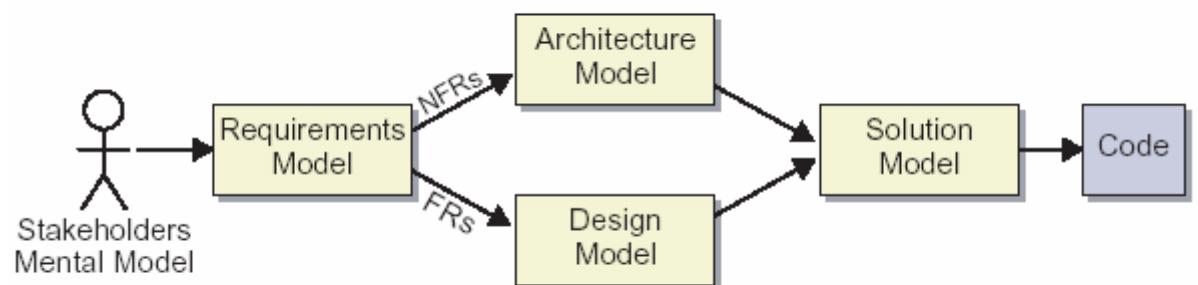
솔루션 모델은 보강된 **Domain Model**도 포함하고 있습니다. 솔루션 모델은 다양한 디자인 패턴을 통해 보강될 수도 있습니다.

솔루션 모델 안의 컴포넌트는 코드화 될 수 있습니다.

디자인 모델은 요구 사항 모델과 솔루션 모델 사이에 있습니다.

이미지

n 개발단계별 모델



디자인 단계의 목적은 요구 사항을 만족시키는 시스템을 구축할 수 있는 솔루션 모델을 생성하기 위한 것입니다.

Workflow	목적	설명
디자인	시스템이 use case 를 지원하는 방법을 모델하기 위함	<ul style="list-style-type: none"> use case를 위한 디자인 모델 생성 솔루션 모델 생성 Domain Model(도메인 모델) 보강 솔루션 모델과 도메인에 디자인 패턴 적용 복잡한 객체 상태를 모델

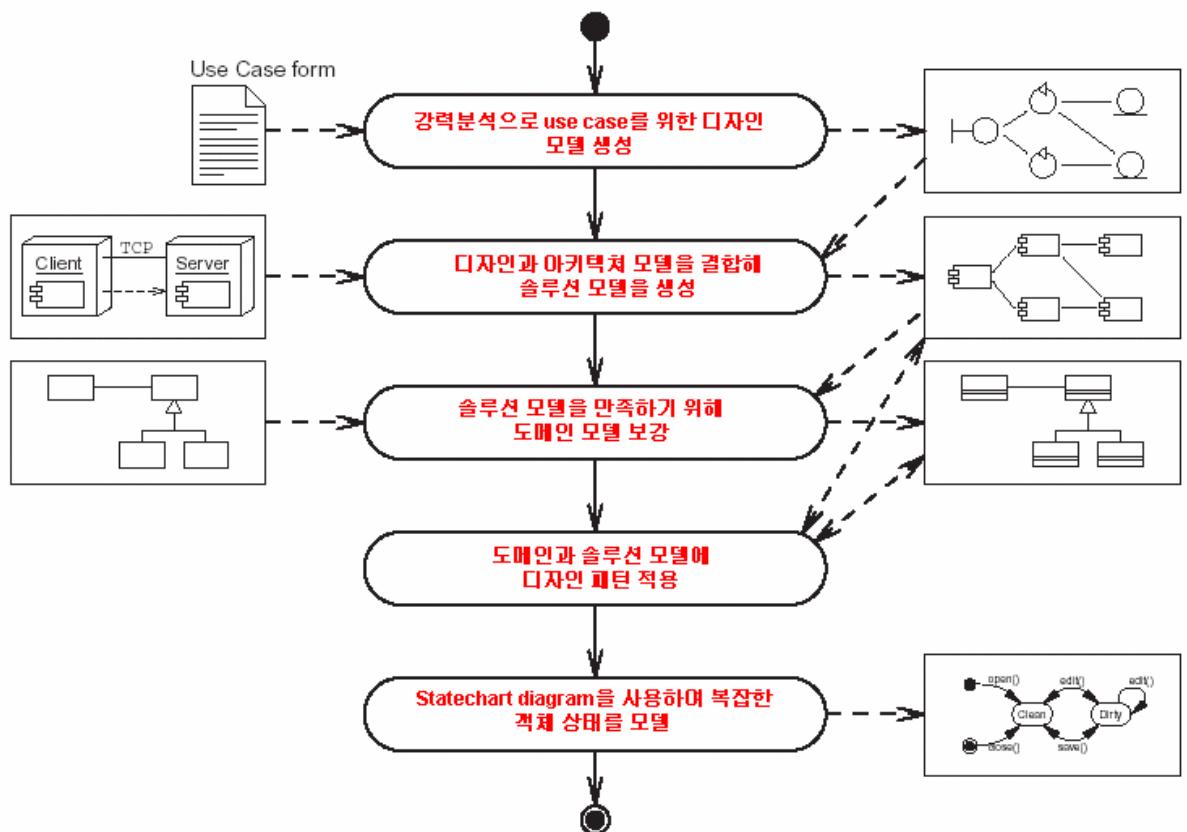
디자인 단계에서의 **Activity**는 **software designer**가 수행합니다.

2) 디자인 단계의 Activities와 Artifacts

다음 그림은 디자인 단계에서의 **activity**와 **artifacts**를 보여 줍니다.

[이미지]

n 디자인 단계의 Activities와 Artifacts



| 강력분석으로 use case를 위한 디자인 모델 생성

이 단계에서는 유즈케이스 시나리오를 통해 강력분석을 한 뒤 그 **use case**를 만족시키기 위한 소프트웨어 컴포넌트는 어떤 것이 필요하며 컴포넌트들 간의 관계는 어떤지에 대해 파악하고 그림을 그립니다. 이것을 디자인 모델이라고 합니다.

| 디자인과 아키텍쳐 모델을 결합해 솔루션 모델 생성

아키텍쳐 워크플로우에서 그렸던 아키텍쳐 모델과 위의 디자인 모델을 결합해 솔루션 모델을 만들어냅니다. 이러한 구조는 구현 단계에서 얻어져야 하는 컴포넌트 코드를 **80%** 정도 제공합니다.

| 솔루션 모델을 만족하기 위해 도메인 모델 보강

요구 사항 분석 단계에서의 도메인 모델은 클래스의 변수명과 메소드명 정도만을 가지고 있는 간략한 것이었습니다. 이 단계에서 디자이너들은 구현에 필요한 모든 정보를 가지고 있는 자세한 도메인 모델을 구축해야 합니다.

| 도메인과 솔루션 모델에 디자인 패턴 적용

현재 수백가지의 디자인 패턴이 문서화되어 있습니다. **이 단계에서 디자이너가 해야 할 일은 좀 더 유연하고 견고한 소프트웨어를 만들기 위한 적당한 패턴을 찾아내어 솔루션 모델과 도메인 모델에 적용하는 것입니다.**

| Statechart diagram을 사용하여 복잡한 객체를 모델화

경우에 따라서, 시스템은 현재 객체의 상태가 상황에 따라 변화할 수도 있습니다. 이러한 객체 상태의 변화를 쉽게 이해할 수 있도록 모델화하는 것이 필요합니다.

5. 시스템 구축 단계의 목적과 산출물

1) 시스템 구축 단계의 목적과 Job Roles

이 과정에서 시스템 구축단계는 다음 세가지를 포함합니다.

| 구현(Implementation)

구현 단계의 목적은 솔루션 모델안에 정의된 소프트웨어 컴포넌트를 만들어 내는 것입니다.

구현 단계에서의 activity는 **software programmer**가 수행합니다.

| 테스트(Testing)

테스트 단계의 목적은 원하는 기능을 수행할 수 있는가를 검진하기 위함입니다.

테스트 단계에서의 activity는 **software tester**가 수행합니다.

| 배치(Deployment)

배치 단계의 목적은 생산 환경 속으로 구축된 시스템을 배치하기 위함입니다.

배치 단계에서의 activity는 deployment specialist가 수행합니다.

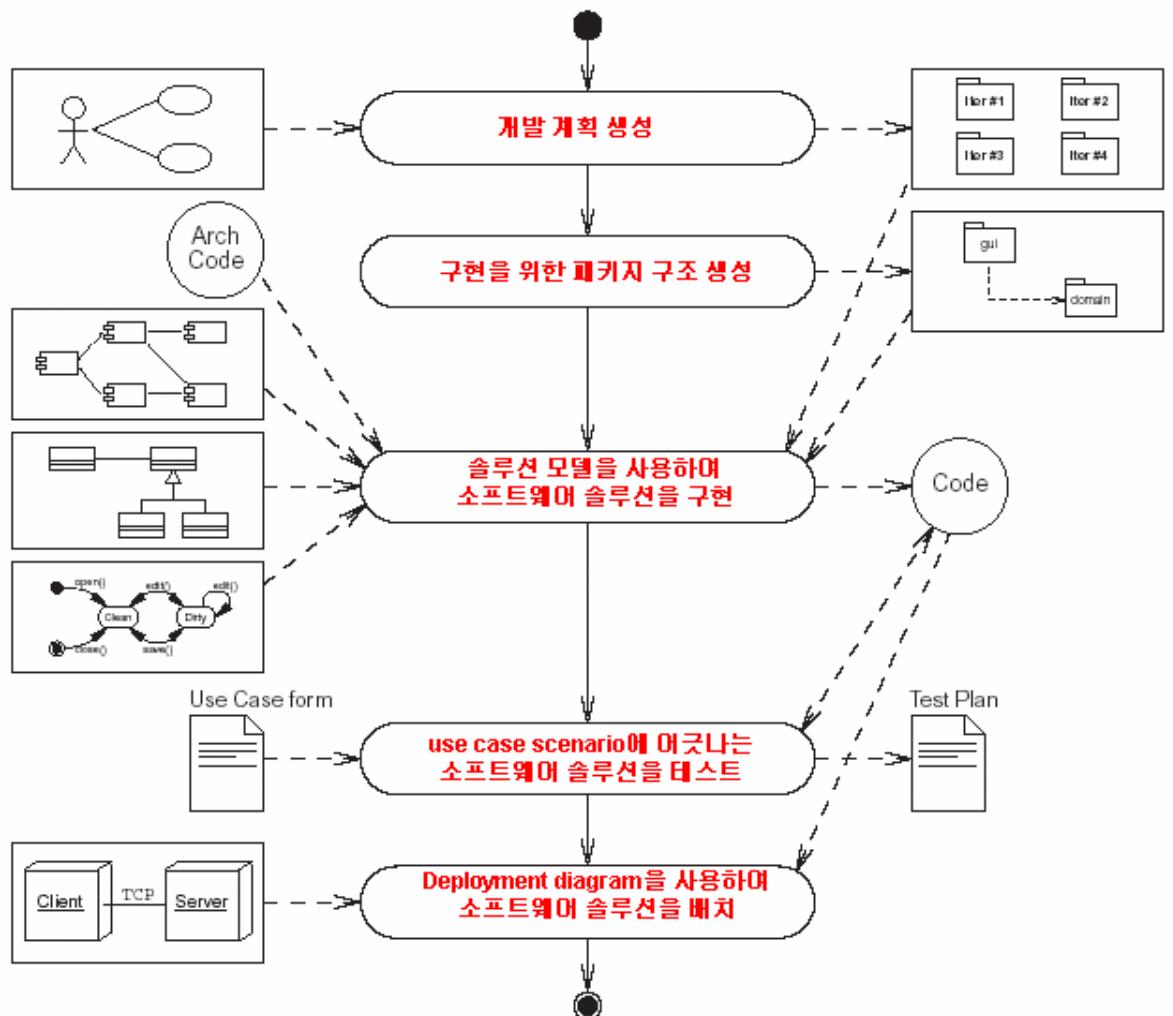
Workflow	목적	설명
시스템 구축	시스템을 구현, 테스트, 배치하기 위함	<ul style="list-style-type: none"> 소프트웨어 구현 테스트 수행 Domain Model(도메인 모델) 보강 생산 환경으로 소프트웨어 배치

2) 시스템 구축 단계의 Activities와 Artifacts

다음 그림은 시스템 구축 단계에서의 **activity**와 **artifacts**를 보여 줍니다.

[이미지]

n 시스템 구축 단계의 Activities와 Artifacts



| 개발 계획 생성

이 단계에서는 **use case**의 우선 순위에 따른 개발 계획을 세워서 개발팀 내에서 원활한 분업이 일어 날 수 있도록 해야 합니다.

| 구현을 위한 패키지 구조 생성

이 단계에서는 아키텍처 모델에 정의된 패키지 구조를 실제 **Java technology package**로 매핑합니다.

| 솔루션 모델을 사용하여 소프트웨어 솔루션을 구현

이 단계에서는 보강된 도메인 모델에 정의된 대로 자바 클래스를 만들어야 합니다.

| **use case scenario**에 어긋나는 소프트웨어 솔루션을 테스트

테스트 요소는 아주 다양하고 많습니다. 일단 원래 목적했던 **use case**가 **use case scenario**대로 완전히 만족되는지를 테스트하고 최종적으로 이 시스템의 기능적 요구 사항이 **use case**와 일치하는지를 테스트합니다.

| Deployment Diagram을 사용하여 소프트웨어 솔루션을 배치

이 단계에서 **deployment specialist**는 구축된 소프트웨어 솔루션을 **Deployment Diagram**을 사용하여 각 하드웨어에 배치 해야 합니다.

과정명

OO- 226 Object- Oriented Analysis and Design Using UML

단원1 : OOSD Process와 객체지향 방법론

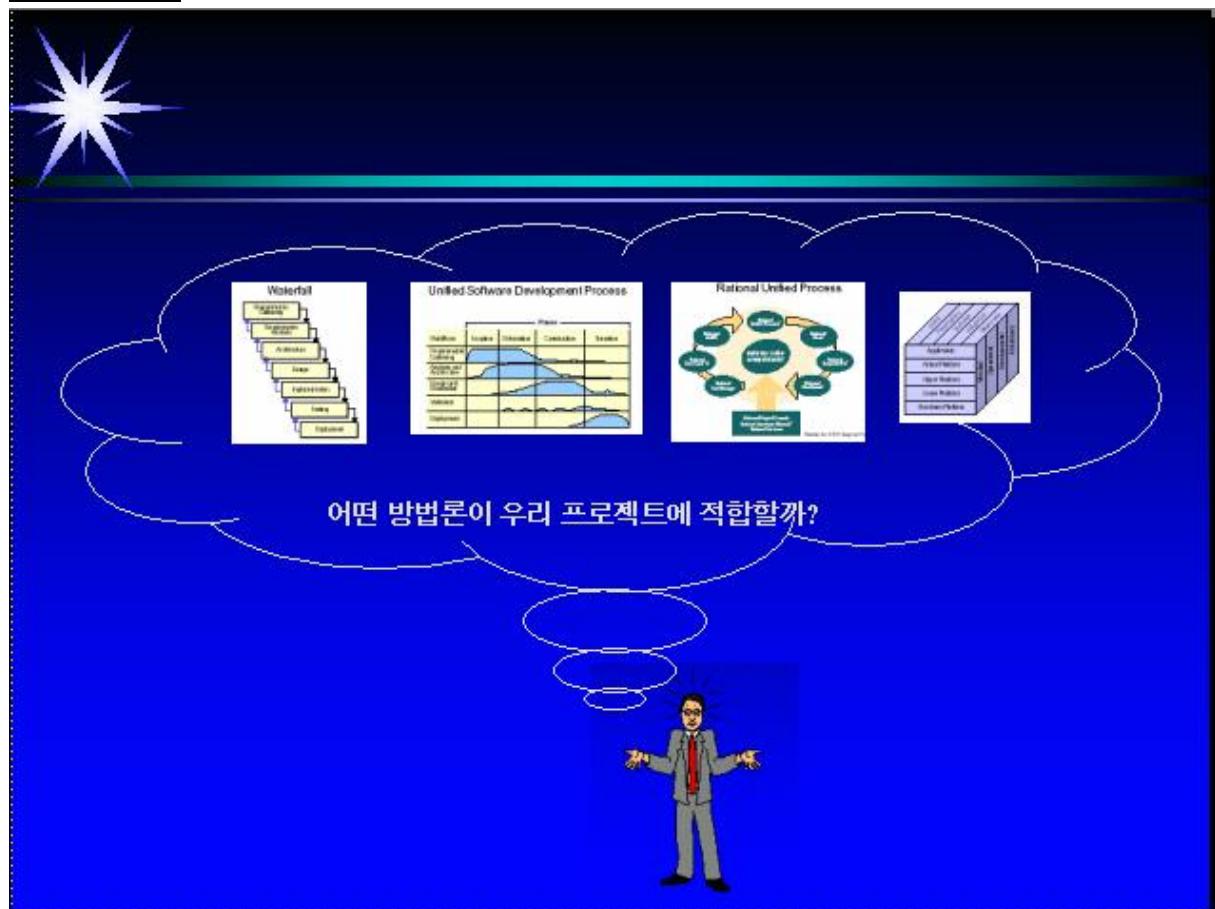
4 모듈 : 객체지향 방법론과 올바른 선택

담당강사 : 전은수

■ 생각해봅시다 ■

여러분들은 지금 소프트웨어 개발 과정에 참여하고 있습니까? 그렇다면 혹시 어떤 방법론을 사용하고 계십니까? 선택하신 방법론의 장단점을 잘 파악하고 계신가요? 만약 현재 프로젝트에 다른 방법론을 사용하신다면 어떤 면에 중점을 두어 선택하시겠습니까? 객체지향 개념이 대두된 이래 꾸준히 객체지향 방법론도 발전되어 왔습니다. 현재 그 종류는 무수히 많고 각 소프트웨어 개발팀마다 자신들의 독특한 개발 방법론을 사용하기도 합니다. 그렇다면 객체지향 방법론에는 어떤 것이 있고 어떤 장단점이 있으며 각 프로젝트 상황에 맞게 선택하는 방법은 무엇일까요? 이 과정에서 그 해답을 제시합니다.

애니메이션



■ 학습하기 ■

1. 소프트웨어 방법론의 올바른 이해

1) 소프트웨어 방법론이란

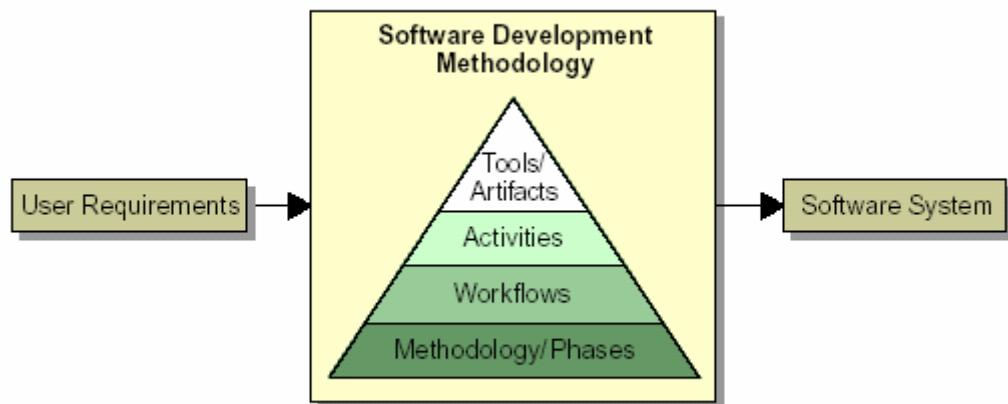
소프트웨어 방법론이란 소프트웨어 개발 과정에서 가장 기본이 되는 구조를 의미합니다. 이 구조는 일반적으로 **phases, workflow, activities, artifacts**로 상층화 됩니다.

다음 그림에서 보듯이 사용자 요구 사항을 수용한 소프트웨어 시스템을 구축하기 위해서는 잘 짜여진 어떤 단계들을 거쳐 일률적이고 조직적인 작업을 하면서 그 산출물을 얻게 되는데 이때 프로젝트 상황에 맞는 적합한 구조여야만 최상의 산출물을 얻어낼 수 있습니다.

우리는 다양한 개발 단계를 프로젝트 유형에 맞게 제시한 구조를 소프트웨어 방법론이라 합니다.

[이미지]

□ Software Development Methodology



[참고하세요]

□ Phases의 동의어

어떤 개발자들은 **Phases**를 **macro- phase**라고, **workflows**를 **micro- phase**라고도 부릅니다.

2. 좋은 방법론이 되기 위한 조건

1) 방법론의 원리

현재까지 수많은 객체지향 개발 방법론이 제시되어 왔습니다. 최근에는 그 방법론의 평가도 함께 이뤄지고 있는 추세입니다. 개발 방법론이 일정한 규칙과 목적을 가지고 만들어 져야만 그 방법론에 의거한 소프트웨어 개발이 순조롭고 목적에 맞게 진행될 수 있을 것입니다.

따라서 이 과정에서는 방법론의 종류와 장단점을 먼저 파악하기 보다 어떤 방법론이 어떤 원리를 바탕으로 제시되었는지를 원론적으로 알아보겠습니다.

특별히 객체지향 개발 방법론들은 다음과 같은 원리에 주안점을 두고 제시됩니다.

- | **Use- Case- Driven**
- | **Systemic- quality- driven**
- | **Architecture- centric**
- | **Iterative and incremental**
- | **Model- based**
- | **Design best practices**

i. Use- Case- Driven

“A software system is brought into existence to serve its users.” (Jacobson USDP page 5)

소프트웨어 시스템이 어떻게 만들어지는지 알기 위해서는 먼저 시스템이 사용자를 위해 무엇을 하는지를 이해해야 합니다.

이것은 시스템의 기능 즉, 유즈케이스(**Use- Case**)라고 하는데 개발 프로세스 전체가 이 유즈케이스 중심적이어야만 최종적으로 시스템이 제대로 기능을 수행 할 수 있습니다.

이를 **Use- Case- Driven**이라 합니다.

이 원리의 핵심 사항은 다음과 같습니다.

1. 모든 소프트웨어는 ‘사용자’를 갖는다는 것입니다.

사용자는 인간일 수도 있고 또 다른 기계일 수도 있습니다. 이를 통칭하여 **actor(액터)**라고 합니다. 이 과정에서는 **user(유저)**와 **actor(액터)**를 같은 의미로 사용하겠습니다. 액터는 시스템의 바깥 영역에 있는 엔티티입니다. 예를 들어 신용카드 인증 시스템이라는 소프트웨어 시스템 영역을 사용하는 사용자는 인간일 수도 있고 카드 입력 단말기 일 수도 있습니다.

이때 인간이나 단말기는 시스템 영역 바깥에 위치하는 엔티티라고 표현합니다.

2. 사용자는 행동을 하거나 목적을 얻기 위해 소프트웨어를 사용합니다.

우리는 이것은 유즈케이스(**use case**)라고 부릅니다. 어떤 방법론에서는 **user stories**라고 하기도 합니다. 그러나 개념은 같습니다.

3. 소프트웨어 개발 방법론은 **use case**를 쉽게 할 수 있는 시스템을 지원합니다

다.

적합한 방법론을 통해 개발되는 소프트웨어 시스템은 원래 그 시스템이 가져야 하는 **use case**를 가장 쉬운 방법으로 획득되고 구축되게 도울 수 있습니다.

4. 유즈케이스는 시스템의 디자인을 결정짓습니다.

어떤 유즈케이스의 특별한 특징을 구현하기 위해서는 소프트웨어 컴포넌트 세트가 요구되어 질 수 있습니다. 이때 컴포넌트를 어떻게 구축할 것인지에 대한 일정한 설계(디자인)가 이뤄져야 합니다.

ii. Systemic-quality-driven

소프트웨어 시스템의 요구는 기능적 요구뿐만 아니라 비기능적 요구도 포함합니다. **System-Quality**란 **SunTone Architecture**(:방법론의 일종)에서 얘기하는 비기능적 요구 사항의 표현입니다. 이는 “서비스의 질”이라고 생각하시면 쉽습니다. **System-Quality**는 다음을 포함합니다.

1. **performance** (속도) – 응답시간
2. **Reliability**(신뢰성) – 컴포넌트 실패의 감소
3. **Scalability** (확장성) – 보다 많은 사용자를 수용할 수 있는 능력

Systemic qualities는 소프트웨어 아키텍처를 결정합니다. 아키텍처 기준이 결정된 후에 소프트웨어 디자이너가 이 아키텍처 속으로 들어갈 기능적 요구에 대해서 디자인 하게 됩니다.

어떤 소프트웨어든 제대로 만들어지기 위해서는 비기능적 요구 사항까지도 모두 만족해야 합니다.

iii. Architecture-centric

소프트웨어 프로젝트에서 위험 요소의 상당한 부분은 확장성(**scalability**), 유용성(**availability**), 신장성(**extensibility**)등 과 같은 서비스의 질적인 면에서 발생합니다.

아키텍처 워크플로우는 이러한 문제들을 다루어야 합니다.

그러므로 가능한 초기에 아키텍처 워크플로우를 제대로 수립하는 것은 좋은 방법입니다. 이것은 초기에 위험을 감소 시킵니다.

어떤 방법론이 아키텍처 수립에 중점을 둔다면 다음과 같은 것을 고려해야 합니다.

① **Systemic Qualities**는 아키텍처 컴포넌트와 패턴을 좌우합니다.

아키텍처 구조는 비기능적 요구 사항을 만족시키기 위한 방향으로 컴포넌트와 패턴을 결정짓습니다.

예를 들어, **Scalability**(확장성)은 다양한 분산 환경으로 서버를 구축함으로써 더 많은 서비스를 제공할 수 있습니다.

② **Use case**는 아키텍쳐에 합당해야 합니다.

아키텍쳐 구조는 모든 기능적 요구 사항을 만족 시키는 소프트웨어 컴포넌트를 디자인 하는 데에도 적합해야 합니다. 비기능적 요구 사항을 만족시키는 아키텍쳐가 정작 기능적 요구 사항을 수용하지 못한다면 좋은 방법론이라 할 수 없습니다.

예를 들어, 웹어플리케이션에서 사용자 인터페이스 컴포넌트는 **Model 2** 아키텍쳐 구조에서 서블릿과 **JSP**로 디자인 되게 될 것입니다.

① 아키텍쳐 중심적인 방법론들의 요소

아키텍쳐 중심적인 방법론들은 다음과 같은 요소에 역점을 둡니다.

① Tier(티어)

소프트웨어 시스템은 몇 개의 논리적인 구분을 갖는 컴포넌트로 그룹화되어 있습니다.

클라이언트 인터페이스를 제공하는 컴포넌트 그룹을 클라이언트 티어(**Client tier**), 모든 비즈니스 로직과 도메인 엔티티 컴포넌트 그룹을 비즈니스 티어(**business tier**), 비즈니스 엔티티의 영구적인 관리를 위한 컴포넌트 그룹을 리소스 티어(**Resource tier**)라고 합니다.

② Tier Components와 communication Protocol

근접한 티어의 컴포넌트끼리는 서로간에 작업을 교류하게 됩니다. 이때 어떤 방식으로 커뮤니케이션 할 것인가에 대해 정의한 것을 **프로토콜(Protocol)**이라 합니다.

프로토콜은 아키텍쳐 수립 시 아주 논쟁적인 요소가 될 수 있습니다.

③ Layer

레이어는 어플리케이션을 구축하기 위한 시스템의 수직적 구조를 말합니다. 보통 **application Layer**, **platform Layer**, **hardware Layer**로 나뉩니다.

컴포넌트 기반 시스템에서 어플리케이션 레이어는 특별한 **API**로 구성되어집니다. 예를 들어, 자바기술로 웹어플리케이션을 구축한다면 어플리케이션 레이어는 **Servlet & JSP API**로 구성될 것입니다.

어플리케이션 레이어는 플랫폼에 영향 받을 수 있습니다. 또한 플랫폼은 하드웨어에 종속적일 수 있습니다. 그러므로 시스템

레이어는 아키텍쳐 수립의 중요한 이슈가 됩니다.

iv. Iterative and incremental

반복적인 개발은 시스템을 점점 더 크고 막강하게 만듭니다.

요구 사항이 변했는데 이것이 소프트웨어 개발 내용에 포함되어 있지 않은 것이라면 어떻게 하겠습니까?

어떤 경우에는 여러분들이 처음 이해한 시스템의 요구 사항이 완전하지 않을 수도 있습니다.

반복 개발이란 이런 상황에서도 실패 없이 프로젝트를 수행해 나갈 수 있는 기회를 제공해 주는 것입니다.

방법론을 제시할 때 다음과 같은 점을 염두에 둔다면 보다 좋은 소프트웨어를 개발할 수 있는 길을 제시하게 되는 것입니다.

| 반복 단계마다 완전한 **OOSD life cycle**을 포함해야 합니다.

OOSD life cycle이란

분석(analysis)- 설계(design)- 구현(implementation)- 테스트(test)를 거치는 개발 과정을 말합니다.

각 반복 단계에서는 **OOSD workflow**를 수행해야 합니다. **OOSD workflow**는 요구사항 수집, 요구 사항 분석, 아키텍쳐 수립, 디자인, 구현 테스트로 이루어집니다.

이때 구현과 테스트보다 분석과 아키텍쳐 수립에 더 많은 시간을 할애하게 됩니다. 그러나 각 반복 단계의 끝에는 비록 불완전 하기는 하지만 통합과 실행을 통해 요구 사항이 충분히 반영되었는지, 추가적인 기능이 있는지를 알아보게 됩니다.

개발 초기에 시스템의 실행을 보는 것은 개발팀 내에 작업의 추진력을 갖게 하고 클라이언트에게 실제 얼마나 진행되었는지를 보여줄 수 있는 기회를 제공합니다.

| 모델과 소프트웨어는 여러 번 반복되면서 점진적으로 확대됩니다.

요구 사항 스펙이 완벽하지 않기 때문에 시스템의 모델도 완벽할 수가 없습니다. 이러한 제약 사항은 아래 상황과 중요한 연관관계를 맺게 됩니다.

첫째, 모델을 만들 때는 오직 개발팀내의 커뮤니케이션이 가능하게 하는 수준에서 만들어야만 합니다. 너무 많은 모델과 디아그램을 만드는 것은 개발 기간을 늘리는 결과를 초래합니다.

둘째, 클라이언트는 적당한 목표치를 가져야 합니다.

만약 클라이언트가 사용자 인터페이스를 본다면 시스템이 구축되기 전에 좀 더 확실한 진단을 내릴 수 있을 것입니다.

I. 유지보수는 단순히 다른 반복입니다. (또는 반복의 시리즈입니다)

유지보수는 워크플로우가 아닙니다. 그러나 시스템을 완전히 수정하는 것보다 유지보수하는 편이 훨씬 낫습니다. 버그를 잡는 간단한 유지보수는 ‘반복’을 한번만 수행하는 것으로도 가능할 수 있습니다. 좀 더 복잡한 유지보수는 ‘반복’을 여러 번 수행하도록 합니다.

V. Model-based

모델이란 프로젝트의 모든 참가자들이 의사 소통 할 수 있는 가장 기본적인 의미입니다. 산출물은 우리들 마음속의 모델(**mental model**)을 물리적으로 표현한 것입니다.

모델링하는 것이 인간의 생각을 표현하는 방법으로 가장 적당합니다. 모든 인간은 실체에 대한 마음속 모델을 갖고 있고 이것을 개발하는 것이 시스템을 구축하는 일입니다.

어떤 산출물을 통해서 이러한 모델을 표현할 것인지를 선택하는 것은 프로젝트 매니저가 결정할 일입니다.

모델 산출물에는 다음과 같은 몇 가지 타입이 있습니다.

① 문서 형태

요구사항 스펙과 유즈케이스 시나리오는 형식적, 비형식적 문서 모두를 필요로 합니다. 제안서나 **SRS(System Requirements Specification)** 같은 문서는 요구사항- 레벨의 산출물입니다.

② UML 다이어그램

UML은 풍부한 다이어그램들로 소프트웨어 시스템에 대한 우리의 생각을 가시화하는 것을 가능하게 합니다.

③ Prototypes

프로토타입이란 목적하는 소프트웨어 시스템을 작은 규모로 만들어 놓은 것을 말합니다.

여기에는 두 가지 주요 유형이 있습니다. 하나는 기술에 관계된 것이고 다른 하나는 사용자 인터페이스에 관계된 것입니다. 기술적 프로토타입은 새로운 기술이나 알려지지 않은 기술, 위험부담이 큰 기술들을 미리 구축해 보는 것이고 사용자 인터페이스 프로토타입은 클라이언트나 **UI** 디자이너가 사용성을 높일 수 있는 형태를 제시하도록 가시적인 표현을 도출합니다.

I. 모델의 목적

모델은 아래와 같은 목적에 쓰입니다.

i. Communication

모델은 프로젝트 참가자들간의 커뮤니케이션이 가능하게 합니다

다.

ii. Problem solving

어떤 모델은 어려운 문제를 해결하는데 사용될 수 있습니다.

그러나 짧은 개발 기간 때문에 소외되는 경향이 있습니다.

iii. Proof-of-concept

프로토타입과 같은 모델은 가능성 있는 개념을 증명하기도 합니다.

이것도 또한 소홀히 취급되는 경향이 있습니다. 그러나 만약 크라이언트가 어떤 프로토타입이 개념을 증명하는 것을 본다면 최종 시스템에 그것을 포함시킬지를 결정할 수 있습니다.

결국 모델을 기반으로 하는 방법론을 제시한다면 처음에 목적했던 시스템을 최종적으로 실현할 수 있는 개발과정을 얻게 될 것입니다.

vi. Design best practices

소프트웨어 솔루션의 디자인을 고려하는 것은 이 시스템을 유동적이면서도 수정이 쉽게 만드는 장점이 있습니다.

이 원리는 다음을 고려합니다.

① Design principles

디자인 원칙은 **flexibility, extensibility, decoupling**을 제공하도록 컴포넌트 간의 상호 관계를 정해 놓은 규칙을 말합니다. 이러한 원칙은 패턴으로 승화됩니다.

심화학습

n Design principles

디자인 원칙은 다음과 같이 세분화됩니다.

u Separation of Concerns

u Dependency Inversion Principle

u Stable Dependency Principle

이것은 [단원 6 솔루션 모델의 구축과 테스트. 모듈 3. 솔루션 모델에 디자인 패턴 적용 1]에서 자세히 다루도록 합니다.

② Software patterns

패턴이란 유사한 상황에 동일하게 적용할 수 있는 솔루션을 말합니다.

패턴은 특정한 상황에서 발생하는 문제들을 다루고 그 해결책을 제시합니다. 따라서 패턴은 개발되는 것이 아니라 경험을 통해 발전시키는 것입니다.

③ Refactoring

리팩토링이란 외적인 기능의 변화는 없이 코드를 개선하는 작업을 말합니다.

이것은 중복 코드의 삭제, 긴 메소드의 단축, 큰 클래스의 압축, 파라미터 리스트를 간략하게 하는 것 등을 포함합니다.

아직 시스템의 내부 구조가 향상되는 것은 시스템의 효율성을 높입니다.

④ Sun Blueprints

Sun Microsystems는 J2EE 개발을 포함한 다양한 소프트웨어 솔루션의 좋은 예를 제공합니다.

[참고하세요]

Sun blueprints는 <http://www.sun.com/blueprints/>에서 얻을 수 있습니다.

3. 몇 가지 객체지향 방법론

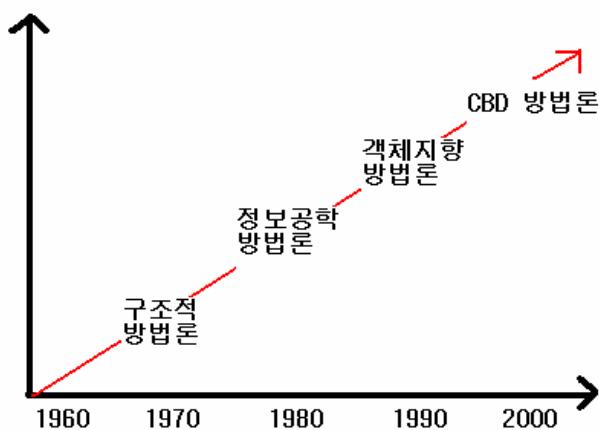
이 단계에서는 주류를 이루고 있는 몇 가지 객체지향 방법론을 알아봅니다.

- ① Waterfall
- ② Unified Software Development Process(USDP or just UP)
- ③ Rational Unified Process(RUP)
- ④ SunTone Architecture Methodology
- ⑤ extreme Programming(XP)

[심화학습]

n 방법론의 진화

구조적 방법론 \Leftarrow 정보공학 방법론 \Leftarrow 객체지향 방법론 \Leftarrow CBD 방법론

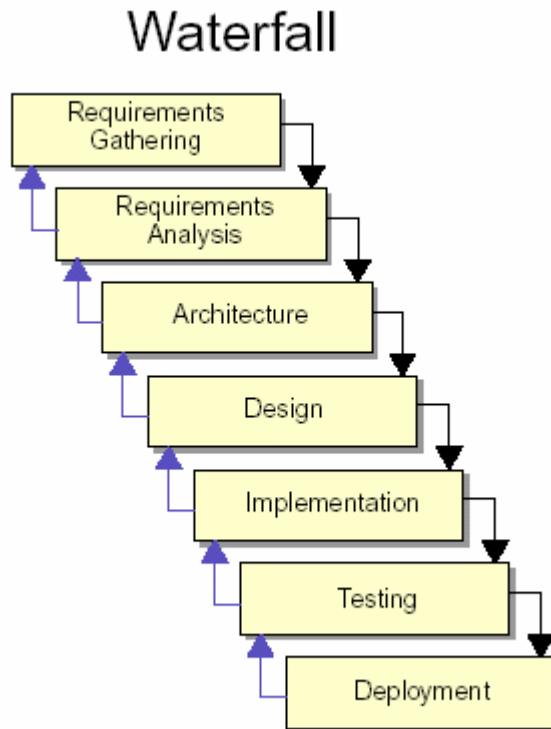


1) Waterfall

일명 폭포수 모델이라 불리는 이 방법론은 오래된 개발 프로세스 중의 하나입니다.

[이미지]

n Waterfall



특징:

- | 모든 워크플로우가 일렬구조를 갖으면서 오직 한 **phase**로 이루어집니다.
요구사항 수집- 요구사항 분석- 아키텍쳐 수립- 디자인- 구현- 테스팅- 배치의
워크플로우가 오직 한번 씩 만 이루어지며 이것이 한 **Phase**를 이룹니다.
- | ‘반복’을 지원하지 않습니다
한번 진행이 완료된 워크플로우는 다시는 재수행하지 않습니다.
- | 요구 사항이 변경되더라도 수용할 수 없습니다.
설사 클라이언트 요구 사항이 처음과 달라졌다 해도 수용할 수가 없습니다.
때문에 클라이언트 요구 사항이 나중에 변경될 수 없음을 확인해 두셔야 합니다.
- | 몇몇 정부에서 행하는 시스템에 이 방법론이 사용되기도 합니다.
- | 고정된 개발 기간이나 비용 때문에 이 방법론이 사용되기도 합니다.

Key Point

n Waterfall

오래된 방법론 중의 하나인 폭포수 모델은 한번 진행된 워크플로우는 다시는 반복되지 않으므로 요구 사항 변경을 수용할 수 없다는 것을 염두에 두어야 합니다.

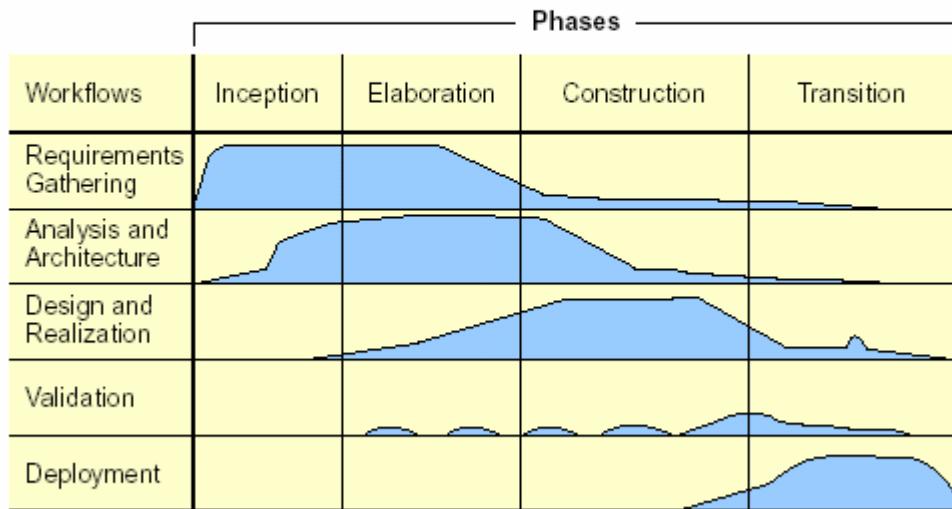
2) Unified Software Development Process(USDP or just UP)

USDP는 Grady Booch, Ivar Jacobson, James Rumbaugh에 의해 창안된 RUP(Rational's methodology)의 전신입니다.
USDP는 UP라고도 합니다.

[이미지]

n USDP

Unified Software Development Process



UP는 다음 4 Phase로 구성되어 있습니다.

I Inception

- 도입 : 소프트웨어의 비전을 생성하는 단계
이 단계는 목적하는 시스템의 비즈니스 유형을 이해하는데 중점을 둡니다.
또한 임시 아키텍쳐가 수립되고 중요한 프로젝트 위험요소가 제시되며 전개 단계의 계획이 자세히 짜여집니다.

I Elaboration

- 전개 : 시스템 아키텍쳐가 수립되고 대부분의 기능이 정의되는 단계
이 단계의 목적은 적절한 아키텍쳐를 완성함으로써 프로젝트 위험부담을 감소시키는데 있습니다. 이 단계의 끝에서는 프로젝트 매니저가 구현과 전이에 대한 충분한 정보를 이미 가지고 있어야 합니다.

I Construction

- 구축 : 소프트웨어를 구축하는 단계
여러 번의 ‘반복’을 통해 유즈케이스가 추가됨으로써 소프트웨어는 점점 확대되어집니다. 이 단계의 끝에서 시스템은 베타 버전을 발표하게 될 것입니다.

| Transition

- 전이 : 프로젝트가 배치되는 단계
이 단계는 시스템이 가동을 위한 준비를 하게 됩니다.
이 단계의 **activity**는 테스팅, 디버깅, 트레이닝, 빌딩을 포함합니다.

3) Rational Unified Process(RUP)

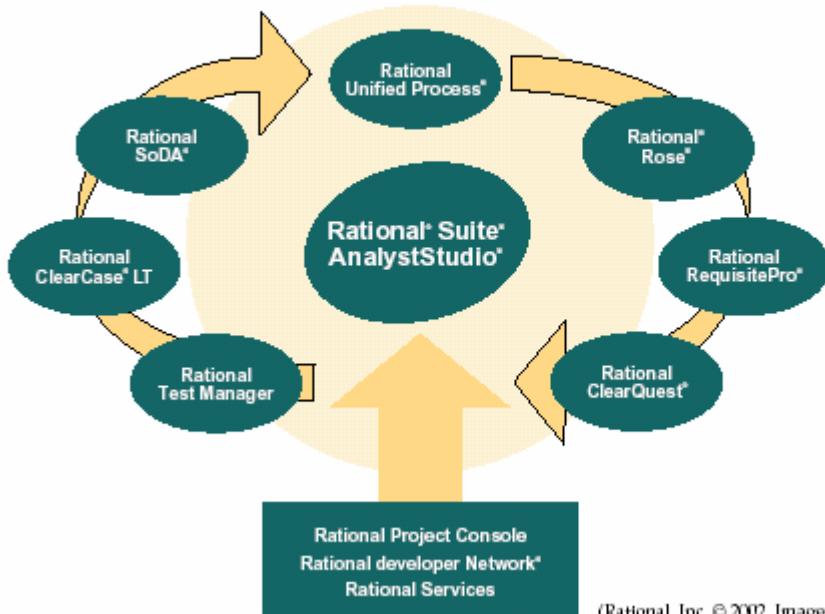
RUP은 UP의 상용 버전입니다. 다시 말해 RUP는 Rational tool set에서 지원하는 UP입니다.

이 툴은 각 Phase에서 진행되는 워크플로우를 지원하고 그 산출물들이 쉽게 도출되도록 돕습니다.

[이미지]

n RUP

Rational Unified Process



(Rational, Inc. © 2002. Image used with permission.)

4) SunTone Architecture Methodology

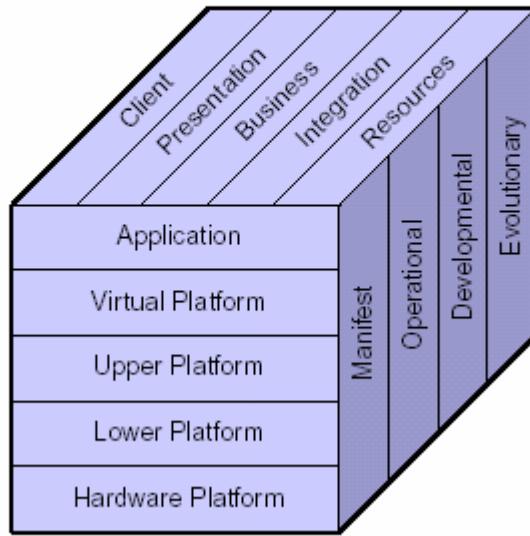
SunTone Architecture 방법론은 UP 방법론과 융화되어 있습니다.

이 방법론은 Sun Microsystems사에서 개발한 것입니다.

[이미지]

n SunTone Architecture Methodology

SunTone Architecture Methodology



특징 :

- | UP의 Phase와 워크플로우를 사용합니다.
이 방법론은 UP나 RUP와 완전히 융화되어 있습니다.
- | 특히 엔터프라이즈 어플리케이션의 아키텍처를 강조합니다.
시스템의 비기능적 요구사항(NFR)을 지원하는데 중점을 둡니다.
이것은 시스템의 질을 높임으로써 위험 요소를 줄이고 안정된 시스템을 구축할 수 있게 합니다.
따라서 엔터프라이즈한 목적을 갖는 소프트웨어 구축 시 좋은 개발 가이드가 될 수 있습니다.
- | 3D 큐브를 통해서 가시화합니다.
3차원 큐브의 가로는 수평적인 Tier 구조를 나타냅니다.
높이는 수직적인 Layer구조를 표현하고 세로는 각 개발 단계를 명시합니다.

심화학습

n SunTone Architecture Methodology

3차원 큐브의 세로(옆면)을 이루는 Phase는 Manifest- operation-developmental- evolutionary를 포함하고 있습니다.

Manifest	Operational	Developmental	Evolutionary
성능	처리량	실체화	Scalability
가용성	관리 용이성	계획성	유지보수성
사용성	보안성		확장성
	편리성		유연성
	Testability		재사용성
	신뢰성		이동성

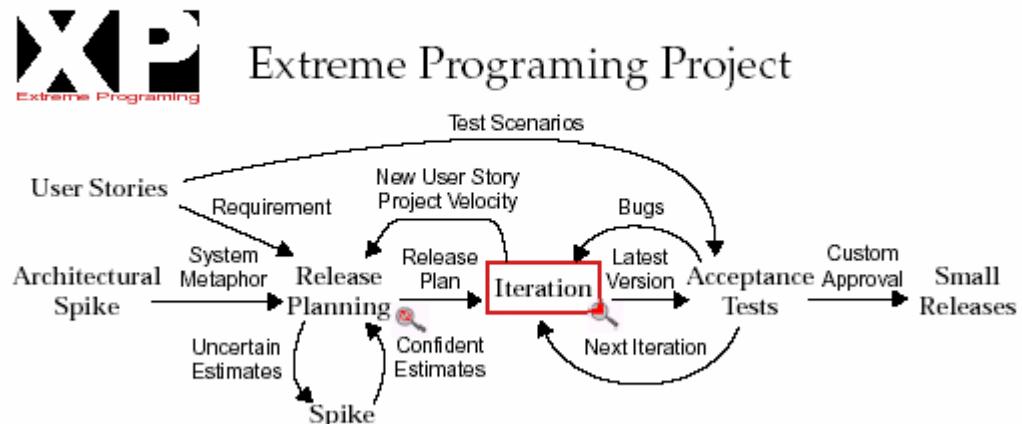
5) extreme Programming(XP)

XP는 최근 개발방법론 중에서 급부상하고 있는 애자일 소프트웨어 개발론의 하나로 단순성, 상호 소통, 피드백, 용기등의 원칙에 기반해서 고객에게 최고의 가치를 가장 빨리 전달하도록 하는 경량 방법론입니다.

[이미지]

▪ extreme Programming(XP)

eXtreme Programming



(J. Donvan Wells © 2002. Image used with permission.)

Copyright 2000 J. Donvan Wells

특징 :

| Pair programming

개발자는 코딩하는 동안에 짹과 함께 작업하게 됩니다. 한 사람이 코딩하면 다른 사람이 그것을 리뷰하고 더 좋은 코드로 리팩토링해줍니다.

| Testing

코딩하기 전에 테스트 프로그램을 먼저 만듭니다.

| Refactoring

pair programming은 리팩토링을 부추깁니다. 리팩토링은 코딩하는 동안에 중심 되는 이슈가 됩니다.

| Simplicity

이 시스템이 가능한 많은 유즈케이스를 최대한 간결하게 지원할 수 있도록 디자인합니다.

[참고하세요]

▪ extreme Programming(XP)

요구사항등의 변화가 자주 발생하거나 개발자가 10명이내의 소규모이고 팀

원들이 같은 공간을 사용하는 프로젝트일 경우에 적용되는 개발 방법론입니다.

4. 프로젝트 상황에 맞는 방법론 선택

각 소프트웨어 개발팀에서 상황에 맞는 방법론을 선택하기 위해서는 많은 사항을 고려 해야만 합니다. 그 중에 기본이 되는 몇 가지만 살펴 본다면 아래와 같습니다.

- | 기업 문화 – 기업 문화가 개발과정 지향적이나 생산물 지향적이냐를 파악해야 합니다
- | 팀 구성원 – 개발 경험이 많은 구성원들이 효율적인 조직으로 분업화 되어 있는지를 봐야합니다.
- | 프로젝트의 크기 – 큰 프로젝트는 좀 더 많은 문서를 필요로 할 것입니다. (프로젝트 참가가간의 의사소통을 위해서)
- | 요구 사항의 안정성 – 얼마나 자주 요구 사항이 변경되는지를 파악해야 합니다.

1) Waterfall을 선택할 때

적합한 상황 :

- | 역할이 구별된 큰 팀일 때
예를 들어 분석팀에서 일정 기간동안 프로젝트에 대한 분석만 완료하고 철수하면 그 다음 팀들이 들어와 설계만 도맡아 한 뒤 다시 철수하는 식의 개발을 할 때 적합한 방법론입니다.
그러므로 전팀에서 행한 워크플로우를 다시는 재수행 할 수 없습니다.
- | **리스크가 낮은 프로젝트일 때**
이것은 폭포수 모델을 선택하는 가장 큰 이유입니다. 프로젝트의 요구 사항이 변하지 않고 리스크가 낮아 한번에 원하는 프로젝트를 성공 시킬 수만 있다면 이것은 좋은 방법론이 될 것입니다.

특징 :

- | 요구사항 변경을 수용하기 어렵습니다.
- | 문서가 큽니다.

2) UP를 선택할 때

적합한 상황 :

- | 기업문화가 절차 지향적일 때
- | 팀멤버가 유동적인 **Job Role**을 갖을 때
- | 큰 스케일의 프로젝트일 때
- | **요구사항 변화를 수용해야 할 때**

UP는 각 **Phase**별로 워크플로우를 반복하므로 초기 요구 사항의 변경에 유

연히 대처 할 수 있습니다.

특징 :

- | 문서가 많습니다.
- | 작은 프로젝트에서는 오히려 역효과가 납니다.

3) RUP를 선택할 때

적합한 상황 :

- | UP와 같은 이유일 때
 - | 회사에서 Rational's Tool set을 사용할 때
- RUP는 캐셔날사의 툴이 필요합니다. 이미 회사내에서 이 툴을 사용하고 있다면 RUP를 따르는 것이 좋습니다.

특징 :

- | UP와 같습니다.
 - | 캐셔날사의 툴을 익혀야 합니다.
 - | 툴이 팀을 고정된 개발 과정으로 묶어 둘 수 있습니다.
- 툴은 팀 프로젝트 성격과 상황에 맞게 최적화 되어야 합니다. 그러나 이것이 어려운 일이라서 자칫 툴이 제공하는 프로세스대로 팀 개발이 어렵게 진행 될 수도 있습니다.

4) SunTone Architecture Methodology를 선택할 때

적합한 상황 :

- | UP와 같은 이유일 때
 - | 엔터프라이즈 어플리케이션 일 때나 아키텍쳐 중심적인 시스템 구축 시에
- SunTone Architecture Methodology** 는 아키텍쳐 중심적인 **n-tier**를 구축함으로써 '**heavy design**' 되어질 수 있습니다. 만약 대형 웹어플리케이션을 구축하고자 한다면 이 방법론이 적당할 것입니다.

특징 :

- | UP와 같습니다.
- | 최근까지 거의 알려져 있지 않습니다.

5) XP를 선택할 때

적합한 상황 :

- | 회사문화가 실험을 허락할 때

XP는 전통 개발 프로세스에 비해서 상당히 파격적인 방법론입니다.
회사내에서 전통적인 개발 프로세스에서 벗어나 새로운 개발 프로세스를 허

락 할 수 있을만큼 개방적이면서도 위험부담을 감수할 수 있어야 합니다.

- | 팀원들의 수가 많지 않고 같은 공간에서 작업할 때
Pair Programming을 하기 위해서는 같은 공간에서 작업 할 수 있는 동료가 있어야 합니다. **XP**는 비록 한 빌딩에 있을지라도 작업 공간이 다르면 실패할 수도 있습니다.
- | 팀내에 개발 경험자가 많을 때
XP는 초보 개발자들이 해내기 어렵습니다. 바로 코딩에 들어가고 바로 **refactoring**에 들어가고 해야 하기 때문에 노련한 개발자들에게나 적합합니다.
- | 요구 사항이 자주 변경 될 때
XP의 장점은 쉽게 변경 할 수 있다는 것입니다.

특징 :

- | 문서가 적습니다.

과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원2 : 요구공학

1 모듈 : 프로젝트 제안서 작성

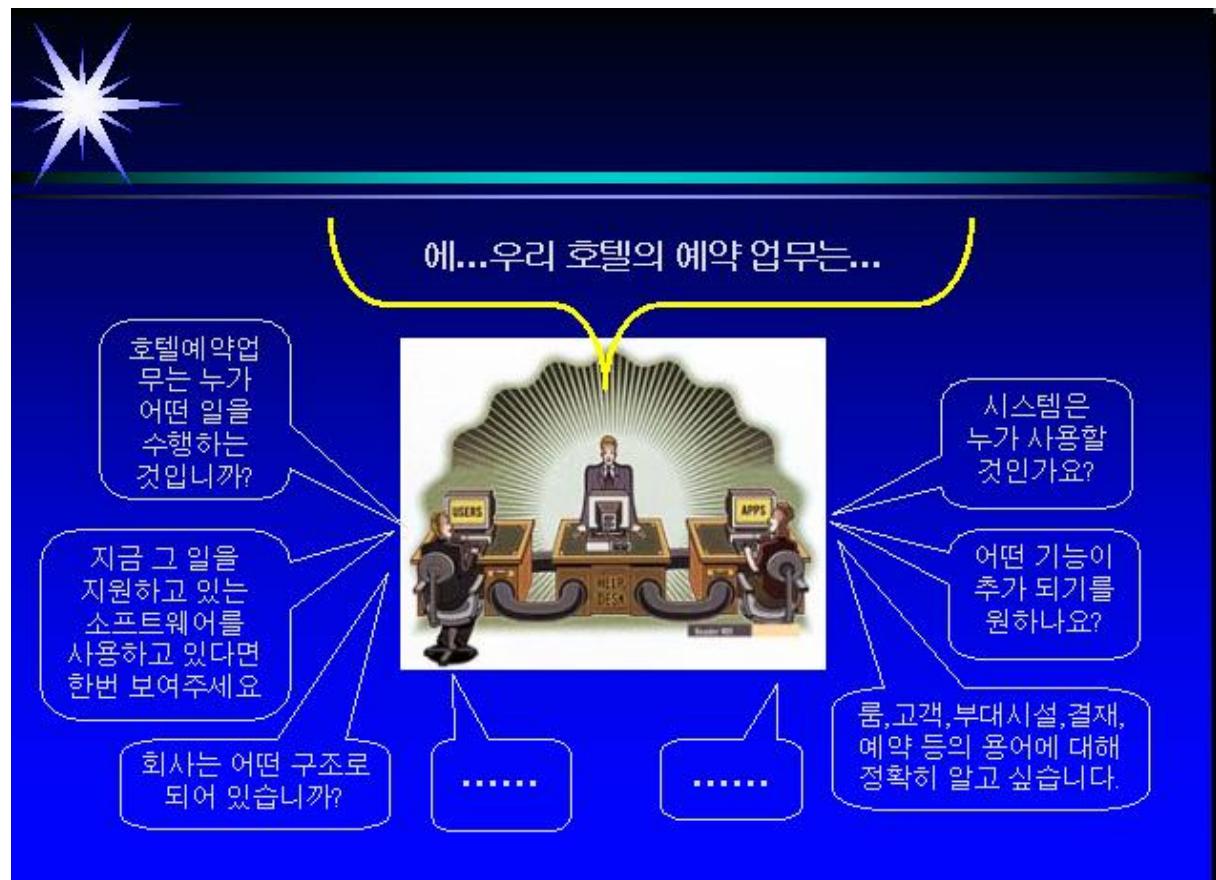
담당강사 : 전은수

■ 생각해봅시다 ■

소프트웨어 프로젝트를 개발하는 첫번째 단계가 바로 제안서를 작성하는 것입니다. 제안서는 프로젝트를 위한 작업 내용을 기록한 것으로, 클라이언트 측의 요구 사항은 인터뷰를 통해 얻게 됩니다. 인터뷰에서 필요한 정보를 얻고, 상대방의 말이 의미하는 바가 무엇인지 확실히 알아야 제안서를 작성할 수 있고 프로젝트를 무리 없이 진행할 수 있습니다. 그렇다면 인터뷰는 어떻게 해야 합니까? 효과적인 인터뷰 기법에 대해 알고 있습니까?

이 부분은 프로젝트 단계에서 가장 중요한 부분으로, 제안서를 작성하는 일은 요구 사항을 수집하는 과정의 첫번째 단계입니다.

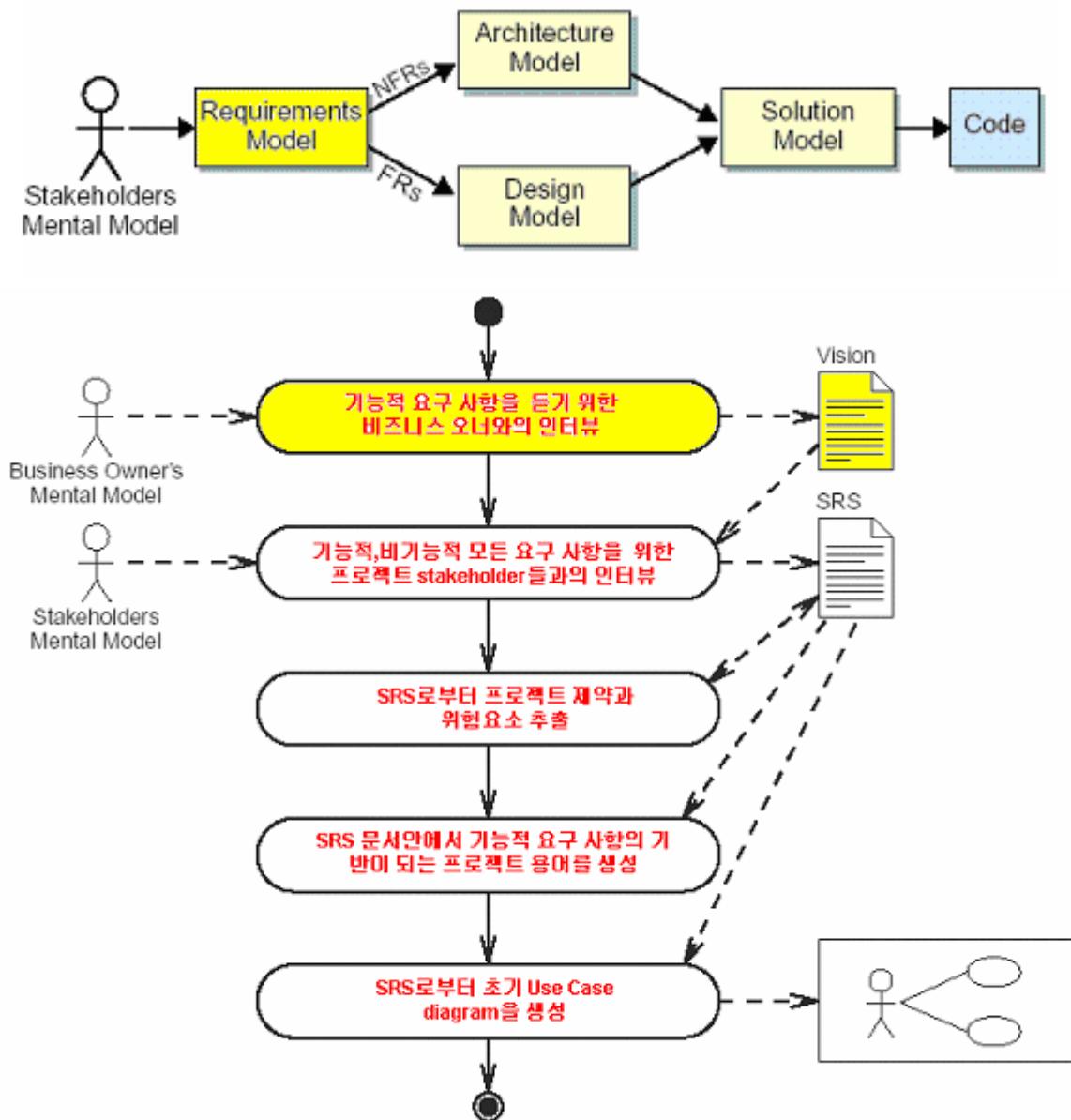
에니메이션



■ 학습하기 ■

[참고하세요]

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. 비즈니스 오너와의 인터뷰 요령

1) 비즈니스 오너와의 인터뷰

(1) 개요

프로젝트 제안서를 작성하기 위해서는 우선 비즈니스 오너와 인터뷰를 합니다. 이 인터뷰를 통해서 고차원적인(**high-level**) 기능적 요구 사항을 정의할 수 있습니다.

이번 과정에서는 호텔 예약 시스템(**Hotel Reservation System : HRS**)의 사례를 예로 들겠습니다. 클라이언트는 **Bay View Properties**이고, 비즈니스 오너는 **Peter**와 **Mary Jane Parker**입니다. 그리고 소프트웨어를 설계하는 것은 **Sypher Dies**의 **ACME 컨설팅** 업체입니다.

제안서 작성을 위한 인터뷰는 기본적으로 다음의 내용을 포함해야 합니다.

- **비즈니스 오너는 프로젝트의 주인입니다.**

비즈니스 오너는 작은 기업체의 사장일 수도 있고, 어느 부서의 책임자일 수도 있습니다. 비즈니스 오너는 한명으로 정하는 경우가 제일 좋습니다. 왜냐하면 비즈니스 오너가 여러 명일 경우, 최종 결정권이 여러 명에게 분산되고, 또한 제안서의 내용을 서로 다르게 해석할 수 있기 때문입니다.

- **기능적 요구 사항은 시스템이 사용자를 위해 어떤 작업을 반드시 수행해야 하는지를 기술한 것입니다.**

참고하세요

- **비즈니스 오너(business owner)의 의미**

비즈니스 오너(**business owner**)는 기업체의 소유주를 의미하는 것이 아니라, 일반적인 업무의 책임자를 의미합니다. 예를 들어, 어느 대학의 과학관련 시스템의 비즈니스 오너는 그 프로젝트의 책임 연구원일 수 있습니다.

2) 요구 사항의 종류

요구 사항은 기능적 요구 사항과 비기능적 요구 사항으로 구분할 수 있습니다.

- **(1) 기능적 요구 사항(Functional requirements : FRs)은 사용자의 관점에서 본 시스템의 행동을 정의합니다.**

- **FRs**은 사용자의 작업을 지원하는 시스템의 기능을 말합니다.

시스템이 사용자를 위해 할 수 있는 일과, 자체적으로 수행할 수 있는 일이 기능적 요구 사항(**FRs**)에 해당합니다.

- 예)

시스템은 고객의 정보 – 이름과 주소- 를 모아야 합니다.

(2) 비기능적 요구 사항(Non-functional requirements : NFRs)은 시스템의 질적 특징을 정의합니다.

- **NFRs**은 사용자의 작업을 시스템이 어떻게 지원하는지를 의미합니다.
NFRs에 해당하는 것으로 시스템의 작업 처리 속도, 동시 처리량, 유지보수의 용이성, 사용의 편리함 등이 있습니다.
- 예)
시스템은 웹 환경에서 **10명**의 동시 사용자를 지원해야 합니다.

보충

n FRs과 NFRs

시스템을 동물에 비유한다면, **FRs**은 동물의 외양, 즉 그 동물은 무슨 행동을 할 수 있고, 어떤 특별한 능력을 갖고 있는지 등에 해당합니다.

그리고 **NFRs**은 그 동물의 내적인 특징, 즉 그 동물이 얼마나 빨리 달릴 수 있는지, 얼마나 무거운 물건을 옮길 수 있는지 등에 해당합니다.

3) 인터뷰 기술

시스템의 **FRs**와 **NFRs**을 알아내는 것은 대부분 시스템 관련 인물들(**stakeholders**)을 인터뷰하는 과정에서 이루어집니다. 인터뷰 기술을 배우는 것은 쉽지 않습니다. 하지만 하나의 인터뷰를 잘 해두면 계속되는 인터뷰 회수를 줄일 수 있습니다.

인터뷰 방법을 설명하는 것은 이 과정에서 벗어나는 내용이지만, 몇 개의 팁을 알려드리겠습니다.

- 비즈니스 오너와 상호 신뢰관계를 구축하십시오.
농담은 피하고 가벼운 대화를 나누는 것이 좋습니다.
- 주의 깊게 들으십시오.
인터뷰에서 가장 중요한 부분입니다. 실제로 중요한 부분을 상대방이 지나가는 말로 할 수도 있습니다.
- 인터뷰의 목적에 맞게 진행해야 합니다.
비즈니스 오너가 질문의 요지를 벗어난 말을 너무 길게 하면, 조심해서 다시 원래 주제로 돌아오게 해야 합니다.
- 주의 깊게 들으십시오.
인터뷰는 고도의 집중력을 요구합니다. 상대방이 지금 말하고 있는 것은 매우 중요합니다. 다른 생각을 해서는 안됩니다.

- 불분명한 구문은 명확히 하고, 확인을 요구하십시오.
여러 가지 뜻으로 해석할 수 있는 부분은 여러분이 이해한 대로 상대방에게 말한 다음 맞는지 확인하는 것이 좋습니다.
- 주의 깊게 들으십시오.
다의적인 구문은 이해하기 힘듭니다. 주의 깊게 들으면, 언제 중요한 정보를 놓쳤는지 알 수 있습니다.
- 세부사항까지 적어놓으십시오
인터뷰한 상대방에 대한 정보를 메모해 두는 것이 좋습니다. 또 각 유즈케이스마다 충분한 여백을 두고 그 유즈케이스와 관련된 내용을 적어두는 것도 좋습니다.
- 주의 깊게 듣는 것이 가장 중요합니다.

참고하세요

n 인터뷰 기술

기억력에 의존하지 마십시오! 인터뷰하는 내용을 녹음하는 것은 좋은 방법입니다. 그러면 적어둔 내용 간의 갭을 채우기 위해 인터뷰의 내용을 되돌아볼 수 있습니다.

4) 인터뷰의 주요 내용

프로젝트 제안서 작성을 위한 인터뷰는 다음과 같은 점에 중점을 두고 있습니다.

- 프로젝트에서 다뤄야 하는 업무 내역
- 프로젝트에 대한 기능적 요구 사항
- 위험 요소(**Risks**)
- 제약 사항(**Constraints**)
- 관련 인물(**Stakeholders**)

(1) 업무를 알기 위한 질문

소프트웨어의 필요성에 대한 논의는 **enduring business themes(EBTs)**를 이해하기 위해 중요합니다. **EBTs**는 그 회사가 계속 하게 될 일을 의미하는 것으로, **Bay View**의 경우는 고객을 위한 숙박 시설 관리일 것입니다.

- 현재 하고 있는 일은 무엇입니까?
이 질문을 통해 그 회사가 무슨 일을 하는지를 알 수 있습니다.

보충

n 업무를 알기 위한 질문

예)

- .. 이 회사는 무슨 일을 합니까? 무엇을 만듭니까? 무엇을 팝니까?
- .. 회사는 어떤 구조로 되어 있습니까?
- .. 조직 구성도(또는 **relevant business unit**)를 볼 수 있을까요?

참고하세요

n 업무를 알기 위한 질문

인터뷰를 하기 전에 고객사에 대한 기본적인 이해를 하고 있는 것이 중요합니다. 고객사의 웹사이트나 판촉물을 통해서 정보를 얻을 수 있습니다.

– 새 소프트웨어가 그 일을 어떻게 지원해주길 바라십니까?

이 질문은 인터뷰의 목적입니다. 이 질문을 통해 기능적 요구 사항에 대한 논의를 하게 됩니다.

– 그 일이 어떻게 변경됩니까?

이 질문을 통해 두 가지를 알 수 있습니다. 우선, 변경되지 않는 작업을 알면 **EBTs**를 결정할 수 있습니다. 또한 앞으로 변경 가능성이 있는 부분을 알 수 있기 때문에, 시스템의 일부분을 좀더 확장 가능하게 만들 수 있습니다.

보충

n 업무를 알기 위한 질문

예)

- .. 그 일을 확장할 계획입니까?

Key Point

n 업무를 알기 위한 질문

소프트웨어의 필요성에 대한 논의는 **enduring business themes(EBTs)**를 이해하기 위해 중요합니다. **EBTs**는 그 회사가 계속 하게 될 일을 의미하는 것입니다.

(2) 기능적 요구 사항에 대한 질문

비즈니스 오너가 소프트웨어가 해야 할 일이 무엇인지 설명하면, 여러분은 고객에게 유즈케이스가 무엇인지 설명하고, 가장 중요한 유즈케이스를 나열하게 합니다. **중요한 점은**

유즈케이스란 사용자가 시스템을 어떻게 사용할지를 기술한 것이라는 점입니다. 사용자를 위해 시스템이 어떤 기능을 하는지에 기억해야 합니다.

- 비즈니스 오너에게 시스템의 유즈케이스 중 중요한 것 **10가지를** 나열하게 합니다.
- **비즈니스 오너에게 각각의 유즈케이스의 정확한 의미를 확인합니다.**
 - .. 각 유즈케이스를 바르게 이해했는지 확인합니다.
 - .. 비즈니스 오너에게 각 **유즈케이스의 우선순위를 정하게 합니다.**
 - .. 유즈케이스를 반복해서 말하고, 중요한 유즈케이스를 잊지 않았는지 확인합니다.

보충

n 기능적 요구 사항에 대한 질문

각 유즈케이스에 대해 제대로 이해하고 있는지 확인하되, 세부적인 사항에 대해서는 이 때 질문 할 필요가 없습니다.

유즈케이스의 우선순위를 정할 때는 그 중요도에 따라 **3단계** 정도로 나누어, **Essential, High-level, Follow-on** 등으로 구분할 수 있습니다.

이 인터뷰의 목적은 **high-level** 유즈케이스의 **80%** 정도를 발견하는 것으로, 잊은 항목을 반복하는 것은 한번으로 충분합니다.

Key Point

n 기능적 요구 사항에 대한 질문

비즈니스 오너가 소프트웨어가 해야 할 일이 무엇인지 설명하면, 여러분은 고객에게 유즈케이스가 무엇인지 설명하고, 가장 중요한 유즈케이스를 나열하게 합니다.

(3) 위험 요소를 파악하기 위한 질문

모든 프로젝트는 위험 요소를 내포하고 있습니다. 위험 요소란 프로젝트를 실패로 이끄는 어떤 상황이나 요소를 말합니다.

다섯 개의 영역에 따라 위험 요소를 파악하는 질문은 다음과 같습니다.

- 비슷한 일을 하는 다른 조직이 있습니까?
한 기업 내에서 경쟁 관계의 프로젝트가 있는 경우, **정책적 위험 요소(political risk)**라고 합니다.
- 프로젝트에 새로운 기술을 사용할 계획입니까?
개발팀이 익숙하지 않은 기술 사용을 요구할 경우, **기술적 위험 요소(technology risk)**라고 합니다.
- 개발에 참여할 인원이 있습니까? 혹은 외주(**outsourcing**)에 맡길 계획입니까?

프로젝트를 완성할 충분한 자원을 갖고 있습니까? 이것을 **자원의 위험 요소 (resource risk)**라고 합니다.

- 팀원들이 필요한 기술을 습득하고 있습니까?

Java를 사용해서 개발해야 하는 프로젝트인데, 팀원들이 **Visual Basic**만 사용할 수 있다면, **기능적 위험 요소(skill risk)**에 해당합니다.

- 시스템에 영향을 주는 작업 내용에 변경 사항이 있습니까?

프로젝트를 개발하는 동안 작업 내용이 변경된다면, 요구 사항도 변경해야 됩니다. 이것을 **요구 사항의 위험 요소(requirements risk)**라고 합니다.

(4) 제약 사항을 파악하기 위한 질문

다음과 같은 질문을 통해 제약 사항을 파악할 수 있습니다.

- 이 **프로젝트를 특정 플랫폼 환경에서 개발할 것입니까?**

이 질문을 통해 비즈니스 오너가 시스템을 특정 하드웨어나 운영체제 또는 특정 제품과 연계해 사용할 것인지 알 수 있습니다. 아직 구체적인 실행 환경이 정해지지 않은 경우에는 **architect**가 추천할 수 있습니다.

- 이 **프로젝트에 특정 기술이 필요합니까?**

웹 어플리케이션입니까? J2EE 플랫폼이 필요하다거나, **SOAP(Simplified Object Access Protocol)**이 필요합니까? 데이터 저장소는 어떤 것을 사용할 계획입니까?

- **프로젝트 마감 기일이 정해졌습니까?**

프로젝트의 긴박함을 파악하기 좋은 질문입니다. 시간적인 제약 사항과 위험 요소를 파악하기 위해 이 질문을 해야 합니다.

- **시스템이 다른 외부의 시스템과 연계되어야 합니까?**

외부 시스템과의 통합은 프로젝트의 중요한 제약 사항입니다.

예를 들어, 호텔 고객이 주문한 영화를 보여주는 시스템은, 고객이 호텔에 들어왔을 때 예약 담당자가 작동시켜야 합니다.

- **시스템 실행 측면에서의 제약 사항은 무엇입니까?**

시스템의 실행 측면이라는 것은 소프트웨어를 관리할 수 있는 툴을 말합니다. 예를 들어, 웹 서버나 원격 서버를 실행시키는 툴, 시스템에 사용자를 추가하는 툴, 데이터베이스에 데이터를 추가하거나 수정하는 툴 등이 이에 해당합니다.

(5) Stakeholder를 알아내기 위한 질문

비즈니스 오너에게 시스템과 관련된 주요 인물들의 이름을 물어봐야 합니다. **Stakeholder**란 프로젝트와 연관이 있는 사람 또는 조직을 의미합니다. 개발팀뿐만 아니라 클라이언트 측에도 **stakeholder**가 있습니다.

다음과 같은 질문들은 **stakeholders**를 구분하는데 도움이 됩니다.

– 시스템의 기능적 요구 사항에 대한 결정권이 누구에게 있습니까?

이 질문은 어느 **FR**을 이번 시스템에 구현하고, 어느 **FR**을 차기 버전에 구현할지에 대한 결정권을 알아내기 위해 중요합니다.

– 누가 시스템을 사용할 것입니까?

시스템의 작업 내용을 나열하고, 그 일과 직접 관련된 사람들을 알아야 합니다. 예를 들어, 호텔 예약 시스템을 사용할 사람들은 예약 담당자, 접수 담당직원, 매니저, 오너 등입니다.

– 시스템의 사용자를 관리할 사람이 누구입니까?

여러 사용자가 사용하는 시스템의 경우, 회사의 절차와 정책을 잘 알고 있는 관리자를 알아야 합니다.

– 시스템 관리자가 누구입니까?

클라이언트 측에 **COO(Chief Operation Officer)**, 시스템 관리자, 네트워크 관리자가 있습니까? 누구에게 시스템 관리의 책임이 있습니까? 누가 시스템에 사용자를 추가할 수 있습니까?

– 프로젝트 개발은 누가 관리합니까?

클라이언트 측에 **CIO(Chief Information Officer)**, **PM**, **architect** 등이 있습니까? 누가 책임자입니까?

2. 인터뷰 분석

1) 개요

제안서 작성을 위한 인터뷰를 마친 후에는, 다음의 사항을 파악하기 위해 인터뷰를 분석합니다.

- 기능적 요구사항
- 비기능적 요구사항
- 위험 요소
- 제약 사항

이 절에서는 **NFR**과 위험 요소를 파악하는 방법을 살펴보겠습니다.

2) 비기능적 요구사항 분석

간혹 비즈니스 오너는 시스템이 유지해야 할 서비스의 질에 대한 요구를 하기도 합니다. 이런 것들을 **NFR(Non-functional requirements : 비기능적 요구 사항)** 이라고 하며, 반드시 기록해 두어야 합니다.

비기능적 요구 사항을 파악하는 것이 제안서의 주요 목적은 아닙니다. 하지만 클라이언트 측에서 언급했다면 기록해두는 것이 좋습니다.

인터뷰의 내용 중 비기능적 요구사항을 파악하는 방법 몇 가지가 있습니다.

- **부사구 표현은 NFR일 수 있습니다.** 소프트웨어 시스템에 있어서 부사구 표현은 시스템이 어느 정도 실행되어야 하는지의 질을 표현하기도 합니다.
예) “응답시간이 아주 빨라야 합니다.”
“데이터베이스에 매우 많은 고객 정보가 있습니다.”
“시스템은 **100명** 이상의 사용자를 지원해야 합니다.”
- **특정 기술을 언급한 것도 제약 사항이나 NFR이 될 수 있습니다.**
예) “고객들이 온라인으로 시스템을 사용하길 바랍니다.”
“이와 비슷한 프로젝트에서 **ORACLE**을 사용했습니다.”

3) 위험 요소 분석

(1) 개요

위험 요소는 프로젝트 실패의 주 원인입니다. 성공한 프로젝트는 위험 요소를 초기에 분석해서 대책을 세웁니다. 제안서 작성시에는 위험 요소를 분석하는 것만으로 충분합니다. **SRS document**에서 위험 요소에 대한 대책을 간구합니다.

다음은 위험요소의 다섯 가지 종류입니다.

- 정책적 위험 요소(**Political risk**)
- 기술적 위험 요소(**Technical risk**)
- 자원적 위험 요소(**Resource risk**)
- 기능적 위험 요소(**Skill risk**)
- 요구 사항의 위험 요소(**Requirement risk**)

(2) 정책적 위험 요소

정책적 위험 요소란 프로젝트가 내부 또는 외부의 다른 프로젝트와 경쟁하고 있는 경우나 프로젝트가 법률에 위배되는 경우를 말합니다. 이것은 비즈니스 오너와 논하기에 매우 민감한 부분입니다. 가장 좋은 방법은 비즈니스 오너가 말하는 것에서 경고를 잘 듣는 것입니다.

다음과 같은 경우 주의해야 합니다.

- 경쟁 프로젝트나 경쟁자가 있는 경우

한 기업 내에 같은 문제를 해결하기 위한 조직이 여러 개 있는 경우가 있습니다. 이런 경우 대개는 공개하지 않습니다.

다른 경우는 경쟁회사에서 비슷한 제품을 만드는 경우인데, 경쟁사의 제품이 더 좋거나 시장의 우위를 점했다면 클라이언트 측에서는 프로젝트를 종료해야 할 것입니다.

- **PM**의 상사가 자금, 장비, 인력 등을 사전 경고 없이 취소한 경우

자금 상황에 따라 프로젝트가 취소될 수 있습니다. 만일 이 조직 내에 다른 프로젝트가 취소되었다면 이 점을 확인해 봄야 합니다.

- 개발 팀 내부 또는 관리 체계에서 대인관계로 인한 문제가 발생한 경우

규모가 큰 회사의 경우 조직간에 서로 협동적이지 않을 수 있습니다. 또는 팀 내부에서 디자인이나 구조에 대한 의견차이로 다투는 사람이 있습니까? 이런 문제가 어떻게 해결되었습니까?

- 정부의 법률에 위배 된 경우

국내에서 제작되어 해외에 유통되고 암호화 기술을 사용할 경우, 수출과 관련된 부분을 심사숙고 해야 합니다.

(3) 기술적 위험 요소

아직 증명되지 않은 최신 기술을 사용하는 프로젝트는 기술적 위험 요소를 갖고 있습니다.

간혹 소프트웨어 개발 방법론이 기술적 위험 요소가 되기도 합니다.

아래와 같은 경우 주의해야 합니다.

- 비즈니스 오너가 상당히 많은 기술 용어를 사용하고 그 뜻을 정확이 모르는 경우

이런 경우는 초기에 수습되어야 할 위험한 상황입니다. 비즈니스 오너가 특정 기술에 대한 얘기를 접고 시스템이 반드시 해야 할 일에 대한 논의를 할 수 있게 해야 합니다.

- 비즈니스 오너가 프로젝트가 특정 기술적 방향으로 진행되어야 한다고 주장할 때

만일 비즈니스 오너가 주장한다면, 특정 기술에 대해 적어두고, 다음 인터뷰에서 위험 요소에 대하여 논의할 수 있습니다.

- 비즈니스 오너가 그 회사 업무와 관련된 최신 기술을 사용하길 원할 경우

이 경우도 위험합니다. 비즈니스 오너는 특정 기술에 매우 박식하고 적절한 신기술을 사용하도록 권할지도 모릅니다. 하지만 개발팀이 사용해 보지 않은 기술이라면 위험

요소입니다.

- 새로 개발한 시스템이 기존 시스템과 통합되어야 할 경우

기존 시스템과의 통합은 거의 대부분 위험한 작업입니다. 종종 기존의 설계자나 개발 팀이 회사에 남아 있지 않기 때문입니다. 이 경우는 기존의 시스템을 이해하고, 새 시스템과의 작업에서 발생할 수 있는 문제에 대해 조사해야 합니다.

(4) 자원의 위험 요소

자원의 위험 요소는 프로젝트를 성공적으로 마치기 위해 필요한 자원(인력, 장비, 자금)이 부족한 경우에 발생합니다.

다음과 같은 경우 주의해야 합니다.

- 비즈니스 오너가 프로젝트의 예산이 부족하다는 말을 한 경우

시간은 비용과 같기 때문에, 예산 부족은 일정 문제처럼 해결하기 어렵습니다. 클라이언트 측에 실제 작업 기간과 비용에 대한 정보를 요구해야 합니다. 그리고 비즈니스 오너가 프로젝트를 계속 추진할 지의 여부를 알아야 합니다. 때로는 문제가 생기면 협상을 통해서 첫번째 배포판과 차기 버전에 포함할 내용을 결정합니다.

비즈니스 오너에게 유즈케이스의 우선순위를 두도록 한 이유가 여기 있습니다.

- IT 인력에 무리가 있는 경우

대부분의 경우 클라이언트 사의 개발자들은 다른 프로젝트를 진행하고 있습니다. 따라서 클라이언트 측 개발 인원을 요청할 경우 문제가 될 수 있습니다. 클라이언트 측에서 온 개발자들은 컨설팅 회사의 기술을 클라이언트 사로 전수하게 됩니다.

- 프로젝트가 **calendar-driven**(정해진 일정에 맞추는 방식)으로 진행 될 때

첫번째 문제인 예산 부족과 같습니다.

(5) 기능적 위험 요소

기능적 위험 요소는 개발팀이 필요한 지식이나 경험이 없는 경우에 발생할 수 있습니다.

다음과 같은 경우 주의해야 합니다.

- 프로젝트 개발에 특정 기술을 사용해야 하는데, 개발 팀의 경험이 없을 때

예) 프로젝트에서 웹 어플리케이션을 작성해야 하는데 팀원 중 누구도 경험이 없는 경우

- 개발팀이 모르는 언어로 개발해야 할 때

구조적 언어에서 객체 지향 언어로 이동한 것과 같은, 언어 패러다임에 변화가 있을

때 특히 문제가 됩니다.

(6) 요구 사항의 위험 요소

요구 사항의 위험 요소는 유즈케이스나 **FR**을 정확히 알지 못할 때 발생할 수 있습니다.

다음과 같은 경우 주의가 필요합니다.

- 비즈니스 오너가 “봐야 알겠다” 같은 말을 한 경우

비즈니스 오너가 유즈케이스의 기능에 대해 말하는 게 어려울 수 있습니다. 이런 경우, 비즈니스 오너에게 그 일이 어떻게 이루어지는지 다이어그램을 그리거나 직접 보여주도록 하는 게 좋습니다.

- 비즈니스 오너가 유즈케이스에 대한 시나리오를 제공하지 못할 경우

어떤 유즈케이스는 도메인 전문가와 인터뷰를 해야 할 경우가 있습니다. 제안서 작성 단계에서는 이 부분을 위험 요소로 구분하고 전문가와 다시 평가해도 됩니다.

- 비즈니스 오너가 유즈케이스의 유무를 모르는 경우

이 경우는 위험한 상황입니다. 회사가 구조조정 중이거나 다른 회사와 통합하는 과정에서 발생할 수 있습니다. 이럴 때는 비즈니스 오너가 어떻게 회사가 변하게 될지를 기록해두고, 다른 **stakeholder**와 **SRS** 인터뷰에서 얘기하도록 합니다.

Key Point

n 요구사항의 위험 요소

요구 사항의 위험 요소는 유즈케이스나 **FR**을 정확히 알지 못할 때 발생할 수 있습니다.

3. 제안서 작성

1) 개요

제안서는 시스템에 대한 비즈니스 오너의 제안과 위험 요소, 제약 사항 등을 기록한 것입니다.

제안서가 작성되는 데는 대개 1~2주 정도가 소요되며, 제안서에 따라 프로젝트의 규모가 결정됩니다.

다음은 제안서에 들어가는 다섯 가지 소제목입니다.

- **Introduction (problem statement 포함)**
- **Business opportunity**
- **Proposed solution(FRs과 NFRs 포함)**

- Risks
- Constraints

제안서는 가능한 간결하고 명확한 게 좋습니다.

2) Problem statement 작성

Problem statement는 작업 내용을 간결하게 요약한 것입니다. 모든 **FR**이나 유즈케이스를 포함할 필요는 없지만, 최소한 특징적인 것 한가지는 포함해야 합니다.

예)

호텔 예약 시스템은 침대와 아침식사(**B&B : bed and breakfast**), 비즈니스 관련 시설, 그 외에 여러 개의 숙박시설을 관리해야 합니다. 또한 고객들에게 각종 시설과 방을 보여주는 화면, 과거와 현재의 예약 상황을 보여주는 화면과 새로 예약을 하는 화면 등을 지원하는 웹 어플리케이션 시스템에 포함되어야 합니다.

3) Business opportunity 작성

이 절은 비즈니스 오너의 회사에 대한 비전과, 소프트웨어 시스템이 어떻게 지원할지를 기록하는 부분입니다.

예)

Bay View B&B는 Peter와 Mary Jane Parker가 소유한 가족 소유의 회사입니다. 1995년 그들은 Sonoma에 있는 다른 B&B를 인수했습니다. … 2001년 두 사람이 휴가차 Sierra Madre 리조트를 방문해서 주인과 얘기를 나누고 그가 은퇴하려는 것을 알았습니다. … 이것이 그들의 사업을 확장하기 위해 찾던 기회였습니다. 그들은 현재도 이 계약을 마무리하기 위해 일하고 있습니다. 호텔 예약 시스템은 이런 시설들을 통합하기 위해 제안한 것입니다.

4) Proposed Solution 작성

이 절은 비즈니스 오너에게 확인한 **high-level** 요구 사항(**FR**과 **NFR**)을 기록하는 부분입니다.

- **FR**은 사용자가 시스템을 어떻게 사용하는지를 간단한 문장으로 표현합니다.

예)

시스템은 단일화된 웹 환경에서 다양한 언어로 방과 여러 부대 시설의 사진을 보여줍니다.

접수 담당 직원은 고객의 체크인과 체크아웃을 할 수 있습니다.

- **FR**을 우선순위에 따라 분류합니다.

제안서에는 **FR**을 세가지로 분류합니다 : **Essential, High-value, follow-on**

- 비즈니스 오너와의 인터뷰를 통해서 알게 된 **NFR**을 간단하게 정리합니다.

예)

온라인을 통한 예약은 시작부터 완료까지 **10분**을 초과해서는 안됩니다.

예상 처리량은 분당 **10개**의 트랙잭션 이하 입니다.

(1) 파악된 위험 요소 정리

이 절에는 **비즈니스 오너와의 인터뷰**를 통해 알게 된 위험 요소를 기록합니다.

예)

이 프로젝트의 가장 큰 위험 요소는 기존의 데이터 저장소에 있는 데이터를 새로운 데이터 저장소에 맞춰 컨버전하기 위한 방법을 정하는 것입니다. 또 다른 위험 요소는 내부 개발팀을 사용에서 개발하는 경우와 외주를 맡길 경우의 손익 분기점입니다.

(2) 파악된 제약 사항 정리

이 절에는 **비즈니스 오너와의 인터뷰**를 통해 알게 된 제약 사항을 기록합니다.

예)

리조트의 부대 시설 인수비용에 따라 **Parker** 부부는 데이터베이스 서버나 웹 서버를 사지 못 할 수도 있습니다. **Open source** 툴을 사용해서 작업해야 합니다.

과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원2 : 요구공학

2 모듈 : 시스템 요구사항 수집

담당강사 : 전은수

■ 생각해봅시다 ■

제안서는 시스템의 기능적 요구사항의 일부에 지나지 않습니다. **Stakeholder**로부터 요구 사항을 수집하는 것은 문제 영역을 인식하는데 있어 중요한 부분입니다. 어떤 방법으로 요구 사항을 수집하고, 수집된 정보를 이용해서 사용자가 원하는 시스템을 구현할 수 있을까요? 수집된 정보는 어떻게 정리 해 두는 게 좋겠습니까?

애니메이션

비즈니스 분석가(**Business analyst**)가 다양하고 정확한 요구 사항을 수집하여 자세한 문서를 만들기 위해 고민하고 있습니다.

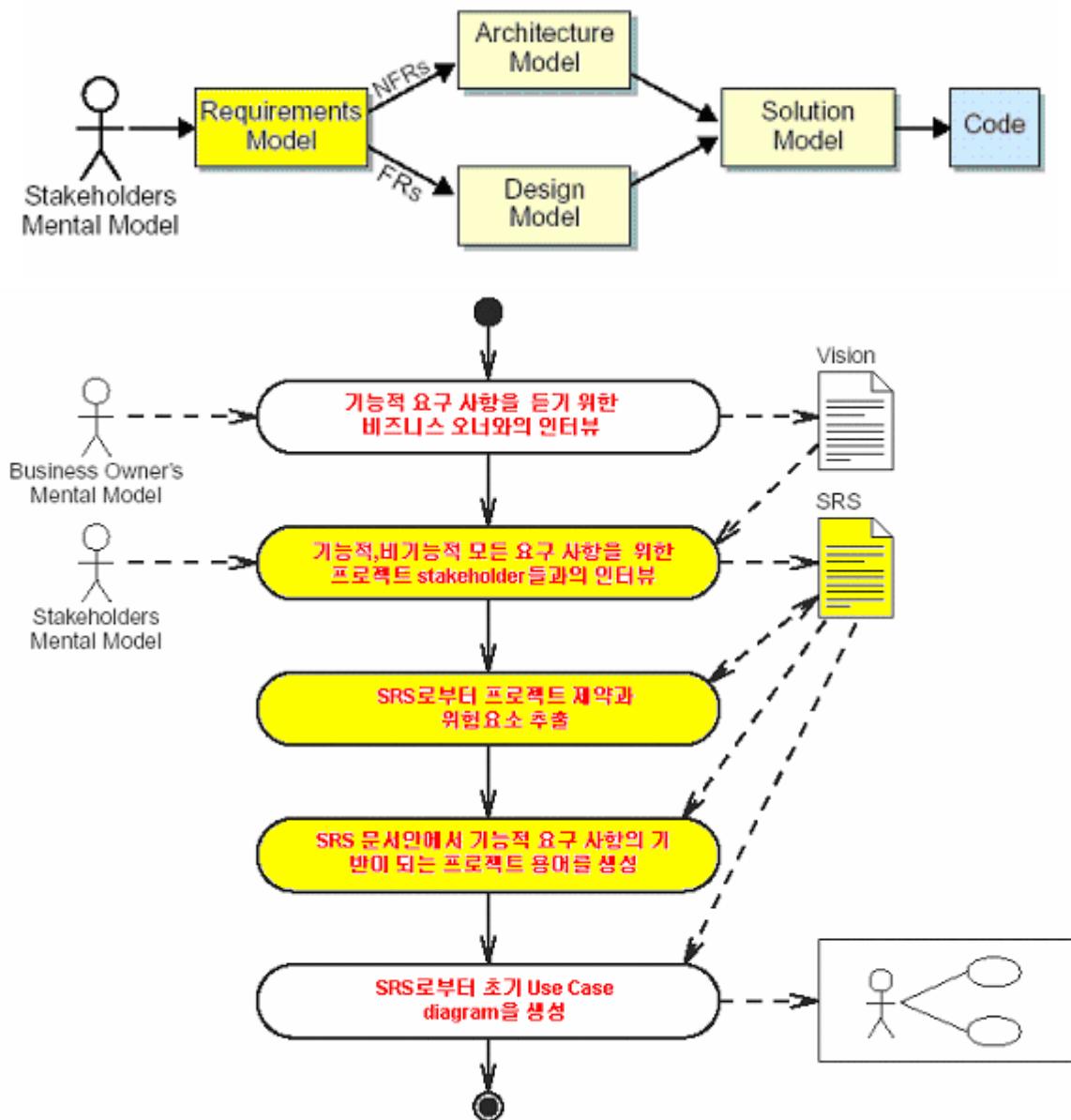
비즈니스 분석가 : “이 프로젝트 참가자 중에 누구를 만나 어떤 정보를 얻어야 할까? 오너? 관리자? 사용자? 고객?... 이들을 만나면 각각 무엇을 물어봐야 하지? 또 그것은 어떻게 정리를 해 두는 것이 효율적일까?”



■ 학습하기 ■

참고하세요

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. 요구 사항 수집을 위한 계획 수립

1) 요구 사항의 출처 분석

요구 사항 수집을 위한 계획 과정은 요구 사항의 출처 분석과 **stakeholder**와의 인터뷰 분석으로 이루어져 있습니다.

요구 사항의 출처는 다양합니다. 다음은 주요 출처입니다.

- Stakeholder들과의 인터뷰

클라이언트측의 **stakeholder**(참가자)들은 시스템이 무엇을 해야 하는지 알고 있습니다. **Stakeholder**들과의 인터뷰는 요구 사항을 구체화하는데 가장 중요한 일입니다.

- 사용자들의 업무 관찰

간혹 사용자들은 자신의 업무를 표현하지 못할 수 있습니다. 때로는 사용자들이 일하는 모습을 지켜보는 것이 유용할 때도 있습니다. 사용자들의 업무를 관찰한 후에 적어두고 질문하도록 합니다.

- 정책과 과정의 분석 및 문서

시스템에서 반드시 지원해야 하는 기능에 대한 정보를 얻는 또 다른 방법은 클라이언트 측의 내부 문서입니다. 이 문서들을 얻을 수 있는지 요청합니다.

- 기존 시스템의 분석

제안서의 시스템이 기존의 시스템을 대체한다면, 사용자들이 기존의 시스템과 어떻게 상호 작용을 할지 그 시스템의 경계를 분석해야 합니다.

그리고 새로운 시스템이 기존 시스템과 통합되어야 한다면 기존시스템의 공용 인터페이스 (**public interface**)를 분석해야 합니다. 공용 인터페이스가 없을 경우에는 시스템 경계를 분석해서 두 시스템을 연결하기 위한 어댑터를 사용할 수 있습니다.

- 시스템의 입출력 분석 및 문서화

시스템에 입력될 데이터 파일과 시스템에서 생성한 데이터 파일이나 보고서는 분석하고 문서화해야 합니다.

2) Stakeholders 분석

Stakeholder 명단은 비즈니스 오너에게서 얻게 됩니다. 하지만, **stakeholder**들과의 인터뷰를 통해서 명단에 인원을 추가할 수도 있습니다. **Stakeholder**는 다음과 같은 사람들을 말합니다.

- 시스템의 주요 사용자

시스템이 지원하게 될 기능에 따라 **job role**을 구분하고, **stakeholder**들과의 인터뷰를 통해 다른 사용자들과 그들의 **job role**을 파악해야 합니다.

- 시스템 운영자, 시스템 관리자, 네트워크 관리자

이 **stakeholder**들은 시스템 개발의 제약 사항과 **NFR**에 대해 많이 얘기할 것입니다. 기존의 하드웨어와 소프트웨어 관리 과정의 세부 사항에 대해 알 수 있습니다.

- 주요 사용자와 운영자들의 총관리자

관리자들은 업무상 정책과 절차를 분석하는데 도움을 줄 수 있습니다. 이 사람들에게 내부 문서를 요청할 수도 있고, 주요 사용자들의 업무에 대한 정보도 얻을 수 있습니다.

- 도메인 전문가

이 사람들은 관련 업무 분야의 전문가입니다.

- 마케팅 관리자

제안된 시스템이 특화된 애플리케이션일 경우, 마케팅 관리자는 예상되는 외부 사용자에 대한 정보를 제공할 수 있습니다.

보충

□ Table :: Job role에 따른 Bay View Stakeholders

Job Role	주요 Stakeholder	2차 Stake holders
오너	Mary Jane Parker	Mary Jane Parker Peter Parker
관리자	Frodric Bagend	Samuel Gamgee(Santa Cruz 지점) Luz Hammarstrom(Sonoma 지점) Frodric Bagend(Sierra Madre 지점)
접수 담당 직원 (subsumes 예약 담당자)	Medoca Sansumi	Medoca Sansumi(Santa Cruz 지점) David Hammarstrom(Sonoma 지점) Judith Brown(Sierra Madre 지점)
이벤트 coordinator	Harold Harkening	Harold Harkening
B&B 고객	Peter Parker(예를 들어)	Peter Parker

2. Stakeholder들과의 인터뷰 요령

1) 개요

Stakeholder들과의 인터뷰는 비즈니스 오너와의 인터뷰와 크게 다르지 않습니다. 주요 차이점은 세부적인 사항과 **NFR**에 대한 추가적인 내용입니다.

다음 항목에 중점을 두어 **FR**과 관련된 질문 리스트를 작성합니다.

- 유즈케이스에 대한 비즈니스 오너의 생각 확인
- 다음 단계의 세부 항목 파악

– 유즈케이스 시나리오 제공

각 **stakeholder**에 대해 **NFR**과 관련된 질문을 준비합니다.

이제부터 이 부분에 대해 살펴보겠습니다.

2) FR 세부 사항 파악을 위한 질문

사용자들의 직무에 따라, 다음과 같은 질문을 할 수 있습니다.

– 그 일에 “xyz” 유즈케이스가 필요합니까?

이 질문으로 이 유즈케이스가 필요한지, 해당 **actor**가 유즈케이스를 수행하는지를 알 수 있습니다.

– “xyz”을 수행하기 위한 단계를 설명해보십시오.

이 질문으로 그 유즈케이스의 수행방법에 대한 세부 사항을 알 수 있습니다. 추가적으로 할 수 있는 질문은 다음과 같습니다.

- .. 각 단계에서 수집하는 데이터는 무엇입니까?
- .. 반드시 있어야 하는 데이터는 무엇이고, 선택적인 데이터는 무엇입니까?
- .. “xyz”을 지원하는 소프트웨어를 사용하고 있다면 화면 이미지에 대해 알려주시겠습니까?

– “xyz”의 완전한 시나리오를 말해주시겠습니까?

시나리오란 그 유즈케이스가 어떻게 수행되는지에 관한 이야기임을 **stakeholder**에게 설명합니다. 시나리오는 구체적이어야 하고, 정상적으로 수행되지 않은 경우에 대한 시나리오도 요청합니다.

– “xyz” 작업을 위해 보고서를 만들어야 합니까?

시스템에서 생성된 보고서는 모두 **SRS** 문서에 기재되어야 하고, 모든 필드와 생성된 방법을 분석해야 합니다.

– “xyz” 작업을 위해 외부 시스템과의 작업이 필요합니까?

간혹 유즈케이스에서 제3의 애플리케이션이나 툴이 사용될 수 있습니다. 이런 외부 시스템에 대한 정보를 기록해야 합니다.

– “xyz” 작업을 위해 외부 데이터가 필요합니까?

어떤 작업은 지도나 지역 행사, 역사 정보처럼 외부에서 제공되는 데이터가 필요합니다. 외부 데이터의 출처도 기록해 두어야 합니다.

추가적으로 적어둘 사항으로 다음과 같은 것들이 있습니다.

- Actor 간 용어의 차이

도메인 전문 용어나 다른 뜻을 갖는 일반 용어에 대해서는 그 뜻을 정확히 파악하고 프로젝트 **glossary**에 추가해야 합니다. 그리고 같은 용어를 다른 뜻으로 사용할 때는 그 차이점을 비교해서 그 중 하나를 주요 용어로 선택하고 다른 것들은 동의어로 취급합니다.

- Actor 간 유즈케이스의 목적과 흐름의 차이

동일한 유즈케이스에 대해 **actor**의 시각이 다른 것은 자연스러운 일입니다. 이 경우 회의를 통해 그 차이점을 비교해서, 유즈케이스에 맞는 목적과 흐름을 선택해야 합니다.

(1) 요구 사항 추출에 관한 문제

stakeholder의 표현이 부정확해 지는 경우는 다음과 같이 구분할 수 있습니다.

① 삭제(deletion) : 정보를 필터링한 경우

Stakeholder가 정보를 빼먹은 경우에 발생합니다. 세부 사항도 놓치지 않도록 **Stakeholder**에게 질문을 해야 합니다.

② 왜곡(distortion) : 정보를 왜곡한 경우

Stakeholder가 실제로 일어나지 않는 일을 만들어서 말하는 경우에 발생합니다.

③ 일반화(generalization)

Stakeholder가 너무 포괄적이거나 너무 한정적으로 말하는 경우 발생합니다. 일반적으로 ‘항상’, ‘반드시’, ‘절대로’, ‘모두’, ‘아무도’, ... 와 같은 용어를 사용할 때 발생할 수 있습니다.

보충

n 일반화(generalization)

다음의 예를 보겠습니다.

receptionist : 저는 체크인 할 때 신용카드를 항상 스캔합니다.

analyst : 예약 시 신용카드가 필요 없는 고객일 경우는요?

receptionist : 아, 잊었군요. 우리는 종종 회사에서 단체로 예약한 고객을 받기도 합니다. 그럴 경우는 개인 신용카드 조회가 필요 없습니
다.

receptionist는 단체 예약의 경우를 잊은 채로 보통의 경우에 ‘항상’이라는 용어를 사용했습니다. 이 경우 자칫 잘못하면 고객의 체크인에는 신용카드를 조회하는 것을 규칙으로 만드는 실수를 범할 수 있습니다. 경우의 수를 따져서 규칙의 예외 사항을 두거나 아예 여러 규칙을 만드는 분석이 필요합니다.

이런 문제는 **stakeholder**들이 거짓말을 하려 해서 발생하는 것이 아니라, 부주의로 인간의 심리적, 언어적 영향으로 발생하는 것입니다.

3) NFR 세부 사항 파악을 위한 질문

(1) 개요

NFR(비기능적 요구 사항)에 대한 질문은 때로는 좀더 모호합니다. NFR에 대한 인터뷰는 복잡하고 이 과정에서 벗어나는 부분이기 때문에 몇 가지 팁만 알려드립니다.

아래 제시된 것에서부터 시작합니다.

- 시스템의 어떤 성능이 중요한지 알아야 합니다.
- 누구에게 물어볼지 알아야 합니다.
- 잘 들어야 할 중요 구문과 어떤 질문을 해야 할지 알아야 합니다.

NFR은 시스템이 시스템과 사용자 사이의 경계에서 얼마나 잘 작동하는지를 결정짓습니다.

참고하세요

n Table :: 시스템 quality 분류

Manifest	Operational	Developmental	Evolutionary
성능	처리량	실체화	Scalability
가용성	관리 용이성	계획성	유지보수성
사용성	보안성		확장성
	편리성		유연성
	Testability		재사용성
	신뢰성		이동성

- **Manifest** – 엔드 유저의 작업을 수행하는 시스템의 수준
- **Operational** – 엔드 유저에 의해 직접 실행되는 것이 아닌 실행시의 시스템의 수준
- **Developmental** – 시스템의 구현에 대한 수준
- **Evolutionary** – 시스템의 장기간 오너쉽(ownership)의 비용에 대한 수준

(2) 성능(Performance)

성능은 일정 시간 범위 안에서 단일 사용자의 작업을 진행하는 능력을 말합니다.

성능과 관련된 문제는 요구 사항 인터뷰에서 발생할 수도 있습니다.

- 때때로 사용자가 작업 처리 속도를 언급할 수 있습니다.
- 엔드-유저의 관리자들이 단일 유즈케이스를 완료하는 데 소요되는 총 시간과 개

인별 소요시간을 미리 정할 수도 있습니다.

- 시스템 관리자가 시스템 자원의 수용량과 처리량의 기대치를 알고 있는 경우도 있습니다.

사용자가 성능과 관련된 다음과 같은 말을 할 수 있습니다.

“지금 사용하고 있는 시스템은 버튼을 클릭하고 나서 응답시간이 너무 길입니다. 5초 내에 응답이 있어야 합니다.”

분석가는 성능에 관해서 다음과 같은 질문을 할 수 있습니다.

- 처리 속도가 얼마나 빨라야 합니까?
- 평상시와 사용이 집중되는 때에 데이터베이스에 얼마나 많은 트랜잭션이 발생합니까?
- 몇 명의 동시 사용자가 시스템을 사용합니까?

(3) 확장성(Scalability)

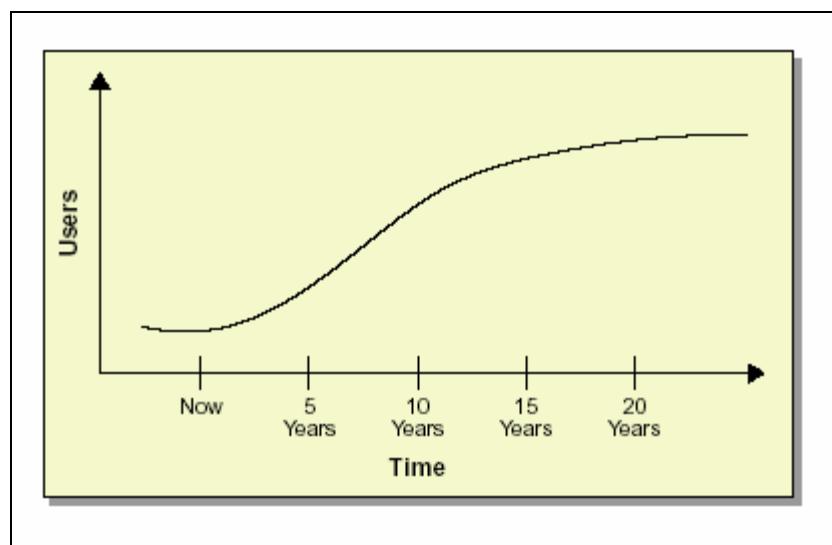
확장성은 처리량을 증가시키는 능력입니다.

확장성은 성능과 관련이 있습니다. 만일 웹 애플리케이션을 수년 후 사용자 증가에 대비해 분산 서버 환경으로 구축했다면, 그 애플리케이션은 확장성이 좋은 것입니다.

[이미지]

■ Scalability

- 시간이 지남에 따라 사용자가 점차 증가하고 있습니다.



확장성은 그 사업의 성장과 관련된 것이기 때문에, 비즈니스 오너와 관리자들이 가장 바

람직한 관점을 갖고 있습니다.

확장성 같은 **NFR**은 시스템 아키텍처에 의해 결정됩니다.

관리자들은 다음 사항에 대해 알아봐야 합니다.

- 평상시와 사용량이 최대인 경우의 시스템 부하
- 평상시와 사용량이 최대인 경우 때의 네트워크 사용량
- 데이터베이스 테이블의 평균 증가량
- 하드웨어 업그레이드 계획

(4) 사용성(usability)

사용성은 쉽게 사용할 수 있도록, 일반적인 사용자의 반응에 맞추는 것입니다

측정하기 어렵지만 목적은 시스템의 사용을 쉽게 하는 것입니다.

- 먼저, 사용자의 능력을 분석해야 합니다.
 - .. 인터뷰 과정에서 사용자를 관찰하거나, 실제 작업하는 모습을 관찰해서 알 수 있습니다.
 - .. 또는 관리자에게 물어볼 수도 있습니다.
- 사용자의 수준에 맞춰 특정 사용성과 관련된 항목을 결정합니다.

-

(5) actor 정보(Actor Information)

시스템을 사용할 사용자에 대해서 다음과 같은 정보가 필요합니다.

- 회사에서의 역할
- 업무 내용 기술
- 시스템의 주 사용자
- 한명의 사용자가 시스템을 한번 사용하는 평균 시간
- 시스템 사용 평균 빈도수
- 사용자의 교육 수준
- 해당 업무에 대한 전문성
- 컴퓨터, 하드웨어, 운영체제 등에 대한 경력
- 시스템을 사용하기 위해 필요한 교육 정도
- 시스템 적응도

(6) 사용성에 대한 연구(Usability Consideration)

소프트웨어 사용성 항목은 매우 다양하고 복잡합니다. 자세한 내용은 이 과정의 내용에서 벗어나지만, 몇 가지만 살펴보겠습니다.

- 사용하고 있는 유사 시스템과 제안된 시스템의 일관성
사용자가 시스템을 쉽게 사용하도록 하기 위해서는 동일한 작업에 대한 프로그램들은 화면 구성에 일관성을 유지해야 합니다.
- 룩앤플(**look and feel**)
사용자가 원하는 인터페이스 형식을 따르도록 합니다.
- 지역화와 국제화
국제적으로 사용될 시스템이라면 다양한 언어를 지원해야 하고, 국한된 지역에서만 사용될 시스템이라면 고유 데이터 표기법이나 특수 문자 등을 지원해야 합니다.
- 화면 이동과 워크플로우
사용자 인터페이스의 이동을 어떻게 할지 정해야 합니다. 작업 순서에 따라서 하나의 창에서 보여줄지, 팝업창을 사용할지, 다른 창에서 보여줄지 등을 정합니다.
- 이해도(**Accessibility**)
이해도란 많은 도움을 주는 보조 기능을 포함해 사람들이 편안하게 소프트웨어를 사용할 수 있는 정도를 말합니다.

개발팀이 사용자 인터페이스에 대한 프로토타입을 만들고 나면, 사용성에 대한 좀 더 구체적인 내용을 다룰 수 있습니다.

보충

n 프로토타입

프로토타입이란 새로운 컴퓨터 시스템이나 소프트웨어의 설계 또는 성능, 구현 가능성, 운용 가능성을 평가하거나 요구 사항을 좀 더 잘 이해하고 결정하기 위하여 전체적인 기능을 간략한 형태로 구현한 초기 모델을 말합니다. 이것은 특히 시스템에 대해서 잘 모르고 있는 비즈니스 오너들에게 초기 요구 사항을 명확히 할 수 있도록 가이드 역할을 하는 좋은 가시(**Visible**)를 입니다.

(7) 보안성(Security)

보안성은 비즈니스 서비스와 정보에 대한 접근을 제한하고, 보안의 취약점을 발견하고 분리, 재생하는 것을 말합니다.

특히, 기업 환경의 애플리케이션에서는 누가 시스템의 어느 부분에 접근하는지에 대한 철저한 제어가 필요합니다.

- 대개 사용자의 직무에 따라서 초기 보안 수준을 결정합니다.
사용자들의 **job role**은 실행 가능한 유즈케이스이기 때문입니다. 예약 담당 직원은 예약 관련 유즈케이스는 실행할 수 있지만, 판촉 관련 유즈케이스는 실행할 수 없습니다.

- 다음의 문제에 대해서 생각해봐야 합니다.
 - .. 사용자와 비밀번호는 어떻게 저장할 것입니까?
 - .. 회사 안팎에 보안 관련 위험 요소가 존재합니까?
 - .. 보안 수준을 어느 정도 세분화해야 합니까?
 - .. 웹 어플리케이션을 사용한다면 인터넷을 통해 얻게 되는 중요한 데이터(신용카드번호, 비밀번호 등)로 어떤 것이 있습니까?

3. SRS 문서 작성

1) 개요

SRS 문서는 시스템 요구사항을 종합해서 기록한 문서로, 클라이언트 측과 **stakeholder** 그리고 개발팀 간의 계약입니다. 이 문서는 주로 개발팀에서 사용하지만, 비즈니스 오너와 다른 **stakeholder**도 **SRS** 문서를 사용할 수 있기 때문에, 비전문가도 이해할 수 있게 작성해야 합니다.

SRS 문서는 다음과 같은 내용으로 구성되어 있습니다.

- 도입
- 제약 사항과 가설
- 위험 요소
- 기능적 요구 사항
- 비기능적 요구 사항
- 프로젝트 용어집
- 초기 유즈케이스 다이어그램

SRS 문서는 초점을 잊지 않고 자세하게 기술해야 합니다.

2) Introduction 작성

도입부분에는 다음과 같은 내용이 포함됩니다.

- 목적 - 문서의 목적
- 범위 - **problem statement**
- 시스템 환경 - 외부 시스템 인터페이스
- 주요 **stakeholders** - 파악된 **stakeholder** 리스트
- 약어 - 축약어 리스트
- 문서의 구성 - 이 문서의 주요 절에 대한 내용 기술
- 설계 변경 지시 - 요구 사항 변경 시의 처리 방법 기술
- 참고문헌 - **SRS** 문서 작성에 사용된 모든 문서

3) 기능적 요구 사항 작성

기능적 요구 사항에 대한 절은 다음과 같은 내용을 다룹니다.

- 주요 기능 - 제안서의 **high-level FR**
제안서에서 파악한 주요 기능을 다시 적어둡니다.
- 사용자(**actors**) - 시스템의 **actor**와 **actor role**에 대한 세부 사항을 정리해 놓습니다.
- 유즈케이스 - 유즈케이스의 우선순위, 고유 번호, 간단한 설명을 표로 작성합니다.
- 어플리케이션 - 시스템의 독립적인 어플리케이션과 세부 사항, 그리고 지원하는 유즈 케이스 등을 표로 작성합니다.
- 유즈케이스의 세부 요구 사항 - 유즈케이스에 대한 자세한 요구사항을 표로 작성합니다.

(1) Actors

알고 있는 모든 **actor**들의 리스트와 각 **actor**의 업무에 대한 간결한 설명을 제공해야 합니다.

Actor Section에서 다음과 같은 내용을 기술합니다.

예) 예약 담당 직원에 대한 기술 내용

이 직원은 전화로 예약을 받습니다. … 고등학교 졸업의 학력이지만, 마이크로소프트 사의 윈도우 **OS**를 잘 사용합니다. … 그러나 시스템 사용에 대한 교육이 필요합니다.

(2) 유즈케이스

유즈케이스 절에서는 모든 유즈케이스의 우선 순위, 고유 번호, 세부 설명을 표로 작성합니다.

Use Cases Section에서는 다음과 같은 내용을 기술합니다.

Use Case Name	Priority	Number	Description
예약 관리	E	1	이 유즈케이스는 예약 담당 직원이 예약 정보와 고객 정보를 입력, 수정, 삭제하는 것을 허용합니다.
이벤트 관리	H	8	이 유즈케이스는 행사 담당 직원이 컨퍼런스에 대한 정보를 등록, 수정, 삭제하는 것을 허용합니다.

평가서 전송	F	12	이 유즈케이스는 시스템이 고객의 평가서를 전송할 시간을 프로그래머가 저알 수 있도록 합니다.
--------	----------	-----------	---

유즈케이스의 이름은 유즈케이스 워크플로우의 특징을 간단한 구문으로 표현합니다. 우선순위는 **E**는 필수(**essential**), **H**는 **essential** 보다는 낮지만 높은 경우(**High-Value**), **F**는 그리 높지 않은 경우(**Follow-on**)를 의미합니다. 우선순위와 고유 번호와의 조합으로 유즈케이스를 표현할 수 있습니다.

(3) 어플리케이션

어플리케이션 절은 시스템의 독립적인 어플리케이션들을 표로 보여줍니다.

Application Section에서는 다음과 같은 내용을 기술합니다.

Application	Description / Use Cases
HotelApp	이 독립형 어플리케이션은 호텔과 소규모 이벤트 센터를 관리하는 주요 기능을 자동화합니다. 관련 UC : E1, E2, E3, E6, H7, H8, F9, F10, F11, F12
WebPresenceApp	이 웹 사이트 어플리케이션은 고객이 호텔의 부대시설을 보고 예약을 할 수 있게 해줍니다. 관련 UC : E4, E5
KioskApp	...

(4) 세부 요구 사항

세부 요구 사항 절은 각 유즈케이스에 있는 세부적인 **FR**에 대한 전체 리스트입니다. 구체적인 **FR**은 유즈케이스 워크플로우의 기능적인 면에 대한 간결한 설명이 포함되어 있습니다.

각 **FR**은 유즈케이스의 우선순위와 고유 번호를 조합한 코드로 표현할 수 있습니다.

FR	Requirement Description
E1- 1	시스템에서는 예약 담당 직원이 예약 정보를 추가, 삭제, 변경 하는 작업이 허용됩니다.
E1- 2	예약은 일정 기간동안 하나 또는 그 이상의 방에 대해 유효합니다.
...	...

① 세부 요구 사항 작성

FR 설명서를 작성하는데 중요한 점은 “~해야 한다, ~하기로 되어 있다(**shall**)”의 표현을 사용하는 것입니다. 그 보다 약한 표현인 “~할 수 있다, ~할지도 모른다”는 표현은 사용해서는 안됩니다.

가능한 자세하게 작성해야 하는데, 아래의 몇 가지를 고려해야 합니다.

- 유즈케이스에 필요하거나 수집된 데이터를 기술합니다.
예) **E1-3**에서, 시스템은 고객의 이름과 전화번호를 수집해야 합니다.
- 객체간 또는 시스템의 기능간의 관계를 기술합니다.
예) **E1-2**에서 ‘예약’은 하나 또는 그 이상의 ‘방’과 관련이 있습니다.
- 해당 작업을 완수하기 위한 모든 방법을 기술합니다.
- 객체와 기능에 대한 모든 제약 사항을 기술합니다.
예) **E1-3**은 고객의 이름은 두 단어로 작성합니다.

② 기록의 중요성

세부적인 **FR**은 시스템이 반드시 수행해야 하는 작업을 정의하고 있습니다. 다른 워크플로우를 통해서 시스템이 요구사항을 만족하도록 검증하는 것이 중요합니다.

- **UML** 표기법에 **FR** 코드를 사용해서 그 컴포넌트가 어떤 요구 사항을 만족시키는지 표시합니다.
- 소스 코드에 **FR** 코드를 사용해서 그 코드가 어떤 요구 사항을 구현했는지 표시합니다.
- 테스트 계획을 세울 때 **FR** 코드를 사용해서 그 테스트가 어떤 요구 사항과 관련된 것인지 표시합니다.

4) 비기능적 요구 사항 작성

비기능적 요구 사항 절은 구체적인 **NFR**의 모든 항목을 말합니다. **NFR**을 잘 작성하는 방법은 **FR**을 작성하는 방법과 같습니다.

- 각 **NFR**은 유즈케이스 코드를 사용합니다.
- 각 **NFR**은 간결한 설명을 포함하고 있습니다.

NFR section에서는 다음과 같은 내용을 기술합니다.

표) 호텔 예약 시스템의 **NFR**

NFR	Required Description
E1- 101	과거의 기록을 통해서 볼 때, B&B 에는 한 달에 약 150 건의 예약이 있고, 리조트 시설에는 한 달에 약 1000 건의 예약이 있습니다. 따라서, 시스템에서는 최소한 한 달에 1300 건의 예약을 처리가 가능해야 합니다.
E1- 102	HotelApp 의 GUI 는 사용자의 입력에 대한 응답시간이 5초 이내여야 합니다. 이것은 네트워크의 차이를 무시하기 위하여 어플리케이션 서버에서 확인할 것입니다.
...	...

5) 프로젝트 용어집

SRS 문서의 용어집 부분에는 다음과 같은 내용이 들어갑니다.

- **도메인 전문 용어(Domain terms)**

업무에서 사용되는 일상 회화 속의 단어와 구문을 나열하고 정의해야 합니다.

- **동의어**

다른 용어를 사용자에 따라서 같은 뜻으로 사용하는 경우, 한 가지는 주요 용어로 정의하고 나머지는 그 말과 동의어로 구분해야 합니다.

- **기술 용어**

SRS 문서에서 사용하는 모든 기술 용어를 나열하고 정의해야 합니다.

- **소프트웨어 개발 용어**

SRS 문서에서 사용하는 모든 소프트웨어 개발 용어를 나열하고 정의해야 합니다.

과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원2 : 요구공학

3 모듈 : 초기 유즈케이스 다이어그램 작성

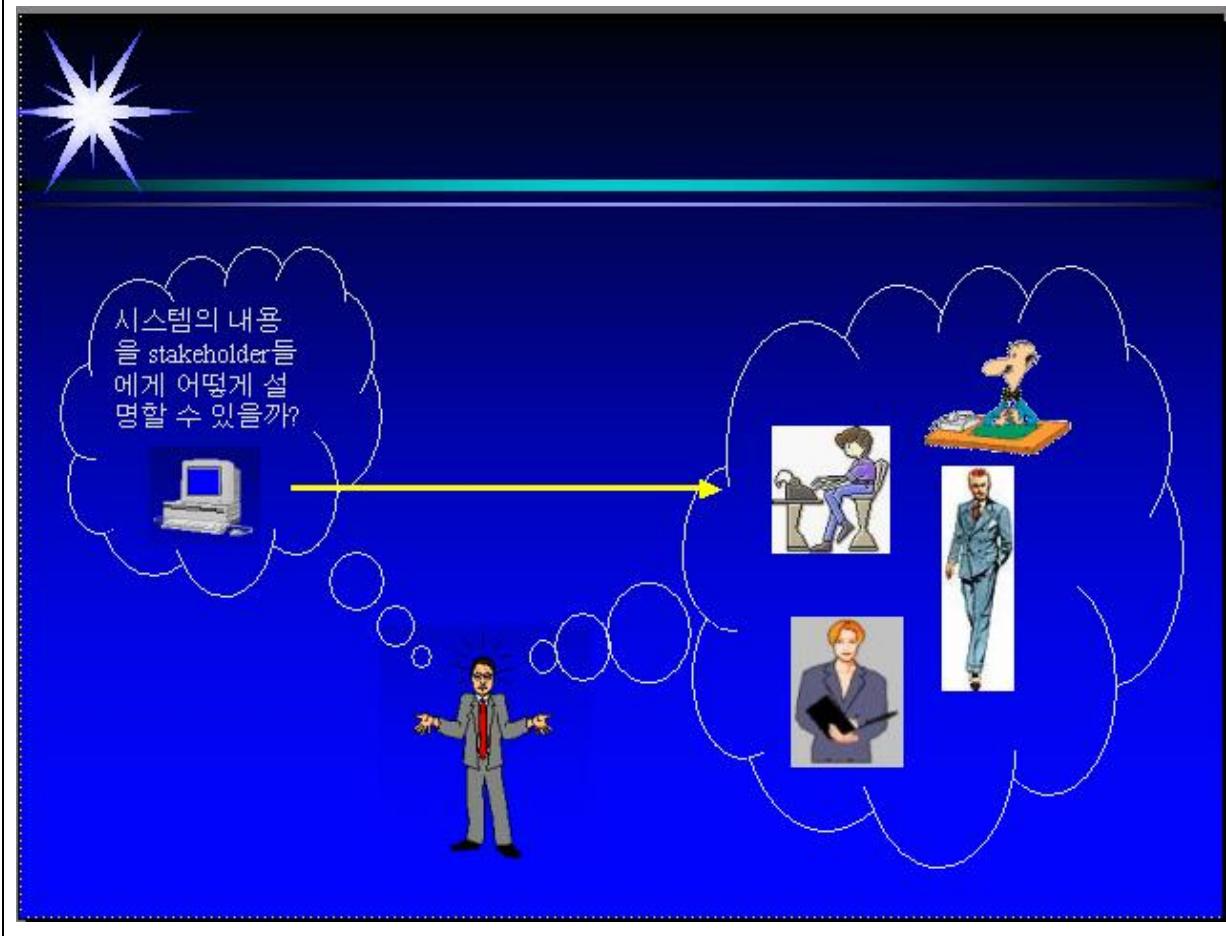
담당강사 : 전은수

▣ 생각해봅시다 ▣

이전에 배운 **SRS** 문서는 모든 **stakeholder**들의 요구 사항을 자세하게 기술한 문서이기 때문에, 종종 클라이언트 측의 다른 **stakeholder**들은 이해하기 힘들 수 있습니다. 그 사람들에게 시스템 전체를 보여주려면 어떻게 해야 할까요? 어려운 용어와 긴 말로 설명하는 것 보다는 시스템을 간단한 그림으로 표현하는 게 이해를 도울 수 있을 것입니다. 이번 장에서 배우게 될 유즈케이스 다이어그램을 사용하면 시스템의 주요 기능을 그림으로 쉽게 표현할 수 있습니다.

애니메이션

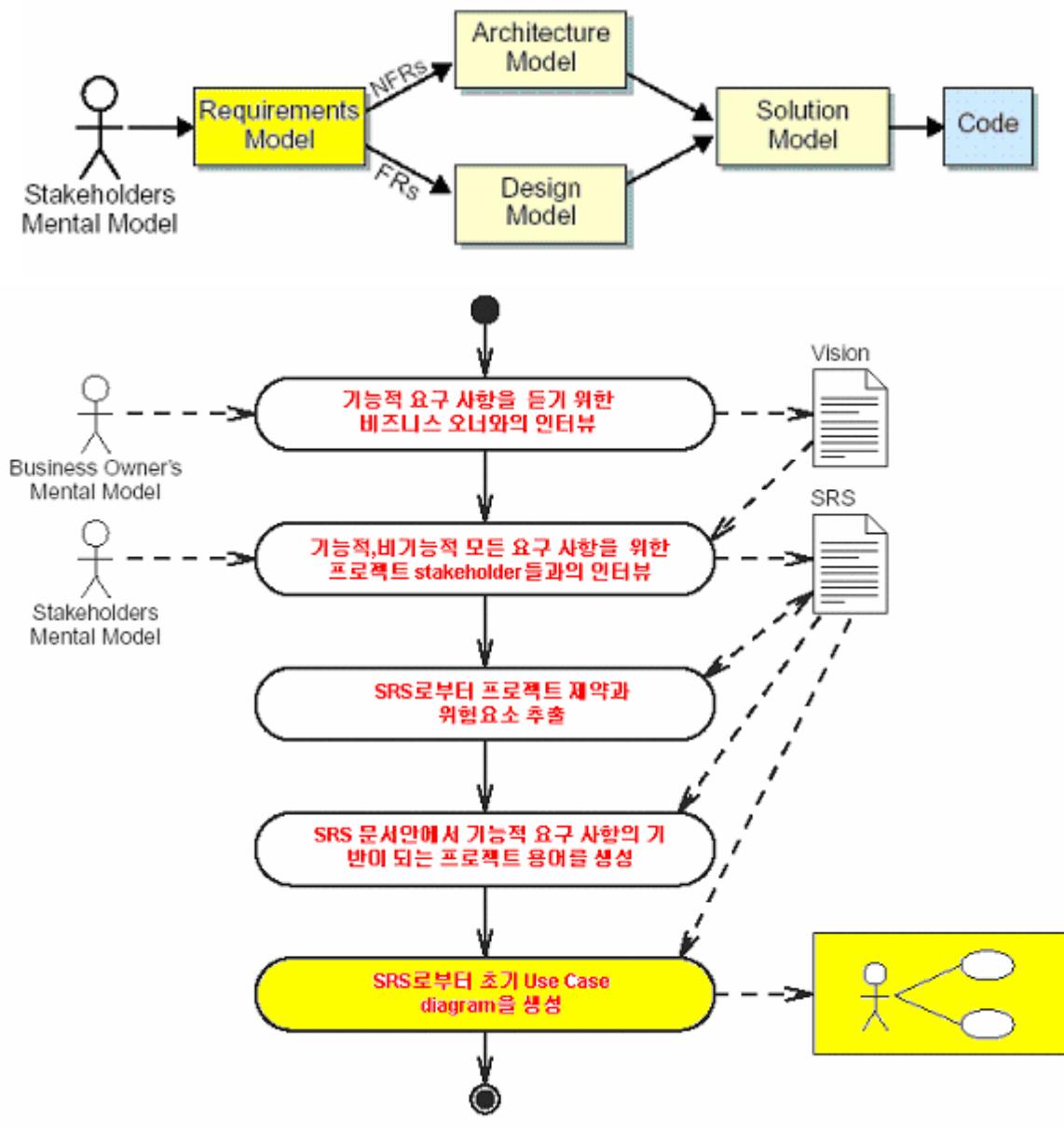
비즈니스 분석가 : “프로젝트의 모든 참가자들과 인터뷰한 내용은 나름대로 잘 정리를 해 두었는데, 과연 모든 참가자들이 이 문서의 내용을 이해할 수 있을까? 글보다 그림으로 표현하면 좀 더 쉽게 이해 할 수 있지 않을까?”



■ 학습하기 ■

참고하세요

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. 유즈케이스 다이어그램의 필요성

이 모듈에서는 **SRS(System Requirement Specification)**의 유즈케이스와 기능적 요구사항(**FR**)을 사용해서 유즈케이스 다이어그램을 작성하는 방법에 대해 설명합니다. 유즈케이스 다이어그램의 목적은 다음과 같습니다.

- **SRS**는 구체적인 요구 사항을 표현하고 있으며, 텍스트 기반의 문서입니다.
유즈케이스 다이어그램은 **FR**을 시각적으로 표현한 것입니다. 경우에 따라 클라이언트측에서 **SRS** 보다 유즈케이스 다이어그램을 더 쉽게 이해할 수 있습니다.
- **클라이언트 측의 stakeholder**에게는 시스템에 대한 큰 그림이 필요합니다.
유즈케이스 다이어그램은 전체 시스템에 대한 **high-level** 관점을 제공합니다. 이것은 시스템을 하나의 이미지로 보여주는 것으로, **개발팀과 클라이언트 측 사이의 의사소통에 기반이 될 수 있습니다.**
- 시스템의 유즈케이스는 개발에 초점을 두어 작성한 것입니다.
유즈케이스는 시스템 개발에 중점을 둔 것이고, 유즈케이스 다이어그램은 개발팀에 대한 **high-level** 가이드가 됩니다.

2. 유즈케이스 다이어그램의 구성 요소

1) 개요

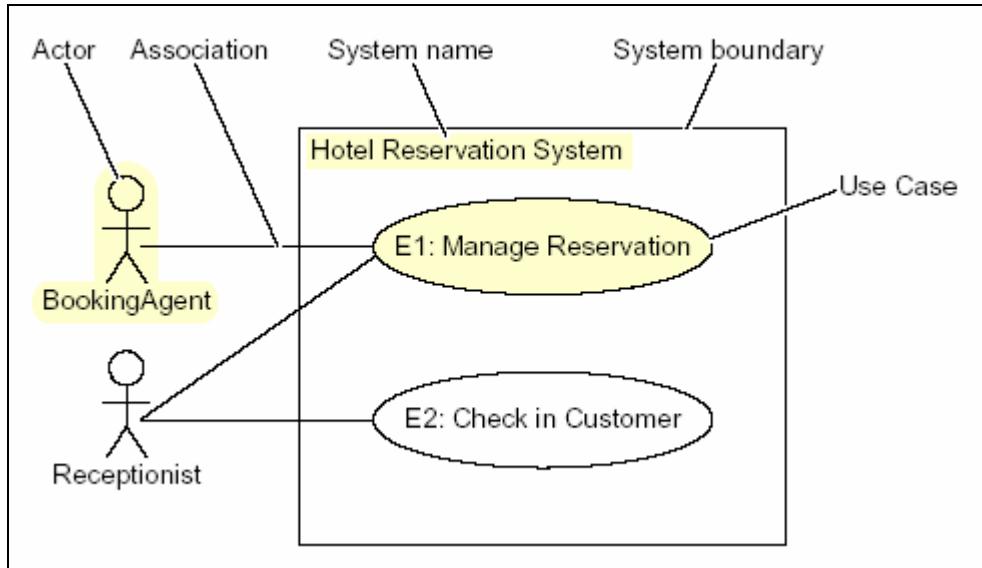
유즈케이스 다이어그램은 “시스템 안의 사용자와 유즈케이스 간의 관계를 보여주는 다이어그램”입니다.(UML v1.4 spec. page B- 21)

유즈케이스 다이어그램은 시스템과 시스템의 유즈케이스, 그리고 특정 기능을 수행하기 위해 시스템을 사용하는 사용자를 시각적으로 표현합니다.

다음 그림은 간단한 유즈케이스 다이어그램입니다.

[이미지]

■ 유즈케이스 다이어그램의 예



2) Actors

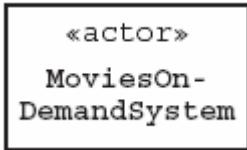
“actor는 시스템과 상호작용을 하는 사용자”를 말합니다.(Fowler UML Distilled page 42)

또 actor는 “유즈케이스들과 상호작용을 하는 유즈케이스의 사용자의 집단”입니다.(UML v1.4 spec. page B- 3)

시스템에서 행동을 초기화하는 어떤 사람 또는 어떤 객체는 모두 actor입니다. Actor에는 기본적으로 세가지 부류- 사람, 외부 시스템, 그리고 시간이 있습니다 .

이미지

■ actor의 타입

		
BookingAgent	«actor» MoviesOn-DemandSystem	Time

이 아이콘은 사람 actor를 의미합니다.

이 아이콘은 다른 actor, 대개는 외부 시스템을 의미합니다.

이 아이콘은 시간(time-trigger mechanism)을 의미합니다.

보충

n time-trigger mechanism

일정 시간 간격으로 어떤 작업이 실행되는 것을 의미합니다. 예를 들어, 회원 정보에 입력된 고객의 생일에 축하 E-메일을 보내주는 것처럼, 시간에 따라서 특정 작업 또는 비즈니스 정책을 실행하는 것을 말합니다.

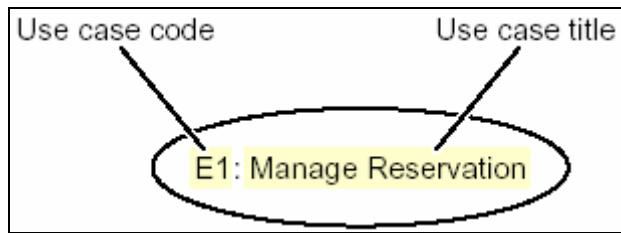
3) Use Case

유즈케이스는 “시스템이 수행할 수 있는 일련의(**sequence**) 작업의 설명서”입니다. (**UML v1.4 spec. page B- 2)**

유즈케이스는 결과값을 얻기 위해 **actor**와 시스템간의 상호작용을 표현한 것입니다.
다음 그림은 유즈케이스 다이어그램의 요소를 보여줍니다.

[이미지]

n 유즈케이스 예



- 유즈케이스는 결과와 함께 시스템 기능의 중요한 부분을 캡슐화합니다.
유즈케이스는 시스템의 주요 기능에 대한 세부사항을 시각적으로 캡슐화해서 보여줍니다.
- 타원의 가운데에 유즈케이스의 이름을 씁니다.
타원 아래에 쓸 수도 있습니다.
- 유즈케이스 이름 앞에 **SRS**의 참조를 위해 유즈케이스 코드를 쓸 수 있습니다.
UML 표준은 아니지만 사용할 수 있습니다. 유즈케이스 코드를 써두면 **SRS** 문서를 참조할 때 해당 유즈케이스를 빨리 찾을 수 있습니다.

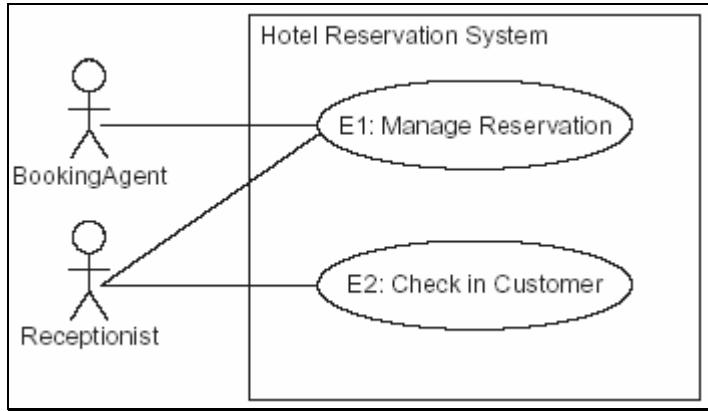
4) System Boundary

“경우에 따라 유즈케이스는 시스템의 경계를 의미하는 사각형 안에 그려질 수도 있습니다.”(**UML v1.4 spec. page 354**)

유즈케이스 다이어그램에는 대개 시스템 경계를 표시합니다.

[이미지]

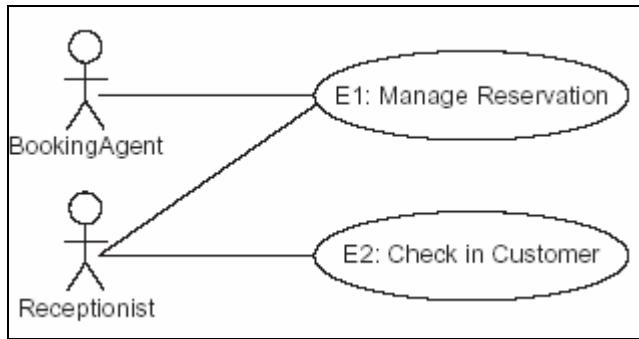
n 시스템 경계(System boundary)를 표시한 유즈케이스 다이어그램



하지만, 유즈케이스 다이어그램에 시스템 경계를 표시하지 않아도 됩니다.

이미지

- 시스템 경계(System boundary)를 표시하지 않은 유즈케이스 다이어그램



명확하게 하기 위해서는, 유즈케이스 다이어그램에 시스템 경계를 표시해 주어야 합니다.

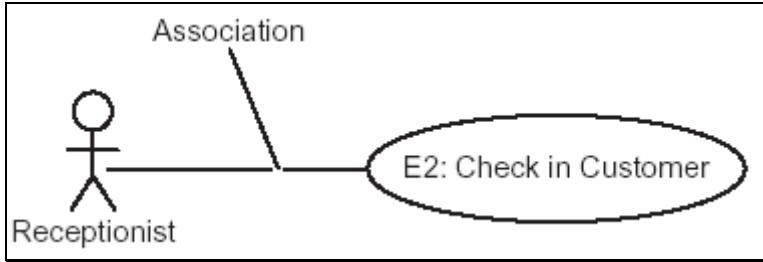
5) Use Case Associations

유즈케이스 연관 관계(association)는 “유즈케이스에 대한 **actor**의 관계”를 나타낸 것입니다.(UML v1.4 spec. page 357)

호텔 예약 시스템의 경우, 예약 담당 직원은 ‘**Check in Customer**’ 유즈케이스를 수행하기 위해 시스템을 사용합니다.

이미지

- 유즈케이스 관계



- **actor**는 반드시 하나 이상의 유즈케이스와 연관이 있어야 합니다.

유즈케이스와 연관이 없는 **actor**는 시스템을 사용하지 않는다는 뜻이므로 다이어그램에 그릴 필요가 없습니다.

- 유즈케이스는 반드시 하나 이상의 **actor**와 연관이 있어야 합니다.

이 부분은 [단원3.기능적 요구사항 분석- 모듈1. 상세 유즈케이스 다이어그램 작성]에서 다시 다루게 됩니다.

- 유즈케이스의 관계는 화살표가 없는 실선으로 나타냅니다.

3. 유즈케이스 다이어그램 작성

1) 유즈케이스 다이어그램 작성

유즈케이스 다이어그램을 작성하는 단계는 다음과 같습니다.

① 시스템 경계를 의미하는 사각형을 만들고 이름을 적습니다.

시스템 안에 무엇이 있는지 명확하게 서술할 수 있기 때문에 시스템 경계를 그리도록 합니다.

② SRS 문서를 통해 시스템의 모든 **actor**를 파악합니다.

SRS 문서에서 시스템을 사용할 **actor** 목록을 가져옵니다.

③ 각 **actor**에 대해서

i. 다이어그램에 **actor** 아이콘을 추가합니다.

다이어그램의 맨 위부터 시작해서, 그 아래에 계속 추가하는 것이 가장 쉽습니다.

ii. 해당 **actor**가 관여하는 유즈케이스를 추가합니다.

Actor와 관련된 유즈케이스를 파악하기 위해 **SRS** 문서의 유즈케이스 표를 참고합니다. 유즈케이스를 **actor**의 오른쪽에 두고, 위에서 아래로 추가하는 것이 가장 쉽습니다.

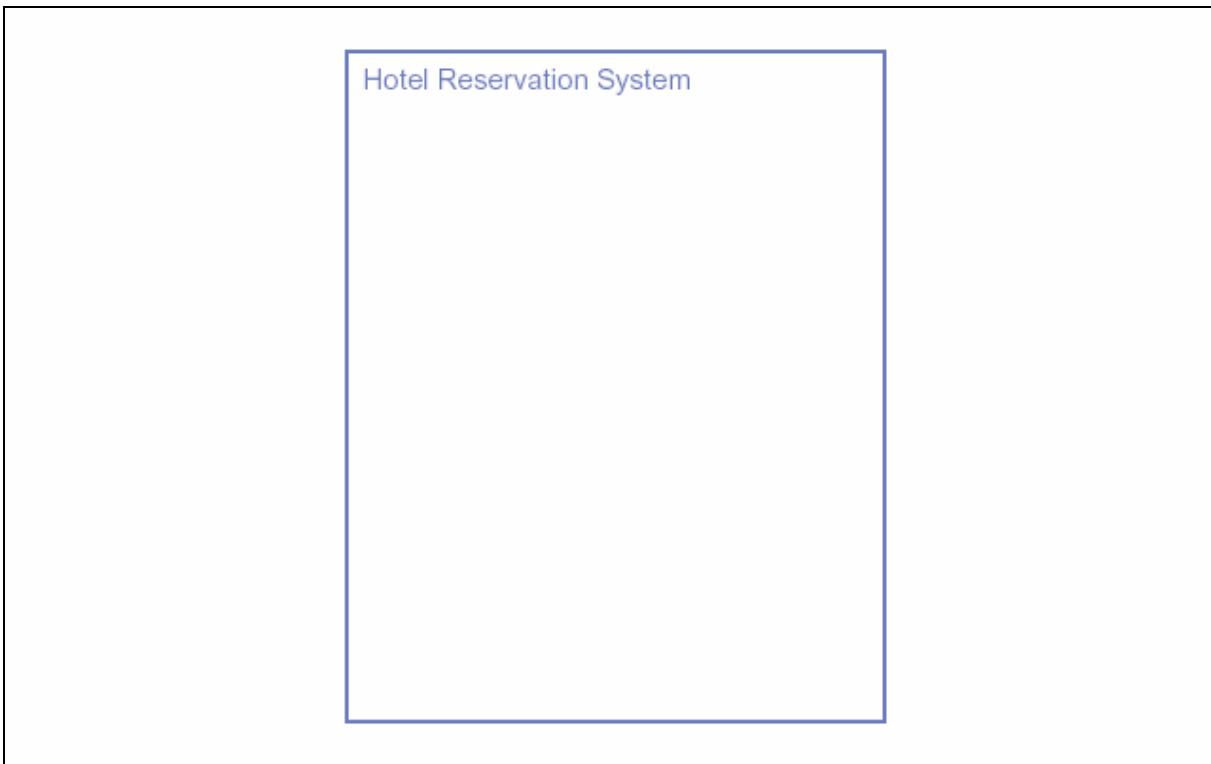
iii. 유즈케이스와 **actor**의 연관 관계를 그립니다.

간혹 여러 **actor**가 하나의 유즈케이스에 관여하는 경우가 있습니다. 유즈케이스가 이미 다이어그램에 존재한다면, 다시 그리지 말고 새 **actor**와의 연관 관계를 표시해 주면 됩니다.

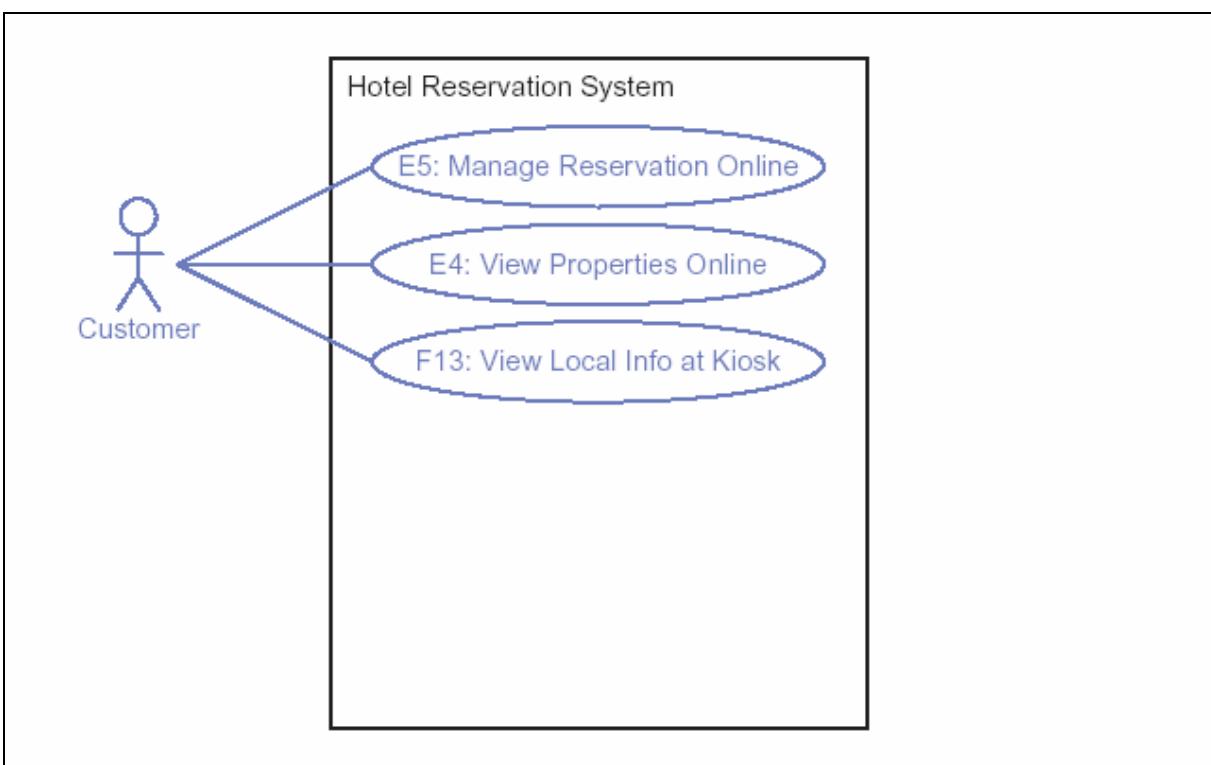
[데모보기]

■ 유즈케이스 다이어그램 작성

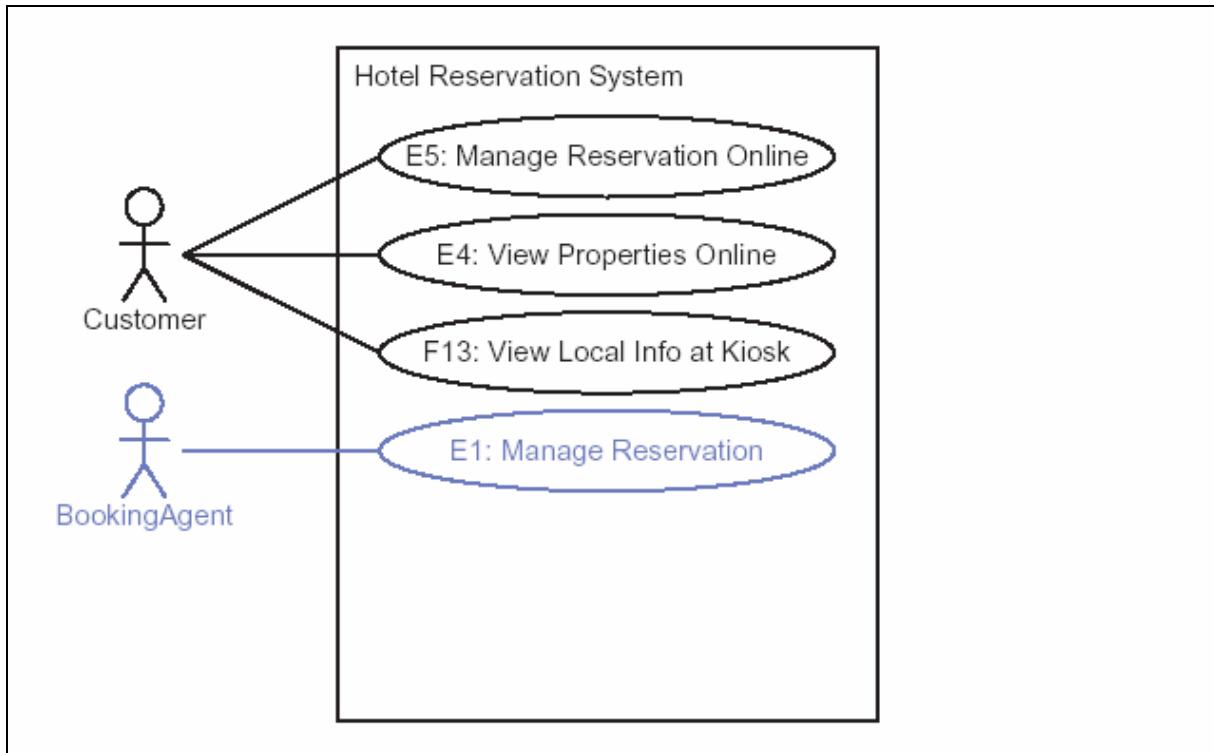
1/4 시스템 경계를 나타내는 사각형 작성



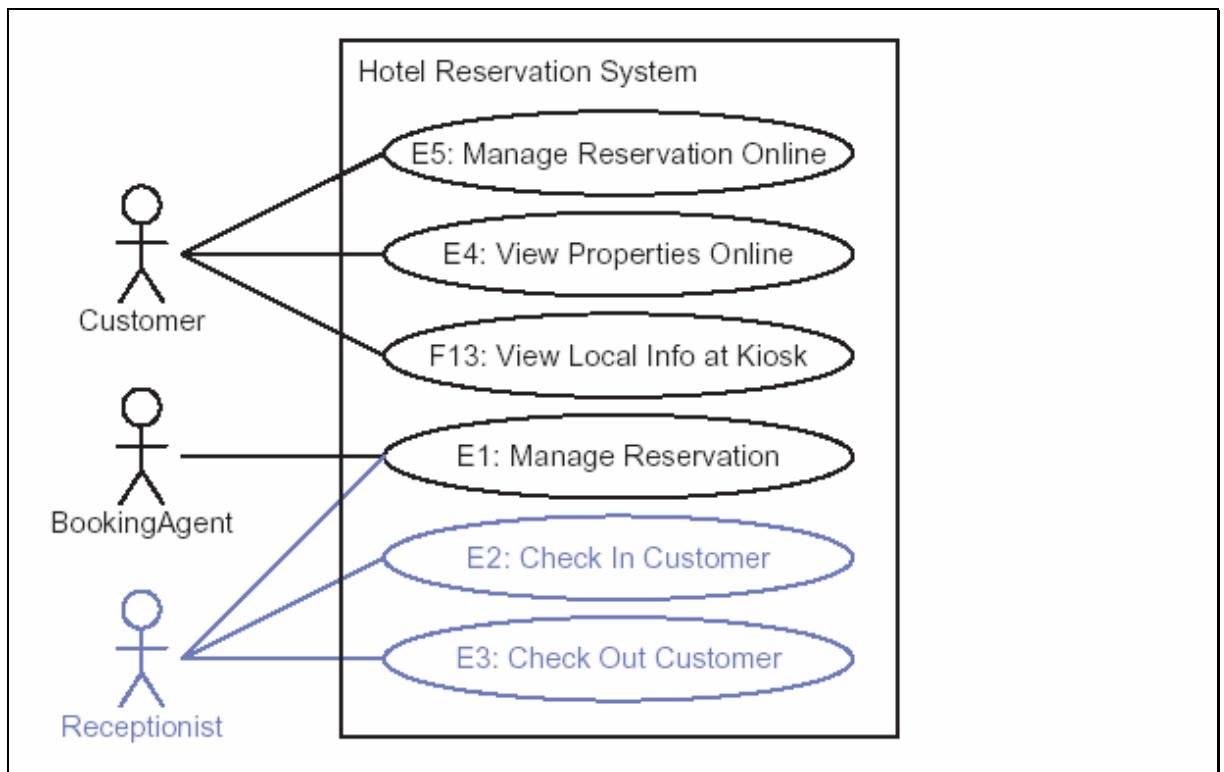
2/4 actor와 관련 유즈케이스 작성



3/4 actor와 유즈케이스 추가



4/4 여러 명의 actor가 유즈케이스를 공유하는 경우



2) 유즈케이스 다이어그램 저장

SRS 문서에 유즈케이스 다이어그램을 저장하는 것은 좋은 방법입니다.

- 유즈케이스 다이어그램은 **SRS** 문서의 기능적 요구사항을 시각적으로 표현하고 있습니다.
SRS 문서는 시스템의 요구사항 모델을 기록한 것입니다. 유즈케이스 다이어그램은 그 모델을 그림으로 표현한 것입니다.
- 유즈케이스 다이어그램을 **SRS** 문서에 두는 것은 두개의 산출물의 동기화를 유지해줍니다.
SRS 문서의 기능적 요구 사항이 변경되었을 때, 유즈케이스 다이어그램도 변경하는 것을 잊지 않게 됩니다.
- 유즈케이스 다이어그램의 주요 목적은 클라이언트에게 시스템의 기능에 대한 단순화된 관점을 제공하는 것입니다.
SRS 문서는 클라이언트 측 **stakeholder**와 개발팀 간의 주요 계약이고, 유즈케이스 다이어그램은 클라이언트가 시스템 요구 사항을 이해하는데 중요한 기능을 하는 시작적 도구입니다.

4. 유즈케이스 시나리오 작성

1) 개요

유즈케이스는 **actor**의 관점으로 시스템을 표현한 것입니다. 유즈케이스 시나리오는 유즈케이스의 구체적인 예입니다. 즉, 유즈케이스와 상호작용을 하는 **actor**의 인스턴스를 말합니다. 유즈케이스 시나리오는 비즈니스 업무나 기존 시스템을 관찰해서 작성할 수 있습니다.

유즈케이스 시나리오는

- 가능한 구체적이어야 합니다.

유즈케이스 시나리오는 유즈케이스를 명확하게 표현해야 합니다. 실제 클라이언트 측의 고객이나 직원들 이름을 사용할 수 있습니다.

- 조건문을 포함해서는 안됩니다.

조건식 표현을 사용하지 말고, 발생 가능한 모든 상황에 대한 유즈케이스 시나리오를 작성해야 합니다.

- 시작은 같지만, 결과는 다릅니다.

하나의 유즈케이스에 대한 유즈케이스 시나리오는 모두 같은 상태에서 같은 **actor**로 시작합니다. 그러나 중간에 발생 가능한 다른 상황에 따라 다른 결과를 얻을 수 있습니다.

- 사용자 인터페이스의 세부 사항에 대해서는 자세히 다루지 않습니다.

유즈케이스 시나리오는 유즈케이스 세부적인 워크플로우에 중점을 두어야 합니다.

UI(User Interface : 사용자 인터페이스)에 관한 내용이 들어갈 수 있지만 모든 **UI** 요소를 설명하지는 않습니다.

- 작업이 정상적으로 이루어진 경우와 그렇지 않은 경우를 별개의 시나리오로 모두 작성합니다.

작업이 성공적으로 끝난 경우 뿐만 아니라 실패한 경우에 대해서도 별도로 유즈케이스 시나리오를 작성해야 합니다.

유즈케이스 시나리오는 다른 몇몇 객체 지향 분석 및 설계의 워크플로우를 따릅니다. 【단원

3. 기능적 요구사항 분석- 모듈1. 상세 유즈케이스 다이어그램 작성】에서 **activity** 다이어그램에서 유즈케이스 시나리오를 사용하는 것을 보게 됩니다.

2) 유즈케이스 시나리오 선택

모든 유즈케이스에 대해 불필요하게 많은 시나리오를 작성하는 것은 시간 낭비입니다. 따라서, 다음과 같은 기준을 정해두고 시나리오 작성을 위한 유즈케이스를 선택할 수 있습니다.

- 유즈케이스가 **actor**와 복잡한 상호작용을 합니다.

단순한 유즈케이스에 대해서는 하나의 시나리오만으로도 충분하지만, 복잡한 유즈케이스의 경우는 워크플로우가 다른 여러 개의 시나리오를 작성하는 것이 중요합니다.

- 유즈케이스에 외부시스템이나 데이터베이스를 사용하는 등의 잠재된 **failure point**가 있습니다.

예를 들어, 고객의 신용카드에 문제가 생겼을 때 담당직원은 고객이 은행과 문제를 해결하거나 아니면 다른 신용카드를 제시할 때까지 예약 상태를 “처리 중”인 것으로 나타낼 수 있습니다. 이것은 서로 다른 두개의 시나리오로 작성해야 합니다.

참고하세요

n 유즈케이스 시나리오의 다른 표현

XP와 같은 방법론에서는 **user story**라고 하는 유즈케이스 시나리오를 광범위하게 사용합니다. **XP**에서는 시스템의 **FR**을 **user story**를 통해서 정의합니다.

시나리오는 두 가지 종류가 있습니다.

- 성공한 결과를 기록하는 **Primary** 시나리오

예를 들어, 고객이 방을 1개 예약하는 경우와 여러 개 예약해서 할인 혜택을 받는 경우에 대해 별개의 시나리오로 작성해야 합니다.

시나리오의 성공 여부는 클라이언트 측 **stakeholder**가 결정합니다.

- 실패한 경우를 기록하는 **Secondary** 시나리오

예를 들어, 고객이 방을 예약하려고 하는데 빈 방이 없다거나 또는 고객의 신용카드에 문제 가 있어 실패한 경우에 대해 별개의 시나리오로 작성해야 합니다.

3) 유즈케이스 시나리오 작성

유즈케이스 시나리오는

- **actor**가 어떻게 시스템을 사용하고, 시스템이 어떻게 응답하는지를 설명합니다.
- 처음- 중간- 끝으로 되어있습니다.
시작부분은 유즈케이스가 시작될 때 **actor**가 무엇을 하고 있는지 알려줍니다. **모든 유즈 케이스의 시작은 같습니다.** 처음 부분을 유즈케이스의 **trigger-point**라고 합니다.
중간부분은 **actor**가 시스템과 어떻게 상호 작용을 하는지에 관한 대부분의 내용을 담고 있습니다.
끝부분은 유즈케이스가 성공했는지 실패했는지에 관한 내용입니다.

다음은 호텔예약 시스템의 예약 시나리오입니다.

구분	시나리오 내용	추가 설명
처음 부분	예약 담당직원 Medoca Sansumi 이 호텔 예약을 원하는 Jane Googol 의 전화를 받고 HotelApp 의 메인 화면에서 “ Create Reservation ” 버튼을 누르면 예약 화면 양식이 나타납니다.	Manage Reservation 유즈케이스의 처음 부분은 예약 담당 직원이 고객의 전화를 기다리는 내용입니다. .HotelApp 의 메인 화면은 “ Create Reservation ” 기능의 버튼이 있어야 합니다.
중간 부분	Medoca 는 도착 날짜, 머무는 기간, 방의 종류 등을 묻습니다. 고객이 원하는 방이 있는지 시스템에서 검색 결과를 알려주고, 사용 가능한 방을 모두 보여줍니다. Medoca 는 그 중 하나를 선택하고 시스템에서는 예약 상태를 “처리 중”으로 둡니다. Medoca 가 시스템에 Jane 의 이름을 입력하면, 기존 고객이기 때문에 예약 화면에 고객 정보가 보여집니다. Jane 은 결제에 사용할 신용카드 번호와 유효 기간을 불러주고 Medoca 는 그 내용을 입력합니다. Medoca 가 “ Verify Payment ”를 선택하면 5초 후에 시스템은 작업이 성공적으로 끝났음을 알리고, 예약 상태를 “완료”로 변경합니다.	이 부분에서는 직원이 시스템을 사용하는데 필요한 사용자 인터페이스에 대한 내용이 나오지만 자세히 다루지 않는 게 좋습니다. 유즈케이스 시나리오는 UI 의 구성 보다는 워크플로우에 중점을 둔 것입니다.
끝 부분	Medoca 는 Jane 에게 시스템에서 부여한 예약 번호를 알려주고 예약 화면을 닫습니다.	시나리오의 끝부분은 유즈케이스의 post condition 을 나타냅니다. 이 단계에서는 작업을 완료하고 담당직원이 고객에게 예약 번호를 알려주면 됩니다.

4) 유즈케이스 시나리오 저장

유즈케이스 시나리오는 대개 **SRS** 문서와 별도로 저장합니다. 유즈케이스 시나리오를 작성하는데 특별한 법칙이 있는 것은 아니지만, 완전한 유즈케이스 단위로 그룹화하는 것이 좋습니다.

즉, 하나의 유즈케이스에 대한 모든 시나리오를 하나의 문서로 저장하는 것이 좋습니다.

과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원3 : 기능적 요구사항 분석

1 모듈 : 상세 유즈케이스 다이어그램 작성

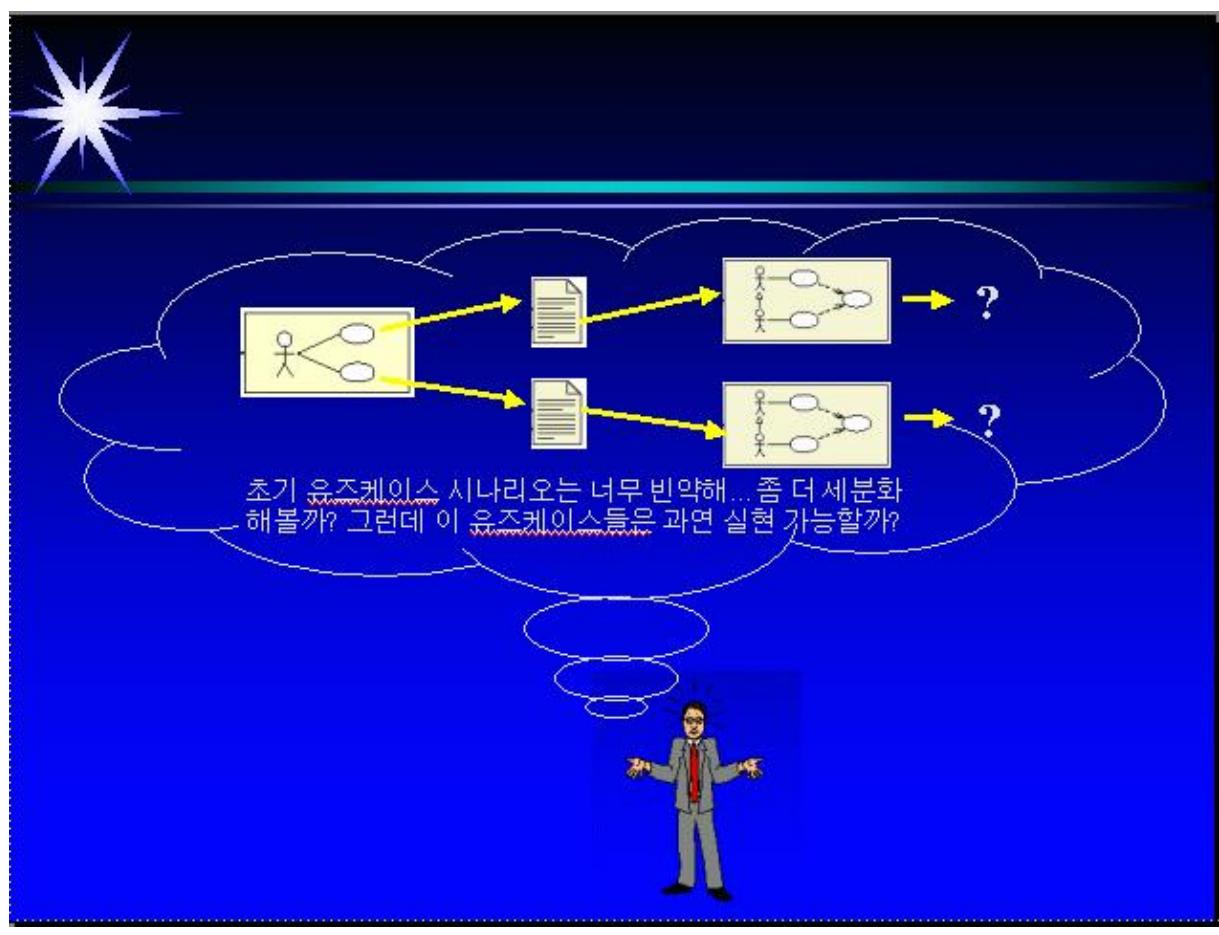
담당강사 : 전은수

■ 생각해봅시다 ■

SRS 문서의 기능적 요구 사항을 표현한 초기 유즈케이스 다이어그램은 다소 추상적입니다. 비슷한 기능을 하나의 유즈케이스로 다루었기 때문에, 하나의 유즈케이스에 대해서도 유즈케이스 시나리오가 다를 수 있습니다. 그렇다면 세부적인 사항까지도 표현할 수 있는 유즈케이스 다이어그램은 어떻게 그릴까요? 또 유즈케이스가 제대로 작성되었는지 검증할 수 있는 방법은 없을까요?

애니메이션

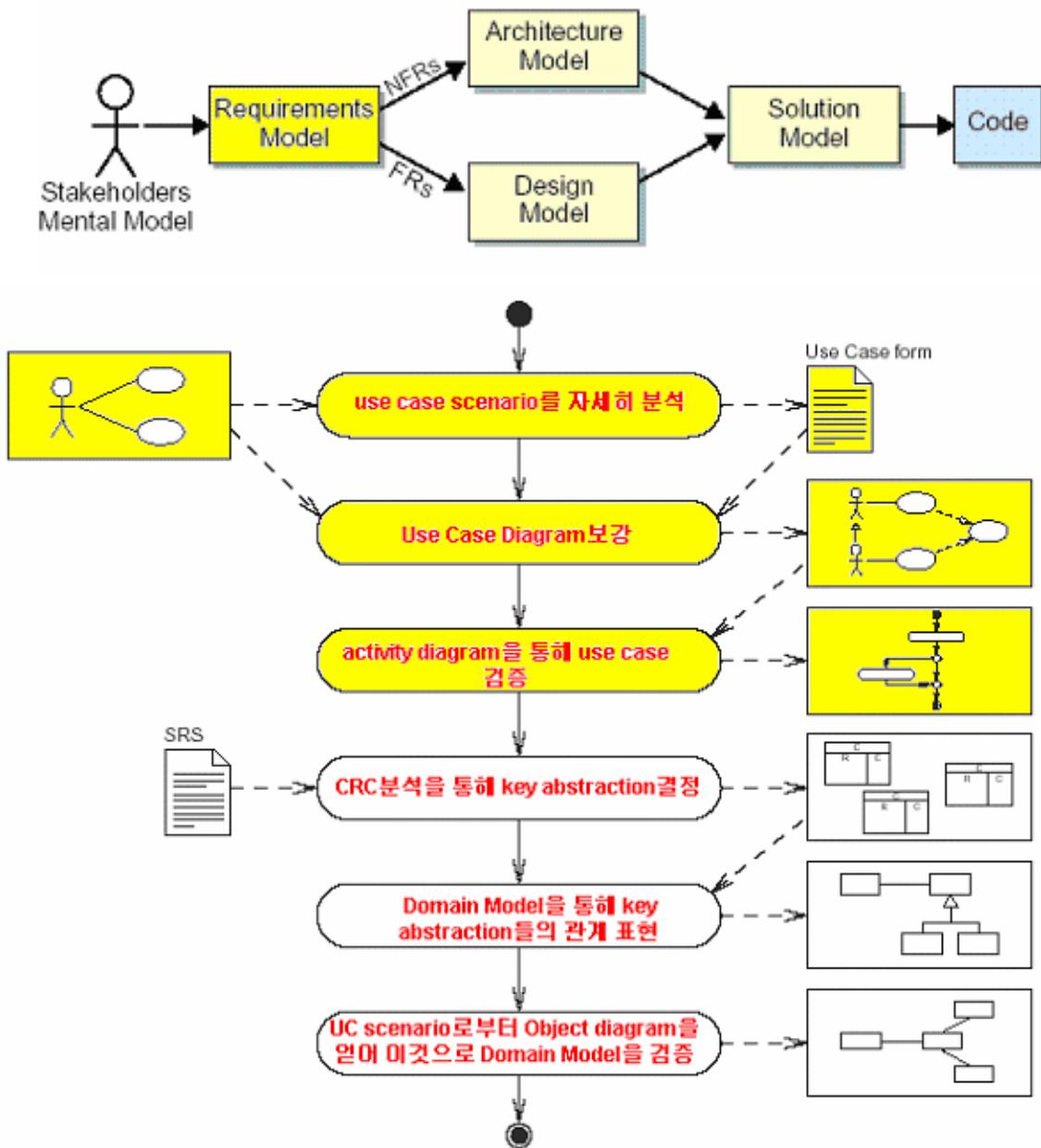
비즈니스 분석가 : “초기 유즈케이스 다이어그램은 너무 단순하고 광범위 해. 유즈케이스 시나리오로 보충을 했지만 그것은 작문으로 되어 있으니 이해하기 어렵고.... 좀 더 세분화된 그림으로 정교하게 표현해 볼 순 없을까?”



■ 학습하기 ■

참고하세요

n 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. 유즈케이스 분석

1) 유즈케이스 폼 작성

유즈케이스 폼은 하나의 유즈케이스와 시나리오의 상세한 분석결과를 기록하기 위한 형식입니다. 유즈케이스 폼의 구성요소는 다음과 같습니다.

— Table :: 유즈케이스 폼 구성요소

Form Element	Description
Use Case ID and Name	SRS 문서에 있는 유즈케이스의 코드와 이름
Description	유즈케이스의 목적을 한 두줄로 요약한다.
Actors	이 유즈케이스를 사용하는 것이 허용된 모든 actor
Priority	SRS 문서에 있는 유즈케이스의 우선순위 Essential, High-value, Follow-on
Risk	유즈케이스의 위험 요소 순위 High, Medium, Low
Pre- conditions and assumptions	유즈케이스가 호출되기 전의 상태나 가정
Trigger	유즈케이스가 호출되어야 하는 조건
Flow of Events	유즈케이스를 구성하고 있는 주요 작업의 흐름
Alternate Flows	이 유즈케이스에서 발생할 수 있는 2차적인 작업의 흐름
Post- conditions	이 유즈케이스가 완료된 뒤의 상태
Non- Functional Requirements	이 유즈케이스와 관련된 NFR, SRS 문서에 있는 NFR 이나 코드를 정리해도 된다.

방법론에 따라 유즈케이스 분석의 구체화 정도에 차이가 있습니다. 이번 장에서는 유즈케이스 분석 워크플로우를 구체적으로 다루겠습니다. 유즈케이스 분석을 세밀하게 하지 않는 경우에는 이벤트 플로우만을 다룹니다. 유즈케이스 폼에 대해 배우고 나면 어떤 구성 요소가 프로젝트에서 중요한지 선택할 수 있습니다.

유즈케이스 폼의 정보를 얻기 위한 과정은 다음과 같습니다.

1. **SRS** 문서에 있는 구체적인 내용을 적습니다.
2. 시나리오를 통해 선조건(**pre-condition**)을 결정합니다.
3. 시나리오를 통해 트리거(**trigger**)를 결정합니다.
4. **primary** 시나리오를 통해 이벤트 플로우를 결정합니다.
5. **Secondary** 시나리오를 통해 **alternate flow**를 결정합니다.
6. 후조건(**post-condition**)을 결정합니다.

각 단계의 내용을 하나씩 알아보겠습니다.

(1) Step 1 SRS 문서에 있는 구체적인 내용 기록

아래 표처럼 SRS 문서에 구체적으로 명시된 내용으로 품을 채웁니다.

■ Table :: Manage Reservation 유즈케이스의 내용

Use Case ID and Name	E1:Manager Reservation
Description	숙박시설에 새로운 예약을 작성하는 유즈케이스
Actor(s)	Primary : 예약 담당 직원 Secondary : 접수 담당 직원, 매니저, 오너
Priority	Essential
Non- Functional Requirements	E1- 102(성능) E1- 105(확장성) E1- 108(신뢰성)

Primary actor는 업무 역할(job role)을 위해 유즈케이스를 반드시 수행해야 하는 actor이고, secondary actor는 이 유즈케이스에서 정의한 일이 아닌 다른 일을 하지만 이 유즈케이스를 수행할 수 있는 actor를 말합니다.

(2) Step 2 시나리오를 통해 선조건(pre- condition) 결정

좋은 유즈케이스는 유즈케이스가 시작하기 전 시스템의 상태를 명시합니다. 이것이 선조건 (pre- condition)입니다.

예)

예약 담당 직원 **Medoca Sansumi**는 **HotelApp**의 메인 스크린을 띄워놓고, 고객의 전화를 기다리고 있습니다.

■ 표 :: Manage Reservation 유즈케이스의 선조건(pre- condition)

Pre- condition	예약 담당 직원은 전화를 기다리고 있습니다. HotelApp 의 메인화면을 띄워놓았습니다.
----------------	--

(3) Step 3 시나리오를 통해 트리거(trigger) 결정

시나리오가 시작하는 부분에는 특정 유즈케이스의 초기화 상태를 actor가 어떻게 알게 되는지 기술해야 합니다.

예)

뉴욕에 사는 고객 **Jane Google**로부터 호텔 예약을 원한다는 전화가 옵니다.

■ 표 :: Manage Reservation 유즈케이스의 트리거(trigger)

Trigger	예약을 원하는 고객에게서 전화가 옵니다.
----------------	------------------------

(4) Step 4 primary 시나리오를 통해 이벤트 플로우 결정

Primary 시나리오를 통해 이벤트 플로우를 결정합니다.

예)

Medoca는 도착 날짜, 머무는 기간, 방의 종류 등을 묻습니다. 고객이 원하는 방이 있는지 시스템에서 검색 결과를 알려주고, 사용 가능한 방을 모두 보여 줍니다.

n 표 :: Manager Reservation 유즈케이스의 이벤트 플로우

Flow of Events	<p>…</p> <p>3. 접수 직원은 검색 범위를 입력합니다.</p> <p>4. 사용 가능한 방을 찾기 위해 Room Schedule을 조회합니다.</p> <p>5. 시스템이 입력한 기간 내에 사용 가능한 방을 보여줍니다.</p> <p>6. 접수 직원이 방을 선택합니다.</p> <p>…</p>
-----------------------	--

시나리오는 매우 구체적이지만, 이벤트 플로우에 적을 때는 단순화해야 합니다.

(5) Step 5 Secondary 시나리오를 통해 alternate flow 결정

Secondary 시나리오를 통해 **alternate flow**를 결정합니다.

- **Primary** 시나리오와 각 **secondary** 시나리오 사이에 차이점을 분석합니다.
- **Alternate flow**는 **primary** 시나리오와 **secondary** 시나리오 사이에 차이가 있는 단계에 대한 표현입니다.

n 표 :: Manage Reservation 유즈케이스의 alternate flow

Alternate Flows	5단계에서, 이용 가능한 방이 없을 경우에는 예약 담당 직원이 고객에게 다른 종류의 방을 제시하거나 날짜를 변경해서 다시 검색하고 3단계로 되돌아갑니다.
------------------------	---

(6) Step 6 후조건(post-condition) 결정

좋은 유즈케이스 시나리오의 마지막 문단에는 유즈케이스가 완료된 후의 시스템 상태에 대해 기술합니다. 이것이 **후조건(post-condition)**입니다.

예)

Medoca는 **Jane**에게 시스템에서 부여한 예약 번호를 알려주고 예약 화면을 닫습니다.

▣ 표 :: Manage Reservation 유즈케이스의 후조건(post-condition)

Post-condition	예약 내용은 데이터베이스에 저장됩니다. 예약 화면은 닫히고 메인화면이 나타납니다.
-----------------------	---

이 과정까지 마치고 나면 유즈케이스 품이 완성됩니다.

2) 광범위한 유즈케이스 확장

요구 사항 수집 단계에서 만든 유즈케이스는 너무 광범위합니다. 하나의 유즈케이스가 여러 개의 워크플로우 기능을 포함하고 있습니다.

예를 들어, 예약은 여러 개의 서로 다른 워크플로우로 나타날 수 있기 때문에, **Manage Reservation** 유즈케이스는 너무 광범위한 표현입니다.

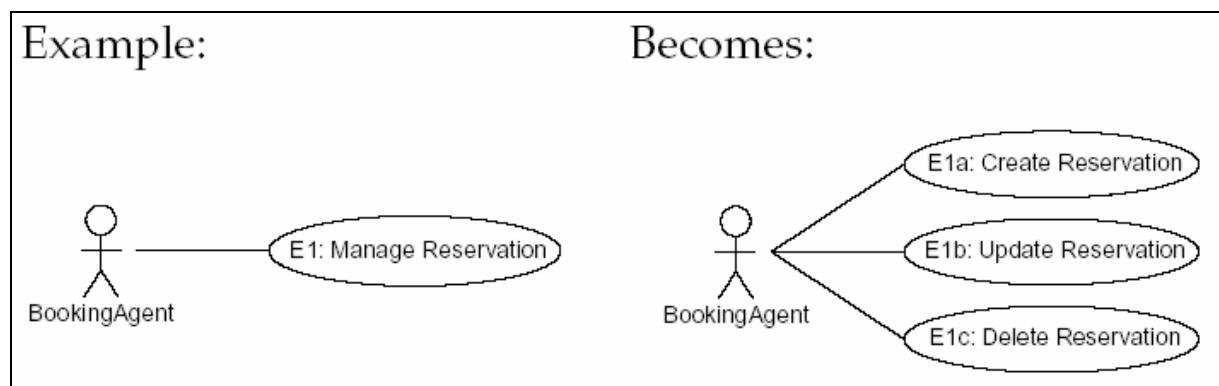
이런 경우 별개의 워크플로우로 구분한 새로운 유즈케이스를 사용하는 것이 좋습니다.

예를 들어, **Manage Reservation** 유즈케이스는 예약을 새로 하는 경우, 예약을 변경하는 경우, 예약을 취소하는 경우 등 세 가지 경우로 구분할 수 있습니다.

다음 그림은 **Manage Reservation** 유즈케이스가 확장되는 것을 보여줍니다.

■ 이미지

■ Expanding High-level Use Case



참고하세요

n 유즈케이스의 확장을 위한 원리

일반적으로 **entity**를 관리하는 데는 생성, 검색, 수정, 삭제(소위 **CRUD- Create, Retrieve, Update, Delete-**라고 합니다) 기능이 필요합니다. ‘예약’이라는 엔티티를 **CRUD**하는 유즈케이스를 만들다고 생각하면 쉽게 확장 될 수 있습니다.

다른 유즈케이스의 경우에도 유즈케이스 시나리오를 분석하고 분기문을 찾아봐야 합니다. 또 어떤 시나리오는 시작점이 다를 수도 있습니다. 이런 시나리오는 다른 유즈케이스로 표현해야 합니다.

유즈케이스를 지나치게 구체화하지 않도록 주의해야 합니다. 왜냐하면 유즈케이스를 상세하게 표현하는 것은 기능 분리가 일어날 수 있고 분석 마비(**analysis paralysis**)를 초래할 수 있기 때문입니다.

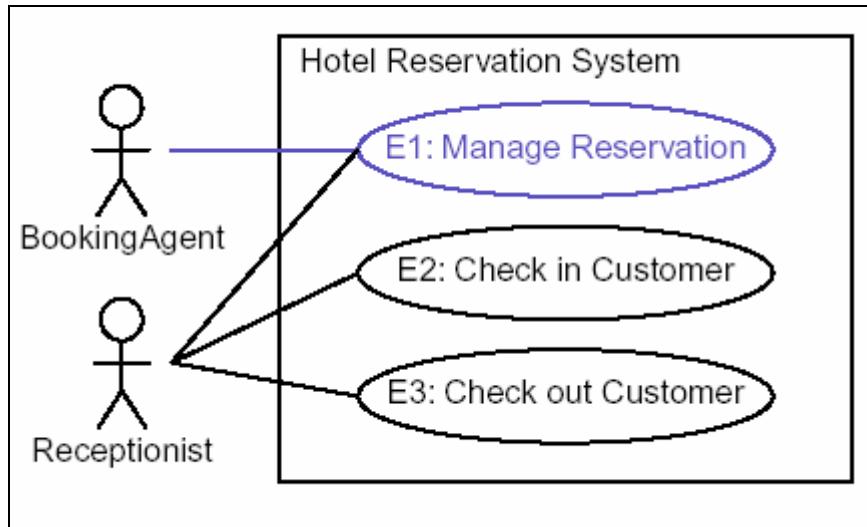
유즈케이스를 세분화 할 때, 유즈케이스의 연관관계도 증가하게 됩니다.

유즈케이스가 확장되는 것은 그만큼 상세화된다는 것을 말합니다.

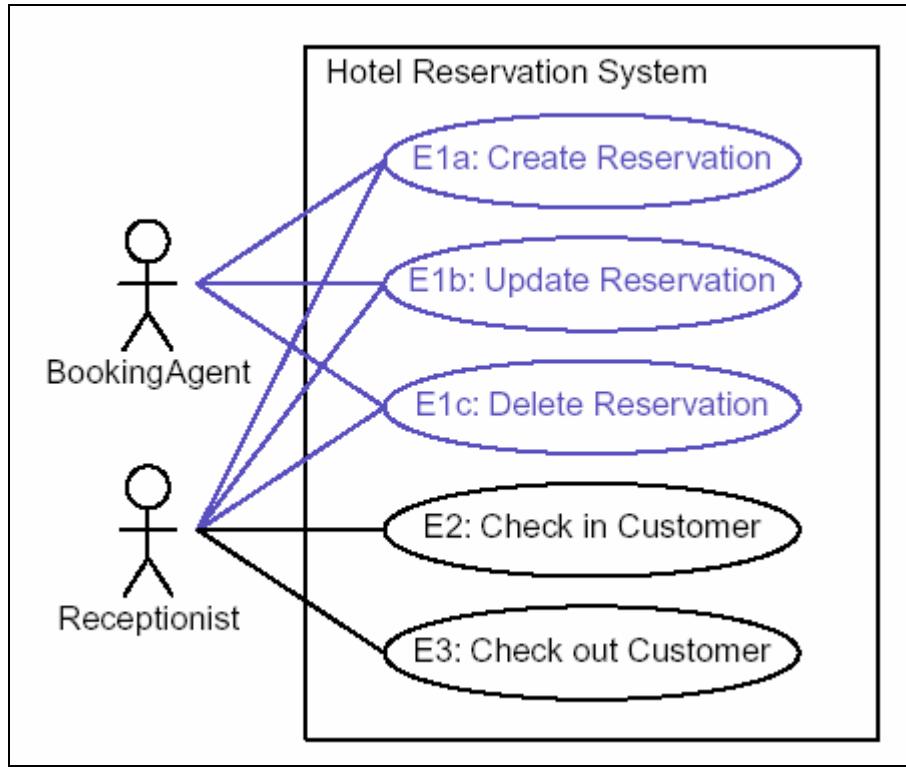
다음 그림은 초기 유즈케이스 다이어그램이 상세화(혹은 확장) 된 후의 차이를 잘 보여줍니다.

이미지

■ 상세화하기 전의 유즈케이스



■ 상세화 한 후의 유즈케이스



두 개의 유즈케이스 다이어그램을 보면 상세화 한 후의 유즈케이스가 이전보다 복잡해진 것을 알 수 있습니다. **Manage Reservation**이라는 유즈케이스를 **Create Reservation**, **Update Reservation**, **Delete Reservation**으로 세분화 한 다음, 두 명의 **actor**와의 연관관계를 모두 표시해 줍니다. 유즈케이스 다이어그램에서는 연관관계가 없는 유즈케이스나 **actor**는 의미가 없기 때문에, 늘어난 유즈케이스 만큼 또는 늘어난 **actor**의 수만큼 연관관계는 증가할 수 밖에 없습니다.

결국, 유즈케이스 상세화 작업은 유즈케이스를 복잡해 보이게 할 수 있습니다.

3) 상속 패턴 분석

유즈케이스 다이어그램에서의 **상속은 actor와 유즈케이스에 대해 발생할 수 있습니다.**

- **Actor**는 부모 **actor**로부터 모든 유즈케이스 연관 관계를 상속 받을 수 있습니다.
- 유즈케이스는 여러 개의 특화된 유즈케이스로부터 상속 받을 수 있습니다.

(1) Actor 상속

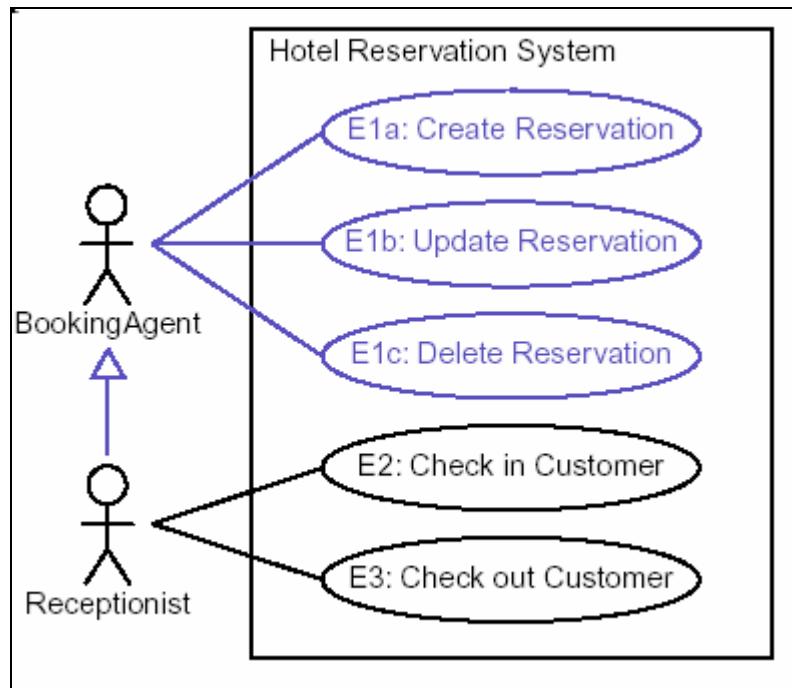
Actor는 부모 **actor**로부터 모든 유즈케이스 연관 관계를 상속 받을 수 있습니다.

예를 들어, 접수 담당 직원은 예약 담당 직원의 모든 일을 할 수 있습니다. 따라서, 이 관계를 나타내기 위해 **actor** 상속을 사용할 수 있습니다.

다음 그림에서 **BookingAgent**와 **Receptionist**의 관계를 주목하세요.

[이미지]

n Actor의 상속



Receptionist는 **BookingAgent**가 사용하는 유즈케이스를 모두 사용하고 그 외에 **Check in/Check out** 유즈케이스도 사용합니다. 즉, **Receptionist**는 **BookingAgent**의 모든 기능을 포함하고 거기에 추가적인 기능을 더 갖고 있습니다. 이런 경우 **Receptionist**는 **BookingAgent**를 상속할 수 있습니다.

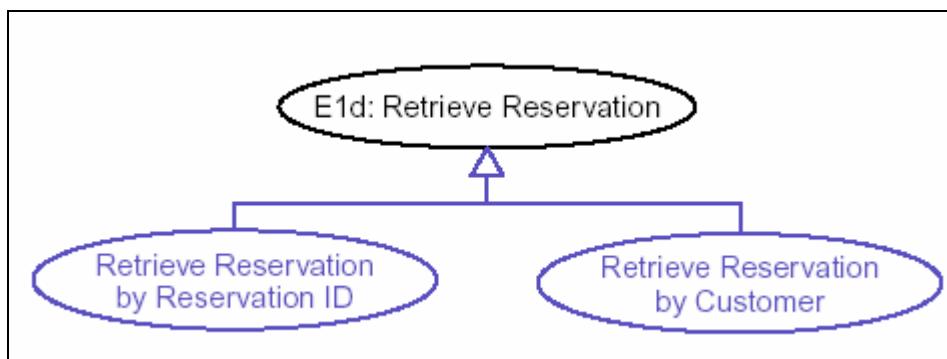
(2) Use case 특화(specialization)

유즈케이스는 여러 개의 특화된 유즈케이스로부터 상속 받을 수 있습니다. 예를 들어, **Retrieve a Reservation** 유즈케이스의 경우 예약 번호로 검색할 수도 있고 고객 이름으로 검색할 수도 있습니다.

다음 그림이 유즈케이스의 특화를 잘 보여줍니다.

[이미지]

n 유즈케이스의 특화



예를 들어, **Retrieve a Reservation** 유즈케이스의 경우 예약 번호로 검색할 수도 있고 고객 이

름으로 검색할 수도 있습니다.

보통

n 유즈케이스 특화에 대한 힌트

유즈케이스의 특화는 대개 유즈케이스 시나리오의 변경으로 파악할 수 있습니다. 유즈케이스를 수행하기 위한 기능적 전략의 차이에 따라 시나리오가 변경됩니다. 즉, 하나의 유즈케이스가 워크플로우 동안 다른 상황을 야기하게 된다면 이것을 초기 유즈케이스로부터 ‘특화’시켜 표현할 수 있게 됩니다.

(3) 상속의 의존 관계 분석

유즈케이스는 두 가지 방법으로 다른 유즈케이스에 의존할 수 있습니다.

i. 하나의 유즈케이스(a)는 다른 유즈케이스(b)를 포함할 수 있습니다.

이것은 (a)유즈케이스는 (b)유즈케이스의 기능을 사용하고, 포함된 유즈케이스를 실행하는 것을 의미합니다.

ii. 하나의 유즈케이스(a)는 다른 유즈케이스(b)를 확장할 수 있습니다.

이것은 (a)유즈케이스는 (경우에 따라) (b)유즈케이스의 기능을 사용하고, (b)유즈케이스를 확장하는 것을 의미합니다.

전자를 **include**라고 하고 후자를 **extend**라고 합니다.

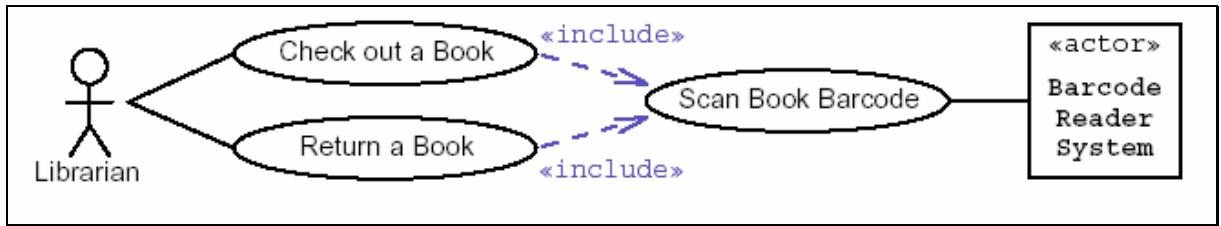
① <<include>> 의존 관계

<<include>> 의존 관계는 여러 개의 유즈케이스에 대한 공통적인 기능을 파악하는 것이 가능합니다.

이 의존 관계는 의존 관계 화살표와 <<include>> 표시를 사용해서 나타냅니다.

이미지

■ <<include>> 의존관계



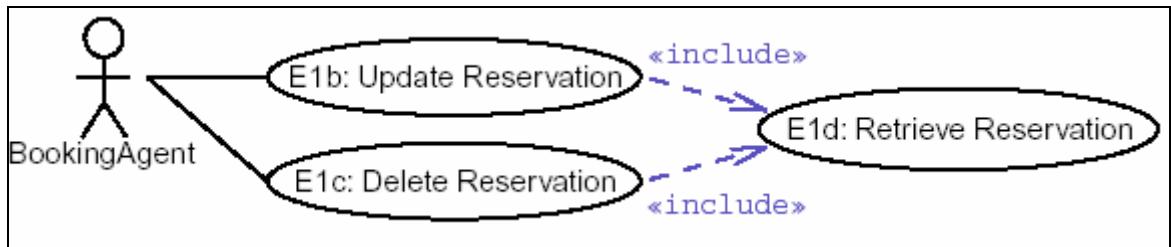
예를 들어, 도서관 사서가 책을 대여할 때나 반납 받을 때는 바코드 리더 시스템을 통해 책에 있는 바코드를 찍어야 합니다. 이것을 유즈케이스 다이어그램으로 나타낸 것이 위 그림인데, **Librarian** 는 **Check out a Book** 유즈케이스와 **Return a Book** 유즈케이스에 대한 **actor**입니다. 그리고 그 두 유즈케이스는 **Scan Book Barcode**라는 유즈케이스의 기능을 사용하고 있다는 것을 <<include>> 의존관계로 표현했습니다. **Scan Book Barcode** 유즈케이

스에 대한 **actor**는 외부 시스템인 **Barcode Reader System**입니다.

다음은 호텔 예약 시스템의 경우입니다.

이미지

■ 호텔 예약 시스템에서의 <<include>> 의존 관계

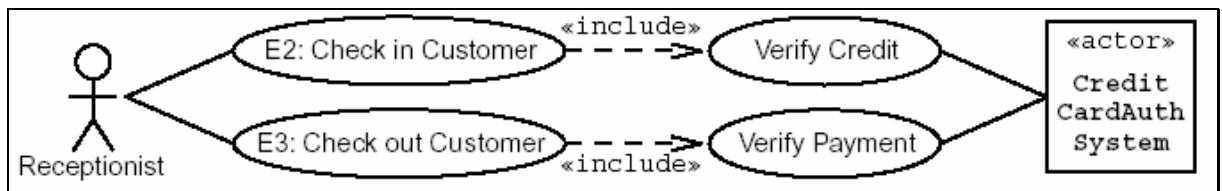


예약을 수정하고 삭제하는 유즈케이스에서는 데이터베이스에 예약을 검색하는 것이 선행되어야 합니다. “**Retrieve Reservation**”이란 이름의 유즈케이스는 **Update Reservation** 유즈케이스, **Delete Reservation** 유즈케이스와 <<include>> 의존 관계에 있게 됩니다.

<<include>>를 사용해서 유즈케이스를 상세화 하는 또 다른 경우는 외부시스템과 상호작용을 할 때입니다. 예를 들어, 호텔 예약 시스템에서는 신용카드 인증 시스템과 접속해야 합니다. 이 시스템은 고객이 체크인을 할 때와 체크아웃을 할 때 필요합니다.

이미지

■ 호텔 예약 시스템에서의 또 다른 <<include>> 의존 관계



참고하세요

▣ **include** 쉽게 이해하기

어떤 유즈케이스를 실행 하기 전에 다른 유즈케이스가 매번 선행되어야 한다면 이 들은 **include** 관계가 있는 것입니다.

② <<extend>> 의존 관계

<<extend>> 의존 관계를 통해서 주요 흐름은 아니지만 대안 시나리오에 있는 시스템의 기능을 파악할 수 있습니다.

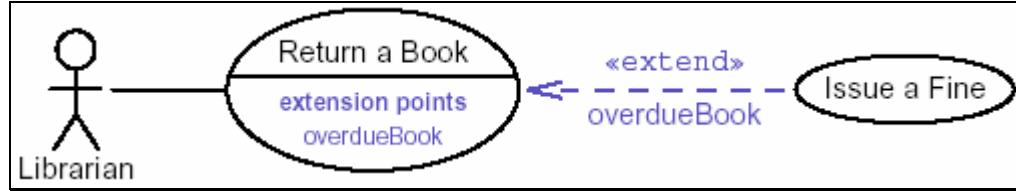
이 의존 관계는 의존 관계 화살표와 <<extend>>, 추가적으로 “**extension point**” 구문의 표시를 사용해서 나타냅니다. “**Extension point**”는 주요 유즈케이스에서 확장 유즈케이스가

필요한 경우를 나타냅니다.

다음 그림으로 예를 보겠습니다.

[이미지]

■ <<extend>> 의존 관계

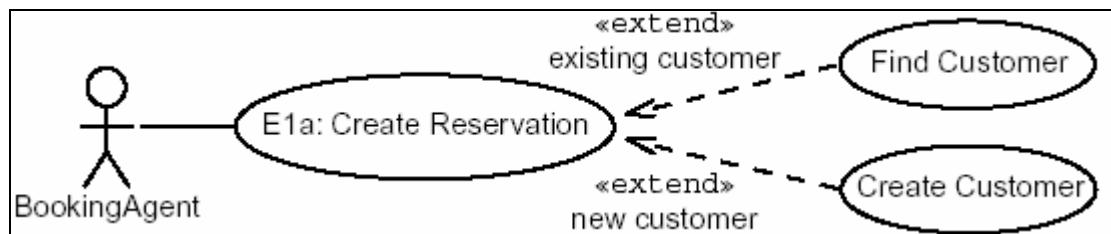


도서관의 사서는 책을 반납 받을 때 대여 기일이 지나서 책을 반납한 경우 벌금을 부과합니다. 이것을 유즈케이스 다이어그램으로 표현하면, **Librarian**이 **Return a Book**이라는 유즈케이스를 실행할 때 대여 기일이 지난 경우 벌금을 부과하는 **Issue a Fine** 유즈케이스를 실행하도록 합니다. **Return a Book** 유즈케이스에 **extension points**로 조건을 주고, 그 조건을 만족할 경우 <<extend>> 의존관계에 있는 **Issue a Fine** 유즈케이스가 실행됩니다.

다음 예는, 호텔 예약을 하려 할 때, 예약은 반드시 고객과 연관이 있고, 고객은 기존 고객일 수도 있고 새 고객일 수도 있는 경우입니다.

[이미지]

■ 호텔 예약 시스템에서의 <<extend>> 의존 관계

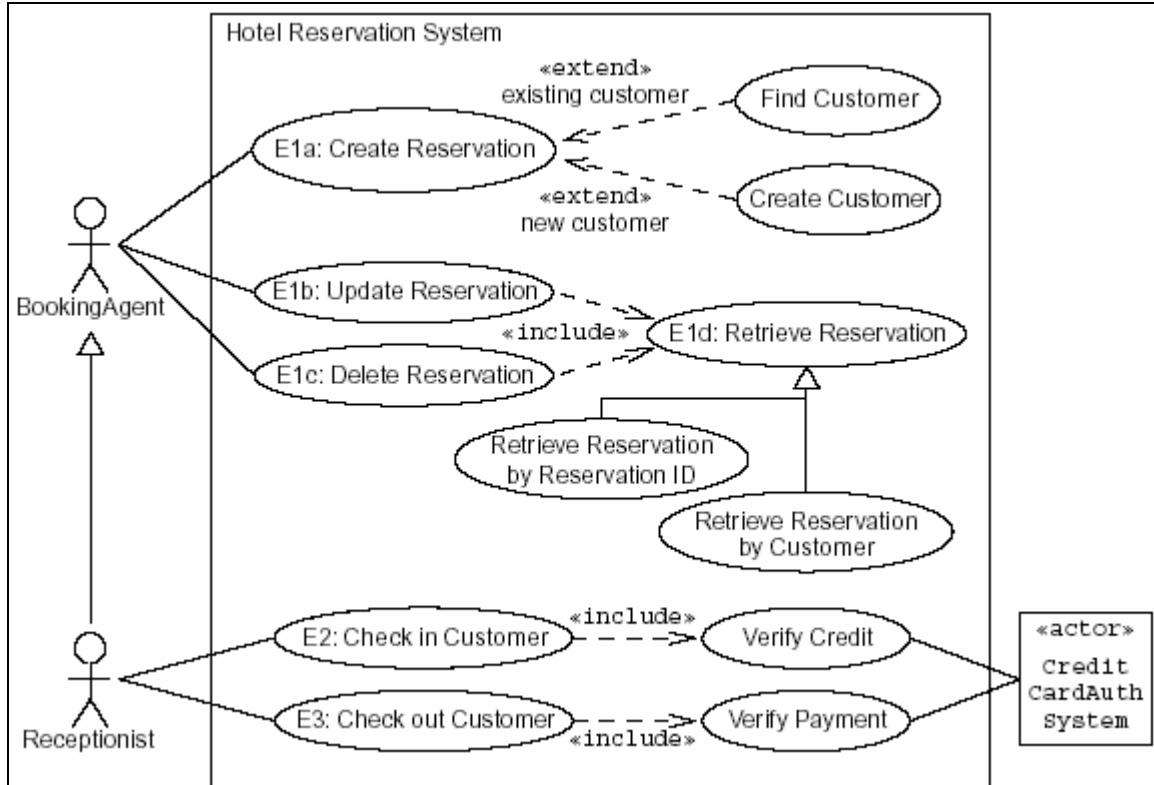


BookingAgent가 **Create Reservation** 유즈케이스를 실행할 때 기존 고객의 경우 고객의 정보를 검색하는 **Find Customer**를 실행하고 새 고객일 경우는 **Create Customer**를 실행합니다.

다음 그림은 호텔 예약 시스템의 다양한 유즈케이스 관계를 종합적으로 보여주는 것입니다.

[이미지]

■ 호텔 예약 시스템의 여러 가지 의존 관계



지금까지 살펴본 내용을 토대로 호텔 예약 시스템에 대한 유즈케이스 다이어그램을 작성하면 이와 같이 나타낼 수 있습니다. **Hotel Reservation System**이라고 쓰여 있는 시스템 경계 안에 각종 유즈케이스들이 들어가 있고, **actor**는 **BookingAgent**와 **Receptionist**, 외부 시스템인 **Credit Card Auth System**이 있습니다. **Receptionist**와 **BookingAgent**는 상속관계이고, **Receptionist**는 **BookingAgent**의 모든 기능을 포함하고 있습니다. 각각의 유즈케이스들도 <>extend>>와 <>include>>관계에 있음을 알 수 있고, 상속관계에 있는 유즈케이스도 있습니다.

참고하세요

▣ extend 쉽게 이해하기

어떤 유즈케이스를 실행 하기 전에 조건에 따라 다른 유즈케이스가 한번 수행되고 이 유즈케이스가 매번 수행되지는 않는다면 이 둘은 **extend** 관계가 있는 것입니다.

2. Activity Diagram을 통한 유즈케이스의 검증

1) 개요

유즈케이스는 소프트웨어 프로젝트의 성공에 필수적이기 때문에, 유즈케이스의 기능에 대해 **stakeholder**와 여러분의 생각이 같은지 확인해야 합니다. 유즈케이스의 기능에 대한 다른 다이어그램을 통해 유즈케이스를 검증할 수 있습니다.

- **Activity** 다이어그램으로 유즈케이스의 이벤트 플로우를 표현할 수 있습니다.
- **Stakeholder**와 함께 **Activity** 다이어그램을 보면서 유즈케이스를 검증할 수 있습니다.

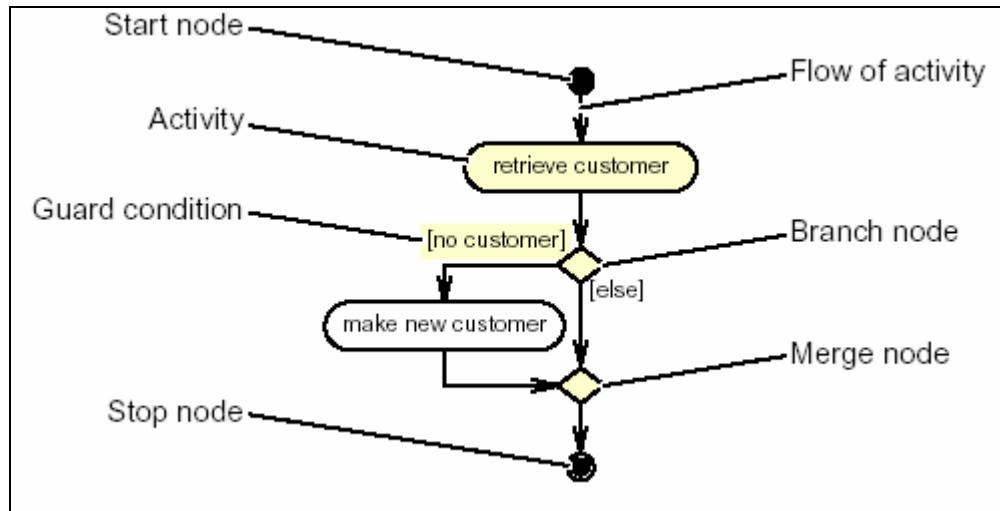
다음은 유즈케이스를 **Activity** 다이어그램으로 변환하는 방법입니다.

2) activity 다이어그램 요소 파악

Activity 다이어그램은 **activity**를 확실한 시작점과 끝점 사이에 시간 순으로 표현하는 것입니다.

이미지

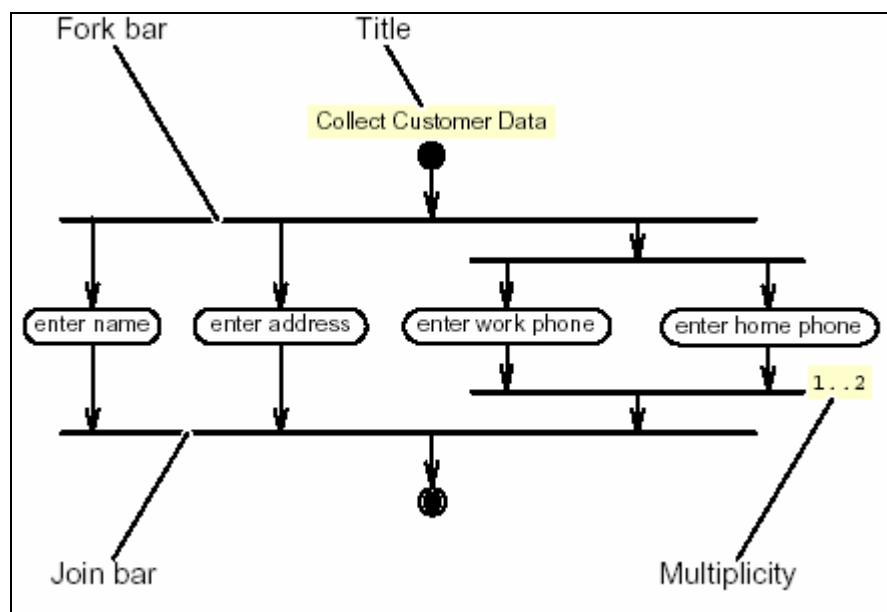
■ Activity Diagram



Activity 다이어그램은 순서대로 일어나야 하는 작업 중 동시에 일어나는 작업도 표현할 수 있습니다. 동시에 일어나는 작업은 분기 막대(fork bar)와 병합 막대(join bar)로 나타냅니다. 이런 작업들은 실제로 동시에 수행되거나 특별한 순서 없이 수행될 수 있습니다.

이미지

■ 병행 Activities에 대한 Activity Diagram



(1) Activities

Activity는 시스템의 프로세스나 actor의 action입니다.

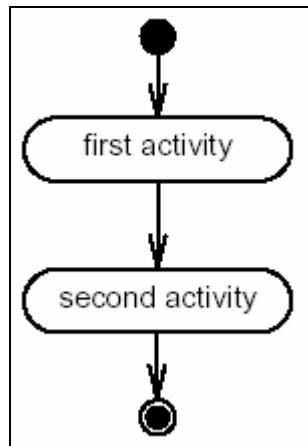
- Activity는 자연어로 쓸 수 있습니다.
예 : 고객 정보 가져오기
- Activity는 pseudo-code로 쓸 수 있습니다.
예 : cust = retrieve customer
- Activity는 프로그래밍 언어로 쓸 수 있습니다.
예 : cust = customerSvc.getCustomer(custID);

(2) 제어 흐름(Flow of Control)

Activity 다이어그램은 Start 노드로 시작해서 Stop 노드로 끝납니다. 제어 흐름은 activity 사이에 화살표로 나타냅니다.

[이미지]

■ 제어 흐름



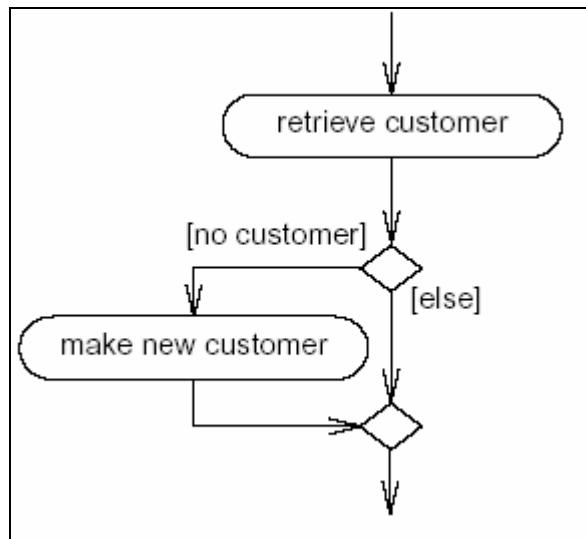
(3) 분기(Branching)

분기와 병합 노드는 activity의 조건적 흐름을 나타냅니다.

- 분기 노드는 선택조건을 가리키기 위한 Boolean 표현에 따라 나가는 부분이 두개입니다.
- 병합 노드는 분기문을 정리합니다.

[이미지]

■ Branch and Merge Nodes



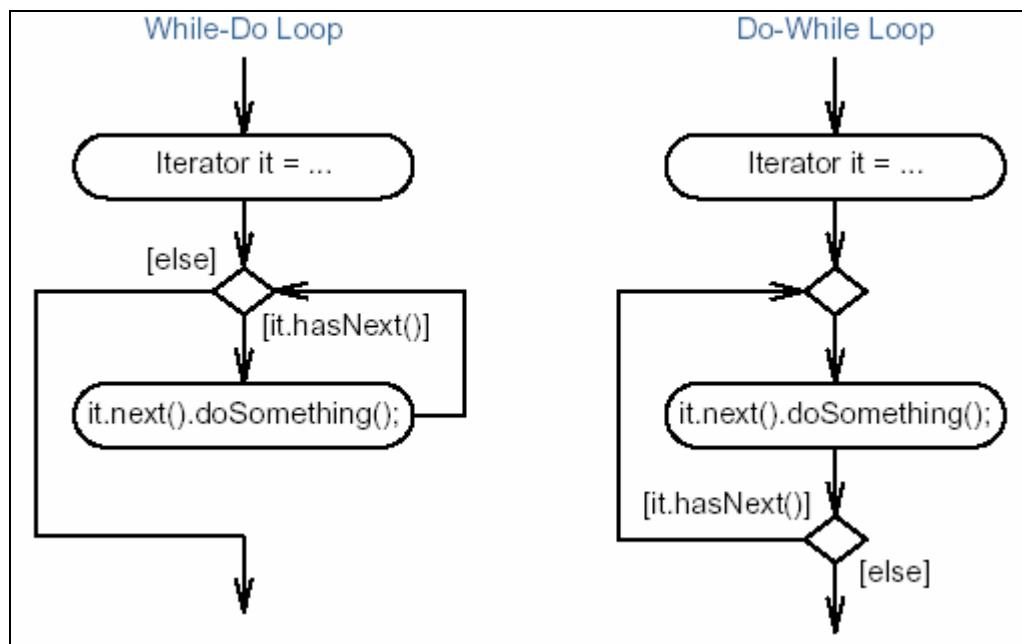
고객 정보를 검색할 때 존재하지 않는 고객일 경우 새로 고객을 추가하고 그렇지 않고 존재하는 고객일 경우는 다음 작업을 진행합니다.

(4) 반복(Iteration)

분기문을 사용해서 반복문을 만들 수 있습니다.

[이미지]

■ Iteration Loop



while- do 루프는 조건을 확인하고 나서 루프의 바디를 수행하고, **do- while** 루프는 조건을 확인하기 전에 최소한 한번은 루프의 바디를 실행합니다.

두 개의 **Activity** 다이어그램의 경우 분기점의 화살표 방향을 주의해서 보기 바랍니다.

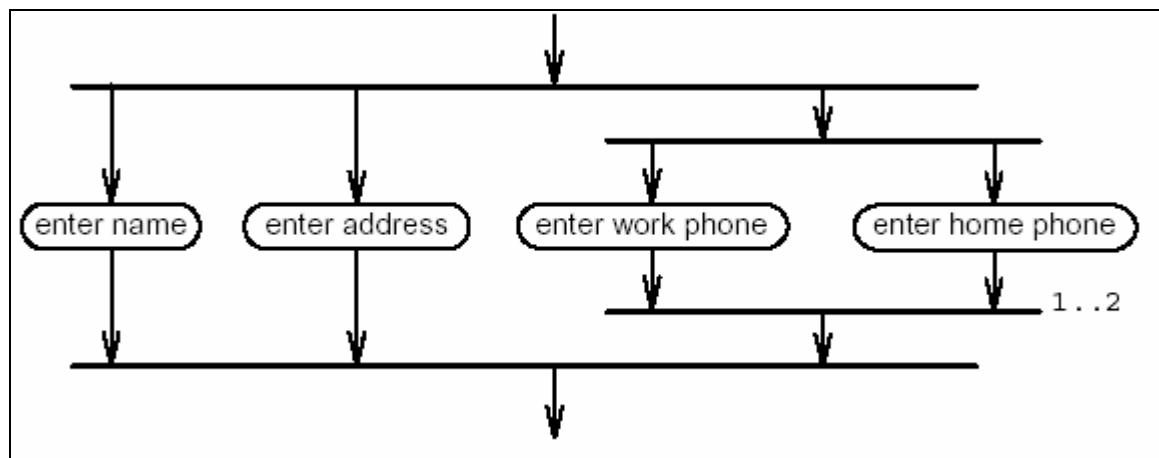
While- do 루프는 분기점에서 나가는 화살표와 들어오는 화살표가 하나씩 있고, **do- while** 루프는 분기점에 들어오는 화살표만 하나 있습니다. **while- do** 루프는 조건을 확인하고 나서 루프의 바디를 수행하고, **do- while** 루프는 조건을 확인하기 전에 최소한 한번은 루프의 바디를 실행하는 것을 표현한 것입니다.

(5) 병행 제어 흐름

병행 제어 흐름을 가리키기 위해 분기 막대와 병합 막대를 사용합니다.

이미지

■ 병행 제어 흐름



- 분기 막대와 병합 막대는 연결된 **activity**나 동등한 **activity**를 나타낼 수도 있고, 특별한 순서를 따르지 않는 **activity**들을 나타낼 수도 있습니다.
- 여러 개의 화살표는 처리되어야 하는 동등한 **activity**의 개수를 의미합니다. 숫자로 표시하는 다중성 표시는 실행해야 하는 **activity**의 수입니다. **1..2**로 쓰여진 경우는 최소한 **1개**의 **activity**는 실행되어야 하고 최대 **2개**의 **activity**가 실행된다는 것을 의미합니다.

참고하세요

■ 다중성 표시(Multiplicity Indicators)

1	한 개
0..*	0 또는 그 이상
4..7	특정 범위(4~7)
4..7,9	조합(4~7, 9)

3) 유즈케이스에 대한 activity 다이어그램 작성

유즈케이스에 대한 activity 다이어그램을 작성하기 위해 유즈케이스 폼에 있는 이벤트 플로우를 분석해야 합니다.

- Activity 파악
- 분기와 루프 파악
- 병행activity 파악
- activity 다이어그램 작성

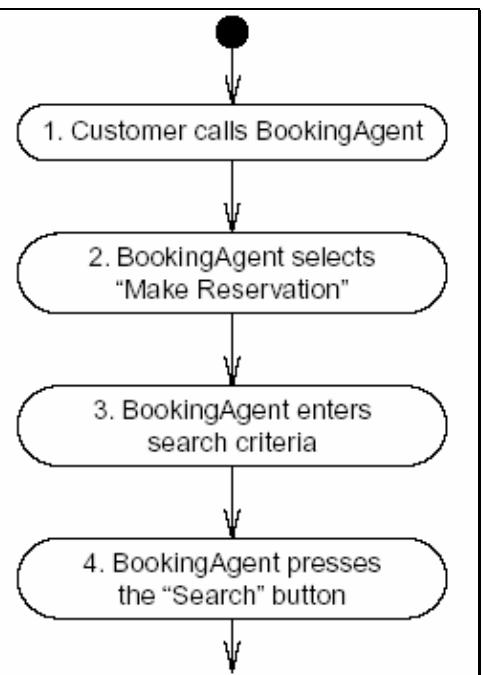
(1) Use Case Activity

유즈케이스의 이벤트 플로우에 있는 activity를 Activity 다이어그램의 activity 노드로 표현합니다.

이미지

■ 이벤트 플로우

1. Customer calls BookingAgent
2. BookingAgent select "Make Reservation" icon
3. BookingAgent enters search criteria
 - 3.1 BookingAgent enters arrival and departure dates
 - 3.2 BookingAgent enters type of room
4. BookingAgent presses the "Search" button
- ...
11. BookingAgent enters customer name
12. BookingAgent presses the "Search" button
13. If a customer match is not found
 - 13.1 BookingAgent enters address info
 - 13.2 BookingAgent enters phone info
 - 13.3 BookingAgent presses "Add New Customer"
14. Else
 - 14.1 The system displays match list
 - 14.2 BookingAgent selects the correct customer
 - 14.3 The System populates the GUI with customer info
- ...
21. The System saves the reservation and displays Res-vID

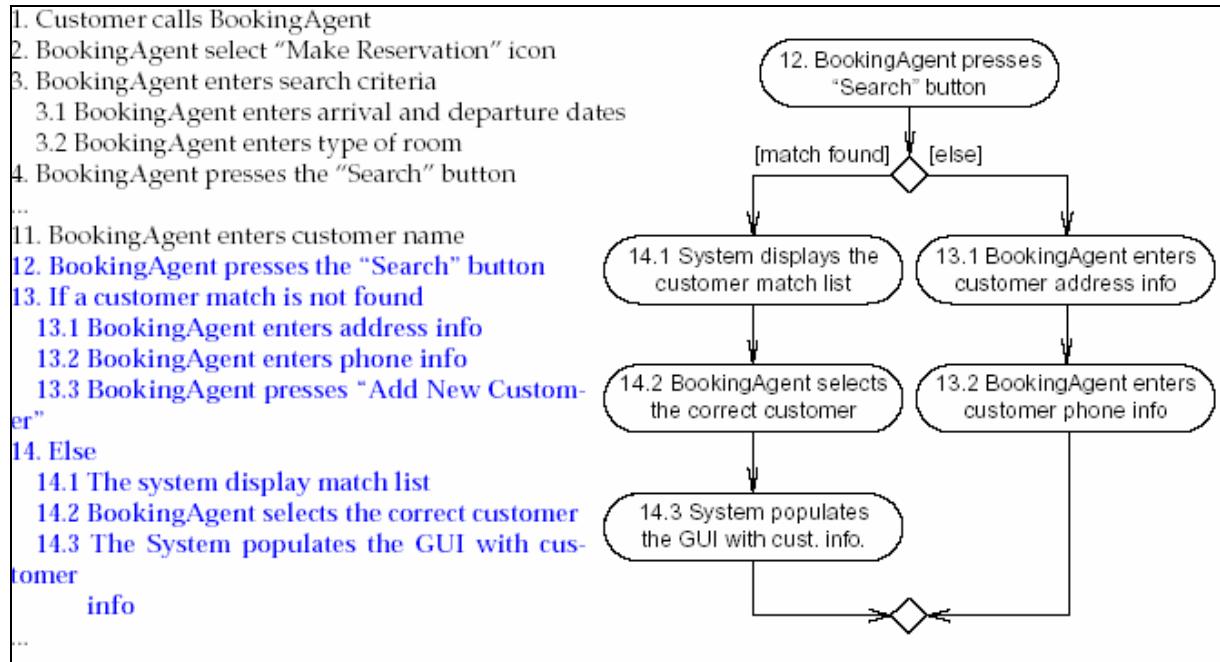


(2) Branching

유즈케이스에 대한 이벤트 플로우에서는 간혹 조건 분기가 발생합니다. Activity 다이어그램에서 조건문은 분기와 병합 표시로 나타냅니다.

이미지

■ Branching in the Flow of Events

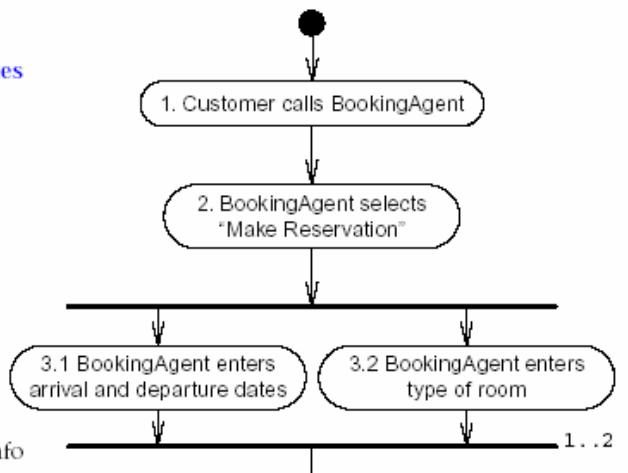


(3) 병행 흐름

유즈케이스에 대한 이벤트 플로우에서는 간혹 동시에 진행해야 하는 작업이 있습니다. 이런 경우는 **Activity** 다이어그램에서 분기 막대와 병합 막대로 나타냅니다.

■ 병행 이벤트 플로우

1. Customer calls BookingAgent
 2. BookingAgent select "Make Reservation" icon
 3. BookingAgent enters search criteria
3.1 BookingAgent enters arrival and departure dates
3.2 BookingAgent enters type of room
 4. BookingAgent presses the "Search" button
 ...
 11. BookingAgent enters customer name
 12. BookingAgent presses the "Search" button
 13. If a customer match is not found
 13.1 BookingAgent enters address info
 13.2 BookingAgent enters phone info
 13.3 BookingAgent presses "Add New Customer"
 14. Else
 14.1 The system display match list
 14.2 BookingAgent selects the correct customer
 14.3 The System populates the GUI with customer info
 ...
 21. The System saves the reservation and displays Res-
 vID



과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원3 : 기능적 요구사항 분석

2 모듈 : 사례연구

담당강사 : 전은수

■ 생각해봅시다 ■

지금까지 비즈니스 오너와의 인터뷰를 통해 제안서를 작성하고 시스템 요구 사항을 수집·분석하여 **SRS** 문서를 작성하는 단계까지 배웠습니다. 그리고 제안서의 내용을 다이어그램으로 표현한 유즈케이스 다이어그램에 대해서도 다뤘습니다. 이번 장에는 지금까지 배운 것을 기반으로 비디오 대여 시스템 사례를 살펴보겠습니다. 이론으로 배운 내용들을 실제 프로젝트에서 어떻게 적용해야 할 지 잘 생각해보시기 바랍니다.

1. 비디오 대여 시스템의 유즈케이스 다이어그램 작성

1) 제안서

제안서는 비즈니스 오너와의 인터뷰를 토대로 만들어지며, 시스템의 주요 목적을 정의하는 문서입니다. 제안서는 가능한 간결하고 명확하게 작성하는 것이 좋습니다.

제안서에는 다음과 같은 내용이 들어갑니다.

- **Introduction (problem statement 포함)**
- **Business opportunity**
- **Proposed solution(FRs과 NFRs 포함)**
- **Risks**
- **Constraints**

(1) Introduction

21세기는 멀티미디어 시대입니다. 여가 문화도 변화되어, 여가 시간 활용에서 멀티미디어의 활용은 높은 비율을 차지하고 있습니다. 그리고 기술의 발전에 힘입어 다양한 미디어의 형태가 나타나고 있습니다. 이에 따라, 다양한 컨텐츠 제작과 더불어 시장의 폭이 점점 증대되고 있습니다.

(2) Business opportunity

여러 미디어 형태 중 대여점에서 빌려보는 비디오 테이프는 가장 낡은 방법 가운데 하나로, 비디오를 즐겨보는 20, 30대는 이미 인터넷이나 DVD를 통하여 영화를 보거나 극장 문화에 익숙해져 있기 때문에 비디오 테이프 대여 시장은 점점 줄어들고만 있습니다.

점점 커져가는 미디어 시장에서 비디오 대여점 매출이 점점 줄어드는 모순을 극복 해야 합니다. 그러기 위해, 현재의 전통적인 대여 체계와 인터넷 마케팅의 신기술을 접목하여 해결책을 제시합니다. 이 시스템은 보편화 되어 있는 인터넷 통신망을 이용해 클릭만으로 비디오를 대여 할 수 있는 시스템입니다.

(3) Proposed solution

① Function Requirements

- Essential

- .. 웹과 모바일을 통한 **DVD**대여와 구매가 가능한 시스템입니다.
- .. 신청한 **DVD**에 대한 배송정보를 보여줍니다.
- .. 회원이 구매하고 대여한 정보에 대해서 확인할 수 있는 페이지가 있습니다.
- .. 가맹점 가입 및 탈퇴가 가능해야 합니다.

- High-value

- .. 회원 가입 및 탈퇴가 가능해야 합니다.
- .. **DVD**에 대한 정보를 검색할 수 있어야 합니다.

- Follow-on

- .. 방문자와 기존회원은 공동구매를 통해 원하는 **DVD**를 싸게 구입할 수 있습니다.
- .. 가맹점에서는 다량구매를 통해 많은 양의 **DVD**를 구입할 수 있습니다.

② Non-Function Requirements

- Performance

- .. 한 달에 약 **300**건의 대여를 지원해야 합니다.
- .. 한 달에 약 **200**건의 판매를 지원해야 합니다.
- .. **DVD** 검색 시 응답시간은 **5초** 이내여야 합니다.

2) 요구 사항 수집 및 SRS 문서 작성

SRS 문서는 시스템 요구사항을 종합해서 기록한 문서로, 비즈니스 오너와 다른 **stakeholder**도 사용할 수 있기 때문에, 비전문가도 이해할 수 있게 작성해야 합니다.

SRS 문서는 다음과 같은 내용으로 구성되어 있습니다.

- 도입
- 제약 사항과 가설
- 위험 요소
- 기능적 요구 사항
- 비기능적 요구 사항
- 프로젝트 용어집
- 초기 유즈케이스 다이어그램

SRS 문서는 초점을 잊지 않고 자세하게 기술해야 합니다.

(1) 도입

n 개발 시스템의 정의

프로젝트 기간동안 개발 할 시스템은 사용자가 인터넷으로 빌려보고자 하는 **DVD** 및 미디어 컨텐츠(비디오 등)를 검색하여 배달 신청을 하면 원하는 배달 장소로 빠른 시간 내에 배달할 수 있는 시스템입니다. 시스템명은 **MDS(Media Delivery System)**로 정의 합니다.

n 개발 시스템의 목적

21 세기는 멀티미디어 시대입니다. 이미 생활에서 여가 시간을 활용에서 멀티미디어를 활용하는 것은 높은 비율을 차지하고 있습니다. 기술의 발전에 힘입어 다양한 미디어의 형태가 나타나고 있습니다. 다양한 컨텐츠 제작과 더불어 시장의 폭이 점점 증대되고 있습니다.

여러 미디어 형태 중 대여점에서 빌려보는 비디오 테이프는 가장 낡은 방법 가운데 하나입니다. 비디오를 즐겨보는 **20, 30** 대는 이미 인터넷을 통하여 영화를 보는데 익숙해져 있고 비디오 테이프 대여 시장은 점점 줄어들고만 있습니다.

점점 커져 가는 미디어 시장에서 점점 줄어드는 매출의 모순을 극복 해야만 합니다. 여기에 전통적인 대여시장과 인터넷 마케팅의 신기술을 접목하여 해결책을 제시하려 합니다. 바로 보편화 돼있는 인터넷 통신망을 이용해 클릭만으로 대여 할 수 있는 시스템입니다.

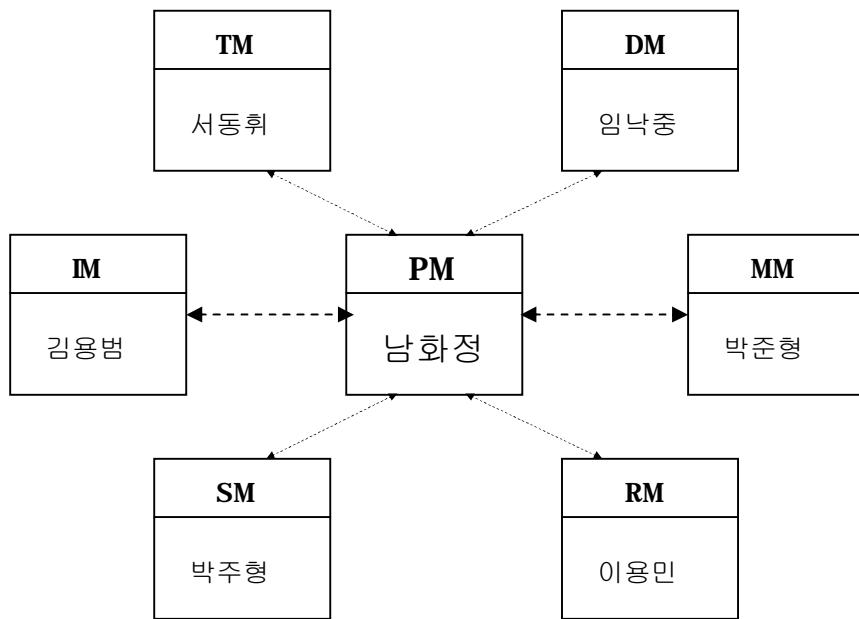
n 동종 업체 운영 현황

동종 업체 중 **DVDMarket**에서 운영하고 있는 <http://www.dvdmarket.co.kr/>를 운영방식을 중심으로 **BenchMarking**하고 이를 참고합니다.

n Job role에 따른 MDS Stakeholders

Job Role	주요 Stakeholder	2차 Stake holders
오너	전은수 (프랜차이즈사 대표)	
관리자	김세룡 (가맹점 관리자)	이보람(서울 본점) 김산(인천점) 홍나리(수원점)
개발	e- pro 팀	
고객	홍길동(예를 들어)	

개발팀(e-pro) 조직 구성도



Position	이름	업무내용
PM (Project Manager)	남화정	프로젝트 진행의 모든 제반사항을 관리하고 결정
TM(Technical Manager)	서동휘	프로젝트 전과 진행 중 각 팀원의 프로그램 처리 시 의문사항 해결 및 중요한 시스템 Logic을 처리, 결정
IM (Information Manager)	김용범	프로젝트와 관련된 모든 정보(기술적, 관리적)를 수집하고 취합 및 정보제시
SM (Study Manager)	박주형	프로젝트 전과 진행 중 각 팀원의 프로그램 처리 시 의문사항 해결을 위해 노력하고 방안 제시
DM (Document Manager)	임낙중	프로젝트에 관련된 모든 문서 양식 통합 체크 및 취합 관리, 문서 표준안 작성
AM(Analysis Manager)	박준형	문제를 분석하여 시스템의 기능, 범위, 제약사항 등의 정확한 요구사항을 도출. 위험요소를 분석 제거하면서 개발과정을 관리하고 조정
RM(Repository Manager)	이용민	회의록 작성(서기 역할) 및 모든 문서 파일 저장 관리

(2) 제약 사항과 가설

n MDS의 제약 사항

MDS는 J2EE와 J2ME 기술로 구축되어야 합니다.

또한 **3tier**를 기반으로 하여 다음과 같이 구축되어야 합니다

- user interface : HTML 기반
- middle tier : servlet, jsp, ejb
- database : oracle 8i

(3) 위험 요소

n DB 연동 문제

- 실시간으로 **DB**을 연동하는 방법이 가장 큰 문제
- 가맹점 간의 **DB** 동기화 문제

(4) 기능적 요구 사항 (FRs)

n Essential

- .. 웹, 모바일을 통한 **DVD**대여 및 구매 시스템
- .. 신청한 **DVD**에 대한 배송정보 시스템
- .. 회원이 구매하고 대여한 정보에 대해서 확인할 수 있는 페이지
- .. 가맹점 가입 및 탈퇴

n High-value

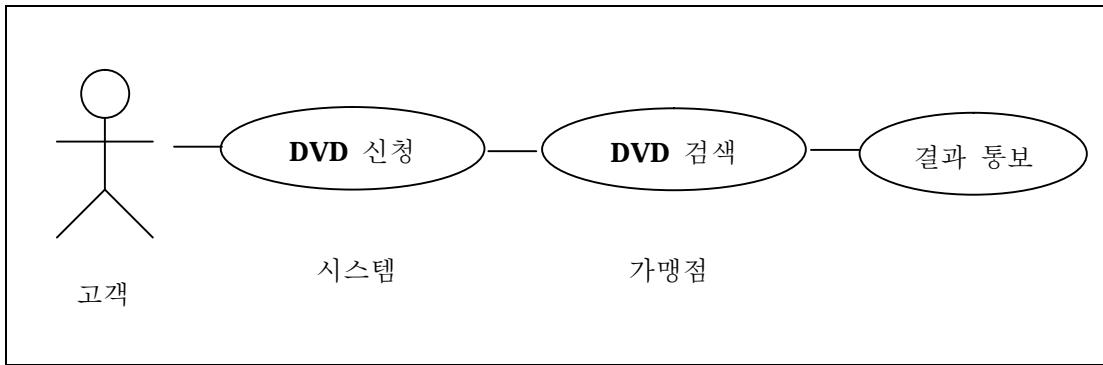
- .. 회원 가입 및 탈퇴
- .. **DVD**에 대한 정보 검색

n Follow-on

- .. 방문자와 기존회원은 공동구매를 통해 원하는 **DVD**를 싸게 구입
- .. 가맹점에서는 다량구매를 통해 많은 양의 **DVD**를 구입

① Use Case

Use Case Name	Priority	Number	Description
Rent DVD	E	1	DVD 대여
Purchase DVD	E	2	DVD 구매 및 판매
Information of Delivery	E	3	배송 정보 확인
Manage DVD Shop	E	4	가맹점 관리
Manage Member	E	5	회원 관리
Rent Info.	H	6	회원이 구매, 대여한 정보 확인
Search a DVD	H	7	DVD 검색
Login in	H	8	로그인
Large Purchase	F	9	대량 구매
Joint Purchase	F	10	공동 구매



② 유즈케이스 상세 요구

FR	상세 요구
E1- 1	이 시스템은 회원으로 하여금 대여에 대해서 DVD 를 추가, 삭제 할 수 있도록 해야 한다.
E1- 2	대여할 수 있는 DVD 는 1개일 수 있고 그 이상일 수 있다.
...	...
E1- 4	대여는 대여신청을 한 후 아직 배송 작업이 시작되기 전 까지는 대여 취소가 가능하다.
E2- 1	이 시스템은 회원으로 하여금 판매에 대해서 DVD 를 추가, 삭제 할 수 있도록 해야 한다.
E2- 2	구매할 수 있는 DVD 는 1개일 수 있고 그 이상일 수 있다.
...	...
E8- 2	한명의 고객(회원)은 하나의 대량 구매와 연관이 있다. 하지만 구매 품목은 여러 개일 수 있다.
E9- 1	시스템은 공동 구매의 대상이 되는 DVD 목록을 화면에 보여 주어야 한다.
E9- 2	한명의 고객(회원)은 하나의 공동 구매와 연관이 있다. 하지만 구매 품목은 여러 개일 수 있다.

(5) 비기능적 요구 사항(NFRs)

① 성능

NFR	상세 요구
E1- 101	한 달에 약 250건의 대여가 생긴다. 앞으로 한 달에 약 300건의 대여를 지원할 수 있어야 한다.
E2- 101	한 달에 약 100건의 판매가 생긴다. 앞으로 한 달에 약 200건의 판매를 지원할 수 있어야 한다.
H7- 101	찾고자 하는 DVD 를 검색 시 응답시간은 5초를 넘지 말아야 한다.

② 사용성

NFR	상세 요구
E5- 101	회원 가입(생성) 시에 사용자가 이 시스템에 쉽게 접근하고 사용할 수 있어야 한다.
E7- 101	DVD 검색 시 단어 입력의 불편함을 최소화 하고 클릭만으로도 쉽게 검색 할 수 있어야 한다.

③ 보안

NFR	상세 요구
E5- 101	회원에 대한 정보 입력, 수정 시에 외부 시스템에 의해 정보 유출이 되는 것을 막을 수 있어야 한다.

(6) Project Glossary

용 어	설 명
E- MAIL	이메일 주소
장바구니	쇼핑 카트와 동일
공동 구매	이벤트로 제공하는 구매를 의미한다. 몇 가지 DVD 품목을 정해 놓고 그 목록에 대해서 구매 신청을 하면 신청한 사람이 몇 명이냐에 따라 신청 인원수가 많으면 보다 싼 가격에 DVD를 구매할 수 있다.
대량 구매	한 고객(회원 또는 가맹점 주인)이 많은 DVD를 구매함을 의미한다. 한 번의 구매 시, 20장 이상일 경우 대량 구매라고 부른다.

3) 유즈케이스 시나리오

유즈케이스 시나리오는 작업이 성공적으로 수행된 경우와 실패한 경우를 별개의 시나리오로 작성합니다.

여기서는 DVD 대여에 관한 시나리오를 보겠습니다.

1. 대여신청

1.1 원하는 영화의 대여가 가능한 경우

회원은 대여를 원하는 영화를 선택한다.

회원은 선택한 영화의 ‘대여하기’ 버튼을 클릭하면 ‘장바구니’화면으로 넘어간다.

‘장바구니’화면에서는 선택한 ‘영화제목’, ‘대여 예정일’, ‘회수 예정일’, ‘총 대여가격’ 등의 정보를 보여준다.

‘장바구니’화면에 있는 ‘대여신청’ 버튼을 클릭한다.

1.2 원하는 영화의 대여가 불가능한 경우

대여가 불가능함을 알리는 화면을 보여준다.

1.3 로그인 하지 않은 상태이고 회원가입이 되어 있는 경우

대여자는 대여하기 버튼을 클릭 한다.

로그인 화면을 보여준다.

아이디와 비밀번호를 입력하고 로그인 버튼을 누른다.

전체 영화 목록 페이지를 보여준다

1.3 로그인 하지 않은 상태이고 회원가입이 않되 있는 경우

대여자는 대여하기 버튼을 클릭 한다.

로그인 화면을 보여준다.

아이디와 비밀번호를 입력하고 로그인 버튼을 누른다.

4) 유즈케이스 다이어그램 작성

(1) 유즈케이스 품(based on SRS)

유즈케이스 품은 하나의 유즈케이스와 시나리오의 상세한 분석결과를 기록하기 위한 틀입니다.

① Primary Form

Use Case#/ID And Name	E5 Manager Member
Description	홈페이지 이용 회원가입을 위한 use case
Actor	Primary : Customer Secondary :
Priority	Essential
Risk	High
Pre- conditions /Assumptions	홈페이지 초기화면 보여져 있다.

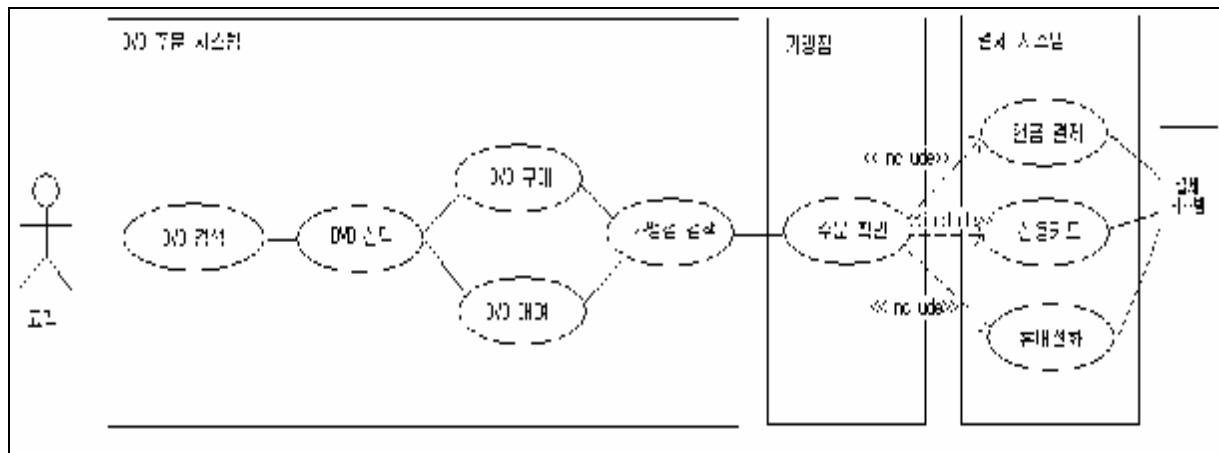
Trigger	회원가입을 위해 Customer 가 홈페이지 방문하여 회원가입을 선택한다.
Primary Scenario Flow of Events	<ol style="list-style-type: none"> 1. 회원가입을 위해 홈페이지를 방문한다. 2. 회원가입 버튼을 선택한다. 3. 홈페이지 이용약관을 보여준다. 4. 만약 이용약관에 동의하지 않으면 <ol style="list-style-type: none"> 4.1 회원가입을 할 수 없다. 5. 그렇지 않으면 이용약관에 동의한다. 6. 개인정보란에 ID를 입력한다. 7. 만약 입력한 ID가 존재한다면 <ol style="list-style-type: none"> 7.1 다른 ID를 입력한다. 8. 그렇지 않으면 비밀번호를 입력한다. 9. 비밀번호를 재입력한다. 10. 개인정보란에 성명, 주민번호, 휴대폰번호, 자택전화번호, e-mail을 입력한다. 11. 우편번호를 선택한다. 12. 나머지 주소를 입력한다. 13. 회원가입 버튼을 선택한다. 14. 데이터베이스에 저장하고 홈페이지 초기화면으로 돌아간다.
Post- conditions	Customer 의 정보가 데이터베이스에 저장되어진다. 회원가입화면 닫힌다.
Secondary Scenarios	홈페이지 이용약관에 동의하지 않을 경우

② Secondary Form

Use Case#/ID and Name	E5 Manager Member
Actor	Primary : Customer Secondary Scenario : 이용약관에 동의하지 않을 때
Secondary Scenario	<ol style="list-style-type: none"> 1. 홈페이지 이용약관을 보여준다. 2. 만약 이용약관에 동의하지 않으면 <ol style="list-style-type: none"> 2.1 회원가입을 할 수 없다.
Alternative Flow	홈페이지 이용약관에 동의하지 않고 회원가입을 하지 않을 수 있다.

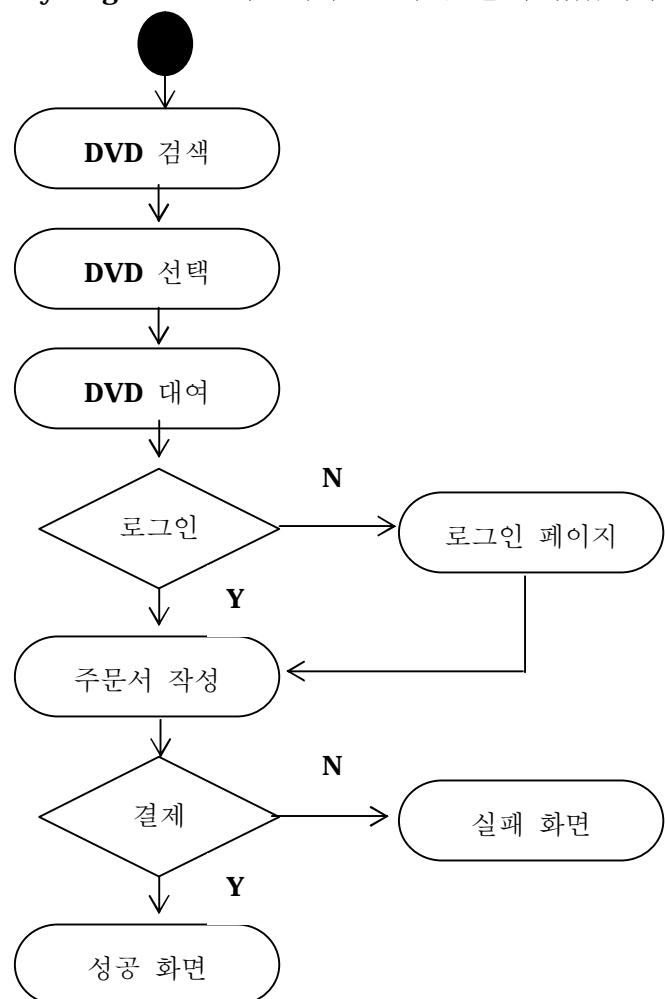
(2) 상세 유즈케이스

상세 유즈케이스 다이어그램에서는 **actor**와 **actor**간의 상속 관계나 유즈케이스와 다른 유즈케이스간의 상속 관계를 표현합니다.



(3) Activity Diagram

activity diagram으로 유즈케이스를 검증 할 수 있습니다.



과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원3 : 기능적 요구사항 분석

3 모듈 : Key Abstraction 과정

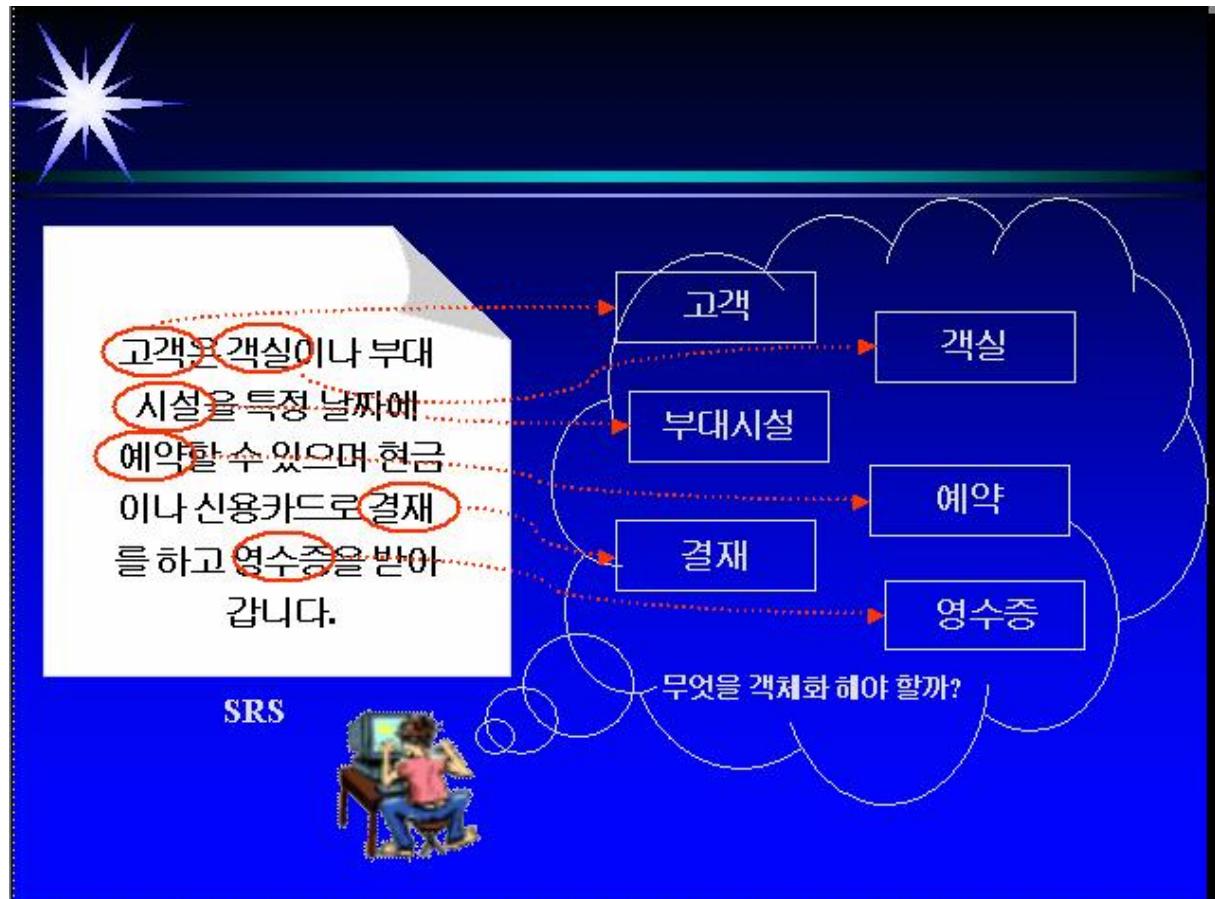
담당강사 : 전은수

■ 생각해봅시다 ■

지금까지 해온 작업을 토대로 시스템을 개발하려 합니다. 우리는 객체지향 원리에 따라 개발 프로세스를 진행해 왔습니다. 객체 지향 원리는 시스템의 내용을 객체화하여 서로 유기적인 관계로 실행해 나갈 수 있게 하는 원리입니다. 하지만 **SRS** 문서내의 모든 내용을 객체라고 볼 수는 없습니다. 그 많은 내용 중에서 객체가 될 수 있는 것과 없는 것은 무엇일까요? 어떤 기준으로 구분하겠습니까? 그리고 어떤 방법으로 작성한 문서들을 분석하겠습니까? 이번 모듈을 통해 **SRS** 문서의 핵심 내용을 분석하고 그것을 객체화 하도록 할 수 있는 방법을 알아보겠습니다.

애니메이션

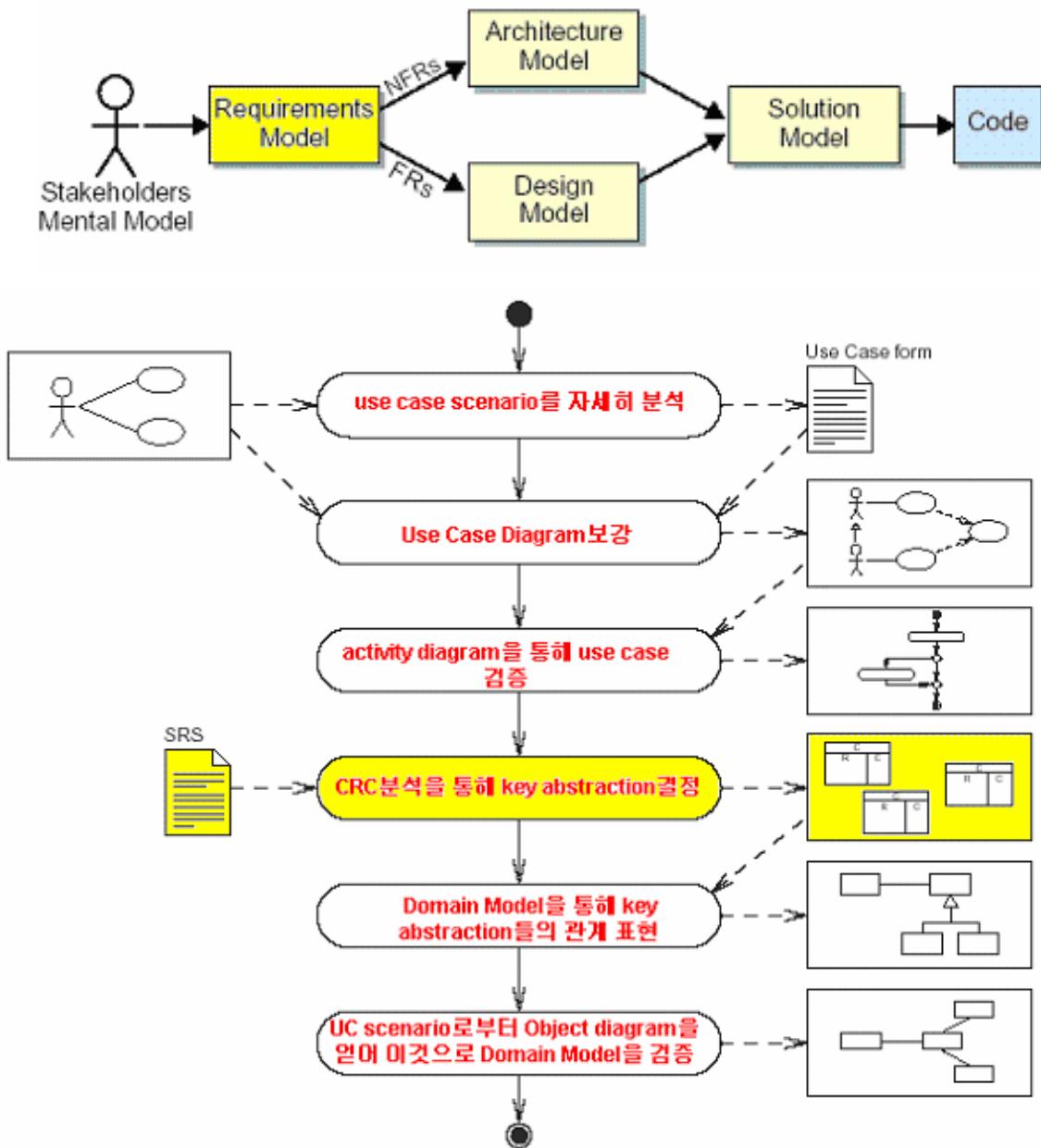
비즈니스 분석가 : “ 시스템의 요구 사항 문서인 **SRS**에서 객체가 될 수 있는 엔티티만을 골라 내야겠구나...”



■ 학습하기 ■

참고하세요

n 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. Candidate Key Abstraction

1) Key Abstraction

“Key Abstraction은 problem domain의 어휘를 구성하고 있는 클래스나 객체를 말합니다.”(Booch, page 162)

Key abstraction은 시스템에 있는 주요 객체들의 이름입니다. 이것은 도메인 전문가와 사용자가 업무에 대한 얘기를 할 때 사용하는 공통된 언어의 구성요소 일부를 말합니다.

도메인을 위한 key abstraction을 결정하는 방법은 다음과 같습니다.

1. SRS 문서에 있는 모든 명사를 “Candidate Key Abstractions Form”에 나열해서 모든 후보 key abstraction을 파악합니다.
2. key abstraction의 핵심 부분을 알아내기 위해 CRC 분석기법을 사용합니다. Key abstraction은 기능이 있는 객체(responsibility) 또는 다른 객체에 의해 사용되는 객체 (collaborator)로 볼 수 있습니다.

이 모듈에서는 key abstraction을 알아내기 위한 한가지 기술만 소개합니다. 다른 방법도 있지만 모든 기술을 다루는 것은 이 과정의 목적에서 벗어납니다.

2) Candidate Key Abstractions 알아내기

(1) 개요

SRS 문서를 다음과 같은 부분에 초점을 두어, SRS 문서의 특별한 명사 어휘를 모두 파악하는 작업을 시작하겠습니다.

SRS는 여러 section(절)로 구성되어 있습니다.

각 절에서 다음과 같은 내용에 유의하면서 key abstraction을 합니다.

– Scope 절과 Context 절

이 부분들을 통해 제안된 시스템의 high-level 관점과 시스템의 외부 “touch point”를 알 수 있습니다. 이 내용에 속하는 명사들은 problem domain을 이해하는 데 필수적이라고 할 수 있습니다.

– Functional Requirements 절

SRS 문서의 이 절은 problem domain에 있는 기술적인 내용들을 모두 다루고 있습니다.

여기서 problem domain의 세부사항과 기능들을 찾을 수 있습니다. 이 때 가장 중요한 부분은

- 유즈케이스와 유즈케이스 시나리오
- 기능적 요구 사항

입니다. 대규모 프로젝트에서는 **SRS** 문서도 분량이 많을 것입니다. 명사를 찾기 위해 **SRS** 문서를 분석하는 작업은 시간이 오래 걸릴 수 있습니다. 연습을 통해서 도메인의 일부가 확실히 아닌 명사는 스kip하는 능력을 키울 수 있습니다.

참고하세요

n touch point

touch point : 외부 시스템과 제안된 시스템의 접점을 의미하는 말로, 시스템이 외부의 다른 시스템을 통해서 작업하는 부분을 말합니다.

(2) SRS 문서의 명사들

명사를 찾기 위해 **SRS** 문서를 탐독하는 쉬운 방법은 **SRS** 문서의 사본을 출력해서 중요한 명사에 형광펜으로 체크하는 것입니다.

호텔 예약 시스템의 **SRS**에서 발췌한 부분에서 명사를 긁은 글씨로 체크해 보겠습니다.

– Scope절

“호텔 예약 시스템은 숙박과 아침식사(**bed and breakfast- B&B**), 비즈니스 **retreat** 시설을 포함한 여러 개의 숙박 시설에 대한 예약을 관리할 수 있어야 합니다. 시스템은 고객들이 이곳의 각종 시설과 방의 상태, 현재 예약 상황 및 과거의 예약 이력 등을 볼 수 있고, 새로운 예약도 할 수 있는 웹 어플리케이션도 제공해야 합니다. 그리고 시스템은 소규모 이벤트(수용 시설과 소규모 컨퍼런스 등)를 조정해야만 합니다.”

– Project Glossary절

Term	Definition
도착 예정일 (Arrival Day)	고객이 요구한 숙박 시설에 도착하게 될 날짜, 예약할 때 고객이 요청한 날짜
기본 요금 (Basic Rate)	각종 추가 요금을 제외한 1일 방 이용요금
예약 담당 직원 (Booking Agent)	숙박 시설의 직원 중 전화로 예약을 접수하는 일을 하는 직원

– Context절

“호텔 예약 시스템에는 세 개의 주요 ‘**touch point**’가 있습니다 : 데이터 저장을 위한 주 **DBMS**와 외부 시스템인 신용카드 승인 시스템(**Authorize.net**), 각 방의 **TV**에 대한 영화 공급을 조정하는 그 지역의 영화 주문 시스템입니다.”

- Functional Requirement 절

“시스템은 고객에 대해 다음과 같은 정보를 수집해야 합니다 : 이름, 주소, 집 전화번호, 주로 사용하는 신용카드(타입, 번호, 유효기간)”

(3) Candidate Key Abstractions Form

후보 key abstraction을 기록하기 위한 폼은 세가지 필드로 구성되어 있습니다.

- Candidate key abstraction

SRS에서 찾은 명사들을 기록하는 필드입니다.

- Reason for Elimination

후보였던 것이 key abstraction이 되면 빈칸으로 남겨지게 되고, 그렇지 않을 경우는 key abstraction에서 삭제된 이유를 기록합니다.

- Selected name

결국, key abstraction은 클래스 같은 소프트웨어 컴포넌트를 정렬한 것이 될 수 있습니다. 이 필드는 key abstraction으로 선택될 경우의 클래스 이름을 기록하는 곳입니다.

SRS 문서에 있는 명사들 중 잠재적인 key abstraction을 기록하기 위해 이 폼을 사용합니다. 첫번째 컬럼에는 명사들을 기록합니다. 뒤에 나올 CRC 분석과정에서 나머지 필드를 채우게 될 것입니다.

① Candidate Key Abstraction Form 작성의 예

다음 표는 호텔 예약 시스템에 대한 Candidate Key Abstraction Form입니다.

표 :: 호텔 예약 시스템에 대한 초기 Candidate Key Abstraction Form

Candidate Key Abstraction	Reason for Elimination	Selected Component Name
예약 (Reservation)		
숙박 시설(lodging properties)		
수용 시설(retreat properties)		
고객(customers)		
방(rooms)		
소규모 비즈니스 컨퍼런스 (small business conference)		
신용 카드 인증 시스템 (credit card authorization system)		
이름(first and last name)		
주소(address)		

호텔 예약 시스템에 대한 초기 Candidate Key Abstraction Form입니다. Candidate Key Abstraction에는 후보 키들을 적어두고, 나머지 두개의 필드는 CRC 분석과정을 통해 채워

나가게 됩니다.

(4) Project Glossary

Candidate key abstraction 과정은 프로젝트 용어집이 적절한지 검증하기에 좋은 기회입니다.

- 모든 도메인 전문 용어들을 검증해야 합니다.

Candidate key abstraction은 일반적으로 도메인 전문 용어입니다. 이것들은 모두 용어집에 기록되어 있어야 합니다.

- 프로젝트 용어집에 있는 동의어를 파악하고 문서와 소스코드에서 사용하기 위한 주요 용어를 선택해야 합니다.

Candidate key abstraction 리스트에 종종 동의어가 있을 수 있습니다. 예를 들어, 예약 담당 직원은 예약 업무를 ‘booking’이라고 부를 수 있습니다. 따라서 ‘booking’의 정의에 ‘예약’을 참조하도록 해서, ‘예약’과 ‘booking’이 모두 프로젝트 용어집에 기록되도록 해야 합니다.

2. CRC Analysis

1) CRC 분석 기법을 사용한 Key Abstraction

(1) 개요

candidate key abstraction(후보키) 리스트를 완성한 후에, **key abstraction**의 핵심 내용을 파악하기 위해 리스트를 걸러내야 합니다. **CRC(class- responsibility- collaboration)** 분석 기법을 사용하는 것도 하나의 방법입니다.

CRC 분석을 위해

- 1) **candidate key abstraction**을 하나 선택합니다.
- 2) 이 후보기가 적합한지 유즈케이스를 분석합니다.
- 3) 기능과 협력 관계를 결정하기 위해 유즈케이스 시나리오와 **FRs**를 탐독합니다.
- 4) **CRC 카드**에 이 **key abstraction**을 기록합니다.
- 5) 이상의 내용을 토대로 **Candidate Key Abstractions Form**을 수정합니다.

이 과정은 반복적으로 이루어집니다. 하나의 후보키에서 시작해서 그 키가 진짜 **key abstraction**인지 평가하게 됩니다. 품을 수정하고 나서는 다른 후보키에 대해서 같은 작업을 하게 됩니다.

참고하세요

n CRC 분석

CRC 분석기법은 **Ward Cunningham**과 **Kent Beck**이 1980년대 후반 객체지향 설계에 대해 설명하기 위해 고안한 방법입니다. 좀더 많은 정보를 원한다면

<http://c2.com/doc/oppsla89/paper.html> 을 방문하십시오.

(2) Key Abstraction Candidate 선택

좋은 **key abstraction candidate**를 선택하는 것은 매우 직관적인 부분입니다. 여기서는 몇 가지 방법을 소개합니다.

- 도메인 전문가에게 물어봅니다.

도메인 전문가는 후보키가 진짜 **key abstraction**에 적합한지 즉시 말해줄 수 있습니다.

도메인 전문가를 만날 수 없다면 다음 두 방법이 유용할 수 있습니다.

- 유즈케이스 이름으로 사용된 **key abstraction candidate**를 선택합니다.

유즈케이스의 이름에는 주요 동사와 명사가 사용됩니다. 유즈케이스의 이름으로 사용되는 명사는 **key abstraction**이 될 수 있습니다.

- SRS 문서의 Scope 절에서 언급된 **key abstraction candidate**를 선택합니다.

SRS의 **Scope** 절은 시스템에서 자동으로 처리 해야 할 업무에 대한 설명입니다. 이 부분은 **high-level** 관점으로 쓰여졌기 때문에, 이 문서에 있는 명사는 대개 **key abstraction**이 됩니다.

보충

n key abstraction

호텔 예약 시스템에서, “예약(Reservation)”이라는 명사는 **SRS** 문서 전반에 걸쳐 여러 번 나타나기 때문에 좋은 후보가 됩니다.

- Scope 절

“호텔 예약 시스템은 여러 숙박 시설에 대한 예약을 관리해야 합니다.”

- 유즈케이스 이름

E1 : 예약 관리(Manage Reservation)

E5 : 온라인 예약 관리(Manage Reservation Online)

- FR 전반에 걸쳐 등장

“**E1-1 : 시스템은 예약 담당 직원이 예약을 새로 추가하고, 변경하고, 삭제할 수 있게 합니다.**”

(3) 관련된 유즈케이스 파악

Candidate key abstraction이 진짜 **key abstraction**인지 결정하기 위해, 해당 후보키가 어떤 기능이나 관련 업무가 있는지 알아야 합니다. 그러기 위해 유즈케이스의 세부 **FR**과 시나리오를 살펴봐야 합니다.

관련된 유즈케이스를 찾기 위해 후보키의 기능(**responsibility**)과 관련 업무(관련 업무, **collaborators**)를 알아야 합니다.

1. **Candidate key abstraction**을 위해 유즈케이스 이름을 자세히 봐야 합니다.

Candidate key abstraction이 유즈케이스의 이름으로 언급되었다면, 그 유즈케이스와 관련 있는 것입니다.

2. **Candidate key abstraction**을 위해 유즈케이스 설명서를 살펴 봅니다.

Candidate key abstraction이 유즈케이스 설명서에 언급되었다면, 그 유즈케이스와 관련

있는 것입니다.

3. **Candidate key abstraction**을 위해 유즈케이스 시나리오를 봅니다.
4. **Candidate key abstraction**이 언급되었는지 확인하기 위해 유즈케이스 시나리오의 내용을 탐독합니다.

만일 그렇다면 그 시나리오와 관련된 것입니다.

호텔 예약 시스템에서는 “예약”이라는 명사가 두 개의 유즈케이스에 있습니다.

E1 : 예약 관리(Manage Reservation)

E5 : 온라인 예약 관리(Manage Reservation Online)

관련된 유즈케이스들은 **CRC** 분석 과정을 수행하기 위해 사용될 것입니다 : 선택한 **Candidate key abstraction**의 기능(**responsibility**)과 관련 업무(**collaborators**)를 확인하는 작업을 하게 됩니다.

(4) 기능(**Responsibilities**)과 관련 업무(**Collaborators**) 결정

Candidate key abstraction의 기능과 관련 업무를 위해서 파악된 유즈케이스의 시나리오와 **FR**을 조사해야 합니다. **Key abstraction**의 기능은, 정적인 변수의 경우, 속성, 기능 또는 속성이 갖는 데이터 값의 특정 범위를 말합니다. 관련 업무는 **candidate key abstraction**과 관련된 다른 객체, 대개 다른 **key abstraction**을 말합니다.

기능을 발견하지 못하면, 그 후보키를 삭제할 수 있습니다. 제거 이유를 기록할 때는 **Candidate Key Abstraction Form**의 두 번째 필드를 사용합니다.

① 호텔예약 시스템의 ‘예약’ **key abstraction**에 대한 기능(**Responsibilities**)과 관련 업무(**Collaborators**) 파악의 예

호텔 예약 시스템의 **SRS** 문서에는 ‘예약(**Reservation**)’ **key abstraction**에 대한 기능과 관련 업무에 대해 구체화한 다음과 같은 기능적 요구사항이 포함되어 있습니다.

E1- 1 : 시스템은 예약 담당 직원이 예약을 추가, 변경, 삭제할 수 있게 합니다. 예약(**reservation**)에는 도착 날짜(**arrival date**), 출발 날짜(**departure date**), 예약 **ID(reservation ID)** 등의 내용이 있습니다.

이 **FR**은 “예약”이 세 가지 중요한 데이터 항목을 포함하고 있다는 것을 구체적으로 설명하고 있습니다.

E1- 2 : 예약은 정해진 기간동안(도착 날짜와 출발 날짜 사이) 하나 또는 그 이상의 방에 대해 이루어 질 수 있습니다.

이 **FR**은 “예약”이 하나 또는 그 이상의 방과 관련된다는 것을 설명합니다. 이 관계는 협력(**collaboration**)으로 볼 수 있습니다.

E1-3 : 하나의 예약은 단 한 명의 고객과 이루어집니다.

이 **FR**은 “예약”과 고객 사이의 다른 협력관계를 나타냅니다.

E1-5 : 예약은 “처리 중”인 상태에서 시작합니다.

이 **FR**은 예약의 상태 값의 이름을 설명합니다. 이런 것들은 속성의 기능으로 기록될 수 있습니다.

‘예약’ **Key abstraction**에 대한 분석은 파악된 속성(예를 들어, 도착 날짜 같은)과 관련 업무 (**collaborators**)를 포함한다는 것을 알아야 합니다. 속성은 **candidate key abstraction** 리스트에 또 있을 수 있습니다. 이런 것들은 속성이라는 이유로 삭제 될 수 있습니다.

(5) CRC Card를 사용해서 Key Abstraction 기록하기

Key abstraction을 파악한 후에, 해당 **key abstraction**의 기능과 관련 업무를 파악하기 위해 **CRC** 카드를 작성할 수 있습니다.

[이미지]

■ CRC 카드 형식

Class Name	
Responsibilities	Collaborators

CRC 카드의 맨 위는 클래스 이름 필드로, **Key abstraction**의 이름을 적는 곳입니다. 적당한 이름을 찾는 것은 중요한데, 특히 그 개념과 비슷한 용어가 있을 경우는 더욱 그렇습니다. 추천하는 바로는, 간단하고 짧은 것으로 이름이 지나치게 길지 않다면 약어는 사용하지 않는 게 좋습니다.

Key abstraction의 기능과 속성을 기록하기 위해 **responsibility** 컬럼을 사용하고, **key**

abstraction에 대한 관계나 행동적 협력을 기록하기 위해 **collaborators** 컬럼을 사용합니다.

② 호텔예약 시스템의 ‘예약’ **key abstraction**에 대한 CRC 카드 작성의 예
호텔 예약 시스템에 대한 CRC 카드는 다음과 같이 작성할 수 있습니다.

[이미지]

n ‘예약(Reservation)’ Key Abstraction의 CRC 카드

Reservation	
Responsibilities	Collaborators
Reserves a Room	Room Customer
status (New, Held, Confirmed) arrival date departure date form of payment reservation number	

CRC 카드의 맨 위칸에는 **Reservation**이라는 **Key Abstraction**의 이름을 씁니다. 왼쪽의 **Responsibility**에는 **Reservation**과 관련된 속성이나 업무 내용에 대해 기록하고, 오른쪽의 **Collaborator** 필드에는 **Reservation**과 관련된 다른 **Key Abstraction**을 기록합니다.

Role- playing exercises로 CRC 카드를 사용하면 팀과 도메인 전문가가 유즈케이스를 완수하기 위해 시스템에 의해 어떻게 각각의 **key abstraction**이 사용되었는지 파악하고 분석하는데 도움이 됩니다. 유즈케이스 시나리오의 이런 **role- playing**은 CRC 과정에서 필수적입니다.

Role- playing exercise는 일반적인 방법은 아니지만, **problem domain**에 대해 자세히 이해하기 위한 강력한 기술이 될 수 있습니다.

[보충]

n Role- playing exercise

예를 들어, **Ms. Jane Google**의 예약 관리(**Manage Reservation**) 유즈케이스 시나리오의 경우, 분석팀의 한 팀원은 **Reservation CRC** 카드에 대한 예약(**reservation**) 객체를 나타내고, 다른 팀원은 **Customer CRC** 카드에 대한 고객(**customer**) 객체를 나타냅니다. 세 번째 팀원은 예약 담당 직원인 **Booking Agent actor**를 표현합니다. 이 사람은 다른 두 사람과 상호작용을 합니다. 예를 들어, 예약(**reservation**)에 고객(**customer**)을 추가하는 작업을 합니다.

(6) Candidate Key Abstractions Form 수정

선택한 후보키가 기능을 갖고 있다면, **key abstraction**의 이름을 “**Selected Name**” 필드에 적습니다.

선택한 후보키가 기능은 갖고 있지만 관련 업무가 없는 경우, 이 후보키는 삭제되어야 합니다.

Key abstraction으로 선택되지 않은 이유를 기록합니다. 후보키가 **key abstraction**으로 선택되지 못한 이유로 다음과 같은 것을 들 수 있습니다.

- 속성의 이름
- 하위 타입의 이름
- 외부 시스템 또는 **actor**의 이름
- 속성 값의 이름(상태 속성과 같은)

③ 호텔예약 시스템의 ‘예약’ **key abstraction**에 대한 최종 Candidate Key Abstraction Form

CRC 분석 과정을 몇 번 반복하면, 호텔 예약 시스템에서 파악한 **key abstraction** 몇 가지를 알 수 있습니다. 여러 후보키들이 **key abstraction**이 되지 못했습니다.

아래 표는 호텔 예약 시스템의 최종 **Candidate Key Abstraction Form**입니다.

■ 표 :: 호텔 예약 시스템의 최종 Candidate Key Abstraction Form

Candidate Key Abstraction	Reason for Elimination	Selected Component Name
예약 (Reservation)		Reservation
숙박 시설(lodging properties)		Property
수용 시설(retreat properties)	Property의 하위 타입	
고객(customers)		Customer
방(rooms)		Room
소규모 비즈니스 컨퍼런스 (small business conference)	Reservation의 하위 타입	
신용 카드 인증 시스템 (credit card authorization system)	외부 시스템	
이름(first and last name)	Customer의 속성	
주소(address)	Customer의 속성	

과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원3 : 기능적 요구사항 분석

4 모듈 : Problem Domain Model 의 생성

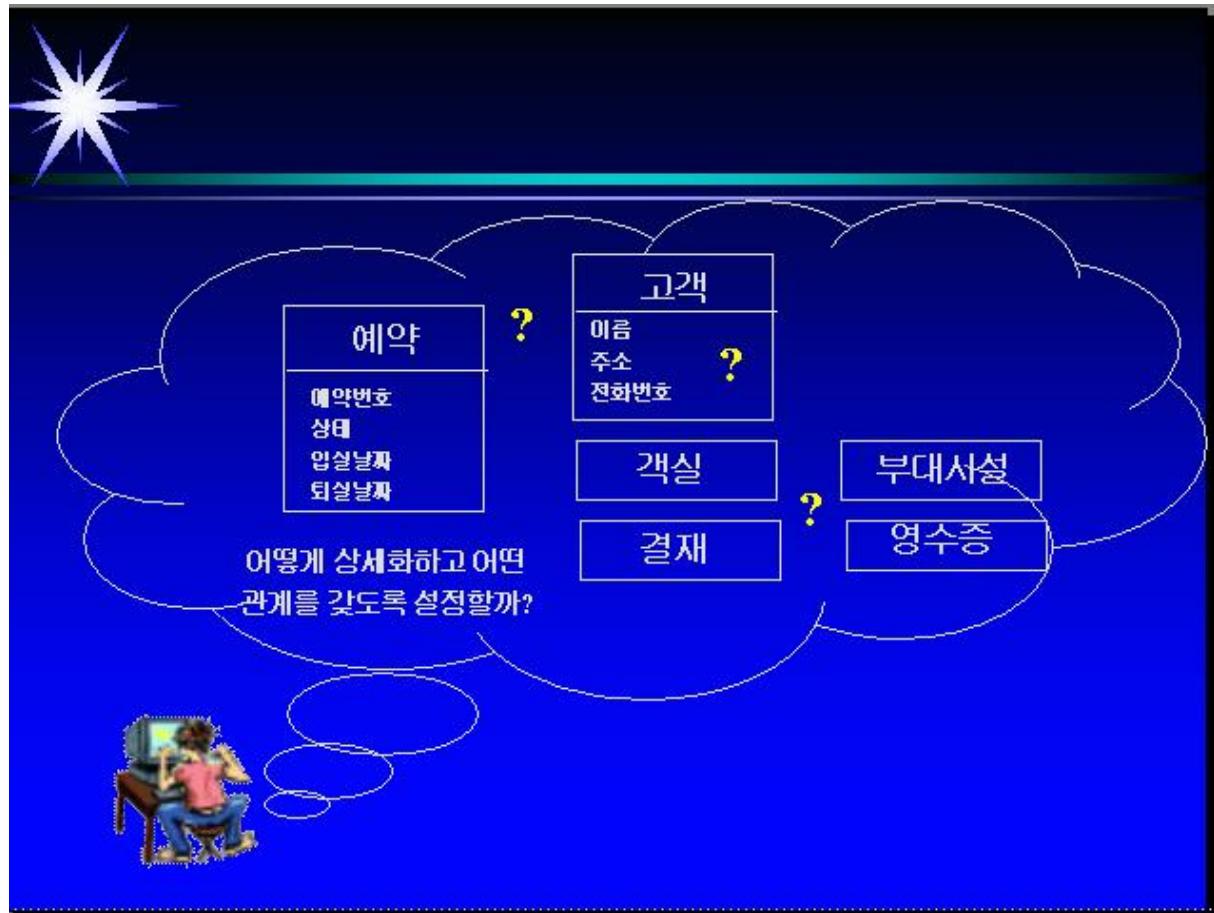
담당강사 : 전은수

■ 생각해봅시다 ■

Problem domain에서 **key abstraction**을 파악했습니다. **Key abstraction**을 알아내는 것 만으로 요구 사항 분석 단계가 끝나는 것은 아닙니다. 시스템을 구현하기 위해서는 알아낸 정보를 다시 분석하고 **UML**을 사용해서 모델링 해야 합니다. 어떻게 **key abstraction**의 내용을 모델링 할 수 있을까요?

애니메이션

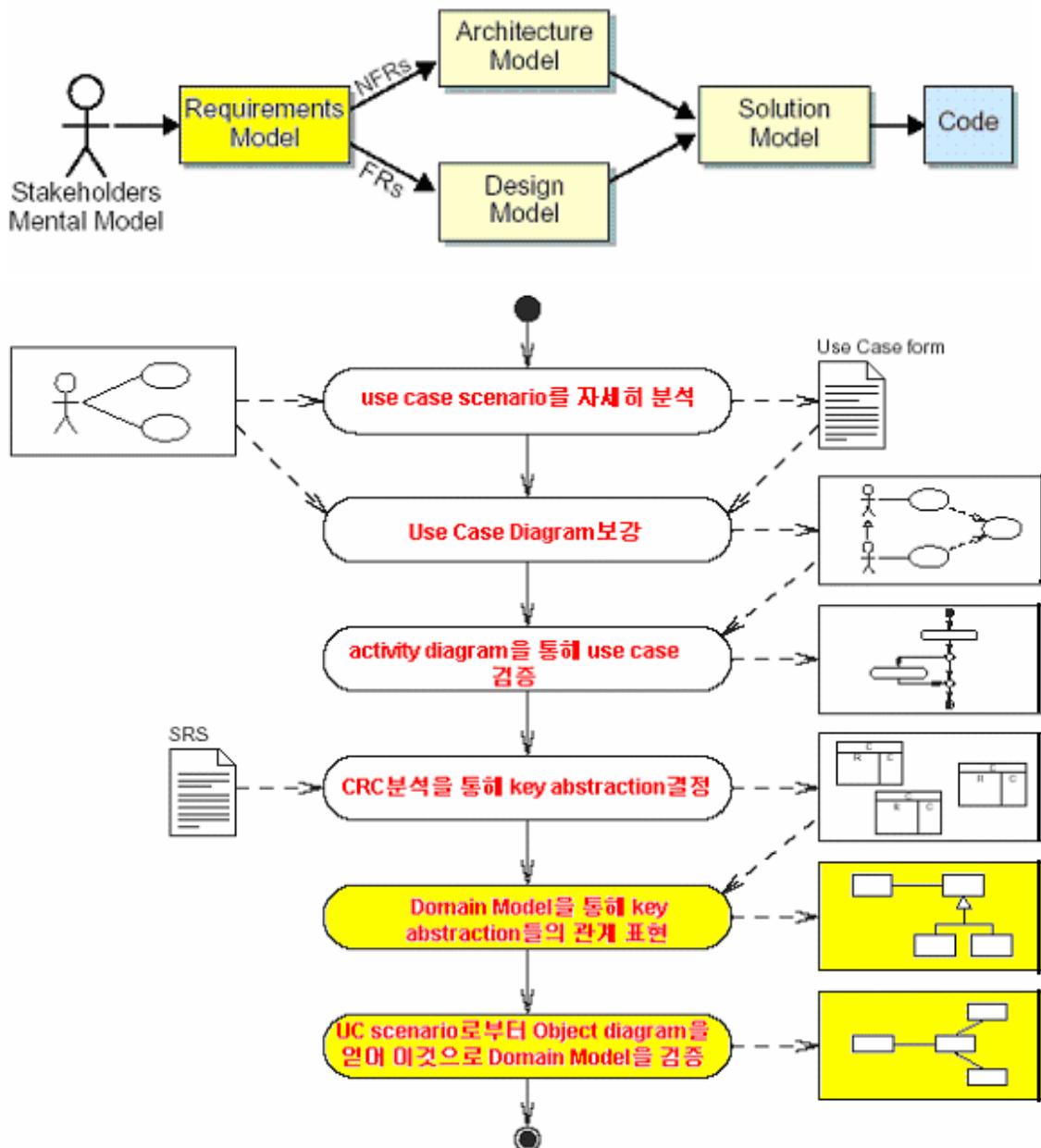
비즈니스 분석가 : “**key abstraction**으로 객체가 될만한 엔티티를 뽑았는데, 이것이 실제 클래스가 되기 위해선 어떤 구조를 가져야 할까? 그리고 각 클래스들끼리의 연관 관계도 표현해야 할텐데...”



■ 학습하기 ■

참고하세요

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. Class Diagram의 구성 요소

1) Domain Model

“시스템에 있는 클래스의 바다로, 문제 영역의 어휘를 포착해서 제공합니다. 이것은 또한 개념적 모델이라고도 합니다.”(Booch Object Solution page 304)

Domain model은 요구 사항 모델의 세가지 주요 관점 중 하나입니다. 다른 두 가지는 **SRS** 문서 본문에 있는 요구 사항과 **Use Case** 모델입니다.

- **Domain model**의 클래스들은 시스템의 **key abstraction**입니다.

Domain model은 **key abstraction**을 시각적으로 표현한 것으로, 각 **key abstraction**은 **Class** 다이어그램의 클래스가 됩니다.

- **Domain model**은 **key abstraction** 간의 관계(관련 업무)를 보여줍니다.

CRC 분석 기법은 클래스(**key abstraction**), 클래스간의 기능, 관련성 등을 파악합니다. 이런 것들이 모두 **Domain model**에 나타날 수 있습니다. **Domain** 모델은, **CRC** 카드가 아니라, 프로젝트의 장기간에 걸친 산출물입니다.

이 모듈에서는 **Domain model**과 생성 방법, 그리고 **Domain model**을 나타내는 **UML** 다이어그램에 대해 알아보겠습니다.

참고하세요

- 모든 **Key abstraction**은 **class**?

key abstraction이 **class**로 정확히 맵핑되는 것은 아닙니다. **EJB**의 **entity bean** 형태로 나타날 수도 있습니다. **Key abstraction**은 다양한 컴포넌트로 구현될 수 있습니다.

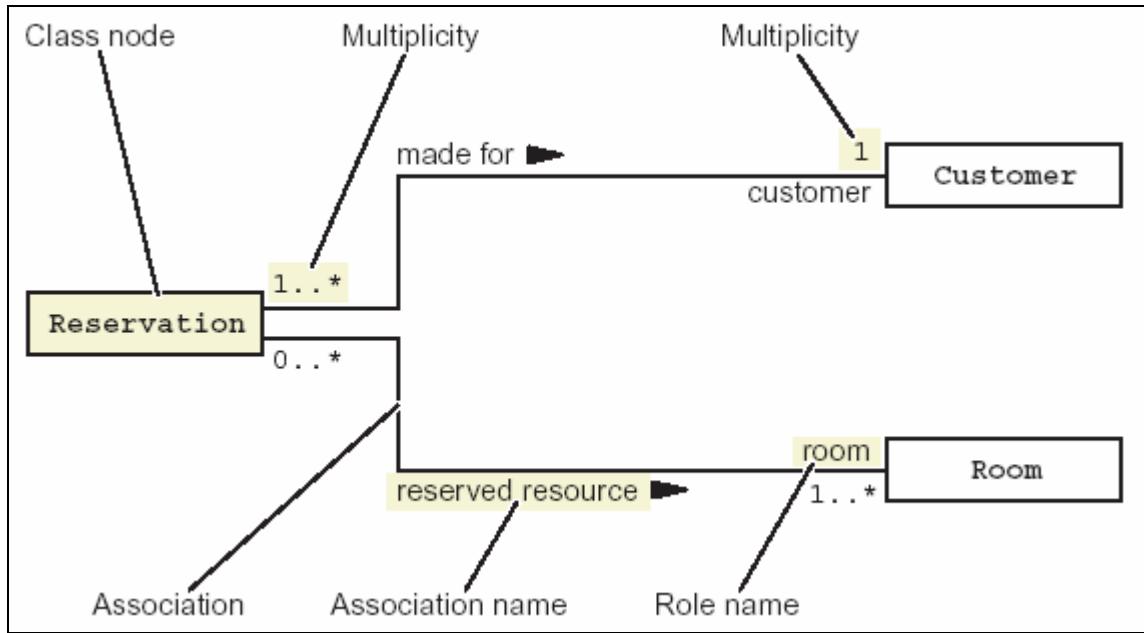
2) Class Diagram의 구성 요소 파악

(1) Class Diagram

UML의 **Class** 다이어그램은 많이 알고 있을 겁니다. **Class** 다이어그램은 클래스와 클래스의 멤버, 그리고 클래스 간의 관계(대개 **association**이라고 하는)를 시각적으로 표현한 것입니다. 다음은 라벨이 붙은 모양의 **Class** 다이어그램입니다.

이미지

- **UML Class** 다이어그램의 요소

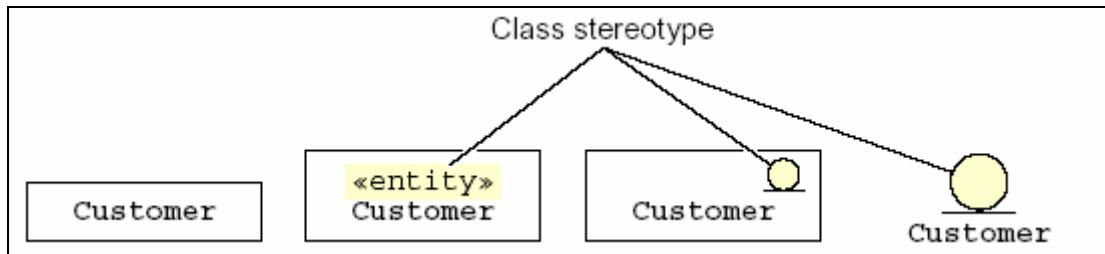


(2) Class Nodes

Class node는 모델 안에 있는 객체의 클래스를 의미합니다. **Class node**는 UML에서 여러 시각적 형태로 나타낼 수 있습니다만, **class node**는 사각형 안에 굵은 글씨로 클래스 이름을 써서 나타냅니다.

이미지

n Class node를 나타내는 서로 다른 방법



class node는

- **key abstraction** 같은 개념적 **entity**를 나타냅니다.

분석 단계 워크플로우에서, **Domain model** 안의 클래스 노드는 **key abstraction**을 나타냅니다. 이 개념적 **entity**는 소프트웨어 컴포넌트에 일치하지 않습니다.

- 실제 소프트웨어 컴포넌트를 나타냅니다.

설계 단계 워크플로우에서, 클래스 노드는 대개 물리적 소프트웨어 컴포넌트를 나타냅니다. 컴포넌트는 보통 클래스지만, 다른 컴포넌트가 될 수도 있습니다. 예를 들어, 클래스 노드는 **EJB entity bean**을 의미할 수도 있습니다.

스테레오타입은 클래스 노드의 타입을 파악하는데 도움이 됩니다. **UML**의 스테레오타입은 문

자를 ‘‘<<’와 ‘>>’가 감싸는 형태와 아이콘 형태가 있습니다. 예를 들어, **Entity**는 밑줄이 있는 원 모양의 심볼로 나타낼 수 있습니다.

(3) Class Node Compartment

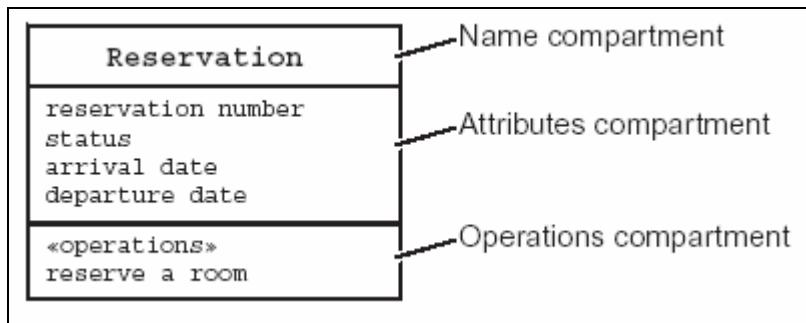
Class node는 세 칸으로 나눌 수 있습니다.

- 이름 칸은 클래스의 이름을 기록합니다.
- 속성 칸은 클래스의 속성을 기록합니다.
- 기능 칸은 클래스의 기능을 기록합니다.

다음은 **Reservation Class node**를 세 칸으로 시각적으로 나타낸 형태입니다.

[이미지]

n Class Node Compartment의 예



(4) Associations

Association(연관관계)은 클래스 간의 관계를 의미합니다. **Association**은 일반적으로 객체를 참조함으로써 두 객체가 실행할 때 서로 관련이 있다는 것을 명시합니다.

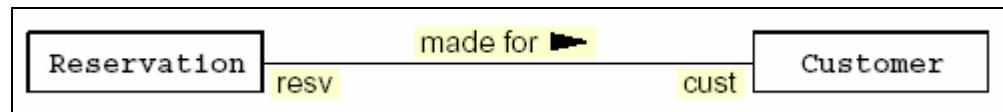
Class 다이어그램이 클래스 간의 관계에 대해 정적인 정보를 나타낸다는 것을 이해하는 것이 중요합니다. **Class** 다이어그램은 실행 시 객체의 동적인 관계를 나타내지는 않습니다. **Class** 다이어그램은 실행 시 객체의 동적인 상태를 표시하는 데는 제약이 있습니다.

① Relationship and Roles

다음은 **association**에 대한 예입니다. 이 연관관계는 “**Reservation**이 **customer**에 대해 이루어진다”는 것으로 읽을 수 있습니다.

[이미지]

n 관계와 Role Label이 있는 association



화살표 방향은 **association**을 읽는 방향을 가리킵니다. **Role name**은 연관관계에 있는 특정 클래스에서의 **role**을 가리킵니다. 앞의 그림에서 **Reservation**과 **Customer** 사이의 관계의 **role**은 **customer**입니다. 하지만, 클래스 이름과 **role**의 이름이 같을 경우, **role**의 이름은 삭제할 수 있습니다. 이 예에서, 두개의 **role** 이름은 쓰지 않을 수도 있습니다.

② Multiplicity(다중성)

다중성은 몇 개의 객체가 그 관계에 관여될 수 있는지를 결정합니다. 다음 그림으로 다중성 라벨을 사용하는 **association**의 예를 볼 수 있습니다. 이 관계는 “하나의 예약은 한명의 고객에 대해 1:1로 이루어진다”로 읽을 수 있습니다. 반대 방향으로 “한명의 고객은 하나 또는 그 이상의 예약을 할 수 있다”로 읽을 수도 있습니다.

[이미지]

n Multiplicity label로 나타낸 Association



기본 다중성 값은 **1**입니다. 하지만, 명시적으로 “**1**”이라고 표시하는 것이 좋습니다. 다중성 라벨은 다음과 같은 몇 개의 형태가 있습니다.

n 표 :: Multiplicity의 여러 타입

Multiplicity syntax	Meaning
1	객체간 1 : 1 대응 관계
n .. m	최소한 n 개의 객체, 최대한 m 개의 객체
0 .. 1	0 또는 1 개의 객체. 이 값은 객체에 대한 관계가 선택적일 경우를 의미할 때 사용합니다.
n,m	정확히 n 개의 객체 또는 정확히 m 개의 객체. 이 구문은 double-dot 표기법의 조합으로 나타낼 수 있습니다. 예를 들어, 1..2 , 4 , 6..10 으로 쓸 수도 있습니다.
0 .. *	0 또는 그 이상의 객체
1 .. *	최소한 1 개 이상의 객체

③ Navigation

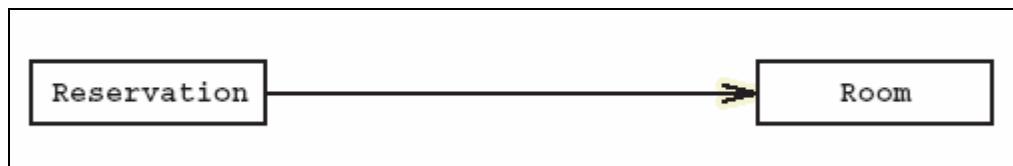
Association의 방향 화살표는 실행 시 연관관계가 적용될 수 있는 방향을 의미합니다. 이 정보는 설계 과정에서도 중요하지만, 분석 단계에서도 유용할 수 있습니다. 방향 화살표가 없는 연관관계는 한 객체에서 다른 객체로 또는 그 반대로 진행할 수

있음을 의미합니다. 예를 들어, 앞에서 본 **Customer**와 **Reservation**의 관계는 **Reservation** 객체가 **Customer** 객체를 수정할 수 있음을 의미하기도 하고, **Customer** 가 **Reservation**을 수정할 수도 있다는 것을 나타냅니다.

이것은 매우 유연하지만, 경우에 따라서 **problem domain**에 대한 의미가 충분하지 않을 수도 있습니다. 예를 들어, 호텔 예약 시스템의 기능적 요구 사항은 특정 방에서 일련의 예약에 대한 진행방향을 필요로 하지는 않습니다. 이런 제약 사항은 다음 그림으로 볼 수 있습니다. 이런 연관관계는 “**Reservation** 시스템에서 방을 변경할 수 있다”라고 읽을 수 있습니다. 하지만, 이런 모델은 “방에 대한 정보를 수정해서 시스템은 그 방에 대한 예약을 변경할 수 있다”는 가능성을 제외합니다.

이미지

- 진행 방향 화살표로 나타낸 association

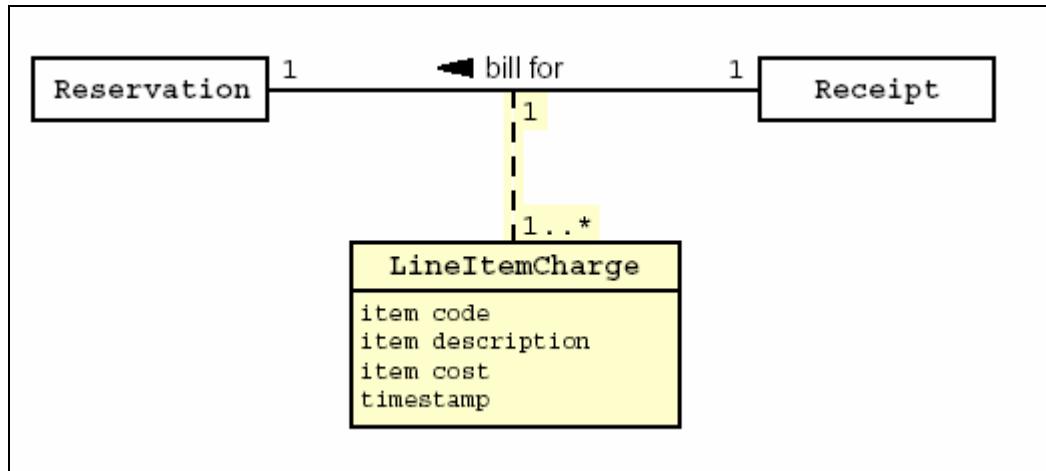


④ Association Classes

간혹 정보가 두 클래스간의 연관관계에 포함될 수 있습니다. 예를 들어, 고객이 호텔을 나갈 때, 예약은 영수증(계산서(bill)라고 하는)을 포함합니다. 영수증 그 자체는 정보를 가진 게 별로 없습니다. 하지만, 계산서는 실제 영수증에 기록이 되는 하나 또는 그 이상의 요금에 대한 내용을 담고 있습니다. **LineItemCharges**는 예약 영수증에 있는 비용과 그 항목을 기록한 연관관계 클래스(**association class**)입니다.

이미지

- Association class 예



2. Domain Model의 생성

1) Domain Model 생성

Key abstraction을 사용해서, 다음의 과정을 거쳐 Domain model을 생성할 수 있습니다.

1. 각 key abstraction에 대한 클래스 노드를 그립니다. 그리고
 - a. 알고 있는 속성을 나열합니다.
 - b. 알고 있는 기능을 나열합니다.
2. 협력 클래스 간의 관계를 그립니다.
3. 관계와 role name을 파악하고 기록합니다.
4. association multiplicity를 파악하고 기록합니다.
5. association 진행 방향을 파악하고 기록합니다.
6. association 클래스를 파악하고 기록합니다.

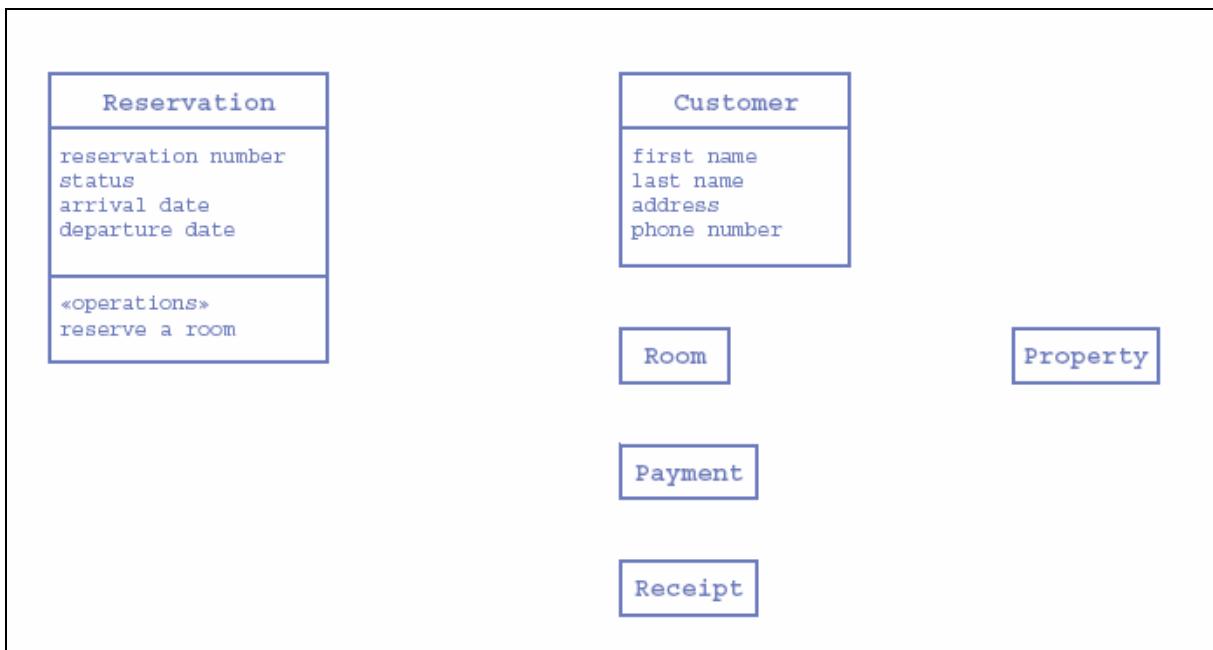
이제부터 이전 분석 단계에서 작성한 CRC 카드에 기록된 key abstraction을 이용해서 Domain model을 작성하는 6여섯 단계에 대해 살펴보겠습니다.

(1) Step 1 - Class Node 그리기

첫 번째 단계는 CRC 분석을 통해 알게 된 key abstraction에 대한 클래스 노드를 그리는 것입니다. Domain model을 좀더 구체적으로 작성하고 싶다면, 모든 속성과 기능을 기록합니다.

[이미지]

- Class Node 그리기



이 다이어그램은 각 클래스 노드의 중요한 **orientation**과 함께 그려졌습니다.

연습 삼아, 하나의 클래스 노드로 시작해보십시오. 다른 클래스 노드들은 먼저 그린 클래스 노드의 주위에 배치합니다. 다이어그램을 확장하면서 두 클래스의 관계가 필요하고 다이어그램의 서로 다른 곳에서 끝나게 됩니다. 이런 문제 때문에 적당한 시작적 구조를 찾기 전에 **Domain model**을 여러 번 작성해야 합니다.

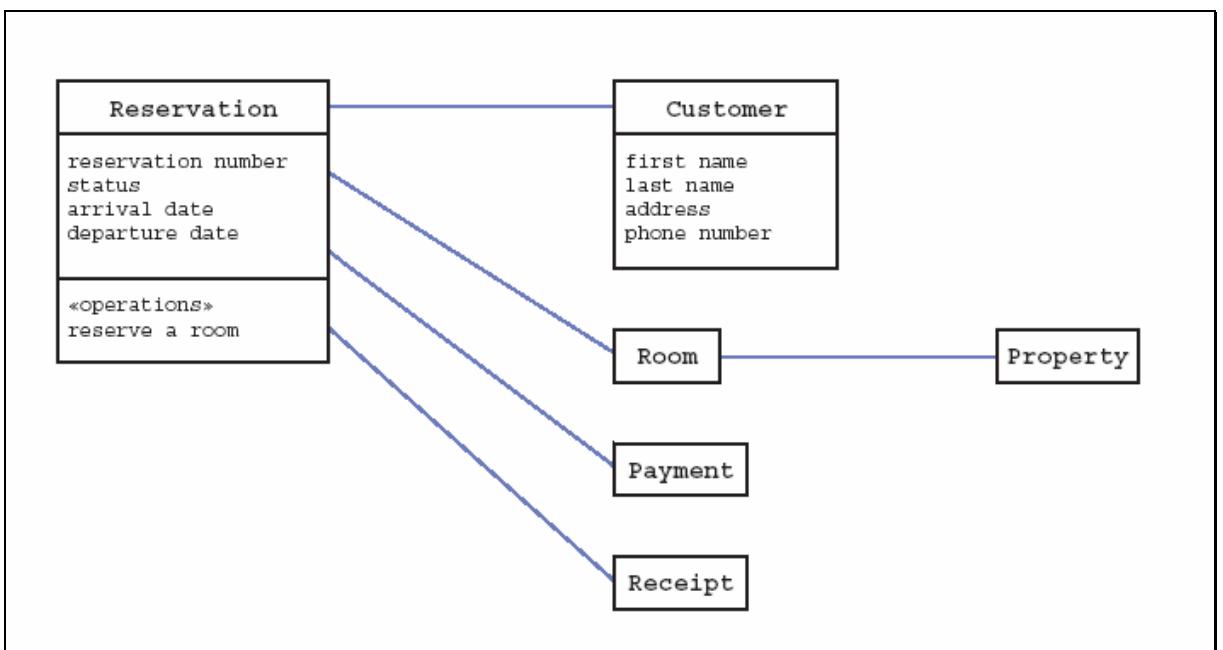
(2) Step 2 – Associations 그리기

다음 단계는 협력 클래스 사이의 **association** 선을 그리는 것입니다. 이 정보는 **CRC 카드**의 **Collaboration** 컬럼에서 얻을 수 있습니다.

다음은, 호텔 예약 시스템의 **key abstraction**에 대한 **association**을 표시한 것입니다.

[이미지]

– 협력 클래스 간의 **association** 그리기



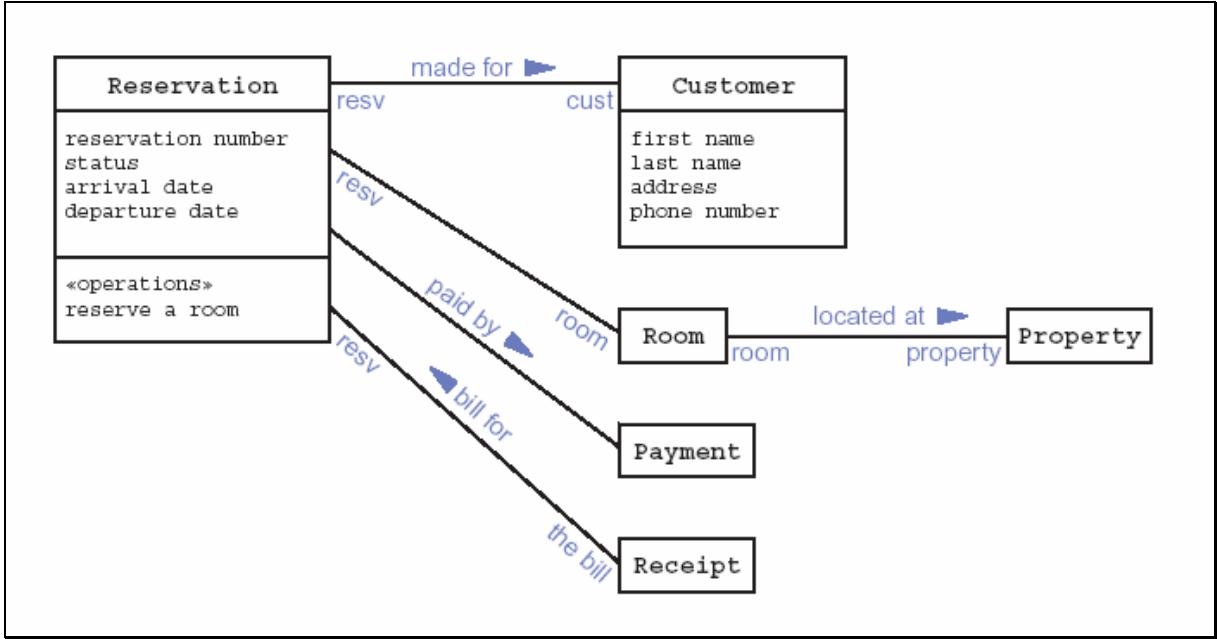
(3) Step 3 - Association과 Role 이름 작성

다음 단계는 **association**에 연관 관계의 이름과 **role** 이름을 작성하는 것입니다.

Association과 **role** 이름은 명백한 것이 좋습니다. 그래서 많은 분석가들이 이 단계를 무시합니다.

[이미지]

– **Relation**과 **Role** 이름 기록

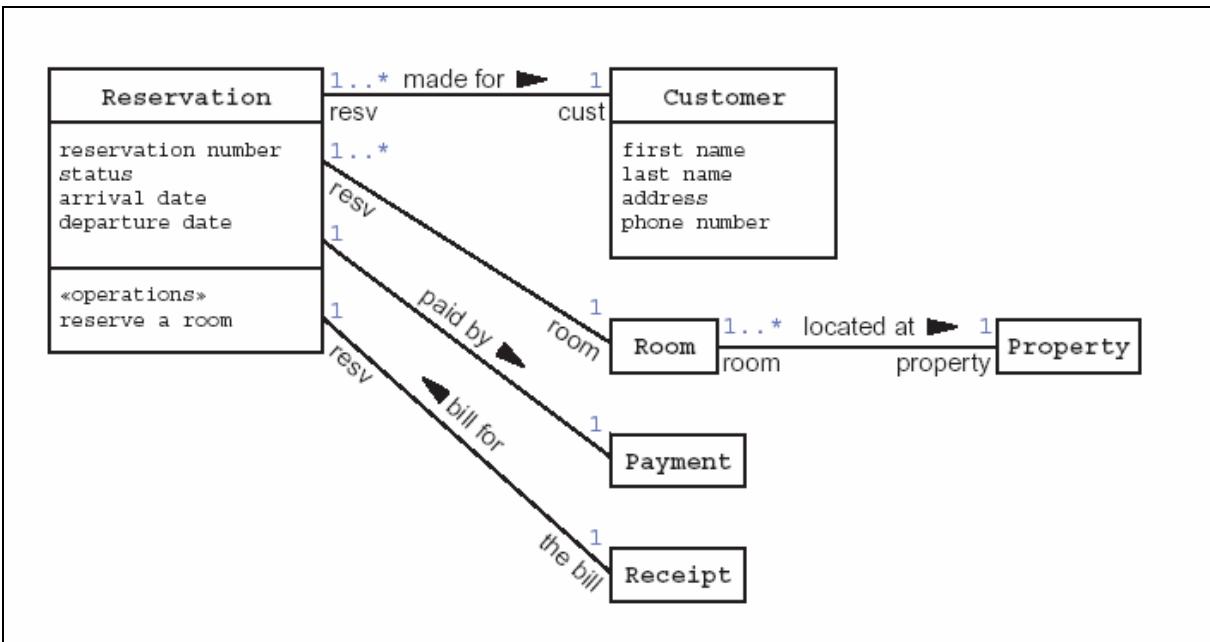


(4) Step 4 – Association Multiplicity 표시

다음 단계는 **association** 다중성을 추가하는 작업입니다. 다중성 정보는 대개 **CRC 카드** 분석 단계에서 파악되지 않습니다. 이 정보는 **SRS** 문서의 세부적인 **FR**이나 **domain** 전문가를 통해 알 수 있습니다. 예를 들어, **FR E1- 3**은 “하나의 예약은 단 한명의 고객과 관련 있습니다”라고 기술하고 있습니다.

[이미지]

– Association Multiplicity 기록



분석 단계에서 **association**의 다중성에 대해 실수를 하는 경우가 많습니다. ‘Object’ 다이

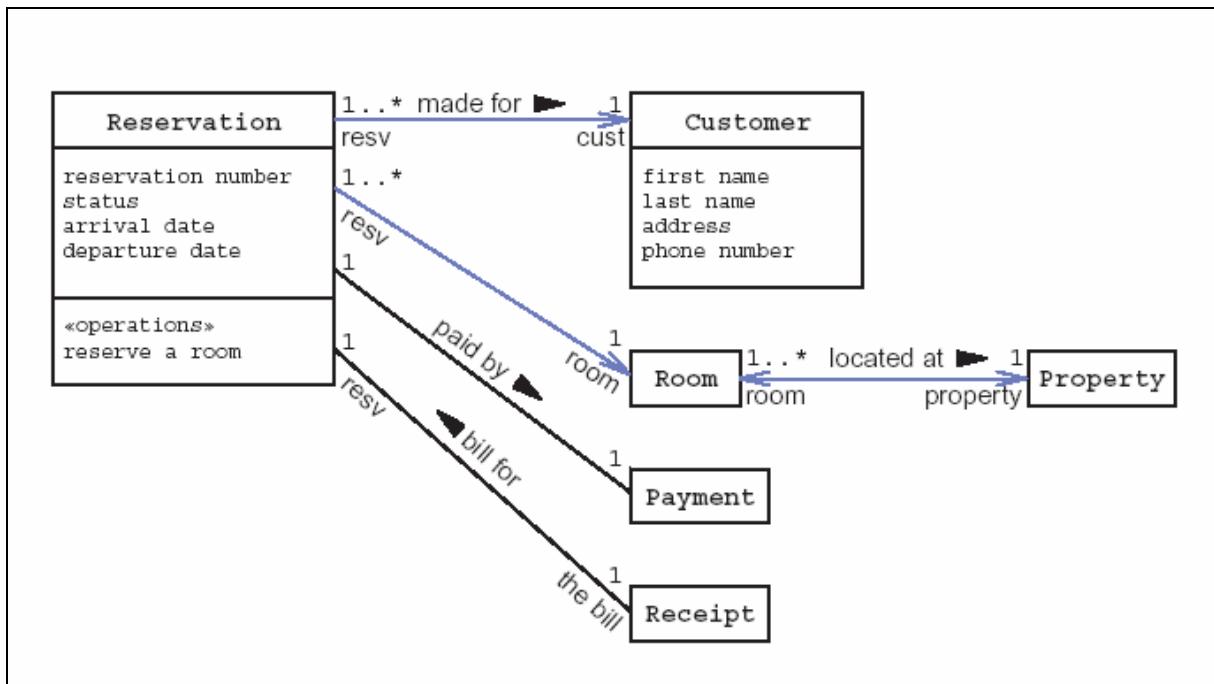
어그램을 통한 **Domain model** 검증' 부분에서 이 문제를 해결하는 법을 배우게 됩니다.

(5) Step 5 – Navigation Arrows 그리기

다음 단계는 **association**에 대한 진행 방향 화살표를 그리는 작업입니다.

[이미지]

– Association 진행 방향 기록



이것은 FR이 **Domain model**을 제약하거나 업무 분석가가 특정 연관관계의 진행방향을 정할 때만 이루어집니다. 예를 들어, **Room** 객체에서 예약에 대한 진행 방향은 필요하지 않습니다. 그래서 이 관계에서는 **Reservation**에서 **Room** 클래스를 가리키고 있습니다.

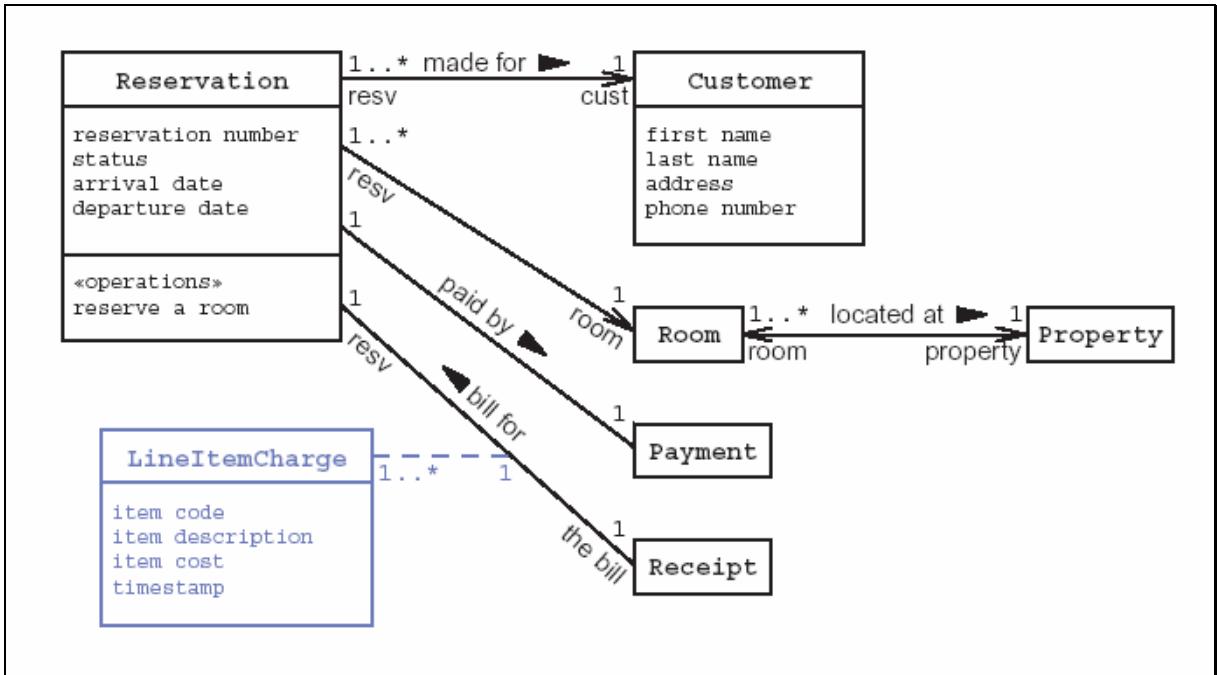
Room과 **Property**는 양방향을 가리키고 있습니다. 이것은 그 둘의 연관관계가 명시적으로 양방향임을 표시한 것입니다. 화살표가 없는 경우 묵시적으로 양방향임을 의미합니다.

(6) Step 6 – Association Classes 그리기

마지막 단계는 **association** 클래스를 추가하는 것입니다. 예를 들어, **LineItemCharge** 클래스는 예약과 예약 영수증 사이의 관계에 대한 정보를 기록하고 있습니다.

[이미지]

– Association class 기록



(7) Domain Model 검증

지금까지 **Domain model**을 생성하는 방법에 대해 알아보았습니다. 이 다이어그램이 **problem domain**을 정확하게 모델링한 것인지 어떻게 알 수 있습니까? 한가지 방법은 **Object** 다이어그램을 작성해서 **Object** 다이어그램이 **Class** 다이어그램에 맞는지 확인하는 것입니다. 이 내용은 본 모듈의 [주제 4. **Object** 다이어그램을 이용한 **Domain model** 검증]에서 다루겠습니다.

3. Object Diagram의 구성 요소

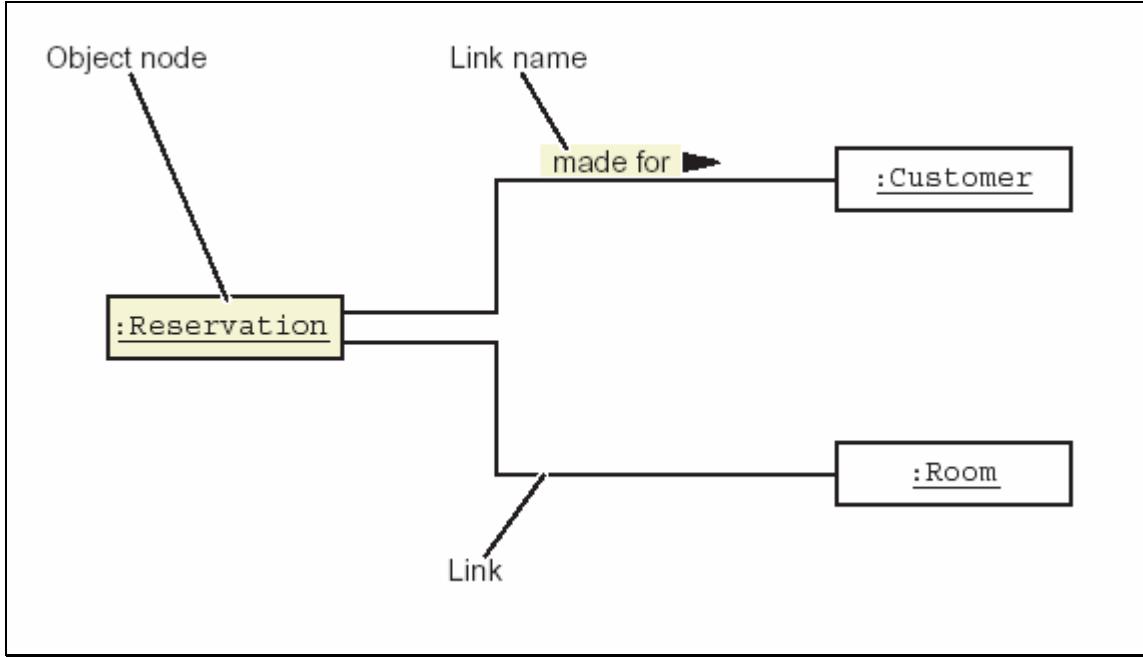
1) Object Diagram

“정적인 **Object** 다이어그램은 **Class** 다이어그램의 인스턴스로, 어느 한 순간의 시스템의 세부적인 상태를 스냅샷 처럼 보여줍니다.”(**UML, v1.4 page 3- 35**)

UML의 **Object** 다이어그램은 객체와 객체의 실행 **link**를 시각적으로 보여줍니다. 다음은 **Object** 다이어그램의 예입니다.

[이미지]

- **UML Object** 다이어그램의 구성 요소



[참고하세요]

- **link name**

Object 다이어그램에서는 **link** 이름은 대개 기록하지 않습니다.

2) Object Diagram의 구성 요소

(1) Object Nodes

일반적으로 객체 노드는 다음과 같이 이름과 데이터 타입을 포함합니다.

[이미지]

- **Object Node Types**

Object name without type	Type without a name	Type with a name
<u>Victoria</u>	<u>:Room</u>	<u>Blue:Room</u>

첫번째 노드는 데이터 타입이 없는 노드입니다. 이것은 객체는 가지고 있지만 객체의 타입을 아직 모른다는 것을 의미합니다. 이런 경우는 드뭅니다.

두 번째 노드는 객체의 타입을 보여주고 있습니다만, 이름이 없습니다. 이것은 매우 흔한 경우입니다.

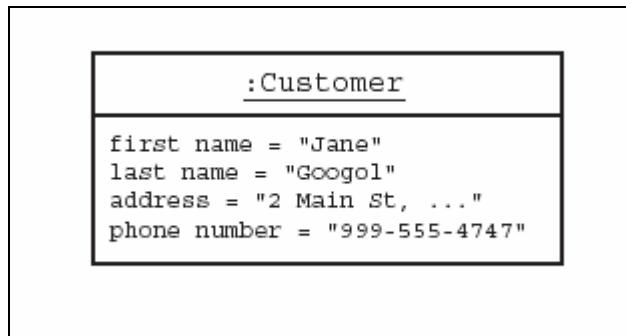
세 번째 노드는 이름과 데이터 타입을 모두 보여줍니다. 이름은 대개 객체를 참조하는 프로그램의 변수를 가리킵니다. 이름은 객체의 이름 속성을 나타낼 수도 있습니다.

객체 노드는 속성을 포함할 수도 있습니다. 객체 노드는 속성 칸을 갖고 있습니다. 각 속

성은 이름과 값을 = 기호를 사용해서 나타낼 수 있습니다.

[이미지]

- 속성을 갖는 Object Node



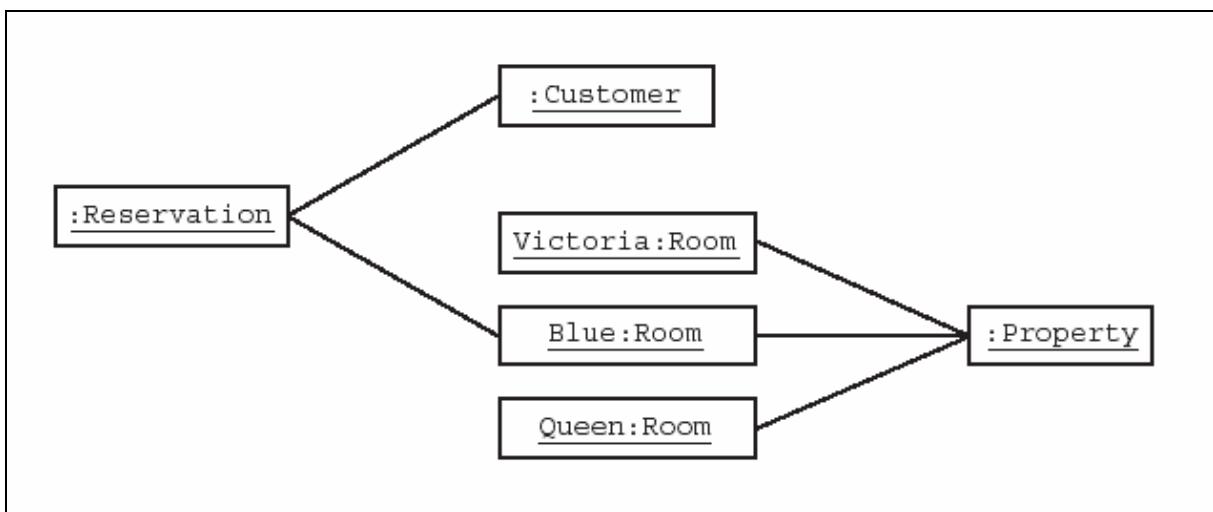
(2) Links

Object 다이어그램에서, 각 링크는 유일하고 각 노드에 **1대1**로 대응됩니다. 링크는 클래스 연관관계의 고유한 인스턴스입니다. 그 말은 각 링크는 단 두개의 객체를 서로 연결한다는 것을 의미합니다. 이 링크에 대해서는 모든 것이 **1 대 1**로 대응되기 때문에 **multiplicity**가 존재하지 않습니다.

예를 들어, **Property** 객체는 세 개의 서로 다른 방과 연관관계를 갖습니다. 이 때 각 관계는 고유의 링크를 갖습니다.

[이미지]

- 링크를 표현한 Object 다이어그램



4. Object Diagram을 통한 Domain Model의 검증

1) Domain Model 검증

Domain Model을 검증하기 위해서

① Domain model을 표현하는 하나 또는 그 이상의 유즈케이스를 고릅니다.

모든 유즈케이스에 대한 시나리오를 나타내는 Object 다이어그램을 작성하는 것이 이상적이지만, 시간이 너무 많이 소요됩니다. Domain model의 특징적인 부분을 담당하는 유즈케이스를 선택합니다.

② 선택한 유즈케이스에 대한 하나 또는 그 이상의 유즈케이스 시나리오를 고릅니다.

다양한 상황을 고려할 수 있는 시나리오를 고릅니다. Domain model의 제약 조건을 갖는 시나리오를 선택합니다.

예를 들어, 방 하나를 예약하는 “Create a Reservation” 유즈케이스는 간단하지만, 여러 개의 방을 예약할 경우 복잡해 집니다.

③ 각 시나리오에 대해 개별적이고 단계적으로 생각하여 언급된 객체와 데이터를 생성합니다.

시나리오는 구체적이기 때문에, 세부사항이 많습니다. 이 세부사항은 매우 유용합니다. 유즈케이스 시나리오를 사용해서 Object 다이어그램을 작성하는 방법을 다음 프레임에서 배우게 됩니다.

④ 연관관계의 제약조건이 지켜졌는지, Domain model에 대해 각 Object 다이어그램을 비교합니다.

다양한 유즈케이스 시나리오에 대한 Object 다이어그램들을 비교함으로써, Domain model이 정확하게 작성되었는지 확인할 수 있습니다.

2) Object Diagram 시나리오 작성

이제부터 유즈케이스 시나리오를 통해 Object 다이어그램을 작성하는 과정을 살펴보겠습니다.

시나리오의 각 문단은 시나리오 상에서의 시간적 단계를 나타냅니다. 각 단계에 따라 시간적 흐름에 맞춰 Object 다이어그램을 작성할 수 있습니다.

① 예약 추가 시나리오 1

① Step 1

Santa Cruz B&B의 예약 담당 직원 Medoca Sansumi는 HotelApp의 메인 화면을 띠워놓고, 고객의 전화를 기다리고 있습니다. 뉴욕시에 사는 고객 Ms.Googol에게서 전화가 왔습니다.
“여보세요, Jane Googol입니다. 새해 전날 예약을 하려고 하는데요..”
Medoca는 HotelApp의 메인 화면에서 “Create Reservation”을 선택합니다. 내용이 적혀 있지 않은 Reservation form이 화면에 나타납니다.

이 시나리오의 시작 단계에서는 고객이 방을 예약하길 원하는 호텔의 **Property** 객체에 대해 알 수 있습니다. 따라서 **Property** 객체 다이어그램을 그립니다.

[이미지]

– Step 1 - Create Reservation Scenario 1



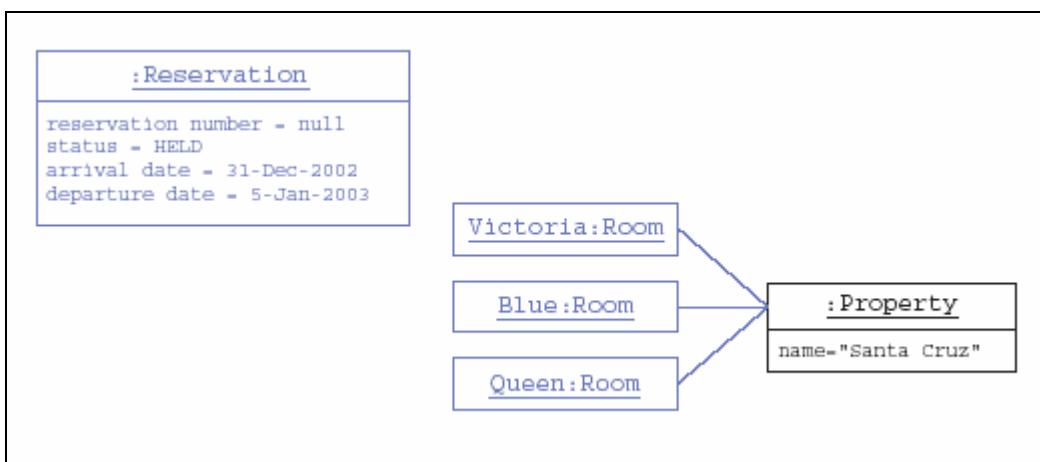
② Step 2

“언제 도착하십니까?” **Medoca**가 묻습니다. “**12월 31일**에요. **1월 5일** 까지 묵을 예정입니다.”라고 **Jane**이 말합니다. **Medoca**는 입력 폼에 날짜를 입력합니다. “어떤 방을 원하십니까?” **Medoca**가 묻자 “남편과 함께 가니까 싱글 룸이면 충분해요” **Jane**이 답합니다. **Medoca**는 예약 폼에서 “**single**”을 선택하고 검색합니다. 시스템은 **3개의 방**, **Victoria, Blue, Queen**이 사용 가능한 방이라고 알려줍니다.

시나리오의 이 단계에서는 시스템에, 도착날짜와 출발 날짜가 있는 **Reservation** 객체가 생성이 됩니다. 또한 예약 담당 직원은 **Bay View Property**의 방들을 검색할 수 있습니다. 검색 결과 세 개의 방이 리턴됩니다. 시나리오에 대한 **Object** 다이어그램은 이런 객체들을 포함하도록 확장됩니다.

[이미지]

– Step 2 - Create Reservation Scenario 1



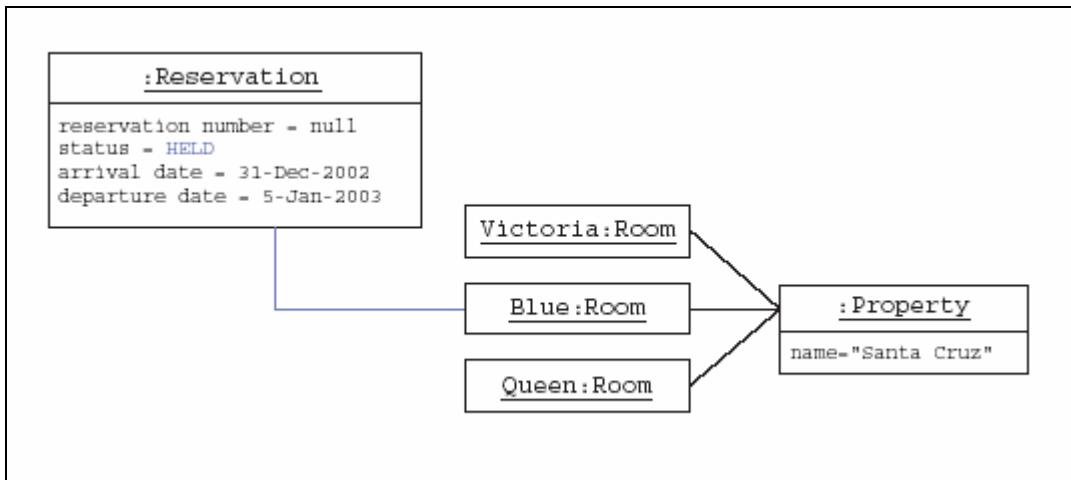
③ Step 3

“예 알겠습니다.” **Medoca**가 대답하고 **Blue room**을 선택합니다. 시스템은 **reservation**을 처리하고 예약이 “진행 중(HELD)”임을 표시합니다.

이 단계에서는 고객이 방을 선택합니다. 예약과 방을 연결하는 **association**을 추가합니다. 또한, 예약의 상태가 “진행 중(HELD)”으로 변경됩니다.

[이미지]

– Step 3 - Create Reservation Scenario 1



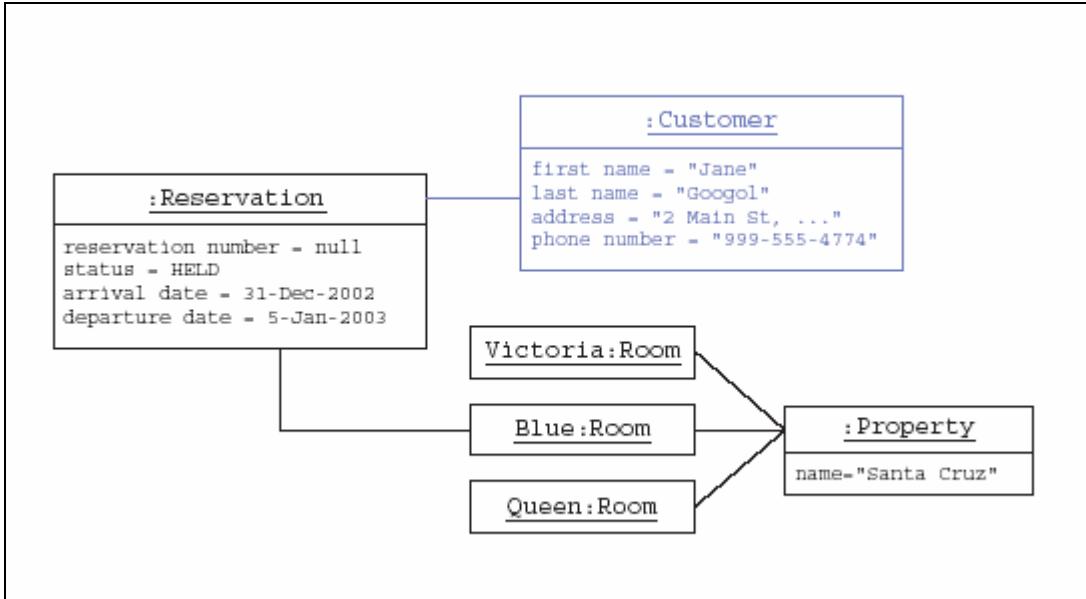
④ Step 4

Medoca는 **Jane**의 이름을 시스템에 입력합니다. **Ms. Googol**은 기존의 고객이므로, 시스템이 예약 품에 고객 정보를 채워줍니다.

이 단계에서는 **Customer** 객체가 수정되고, 예약과 연관관계를 갖게 됩니다.

[이미지]

– Step 4 – Create Reservation Scenario 1



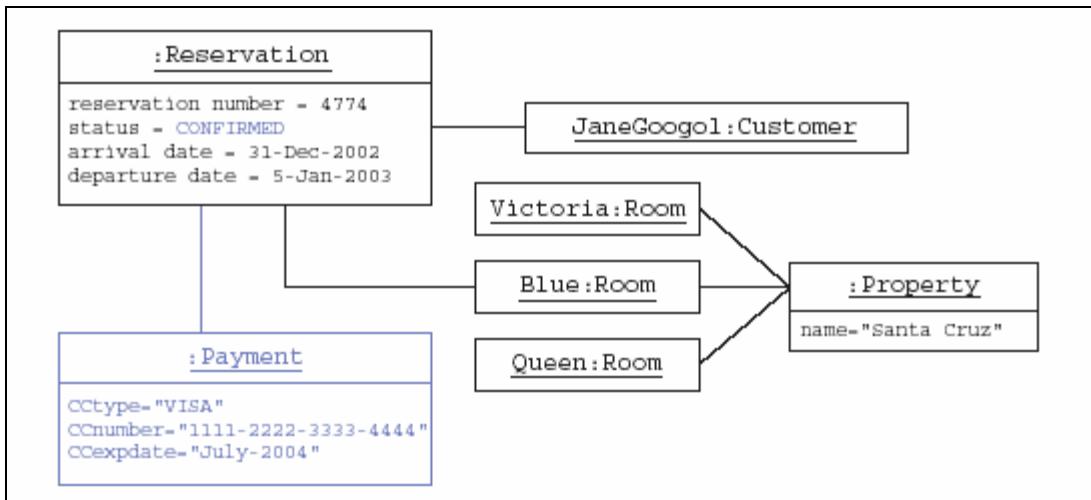
⑤ Step 5

“오늘 확인하시겠습니까?” Medoca가 묻습니다. “네. VISA 카드 번호는 **1111-2222-3333-4444**이고 유효 기간은 **2004년 7월입니다.**” Jane이 말하고, Medoca는 그 내용을 입력합니다. Medocar가 “Verify Payment”를 선택하자 5초 후에 신용카드가 인증되었음을 알려줍니다. 시스템은 예약의 상태를 “완료(**CONFIRMED**)”로 변경합니다.

이 단계에서는, 고객이 예약을 확인하기 위해 VISA 카드를 사용합니다. Payment 객체가 생성되고 예약에 대한 연관관계가 성립됩니다. 예약의 상태는 “완료(**CONFIRMED**)”로 변경됩니다.

[이미지]

– Step 5 – Create Reservation Scenario 1



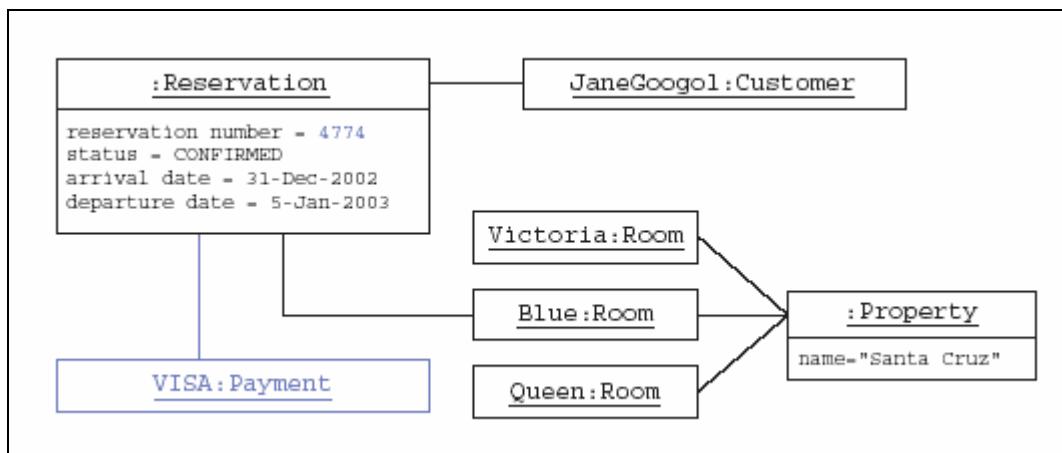
⑥ Step 6

Medoca는 **Jane**에게 시스템이 정해준 예약 **ID**를 알려주고, “오늘 또 요청할 것은 없습니까?”라고 묻습니다. **Jane**은 고마워하며 전화를 끊습니다. **Medoca**는 예약 폼을 닫고, 화면은 **HotelApp**의 메인 화면으로 돌아갑니다.

마지막 단계에서, 시스템은 예약 **ID**를 생성합니다.

[이미지]

– Step 6 – Create Reservation Scenario 1

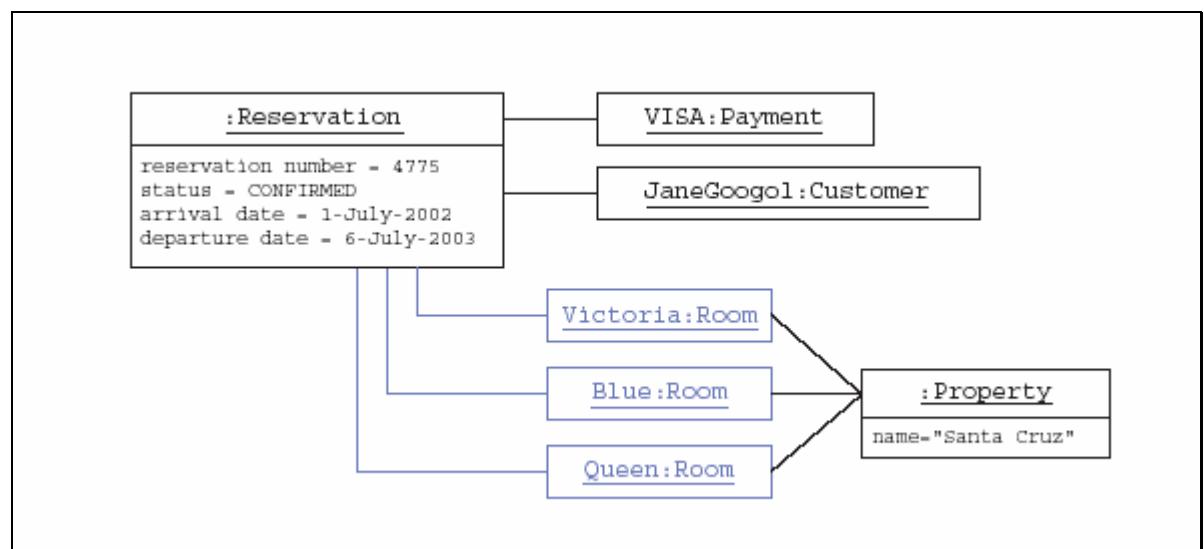


(2) 예약 추가 시나리오 2

Object 디어그램을 생성하는 유즈케이스 시나리오 검토 작업을 해봤습니다. 다른 유즈케이스 시나리오는 **Jane Google**이 예약 내용을 변경하는 경우입니다. 다른 가족들을 위해 세 개의 방을 예약하려고 합니다.

[이미지]

– Create Reservation Scenario 2



이 두 개의 시나리오를 비교함으로써, **Domain model**을 검증하는 방법과 모델의 오류를 발견하는 방법을 보게 됩니다.

3) Object Diagram을 통한 Domain model 검증

Domain model을 검증하기 위해, Class 다이어그램과 시나리오 Object 다이어그램을 비교합니다.

- 시나리오에서 언급된 속성이나 기능 중 Domain model에 없는 것이 있습니까?

시나리오의 데이터는 시나리오 Object 다이어그램의 일부 객체를 반영한 것입니다. 객체의 데이터 속성이 Class 다이어그램의 데이터 속성값에 대응되는지 확인해야 합니다.

- Object 다이어그램에 Domain model에 표현되지 않은 연관관계가 있습니까?

시나리오 Object 다이어그램의 모든 연관관계는 Domain model의 연관관계에 대응이 됩니다. Domain model에 존재하지 않는 연관관계가 있다면, 도메인 전문가에게 물어보고 다이어그램을 수정합니다.

- 다중성을 잘 표현했습니까?

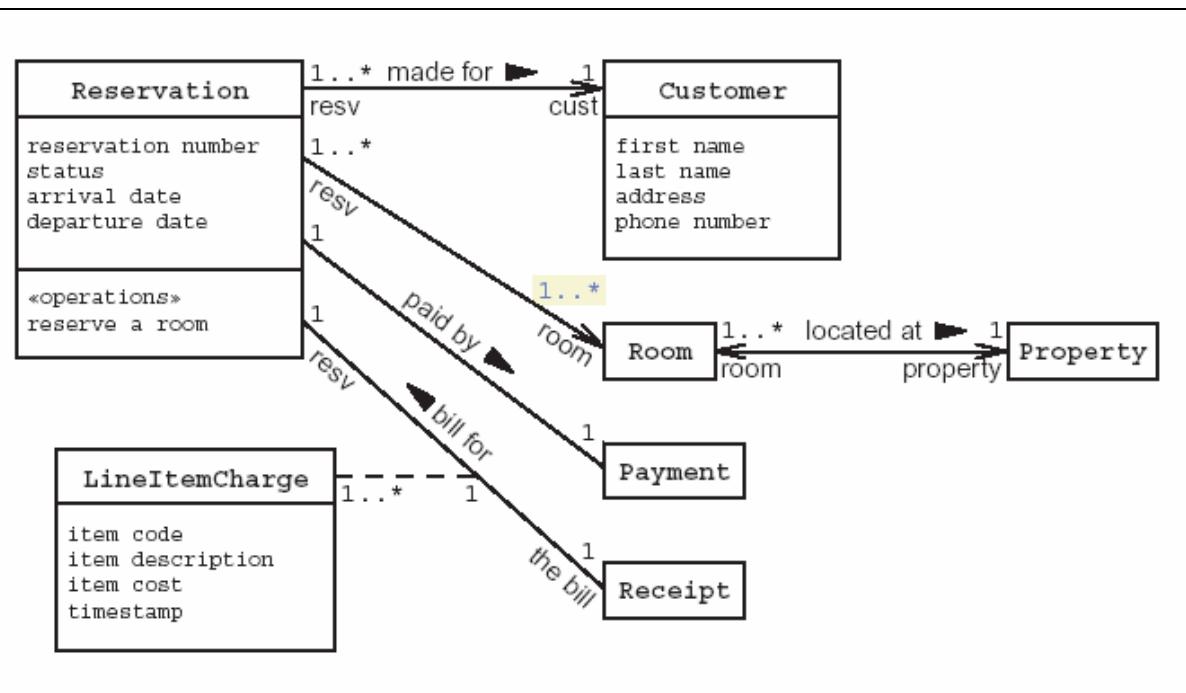
모든 Object 다이어그램에 대해, 주어진 타입에 대한 연관관계의 수를 세고, Domain model에서 구체화된 제약 조건에 숫자가 맞는지 확인합니다.

(1) 호텔 예약 시스템의 Domain Model 검증

첫번째 시나리오 Object 다이어그램은 Domain model을 검증하지만, 두 번째 시나리오는 Domain model의 문제를 파악합니다. 방을 예약할 때, 현재는 한 사람이 하는 하나의 예약은 하나의 방만 가능하도록 되어 있습니다. 두 번째 시나리오의 경우 하나의 예약에서 세 개의 방을 예약합니다. 따라서, Room 클래스의 multiplicity 라벨은 **1 .. ***로 바꿔야 합니다.

[이미지]

- 호텔 예약 시스템의 Domain Model 검증



과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원4 : 기능적 요구사항 설계

1 모듈 : Design Model 생성

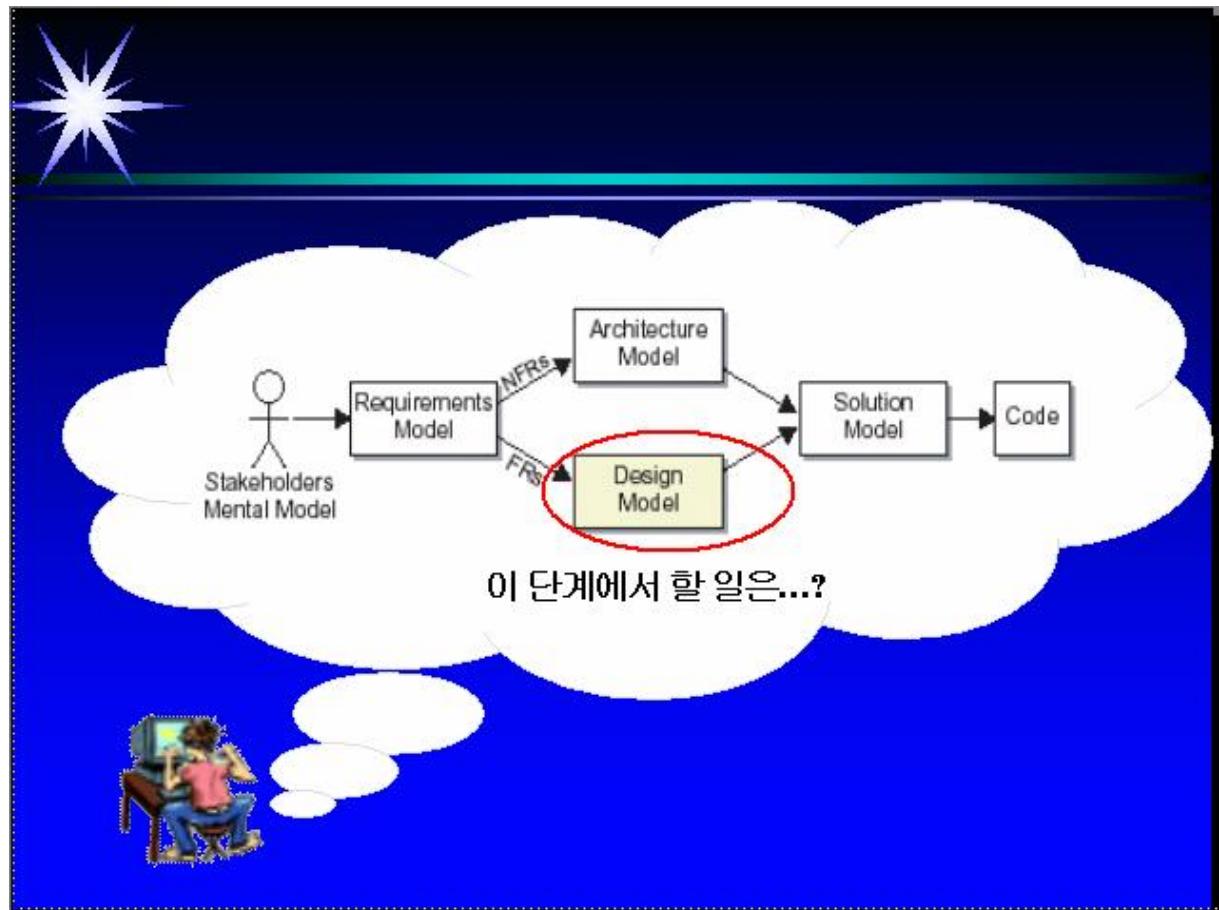
담당강사 : 전은수

■ 생각해봅시다 ■

Problem domain을 분석하는 것과 비즈니스 요구에 따른 소프트웨어를 만드는 것은 많은 차이가 있습니다. 과거에 진행했던 프로젝트에서는 소프트웨어 솔루션을 어떻게 설계했습니까? 분석과 설계의 차이에 대해 아십니까? 설계 단계에서 작성하는 **Collaboration** 다이어그램, **Sequence** 다이어그램 등은 어떻게 그릴 수 있을까요?

애니메이션

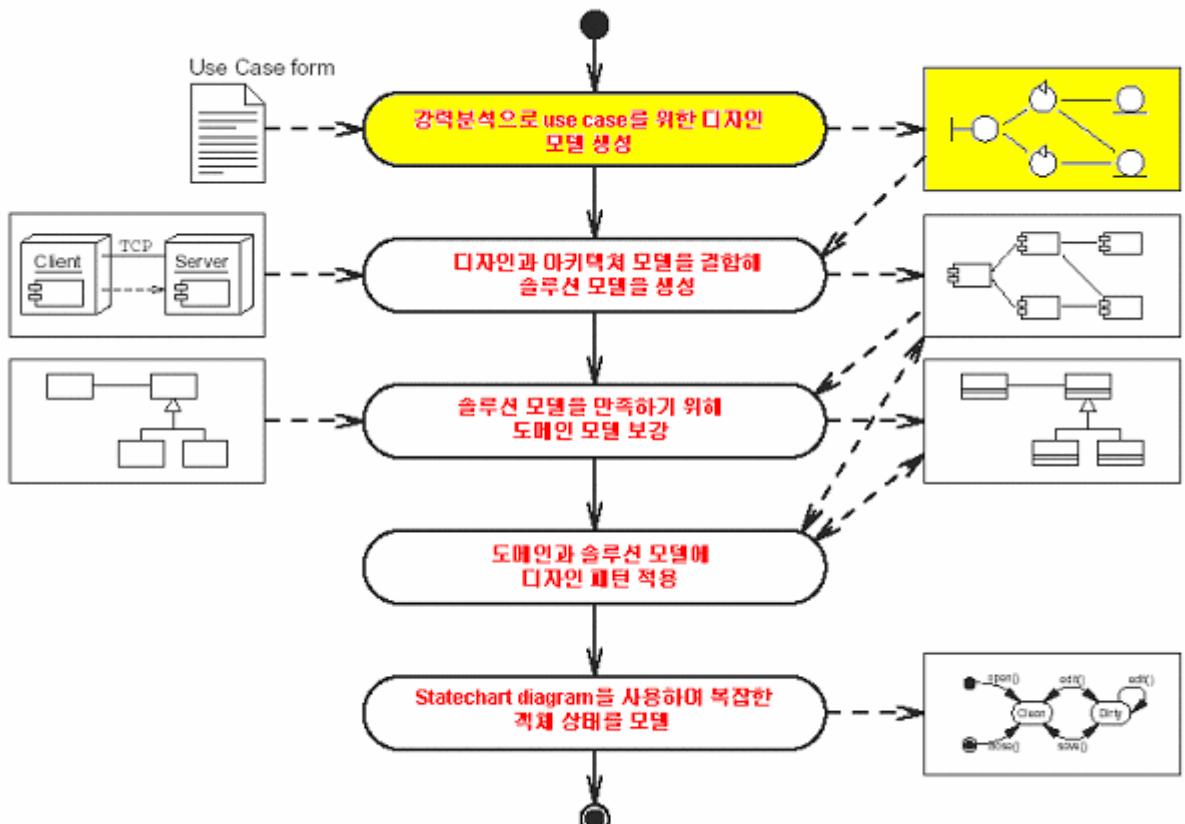
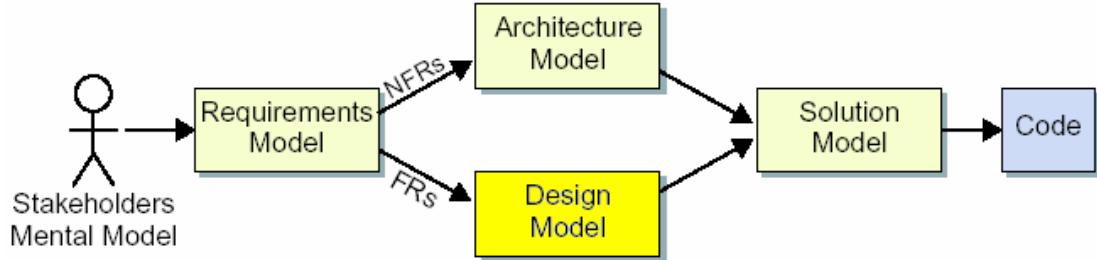
소프트웨어 디자이너 : “시스템에 대한 모든 요구 사항의 분석이 끝났네. 이제부터 본격적인 설계에 들어 가야 하겠지. 자, 그럼 시스템이 해야 할 일부부터 설계해 볼까? 이 단계를 디자인 모델링이라고 하던데, 구체적으로 어떤 일을 해야만 하는 거지?”



■ 학습하기 ■

참고하세요

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. 분석과 설계의 차이

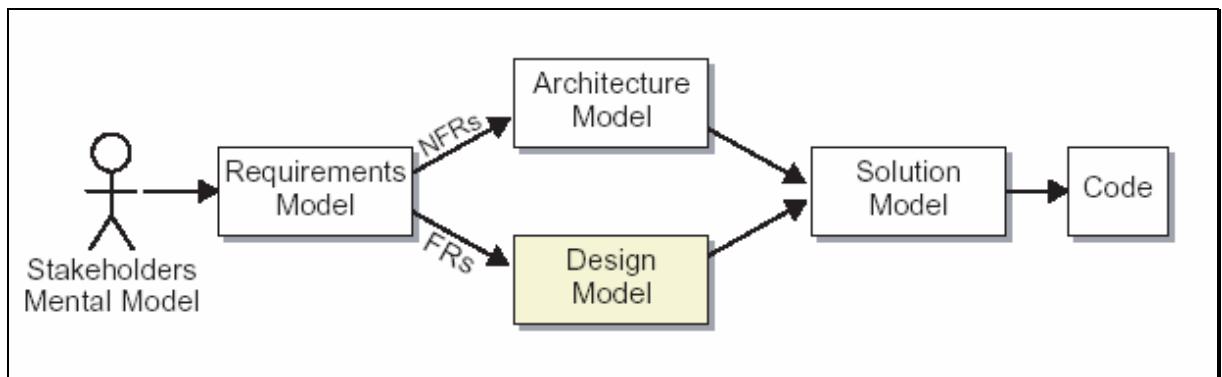
1) Design Model

(1) Design Model

Design model은 Requirements model의 기능적 요구 사항에서 생성되었습니다. Design model은 유즈케이스를 구현한 것입니다. Design model은 Solution model을 만들기 위해 Architecture model과 함께 합쳐집니다.

[이미지]

▫ Design model



(2) 분석과 설계의 차이

분석은 시스템이 반드시 지원해야 하는 것이 무엇(**what**)인가를 알아내는 것입니다.

분석을 통해 얻는 산출물은 다음과 같습니다.

- 유즈케이스
관련 사용자에 대한 시스템의 행동
- Domain model
비즈니스 프로세스에서 발생하는 객체

설계는 시스템이 비즈니스 프로세스를 어떻게(**how**) 지원하는지에 관한 것입니다.

설계의 구성 요소는 다음과 같습니다.

- Boundary (UI) Components

사용자와 직접 상호작용을 하는 컴포넌트입니다. 이 컴포넌트는 Service와 Entity Components의 접속 기능에 따라 시스템에 영향을 줍니다.

- Service Components

이 컴포넌트는 비즈니스 업무와 유즈케이스 워크플로우를 관리합니다.

- Entity Components

이 컴포넌트는 **Domain model**의 **entity**와 일치합니다.

2. Design Model의 구성 요소

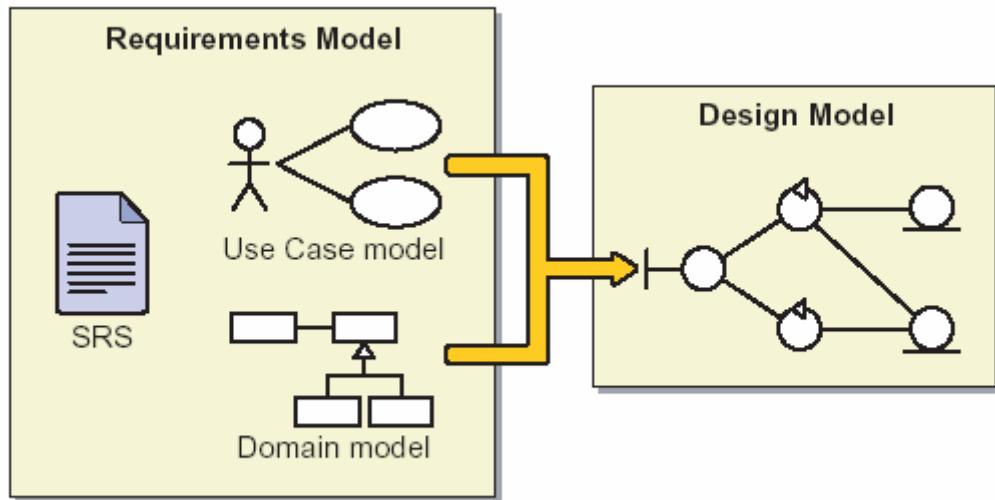
1) Robustness Analysis

앞에서 본 컴포넌트들(Boundary Components, Service Components, Entity Components)을 설계 컴포넌트라고 합니다.

강력 분석 또는 견고성 분석(Robustness Analysis)은 유즈케이스에서 유즈케이스를 지원하는 **Design model**을 이끌어내는 과정입니다.

[이미지]

n Requirements model에서 도출된 Design model



강력 분석은 **Design model**을 생성하는 프로세스에 부여된 이름입니다. 이 이름은 많이 사용되지 않지만, 그 프로세스는 **Jacobson**의 **OOSE** 방법론과 함께 깊은 뿌리를 가지고 있습니다.

Robustness analysis는 네 가지를 입력값으로 사용합니다.

- 유즈케이스

설계를 위해 유즈케이스를 하나 선택합니다. 한번에 여러 개의 유즈케이스에 대해서 강력 분석을 실시할 수 있습니다. 하지만 그렇게 하면 대규모 **Design model**이 만들 어질 것입니다. 시스템의 각 유즈케이스에 대해 하나씩 여러 개의 소규모 **Design model**을 만드는 것이 좋습니다.

- 그 유즈케이스에 대한 유즈케이스 시나리오

CRC 분석처럼 **Design model**을 만들기 위해서 유즈케이스 시나리오를 천천히 살펴보며 강력 분석을 진행합니다.

- 유즈케이스 Activity 다이어그램

유즈케이스 시나리오를 보는 대신 **Activity** 다이어그램을 사용할 수 있습니다.

- Domain model

Domain model은 **Design model**을 위한 **entity** 컴포넌트의 출처입니다. 모든 **Domain entity**가 사용되는 것은 아닙니다. 그래서 모든 **Domain entity**가 **Design model**에 나타나진 않습니다. 유즈케이스에 사용된 **entity**만 **Design model**에 나타납니다.

이 **Design model**은 대개 설계 컴포넌트와 함께 **UML Collaboration** 다이어그램으로 나타납니다.

2) Boundary Components

“**Boundary class(components)**는 시스템과 사용자(즉, 사용자와 외부 시스템) 사이의 상호작용을 모델링 하는데 사용됩니다.” (**Jacobson, Booch, Rumbaugh page 183**)

Boundary components는 다음과 같은 특징을 갖고 있습니다.

- UI 화면, 센서, 의사소통 인터페이스 등의 추상화

Boundary components는 인간 사용자나 외부 기계 **actor**와의 상호작용을 위해 존재합니다. 이 컴포넌트는 시스템 외부와 접촉하는 부분을 나타냅니다.

- High-level UI components

Boundary components는 **high-level**로 남아야 합니다. 텍스트필드, 라벨, 슬라이더, 라디오버튼 등과 같은 각각의 **GUI** 소품으로 나누지 말아야 합니다.

- 모든 boundary component는 최소한 하나의 actor와 관련이 있습니다.

Boundary 컴포넌트는 사용자와의 상호작용을 위해 존재합니다. **Boundary component**는 반드시 최소한 한 명의 **actor**와 연관 있어야 합니다. 여러 명의 **actor**와 관련 있을 수도 있습니다. 만일, **Boundary component**가 설계되고, 다른 **actor**와도 연관이 없다면 **component**는 쓸모 없는 **component**가 됩니다.

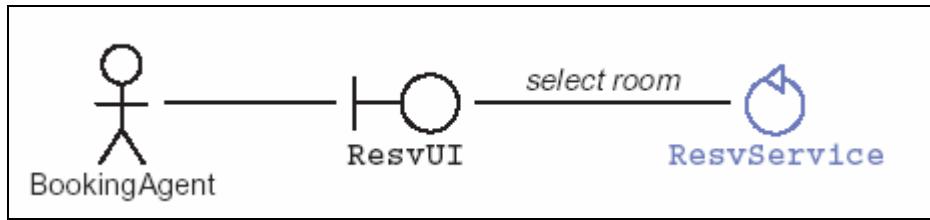
3) Service Components

“**Control(Service) classes(components)**는 다른 객체에 대한 조정이나 트랜잭션으로 나타나고, 특정 유즈케이스와 관련된 제어를 캡슐화하기 위해 사용기도 합니다.” (**Jacobson, Booch, and Rumbaugh page 185**)

Service component는 끝에 화살표가 달린 원으로 표시합니다.

[이미지]

n Service components 예



Jacobson은 이 컴포넌트를 제어 클래스(**Control classes**)라고 불렀지만, “제어”는 여러 가지 다른 뜻이 있습니다. 이 과정에서는 이 컴포넌트를 가리킬 때 “**Service**”라는 용어를 사용하겠습니다. 이 컴포넌트는 뒤에서 다루게 될 “일반 어플리케이션 컴포넌트”의 4 가지 기본적인 어플리케이션 컴포넌트와 관계가 있기 때문입니다.

대개 “**control**”이나 “**controller**”라는 용어는 사용자의 반응을 다루는 컴포넌트를 가리킵니다.

Service components의 특징은 다음과 같습니다.

- 제어 흐름을 조정합니다.

Service components의 목적은 다양합니다. 하지만, 대개 최상위에서는 이 컴포넌트가 시스템의 유즈케이스와 일치합니다. 즉, **Service component**는 유즈케이스 워크플로우를 다룹니다. **Service component**는 또한 여러 유즈케이스 사이의 일반적인 기능을 구현할 수 있습니다. **Service component**는 복잡한 알고리즘도 구현할 수 있습니다.

- **Boundary components**와 **entity components**의 워크플로우에서 차이가 있는 부분을 캡슐화합니다.

서로 다른 기능을 서로 다른 컴포넌트로 캡슐화하게 되면 시스템의 유지가 용이해지기 때문에, UI관련 워크플로우를 다른 워크플로우에서 분리해야 합니다.

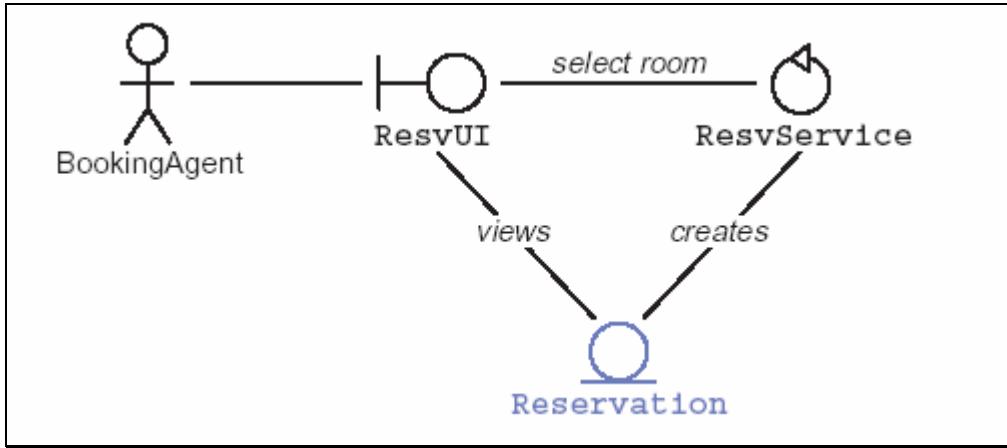
4) Entity Components

“**Entity components**는 장기적이고 때로는 영속적인 정보를 모델링 하기 위해 사용합니다.”(Jacobson, Booch, Rumbaugh page 184)

Entity component는 밑에 줄을 그은 원으로 표시합니다.

[이미지]

- n Entity component 예



Entity component는 다음과 같은 특징이 있습니다.

- **Entity**는 대개 **domain** 객체와 일치합니다.

대개 **Domain model**과 **Entity component**는 1대1로 매핑이 됩니다. 하지만 보조 **entity**는 강력 분석 단계를 거치면서 파악할 수도 있습니다.

- 대부분의 **entity**는 영속적입니다.

Entity는 영속적인 객체가 되고자 합니다. 하지만 이것이 고정된 법칙은 아니고, 일부 **entity**의 경우 다른 **entity**의 결과로 생성될 수도 있습니다.

- **Entity**는 복잡한 기능을 가질 수 있습니다.

Entity는 속성을 가져야만 한다고 생각할 수도 있습니다. 하지만, **Entity**는 복잡한 기능을 가질 수도 있습니다. **Service component**와 **Entity component**의 이런 기능 사이에는 **trade-off**가 있습니다. 일반적으로 어떤 기능이 여러 **entity**의 상호작용을 유도하는 경우, 그 **entity**는 아마 **Service component**일 것입니다.

5) Robustness Analysis Process(강력 분석 과정)

강력 분석 과정은 다음과 같습니다.

1. 유즈케이스를 선택합니다.

강력 분석은 유즈케이스로 이루어집니다. **Design model**로 개발할 하나의 유즈케이스를 선택합니다.

2. 그 유즈케이스의 기능을 만족하는 **Collaboration** 다이어그램을 작성합니다.

그 유즈케이스의 **Activity** 다이어그램을 사용해서 **Activity** 다이어그램을 분석하고, 다음의 작업을 합니다.

- 유즈케이스의 **Activity** 다이어그램을 지원하는 디자인 컴포넌트를 파악합니다.

사용자의 반응을 가리키는 작업이라면, **Boundary component**를 생성합니다. 또 시스템에 의해 이루어지는 작업이라면, **Service component**를 생성합니다. 그리고 **Domain** 객체의 기능을 가리키는 것이라면 **Entity component**를 생성합니다.

- 이 컴포넌트 사이의 관계를 그립니다.

Activity 다이어그램의 작업을 수행하기 위한 컴포넌트 사이의 관계를 파악합니다.

c. 연관관계에 라벨을 붙입니다.

그 작업을 수행하기 위해 컴포넌트 사이에 주고받게 될 메시지를 나타내는 라벨을 연관관계 선에 표시합니다.

3. Collaboration 다이어그램을 Sequence 다이어그램으로 변환합니다.

이 단계에서는 **Collaboration** 다이어그램을, 관련된 객체들을 시간순으로 보여주는 **Sequence** 다이어그램으로 변환할 수 있습니다.

이상은 강력 분석 과정에 대한 대략적인 설명입니다. 뒤에서 좀더 세부적으로 살펴보겠습니다. 다음 프레임은 **UML Collaboration** 다이어그램의 특징을 설명하고 있습니다.

3. Collaboration Diagram의 구성 요소

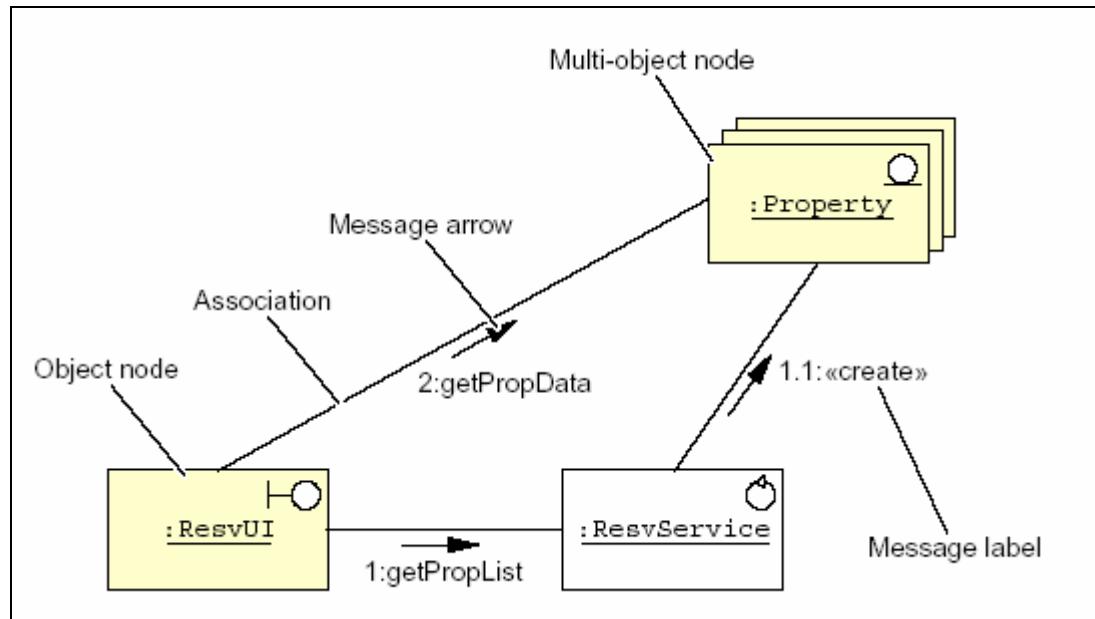
1) Collaboration Diagram

Collaboration 다이어그램은 “컴포넌트 사이의 의존도와 구성을 보여주는 다이어그램입니다”(UML v1.4, page B- 14)

Collaboration 다이어그램은 시스템의 객체, 그들의 관계, 그리고 객체 사이에 전송하는 메시지를 나타냅니다.

[이미지]

n Collaboration 다이어그램 예



메시지 화살표는 관련 객체로의 방향을 보여줍니다. 예를 들어, “1:getPropList”는 ResvUI 객체가 ResvService 객체로 getPropList 메시지를 보낸다는 것을 의미합니다.

메시지 화살표는

- 메소드 호출을 가리킵니다.

- 원격 메소드 호출을 가리킵니다.
- 비동기적 메시지를 가리킵니다.

Sequence 라벨은

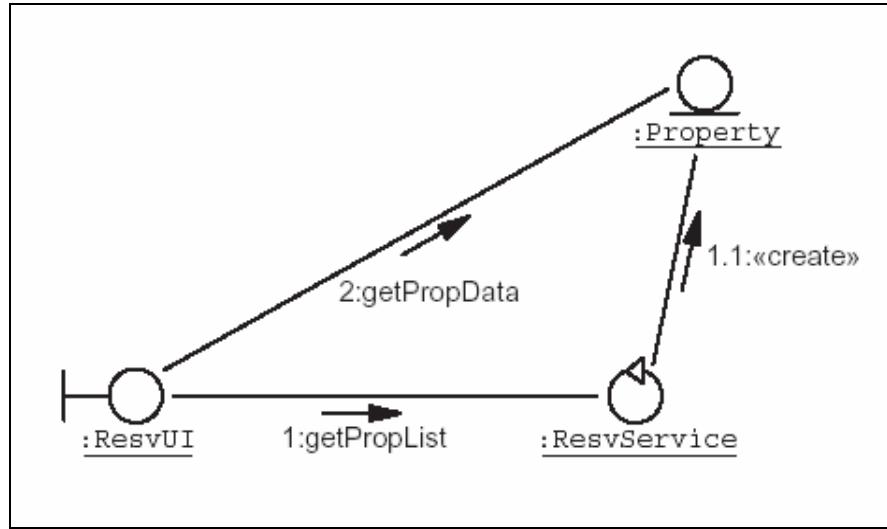
- 메시지의 순서를 가리킵니다.
라벨의 앞부분은 메시지의 계층적 순서를 의미합니다. 예를 들어, “1.1:<<create>>”의 “1.1”은 메시지가 보내지는 순서를 의미합니다. <<create>>메시지는 **getPropList** 다음에 호출되고, **getPropData** 전에 호출됩니다.
- 메시지가 발생시킬 작업을 가리킵니다.
라벨의 뒷부분은 실제로 보낼 메시지를 의미합니다. 예를 들어, “1:getPropList”的 “getPropList” 부분은 **ResvService** 객체에서 **getPropList** 메소드를 호출한다는 것을 의미합니다.

Multi-object는 같은 타입의 관련된 객체의 집합을 의미합니다. 이 다이어그램에서는 시스템이 실제로 여러 타입의 객체들을 다루는 것을 볼 수 있습니다.

강력 분석에서는 노드에 아이콘을 사용한 객체 노드의 스테레오타입으로, **Collaboration** 다이어그램의 다양함을 볼 수 있습니다.

[이미지]

n 스테레오타입 아이콘의 사용



2) Robustness Analysis

(1) 개요

강력 분석은 다음 순서대로 수행할 수 있습니다.

1. 적당한 유즈케이스를 선택합니다.
2. **Collaboration** 다이어그램에 **actor**를 표시합니다.
3. **유즈케이스(Activity 다이어그램)**을 분석합니다. 각 유즈케이스에 대해
 - a. **Boundary component**를 파악하고 추가합니다.
 - b. **Service component**를 파악하고 추가합니다.

- c. Entity component를 파악하고 추가합니다.
- d. 이런 컴포넌트들 사이의 연관관계(**association**)를 그립니다.
- e. 각 컴포넌트가 유즈케이스의 상호작용을 만족하기 위해 수행하는 작업을 기록합니다.

다음 프레임부터 “**Create a Reservation**” 유즈케이스에 대하여 강력 분석을 수행하는 세부적인 내용을 보도록 하겠습니다.

(2) Step 1 – 유즈케이스 선택

Design model의 범위를 한정하기 위해, 강력 분석을 하나의 유즈케이스에 대해 수행해야 합니다. 하나 이상의 유즈케이스를 분석할 수도 있지만, **Collaboration** 다이어그램이 너무 커지게 됩니다. 여기서는 예제로 “**Create a Reservation**” 유즈케이스를 선택하겠습니다.

보충

n 이벤트 플로우

다음은 “**Create a Reservation**” 유즈케이스에 대한 이벤트 플로우를 나열한 것입니다.

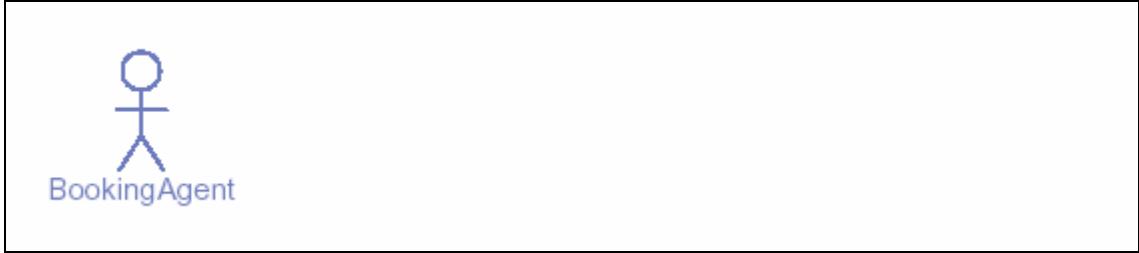
1. 고객이 예약 담당 직원에게 전화를 합니다.
2. 예약 담당 직원이 “예약하기” 아이콘을 누릅니다.
3. 예약 담당 직원이 예약 화면에 여러 내용을 입력합니다.
 - 3.1 예약 담당 직원이 도착 날짜와 떠날 날자를 입력합니다.
 - 3.2 예약 담당 직원이 객실 종류를 입력합니다.
4. 예약 담당 직원이 “찾기”버튼을 누릅니다.
- ...
11. 예약 담당 직원이 고객의 이름을 입력합니다.
12. 예약 담당 직원이 “찾기”버튼을 누릅니다.
13. 만약 해당 고객이 없으면
 - 13.1 예약 담당 직원이 주소 정보를 입력합니다.
 - 13.2 예약 담당 직원이 전화번호를 입력합니다.
 - 13.3 예약 담당 직원이 “새 고객 추가”버튼을 누릅니다.
14. 만약 해당 고객이 있으면
 - 14.1 시스템이 고객의 정보를 보여줍니다. 같은 이름의 고객이 여러 명일 수 있으므로 리스트를 보여줍니다.
 - 14.2 예약 담당 직원이 해당 고객을 선택합니다.
 - 14.3 시스템이 해당 고객의 정보를 보여줍니다.
- ...
21. 시스템이 예약을 저장하고 예약번호를 보여줍니다.
22. 예약 담당 직원이 “마침”버튼을 누릅니다.

(3) Step 2 - 다이어그램에 Actor 두기

모든 유즈케이스는 유즈케이스를 사용하는 **actor**에서부터 시작합니다. 예를 들어, “Create a Reservation”의 **action 1**은 “고객이 예약 담당 직원에게 전화를 합니다”입니다. 따라서 예약 담당직원(**BookingAgent**)이 이 유즈케이스의 **actor**가 됩니다. **Collaboration**의 왼쪽 끝에 **BookingAgent actor**를 둡니다.

[이미지]

- Collaboration 다이어그램에 actor 배치



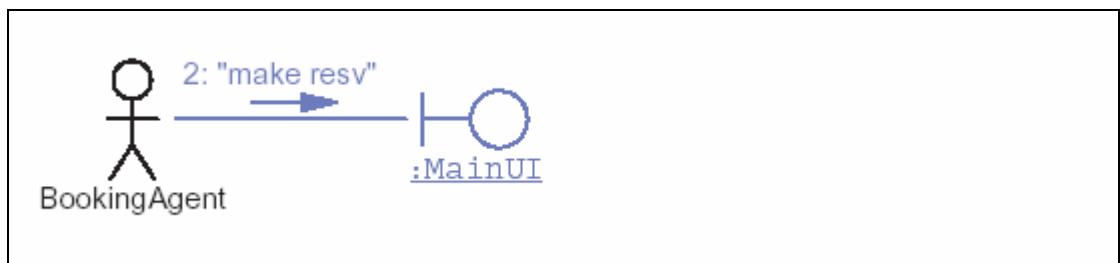
(4) Step 3a – Boundary components 파악

이벤트 플로우의 **action**을 분석해서, 설계 컴포넌트의 세가지 타입을 파악해야 합니다.

Boundary component는 쉽게 알 수 있습니다. 이 컴포넌트는 **actor**가 시스템과 상호작용을 할 수 있게 합니다. 시스템과 어떤 작업을 하는 **actor**가 있는 **action**은 하나 이상의 **Boundary component**를 필요로 합니다. 예를 들어, 예약 담당직원이 예약을 추가하려면 시스템에 ‘예약 하기’아이콘을 선택해야 합니다.

[이미지]

- BookingAgent와 MainUI 컴포넌트의 상호작용

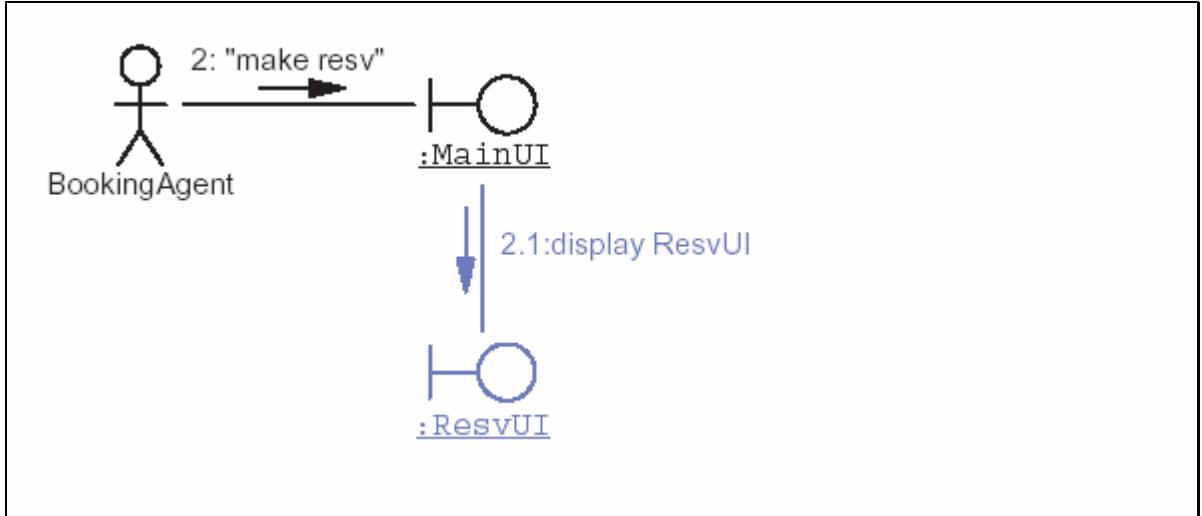


이 다이어그램은 **actor**와 **MainUI** 사이의 관계를 보여줍니다. 연관관계 선 위에 이벤트 흐름의 내용을 참고해서 메시지 화살표와 라벨을 추가했습니다.

다음에는, **MainUI** 컴포넌트가 예약 폼을 실행시킵니다. 이것도 추가적인 **Boundary component**인 **ResvUI**를 필요로 합니다.

[이미지]

n MainUI 컴포넌트에 의한 ResvUI 컴포넌트 실행



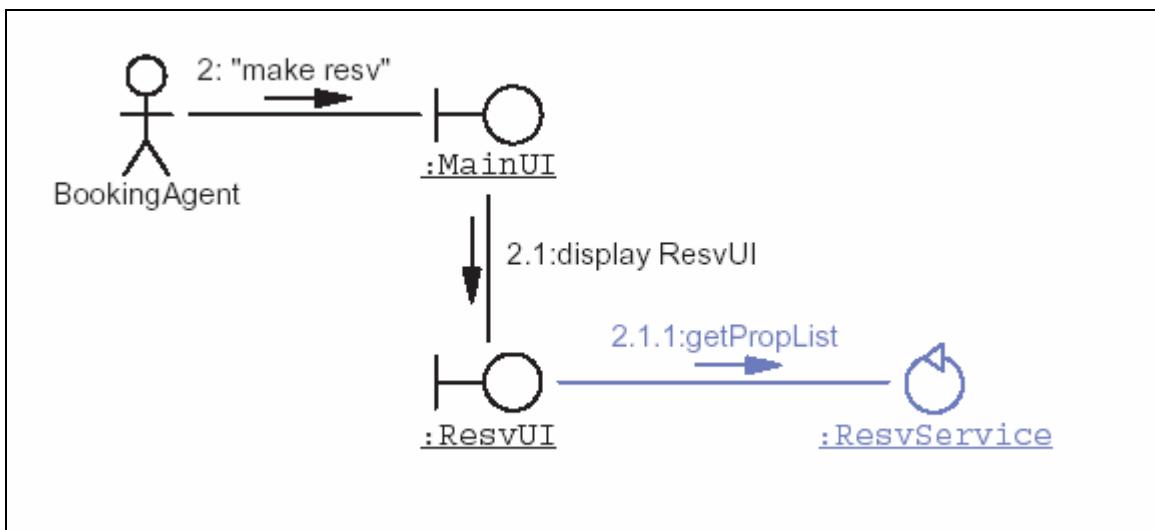
이런 메시지들은 이벤트 플로우에 자세히 나타나 있지 않습니다. 유즈케이스의 목적을 자세히 이해하고 그에 합당한 내용을 알아내야 합니다. **Boundary component**는 UI 프로토 타입의 화면에 의해 이미 정의되기 때문에, 강력 분석 단계에서는 사용자 인터페이스의 프로토타입이 유용할 수 있습니다.

(5) Step 3b – Service components 파악

간혹, 시스템이 어떤 **action**을 해야 할 수가 있습니다. 이것은 **Service component**로 나타낼 수 있습니다. 예를 들어, ResvUI화면이 나타날 때, ResvUI 폼에 있는 **Property** 객체의 내용이 변경되어야 합니다. 따라서, **service**는 그 작업을 수행해야 합니다.

이미지

n ResvUI 컴포넌트에 의한 ResvService 컴포넌트의 **Property** 리스트를 요청



ResvService 컴포넌트는 “**Create a Reservation**” 유즈케이스의 가장 중심이 되는 작업

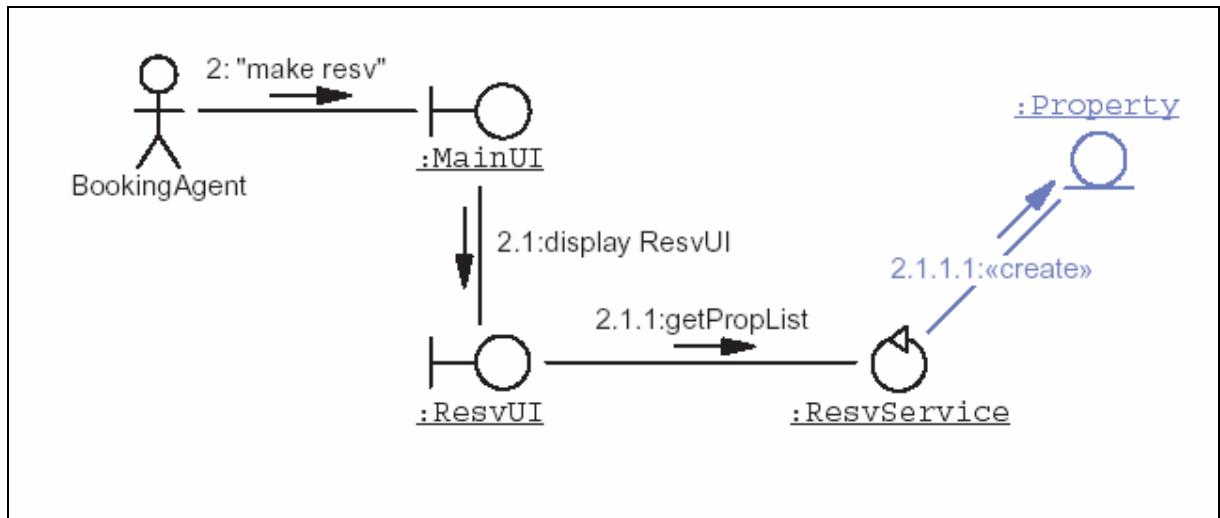
을 지원합니다. 하지만, 필요에 따라 다른 객체와 **ResvUI**가 상호작용을 할 수도 있습니다.

(6) Step 3c – Entity component 파악

경우에 따라, 시스템이 **entity**를 생성하고 조정해야 할 수도 있습니다. 이런 경우는 **Entity component**를 사용합니다. 예를 들어, **ResvService** 컴포넌트는 **Property** 객체를 수정합니다. 데이터베이스에서 수정된 값은 **Design model**에 반영되지 않습니다. **ResvService** 컴포넌트가 **Property**의 객체를 생성해서 보여주어야 합니다.

[이미지]

n ResvService 컴포넌트에 의한 Property 리스트를 수정



[참고하세요]

이 단계에서, **Design model**은 데이터베이스와의 작업에 대한 부분을 효과적으로 보여주지 못합니다. 지금의 경우는 데이터베이스와 관련된 작업을 수행하기 위해 서비스 컴포넌트가 생성됩니다. 데이터베이스 통합에 대한 부분은 [단원 5. 비기능적 요구사항 구축 모듈 5. Resource & Integration Tier의 아키텍쳐 모델 생성]에서 배우게 됩니다.

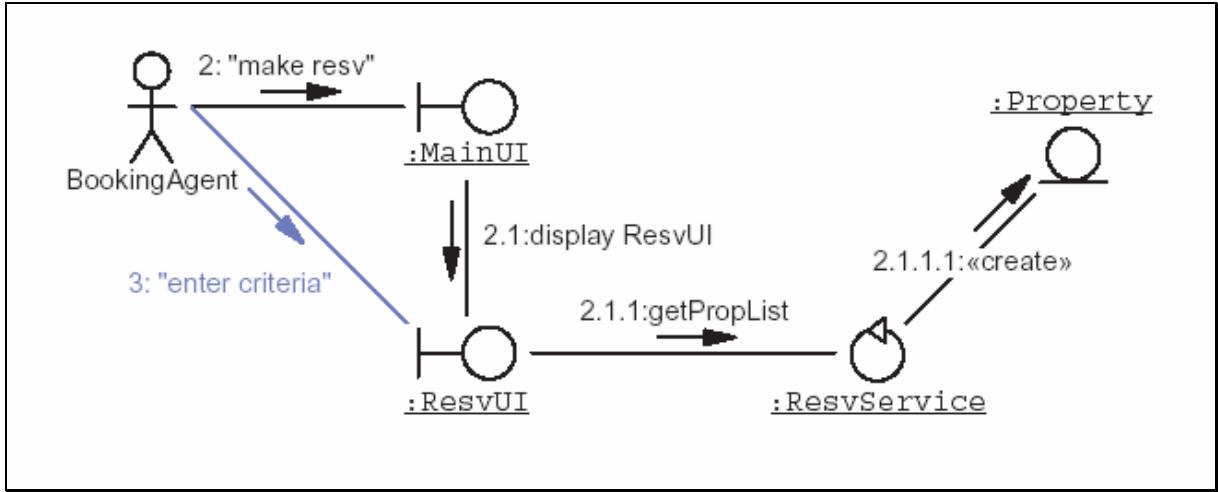
(7) Activity 디어그램의 모든 Actor 분석

앞에서는 유즈케이스 이벤트 플로우의 단일 작업에 대한 강력 분석을 봤습니다. 이벤트 플로우의 모든 유즈케이스에 대해 이런 작업을 반복해야 합니다. 이 디어그램은 유즈케이스를 완벽하게 다루는 **Design model**을 작성하는데 도움이 됩니다.

예를 들어, 이벤트 플로우의 **action 3**은 방을 선택할 때 **actor**가 검색 범위를 입력할 수 있습니다. 이것은 **actor**와 **ResvUI** 컴포넌트 사이의 반복 작업을 필요로 합니다.

[이미지]

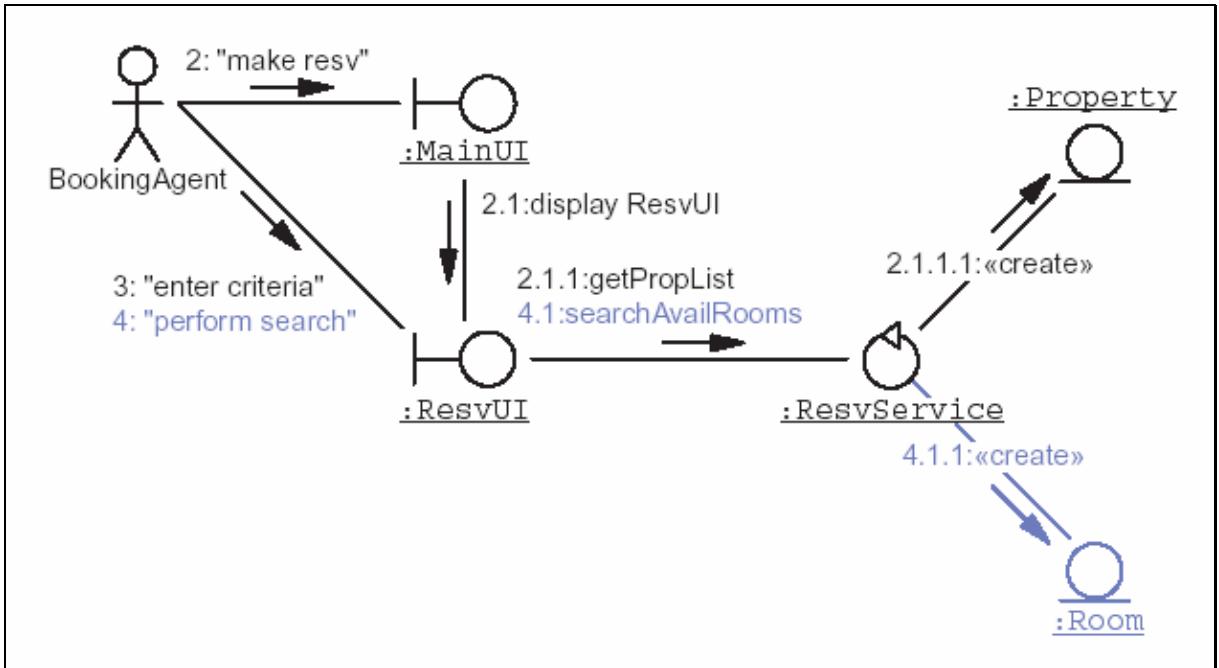
n Action3에 대한 Design Model 구체화



이벤트 플로우의 **action 4**의 경우 **actor**는 **ResvUI**에게 **action 3**에서 입력한 범위에 대한 검색 작업을 요청합니다. **ResvUI**는 그 작업을 수행하기 위해 **ResvService** 컴포넌트를 사용합니다.

[이미지]

n action 4에 대한 Design Model 구체화



(8) Collaboration Design을 Sequence 다이어그램으로 변환하기

강력 분석에 대한 다른 관점을 제공하기 위해, **Collaboration** 다이어그램을 **Sequence** 다이어그램으로 변환할 수 있습니다. 이 다이어그램은 개발자에게 좀 더 유용합니다. 많은 UML 툴들이 **Collaboration diagram**을 **Sequence diagram**으로 상호 전환하는 자동 기능을 제공합니다.

4. Sequence Diagram의 구성 요소

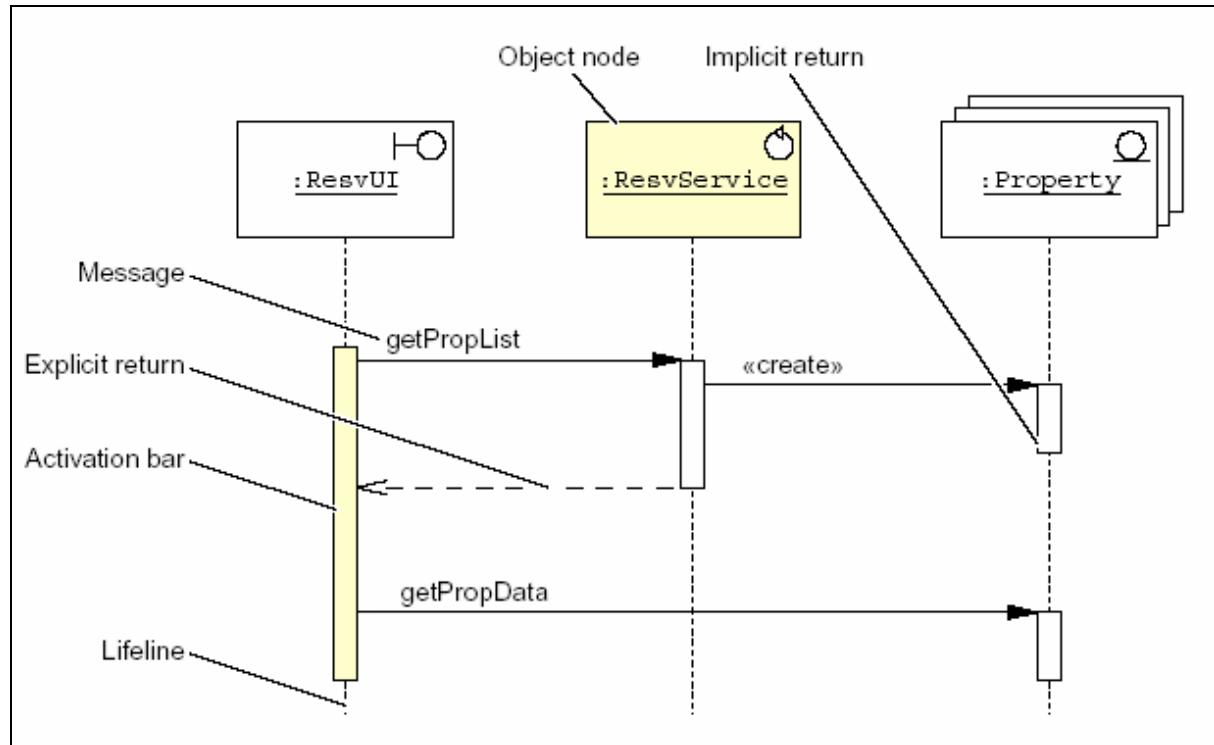
1) Sequence 다이어그램

Sequence 다이어그램은 “객체의 상호 작용을 시간 순서에 따라 정렬하여 보여주는 다이어그램”입니다.(UML V1.4, page B- 17)

Sequence 다이어그램은 시스템의 객체와 시간 순으로 구성된 객체들 간의 메시지 송신을 보여줍니다.

이미지

n Sequence 다이어그램 예



협력 관계의 객체들은 다이어그램의 상단에 컬럼을 이루고 있습니다. 시간은 위에서 아래로 흘러갑니다.

객체 아래 있는 선은 객체의 생명주기나 경과 시간을 가리킵니다. 만일, 객체가 Sequence 다이어그램 전반에 걸쳐 존재한다면, 그 객체의 선은 다이어그램 끝까지 이어질 것입니다.

Activation bar는 어떤 작업에서 그 객체를 사용하고 있음을 가리킵니다. 대개 이것은 객체에 메시지가 전달되었을 때 발생합니다. 예를 들어, **retrievePropList** 메시지는 **ResvService** 객체에 전달됩니다. 이 시점에서 **activation bar**는 메시지 화살표가 도착했을 때 시작해서, **ResvService** 객체가 일을 마칠 때 까지 이어집니다. 화살표로 표시된 점선은 메시지가 리턴될 방향을 명시적으로 가리켜 줍니다.

리턴 방향 화살표를 명시적으로 표시하지 않은 경우는 **activation bar**의 끝이 메시지 처리가 완료되었을 때를 가리킵니다.

2) Sequence 다이어그램을 사용한 Design Model

(1) 개요

강력 분석은 **Collaboration** 다이어그램 형태의 **Design model**을 생성합니다. 이것은 읽기에 불편할 수 있습니다. 대안으로 시간 순으로 기록한 **Design model**인 **Sequence** 다이어그램을 사용할 수 있습니다.

Collaboration 다이어그램은 다음과 같은 과정을 거쳐 **Sequence** 다이어그램이 될 수 있습니다.

3.2.1 activity의 순서를 반영하여 **Sequence** 다이어그램의 상단에 협력 관계에 있는 객체들을 배열합니다.

이 단계는 **Sequence** 다이어그램의 컬럼을 정하는 단계입니다. 이후에 분석된 내용에서 객체가 추가적으로 필요할 경우 다이어그램의 오른쪽에 추가합니다.

3.2.2 첫번째 activity에 메시지 링크와 activation bar를 추가합니다.

이 단계에서는 **Collaboration** 다이어그램의 메시지 전송 순서를 따라서 메시지 화살표를 추가하고, 각 메시지에 **activation bar**를 추가합니다. **Collaboration** 다이어그램의 중첩된 메시지와 중첩된 **activation bar**는 일치해야 합니다.

3.2.3 객체간의 메시지 전송이 완료될 때까지 2단계의 작업을 반복합니다.

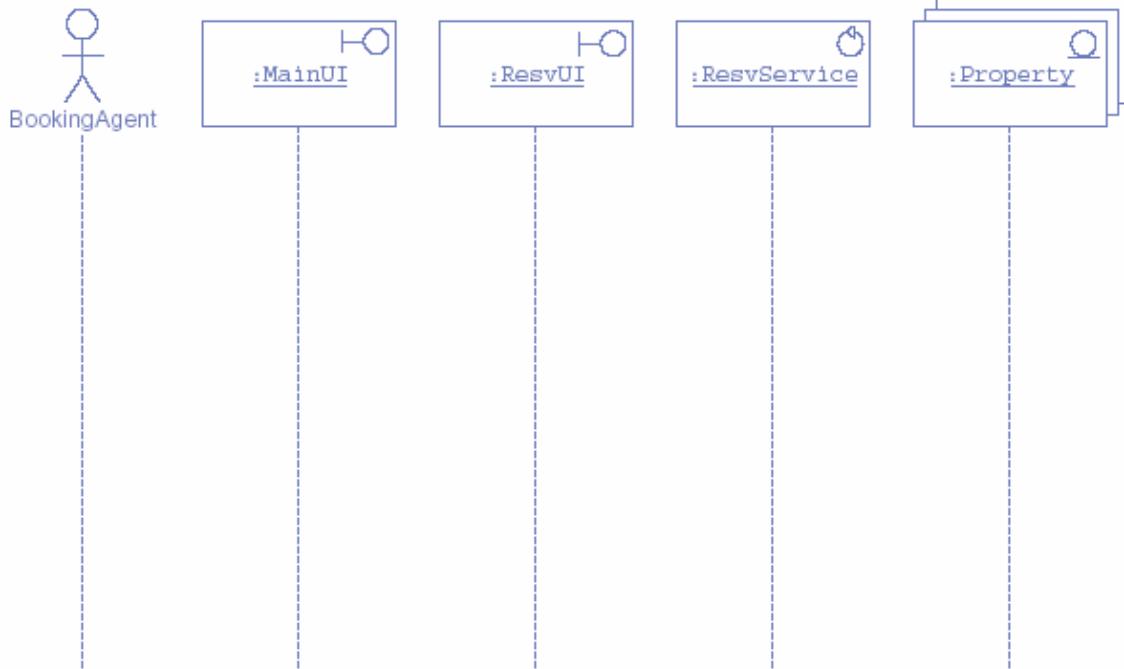
(2) Step 1 – 작업 내용과 관련된 객체 나열

작업 내용에서 생성된 모든 객체를 다이어그램의 상단에 나열합니다.

이미지

n 작업 내용과 관련된 객체 나열

Step 1 – Arrange Components for the First Activity

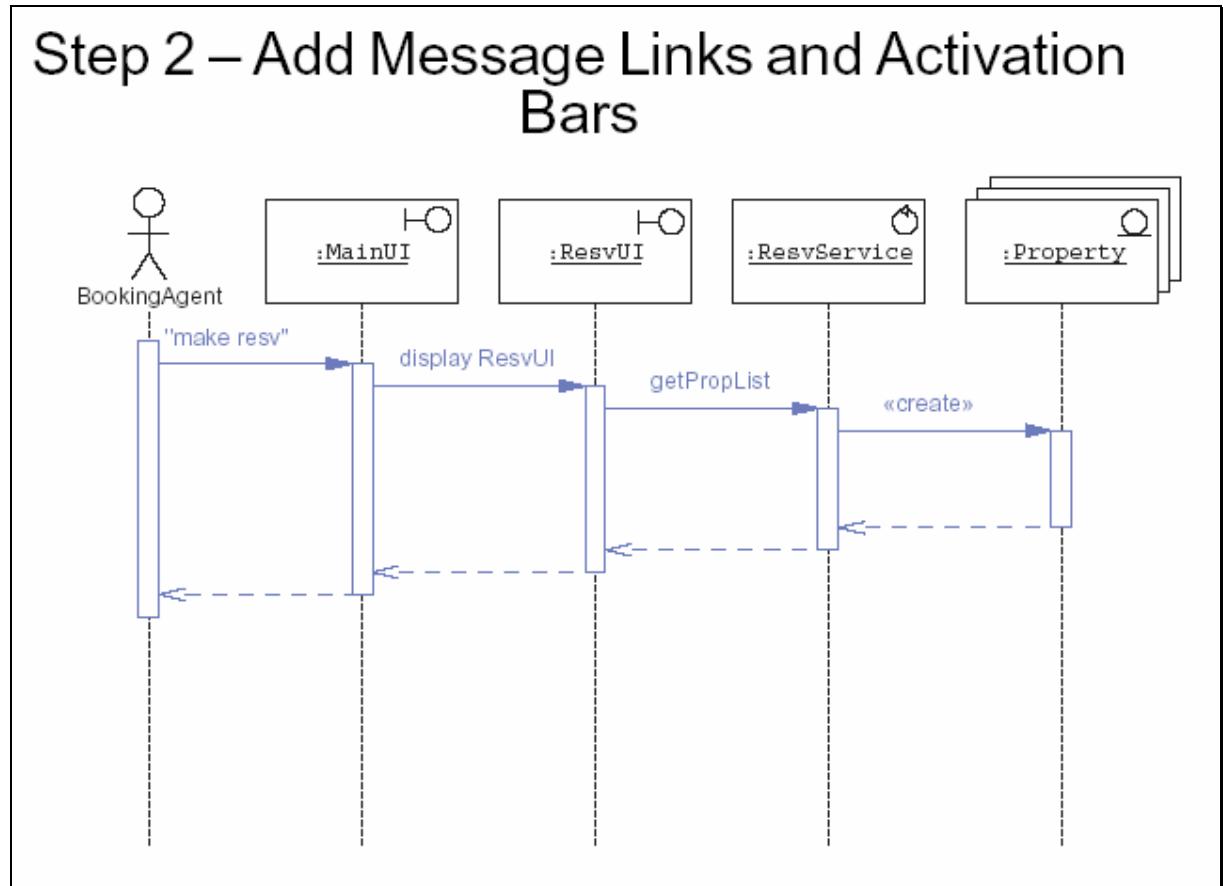


(3) Step 2 – 메시지 링크와 activation bar 추가

“Create a Reservation” 유즈케이스의 activity 2에서 중첩된 activation은 다음과 같이 나타낼 수 있습니다.

[이미지]

n 메시지 링크와 activation bar 추가



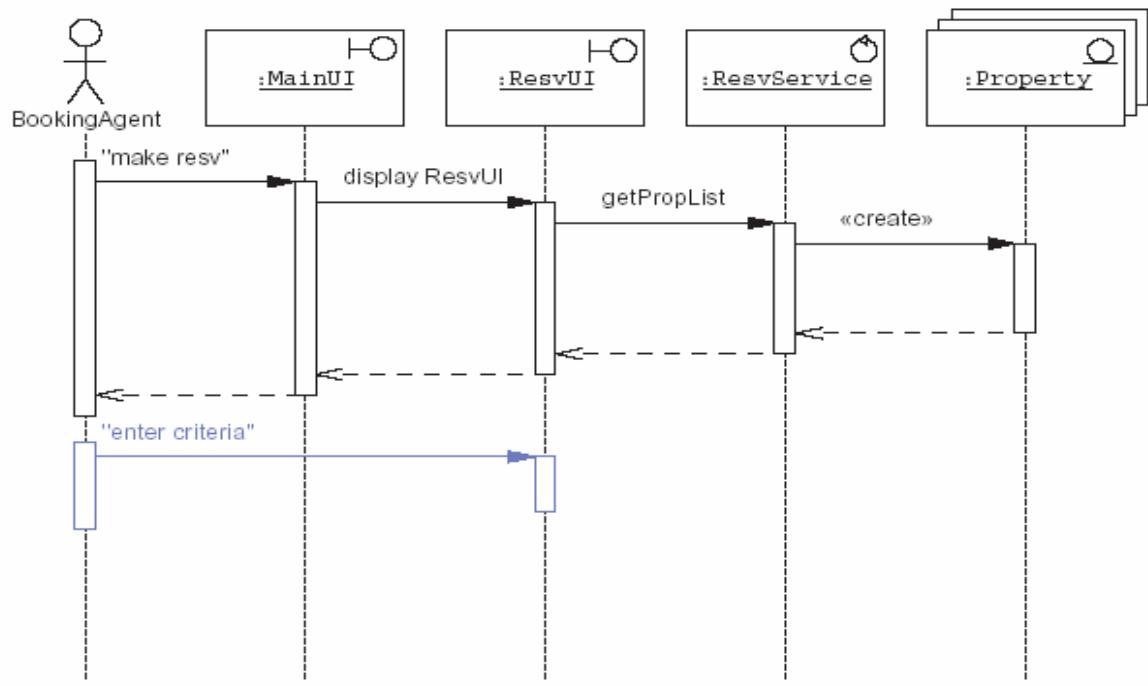
(4) Step 3 – 각 Activity에 대해 Step 2 반복

Activity 3에 대한 일련의 activation입니다. “enter criteria” 메시지는 MainUI 컴포넌트를 건너뛰게 됩니다. 그 작업은 MainUI에서 일어나는 작업이 아니기 때문입니다.

[이미지]

n activity 3에 대한 일련의 activation

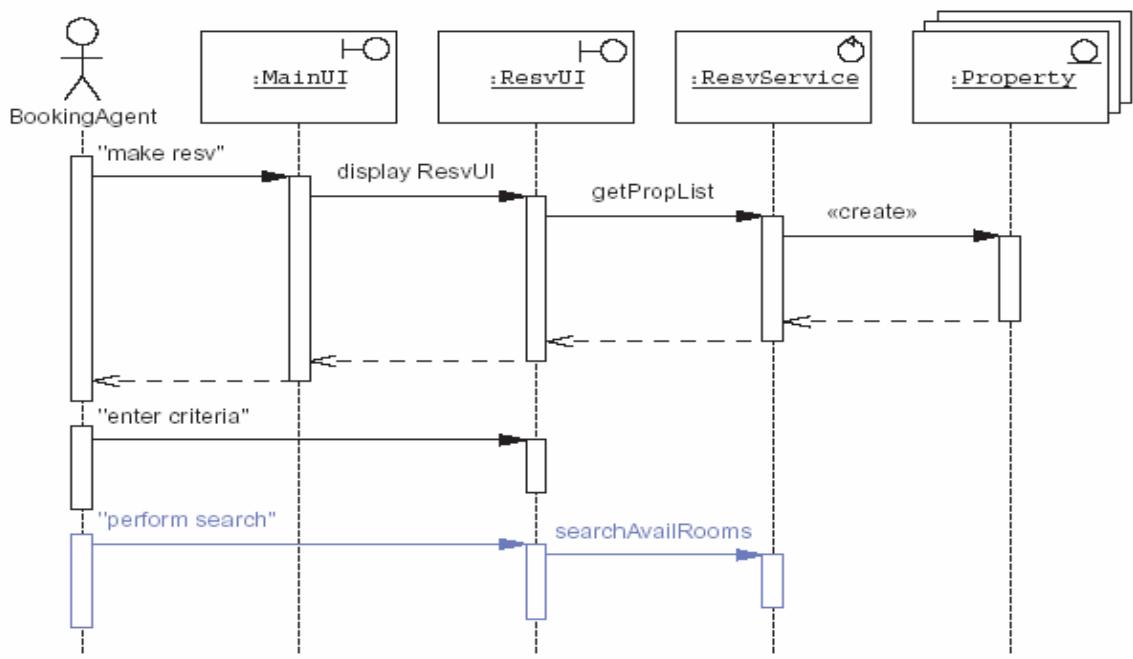
Step 3 — Repeat Step 2 For Each Activity



[이미지]

■ activity 4에 대한 일련의 activation

Step 3 — Repeat Step 2 For Each Activity



과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원4 : 기능적 요구사항 설계

2 모듈 : 사례연구

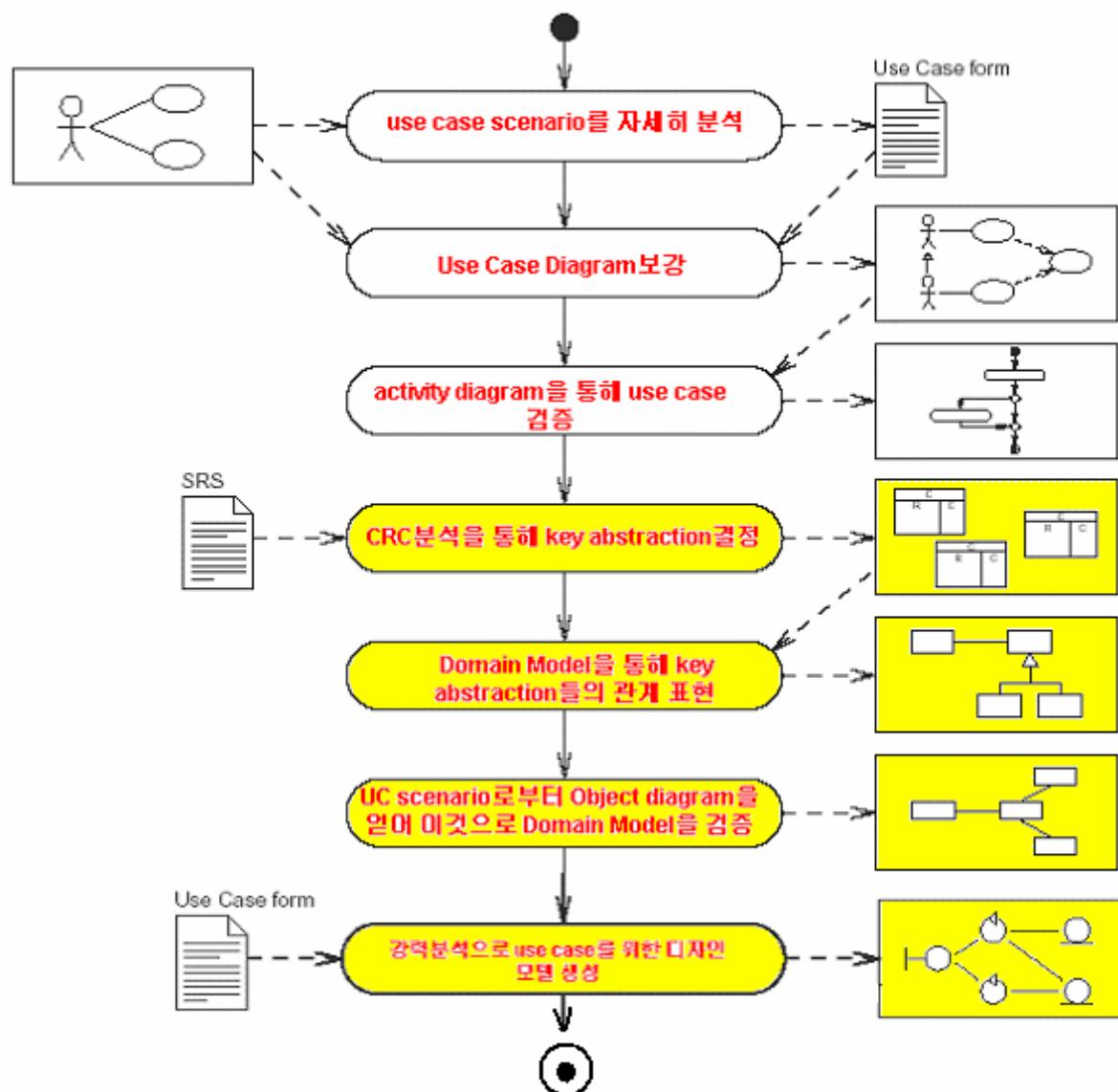
담당강사 : 전은수

생각해봅시다

이번 모듈에서는 지금까지 배운 디어그램과 분석 기법을 실제로 적용하기 위해, [단원3. 기능적 요구 사항 분석 모듈 2. 사례연구]에서 했던 비디오 대여 시스템 구축 작업을 계속해서 진행하겠습니다. 비디오 대여 시스템은 **Activity** 디어그램을 통한 **Use case** 디어그램 검증 작업까지 완료된 상태입니다.

이번 모듈에서는 이 비디오 대여 시스템의 **key abstraction**과 **CRC 분석**, **problem domain model**, **Design model** 생성 단계까지 진행합니다.

[애니메이션]



■ 학습하기 ■

1. 비디오 대여 시스템의 key abstraction

1) Candidate Key Abstraction

(1) SRS 참고 하기

① 참고할 Section

SRS는 여러 **section**(절)로 구성되어 있습니다.

각 절에서 다음과 같은 내용에 유의하면서 **key abstraction**을 합니다.

– Scope 절과 Context 절

이 부분들을 통해 제안된 시스템의 **high-level** 관점과 시스템의 외부 “**touch point**”를 알 수 있습니다. 이 내용에 속하는 명사들은 **problem domain**을 이해하는 데 필수적이 라고 할 수 있습니다.

– Functional Requirements 절

SRS 문서의 이 절은 **problem domain**에 있는 기술적인 내용들을 모두 다루고 있습니다. 여기서 **problem domain**의 세부사항과 기능들을 찾을 수 있습니다. 이 때 가장 중요한 부분은

- 유즈케이스와 유즈케이스 시나리오
- 기능적 요구 사항

입니다. 대규모 프로젝트에서는 SRS 문서도 분량이 많을 것입니다. 명사를 찾기 위해 SRS 문서를 분석하는 작업은 시간이 오래 걸릴 수 있습니다. 연습을 통해서 도메인의 일부가 확실히 아닌 명사는 스킁하는 능력을 키울 수 있습니다.

다음은 비디오 대여 시스템의 SRS의 목차입니다. 표시된 **section**에서 명사를 추출하여야 합니다.

SRS 목차

- 1 소개
 - 1.1 목적
 - 1.2 범위
 - 1.3 시스템 상황
 - 1.4 주요 참가자
 - 1.5 약어 리스트
 - 1.6 문서의 구성
 - 1.7 기능적인 순서 변경시의 핸들링
 - 1.8 레퍼런스
- 2 제약사항과 가정사항
 - 2.1 개발 프로세스와 팀 제약사항
 - 2.2 환경적, 기술적 제약사항
 - 2.2.1 소프트웨어 제약사항
 - 2.2.2 하드웨어 제약사항
 - 2.3 배치 제약사항
- 3 위험요소
 - 3.1 기술적 위험
 - 3.2 스킬과 자원적 위험
 - 3.3 요구사항 위험
 - 3.4 정치적인 위험
- 4 기능요구
 - 4.1 주요 기능 요구
 - 4.1.1 Essential
 - 4.1.2 High-value
 - 4.1.3 Follow-on
 - 4.2 사용자
 - 4.2.1 사용자 : 방문자
 - 4.2.2 사용자 : 회원
 - 4.2.3 사용자 : 가맹점 주인
 - 4.2.4 사용자 : 시스템 운영자
 - 4.3 Use Case
 - 4.4 어플리케이션
 - 4.5 Use Case 상세 요구
- 5 비기능 요구
 - 5.1 성능
 - 5.2 확장성
 - 5.3 사용성
 - 5.4 보안
 - 5.5 유지보수성
- 6 프로젝트 용어

② <범위>에서 명사 골라내기

다음은 비디오 대여 시스템의 **SRS** 내용 중 범위(**Scope**)에 관한 부분입니다. 이 부분에서 명사를 추출해 봅니다.

1 소개

1.1 목적

이 문서의 목적은 총알 배송 시스템(**Media Delivery System**, 이하 “시스템”이라고 함)에 대한 요구사항을 기술하고, 제공하는 시스템의 제약사항 및 위험요소를 기술함으로써 이 시스템을 개발하는데 기초자료가 됨을 목적으로 한다. 기술한 문서의 유효기간은 프로젝트 진행기간으로 한정한다.

1.2 범위

시스템의 범위는 이미 가입한 회원이 **DVD** 대여를 할 수 있으며 구매도 가능하다. 배송의 빠른 배송을 위해 이 시스템에서는 가맹점을 모집할 것이며 전국의 가맹점이 배송을 담당할 것이다. 배송을 담당한 각 가맹점 시스템(본 시스템에서 제공)에서는 배송정보를 알려주도록 한다.

1.3 시스템 상황

총알 배송 시스템의 주요 3가지 외부 시스템 인터페이스 : 데이터 저장을 위한 중앙 및 외부(가맹점) **DBMS**, 크레디트 카드 인증 시스템, 배송정보를 알려 줄 수 있는 시스템이다.

.....

.....

.....

③ <**FRs**>에서 명사 골라내기

다음은 기능적 요구 사항에 관한 **section**입니다. 이 부분에서 명사를 추출해 봅니다.

.....

4. 기능요구

4.1 주요 기능 요구

4.1.1 Essential

- | 웹, 모바일을 통한 대여와 구매 시스템
- | 신청한 DVD에 대한 배송정보 시스템
- | 회원이 구매하고 대여한 정보에 대해서 확인할 수 있는 페이지
- | 가맹점 가입 및 탈퇴 (SRS 첨부#1)

4.1.2 High-value

- | 회원 가입 및 탈퇴
- | DVD에 대한 정보 검색

4.1.3 Follow-on

- | 방문자와 기존회원은 공동구매를 통해 원하는 DVD를 싸게 구입
- | 가맹점에서는 다량구매를 통해 많은 양의 DVD를 구입

4.2 사용자

4.2.1 사용자 : 방문자

이 시스템에 우연히 접근 했거나 아직 회원에 가입하지 않은 사용자이다. 컴퓨터에 대한 사전지식이 전혀 없어도 되며 인터넷을 사용할 줄만 알면 된다.

4.2.2 사용자 : 회원

방문자 중에 정식으로 회원에 가입을 하고 회비를 결제한 후 이 시스템에서 DVD를 빌리거나 사려고 하는 사용자이다. 컴퓨터에 대한 사전 지식은 위의 방문자와 동일하다.

4.2.3 사용자 : 가맹점 주인

우리가 제공하는 회원의 대여 정보를 받아 직접 배송을 담당하는 사용자이다. 우리가 제공하는 어플리케이션을 설치 사용한다. 이 사용자는 고학력일 필요는 없지만 MS원도우를 다룰 수 있어야 하며 컴퓨터에 대한 사전이 있어야 하며 우리가 제공하는 어플리케이션을 설치하고 사용할 수 있어야 한다.

4.2.4 사용자 : 시스템 운영자

이 시스템을 운영하고 관리하는 사람이다. 컴퓨터에 대한 전문지식을 가지고 있어야 한다.

4.2.5 사용자 : 크레디트카드 승인 시스템

이 외부 시스템은 미디어 컨텐츠 총알 배송 시스템(Media Delivery System:MDS)에서 DVD 대여(구매)자가 대여 및 구매를 할 때 호출된다.

4.3 Use Case

Use Case Name	Priority	Number	Description
Rent DVD	E	1	DVD 대여
Purchase DVD	E	2	DVD 구매 및 판매
Information of Delivery	E	3	배송 정보 확인
Manage DVD Shop	E	4	가맹점 관리
Manage Member	E	5	회원 관리
Rent Info.	- 190 -	6	회원이 구매, 대여한 정보 확인
Search - DVD	"	"	DVD 검색

(2) Candidate Key Abstraction Form 만들기

Candidate Key Abstractions Form을 작성하려면 먼저 **Candidate Key Abstraction**을 파악하기 위해 SRS 문서의 **Scope, Context, Functional Requirements**(유즈케이스와 유즈케이스 시나리오, 기능적 요구 사항)의 내용을 중점적으로 분석해야 합니다.

Candidate Key Abstractions Form은 세 개의 필드로 구성되어 있습니다.

- **Candidate key abstraction**

SRS에서 찾은 명사들을 기록하는 필드입니다.

- **Reason for Elimination**

후보였던 것이 **key abstraction**이 되면 빈칸으로 남겨지게 되고, 그렇지 않을 경우는 **key abstraction**에서 삭제된 이유를 기록합니다. CRC 분석을 통해서 삭제 여부를 알 수 있습니다.

- **Selected name**

결국, **key abstraction**은 클래스 같은 소프트웨어 컴포넌트를 정렬한 것이 될 수 있습니다. 이 필드는 **key abstraction**으로 선택될 경우의 클래스 이름을 기록하는 곳입니다. CRC 분석을 통해서 결정되는 필드입니다.

① Candidate Key Abstraction Form에 SRS에서 추출한 명사 옮리기

Key abstraction은 시스템에 있는 주요 객체들의 이름입니다.

key abstraction을 결정하기 위해서는 먼저, SRS 문서에 있는 모든 명사를 “**Candidate Key Abstractions Form**”에 나열해서 모든 후보 **key abstraction**을 파악합니다.

다음 표는 비디오 대여 시스템의 초기 **Candidate Key Abstractions Form**의 일부입니다.

n 표 :: 비디오 대여 시스템의 초기 Candidate Key Abstractions Form

Candidate Key Abstraction	Reason for Elimination	Selected Component Name
대여자		
...		
영화		
대여		
관람등급		
장르		
...		
기타		
로그인		
ID		

대여바구니		
회원가입		
...		
성명		
대여 예정일		
대여주문서		
결재		
대여배송지		

2) CRC Analysis

CRC 분석을 위해

1. **candidate key abstraction**을 하나 선택합니다.
2. 이 후보키가 적합한지 유즈케이스를 분석합니다.
3. 기능과 협력 관계를 결정하기 위해 유즈케이스 시나리오와 **FRs**를 탐독합니다.
4. CRC 카드에 이 **key abstraction**을 기록합니다.
5. 이상의 내용을 토대로 **Candidate Key Abstractions Form**을 수정합니다.

이 과정은 반복적으로 이루어집니다.

(1) 대여(Rent)에 관한 CRC 분석

SRS의 **FRs**와 유즈케이스 시나리오를 탐독하여 “대여”에 연관된 모든 요소들을 다음과 같이 **CRC** 카드에 정리합니다.

다음은 **candidate key abstraction** “대여”에 대한 CRC 카드입니다.

대여(Rent)	
Responsibilities	Collaborators
DVD 를 대여합니다.	대여자
...	DVD
대여일	결제
반납일	
대여 ID	
...	

(2) 대여자(Member)에 관한 CRC 분석

SRS의 **FRs**와 유즈케이스 시나리오를 탐독하여 “대여자(혹은 고객, 회원)”에 연관된 모

든 요소들을 다음과 같이 **CRC** 카드에 정리합니다.

다음은 candidate key abstraction “대여자”에 대한 CRC 카드입니다.

대여자(Member)	
Responsibilities	Collaborators
ID	
이름	
나이	
주소	
전화번호	
...	

(3) 영화(DVD)에 관한 CRC 분석

SRS의 **FRs**와 유즈케이스 시나리오를 탐독하여 “영화(DVD)”에 연관된 모든 요소들을 다음과 같이 **CRC** 카드에 정리합니다.

다음은 candidate key abstraction “영화”에 대한 CRC 카드입니다.

영화(DVD)	
Responsibilities	Collaborators
DVD_ID	
제목	
감독	
주연	
관람등급	
...	

3) 최종 key abstraction

이상의 내용을 토대로 비디오 대여 시스템의 **key abstraction**을 선택할 수 있습니다.

표 :: 비디오 대여 시스템의 Candidate Key Abstractions Form

Candidate Key Abstraction	Reason for Elimination	Selected Component Name
대여자		Member
...
영화		DVD
대여		Rent
관람등급	Attribute of DVD	
장르		Category
...

기타	Attribute of Category	
ID	Attribute of Member	
대여바구니		ShoppingCart
...
성명	Attribute of Member	
대여예정일	Attribute of ShoppingCart	
대여주문서		Rent_Order
결재		CyberPass
대여배송지	Attribute of Rent_Order	
대여일	Attribute of Rent	
반납일	Attribute of Rent	
대여ID	Attribute of Rent	

2. 비디오 대여 시스템의 Class 다이어그램 작성

1) Class 다이어그램의 구성 요소

Key abstraction을 사용해서, 다음의 과정을 거쳐 **Domain model**을 생성할 수 있습니다.

1. 각 **key abstraction**에 대한 클래스 노드를 그립니다. 그리고
 - a. 알고 있는 속성을 나열합니다.
 - b. 알고 있는 기능을 나열합니다.
2. 협력 클래스 간의 관계를 그립니다.
3. 관계와 **role name**을 파악하고 기록합니다.
4. **association multiplicity**를 파악하고 기록합니다.
5. **association** 진행 방향을 파악하고 기록합니다.
6. **association** 클래스를 파악하고 기록합니다.

이제부터 이전 분석 단계에서 작성한 **CRC 카드**에 기록된 **key abstraction**을 이용해서 비디오 대여 시스템의 **Class** 다이어그램을 작성하겠습니다.

Class 다이어그램은 클래스와 클래스의 멤버, 그리고 클래스 간의 관계(대개 **association**이라고 하는)를 시각적으로 표현한 것입니다.

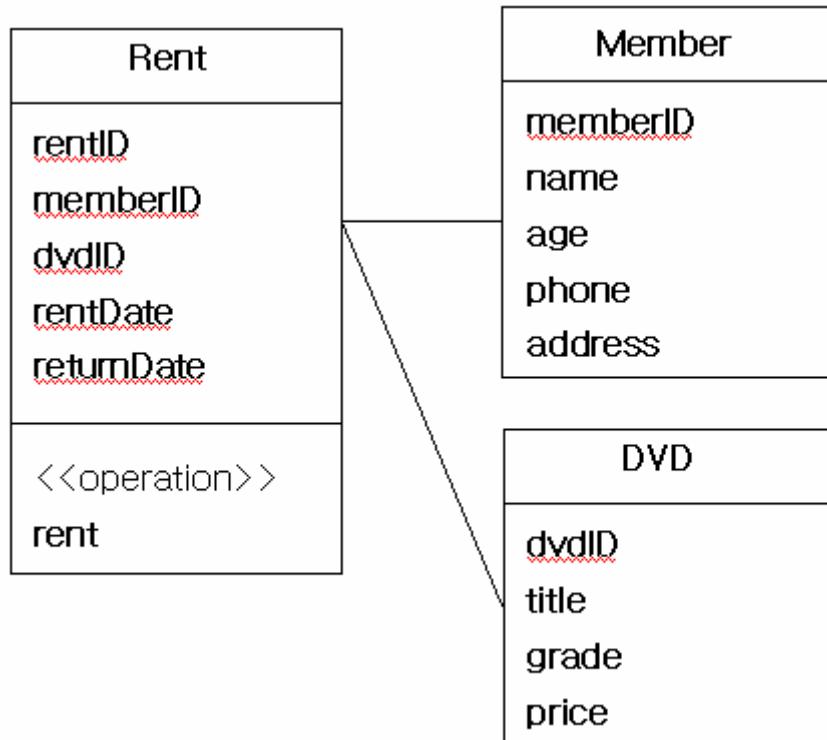
Class 다이어그램의 구성요소로는 **class nodes**, **Associations(Relationship, Roles , Multiplicity, Navigation, Association Classes)** 등이 있습니다.

2) Class 다이어그램 작성

비디오 대여 시스템의 **Class** 다이어그램은 다음과 같이 작성할 수 있습니다.

[이미지]

n 비디오 대여 시스템의 Class 다이어그램



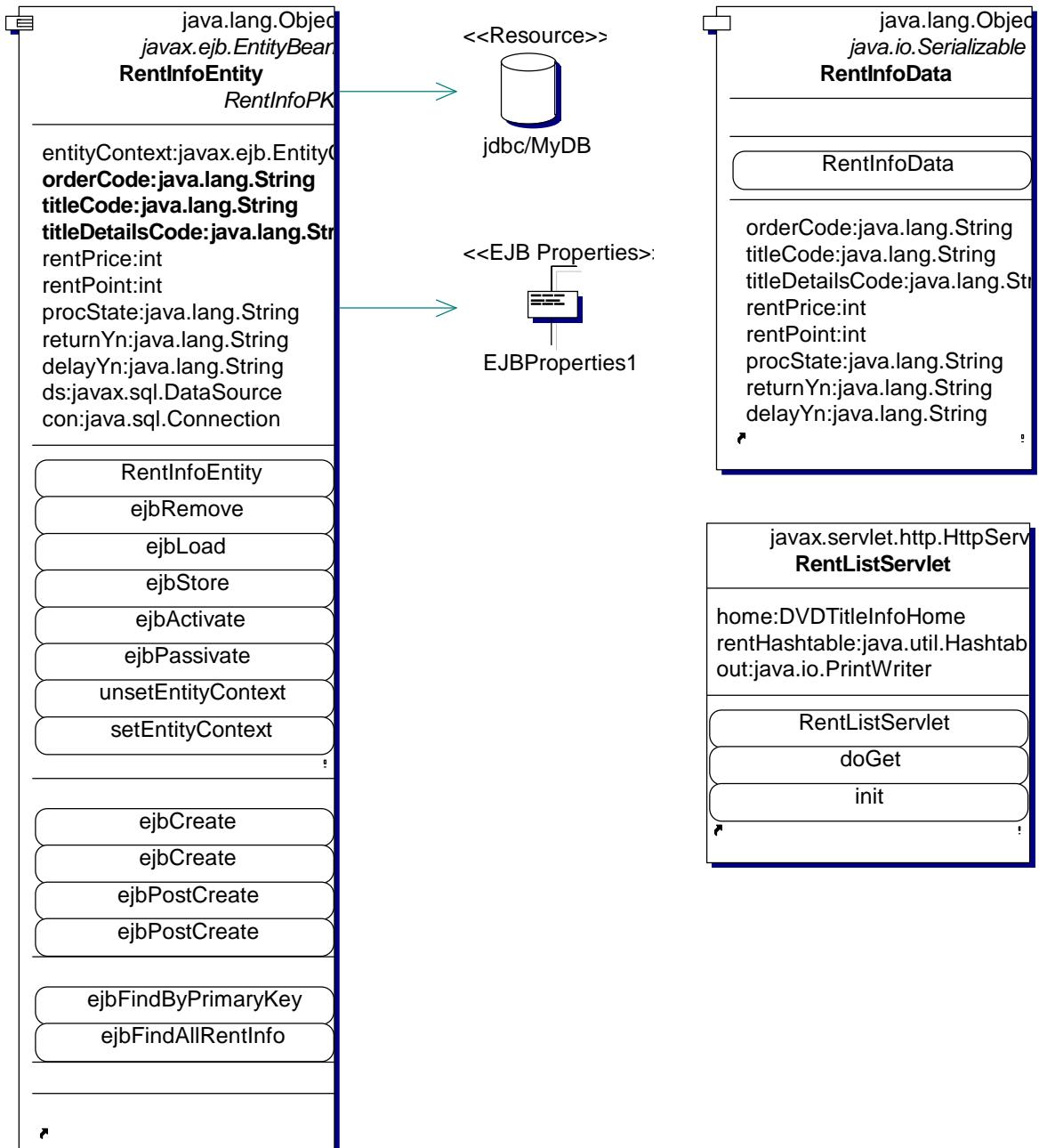
이것은 가장 초보적인 수준의 클래스 다이어그램입니다.

Key abstraction 된 모든 내용에 대해 이런 작업을 반복하면 거대한 클래스 다이어그램이 작성될 것입니다.

다음은 조금 더 많은 클래스 다이어그램을 보여줍니다. 단지 참고만 하십시오. 다이어그램은 여러분들의 역량껏 표현하시는 것이기 때문에 정답이 있을 수 없습니다.

[이미지]

n 관리자 모드의 대여 리스트 Class Diagram



3. 비디오 대여 시스템의 Design Model 생성

1) Design Model의 구성 요소

Design model은 **Requirements model**의 기능적 요구 사항에서 도출됩니다. **Design model**은 유즈케이스를 구현한 것으로, **Solution model**을 만들기 위해 **Architecture model**과 함께 합쳐집니다.

Design components의 종류는 **Boundary(UI) Components**, **Service Components**, **Entity Components** 등이 있습니다.

Design model은 **UML Collaboration** 다이어그램으로 시작화 하고, 유즈케이스 협력 관계에

대한 다른 관점을 제공하는 **Sequence** 다이어그램으로 바꿀 수 있습니다.

이번 모듈에서는 읽기에 쉬운 **Sequence** 다이어그램을 작성하겠습니다.

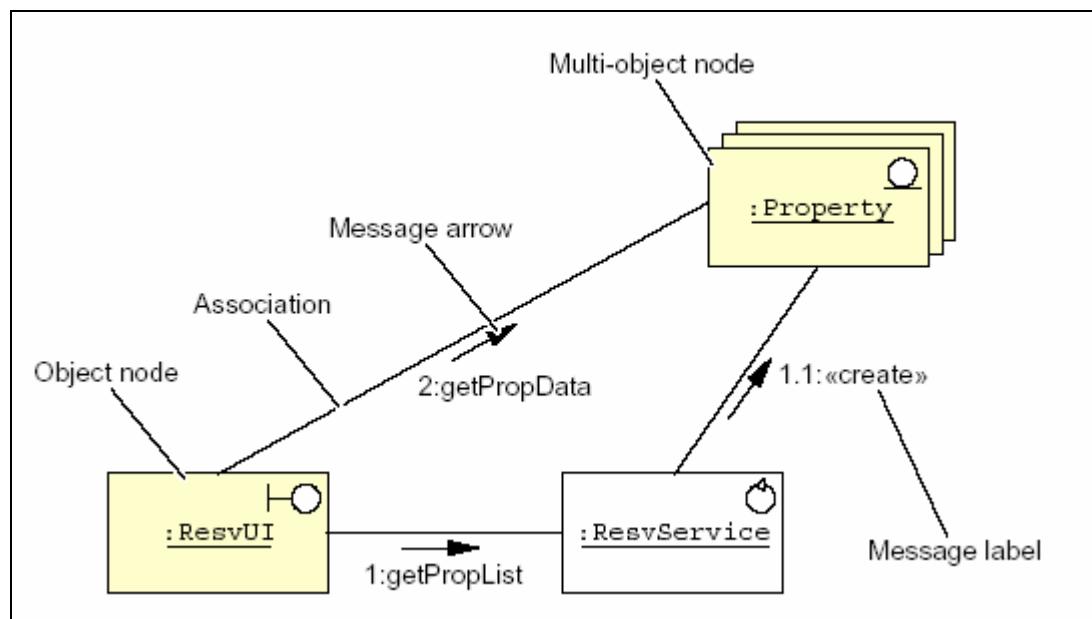
2) Collaboration Diagram

Collaboration 다이어그램은 “컴포넌트 사이의 의존도와 구성을 보여주는 다이어그램입니다”(UML v1.4, page B- 14)

Collaboration 다이어그램은 시스템의 객체, 그들의 관계, 그리고 객체 사이에 전송하는 메시지를 나타냅니다.

[이미지]

n Collaboration 다이어그램 예



3) Sequence Diagram의 구성 요소

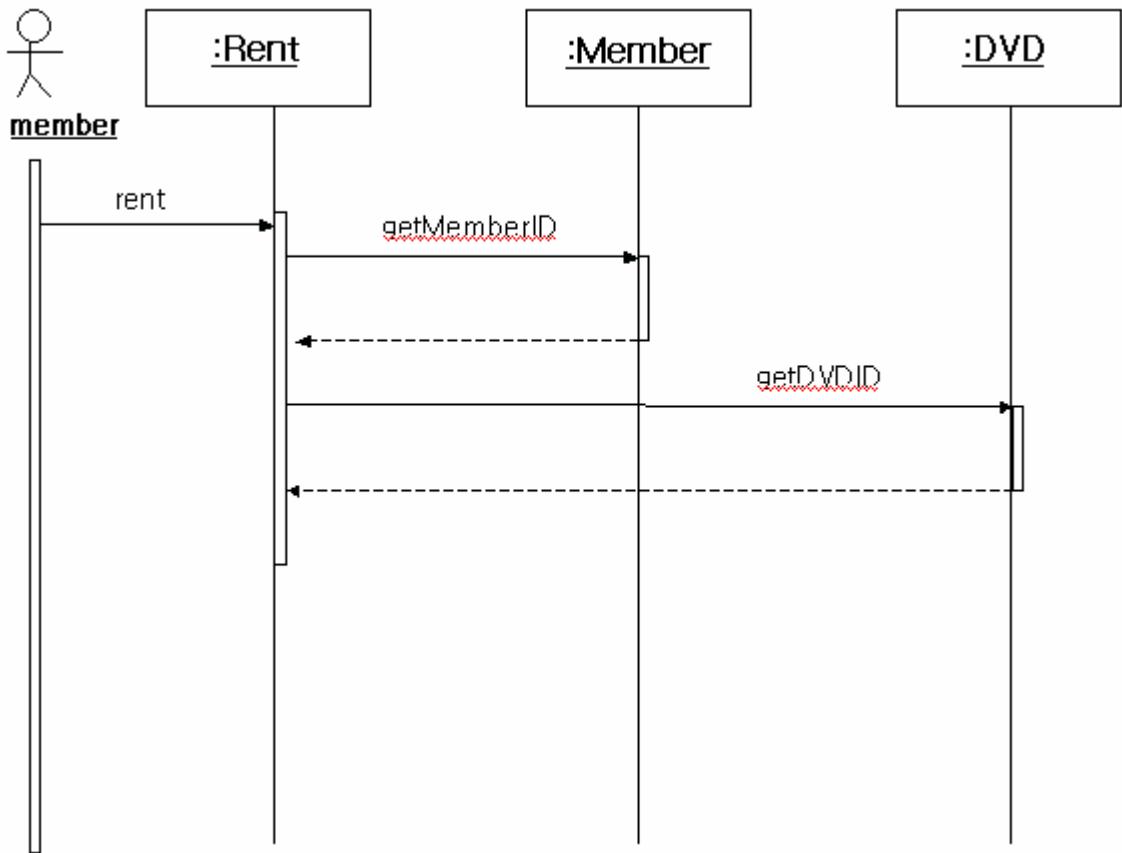
Sequence 다이어그램은 “객체의 상호 작용을 시간 순서에 따라 정렬하여 보여주는 다이어그램”입니다.(UML V1.4, page B- 17)

Sequence 다이어그램은 시스템의 객체와 시간 순으로 구성된 객체들 간의 메시지 송신을 보여줍니다.

다음은 비디오 대여 시스템의 **Sequence** 다이어그램입니다.

[이미지]

n 대여에 관한 간단한 Sequence Diagram



회원이 대여(rent)를 원하면 **Rent** 객체는 **Member** 객체에서 **memberID**를 얻어 오고 (**getMemberID**) 대여를 원하는 **DVD** 객체의 **dvdID**도 얻어 와서(**getDVDID**) 대여 작업을 진행 시킵니다.

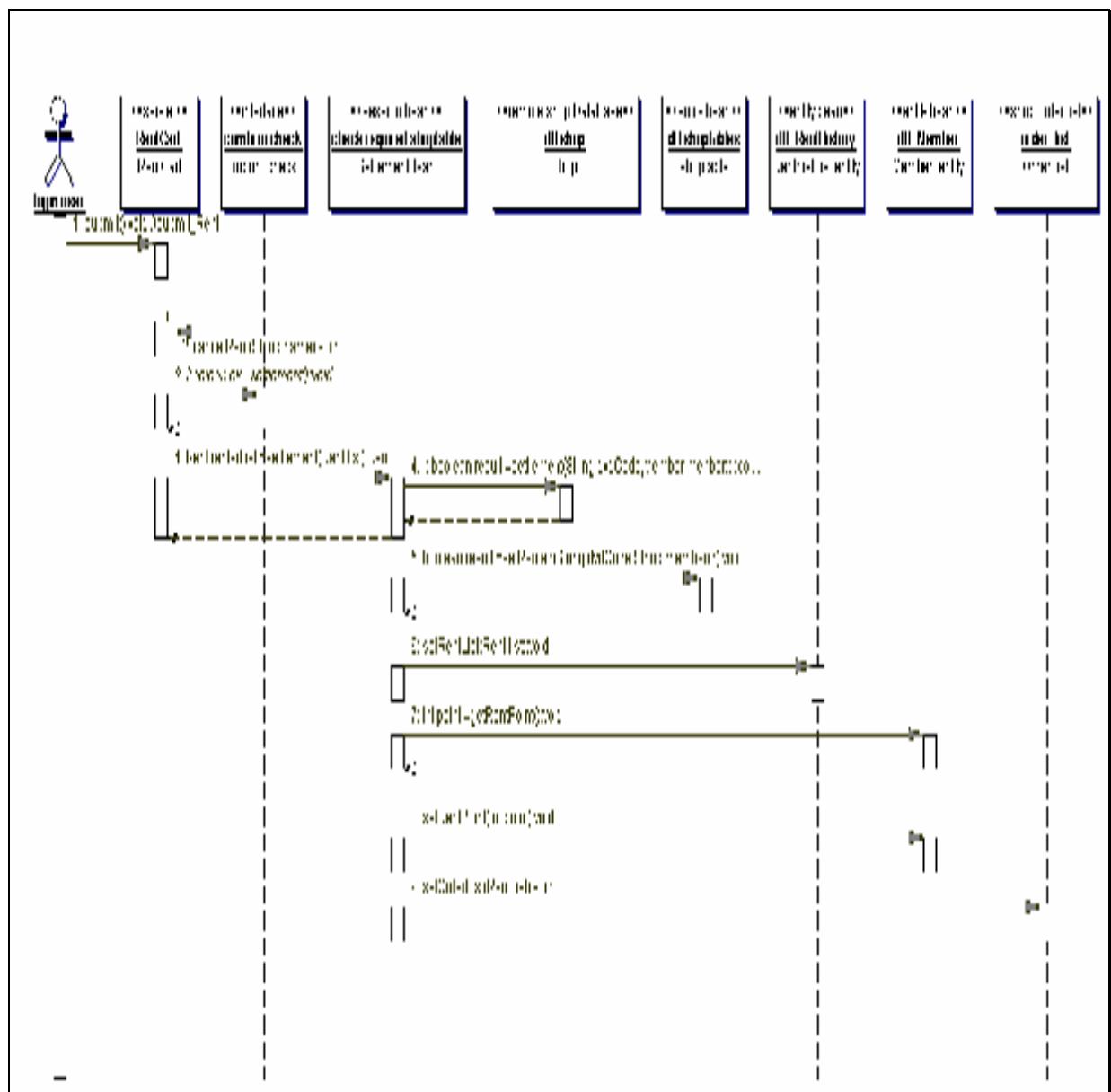
이것은 가장 단순한 대여에 관한 **Sequence diagram**입니다.

각 유즈케이스 시나리오를 보면서 이런 작업을 반복하다 보면 거대한 **Sequence Diagram**이 완성될 것입니다.

다음은 완성된 **Sequence diagram**의 일부입니다. 단지 참조만 하십시오. **Sequence diagram**도 여러분들의 역량에 따라 얼마든지 달라질 수 있습니다.

[이미지]

n 대여에 관한 복잡한 **Sequence Diagram**



과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원5 : 비기능적 요구사항 구축

1모듈 : 아키텍쳐의 원리와 패턴

담당강사 : 전은수

■ 생각해봅시다 ■

이번 절에서는 아키텍트와 아키텍쳐에 대해 배우게 됩니다. 아키텍트의 역할이 필요해진 이유, 그리고 프로젝트를 성공하기 위한 아키텍트의 역할은 무엇일까요? 아키텍쳐와 설계의 차이점은 또 무엇일까요? 이런 것들과 아키텍쳐에 대한 패턴 등 아키텍쳐와 아키텍트에 대하여 알아보겠습니다.

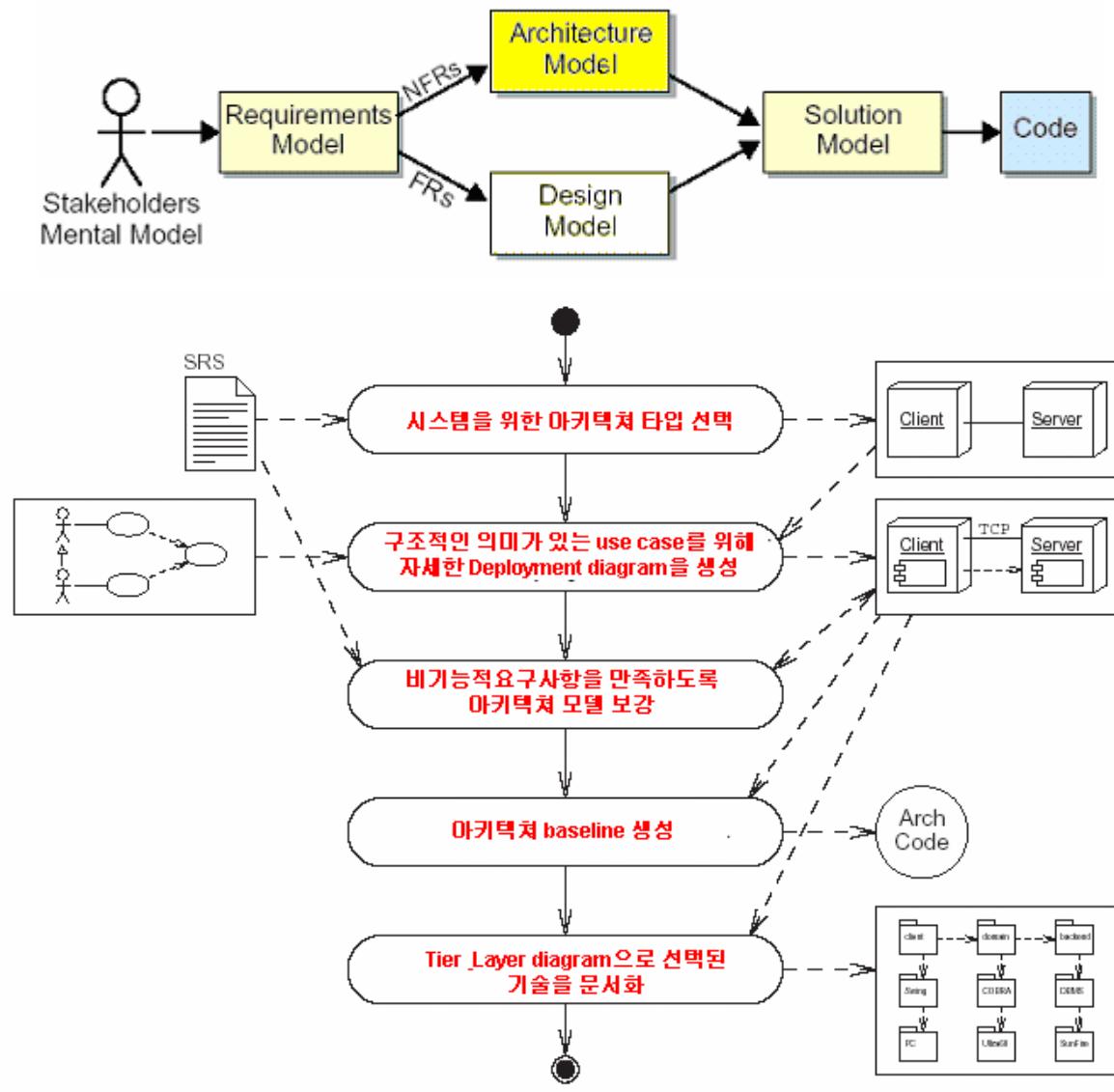
[애니메이션]



■ 학습하기 ■

참고하세요

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. 아키텍트의 역할

1) 아키텍트 역할 정의

그간, 소프트웨어는 **Architect**라고 하는 특별한 역할 분담 없이 발전해왔습니다. 왜 이제서야 소프트웨어 개발 회사에서 이 일을 할 사람들을 고용하는 것일까요?

최근 몇 년간, 소프트웨어 공학에는 두 가지 중대한 변화가 있었습니다. 우선, 가장 눈에 띄는 변화는 스케일입니다. 스케일(예를 들어, 사용자의 수)은 컴퓨터 프로그램이 처음 만들어진 이후로 계속 커지고 있습니다. 웹 어플리케이션의 경우, 정확히 얼마나 많은 사용자들이 서비스를 요청할지 정확히 알 수 없기 때문에, 시스템의 스케일은 중요한 문제입니다.

두 번째 변화는 좀더 급격한 변화인데, 멀티 서버를 통한 소프트웨어 컴포넌트의 분산화입니다. 예를 들어, 웹 어플리케이션은 자연스럽게 분산되어 있습니다. 인터넷을 통해 배포되는 시스템의 경우 대부분 부하가 크기 때문에 필요한 서버를 여러 개 병렬로 구축합니다. 이것이 분산을 더욱 증가시킵니다.

그러면, 왜 분산 시스템이 소프트웨어 공학에 새로운 역할을 만드는 원인이 되었을까요? 그리고 왜 그 역할을 **architect**라고 부르는 것일까요? 이 질문에 대한 답을 이 모듈을 통해서 얻을 수 있습니다.

2) 대규모, 분산 환경 시스템의 위험 요소

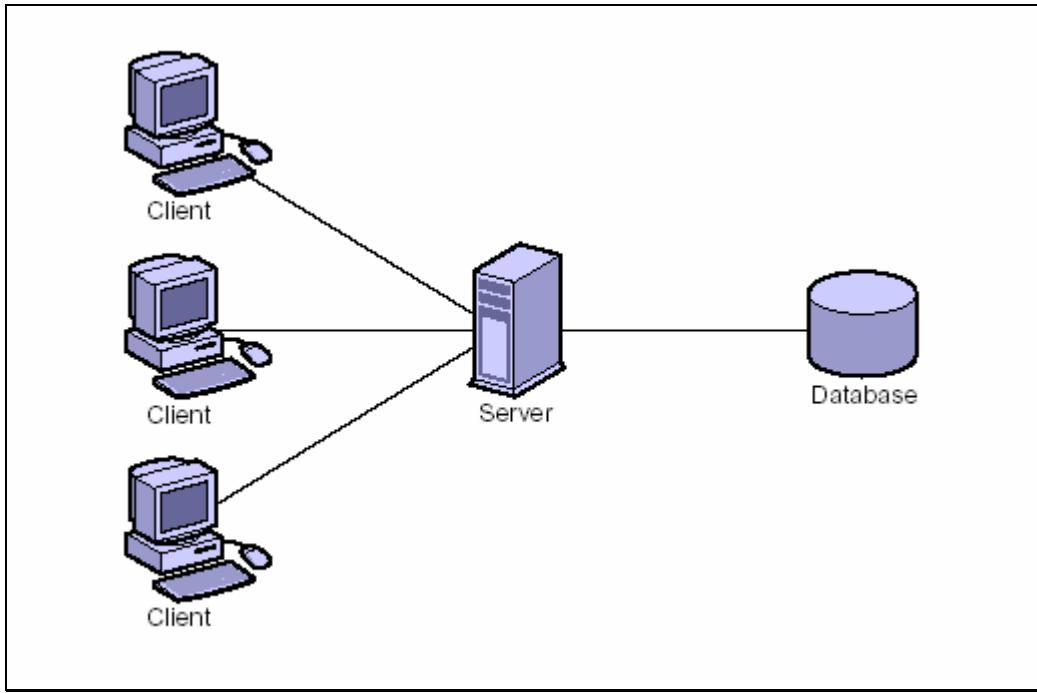
시스템이 단일 호스트에서 분산 기업 시스템과 같은 대규모 환경으로 변함에 따라, 위험요소가 증가하고 프로젝트 요구사항을 만족시키기가 점점 어려워지고 있습니다.

(1) 최소한의 분산 시스템

하나의 호스트에서 돌아가는 시스템은, 시스템의 모든 부분에서 서로에 대한 반응 속도가 빠르고 속도도 거의 비슷합니다. **Client- Server** 시스템일 경우, 시스템의 데이터 소통을 제어하기도 쉽습니다. 이런 환경을 “데이터에 충실한 연산; 사용자에 충실한 검증”이라고 말합니다.

[이미지]

□ **Client- Server System**



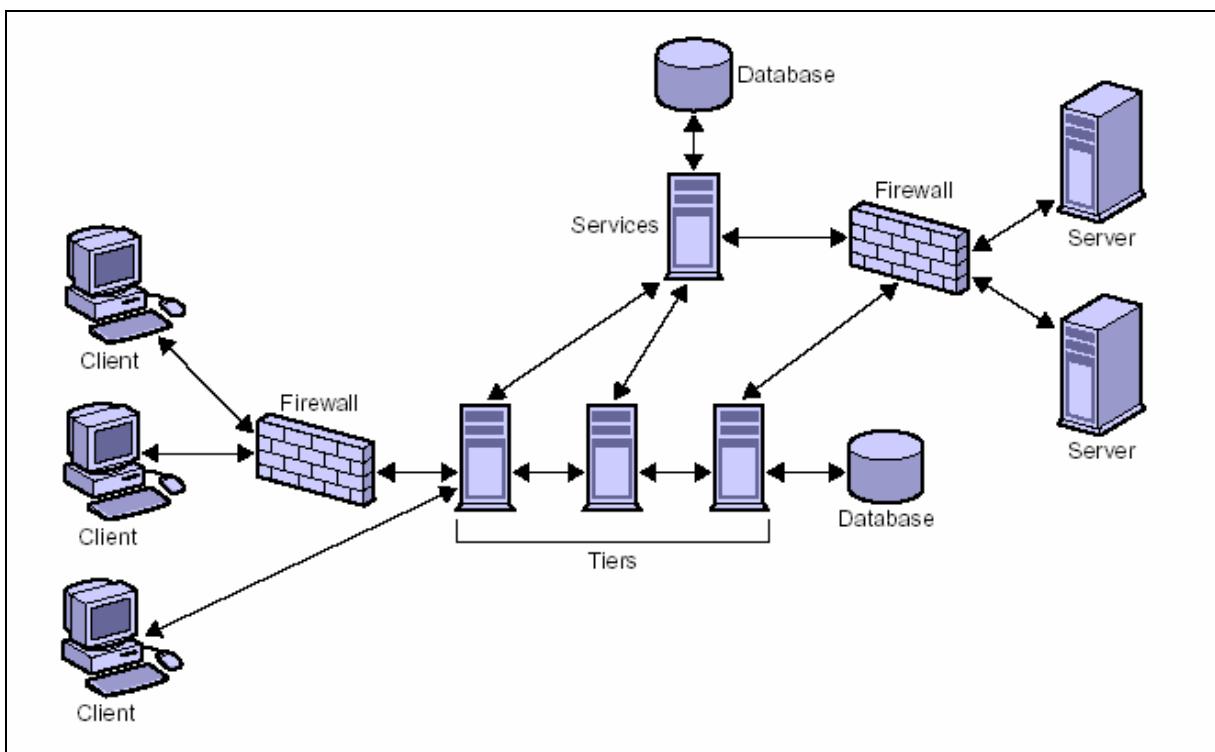
(2) 분산 시스템

좀 더 분산된 시스템에서는, 컴포넌트의 위치와 데이터 송수신 방법에 따라 성능이 좋아질 수도 있고, 나빠질 수도 있습니다. 이런 중요한 결정을 하기 위해 아키텍트(**architect**)가 필요합니다.

아키텍트의 주요 임무는 소프트웨어 컴포넌트 간의 위치, 의사소통 방식 등 **high-level** 계획을 세우는 것입니다. 이 역할은 설계자, 프로그래머, 통합 관리자 등 여러 명의 일로 나누어집니다.

이미지

n 고도 분산 시스템



이런 **high-level** 계획의 목적은

- 시스템이 부분적으로 문제가 발생했을 경우에도 문제가 확대되지 않도록 하는 것입니다.
- 시스템이 요구되는 부하를 다룰 수 있도록 하는 것입니다.
- 시스템이 동시 사용자의 증가에 따른 시스템 확장을 가능하게 하는 것입니다.

Key point

n 아키텍트의 임무

아키텍트의 주요 임무는 소프트웨어 컴포넌트 간의 위치, 의사소통 방식 등 **high-level** 계획을 세우는 것입니다.

3) 서비스의 질

구조는 주로 서비스의 질과 관련이 있습니다. 아키텍트는 유즈케이스를 사용해서 구조적 개발에 대해 적절한 입력 값을 알려줍니다. 하지만 아키텍트가 유즈케이스를 주로 사용하는 것은 아닙니다.

(1) 프로젝트 실패

프로젝트 실패는 대개 기능성 문제에 대한 속성이 되지는 않습니다. 많은 프로젝트가 기능적으로는 완벽에 도달했지만, 최종 사용자에게는 생각한 것처럼 적용되지 않는 경우가 많습니다. 시스템의 질에 대해 주의를 기울이면 이런 문제를 해결하고, 프로젝트를 성공으로 이끌 수 있습니다.

(2) 비기능적 요구 사항

비기능적 요구 사항에 대한 초점은 일반적으로 아키텍처의 주요 목적에 듭니다. 고도의 분산 환경은 시스템이 기능적 요구 사항을 충족시키면서 수행할 수 있는 좋은 구조적 설계를 요구하기 때문입니다.

비기능적 요구 사항은 다음과 같은 카테고리로 분류할 수 있습니다.

- 제약 조건(**Constraints**)

시스템 설계 과정에서 이루어질 수 있는 의사 결정의 제약을 말합니다. 예를 들어, 클라이언트와 데이터를 주고 받기 위해서 **HTTP**를 사용해야 한다든가, 기존의 **DBMS**를 사용해야 한다든가 하는 것들이 모두 제약 조건이 될 수 있습니다.

- 시스템 질(**Systemic qualities**)

서비스의 질을 보장하는 요구 사항은 시스템이 배포된 후에 이루어 질 수 있습니다. 시스템의 질에는 확장성, 성능, 이용가능성, 의존도 등이 포함됩니다.

시스템이 구조적으로 활용할수록 다른 문제들은 해결할 수 있습니다. 반대로, 시스템이 기능

적으로는 완벽하더라도 구조적으로 문제가 많으면 기능성을 수정하거나 사용 가능하도록 전체적으로 다시 작성해야 합니다.

4) 위험 요소 검증 및 제어

훌륭한 아키텍트는 비기능적 요구사항이 시스템 설계에 의해 보장되도록 위험 요소를 제어합니다. 위험 요소를 제어하는 것은 프로젝트 비용과 밀접한 연관관계가 있습니다.

(1) 위험 요소 검증

위험 요소 제어 비용이 적당한지, 너무 많지는 않은지에 대해 결정을 내리기 위해, 위험 요소를 검증하고 비교해야 합니다. 그리고 발생 가능성과 처리 비용을 최소화 해야 합니다. 발생 가능성이 낮은 위험 요소에 대해 고비용 솔루션을 만들어서는 안됩니다. 그런 솔루션을 설계하는 것이 목적이라면, 간단하게 **T1**급의 통신 속도를 위해 필요한 장비를 모든 클라이언트에게 제공하도록 하면 됩니다. 그러나 이것이 최상의 방법은 아닙니다. 그리고 아키텍트는 문제를 해결하기 위해 하드웨어를 사용해서는 안됩니다.

(2) 비용 분석

문제에 접근하는 최선의 방법은 위험 요소를 제어하기 위한 몇몇 옵션의 비용을 분석하는 것입니다. 어떤 옵션은 비용은 적게 들지만 위험 요소를 남겨두고, 어떤 것은 비용이 많이 들지만 위험 요소를 완전히 제거할 수 있습니다.

또한 아무 옵션이 없는 경우도 생각할 수 있습니다. 어떤 구조적 솔루션이 최고인지를 결정하는 것은 투자에 대한 가치에 따라 결정됩니다.

보충

■ 비용 분석

예를 들어, 서버 구축 작업에 대해 생각해보면, 단일 서버로 구축할 경우 초기 비용에서는 현저히 적은 비용으로 구축이 가능합니다. 하지만, 서버가 다운되는 일이 생길 경우, 그로 인한 시간비용 및 비즈니스 업무 비용의 손해가 발생합니다.

즉, 초기 구현 비용과 유지 비용이 적게 들 수는 있지만, 장기적으로는 더 많은 비용을 들일 수도 있습니다.

5) 아키텍트의 역할

아키텍트의 역할은 기술과 관리에 대한 책임을 포함합니다.

(1) 기술에 대한 책임

아키텍트는 반드시 유즈케이스를 구조적으로 확실히 파악해야 합니다. 아키텍트의 업무에서

가장 중요한 것은 기대하지 않은 상황을 준비하는 것입니다.

아키텍트는 구조적 프로토타입의 개발을 안내해야 하는 책임이 있습니다. 프로토타입은 문서와 다이어그램을 통해 순수하게 이론적으로 구조화하고 구현하는 것일 겁니다. 또는 좀 더 일반적으로 구성된 코드의 요소를 유도할 수 있습니다. 프로토타입의 목표는 위험 요소를 발견하기 위해 최종 시스템에 근접하게 만드는 것입니다. 구조적 프로토타입을 개발할 때, 아키텍트는

- 가정과 제약조건을 나열합니다.

알고 있는 모든 가정과 제약조건을 나열합니다. 잘못된 가정을 세우면 시스템에 대한 작업을 다시 해야 하는 수도 있습니다. 가정을 기록하는 것은 어떤 것이 재작업이 필요한 변화가 있는지 최대한 빨리 파악하는데 도움이 됩니다.

- 위험 요소를 분석하고 줄이기 위한 계획을 세웁니다.

아키텍트는 위험 요소를 나열하고 그 위험 요소를 최소화 할 수 있는 계획을 세웁니다.

- 배포 환경 설명서를 작성합니다.

아키텍트는 배포 환경과 소프트웨어와 서비스 컴포넌트가 어떻게 어디에 위치하고, 그 컴포넌트들이 어떻게 의사소통을 하는지에 대해 구체적으로 기술합니다.

- 상호작용 다이어그램

아키텍트는 다이어그램을 사용해서 구조적 프로토타입이 모든 기능적 요구사항을 지원하는 것을 검증합니다.

(2) 관리에 대한 책임

최소한, 아키텍트는 관리의 책임이 있습니다. 일반적으로 비용 관리의 측면을 포함하는데, 다른 예산 관리자는 기술이나 위험 요소를 줄이기 위한 비용에 대해서 결정을 내리기에 적합하지 않기 때문입니다.

아키텍트는 팀원들과 **stakeholder**들과 효율적으로 일하기 위한 스킬이 필요합니다. 다음과 같은 대인관계에 대한 능력이 필요합니다.

- **Stakeholders**에 대해

Stakeholders 입장에서는 시스템에서 자신들이 필요로 하는 부분에 대한 구현이 부족하다거나, 쓸데없이 과해서 비용만 낭비한다는 생각을 할 수 있습니다. 아키텍트는 **stakeholders**에게 모든 부분이 필요한 만큼 구현된다는 것을 부드럽게 처리해야 합니다. 그렇지 않으면, 전체 프로젝트에 문제가 될 수도 있습니다.

- 팀원에 대해

시스템 개발에 사용할 기술을 선택하는 것도 아키텍트의 업무입니다. 신기술이나 팀원들이 익숙하지 않은 개발 기술을 사용하는 경우에는 프로토타입 개발 전반에 걸쳐 신경을 써야 합니다. 팀원들이 개발에 사용되는 기술을 익히게 되면 그 중에는 구현 단계에도 남아서 같이 일할 사람도 있을 겁니다.

2. 아키텍쳐의 원리

1) 아키텍쳐와 design(설계)의 구분

때로는 아키텍트와 설계자의 업무를 구분하기가 어렵습니다. 사실, 아키텍트의 역할은 정의 내리기가 힘듭니다. 하지만, 아키텍트 역할의 중요성은 대다수 프로젝트에 대한 구성 요소 조사를 통해 알 수가 있습니다.

다음 표에서 아키텍트와 설계자를 비교해 볼 수 있습니다.

■ 표 :: 아키텍트와 설계자의 차이(Difference Between Architect and Designers)

	Architect	Designers
Abstraction level	높은 추상화 일부 세부사항에 대한 광범위한 관점	낮은 추상화 많은 세부사항에 대한 구체적인 관점
Deliverables	시스템, 부속시스템 계획, 구조적 프로토타입	컴포넌트 설계, 코드의 구체화
Area of focus	비기능적 요구 사항 위험 요소 관리	기능적 요구 사항

비즈니스 분석가, 시스템 아키텍트, 시스템 설계자 등은 모두 다른 역할이지만, 때로는 아키텍트가 모든 기능을 수행하기도 합니다.

아키텍트는 프로젝트 전반에 걸쳐 책임을 지고 있는 것은 아닙니다. 그 대신 아키텍트는 주로 요구 사항 정의 단계에서부터 세부 구현 단계까지 참여합니다. 하지만, 최종 산출물은 아키텍처 모델의 초기 제안에 대해 따른다는 것을 기억해야 합니다.

2) 아키텍쳐의 원리

아키텍쳐 원리는 시스템 아키텍쳐를 구축할 때 제안한 좋은 습관인 가설을 세우는 것입니다. 이런 원리들은 여러 아키텍쳐 패턴의 근간을 이룹니다. 그리고 아키텍트가 서비스 요구 사항의 질을 만족하기 위한 결정을 내리는데도 영향을 줍니다.

아키텍쳐의 원리는 잠재적으로 수백가지가 있지만, 이 모듈에서는 아래의 원리에 대해 다루겠습니다.

— 관계의 분리(Separation of Concerns)

이 원리는 아키텍트에게 다른 기능이나 내부구조를 가지고 있는 컴포넌트를 분리하도록 하는 것입니다. 예를 들어, 비즈니스 로직(**Model**)과 사용자 인터페이스를 구분하는 것은 중요한 일입니다. 또는 사용자 인터페이스를 시각적 요소(**View**)와 기능 처리 로직(**Controller**)을 구분할 수도 있습니다. 이런 분리 형태를 **MVC** 구조적 패턴 (**Model- View- Controller**)이라고 합니다.

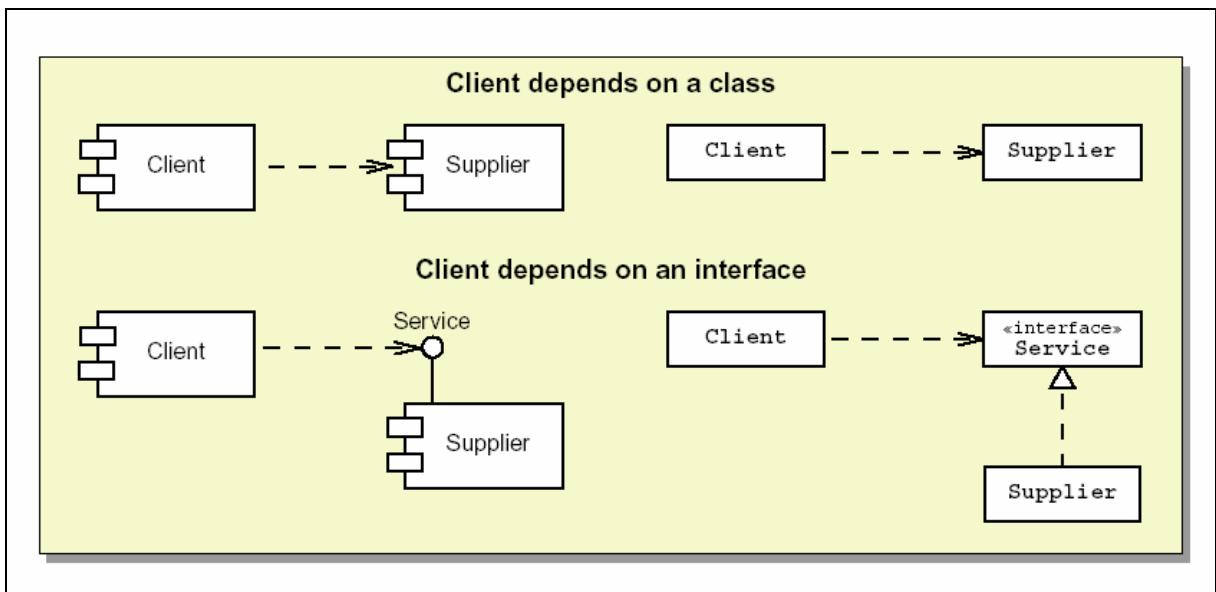
- Dependency Inversion Principle

“추상화에 대한 의존. 구체화에 의존하지 말아야 합니다.”(Knoernschild page 12)

인터페이스에 의존하는 클라이언트 컴포넌트는 좀더 유연한 소프트웨어를 만들 수 있습니다. 클라이언트 쪽 컴포넌트에 영향을 주지 않고, 서버쪽 컴포넌트를 변경하거나 대체할 수 있기 때문입니다.

[이미지]

n Dependency Inversion Principle



- 안정적 컴포넌트에서 휘발성(volatile) 요소 분리

시스템이 변경될 때, 패키지의 일부 내용은 변경하지 않길 원할 경우, 이 원리를 적용합니다. 사용자 인터페이스 같은 휘발성 요소(자주 변경되는)와 **domain entity** 같이 좀더 안정적인 요소를 분리한 컴포넌트 그룹을 제시합니다.

- 컴포넌트와 컨테이너 프레임워크 사용

컴포넌트는 컨테이너가 관리하는 소프트웨어 구성 요소입니다. 예를 들어, 서블릿은 웹 컨테이너가 관리합니다. 일반적으로 소프트웨어 개발자는 컴포넌트를 생성합니다. 이런 컴포넌트들은 인터페이스를 구현하거나 언어 스펙에 맞춰 구성합니다.

컨테이너는 특정 스페어에 맞춰 구현한 컴포넌트를 관리하는 소프트웨어 프레임워크입니다. 예를 들어, 웹 컨테이너는 **HTTP** 요청을 관리하고 서블릿에 넘겨줍니다. 컨테이너는 여러분이 구축하거나 얻을 수 있는 시스템 또는 라이브러리입니다.

컨테이너 프레임워크는 아키텍트에게 강력한 도구입니다. 이 프레임워크는 아키텍트가 시스템 나머지 부분에 대한 내부 구조를 제공하는 소프트웨어를 구입할 수 있게 합니다. 이렇게 하면 설계자나 프로그래머는 내부 구조보다 비즈니스 로직 컴포넌트에 집중할 수 있습니다.

- 컴포넌트 인터페이스를 간결하고 깨끗하게 유지

컴포넌트의 인터페이스가 복잡하면, 소프트웨어 개발자들에게 컴포넌트의 사용 방법을 이해시키기가 힘듭니다. 컴포넌트 인터페이스는 응집력이 높아야 합니다.

- 원격 컴포넌트 인터페이스를 세분화하지 않고 유지

원격 컴포넌트는 다른 컴포넌트와 데이터를 주고 받기 위해 네트워크 작업이 필요합니다. 원격 호출을 최소한으로 유지해야 합니다. 따라서, 원격 인터페이스는 간결하고 세분화 하지 않는 것이 좋습니다.

3) Architecture pattern(아키텍처 패턴)과 Design pattern(디자인 패턴)

패턴은 특정 환경에서 발생한 문제에 대한 표준 솔루션입니다. 시스템을 계획할 때 패턴을 사용함으로써, 아키텍트는 문제와 요구사항을 파악하기 위해 시스템을 분석할 수 있습니다. 이런 문제들은 각각 특정 패턴에 의해 해결할 수 있습니다. 아키텍트는 집합적인 구조체로 패턴 솔루션을 조합해서 시스템을 종합할 수 있습니다.

패턴을 선택하고 적용하는 작업을 효율적으로 하기 위해서는, 아키텍트가 다양한 패턴의 카탈로그에 대해 알고 있어야 합니다. 각 카탈로그는 분석, 설계, 아키텍처, 기술 등 시스템 개발의 특정한 측면에 중점을 둡니다.

아키텍트는 주로 디자인 패턴과 구조적(아키텍처) 패턴을 사용합니다.

- 디자인 패턴은 기능적 요구 사항을 지원하는 효율적이고 재사용 가능한 객체지향 소프트웨어 컴포넌트를 만들기 위한 구조와 기능을 정의합니다.

예를 들어, 일반적으로 사용하는 디자인 패턴은 **Composite** 패턴입니다. 이 패턴은 서로 다른 객체를 같은 방법으로(같은 메소드를 사용하는 식으로) 처리할 수 있도록 해주는 패턴입니다.

보충

■ composite pattern 이해

컴퓨터 제조업체에서 컴퓨터를 제작하는 데 필요한 각각의 부품 가격을 계산하는 시스템을 만들려고 합니다. 각각의 부품들, 하드 드라이브, **CPU**, 키보드 등의 가격을 따로 계산하기도 하고, 부품을 조립해서 만든 컴퓨터의 가격을 계산하기도 합니다. 가격을 계산하는 것은 기능적 요구 사항입니다. 이를 위하여 각 부품에도 가격을 계산하는 기능과 부품을 조립한 완성품에도 가격을 계산하는 기능을 두는 것을 **composite pattern**이라고 합니다. 이것은 디자인 패턴 중의 하나입니다.

- 구조적 패턴은 비기능적 요구 사항을 지원하기 위한 시스템과 하위시스템의 구조와 기능을 정의합니다.

예를 들어, 일반적으로 사용하는 구조적 패턴은 **Layers** 패턴입니다. 이 패턴은 인접한 추상화 사이에서만 커플링이 가능한 추상화 방법으로 파티셔닝 시스템에 적용하는 기술입니다. 흔한 예로 **APIs**, **VM**, 클라이언트-서버 구조를 들 수 있습니다.

- 다른 일반적인 패턴은 **Pipe**와 **Filter** 패턴입니다. 이 패턴은 각 호스트에서 호스트로 데이터가 이동하고, 각 호스트에서는 처리 과정에서 추가된 값을 받는다는데 기반을 두고 있습니다. 데이터가 이동하는 길이 파이프이고, 데이터 처리기는 필터입니다. 이 패턴의 예로는 **UNIX** 툴과 그 외 많은 컴파일러를 들 수 있습니다.

구조적 패턴과 디자인 패턴 사이의 관계는 기본적으로 스케일과 초점 중 하나에 있습니다. 구조적 패턴은 하위 시스템의 용어와 서비스 레벨 요구 사항을 만족하는 **high-level** 컴포넌트로 구체화합니다. 디자인 패턴은 각각의 하위 시스템과 구조적 패턴에 의해 구체화된 하위 시스템을 구성하는 **low-level** 컴포넌트를 만들 때 사용됩니다.

참고하세요

일반적으로 디자인 패턴과 구조적 패턴을 모두 사용하게 됩니다. 대규모, 복잡한 시스템을 구축할 때는 구조와 기능을 이루기 위해 여러 개의 패턴을 적용합니다.

3. 아키텍쳐 패턴의 적용

1) SunTone Architecture 방법론

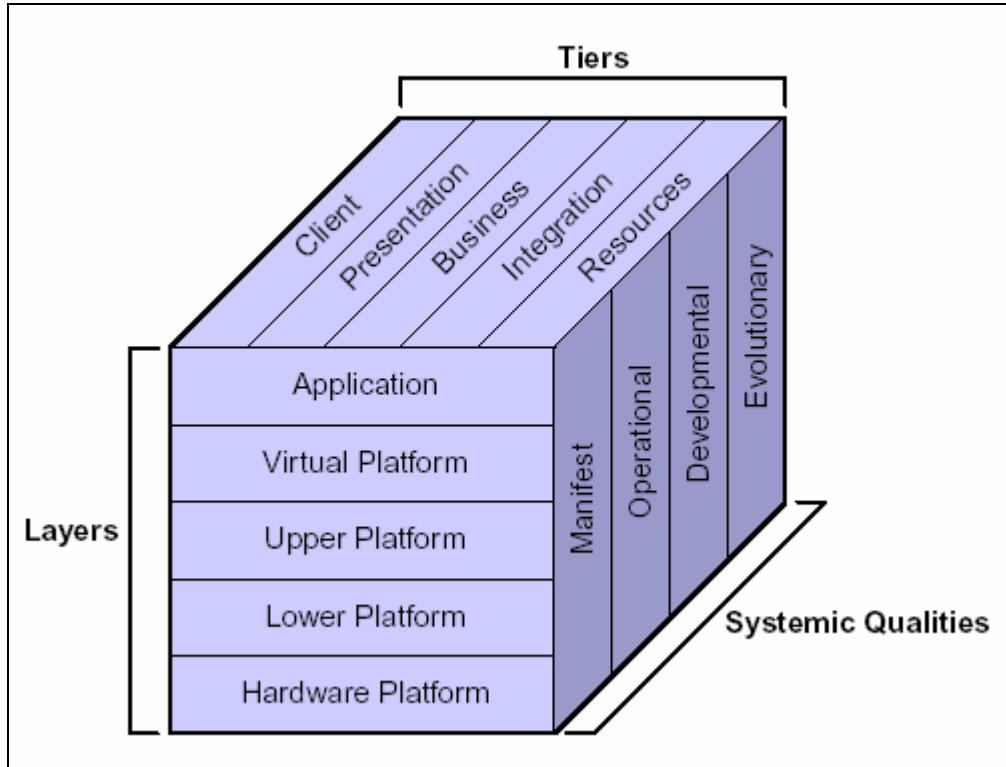
SunTone Architecture 방법론은

- 어플리케이션의 논리적 **concern**을 구분하기 위한 **tier**
- 컴포넌트와 컨테이너의 관계를 구성하기 위한 **layer**
- **tier**와 **layer**에 대한 전략과 패턴을 분석하는 시스템의 질등에 초점을 둔 구조적 다면체를 설명하고 있습니다.

이 다면체는 **3D** 입체 사각형의 다이어그램으로 그릴 수도 있습니다.

이미지

■ **SunTone Architecture** 방법론의 **3D 큐브** 다이어그램



2) Tier

Tier는 “서비스 공급자와 소비자의 ordered chain에 있는 컴포넌트의 논리적 또는 물리적 구조”입니다. (**SunTone Architecture Methodology page 10**)

SunTone Architecture Methodology는 소프트웨어 시스템을 다섯 개의 주요 tier로 나눌 것을 권장합니다.

- **Client** – 웹 브라우저처럼 **thin client**로 구성

클라이언트 티어는 사용자의 반응과 관련된 모든 컴포넌트를 포함합니다.

- **Presentation** – 웹 페이지와 웹 브라우저로 보낼 폼 제공하며, 사용자의 요구 처리

이 티어는 **thin client**와 비즈니스 티어 사이의 베퍼 역할을 하는 티어입니다. 사용자의 **action**에 따라 비즈니스 티어의 메소드를 호출하거나 웹 브라우저에 내용을 보여줍니다. **Thick client**는 **Presentation Tier**를 통하지 않고 곧바로 **Business Tier**로 접근합니다.

- **Business** – 비즈니스 서비스와 엔티티 제공

비즈니스 티어는 어플리케이션의 비즈니스 로직과 규칙, 엔티티를 관리하는 컴포넌트를 제공합니다.

예를 들어, 호텔 예약 시스템에서 **Reservation** 컴포넌트는 비즈니스 엔티티로 나타납니다.

- **Integration** – 비즈니스 티어와 리소스 티어를 통합하기 위한 컴포넌트 제공
이 티어는 근본적인 **CRUD (Create, Retrieve, Update, Delete)** 작업을 수행하는 컴포넌트를 제공합니다.
- **Resource** – 데이터베이스나 **EIS** 같은 백-엔드 자원
이 티어는 비즈니스 데이터를 기록하는 모든 외부 시스템을 의미합니다.

참고하세요

n Fat client and Thin client

fat client : 사용자의 작업이 많이 나타나는 사용자 인터페이스입니다. 예를 들어, 스프레드시트는 사용자의 복잡한 작업 때문에 **fat client**로 구현해야 하는 어플리케이션입니다.
Thin client : 화면에 사용자가 요청한 내용을 보여주는 단순한 작업만 하는 사용자 인터페이스입니다. 주로 웹 서버에서 모든 작업을 해서 정제된 데이터를 보내면 웹 브라우저 (**thin client**)는 화면에 그 내용을 보여주기만 합니다.

3) Layer

Layer는 “주어진 티어 내에서 호스트가 서비스하는 하드웨어와 소프트웨어 스택”입니다.
(**SunTone Architecture Methodology page 11**)

SunTone Architecture Methodology에서는 소프트웨어 시스템을 다섯 개의 주요 레이어로 구분할 것을 제안합니다.

- **Application** – 기능적 요구 사항을 만족하는 컴포넌트의 구현
시스템 솔루션을 구축하기 위해 개발팀에서 만든 모든 컴포넌트입니다. 또는 맞춤형 컴포넌트로 구성된 어플리케이션 레이어를 구입할 수도 있습니다.
- **Virtual Platform** – 어플리케이션 컴포넌트의 구현에 관한 **APIs**
컴포넌트와 컨테이너 인터페이스에 대한 스펙을 제공합니다. 예를 들어, 웹 어플리케이션에서 **Virtual Platform**은 서블릿 인터페이스와 **JSP**의 스펙을 포함하고 있습니다.
- **Upper Platform** – 웹이나 **EJB** 컨테이너, 미들웨어 같은 제품
어플리케이션 레이어 컴포넌트에 대한 패키지를 제공합니다. 여기에는 컴포넌트가 사용하는 컨테이너도 포함됩니다. 예를 들어, **Presentation tier Upper platform**은 톰캣이나 **Sun ONE Application Server** 같은 웹 서버입니다.
- **Lower Platform** – 운영 체제

앞에서 본 레이어들을 지원하는 시스템의 운영 체제를 제공합니다. 경우에 따라 **Upper platform**의 컨테이너의 제약을 받기도 합니다.

- **Hardware Platform** – 서버, 스토리지, 네트워크 디바이스 같은 하드웨어
앞에서 본 레이어들을 지원하는 하드웨어에 대한 설명을 제공합니다.

4) System qualities

“티어와 레이어에 대한 필수적인 서비스의 질을 전달할 전략, 툴, 그리고 수단”
(SunTone Architecture Methodology page 11)

SunTone Architecture Methodology는 다음 네 개의 그룹으로 시스템 질과 관련된 카테고리를 제안합니다.

- **Manifest** – 시스템을 사용하는 엔드-유저를 반영한 시스템 질을 말합니다.
성능, 의존도, 이용가능성, 사용성, 접근성 등이 이에 해당합니다.
- **Operational** – 시스템의 실행을 반영한 시스템의 질을 말합니다.
Throughput, 보안, 서비스 가능성 등이 이에 해당합니다.
- **Development** – 시스템의 물리적 구현, 계획, 예산 등을 반영한 시스템 질을 말합니다.
구현 능력(설계한 내용이 코드상에 쉽게 구현될 수 있는지를 측정), 계획 가능성(예상 비용 추산)
- **Evolutionary** – 시스템의 장기간 소유함에 따른 시스템의 질을 말합니다.
그 예로 코드와 구조의 확장 가능성, 유연성 등을 들 수 있습니다.

아키텍쳐는 취사 선택의 과정입니다. 어떤 시스템 질은 다른 것 보다 더 중요합니다. 아키텍트는 어떤 시스템의 질에 초점을 둘지 적절한 결정을 내려야 합니다.

과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원5 : 비기능적 요구사항 구축

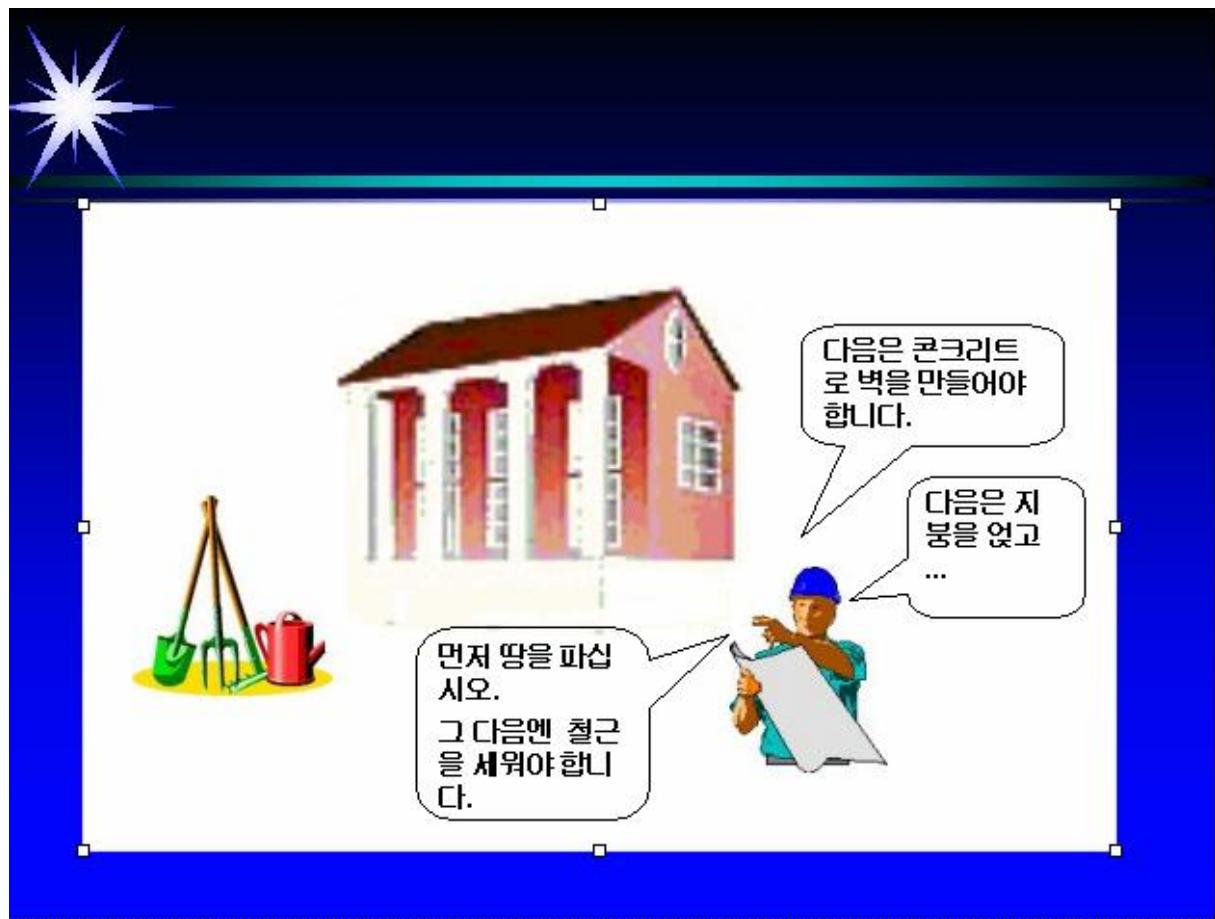
모듈 2 : 아키텍쳐 구축의 Workflow

담당강사 : 전은수

■ 생각해봅시다 ■

지난 시간에 아키텍트의 역할과 아키텍처의 원리 그리고 아키텍처 패턴의 적용에 대해서 알아보았습니다. 그렇다면 아키텍트는 구체적으로 어떤 일을 해야 할까요? 집을 지을 때를 비유한다면 아키텍트는 건축가에 해당합니다. 건축가가 구체적으로 해야 할 일을 알고 계십니까? 또한 한 단계의 작업이 이뤄지고 난 뒤에 어떤 산출물들이 얻어지게 될까요? 얻어진 산출물들은 다음 단계 작업을 위해서 어떻게 쓰여질까요?

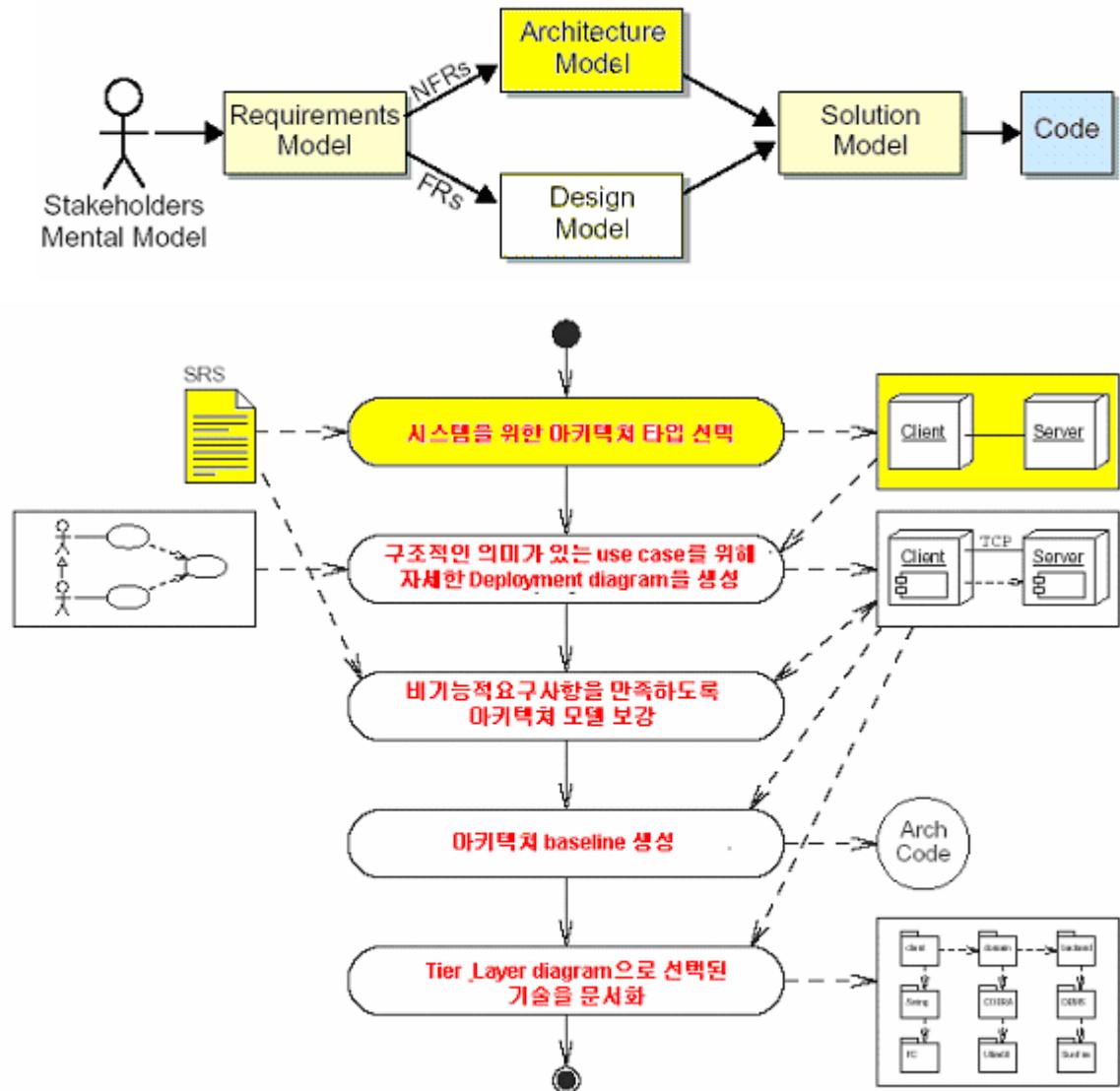
[애니메이션]



■ 학습하기 ■

참고하세요

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. 아키텍쳐 워크플로우

1) 개요

아키텍쳐 워크플로우는 목적한 시스템의 하부구조를 이루기 위해 비기능적 요구 사항을 분석함으로써 시작됩니다.

하부구조 컴포넌트는 비기능적 요구 사항을 만족하는 방법으로 디자인 컴포넌트의 구조를 지원하는 소프트웨어 컴포넌트입니다.

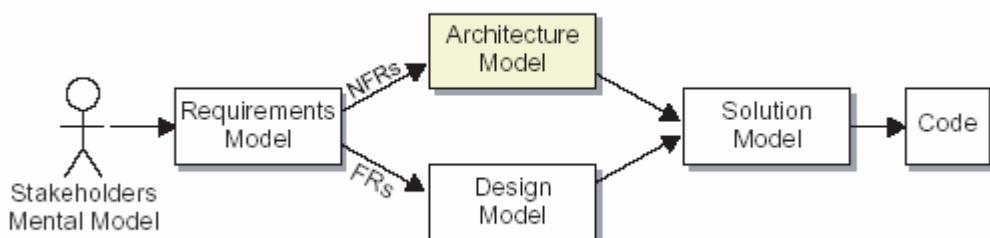
예를 들어, 퍼포먼스나 확장성은 하나 이상의 분산 호스트 서버에서 비즈니스 로직으로 구현이 되어야 합니다. 다중 서버를 통해서 사용자는 좀 더 나은 퍼포먼스를 얻을 수 있습니다. 비즈니스 로직의 분산은 **RMI**나 **CORBA**처럼 분산 하부구조 컴포넌트를 통해서 구현할 수 있습니다.

디자인 워크플로우는 **Requirements Model**(요구 사항 모델)에서 기능적 요구 사항만을 분석한 것입니다. 디자인 컴포넌트는 하나 이상의 유즈케이스를 지원하는 컴포넌트입니다. 예를 들어, 디자인 컴포넌트는 사용자 인터페이스와 비즈니스 서비스, 도메인 엔티티를 포함합니다. 이들 디자인 컴포넌트는 유즈케이스를 지원하기 위해 서로 협력하는 관계를 갖습니다.

아키텍쳐 모델과 디자인 모델은 솔루션 모델로 통합되어 집니다. 솔루션 모델을 통해 프로그래머가 코딩하게 됩니다.

이미지

n OOSD Process에 따른 모델의 변화



2) 아키텍쳐 워크플로우

아키텍쳐 워크플로우는 아키텍트가 초점을 두는 시스템의 질에 상당히 의존적이기 때문에 설명하기가 어렵습니다.

이 과정에서는 아래와 같이 간단하게 정리해 보겠습니다.

| 시스템을 위한 아키텍처 유형을 선택합니다.

아키텍트는 고수준 제약 사항과 비기능적 요구사항을 최대한 만족시킬 수 있는 아키텍처 유형을 선택합니다.

아키텍처 유형이란 **standalone**, **client/server**, **App-centric n-tier**, **web-centric n-tier**, **enterprise n-tier**처럼 가장 기본이 되는 시스템 구조를 말합니다.

선택된 아키텍처 유형에 따라서 고수준 **Deployment Diagram**(배치 다이어그램)이 생성됩니다.

| 아키텍처에 특별한 영향을 받는 유즈케이스를 위하여 상세한 **Deployment Diagram**을 그립니다.

아키텍트는 아키텍처적으로 의미가 있는 유즈케이스를 지원하기 위한 컴포넌트를 보여주기 위해 상세한 배치 다이어그램을 그려야합니다.

이 다이어그램을 그리기 위해 아키텍트는 아키텍처에 의미있는 유즈케이스의 디자인 모델을 그려야만 합니다. 그리고 나서 비기능적 요구 사항을 만족시키는 하부구조 속으로 이 그림을 옮깁니다. 그러므로 아키텍트는 일정 부분 디자이너의 역할까지도 수행해야 합니다.

| 비기능적 요구 사항을 만족 시키는 아키텍처 모델을 보완합니다.

아키텍트는 비기능적 요구 사항을 만족시키는 견고한 하드웨어 구조 속으로 고수준 아키텍처 패턴을 적용시킵니다.

| 아키텍처 **baseline**을 생성하고 테스트합니다.

아키텍트는 “**evolutionary prototype**” 안에서 아키텍처에 특정적인 유즈케이스를 구현합니다. 모든 아키텍처 특정적인 유즈케이스에 대해서 “**evolutionary prototype**”이 구현되었을 때를 아키텍처 베이스라인이 완성되었다고 합니다.

아키텍처 베이스라인은 모든 리스크를 관리하는 시스템의 솔루션 버전을 표현합니다. 아키텍처 베이스라인은 ‘전개’단계의 마지막 포인트이면서 ‘구축’단계의 시작 포인트가 됩니다.

아키텍처 베이스라인은 시스템의 질을 만족시키는가에 대한 검증이 이루어집니다. 여기서 추가적인 패턴을 적용 받기도 하면서 더욱 보완되어 집니다.

| **Tier & Layer Package Diagram**을 통해서 선택한 기술을 문서화합니다.

각 티어와 레이어에서 사용되는 특별한 기술을 구별하고 이것을 **Tier & Layer Package Diagram**으로 그려 문서화합니다.

| 최종적인 **Deployment Diagram**을 통해서 아키텍처 템플릿을 생성합니다.

아키텍처 템플릿은 아키텍처 컴포넌트와 디자인 컴포넌트가 공존하는 상세 배치 다이어그램의 추상 버전입니다. 이것은 구축단계에서 사용될 수 있습니다.

이것은 아키텍쳐 워크플로우의 고정 산출물이 아닙니다.

3) 아키텍쳐 모델

아키텍쳐 워크플로우는 여러 가지 아키텍쳐 모델을 가지고 완결됩니다. 복잡한 시스템에서는 이 모델도 크고 복잡합니다. 다음은 몇가지 아키텍쳐 모델입니다.

| 고수준 배치 다이어그램

이 다이어그램은 분산된 하드웨어 노드에 배치된 컴포넌트들을 보여줍니다.

이 다이어그램은 가장 상위레벨에서 컴포넌트들의 배치를 보여 주는데 사용됩니다.

| 상세 배치 다이어그램

아키텍쳐에 영향을 받는 유즈케이스를 만족시키기 위한 컴포넌트들의 배치를 보여 줍니다. 고수준 배치 다이어그램보다 한층 복잡하고 상세하게 표현되어 있습니다.

| 아키텍쳐 템플릿(Architecture template)

이 템플릿은 상세 배치 다이어그램의 중심 구조를 보여줍니다. 그러나 특별한 디자인 컴포넌트의 표현은 하지 않습니다.

디자이너는 아키텍쳐 모델 속으로 디자인 컴포넌트를 끼워 넣기 위하여 이 템플릿을 사용합니다.

디자인 모델과 아키텍쳐 모델이 합쳐져서 솔루션 모델을 생성합니다.

솔루션 모델은 실제 물리적인 컴포넌트의 생성을 돋는데 사용되는 모델입니다.

아키텍쳐 템플릿은 정해진 산출물은 아닙니다. 그러나 이 과정에서는 디자인 모델로부터 어떻게 솔루션 모델이 도출되는지를 결정할 때 이 템플릿을 사용할 것입니다.

| Tier & Layer 패키지 다이어그램

이 다이어그램은 선택된 기술 안에 있는 패키지의 모형을 보여줍니다.

| 아키텍쳐 베이스라인

아키텍쳐에 영향을 받는 유즈케이스를 구현하기 위한 실제 작업 시스템을 말합니다. 이 코드베이스는 구축 단계에서의 시작점이 되어집니다.

아키텍쳐 모델을 생성하다보면 아키텍트는 디자인 모델이나 솔루션 모델까지도 일부 작성해야 한다는 것을 알게됩니다. 그렇다면 위의 아키텍쳐 모델을 자세히 알아보기 전에 디자인 모델이나 솔루션 모델 그리고 이 절에서 처음 제시된 아키텍쳐 템플릿이 무엇인지를 먼저 알아보겠습니다.

(1) Design Model

아키텍쳐 워크플로우가 진행되는 동안, 아키텍트는 아키텍쳐에 영향을 받는 유즈케이스를 위한 디자인 모델을 생성하면서 일부 디자이너의 역할을 하게 되기도 합니다.

그러나 모든 프로젝트에 아키텍트가 디자이너 역할을 하게 되는 것은 아닙니다. 이미 디자인 모델에 대해 다뤘기 때문에 이 모듈에서는 디자인 모델의 구성 요소 들만 간단히 리뷰하겠습니다.

I Boundary

바운더리 컴포넌트는 사용자 행위에 대응하는 유저 인터페이스 컴포넌트입니다.

I Service

서비스 컴포넌트는 비즈니스 오퍼레이션을 수행합니다. 어떤 서비스 컴포넌트는 유즈케이스의 흐름을 관리하도록 디자인 되기도 합니다.

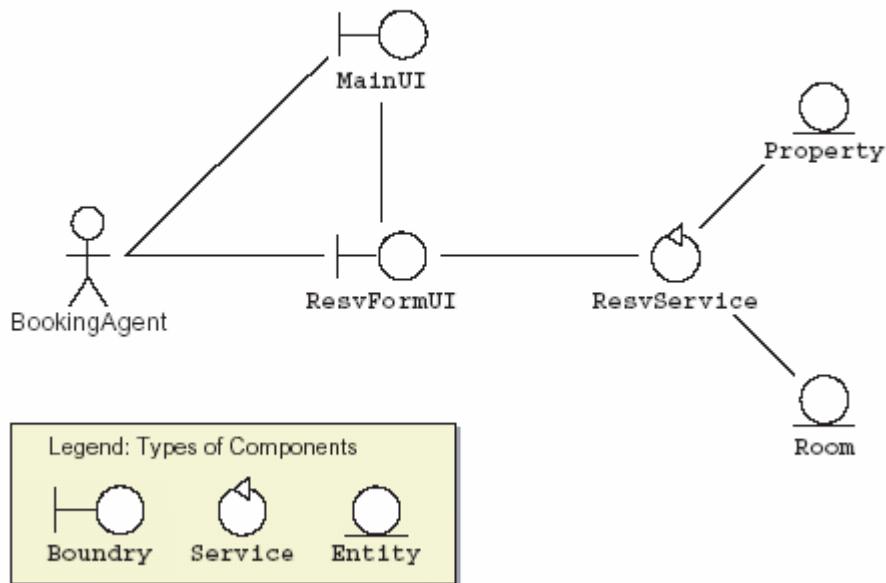
I Entity

엔티티 컴포넌트는 시스템의 도메인 객체를 표현합니다.

이미지

n 디자인 컴포넌트의 예

Example Design Model



(2) Architecture Template

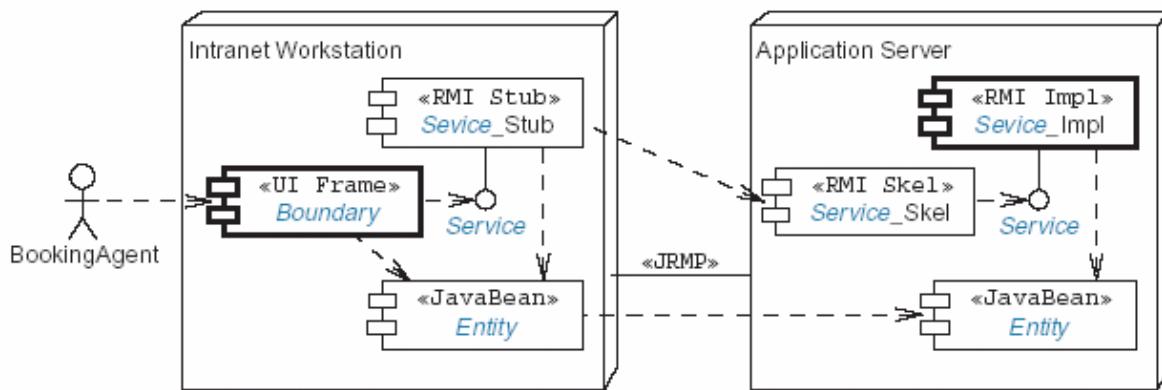
아키텍트는 아키텍쳐 템플릿을 생성합니다. 이 템플릿은 디자인 컴포넌트 구현하는 물리적인 컴포넌트 조합을 보여줍니다. 다시 말해서 디자인 컴포넌트가 각 하드웨어 노드에서 어떻게 유기적인 관계를 맺고 있는지를 보여주는 그림입니다.

따라서 이 템플릿에는 디자인 모델의 요소 (**Boundary, Service, Entity**) 가 보여지고 이들 컴포넌트들이 어떤 역할을 하는지가 간략히 명시되어 있으며 서로간의 관계가 점선 화살표로 표기됩니다.

[이미지]

n 아키텍쳐 템플릿의 예

Example Architecture Template



이 그림으로 아키텍쳐 컴포넌트와 디자인 컴포넌트의 관계를 설명할 수 있습니다.

디자인 컴포넌트들은 유즈케이스를 기능적으로 구현합니다. 아키텍쳐 컴포넌트들은 서비스의 질을 어떻게 만족시킬 것인지에 관한 구조를 보여줍니다.

BookingAgent는 예약 담당 직원입니다. 이 직원이 시스템을 사용할 때의 시스템의 내부적인 구조와 최소한의 컴포넌트를 표현한 것입니다. 시스템은 최소 두대의 분산 컴퓨팅을 이루고 있습니다. 인트라넷 웍스테이션은 예약 담당 직원이 사용하는 하드웨어 노드를 표현한 것이고 어플리케이션 서버는 실제 비즈니스가 실행되는 하드웨어 노드입니다. 이 두 하드웨어 노드 사이에는 **RMI**프로토콜이 사용되고 있습니다. 즉, **RMI**로 하부 컴포넌트들을 구축하겠다는 의미입니다.

예약 담당 직원이 시스템에 접속하면 바운더리 컴포넌트가 **UI** 프레임으로 사용자 인터페이스를 보여주고 어떤 액션이 일어났을 때 **RMI** 컴포넌트들로 그 요청 내용이 전달 되어지며 이것은 어플리케이션 서버에 배치되어 있는 서비스 구현 객체에서 실행되게 됩니다.

(3) Solution Model

솔루션 모델은 아키텍쳐 템플릿 속의 디자인 모델들의 실제 이름을 갖는 그림입니다. 따라서 이 솔루션 모델로는 실제 구현 시에 사용하게 될 컴포넌트의 이름을 제공 받

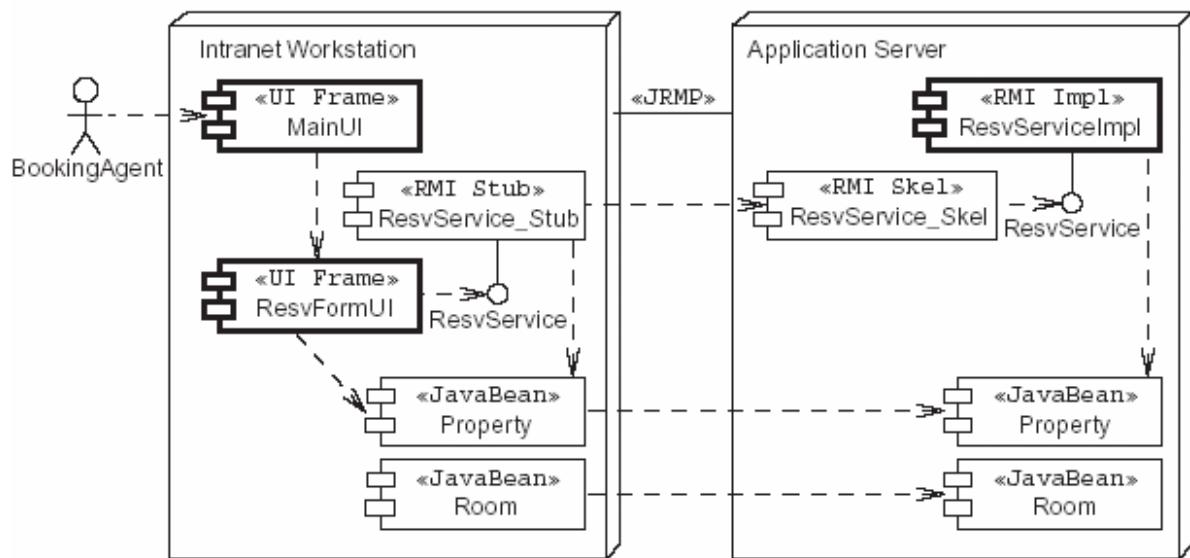
을 수 있습니다.

전 장에서 보았던 아키텍쳐 템플릿 예제의 그림을 솔루션 모델로 생성했을 때는 다음 그림과 같습니다.

이미지

n 솔루션 모델의 예

Example Solution Model



같은 내용의 그림이지만 일단 컴포넌트의 종류를 명시하는 대신에 이름을 기재했다는 것에 주목하십시오. 또한 엔티티 컴포넌트도 필요한 만큼 추가가 됩니다.

2. Key Architecture Diagrams

아키텍쳐 모델의 뷰는 **UML** 다이어그램에서의 **3가지** 타입을 사용합니다.

- | **Package Diagram**
- | **Component Diagram**
- | **Deployment Diagram**

이들을 자세히 살펴보겠습니다.

1) Package Diagram

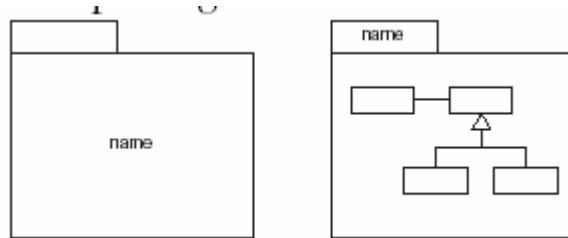
패키지 다이어그램은 어떤 요소들을 그룹화 하기 위해 사용합니다.

UML 패키지 다이어그램은 패키지 사이의 의존관계도 보여줍니다.

패키지에는 다른 **UML** 요소나 다이어그램이 들어갑니다.

[이미지]

n UML Package Diagram의 요소



패키지 이름은 바디 박스안에 둘 수도 있고 이름 박스안에 둘 수도 있습니다. 왼쪽 그림처럼 바디 박스안에 이름을 표현 할 경우 이 패키지의 구성요소에 대해서는 표현하지 않겠다는 것입니다. 오른쪽 그림과 같이 이름 박스안에 이름을 표기 했을 때는 바디 박스안에 이 패키지가 포함하고 있는 요소나 다른 디아이어그램을 표현할 수 있습니다.

[참고하세요]

n 패키지 디아이어그램

패키지 디아이어그램은 표준 **UML** 스펙에 정의된 디아이어그램은 아닙니다. 그러나 패키지 디아이어그램으로 패키지간의 의존 관계를 표현해 놓으면 구현 시 작업의 순서를 정할 수 있는 이점이 있습니다.

n UML 패키지 VS. 자바 패키지

UML 패키지와 자바 패키지는 다릅니다.

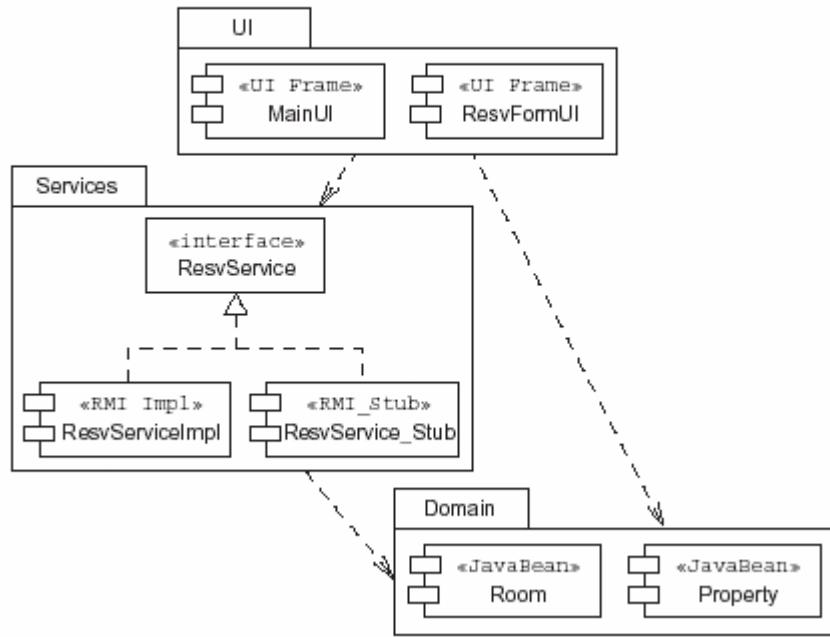
그러나 **UML** 패키지를 자바 패키지로 활용 할 수는 있습니다.

다음 그림은 패키지안에 컴포넌트들이 포함된 예를 보여주고 있습니다.

[이미지]

n 패키지 디아이어그램의 예

Example Package Diagram

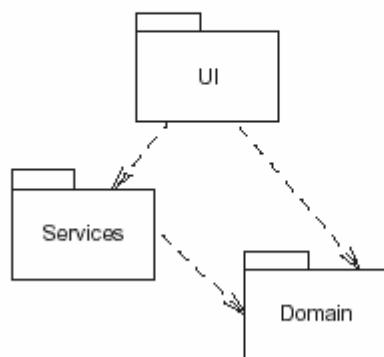


또한 패키지들간의 의존 관계만 묘사할 수도 있습니다.

[이미지]

n 추상적 패키지 다이어그램의 예

An Abstract Package Diagram



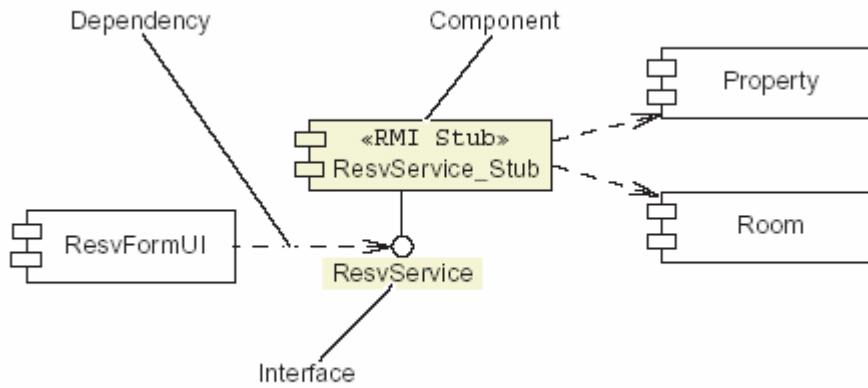
2) Component Diagram

(1) 목적

컴포넌트 다이어그램은 시스템의 물리적인 소프트웨어 조각과 그들의 관계를 표현합니다. 하나의 컴포넌트는 다른 컴포넌트와 협력관계를 맺습니다. 이 협력관계는 **dependency**로 표현됩니다.

[이미지]

n 컴포넌트 디아어그램의 예



협력관계를 표현하는 접선 화살표를 **Dependency**라고 합니다. 이것은 협력 관계의 흐름을 파악하게 해줍니다.

(2) 특징

컴포넌트 디아어그램은 다양하게 사용됩니다. 다음은 컴포넌트 디아어그램의 특징입니다.

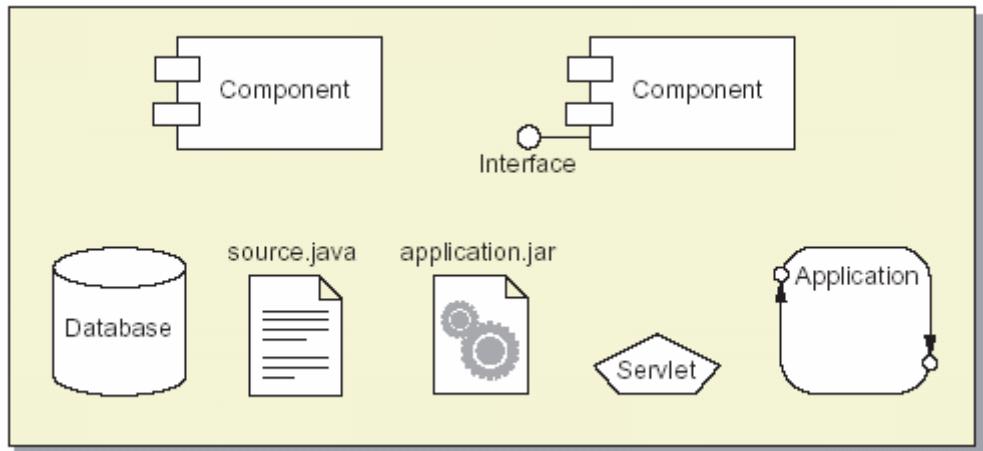
- | 컴포넌트는 소프트웨어 단위입니다
- | 컴포넌트는 크고 추상적일 수 있습니다.
- | 컴포넌트는 작을 수도 있습니다.
- | 컴포넌트는 다른 컴포넌트로 서비스를 제공하는 인터페이스를 갖습니다.
- | 컴포넌트는 소스 코드 파일이나 오브젝트 파일이나 데이터 파일 같은 파일이 될 수 있습니다. 심지어는 **HTML**이나 미디어 파일이 될 수도 있습니다. 많은 것들이 컴포넌트가 될 수 있습니다. 그러나 그들 모두가 ‘수행적’이지는 않습니다. **HTML**같은 것은 정적 컴포넌트의 좋은 예가 될 수 있습니다. 컴포넌트는 또한 다른 컴포넌트와 함께 구성될 수 있습니다. 예를 들어, **.jar** 파일은 컴포넌트 클래스들의 묶음입니다.

(3) 유형

언급한 듯이 컴포넌트의 종류는 다양합니다. 따라서 컴포넌트를 표현하는 그림도 다양합니다. 다음 그림에서 위의 두 아이콘은 일반적으로 사용되는 컴포넌트 타입입니다.

[이미지]

n 소프트웨어 컴포넌트의 유형



참고하세요

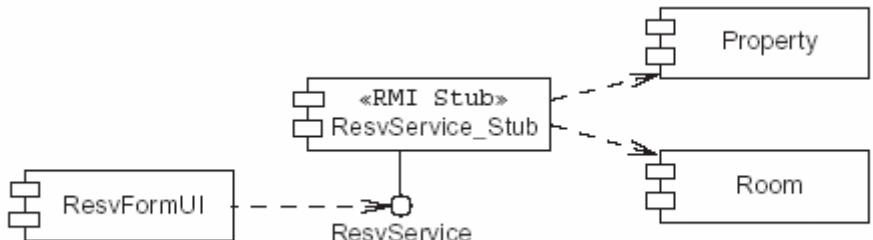
UML에서는 컴포넌트의 특별한 유형을 표현하는 아이콘을 자유롭게 사용하는 것을 허락하고 있습니다. 예를 들어, **Servlet** 컴포넌트를 오각형으로 표현하는 것 등, 여러분들의 프로젝트에서 의미를 가지는 모양으로 자체 제작하여 사용할 수 있습니다.

(4) 사례

- | 컴포넌트 다이어그램은 컴포넌트 사이의 의존도를 보여줍니다.

이미지

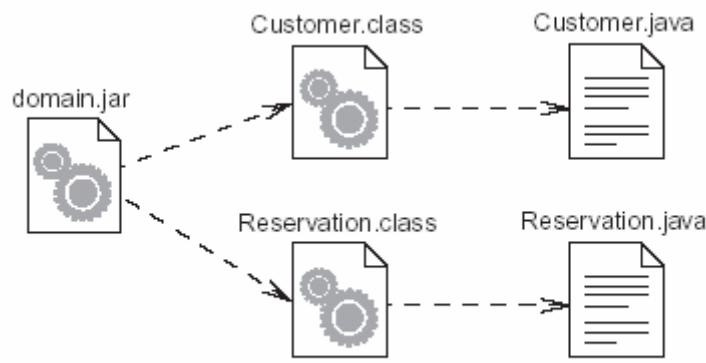
- | 의존관계를 표현한 컴포넌트 다이어그램의 예



- | 컴포넌트 다이어그램은 소프트웨어 구조를 표현할 수 있습니다.

이미지

- | 구조를 표현한 컴포넌트 다이어그램의 예



심화학습

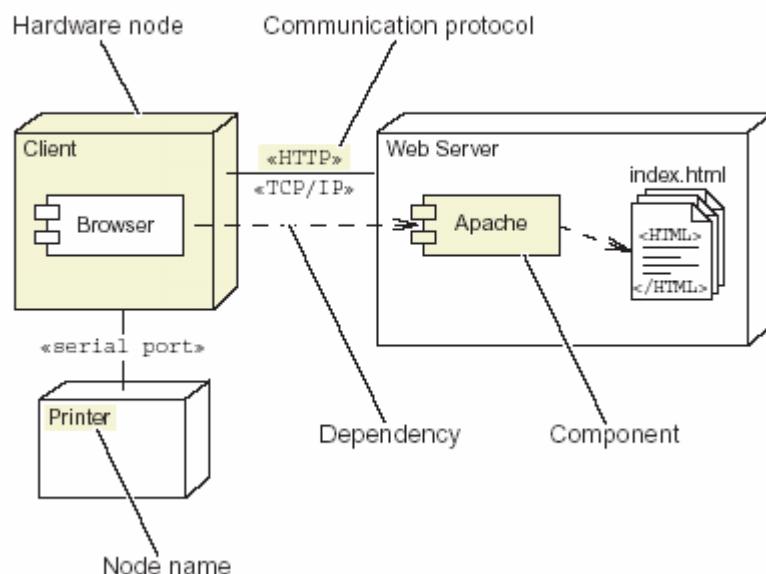
여러분이 경험했던 프로젝트나 앞으로 진행하게 될 프로젝트에서 특별한 컴포넌트 유형이 있는지를 생각해 보십시오. 컴포넌트 디자인그램은 UML에서 가장 유동적인 디자인그램 중의 하나입니다. 어떤 그림이든 그것이 해당 컴포넌트를 잘 표현할 수 있는 의미를 지닌다면 프로젝트에서 참가자들간의 협의 후에 사용 할 수 있습니다.

3) Deployment Diagram

Deployment Diagram(이하 배치 디자인그램)은 시스템의 물리적인 하드웨어와 분산된 컴포넌트들의 관계를 표현한 디자인그램입니다.

이미지

n 배치 디자인그램의 예



(1) 목적

배치 다이어그램의 각 요소들은 다음과 같은 목적을 가지고 표현됩니다.

- | 하드웨어 노드는 물리적인 하드웨어를 표현할 수 있습니다.
하드웨어 노드는 일반적으로 컴퓨터입니다. 그러나 그것은 컴퓨터 네트워크를 통해 커뮤니케이션 할 수 있는 어떤 종류의 디바이스도 의미 할 수 있습니다. 예를 들어, 프린터나 스캐너, **PDA**, **cell phone**, 네트워크 라우터, 방화벽 등도 전부 하드웨어 노드로 표현합니다.
- | 하드웨어 노드 사이의 링크는 커넥션과 프로토콜을 의미합니다.
배치 다이어그램은 실제 컴퓨터 네트워크 상의 하드웨어 노드들의 관계를 보여주기 위해 링크를 사용합니다. 이 링크는 실선으로 표현되고 링크 위에 라벨을 붙임으로써 커넥션이나 프로토콜을 보여 줄 수 있습니다.
- | 소프트웨어 컴포넌트는 네트워크를 경유하여 분산된 소프트웨어를 보여 주기 위해 하드웨어 노드 내에 위치합니다.
배치 다이어그램은 컴포넌트가 어떤 하드웨어 노드 위에 위치 하는지를 보여줍니다.
이러한 컴포넌트들은 일반적으로 스케일이 큰 컴포넌트입니다. 예를 들어, 브라우저처럼 수행 어플리케이션이기도 하고 **JDBC** 컴포넌트처럼 클래스 라이브러리이기도 합니다.

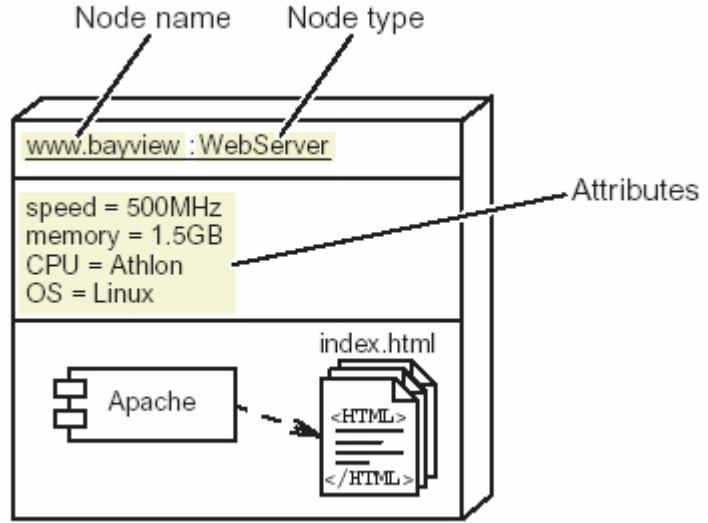
(2) 유형

배치 다이어그램에는 두 가지 유형이 있습니다.

- | **Descriptor** 배치 다이어그램
이것은 개념적인 배치 다이어그램입니다. 하드웨어 노드를 가장 기본적인 형태만 표현합니다.
- | **Instance** 배치 다이어그램
이것은 실제 실행 시에 갖게 될 물리적인 구조를 자세히 표현한 것입니다. 여기에는 하드웨어의 이름 뿐만 아니라 타입과 성능등이 표현되고 컴포넌트도 역시 파일의 이름까지 상세히 표현 될 수 있습니다.

[이미지]

▣ **Instance** 배치 다이어그램의 예



3. Architecture Types

아키텍쳐 타입을 선택하는 것은 시스템의 하드웨어 구조와 최상위 소프트웨어의 형태를 개발팀에게 제공합니다.

선택한 아키텍쳐 유형은 고 수준 배치 다이어그램으로 그려둡니다.

아키텍쳐는 다양한 인자에 의존적입니다.

- ① 시스템의 요구사항 중에서 플랫폼에 관한 제약 사항에 따라 어떤 아키텍쳐를 가질 것인지를 결정됩니다.

SRS에는 종종 플랫폼에 관한 요구 사항이 명시되어 있습니다. 예를 들어, 운영체제라던가 어플리케이션 플랫폼이라던가 하드웨어 같은 것을 언급하는 것입니다. 이것은 아키텍쳐를 결정 짓는 중요한 인자가 됩니다.

- ② 사용자 상호작용의 방식은 아키텍쳐 유형을 결정 짓을 수 있는 인자입니다.

사용자가 시스템을 어떻게 사용하느냐는 아키텍쳐 유형을 결정 짓는 인자가 됩니다. 예를 들어, 호텔 예약 시스템의 경우, 고객은 인터넷으로 시스템에 접속할 수 있다면 이 시스템은 웹 서버를 포함하고 있어야 함을 의미합니다.

웹 뿐만 아니라 **GUI**, 음성, 휴대용 디바이스 등 다양한 상호 작용 방식이 있습니다.

- ③ 저장 메커니즘에 따라 아키텍쳐가 영향 받습니다.

스토리지에 따라 아키텍쳐가 달라질 수 있습니다.

예를 들어, **DBMS**나 **EIS**가 무엇이냐에 따라 플랫폼 유형이 바뀔 수 있는 것입니다.

- ④ 위와 유사하게 데이터와 트랜잭션을 무결성은 아키텍처 유형을 결정하는데 영향을 미칠 수도 있습니다.

수많은 아키텍처 유형 중에서 성공적인 몇 가지만 공부하도록 하겠습니다.

- ① **Standalone applications**
- ② **Client/Server(2-tier) applications**
- ③ **N-tier applications**
- ④ **Web-centric n-tier applications**
- ⑤ **Enterprise n-tier applications**

1) Standalone applications

Standalone application은 사용자 워크스테이션이 호스트가 되는 싱글 어플리케이션 컴포넌트로 구성됩니다.

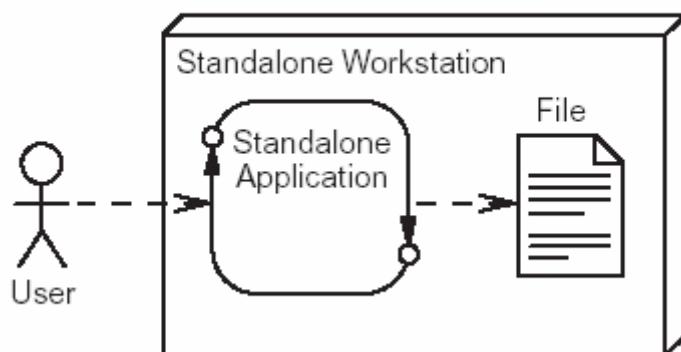
이것은 매우 평범하고 단순한 구조입니다. 워드 프로세서나 그래픽 어플리케이션이나 텍스트 편집기 같은 것이 예가 될 것입니다. 이 아키텍처는 **DBMS** 같은 데이터 스토리지를 사용하지도 않고 네트워크 커뮤니케이션을 필요로 하지도 않습니다.

그러나 이 아키텍처 타입은 어플리케이션의 변경이 일어났을 때 모든 사용자 워크스테이션에서 업데이트가 이뤄져야 한다는 단점이 있습니다. 이것을 ‘**deployment problem**’이라고 부릅니다. 그러나 이것이 대부분의 **Standalone** 어플리케이션에 중대한 문제가 되지는 않습니다. 업그레이드는 사용자가 결정할 문제이기 때문입니다.

하지만 어플리케이션을 통해 생성된 파일이 다른 사용자와 공유되어 할 때는 문제가 될 수 있습니다. 소프트웨어의 다른 버전은 서로 다른 파일 포맷을 가질지도 모르기 때문입니다.

[이미지]

n 일반적인 Standalone Architecture Type



2) Client/Server(2-tier) Applications

Client/server 어플리케이션은 사용자 워크스테이션이 호스트가 되고 데이터 저장 장치와 커뮤니케이션을 하는 구조를 말합니다. 여기에는 두 가지 서브타입이 있습니다.

I thin client

이것은 클라이언트가 어떤 비즈니스 로직도 포함하지 않는 **UI**로만 구성되어 있는 형태를 말합니다.

참고하세요

n Thin client

thin client에 관한 정의는 다음의 웹 사이트에서 얻을 수 있습니다.

<http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?thin+client>

I thick client

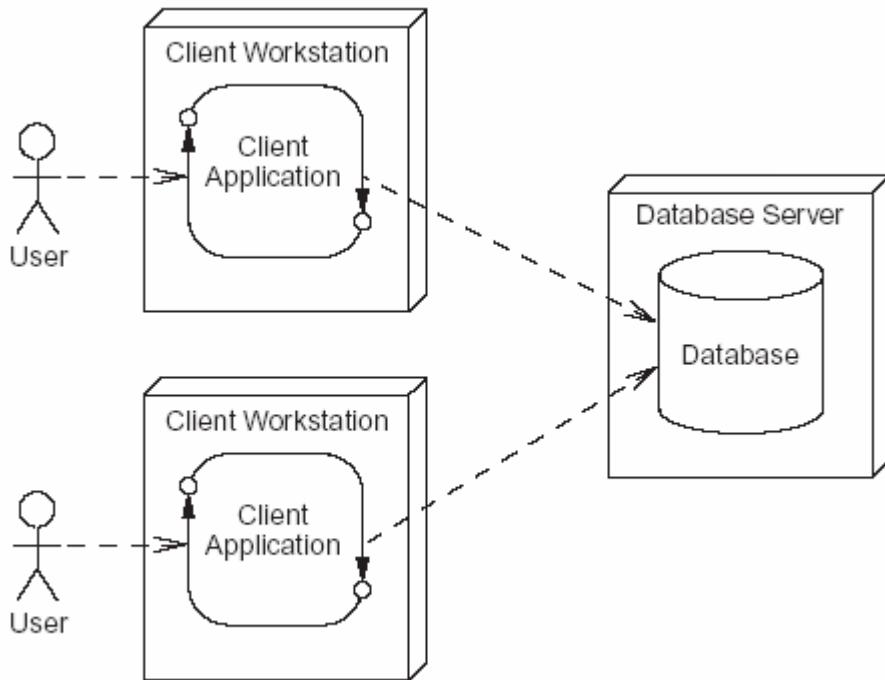
이것은 클라이언트 티어에서 비즈니스 로직이 제공되는 것을 말합니다.

이 아키텍처 타입도 '**deployment problem**'의 단점이 있습니다.

예를 들어, 모든 클라이언트가 최신 버전을 가져야지만 데이터가 제대로 자장 된다든가 데이터 저장 장치가 업그레이드 되었을 때 클라이언트 프로그램도 업그레이드가 되야 한다든가 하는 문제입니다.

이미지

n Client/server architecture type



3) N-Tier Applications

n-tier 어플리케이션은 여러 대의 머신을 가집니다.

n-tier 어플리케이션은 크게 세 종류로 나뉩니다.

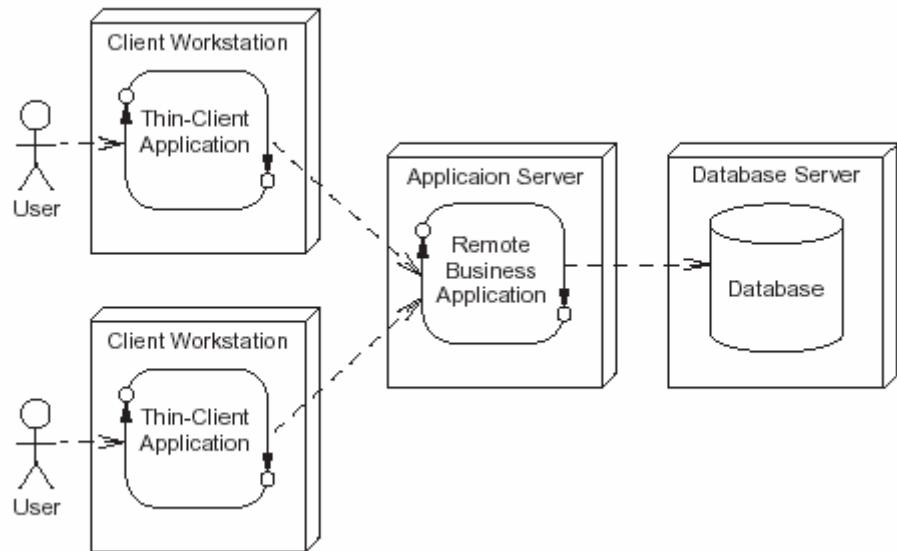
I Application-centric

클라이언트는 **thin client**가 되고 분리된 어플리케이션 서버 위에 비즈니스 컴포넌트가 배치됩니다. 이 구성에서 어플리케이션 서버는 데이터 보존을 관리하고 데이터를 저장하기 위한 모든 오퍼레이션을 수행합니다.

클라이언트가 분리된 구조의 장점은 비즈니스 로직의 변경이 필요 할 때 오직 어플리케이션 서버의 업그레이드만 일어나면 된다는 것입니다.

[이미지]

n Application-centric N-tier Architecture type



I Web-centric

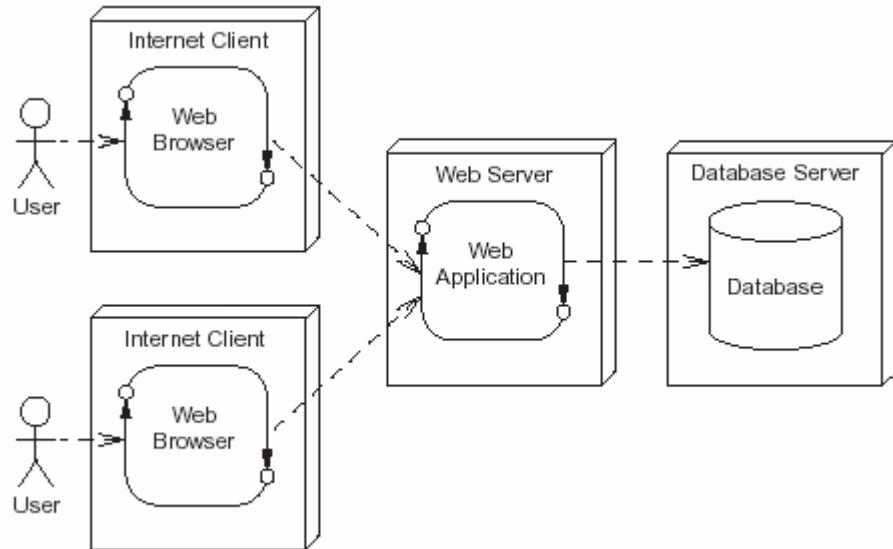
이것은 웹브라우저를 통해 어플리케이션에 접근할 수 있는 구조입니다.

이 아키텍처의 중대한 단점은 사용자가 회사 내의 사원이라 할지라도 웹 어플리케이션을 사용해야 한다는 것입니다. 이것은 인터넷을 통해 내부 비즈니스 평면이 액세스 되도록 만들어지기 때문에 보안에 취약할 수 있습니다.

그러나 웹 어플리케이션이 업그레이드 되면 모든 사용자에게 새 어플리케이션이 즉시 보여질 수 있다는 장점이 있습니다.

[이미지]

n Web-Centric N-tier Architecture Type



I Enterprise

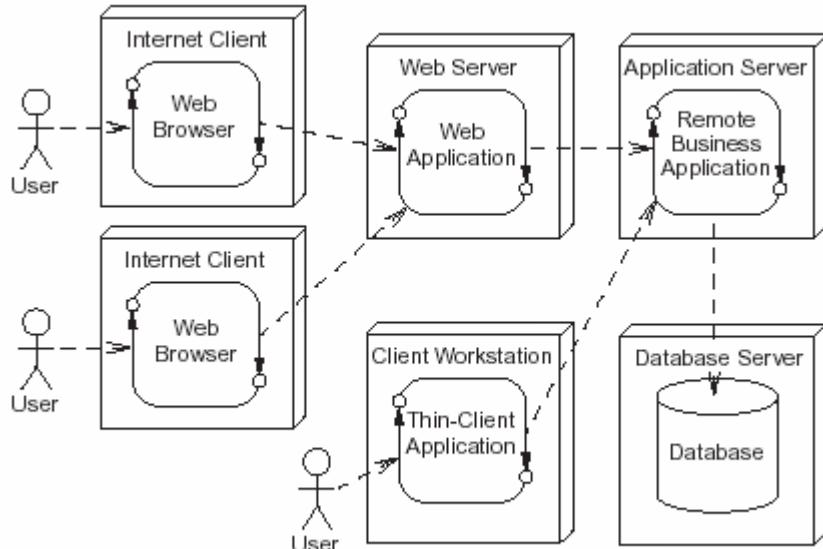
이것은 **web-centric**과 **application-centric** 아키텍처의 혼합 구조입니다.

클라이언트는 웹 브라우저 뿐만 아니라 **thin-client**로도 개발됩니다. 브라우저를 통한 접근은 웹 서버 위의 **Web application**이 맡고 **Thin-client**의 접근은 어플리케이션 서버 위의 원격 비즈니스 어플리케이션이 맡습니다.

이 구조의 최대 단점은 복잡하다는 것입니다.

[이미지]

n Enterprise N-tier Architecture Type



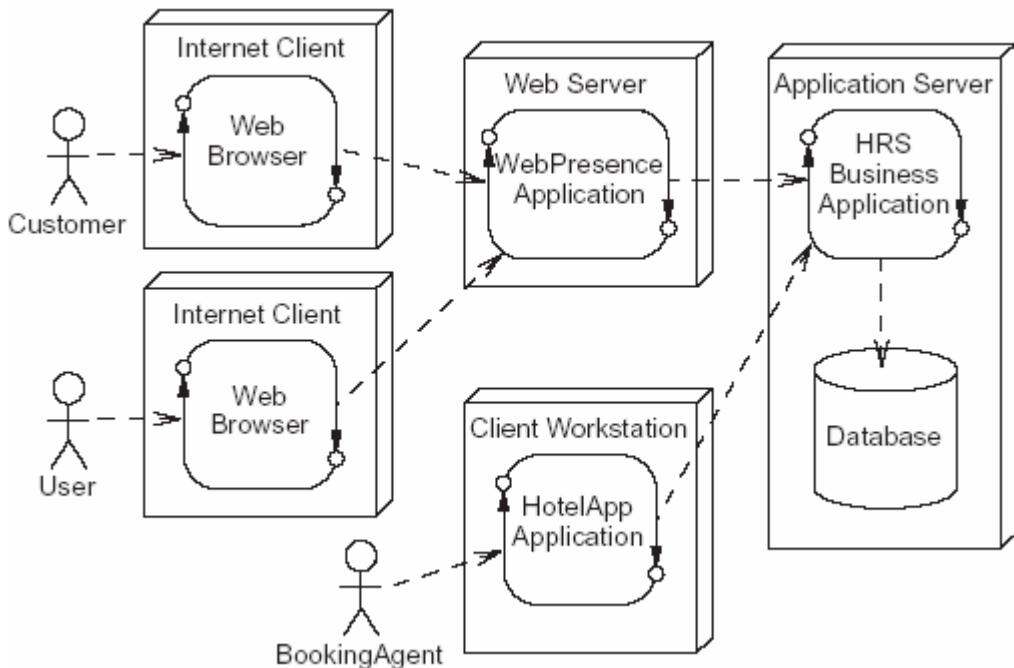
4) 사례

Bay-View 의 경우 ACME 컨설턴트들은 **Enterprise n-tier architecture type**을 선택했습니다.

때문에 고객은 웹 브라우저로 시스템에 접근할 수 있고 호텔 내 직원들은 **thin client** 어플리케이션을 통해 접근할 수 있게 되었습니다.

[이미지]

n 호텔 예약 시스템의 고수준 배치 다이어그램



4. 아키텍쳐 워크플로우의 산출물 생성

1) 상세 Deployment Diagram 생성

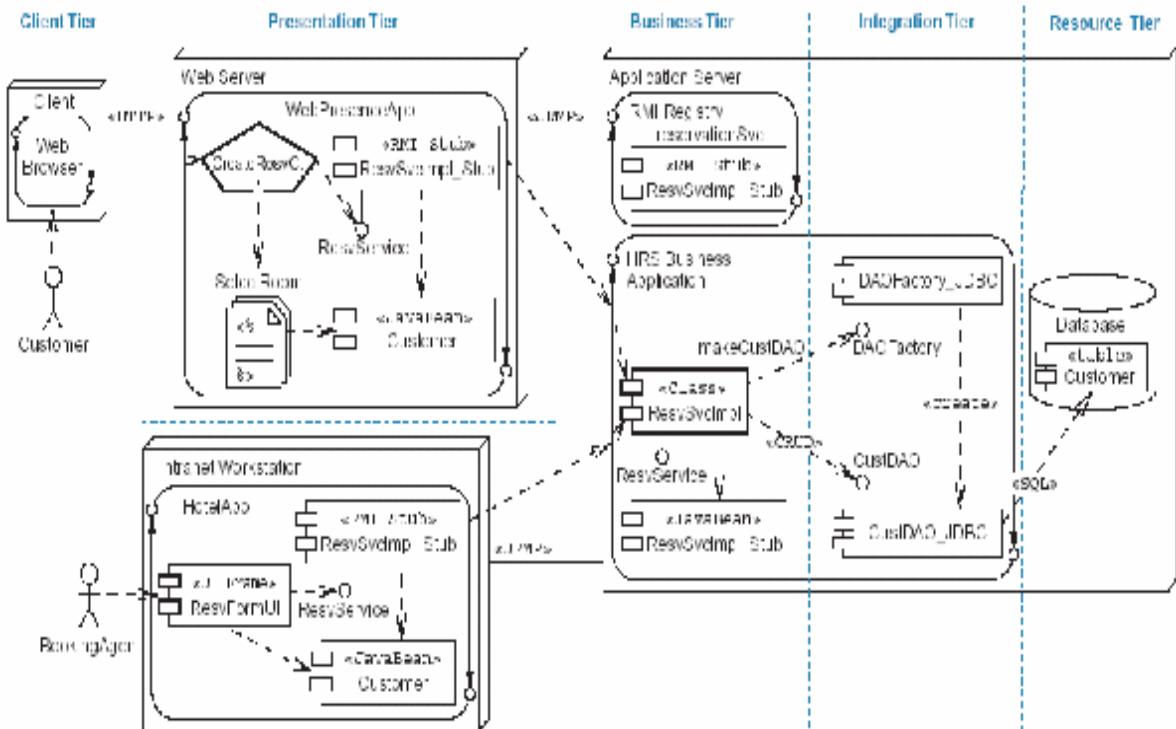
아키텍쳐 워크플로우를 따라 작업을 진행하다 보면 각종 산출물이 생성되게 됩니다. 그 중 상세 배치 다이어그램은 주요 다이어그램입니다.
이 다이어그램에는 컴포넌트들의 구체적인 이름까지 명시되어 있습니다.

상세 배치 다이어그램을 생성하기 위해서는 다음과 같은 작업을 합니다.

1. 아키텍쳐에 영향을 받는 유즈케이스를 위한 컴포넌트를 디자인합니다.
이 단계에서 아키텍트가 바운더리 컴포넌트, 서비스 컴포넌트, 엔티티 컴포넌트를 만드는 디자이너의 역할까지 감당해야 하는 것입니다.
2. 아키텍쳐 모델 속으로 디자인 컴포넌트를 위치 시킵니다.
3. 디자인 컴포넌트와 하부구조 컴포넌트를 융화시킵니다.

[이미지]

n 상세 배치 다이어그램의 예



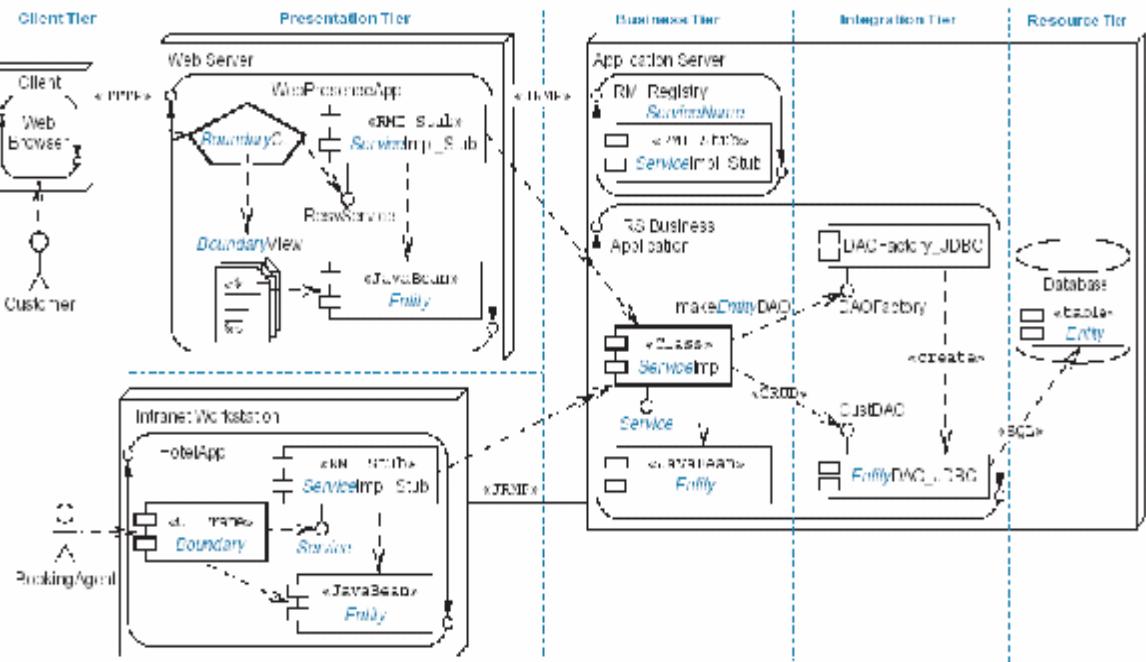
2) Architecture Template 생성

아키텍쳐 템플릿을 생성하기 위해서는 다음과 같은 작업을 합니다.

1. 상세 배치 디아이어그램의 컴포넌트들을 묶어 디자인 컴포넌트 유형으로 표현합니다.
상세 배치 디아이어그램보다 간결해집니다.
2. 디자인 컴포넌트의 이름을 타입으로 대신합니다.
예를 들어, **ResvSvcImpl_Stub**을 **ServiceImpl_Stub**으로 대신하는 겁니다.

[이미지]

■ Architecture Template의 예



3) Tiers and Layers Package Diagram 생성

Tier & Layer Package Diagram을 생성하기 위해서는 다음과 같은 작업을 합니다.

1. 어떤 어플리케이션 컴포넌트가 있는지 결정합니다.

어플리케이션 레이어에서 각 **Tier**마다 어떤 컴포넌트가 필요한지를 결정합니다. 예를 들어, 프리젠테이션 티어에는 **WebPresenceApp component**가 필요하다고 결정하는 것입니다.

2. 어떤 기술 API와 커뮤니케이션 프로토콜이 사용되는지 그리고 특별한 컴포넌트가 사용되는지를 결정합니다.

Virtual Platform Layer에서는 어떤 API 가 사용되는지 어떤 프로토콜이 사용되는지를 표현합니다. 예를 들어 **Java servlet & JSP** 기술이 사용 되었음을 버전과 함께 표현할 수 있습니다.

3. 사용할 컨테이너 제품을 결정합니다.

Upper Platform Layer에서는 어플리케이션 플랫폼을 표현합니다.

예를 들어, **Tomcat**을 사용하거나 **J2SE**를 사용하는 것을 표현하는 것입니다.

4. 사용할 운영체제를 결정합니다.

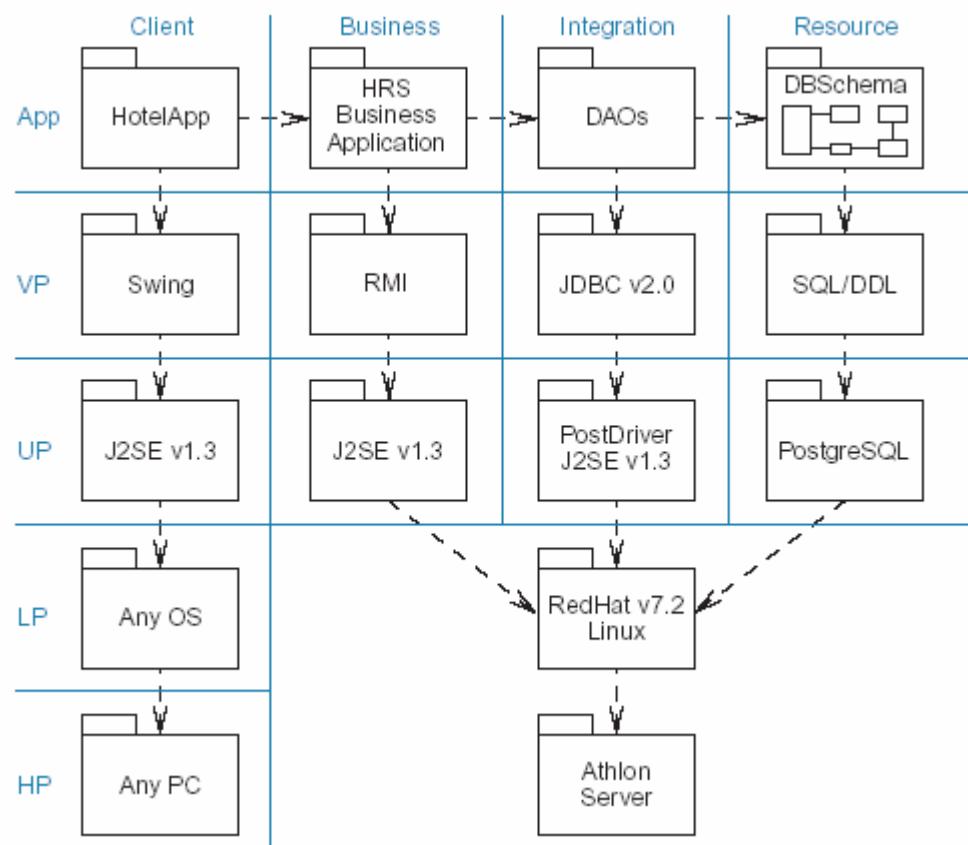
Lower Platform Layer에서는 OS를 표현합니다. 예를 들어, **Windows**나 **RedHat v7.2 Linux**처럼 표현하는 것을 말합니다.

5. 사용할 하드웨어를 결정합니다.

Hardware Platform layer에서는 호스트 머신의 종류를 표현합니다. 이 때 머신의 성능이나 타입등을 같이 표현할 수 있습니다.

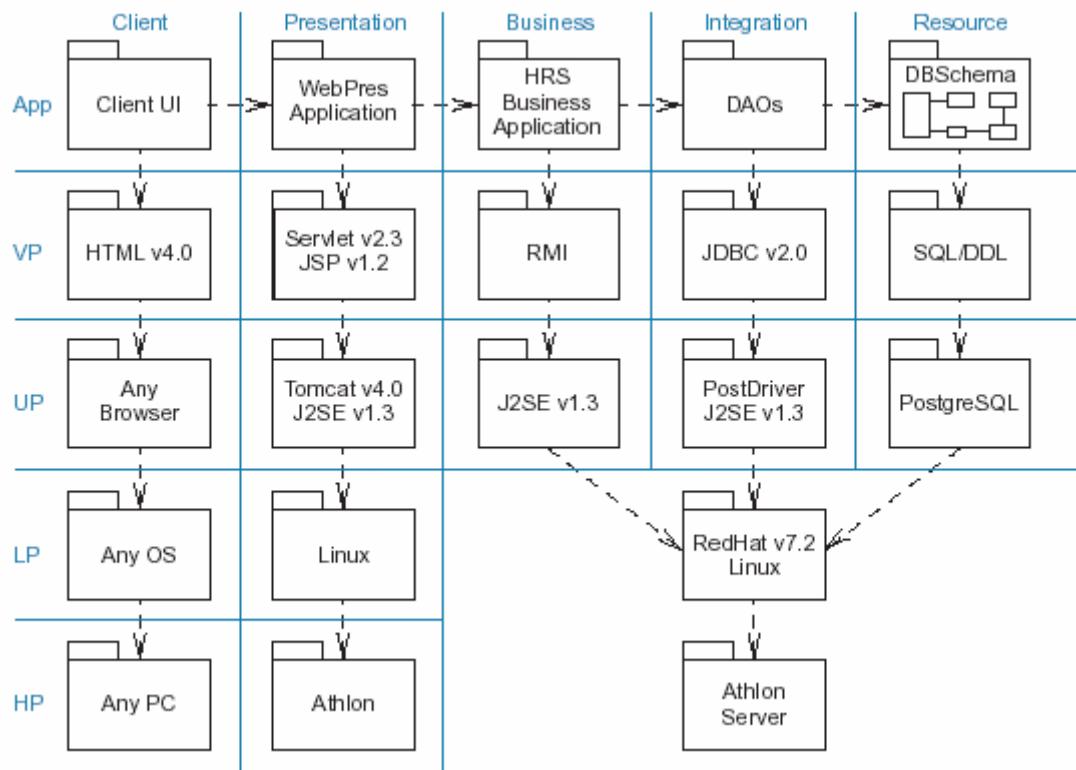
[이미지]

n 호텔예약시스템의 Tier & Layer diagram



[이미지]

n 호텔예약시스템의 웹 어플리케이션의 Tier & Layer diagram



과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원5 : 비기능적 요구사항 구축

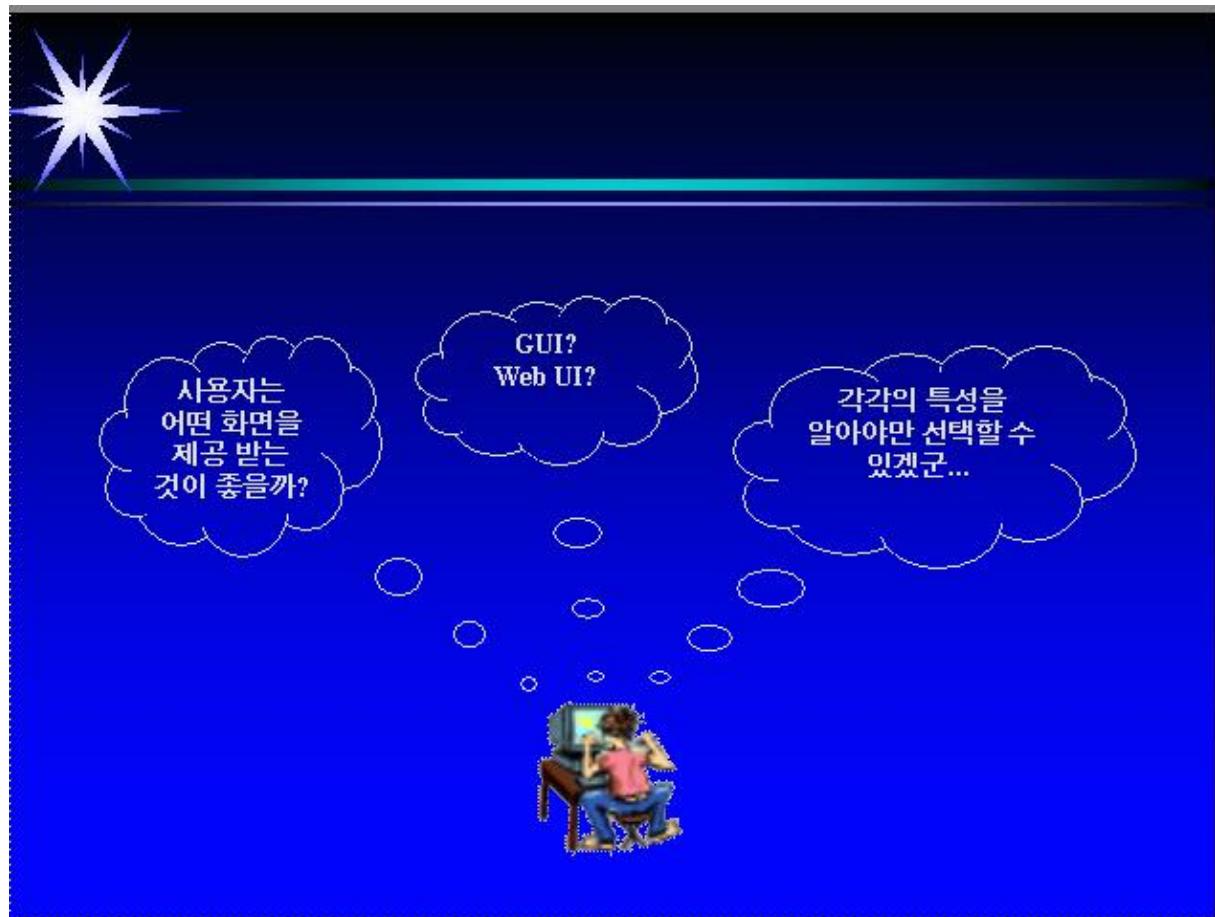
모듈 3 : Client & Presentation Tier 의 아키텍쳐 모델 생성

담당강사 : 전은수

■ 생각해봅시다 ■

이전 모듈(단원5.비기능적 요구 사항 구축- 모듈2.아키텍쳐 구축의 워크플로우)에서 다뤘던 아키텍쳐 워크플로우 산출물 중에서 **Tier & Layer Package Diagram**은 분산 컴퓨팅에서 각 티어별 기술과 컴포넌트를 표현한 것입니다. 그렇다면 클라이언트 티어에서의 어플리케이션 컴포넌트는 구체적으로 어떤 것이 있을까요? 사용자는 시스템에 접근하기 위해 어떤 어플리케이션을 제공 받을까요? 먼저 떠오르는 답이 컴퓨터 모니터에 떠오르는 그래픽컬한 화면일 것입니다. 이것을 **GUI**라고 합니다. 어떤 분들은 “인터넷으로 접근합니다”라고 대답하셨을 것입니다. 네, 그것을 **Web UI**라고 합니다. 그러면 **GUI**는 어떻게 만들어야 할까요? **Web UI**는 어떻게 만들어야 할까요? 또 그들을 문서화 하려면 어떻게 해야 할까요?

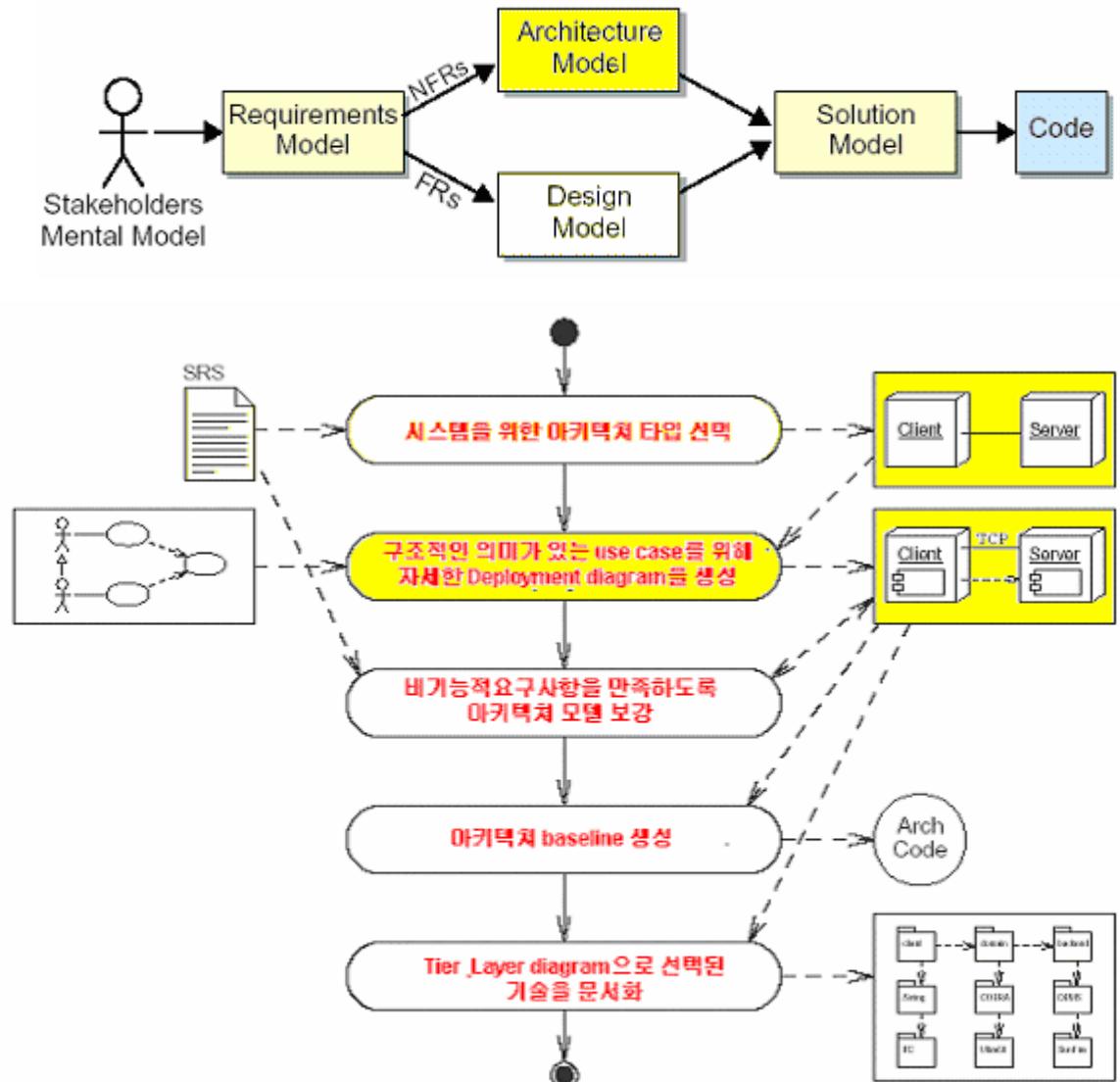
애니메이션



■ 학습하기 ■

참고하세요

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. User Interface Prototypes와 Technologies

1) User Interface

유저 인터페이스란 무엇일까요? 유저 인터페이스는 사용자가 시스템에 접근할 수 있는 모든 것이라고 할 수 있습니다. 우리는 일반적으로 컴퓨터 만을 생각하지만 그 외에 휴대용 디바이스나 스캐너, **Joystick**, **Keypads**, 음성 인식 시스템 등등 많은 종류들이 있습니다.

User Interface(이하 유저 인터페이스)에 대한 분야가 너무 넓고 깊기 때문에 이 과정에서 모두를 다룰 수는 없습니다. 단지 **GUI(Graphical user interface)**와 **Web UI(Web user interface)**만 다루도록 할 것입니다.

쉽게 예를 들자면 **GUI**는 스윙 컴포넌트로 만들어진 화면 같은 것이고 **Web UI**는 서블릿이나 **JSP**로 만들어진 웹 브라우저를 통해 보여지는 화면을 말하는 것입니다.

왜 아키텍트가 UI에 관심을 가져야 할까요?

I UI 기술은 매우 다양합니다.

유저 인터페이스 기술은 아주 다양합니다. 아키텍트는 요즘 유행하는 **UI** 기술을 살피고 시스템에 적용할 지 여부를 잘 가려야 합니다. 어떤 **UI** 기술을 사용하느냐에 따라 시스템의 구조가 많이 달라지기 때문입니다.

I 사용성(**Usability**)은 시스템의 성패를 좌우합니다.

사용자가 시스템을 얼마나 쉽게 잘 사용하느냐는 시스템의 성패를 좌우할 만큼 중요합니다. 사용하기 어렵고 복잡한 유저 인터페이스는 외면당할 것입니다.

UI는 어떤 역할을 할까요?

I 사용자 **input**과 **action**을 시스템에 전달하여 가동시킵니다.

UI는 시스템이 사용자의 **action**의 의미를 알아 듣고 가동되도록 해야 합니다.

사용자는 입력을 하고, 데이터를 선택하고, 그림을 그림으로써 비즈니스 오브젝트 위의 오퍼레이션을 수행 시킵니다.

이러한 기능을 제공하는 컴포넌트를 **Controller** 컴포넌트라고 합니다.

I 시스템의 상태를 가시적으로 보여줍니다.

UI는 시스템의 상태를 사용자에게 보여줄 수 있어야 합니다. 사용자는 비즈니스 데이터나 문자, 그림, 음성 등으로 비즈니스 오브젝트가 수행한 내용을 보거나 들을 수 있어야 합니다. 이러한 기능을 제공하는 컴포넌트를 **View** 컴포넌트라고 합니다.

2) User Interface Prototypes

UI 디자인은 개발 초기 단계에 그 프로토타입을 이미 구축하고 있는 것이 좋습니다. 그래야만 시스템이 목적했던 바를 좀 더 쉽게 얻을 수 있습니다.

UI 프로토타입을 생성하면 다음과 같은 것을 제공 받을 수 있기 때문입니다.

| 개발 참가자들에게 시스템을 즉시 가시화하여 보여 줄 수 있습니다.

클라이언트측은 시스템을 보고 이해하는 것이 이 시스템에 대한 요구 사항을 얻는데 훨씬 편리합니다. 이러한 요구 사항에는 기능적 요구 사항뿐만 아니라 비기능적 요구 사항까지 포함될 수 있습니다. 초기에 요구 사항을 명확히 하는 것은 개발 위험을 훨씬 줄일 수 있습니다.

| 어떤 유즈케이스를 실행하는 방법을 좀 더 쉽게 분석할 수 있습니다.

위에서 언급한대로 초기에 UI 프로토타입을 보고 시스템에 대한 요구 사항을 명확히 얻어냈다면 어떤 유즈케이스에 대해서 그것이 어떤 방법으로 실행될 것인지를 파악하고 필요한 기술과 구현할 컴포넌트를 분석하는 것이 훨씬 수월해 질 수 있습니다. 즉 정보의 수집과 분석에 아주 도움이 된다는 것입니다.

| 좀 더 나은 사용법을 연구하게 합니다.

비록 클라이언트측에서 요구하지 않은 사항이라 할지라도 개발팀 내에서는 UI 프로토타입을 보고 이 시스템이 좀더 쉽게 사용될 수 있도록 개발 초기 단계에서 연구 할 수 있습니다. 가용성은 시스템의 성패를 좌우할 수 있기 때문입니다.

참고하세요

n UI 프로토타입에 대한 유의 사항

UI 프로토타입이 발표 되었다고 해서 이 소프트웨어 생산물이 다 만들어졌다고 생각하면 큰 오산입니다. UI 프로토타입은 그것이 만들어지기 전에 먼저 모든 개발 참가자들의 의견을 수렴하고 있어야 하지만 발표 후에도 많은 수정 사항이나 추가 사항이 반영될 수 있어야 함을 명심해야 합니다.

3) User Interface Technologies

유저 인터페이스 기술은 아주 다양하다고 했습니다.

우선 가장 기본적인 형태 두 가지만 꼽는다면 아래와 같습니다.

| **Graphical user interface(GUI)**

GUI는 어떤 시스템이든 비쥬얼한 화면으로 많이 사용하고 있는 기술입니다. 일반적으로 GUI는 컴퓨터 마우스 같은 포인팅 디바이스를 사용합니다.

| **Web user interface(Web UI)**

웹 기반 시스템에서는 웹 페이지나 폼을 통해 사용자 입력을 받아 들이거나 정보를 제공하는 기술을 사용합니다.

UI 기술의 다른 형태를 보면 다음과 같은 것이 있습니다.

| **Touchpads**

- | **Direct manipulation**
- | **Joystick**
- | **Interactive voice recognition**
- | **Keypads**
- | **Command line**

이 과정에서는 위의 기술에 대해서는 다루지 않습니다.

어떤 **UI** 기술을 사용하든 일반적으로 **4가지** 유형의 어플리케이션 컴포넌트들이 필요합니다.

| **Controller**

이 컴포넌트는 사용자 **action**을 이해하고 그것을 처리 할 수 있는 비즈니스 오브젝트를 선택하는 역할을 합니다.

| **View**

이 컴포넌트는 시스템의 상태를 사용자에게 표현하는 역할을 합니다.

| **Service**

이것은 사용자 요구를 실제 구현하고 있는 컴포넌트입니다.

| **Entity**

이것은 비즈니스 엔티티(도메인 오브젝트)를 표현하는 컴포넌트입니다.

참고하세요

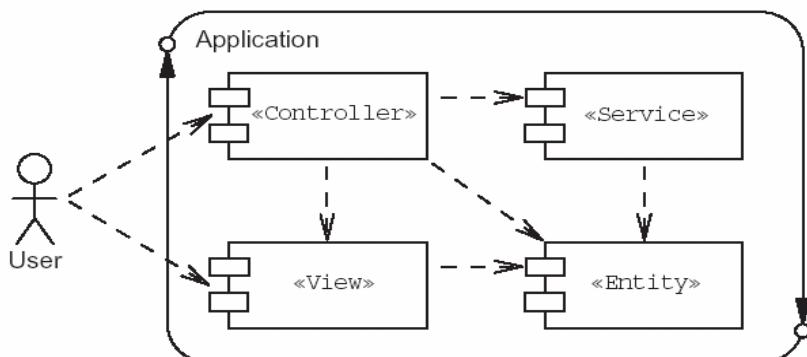
n **비즈니스 엔티티**

비즈니스 엔티티는 데이터를 표현하는 컴포넌트라고 이해하세요.

다음 그림은 일반적으로 **UI** 기술에 사용되는 **4가지** 컴포넌트들이 어떤 관계를 가지는지를 간략히 보여줍니다.

이미지

n **Generic Application Components**



일반적으로 **Controller** 컴포넌트가 가장 앞에서 사용자 액션을 받아들이고 그것을 수

행할 **Service** 컴포넌트에게 전달하면 **Service** 컴포넌트는 수행 중에 **Entity** 컴포넌트와 협력할 수 있습니다. 사용자가 결과를 보고자 하거나 시스템의 상태를 얻고자 할 때는 **View** 컴포넌트가 **Entity** 컴포넌트를 통해 사용자에게 표현해 줍니다.

2. GUI (Graphical User Interface)

자바 기술로 **GUI**를 구현한다면 어떻게 해야 할까요?

먼저 **GUI**의 특징을 제대로 알고 디자인 할 수 있는 능력을 얻은 뒤 적당한 패턴을 적용시키는 것이 좋을 것입니다.

1) GUI의 특징

GUI는 윈도우 기반의 **UI**입니다. 다음과 같은 특징이 있습니다.

- | **시스템은 작고 다양한 사용자 액션을 전부 처리 할 수 있어야 합니다.**
버튼을 누르고, 텍스트 필드에 입력을 하고, 마우스를 움직이는 모든 액션에 대해 시스템이 그 의미를 이해하고 처리해 줄 수 있어야 합니다.
- | **일반적으로 한 화면에서 여러 유즈케이스를 처리 할 수 있습니다.**
한 화면에서 여러 유즈케이스를 처리 할 수 있는 것은 아주 편리합니다.
- | **시스템은 사용자가 무제한적인 action을 할 수 있도록 합니다.**
일반적으로 사용자는 언제든 시스템의 모든 기능을 사용 할 수 있어야 합니다. **GUI**는 사용자가 쉽게 사용할 수 있는 기능을 제공합니다.
- | **한번에 여러 화면을 볼 수도 있습니다.**
간혹 한번에 여러 화면이 뜨기도 합니다. 사용자는 동시에 여러 화면을 보면서 시스템을 가동시킬 수 있습니다.

2) GUI Design

GUI는 보통 프로 디자이너에게 맡겨지지만 소프트웨어 디자이너들이 한다면 다음을 염두에 두어야 합니다.

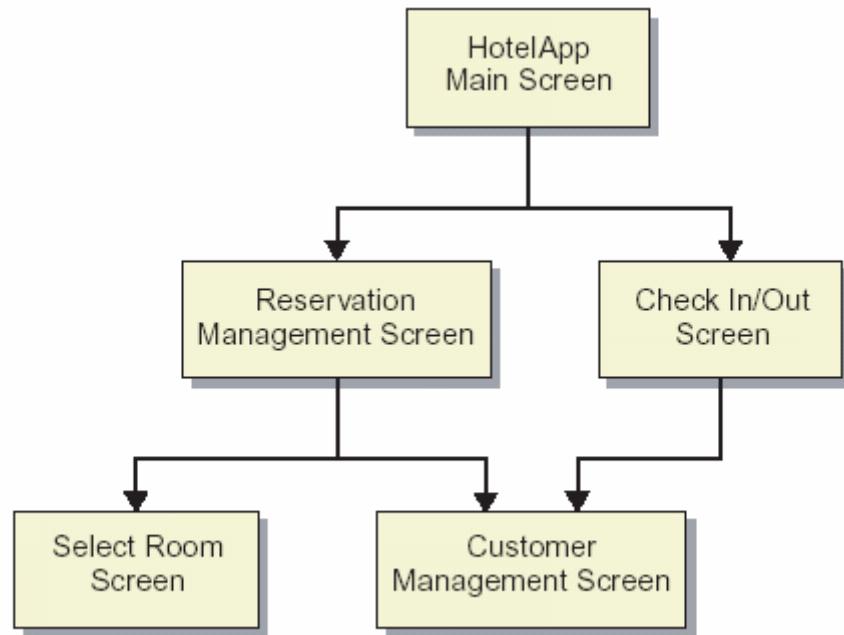
- | **GUI**는 연관된 화면을 계층적으로 설계하는 경향이 있습니다.
스크린 구성 요소로는 **data entry form, information panels, rich content panels, graphical design panels** 등이 있습니다.
- | 각 화면은 **GUI** 컴포넌트의 계층입니다.
GUI 컴포넌트에는 버튼, 텍스트 필드, 리스트 박스, 메뉴 바, 라디오 버튼, 체크 박스 등이 있고 이들은 패널을 통해 그룹화 되어지며 이 패널들은 윈도우 컴포넌트에 놓이게 됩니다.
- | **GUI** 화면은 도메인 모델을 표현하고 사용자 액션을 조절할 수 있습니다.
도메인 모델은 데이터를 표현하는 뷰입니다. 사용자는 어떤 정보(데이터)를 볼 수 있을 뿐만 아니라 **GUI** 컴포넌트들을 통해 **action**을 전달할 수 있습니다.

- | 한 화면으로 여러 유즈케이스를 지원할 수 있습니다.
예를 들어 호텔 예약 시스템의 메인 화면에서 객실을 예약하거나 예약을 수정하거나 취소하는 것이 전부 가능한 것을 말합니다.

다음 그림은 호텔 예약 시스템의 계층적 화면 구성의 예입니다.

[이미지]

n HotelApp Screen Hierarchy



3) PAC Pattern

UI 디자인을 할 때에 특별히 적용 할 수 있는 두 가지 대표적인 패턴이 있습니다.
하나는 **PAC** 패턴이고 다른 하나는 **MVC** 패턴입니다.

먼저 **PAC** 패턴에 대해 알아보겠습니다.

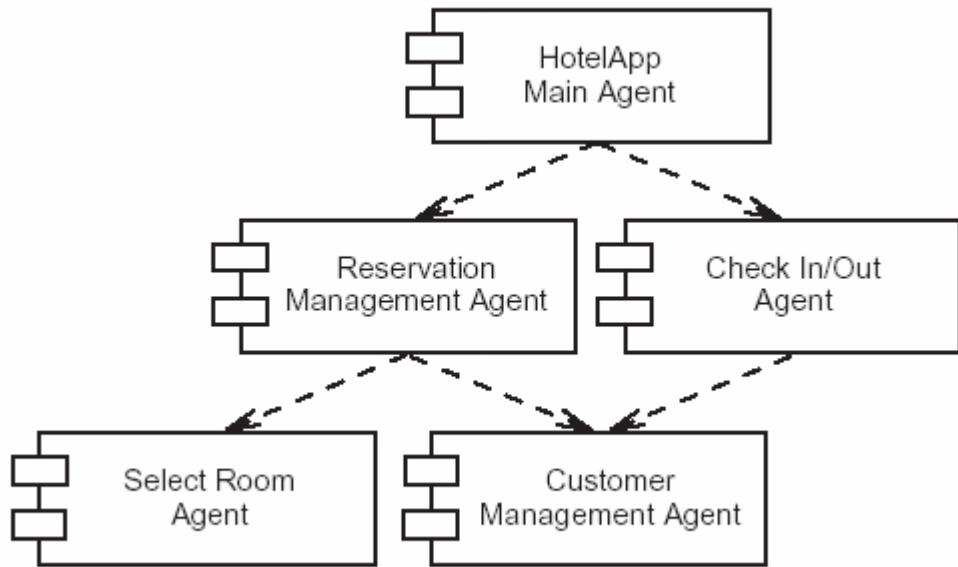
PAC 패턴은 GUI화면의 계층구조를 패턴화한 것입니다.

각 GUI화면은 “**Agent**”라는 컴포넌트로 표현됩니다.

앞 프레임의 호텔 예약 시스템의 계층적 화면구성의 예로 **PAC** 패턴을 적용 시키면 다음 그림과 같이 표현됩니다.

[이미지]

n HotelApp PAC Agents



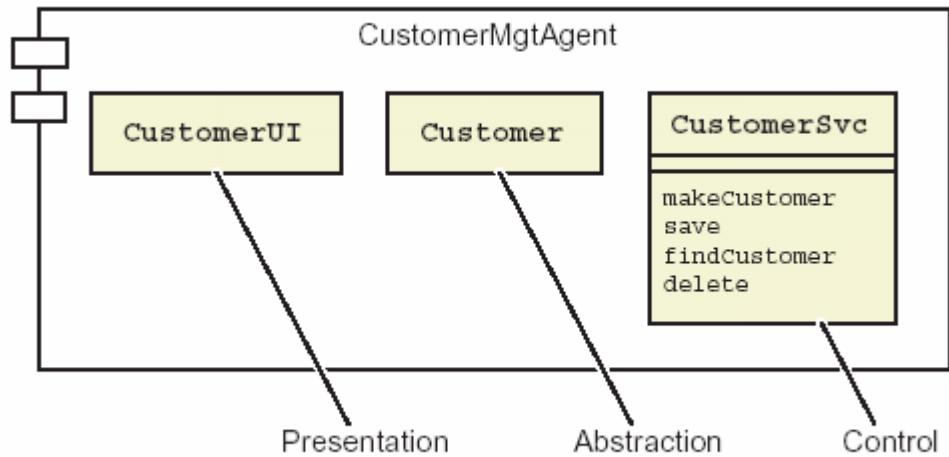
각각의 **PAC agent** 컴포넌트는 다음 세가지 컴포넌트로 구성되어 있습니다.

- | **Presentation** – 사용자에게 제공될 뷰와 사용자 액션을 받아들일 **Control** 컴포넌트를 합해서 **Presentation component**라고 합니다.
- | **Abstraction** – 시스템에서 엔티티를 표현합니다.
- | **Control** – 시스템에서 서비스를 표현합니다.

다음 그림은 **PAC 패턴의 Agent**를 구성하는 세가지 컴포넌트를 보여줍니다.

[이미지](#)

n PAC Agent elements

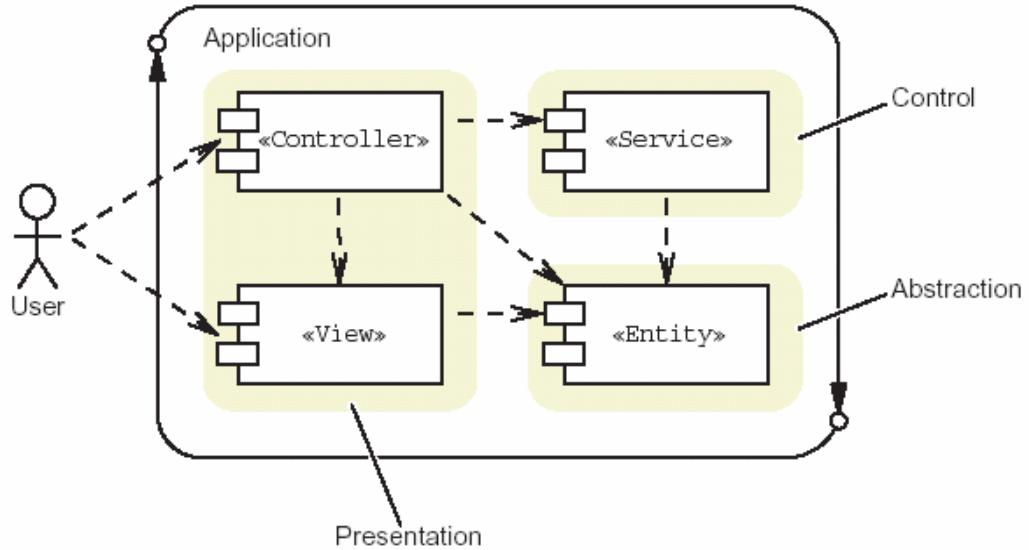


Presentation은 **View**와 **Controller component**를 포함하는 컴포넌트라고 했습니다.

이를 도식화 하면 다음 그림과 같습니다.

[이미지](#)

n PAC Component Types



4) GUI Screen Design

전장에서 **GUI** 화면을 디자인할 때 적용될 수 있는 패턴을 알아봤다면 이제 화면을 얼마나 보기 좋게 구성하느냐도 쟁점화 해 보겠습니다.

먼저, 다음 그림과 같은 고객 관리 화면이 있다면

[이미지]

n Customer Management Screen

Customer Management Screen	
First name:	Jane
Last name:	Gooool
Phone:	303-555-1212
Address	
Street1:	20 Main St.
Street2:	
City:	
State:	
Zip:	
Buttons:	
Search	New
Update	Done

이 화면은 고객의 정보를 입력 할 수 있는 텍스트 필드와 조회, 신규, 수정, 마침의

기능을 제공하는 버튼들로 이루어져 있습니다.

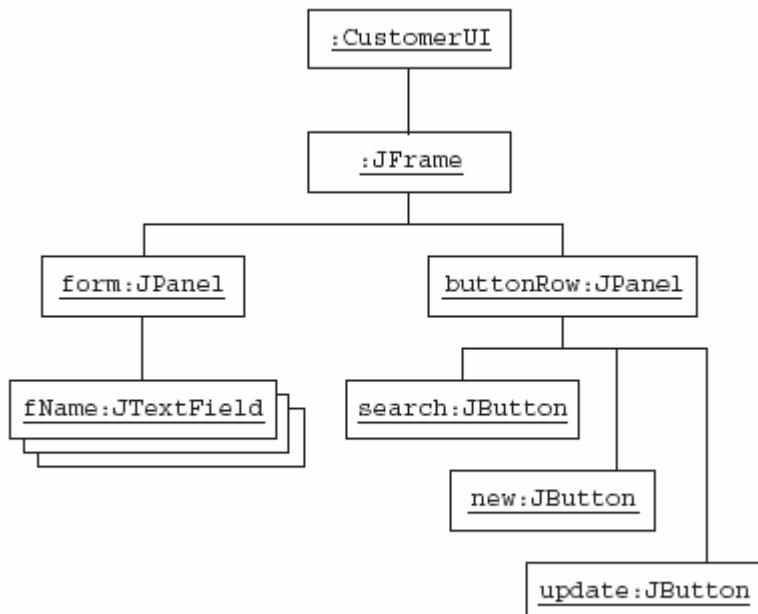
자바 기술에서 이런 **GUI** 구성 요소들은 **AWT** 컴포넌트와 **Swing** 컴포넌트로 제공됩니다.

이런 강력한 **Framework**을 사용하면 화면을 디자인하는 일은 아주 쉬워집니다.

다음 그림은 **Java Swing** 컴포넌트를 사용하여 위의 고객 관리 화면을 구성하기 위한 그림입니다.

이미지

n CustomerUI GUI Component Hierarchy



5) GUI Event Model

GUI는 사용자 액션을 받는 메커니즘을 제공해야 합니다.

예를 들어 버튼을 누르면 시스템이 이 버튼이 눌렸다는 것을 알아챌 수 있어야 한다는 것입니다.

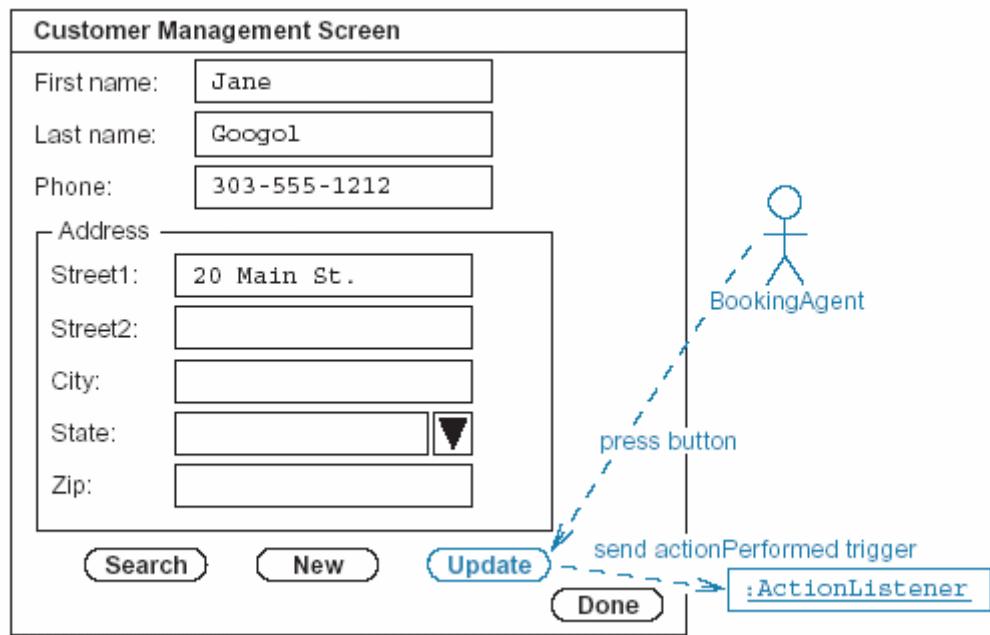
스윙 프레임워크에서는 이 기능이 리스너(**listener**)라는 컴포넌트로써 제공됩니다.

리스너를 구현하고자 한다면, 적당한 **GUI** 컴포넌트와 함께 리스너를 등록해야 합니다.

GUI 컴포넌트에 액션이 발생할 때마다 리스너는 트리거 역할을 하게 됩니다.

다음 그림은 **Update** 버튼을 눌렀을 때 **ActionListener**가 그것을 알아 듣는 메커니즘을 도식화한 것입니다.

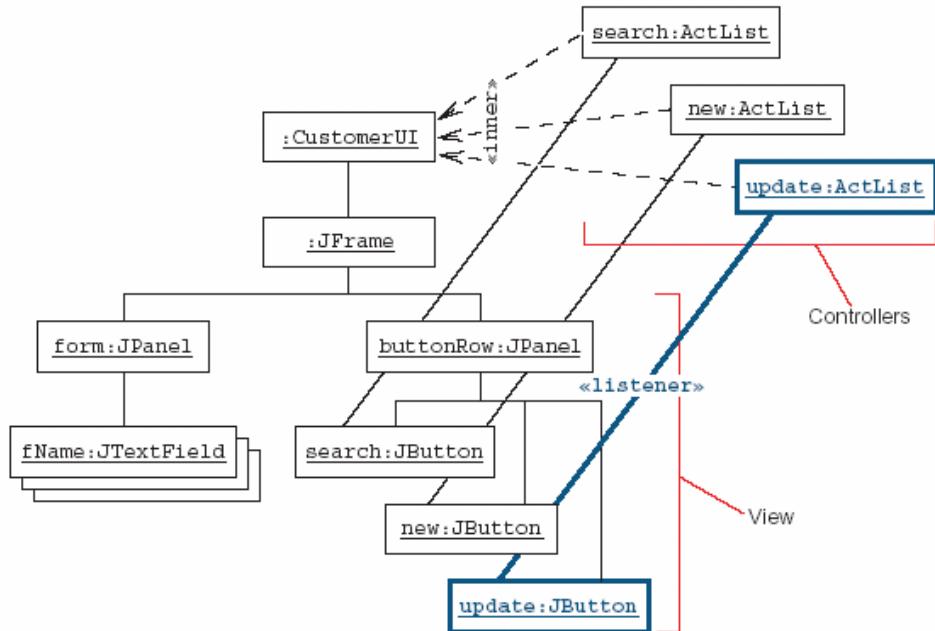
■ Java Technology Event Model



리스너 오브젝트는 **Inner class**를 통해 사용자 액션을 처리하도록 합니다.

만약 버튼이 눌렸을 때 화면의 리스트 내용이 바뀌어야 한다면 이 **Inner class**에서 리스트 컴포넌트를 직접 핸들링 할 수 있습니다.

■ GUI Listeners as Controller Elements



6) MVC Pattern

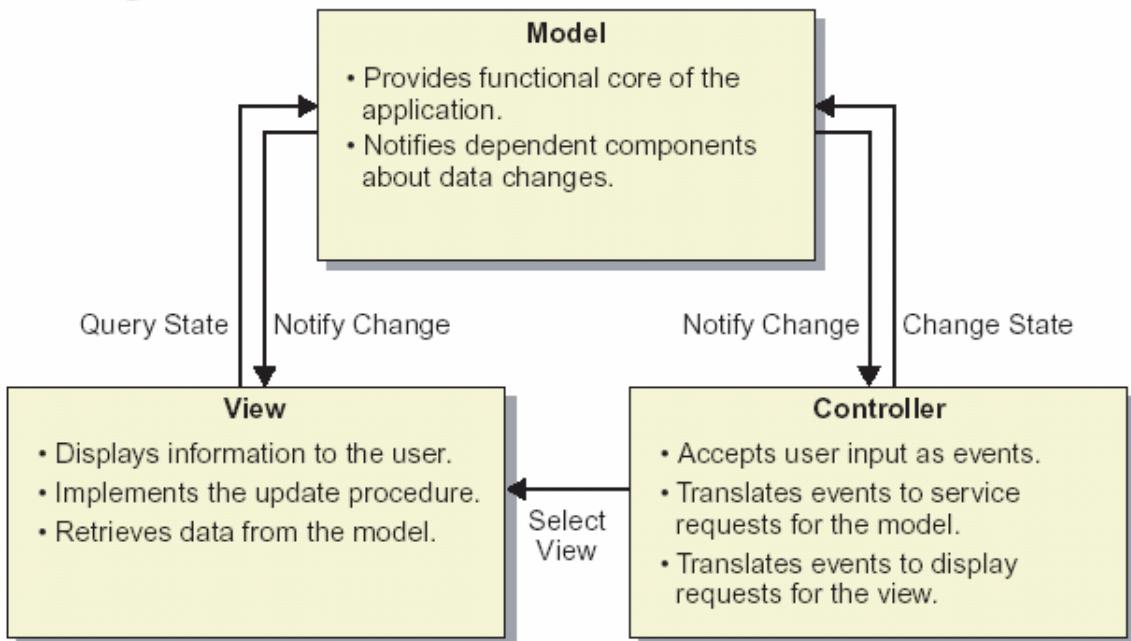
PAC Pattern이외에 **GUI**를 디자인 할 때 적용 할 수 있는 또 하나의 유용한 패턴이 바로 **MVC** 패턴입니다.

이 패턴은 비즈니스 컴포넌트와 엔티티 컴포넌트를 합해서 **Model**이라고 부릅니다.

MVC 패턴은 스윙의 내부 메커니즘과 유사합니다.

[이미지]

n MVC Pattern



View는 사용자에게 정보를 제공합니다.

Controller는 사용자 액션을 받아 들이고 그것을 **Model**에게 전달합니다.

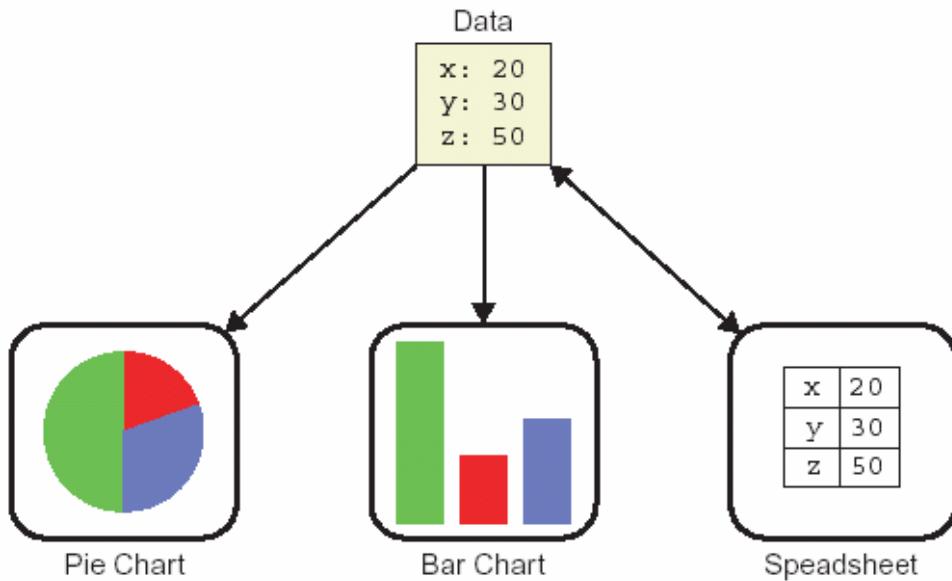
Model은 비즈니스 기능을 수행하고 그 결과를 **View**에게 알립니다.

PAC와 **MVC**의 주요 차이점은 **PAC**는 어플리케이션 안의 모든 **GUI agent**의 최상위 구조를 말하는 것인 반면, **MVC**는 시스템 전체적으로 데이터와 뷰 사이의 분리를 목적으로 하는 구조라는 것입니다. 따라서 **PAC**과 **MVC**는 같은 **GUI**안에서 공존 할 수 있습니다.

MVC 패턴을 사용하면 시스템의 데이터와 뷰를 깔끔하게 분리 할 수 있습니다. 이는 같은 데이터를 가지고도 여러 뷰로 표현 할 수 있음을 의미합니다.

[이미지]

n MVC 패턴의 적용 예



3. Client Tier의 아키텍쳐 모델 생성

사용자 인터페이스를 어떻게 결정하느냐에 따라서 아키텍쳐 모델의 클라이언트 티어의 내용이 달라집니다. **GUI**를 선택 했을 때 클라이언트 티어의 아키텍쳐 모델을 생성 하기 위한 과정은 다음과 같습니다.

- | 클라이언트 티어의 컴포넌트를 가진 상세 배치 다이어그램을 그립니다.
- | 상세 배치 다이어그램을 통해 아키텍쳐 템플릿을 얻습니다.
- | 클라이언트 티어 기술과 컴포넌트를 표현하는 **Tier & Layer Package Diagram**을 작성합니다.

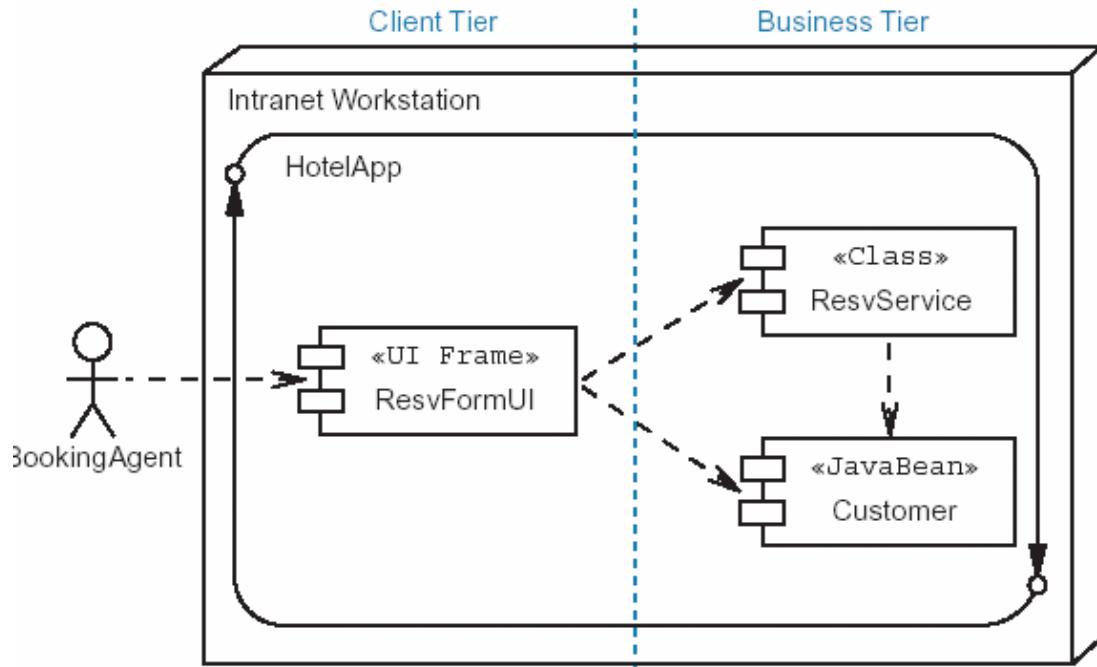
1) 상세 배치 다이어그램의 참조

호텔 예약 시스템은 스윙 프레임워크를 사용하는 **GUI** 기술을 선택했습니다.

이것을 상세 배치 다이어그램으로 그리면 다음과 같습니다.

[이미지](#)

HotelApp Detailed Deployment Diagram



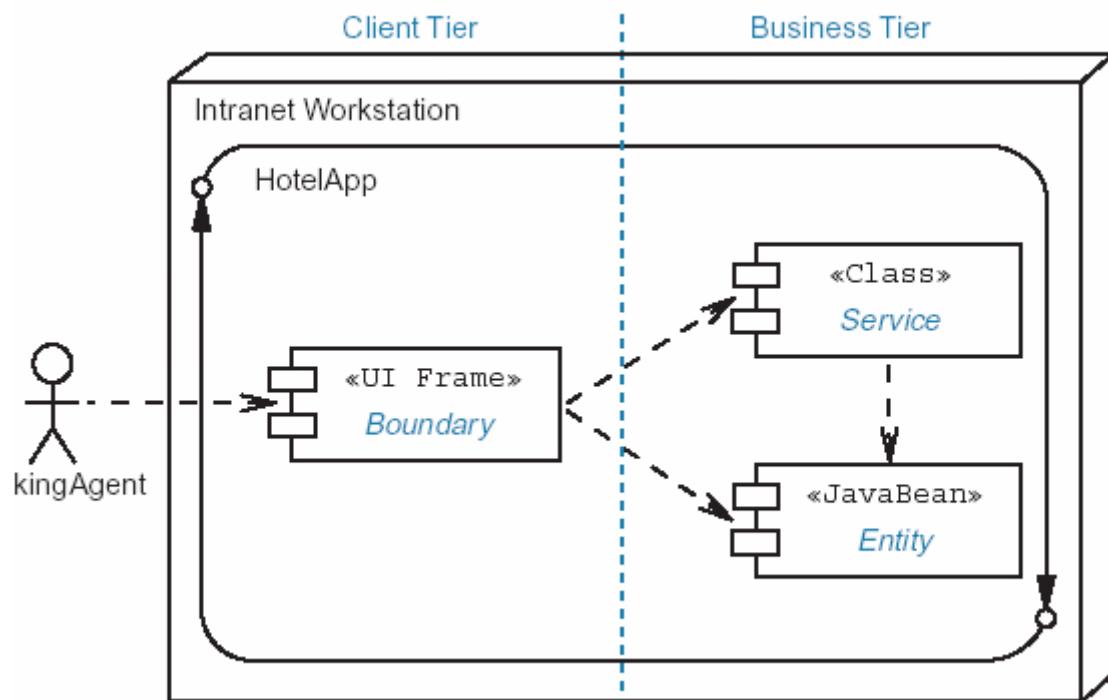
2) 아키텍쳐 템플릿 생성

상세 배치 디어그램에서 **Client Tier**의 **<<UI Frame>>**은 디자인 컴포넌트 유형 중에 바운더리 컴포넌트입니다.

따라서 상세 배치 디어그램을 기초로 한 아키텍쳐 템플릿을 생성하면 다음과 같습니다.

[이미지]

n Architecture Template



바운더리 컴포넌트가 PAC 패턴에서 **Presentation**에 해당하는 **UI Frame**과 매핑되는 것을 볼 수 있습니다. **Presentation** 컴포넌트는 UI의 뷰(View)와 컨트롤러(Controller)를 포함하는 컴포넌트입니다.

3) Tier and Layer Package Diagram 작성

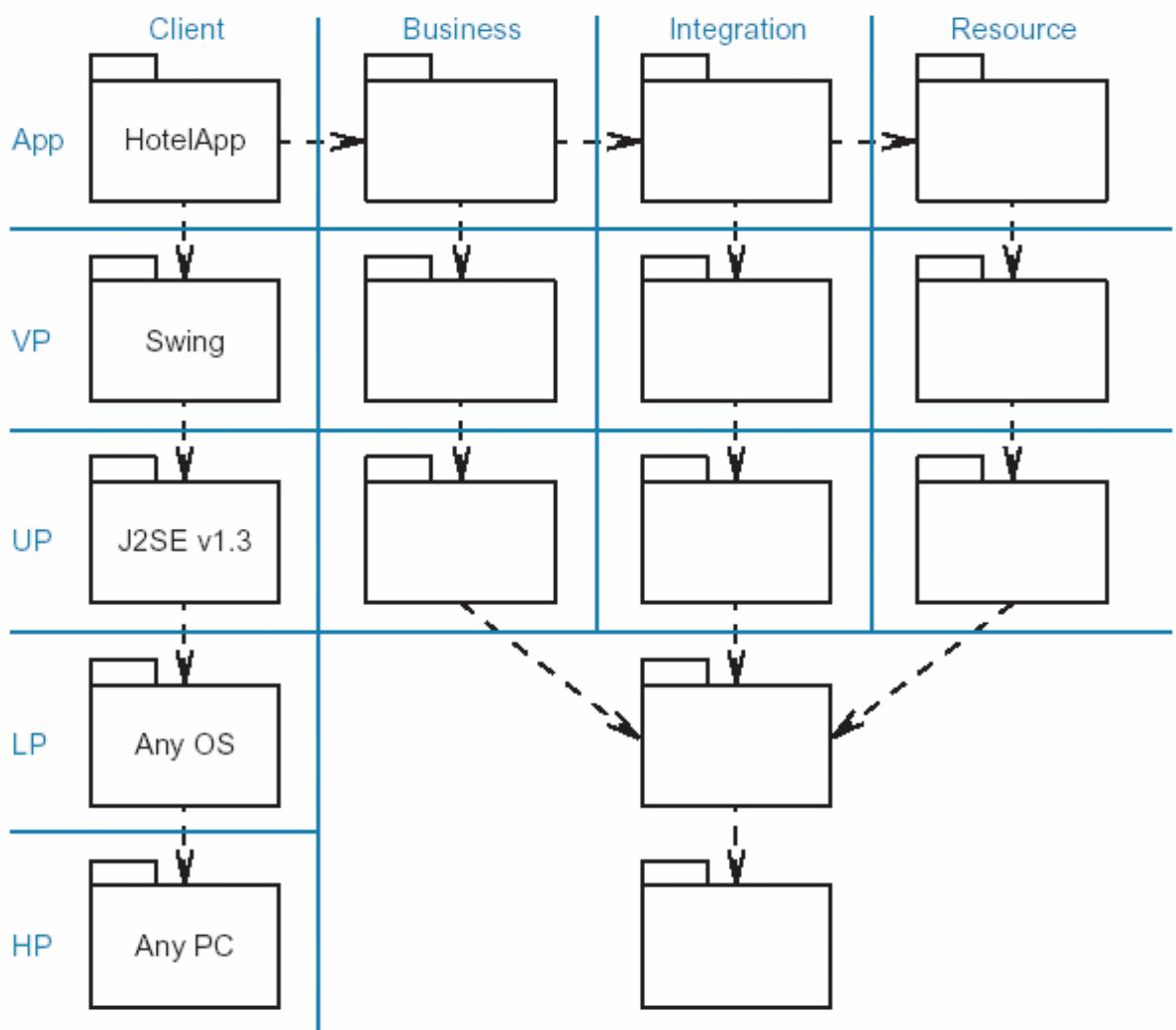
이 단계에서 아키텍처는 오로지 클라이언트 티어의 컴포넌트들만 결정 지을 수 있었습니다.

따라서 아키텍쳐 모델 중 하나인 **Tier & Layer Package Diagram**을 작성할 때 오직 클라이언트 티어만 부분적으로 작성 할 수 있게 됩니다.

다음 그림은 클라이언트에서 사용되는 어떤 하드웨어에서든 또 그들이 어떤 운영체제를 갖든 **J2SE**를 사용하면 가능하다는 것을 표현하며 스윙 프레임워크를 사용하여 호텔 어플리케이션의 유저 인터페이스를 구성하였다는 것을 보여줍니다.

[이미지]

n 부분적인 HotelApp Tier & Layer Package Diagram



참고하세요

자바는 어떤 하드웨어나 어떤 운영체제와도 무관한, 플랫폼 독립적인 프로그램 언어입니다.

4. Web UI (Web User Interface)

1) Web UI 특징

Web UI는 브라우저 기반의 유저 인터페이스입니다. **Web UI**에서 클라이언트 티어는 브라우저를 포함하고 있습니다. 브라우저는 **HTML**페이지를 보여줍니다. **HTML**페이지는 다른 페이지에 연결되어 있기도 하고 폼을 통해서 사용자 정보를 입력 받을 수도 있습니다. 이러한 정보는 **HTTP Request**로 웹 서버에게 전달되고 프리젠테이션 티어에서 이를 처리해 줍니다. 때문에 프리젠테이션 티어를 **웹 티어**라고 부르기도 합니다.

Web UI는 다음과 같은 특징이 있습니다.

| **Web UI**는 몇몇 큰 사용자 액션을 수행합니다.

Web UI를 통해 일어나는 사용자 액션은 **Http Request**로 웹 서버에게 전달됩니다. 이것을 스케일이 큰 사용자 액션이라고 합니다.

비교하여, 스케일이 작은 사용자 액션은 **JavaScript, Flash, Java applet**등과 같이 클라이언트 웹 브라우저에서 사용자 액션이 처리되는 것을 말합니다.

| 하나의 유즈케이스가 여러 화면에 나뉘어 있습니다.

웹 쇼핑몰을 예로 든다면 어떤 상품을 구매하기 위해서 여러분들은 여러 화면을 순차적으로 보게 될 것입니다. 이것은 ‘구매’ 유즈케이스를 진행하는 동안 필요한 정보를 입력 받거나 정보를 보여주는 화면을 순차적으로 배치해서 사용자에게 제공하기 때문입니다.

이것을 웹 페이지의 ‘**sequence**’라고 합니다.

| 때때로 어떤 웹 페이지로의 이동이 한번에 이루어 질 수도 있습니다.

Web UI의 화면이 항상 순차적으로만 보여지는 것은 아닙니다.

사용자는 보고 있던 화면에서 다른 화면으로의 전환이 쉽습니다.

| 오직 한번에 한 화면만이 열립니다.

일반적으로 웹 브라우저는 한번에 한 화면만 보여줍니다.

때문에 프레임 같은, 화면을 여러 개로 분산하는 기술을 사용하기도 합니다.

그러나 이것은 사용자에게 혼동을 가져다 줄 수도 있습니다.

2) Web UI Screen Design

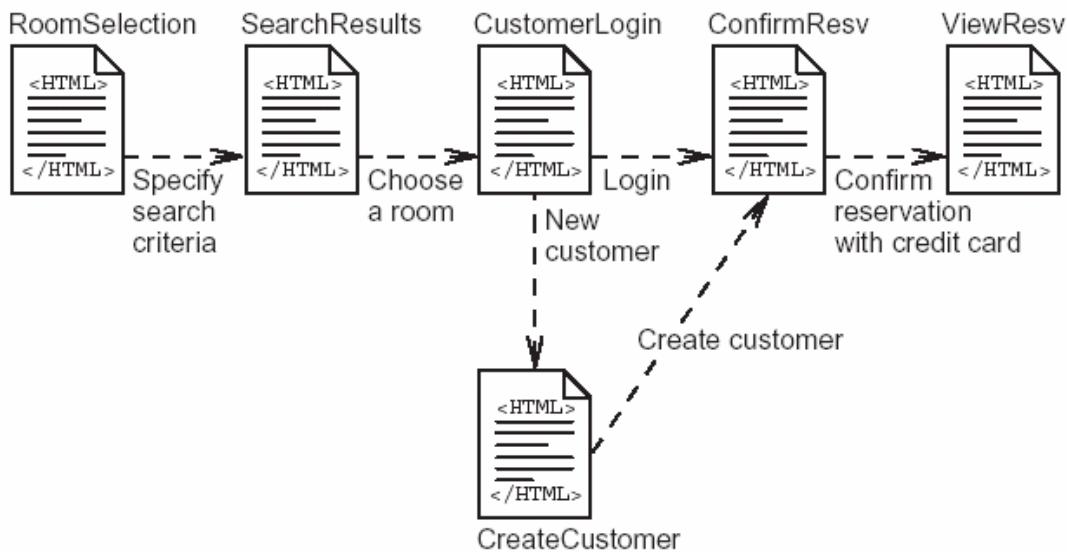
Web UI 화면을 디자인 할 때는 다음과 같은 사항에 유의하여야 합니다.

- | **Web UI** 는 연관된 화면이 순차적으로 구성되는 경향이 있습니다.
Web UI 페이지는 하나의 유즈케이스의 워크플로우를 순차적으로 보여줍니다.
- | 각 화면은 **UI** 컴포넌트의 계층입니다.
각 **UI** 페이지에는 많은 정보가 담겨있습니다. 이 페이지를 구성하고 있는 **UI** 컴포넌트에는 텍스트 필드나 버튼, 리스트 등 많은 종류가 있고 이들은 **GUI** 컴포넌트들처럼 계층적인 관계를 가질 수 있습니다.
- | **Web UI** 화면은 도메인 모델의 사용자 뷰를 표현합니다.
도메인 모델은 시스템의 정보를 의미합니다. 즉, 사용자는 화면을 통해서 자신의 정보를 입력할 수도 있지만 시스템의 정보도 볼 수 있습니다.
- | 간혹 어떤 화면은 여러 유즈케이스에 재사용될 수도 있습니다.
보통 **Web UI** 화면은 한 유즈케이스에 특정적으로 디자인 되지만 어떤 화면은 여러 유즈케이스에서 공통적으로 사용되기도 합니다.

다음은 호텔 예약 시스템의 웹 페이지 흐름을 보여 주는 예입니다.

[이미지]

■ Example Web Page Flow



Web UI 페이지는 **HTML form**을 제공합니다. 이것은 다양한 **UI** 컴포넌트들을 태그를 통해 사용할 수 있게 합니다.

다음은 폼 태그를 통해 여러 **UI** 컴포넌트를 사용하는 예입니다.

소스보기

n Partial Web UI form example

```
<FORM ACTION='makeResv' METHOD='POST'>
<INPUT TYPE='hidden' NAME='action' VALUE='roomSearch'>
Enter arrival date: <INPUT TYPE='text' NAME='arrivalDate'>
<BR>
Enter departure date: <INPUT TYPE='text' NAME='departureDate'>
<BR>
Select room type:
<SELECT NAME='roomType'>
  <OPTION VALUE='Single'> Single
  <OPTION VALUE='Double'> Double
  <OPTION VALUE='Suite'> Suite
</SELECT>
<BR>
<INPUT TYPE='submit' VALUE='Search... '>
</FORM>
```

3) Web UI Event Model

Web UI 어플리케이션은 **GUI** 어플리케이션과는 근본적으로 다른 이벤트 모델을 갖습니다.

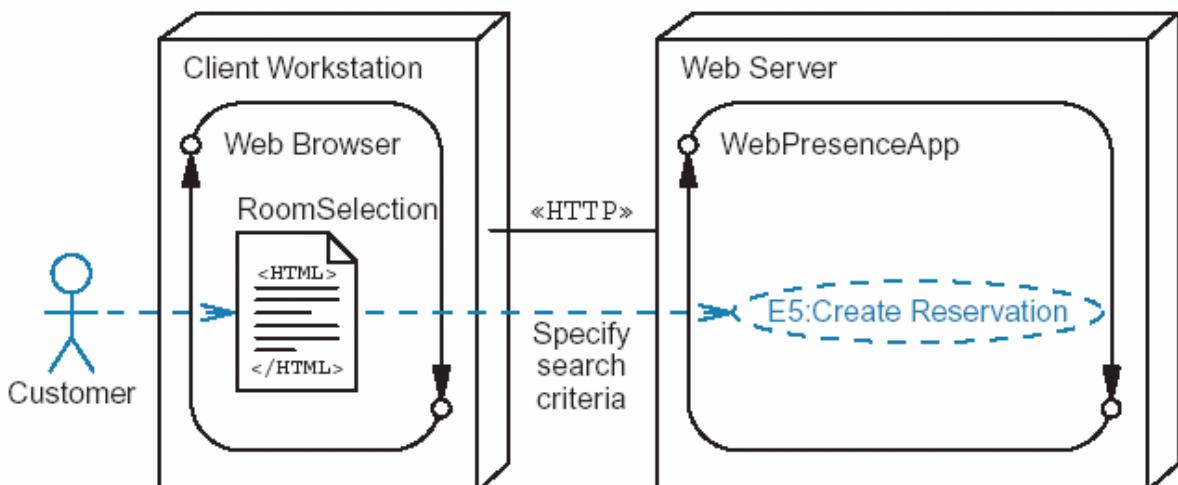
이벤트에는 두 가지 유형이 있습니다.

- | **JavaScript**로 처리되는 소형 이벤트
- | 웹 브라우저로부터 웹서버로 **Http Request**가 전달되는 대형 이벤트

다음 그림은 고객이 호텔 예약 시스템으로 객실을 선택하는 Web UI 이벤트 모델을 도식화한 것입니다.

이미지

n Web UI Event Model



사용자가 객실을 선택하면 이 정보는 **Http Request** 형태로 웹 서버에게로 전달되어집니다. 웹 서버 위에는 **Http Request**를 받아 들이고 분석한 뒤 새 예약을 만드는 로직이 실행됩니다. 이렇게 **HTTP**를 사용하여 사용자 이벤트가 처리되는 것을 대형 이벤트, 혹은 마크로 이벤트라고 합니다.

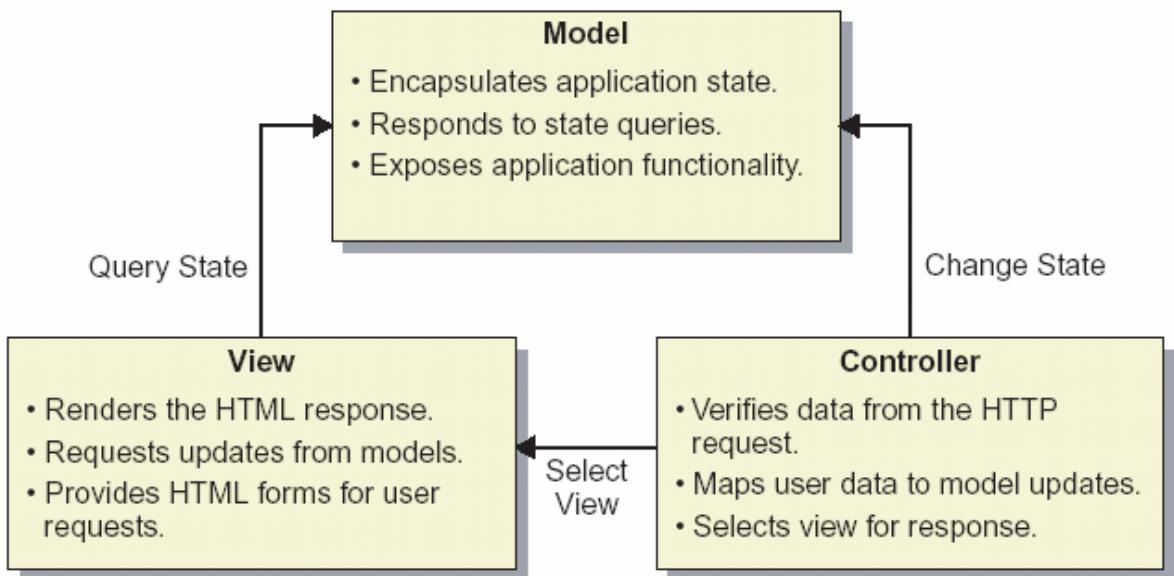
4) WebMVC Pattern

GUI 디자인에 **MVC** 패턴을 적용할 수 있었던 것처럼 **Web UI**에도 **MVC** 패턴을 적용할 수 있습니다.

다음 그림은 **Web UI**에 **MVC** 패턴을 적용할 때의 각 요소들의 내용입니다.

[이미지]

■ WebMVC Pattern



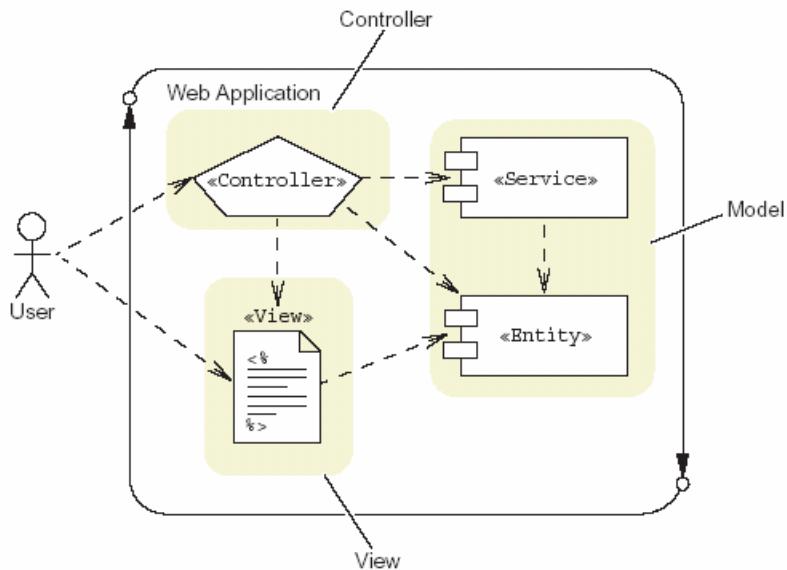
GUI MVC 패턴과 **Web UI MVC** 패턴의 주요 차이는 모델이 변경될 때 **GUI**의 뷰는 자동으로 변경이 가능하지만 **Web UI**의 뷰는 자동 변경이 불가능 하다는 것입니다.

그러나 이러한 제약이 있음에도 불구하고 **UI** 디자인에 **MVC** 패턴을 적용하는 것은 매우 유용합니다. 왜냐하면 프리젠테이션 부분과 비즈니스 로직 부분의 분리가 명확하기 때문에 어느 한 쪽 요소를 변경하거나 재사용하기가 수월하기 때문입니다.

다음 그림은 **WebMVC** 패턴을 컴포넌트 타입을 통해 표현한 그림입니다.

[이미지]

■ WebMVC Pattern Component Types



보충

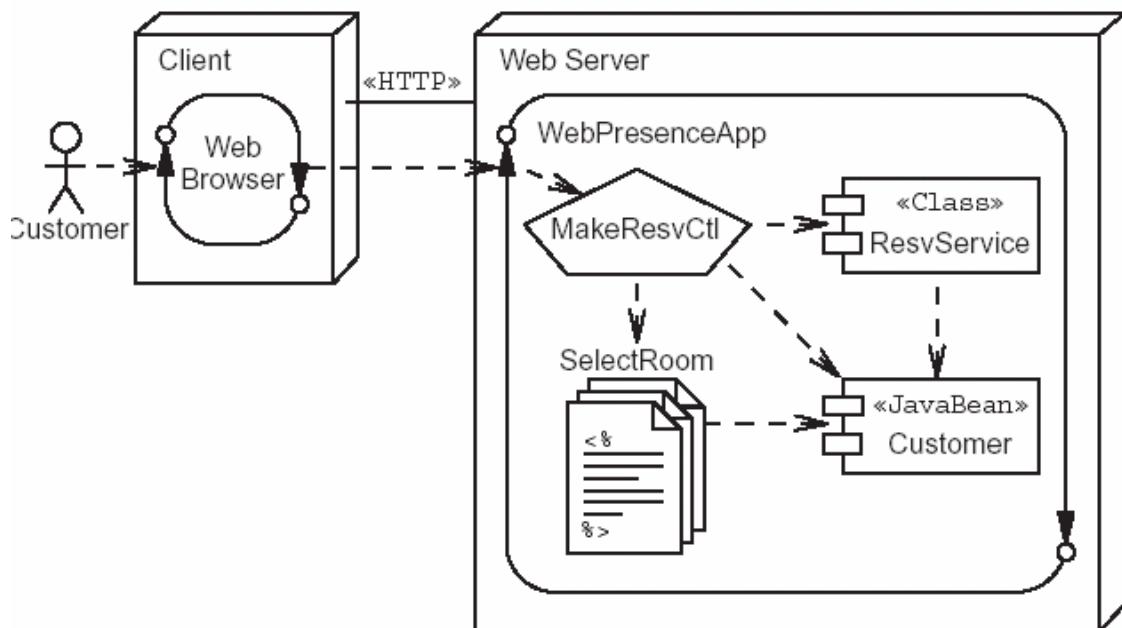
■ 그림 이해하기

위 그림(**WebMVC Pattern Component Types**)에서 ‘오각형’ 그림은 서블릿 컴포넌트를 의미하고 ‘Page’ 그림은 **JSP** 컴포넌트를 의미합니다. 컴포넌트 다이어그램은 다양하게 표현될 수 있습니다.

다음 그림은 자바 기술을 사용하여 웹 어플리케이션을 구축 했을 때 **MVC 패턴**을 따른 구조를 보여줍니다.

이미지

■ An Example Java Technology Web Application



Model 2 Architecture

자바 기술을 사용한 웹 어플리케이션 구축 시 **MVC** 패턴을 적용한 것을 **Model 2 Architecture**라고 합니다. 각 컴포넌트는 다음과 같은 역할을 합니다.

- | 서블릿은 **Controller** 컴포넌트의 역할을 합니다.
- | 자바 클래스나 **JavaBean**은 **Model** 컴포넌트의 역할을 합니다.
- | JSP는 **View** 컴포넌트의 역할을 합니다.

더 자세한 내용은

http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html에서 참고하세요.

5. Presentation Tier의 아키텍쳐 모델 생성

유저 인터페이스로 **GUI**를 선택했을 때와 마찬 가지로 **Web UI** 시에도 다음과 같은 과정을 통해서 아키텍쳐 모델을 생성합니다.

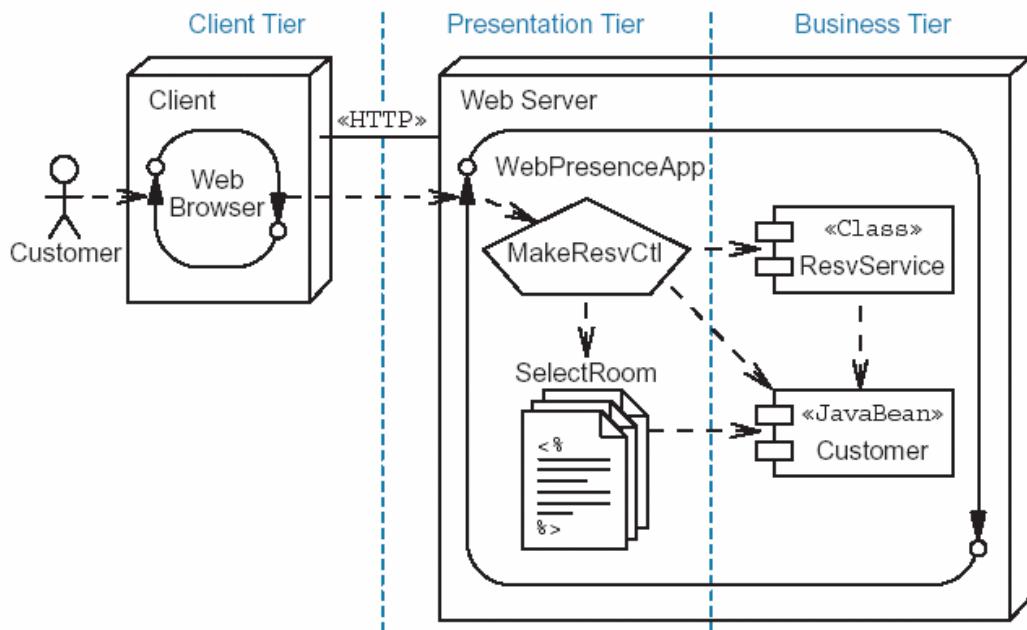
- | 프리젠테이션 티어의 컴포넌트를 가진 상세 배치 다이어그램을 그립니다.
- | 상세 배치 다이어그램을 통해 아키텍쳐 템플릿을 얻습니다.
- | 프리젠테이션 티어 기술과 컴포넌트를 표현하는 **Tier & Layer Package Diagram**을 작성합니다.

1) 상세 배치 다이어그램의 참조

자바 기술로 호텔 예약 시스템을 개발 한다면 다음과 같은 상세 배치 다이어그램을 얻을 수 있습니다.

이미지

HotelApp 상세 배치 다이어그램



클라이언트는 웹 브라우저를 통해서 **HTML** 페이지를 볼 수 있고 어떤 유즈케이스를 요청하면 이것은 웹 서버의 프리젠테이션 티어의 서블릿 컴포넌트에 전달되어 이 유즈케이스를 실행해줄 비즈니스 컴포넌트로 다시 전달됩니다. 또한 클라이언트는 **JSP** 페이지를 통해서 동적으로 만들어지는 **HTML** 페이지를 통해 이 시스템의 정보를 볼 수 있습니다.

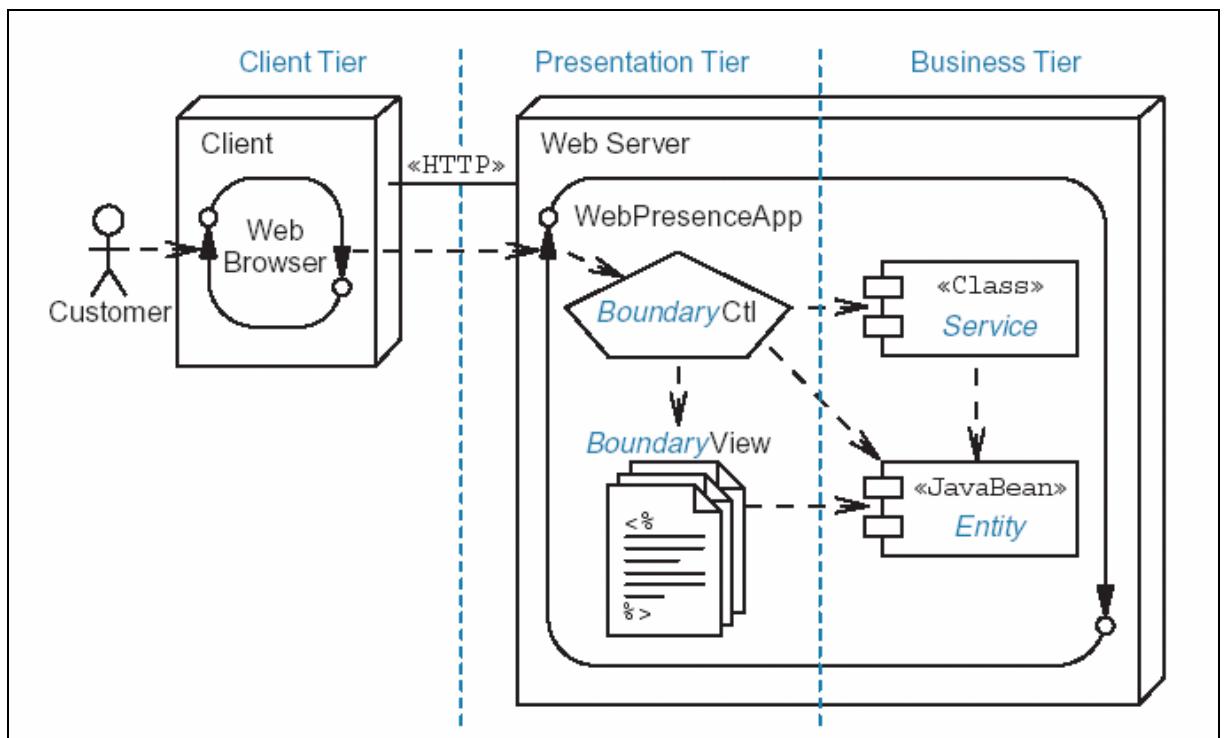
2) 아키텍쳐 템플릿 생성

상세 배치 다이어그램으로부터 아키텍쳐 템플릿을 생성할 수 있습니다.

디자인 컴포넌트들은 이름 표기 대신 유형이 표기되어야 합니다.

[이미지]

■ HRS(Hotel Reservation System)의 아키텍쳐 템플릿

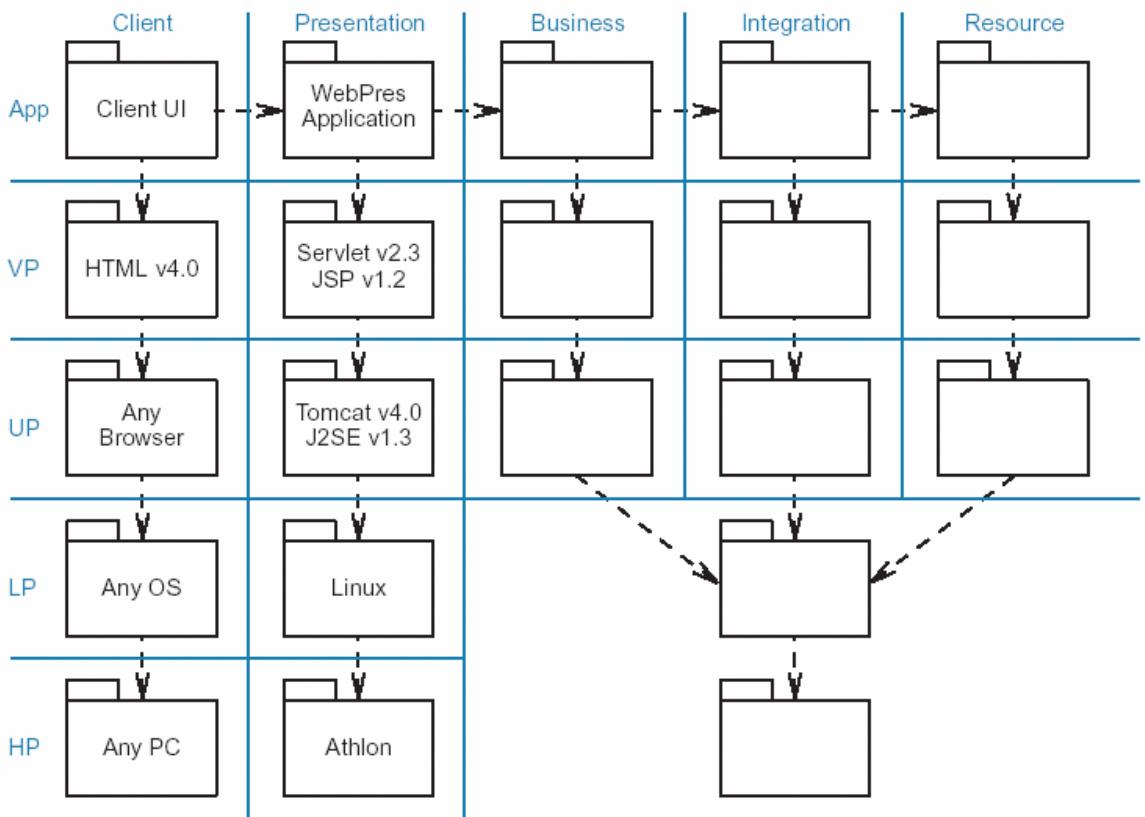


3) Tier and Layer package Diagram 작성

Web UI를 선택했을 때의 **Client**와 **Presentation Tier**의 기술과 컴포넌트를 표현하는 **Tier & Layer Package Diagram**은 다음과 같습니다.

[이미지]

■ A partial Tiers/Layers Package Diagram



과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원5 : 비기능적 요구사항 구축

모듈 4 : Business Tier 의 아키텍쳐 모델 생성

담당강사 : 전은수

■ 생각해봅시다 ■

아키텍쳐 모델의 생성 시 비즈니스 티어의 기술과 컴포넌트를 표현하기 위해서는 어떻게 해야 할까요? 비즈니스 티어란 유즈케이스를 실행하는 로직을 가지고 있는 티어를 말합니다. 클라이언트 티어나 프리젠테이션 티어를 통해 사용자 액션이 들어 오면 이는 비즈니스 티어로 전달 되고 해당 컴포넌트가 실행되게 됩니다. 그렇다면 클라이언트 티어나 프리젠테이션 티어에서의 사용자 액션은 비즈니스 티어로 어떻게 전달 될까요? 이들은 서로 떨어져 있는 시스템입니다. 분산 시스템에서 커뮤니케이션 할 수 있는 기술을 알고 있습니까?

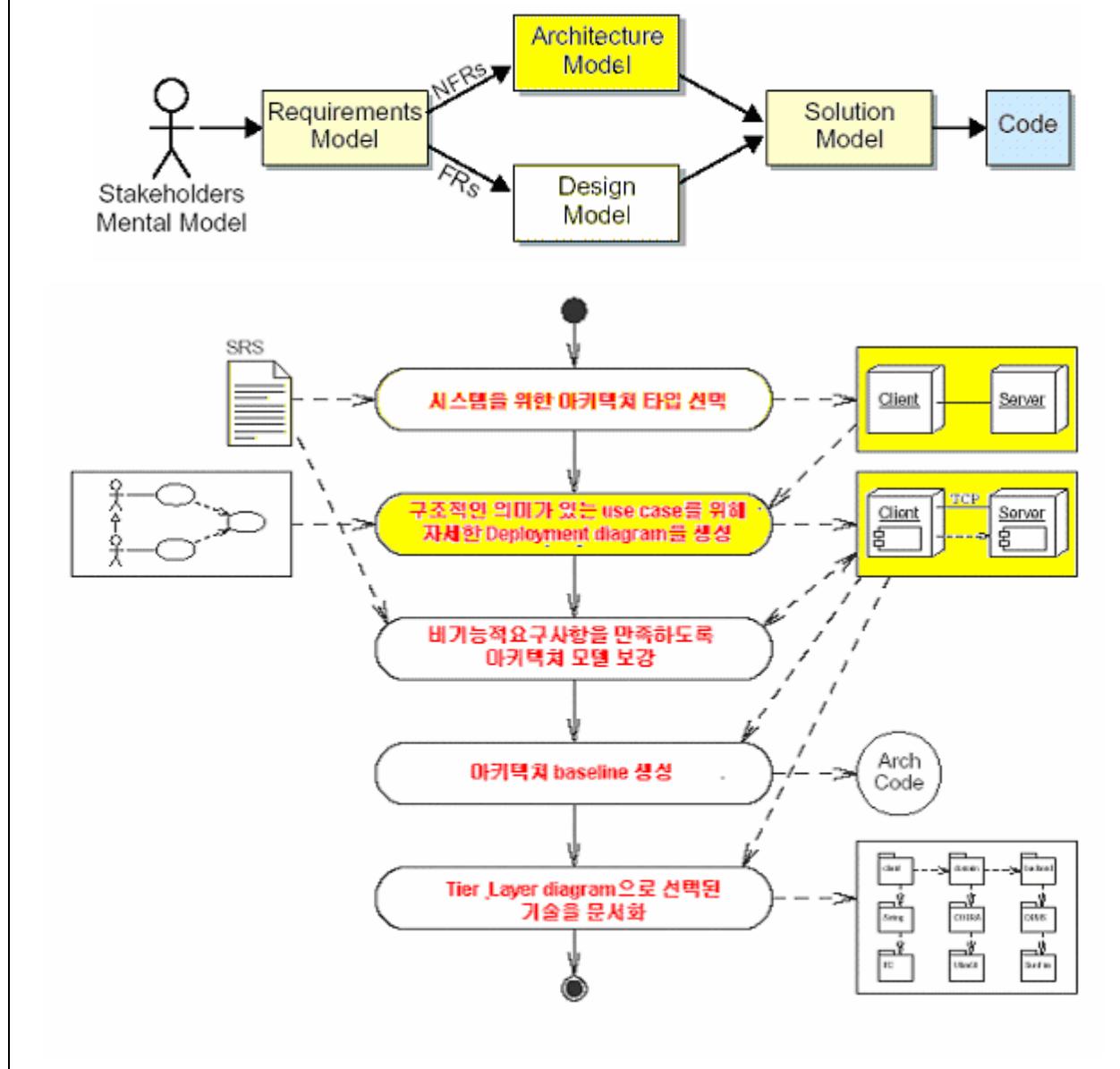
애니메이션



학습하기

참고하세요

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. 분산 객체 지향 컴퓨팅의 이해

분산 컴퓨팅이란 다른 사용자 또는 컴퓨터에게 정보를 공유 시키게 할 수 있는 구조를 말합니다. 분산 컴퓨팅은 하나의 컴퓨터에 있는 어플리케이션으로 하여금 다른 기계의 프로세싱 파워, 메모리 또는 저장장치를 힘을 이용할 수 있게 해줍니다. 이것은 분산 컴퓨팅이 **stand-alone** 어플리케이션의 성능을 향상 시킬 수 있기 때문에 가능합니다. 그러나 이것이 어플리케이션을 분산 하는 이유는 아닙니다. 워드프로세싱 같은 어플리케이션은 분산 컴퓨팅의 이점을 누리질 못할 것입니다. 많은 경우에, 특별한 문제가 분산을 요구합니다. 만약 회사가 많은 곳에 있는 정보를 얻고자 할 경우, 분산이 적당한 기능을 할 것입니다. 다른 경우에, 분산은 성능이나 사용가능성을 향상시킵니다. 만약 어플리케이션이 **PC**에서 수행되고, 어플리케이션이 대단한 길이의 계산을 수행할 필요가 있을 때, 이 계산을 분산시키면 더 빨리 그것을 계산 할 수 있도록 해줍니다.

호텔 예약 시스템의 경우, 고객은 집에서 인터넷을 통한 시스템 접근을 원할 것이고 호텔 직원은 전화 예약을 원하는 고객을 위해 사무실에서 시스템 접근을 원할 것입니다. 이런 사용자 액션을 받아 들이기 위해 유저 인터페이스를 구축하는 기술과 컴포넌트를 알아보았습니다. 그렇다면 사용자 액션을 받아 들인 클라이언트나 프리젠테이션 티어에서는 유즈케이스의 서비스를 직접하게 될까요? 그렇지 않습니다. **인터넷과 인트라넷 요청을 모두 일괄 처리하기 위해서는 분산 컴퓨팅이 절실히 요구됩니다.** 바로 이것이 서비스 컴포넌트를 위한 시스템을 구성해야만 하는 이유입니다.

분산 컴퓨팅 기술은 많은 발전을 거듭했습니다.

특별히 우리는 객체지향 분산 컴퓨팅 기술에 대해서 살펴 보도록 하겠습니다.

1) 객체지향 분산 컴퓨팅 기술

| CORBA

코바(CORBA : Common Object Request Broker Architecture)는 OMG (Object Management Group)의 명세입니다. 이것은 각종 컴퓨터와 각종 랭귀지간에 커뮤니케이션이 가능하게 하는 광범위하고 복잡한 기술입니다. 코바 명세를 구현한 많은 벤더들이 있습니다. 코바는 IIOP (Internet Inter-ORB Protocol)이라고 하는 통신 프로토콜을 사용합니다.

| RMI

RMI (Remote Method Invocation)은 썬 마이크로시스템즈 사의 기술입니다. 이것은 코바처럼 객체지향 분산 컴퓨팅 기술이지만 **100% 자바-자바간 커뮤니케이션만**을 지원합니다. 때문에 코바보다 훨씬 쉽고 간결한 사용법을 가지고 있습니다. RMI는 JRMP(Java Remote Method Protocol)이라는 통신 프로

토콜을 사용합니다.

I EJB technology

EJB 기술도 썬 마이크로시스템즈 사의 명세입니다. 이것은 엔터프라이즈 시스템을 위한 광범위하고 복잡한 기술입니다. EJB 명세를 구현한 많은 벤더가 있습니다. EJB는 자바 어플리케이션 간의 상호작용이지만 CORBA도 지원합니다. EJB는 RMI 프로토콜과 CORBA 프로토콜을 합한 RMI-IIOP 를 사용합니다.

I Web services (SOAP)

SOAP (Simple Object Access Protocol)은 W3C의 명세입니다. SOAP은 원격 객체 메카니즘이라기 보다 원격 프로시저 메커니즘에 더 가깝습니다.

2) 서비스 컴포넌트에 로컬 엑세스를 하는 경우

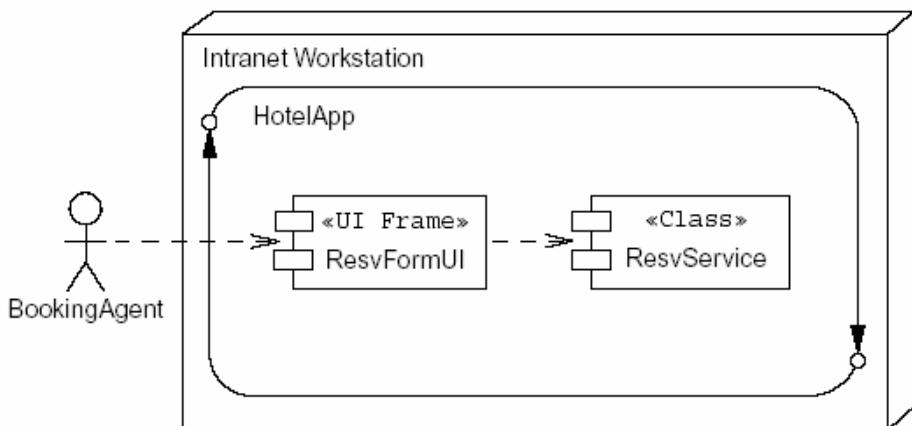
클라이언트 어플리케이션은 어떤 객체에 의해 제공되는 서비스를 받을 필요가 있습니다.

만약 그 서비스가 클라이언트 어플리케이션 시스템내에 존재한다면 이를 로컬 엑세스라고 합니다.

다음 그림은 호텔 예약 시스템의 인트라넷 웍스테이션에서 예약 담당직원이 예약화면을 통해 예약 서비스를 요청할 때 이를 처리해 주는 서비스 컴포넌트가 같은 시스템 내에 있는 경우를 도식화 한 것입니다.

[이미지]

n Local access to a Service Component

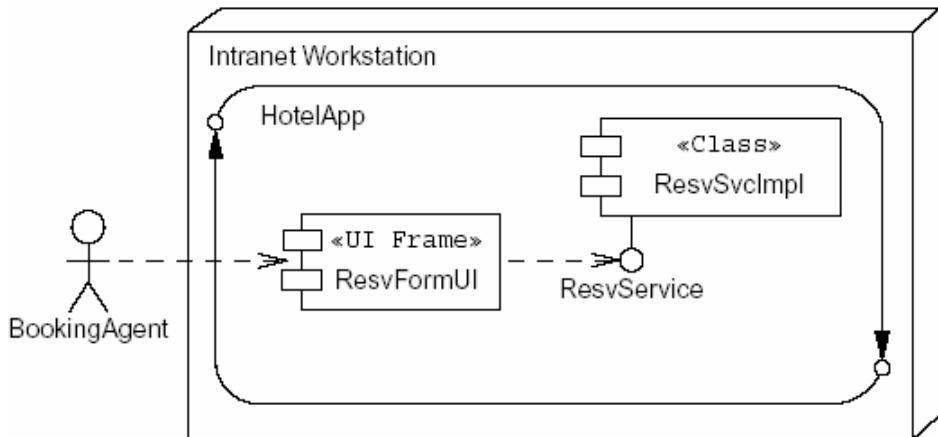


이런 구조의 경우 클라이언트 코드가 서비스 클래스의 메소드를 직접 호출하게 되므로 너무 유연성이 없는 구조를 갖게 됩니다. 만약 서비스 클래스의 내용이 변경된다면 클라이언트의 코드도 수정되어야 되는 번거로움이 있습니다.

다음 그림은 또 다른 로컬 엑세스를 보여줍니다.

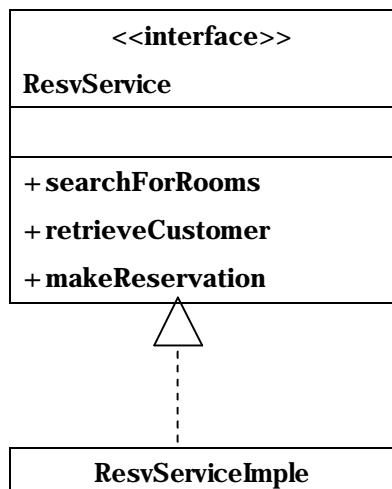
[[이미지](#)]

n Applying the Dependency Inversion Principle



이번에는 클라이언트 컴포넌트가 구현된 서비스 컴포넌트에 직접 엑세스 하는 것이 아니라 서비스 인터페이스를 통하여 간접적으로 엑세스 하는 것을 볼 수 있습니다.
이것이 훨씬 유연한 구조입니다.

서비스 인터페이스와 서비스 구현 클래스 간에 관계는 다음의 예로 볼 수 있습니다.



3) 서비스 컴포넌트에 원격 엑세스를 하는 경우

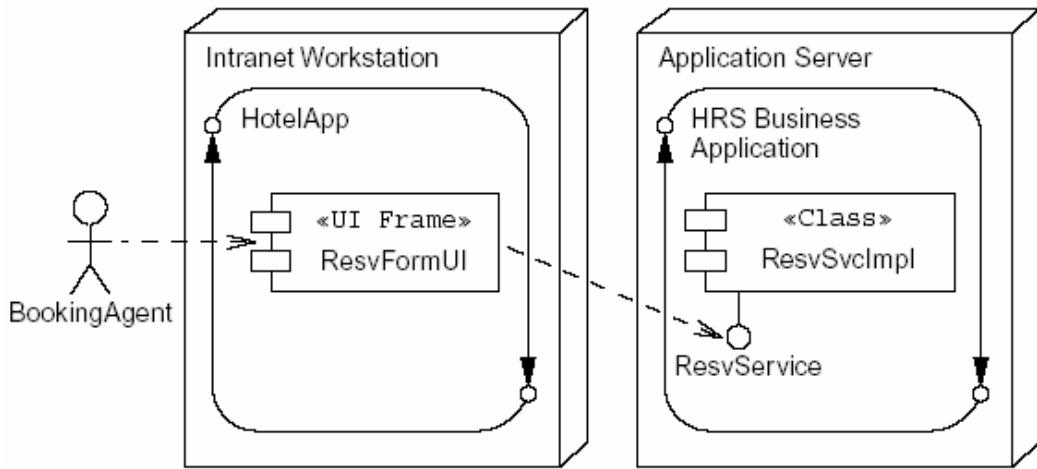
만약 서비스 컴포넌트가 떨어져 있는 시스템에 있다면 어떻겠습니까?

이것을 원격 액세스라고 합니다.

다음 그림은 서비스 컴포넌트에 원격적으로 액세스하는 것을 도식화한 것입니다.

[이미지]

n An Abstract version of Accessing a remote Service



(1) RMI를 사용한 원격 액세스

서비스 컴포넌트에 원격 액세스 하는 방법은 무엇입니까?

이를 위해서는 네트워크 커뮤니케이션 프로토콜뿐만 아니라 잘 구성된 하부구조도 필요로 합니다.

RMI를 사용한다면 어플리케이션에서 두 가지 컴포넌트를 사용해야 합니다.

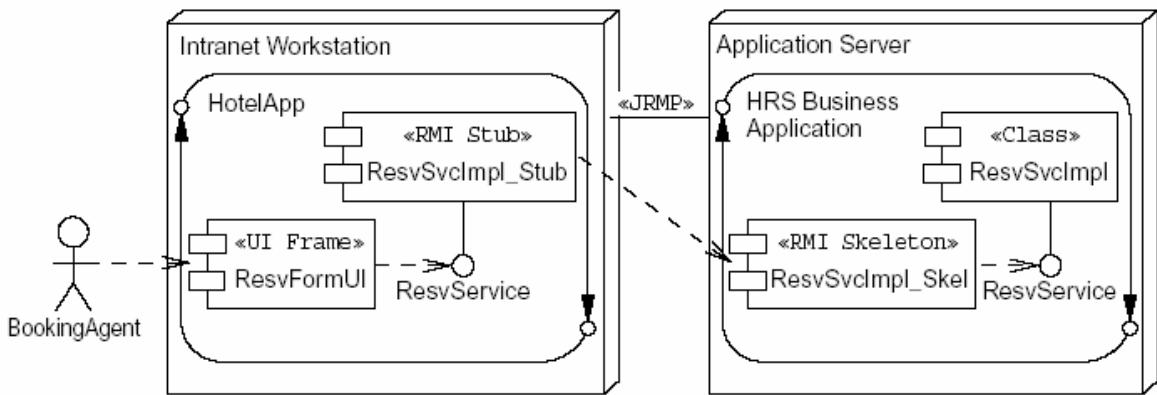
클라이언트 어플리케이션에서는 **RMI** 스텝 컴포넌트를 사용합니다. 이것은 구현된 서비스 클래스의 인터페이스와 통신할 수 있도록 구현된 컴포넌트입니다. 클라이언트 호출은 스텝을 통해서 서버의 스켈레톤 컴포넌트로 전달되어집니다. 이때 사용되는 프로토콜을 **JRMP**라고 합니다.

스켈레톤은 클라이언트 호출을 실제 서비스 컴포넌트가 수행할 수 있도록 전달합니다.

다음 그림은 **RMI**를 사용한 원격 액세스를 보여줍니다.

[이미지]

n Accessing a Remote Service With RMI infrastructure



참고하세요

n 스텁, 스켈레톤 코드의 구현

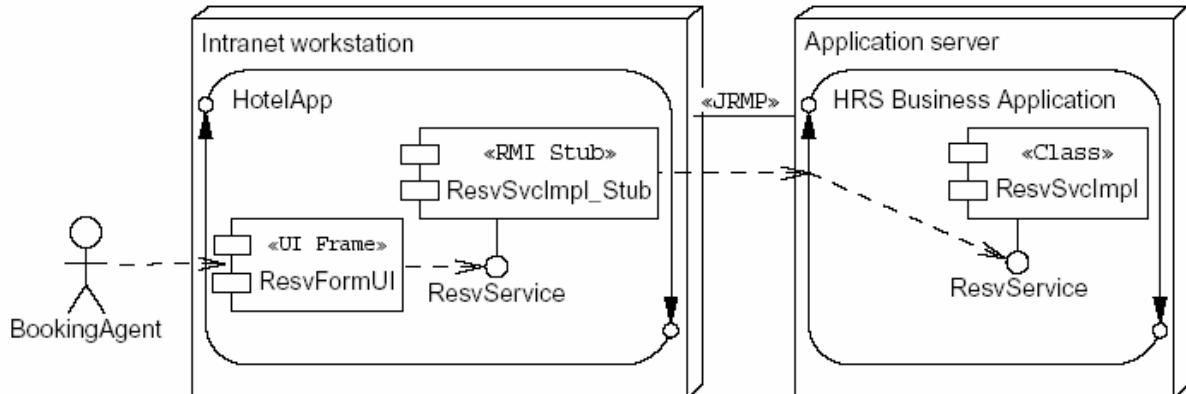
RMI는 개발자가 직접 스텁과 스켈레톤 컴포넌트를 구현할 필요를 요구하지 않습니다. J2SE Software Development Kit에는 **rmic**라는 도구로 스텁과 스켈레톤 컴포넌트가 자동 생성됩니다.

J2SE v1.2부터는 서버측 스켈레톤 컴포넌트가 더 이상 필요하지 않습니다.

다음 그림은 더 단순화된 RMI 아키텍처를 도식화 한 것입니다.

이미지

n Accessing a Remote Service Without a Skeleton component



(2) RMI를 사용한 동시 엑세스

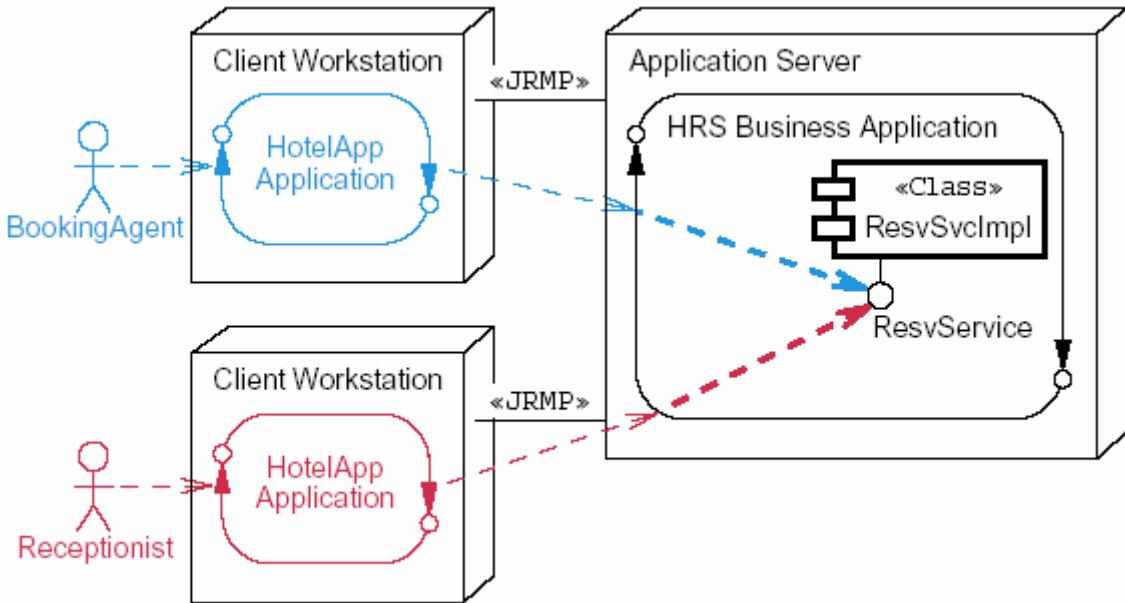
RMI 객체는 여러 사용자에게 동시에 엑세스 될 수 있습니다.

각각의 클라이언트 Request는 각각의 쓰레드를 생성하여 RMI 객체의 서비스 메소드를 수행하게 됩니다.

다음 그림은 **RMI**의 동시 액세스를 보여줍니다

[이미지]

□ A Remote Service Is an Active Component



UML에서 자신의 쓰래드 안에서 수행되는 컴포넌트를 **active** 컴포넌트라고 하며 굵은 테두리로 컴포넌트 다이어그램을 그립니다. **ResvServiceImpl**은 **BookingAgent**와 **Receptionist**에 의해 동시 액세스 되고 있습니다.

(3) RMI를 사용한 파라미터 전달

RMI 원격 서비스 메소드의 호출 시에 파라미터의 전달은 두 가지로 이루어집니다.

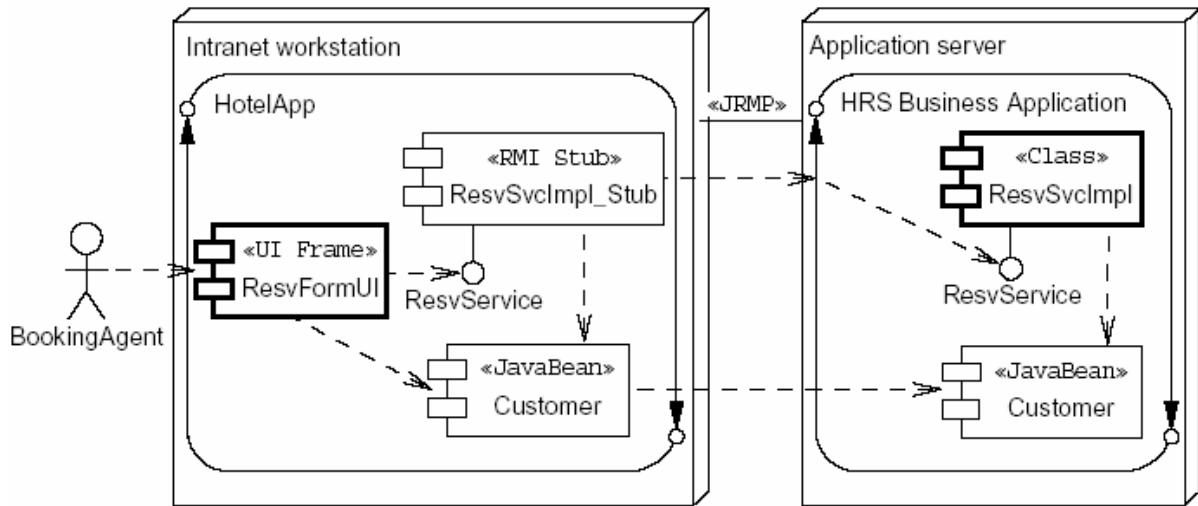
만약 데이터가 **Primitive** 데이터일 때는 값이 직접 전달됩니다.

데이터가 객체일 경우는 객체 자체가 전달되어 집니다. 네트워크를 경유하여 객체가 **Byte- by- Byte**로 전달되어 지는 것을 **Serialization** (직렬화)라고 합니다.

다음 그림은 **Serialization**을 보여주는 **RMI** 아키텍처입니다.

[이미지]

□ RMI Uses Serialization to Pass Parameters



클라이언트 어플리케이션의 **Customer**라는 자바빈이 서버쪽으로 전달되어 집니다. 이는 클라이언트측 객체가 서버측으로 이동하는 것이 아니라 복사가 되는 것입니다. 따라서 직렬화가 막 끝난 시점의 두 객체는 서로 내용이 같지만 양쪽에서 각각의 객체의 내용을 바꾼다 해도 서로 독립적이므로 아무 영향을 받지 않습니다.

(4) RMI를 사용한 서비스 찾기

클라이언트는 원격 서비스 객체를 어떻게 얻을까요?

RMI에서는 RMI 레지스트리를 통해 원격 객체를 찾아옵니다.

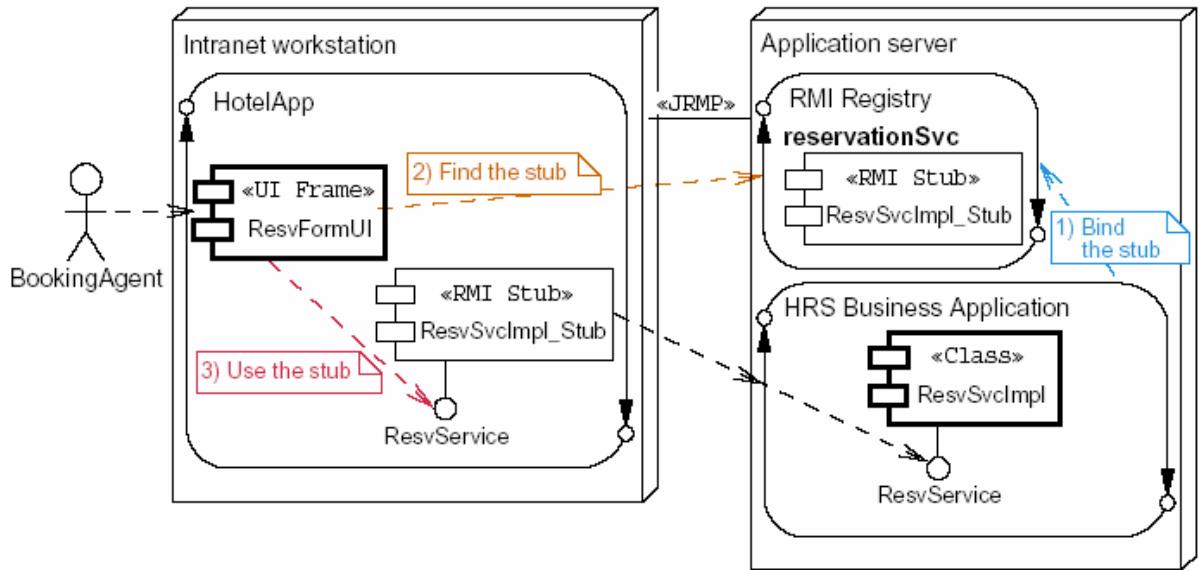
원격 서비스 객체를 찾는 과정은 다음과 같습니다.

1. 원격 서비스 객체가 RMI 레지스트리에 등록됩니다.
2. 클라이언트 어플리케이션은 RMI 레지스트리에서 이 원격 객체를 찾습니다.
이때 서비스 객체가 직접 얻어지는 것이 아니라 스텁 객체가 얻어지게 됩니다.
3. 클라이언트 프로그램이 스텁 객체를 사용하여 서비스를 요청합니다.

다음은 이 과정을 도식화 한 것입니다.

[이미지]

■ RMI Registry Stores Stubs for Remote Lookup



2. Business Tier의 아키텍쳐 모델 생성

비즈니스 티어의 아키텍쳐 모델을 생성하기 위한 과정은 다음과 같습니다.

1. 프리젠테이션 티어가 표현된 상세 배치 다이어그램을 참조합니다.
2. 상세 배치 다이어그램으로부터 아키텍쳐 템플릿을 얻습니다.
3. 프리젠테이션 티어와 함께 비즈니스 티어의 기술과 컴포넌트가 표현된 **Tier & Layer Package Diagram**을 작성합니다.

예를 위해서 호텔 예약 시스템을 **RMI**로 구축했습니다.

실제로 분산 기술을 선택할 때에는 시스템의 규모와 특성을 파악해야 합니다. 만약 호텔 예약 시스템이 대규모 사용자를 위한 서비스를 원한다면 **RMI**보다 **EJB**를 택해야 합니다. 그러나 **EJB** 기술을 사용하기 위해서는 **EJB** 서버를 구입해야만 합니다.

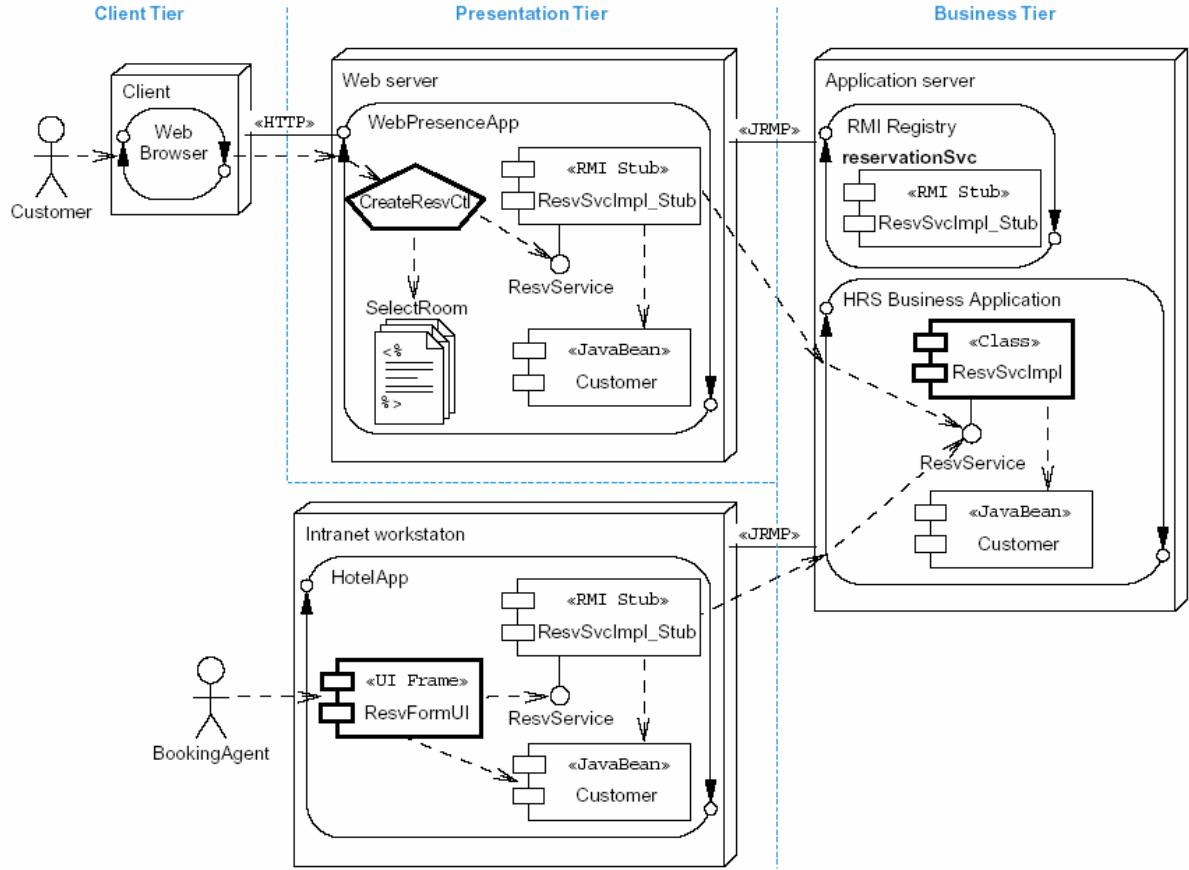
1) 상세 배치 다이어그램 참조

비즈니스 티어의 상세 배치 다이어그램은 다음과 같이 그릴 수 있습니다.

프리젠테이션 티어에서의 **RMI** 호출은 비즈니스 티어에 있는 **ResvService**라는 원격 인터페이스에게 전달되어지고 **ResvSvclmpl**이라는 원격 객체에 의해서 수행되어집니다.

[이미지]

■ HotelApp 상세 배치 다이어그램



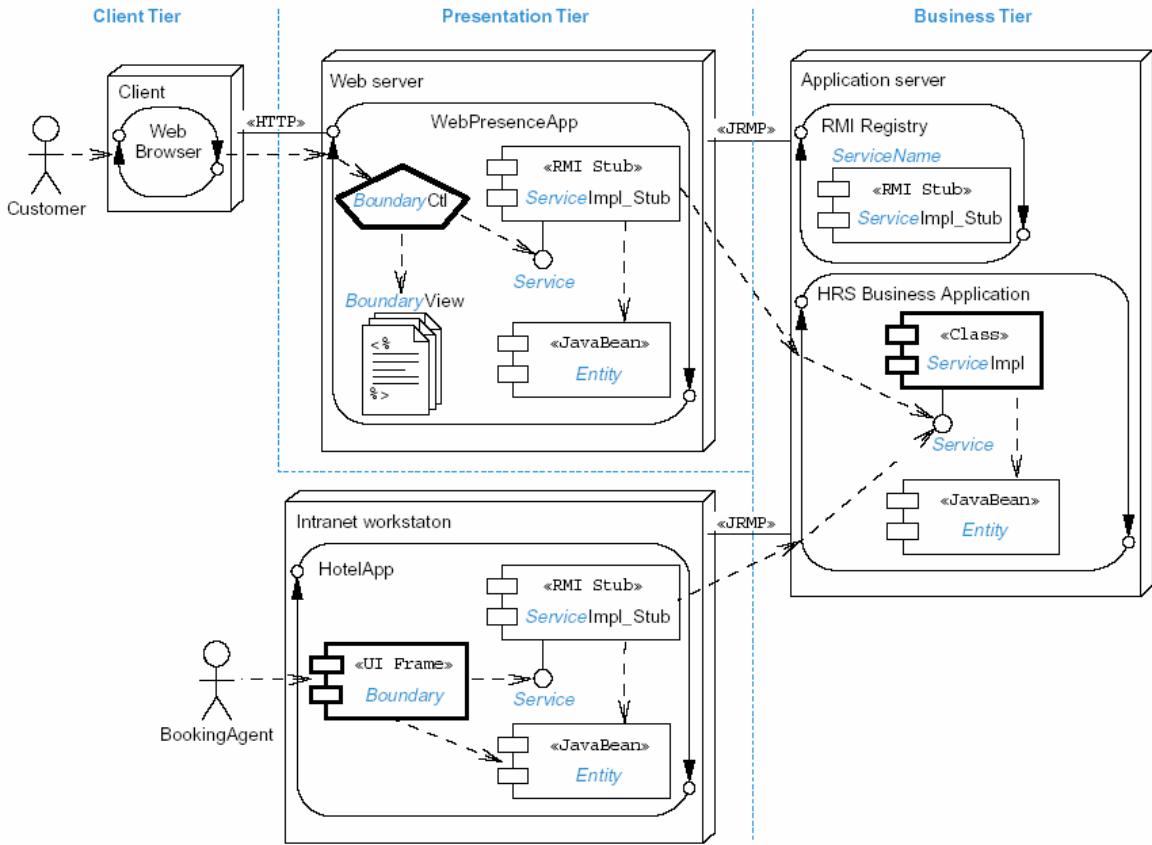
2) 아키텍쳐 템플릿 생성

아키텍쳐 템플릿은 상세 배치 디아이어그램에서 표현했던 컴포넌트의 구체적인 이름 대신에 컴포넌트의 유형을 표현합니다.

다음 그림이 호텔 예약 시스템의 비즈니스 티어의 아키텍쳐 템플릿입니다.

[이미지]

■ HotelApp 아키텍쳐 템플릿

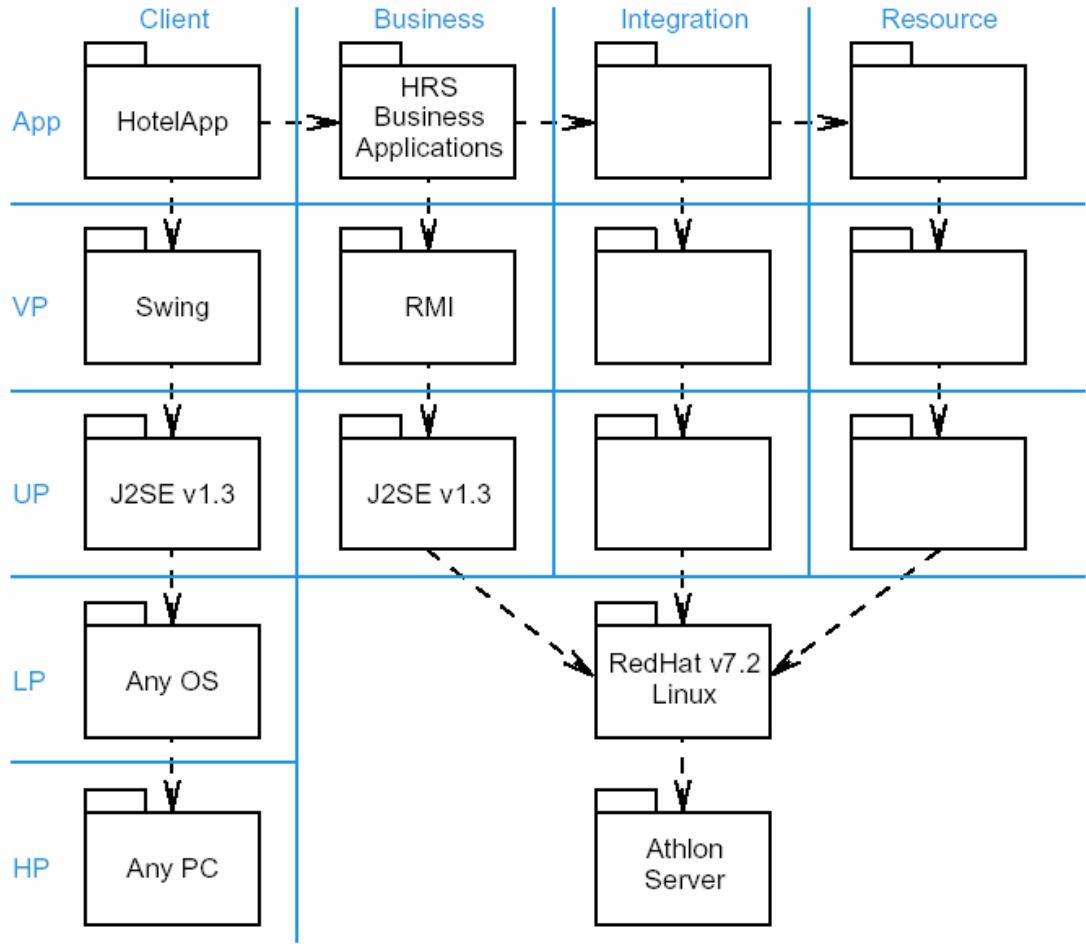


3) Tier and Layer Package Diagram 생성

이제 비즈니스 티어의 **Tier & Layer Package Diagram**을 생성할 단계입니다. 이때 비즈니스 티어를 배치하게 될 시스템에 대한 구체적인 정보를 제공하게 됩니다. 예를 들어 비즈니스 티어를 구축하게 될 시스템의 하드웨어가 **AthlonServer**이고 운영체제로 **RedHat v7.2Linux**를 사용한다면 다음과 같이 그릴 수 있습니다.

[이미지]

■ HotelApp Tier & Layer Package Diagram



이 그림에서 비즈니스 티어를 배치하게 될 하드웨어에 **Integration tier**와 **Resource tier**가 같아 배치되어질 것을 알 수 있습니다.

과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원5 : 비기능적 요구사항 구축

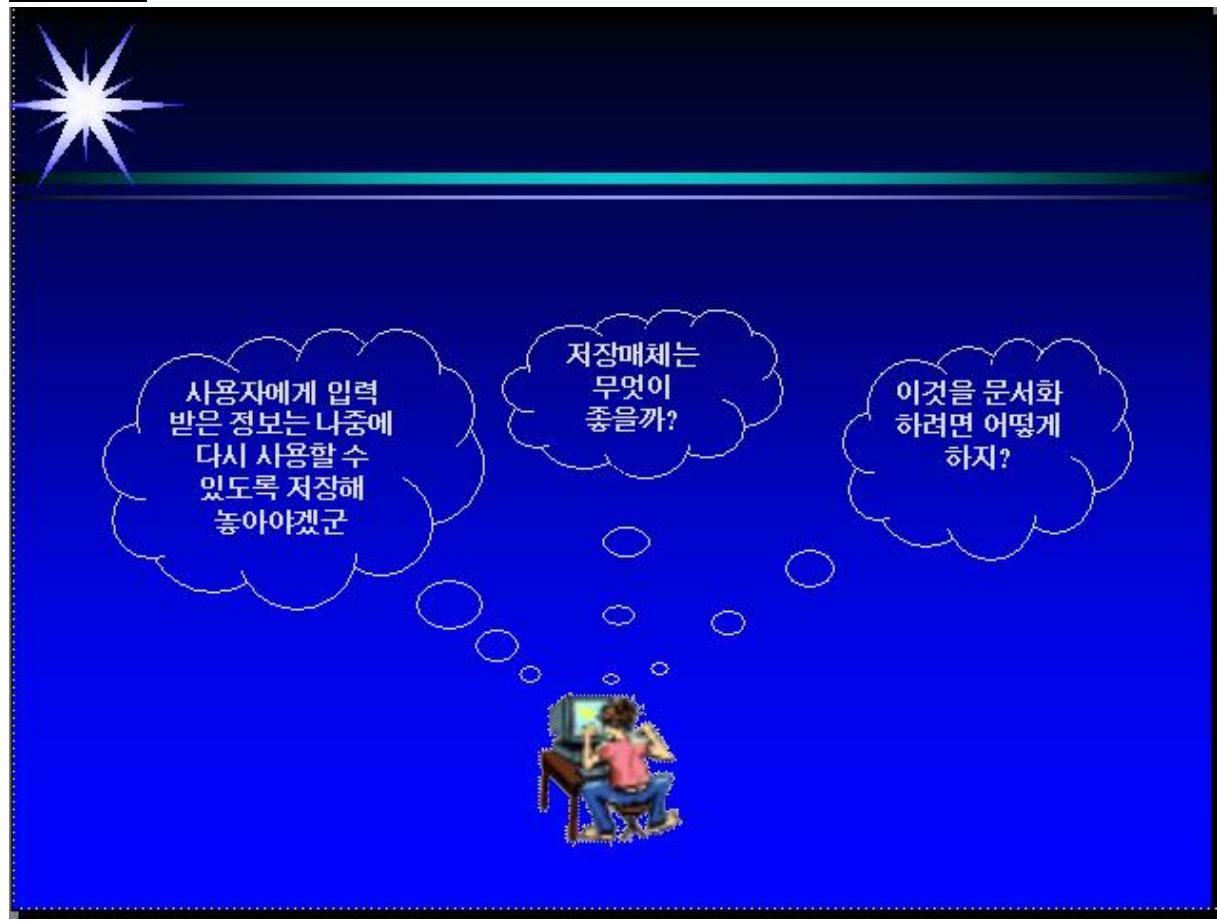
모듈 5 : Resource & Integration Tier 의 아키텍쳐 모델 생성

담당강사 : 전은수

■ 생각해봅시다 ■

호텔 예약 시스템의 경우 고객의 정보를 입력 받았다면 그 정보는 얼마나 오래 유지 되어야 할까요? 또한 고객이 예약한 예약 정보는 얼마나 오랫동안 시스템이 기억할 수 있을까요? 만약 기억해야 한다면 어떤 티어에서 어떤 기술로 기억을 진행시켜야 할까요? 우리 주변의 많은 정보들이 일시적이기 보다는 영속적으로 저장되어 필요할 때마다 다시 사용될 수 있습니다. 소프트웨어 시스템에서도 사용자 입력 정보나 시스템이 사용하는 정보들 조차도 영속적으로 저장되어 재사용되어 집니다. 여러분들은 이런 메커니즘에 대해서 알고 계십니까? 소프트웨어 시스템이 정보를 저장할 어떤 메커니즘을 가져야 한다면 그것은 개발 시에 어떻게 표현되어야 할까요?

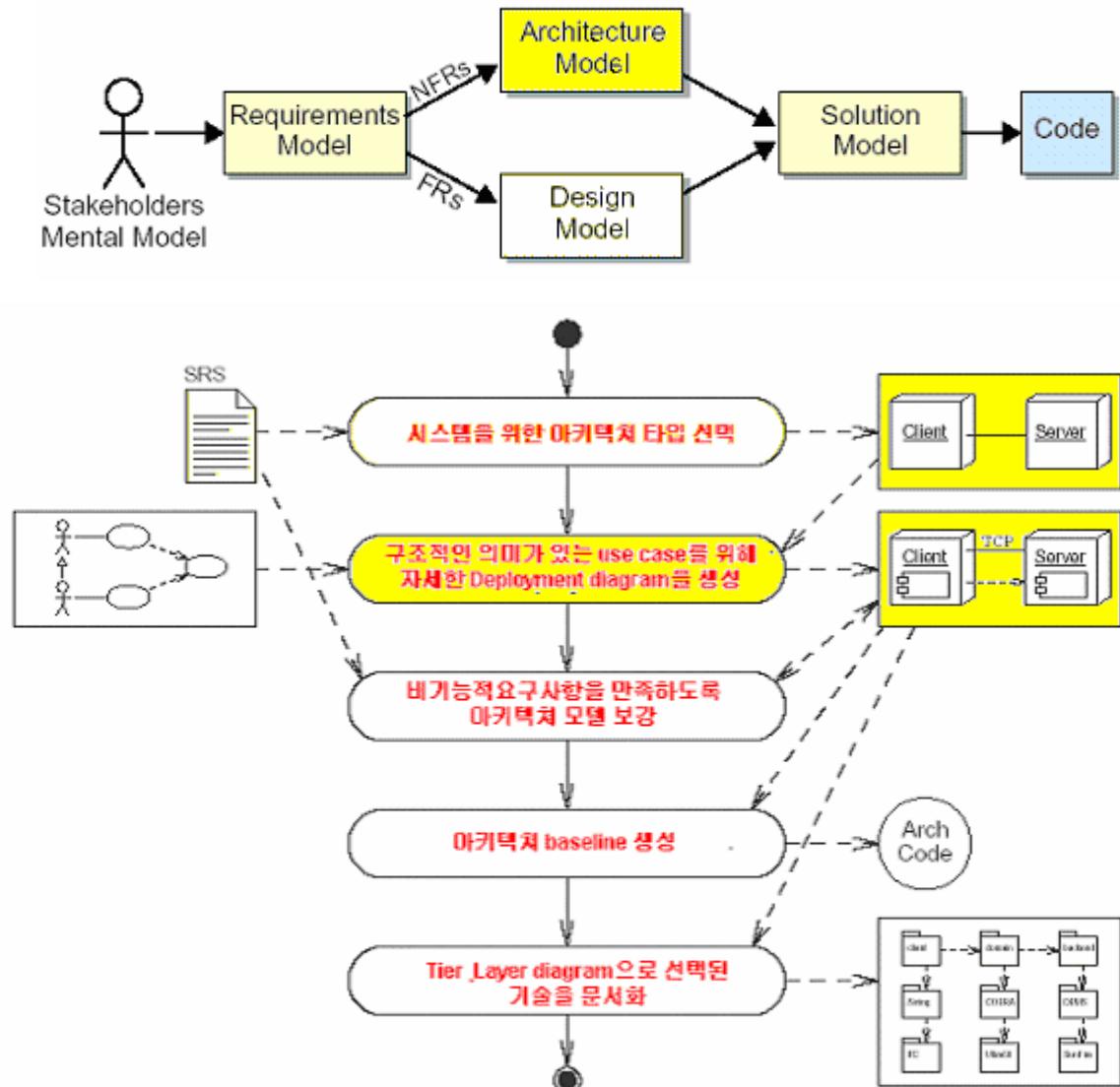
[애니메이션]



■ 학습하기 ■

참고하세요

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. Object Persistence의 이해

대부분의 어플리케이션은 다수의 동시 사용자들의 정보를 공유하기 위해 정보를 저장할 수 있는 다른 매체를 필요로 합니다.

호텔 예약 시스템의 경우 예약을 원하는 고객들이 자신의 정보나 예약 정보를 입력했을 때 시스템은 그것의 처리뿐만 아니라 그 정보를 얼마나 유지 해야 하는지를 판단해서 그 기간 만큼 저장할 수 있어야 합니다. **객체 지향 소프트웨어**의 경우 고객의 정보는 어플리케이션 상에서 객체 단위로 만들어지며 이 객체가 저장되는 것이 곧 정보가 저장되는 것입니다. **Object Persistence**란 바로 이러한 개념을 일컫는 말입니다.

그렇다면 객체, 곧 정보를 저장해야 하는 경우는 언제일까요?

모든 객체를 전부 저장해야 할까요? 이 해답을 얻어 보겠습니다.

1) 영속성이란?

영속성이란 객체의 속성이 시공을 넘어서 존재하는 것을 말한다고 **Booch**가 말했습니다.

Booch의 정의에 따르면 ‘영속 객체’는 다음의 두 가지를 의미합니다.

| 어플리케이션의 한 수행을 넘어서도 존재하는 객체

어플리케이션이 시작되면 객체가 생성됩니다. 어플리케이션이 종료 되면 이 객체의 존재는 더 이상 남아 있지 않고 정보는 전부 지워집니다. 다시 어플리케이션을 시작한다 해도 이전의 객체는 남아 있지 않습니다.

그러나 어떤 객체는 영속적으로 남아 있을 필요가 있습니다. 이 경우엔 시스템이 사용하던 객체를 저장합니다. 어플리케이션이 종료 되었다가 재시작 될 때에도 저장되었던 이전 객체를 다시 사용할 수 있습니다.

| 어플리케이션 **address space**에 무관하게 저장되는 객체

영속적으로 남아 있는 객체를 **Persistence Object**라고 하고 한 수행 시간 동안만 존재하는 객체를 **Transient Object**라고 합니다.

Transient Object는 어플리케이션의 **address space**(저장 공간)에 저장됩니다.

Persistence Object는 이 **address space**에 무관한 다른 공간에 저장되어야 합니다.

어플리케이션은 이 독립적인 다른 공간에 저장되어있던 객체를 필요 시에 자신의 **address space**로 **load**해 올 수 있어야 합니다.

또한 새로 만들어지는 **Persistence Object**도 이 독립된 외부 저장 공간에 어플리케이션이 종료 되기 전이나 그들의 정보가 지워지기 전에 저장될 수 있어야 합니다.

(1) Persistence Issues

영속성을 위해서는 다음 몇 가지를 따져봐야 합니다.

| **data storage**의 타입

어떻게 데이터를 저장할까요?

flat-file, extensible Markup Language(XML) file, relational DBMS, object-oriented DBMS 등등 많은 방법이 있습니다.

어떤 시스템에서는 **flat-file**을 사용하다가 다른 저장 장치로 바꿀 수도 있습니다. 이 경우에 어플리케이션 코드까지도 변경되어야 합니다. 데이터 저장 장치가 바뀌더라도 비즈니스 코드가 바뀌지 않을 수 있는 방법이 있을까요? **DAO** 패턴을 적용한다면 가능할 것입니다. 이 패턴은 어플리케이션을 부터 데이터 저장 방식을 숨기고 있습니다. 곧 **DAO** 패턴에 대해 자세히 다룰 것입니다.

호텔 예약 시스템은 **RDBMS**를 사용하여 데이터를 저장합니다.

| **Data Schema**가 **Domain Model**로 매핑되는지

Persistence Object는 **Domain model**의 특별한 엔티티에 상응됩니다. 우리는 이 **Domain** 엔티티에 매핑되는 **Data schema**를 생성해야 합니다.

예를 들어, 호텔 예약 시스템의 **Domain Model**에서 ‘고객’, ‘예약’이라는 엔티티는 **Data schema**로 생성되어야 합니다.

‘고객’과 ‘예약’은 서로 관계가 있는 엔티티입니다. **RDBMS**는 이러한 엔티티들의 관계도 표현 할 수 있으며 그들의 속성을 저장할 수 있습니다.

SQL의 **DDL**은 **RDBMS**에서 **Data schema**를 생성할 수 있는 랭귀지입니다.

XML에서는 **DTD**가 그러한 역할을 합니다.

| 통합 컴포넌트

강력한 두 가지의 통합 컴포넌트가 있습니다.

통합 컴포넌트란 외부 데이터 저장매체와 어플리케이션이 커뮤니케이션 할 수 있는 기술로 구현된 컴포넌트입니다.

예를 들어, **JDBC(Java Database Connectivity)**는 자바 프로그램과 관계형 데이터베이스와의 커뮤니케이션을 가능하게 하는 컴포넌트입니다.

여기서 자바 프로그램이란 비즈니스 티어의 프로그램을 말합니다. 이 때 우리는 **DAO** 패턴이라는 것을 적용할 수 있습니다.

| **CRUD 기능 : Create, Retrieve, Update, Delete**

CRUD는 데이터를 저장하는 기본적인 기능입니다. **Create**란 새로운 객체를 저장 매체로 삽입하는 것이고, **Retrieve**는 기존 데이터를 찾는 것, **Update**는 기존 데이터의 내용을 변경하며, **Delete**는 기존 데이터를 삭제하는 기능

입니다.

참고하세요

n SQL

SQL에서 **create**는 **insert**, **retrieve**는 **select**, **update**는 **update**, **delete**는 **delete**라는 명령어로 제공합니다.

2) Domain Model을 위한 Database Schema 생성

Domain Model은 일반적으로 두 단계를 거쳐 Database schema로 생성됩니다.

하나는 논리적 스키마이고, 하나는 물리적 스키마입니다.

이 모듈에서는 OO 엔티티가 DB 테이블로 매핑되는 것을 간단히 소개합니다.

논리적 스키마는 **ERD(Entity- Relationship Diagram)**에 의해서 표현됩니다. 도메인 엔티티가 관계형 테이블로 매핑될 때 여러 테이블들의 관계를 표현한 그림을 **ERD**라고 합니다. 도메인 엔티티 클래스의 각 인스턴스(객체)가 대응되는 DB테이블의 한 행으로 저장됩니다. 도메인 엔티티의 변수는 테이블의 **field(열,Column)**로 표현됩니다.

도메인 엔티티 사이의 관계는 테이블 사이에서 외래키(**Foreign- key**)관계로 표현됩니다.

물리적인 **ERD**는 논리적 **ERD**에 필드의 **Data Type**과 테이블의 인덱스, 데이터 제약 사항 등을 더한 것입니다.

논리적 **ERD**는 분석과 아키텍처 수립 시에 도출되고, 물리적 **ERD**는 디자인 워크플로우 동안 논리적 **ERD**를 바탕으로 생성됩니다.

도메인 모델을 논리적 **ERD**로 만드는 작업은 다음과 같이 진행됩니다.

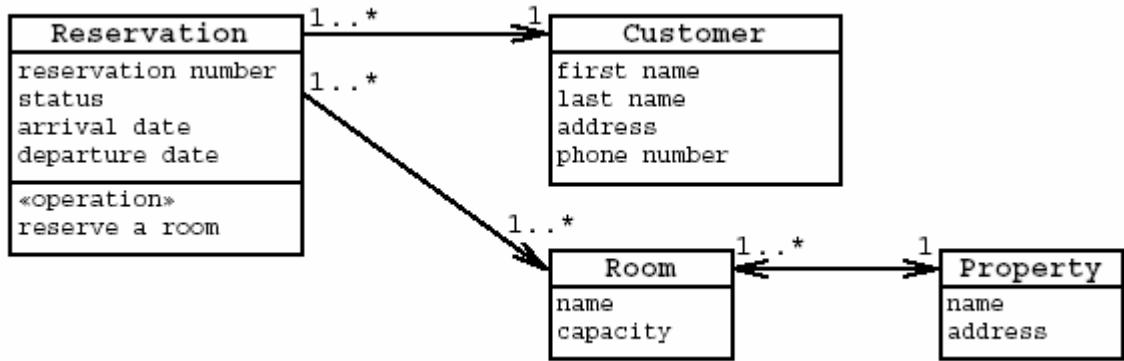
1. 도메인 엔티티를 테이블로 바꿉니다.
2. 각 테이블에서 **Primary Key**(유일 키)를 정합니다.
3. 일대다 혹은 다대다의 **ER** 관계를 생성합니다.

(1) HRS(Hotel Reservation System)의 Database Schema 생성

호텔 예약 시스템의 도메인 모델을 간단히 표현하고 그것을 **Database Schema**로 생성해 봅니다.

[이미지]

n 호텔 예약 시스템의 도메인 모델



간략화 된 호텔예약 시스템은 **Reservation(예약)**, **Customer(고객)**, **Room(객실)**, **Property(부대시설)** 의 클래스로 도메인 모델을 표현하고 있습니다.

Step 1 – OO entity를 DB 테이블로 매핑

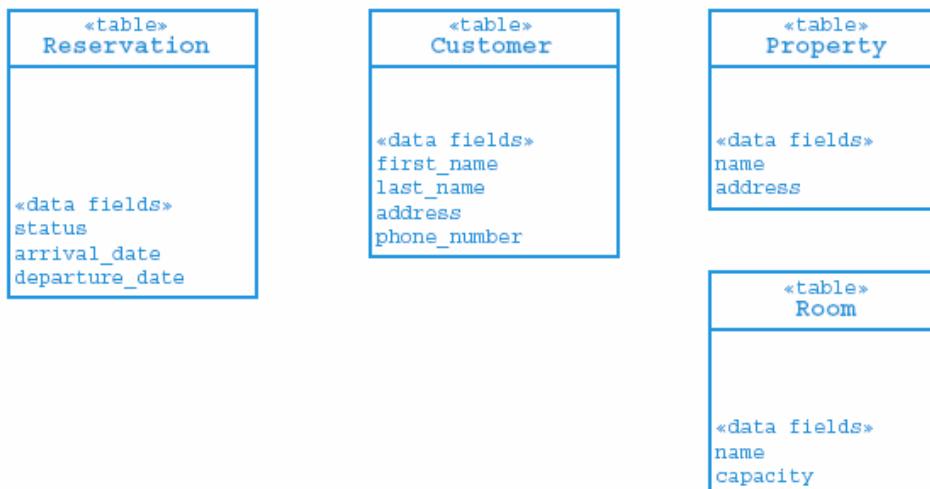
클래스 다이어그램을 사용하여 ERD를 쉽게 그릴 수 있습니다.

각 클래스 노드는 하나의 테이블로 매핑됩니다. 특별히 테이블 노드를 표현하기 위해서 <<table>> 이라는 스테레오 타입을 사용하도록 합니다.

호텔 예약 시스템의 경우 도메인 모델 안에서의 4개의 클래스는 4개의 테이블로 생성될 수 있습니다.

[이미지]

n Step 1 – Creating Entity Tables



Step 2 – 각 테이블에서 유일키(Primary Key) 설정

관계형 테이블의 유일키는 테이블안의 객체를 유일하게 식별할 수 있는 필드입니다. 테이블의 한 행은 도메인 객체 하나를 매핑합니다. 엔티티에는 두 가지 타입이 있습니다.

I Independent (독립적 엔티티)

독립적 엔티티는 다른 엔티티와는 무관하게 존재하는 엔티티를 말합니다.

호텔 예약 시스템의 경우 **Reservation, Customer, Property** 엔티티는 독립적입니다.

I Dependent (의존적 엔티티)

의존적 엔티티는 다른 엔티티의 컨텍스트 내에 존재하는 엔티티를 말합니다.

예를 들어, **Hotel room**은 **Hotel Property**의 컨텍스트 내에서만 존재합니다.

그러므로 **Room** 엔티티는 **Property** 엔티티에 의존적입니다.

[이해하기]

n 의존적 엔티티

증권 관리 프로그램을 만든다면 이 시스템의 도메인 모델에는 고객, 주식, 포트폴리오라는 엔티티를 생성할 수 있습니다. 여기서 포트폴리오는 어떤 고객이 어떤 주식을 얼마나 가지고 있느냐는 정보를 가진 엔티티입니다. 이때 고객과 주식은 독립적인 엔티티인데 반해 포트폴리오는 고객 엔티티와 주식 엔티티가 없으면 존재할 수 없는 엔티티입니다.

이런 엔티티를 의존적 엔티티라고 합니다.

독립적인 엔티티의 유일키는 행을 유일하게 구분할 수 있는 것으면 무엇이든 가능합니다.

Reservation의 경우 예약 번호 같은 것을 부여하고 이것을 유일키로 설정할 수 있습니다.

Customer의 경우 주민번호 같은 것을 고객 번호로 부여해 유일키로 설정할 수 있습니다.

Property의 경우 시설물 고유 번호를 부여해 유일키로 설정할 수 있습니다.

그러나 의존적인 엔티티의 경우는 유일 키가 복합되어져 생성됩니다.

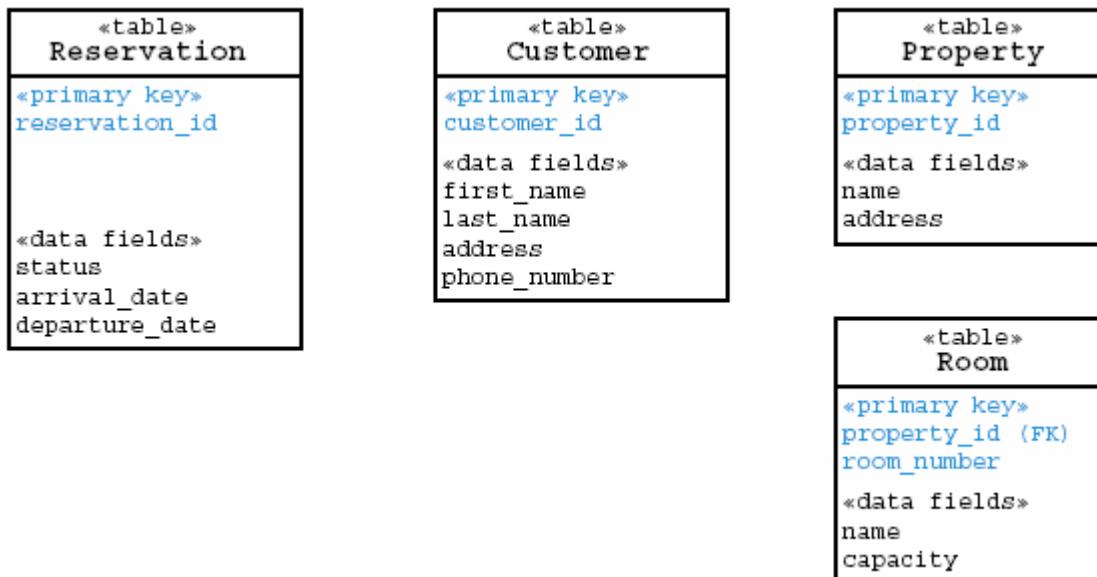
예를 들어, **Room**은 ‘부대시설 몇 번에 속해 있는 객실 몇 번’ 하는 식으로 구별해야만 유일하게 구별 할 수 있을 것입니다.

따라서 **Room** 안에서 **Property**의 유일키를 참조하고 있어야만 하는데 이것을 참조키 혹은 외래키(**Foreign key**)라고 합니다.

다음 그림은 각 테이블에서 유일키를 지정한 상태입니다.

[이미지]

n Step 2 – Specifying Primary key



Step 3 – 일대다 관계 표현

도메인 모델에서의 각 관계는 **ERD**에서는 외래키로 표현되어집니다.

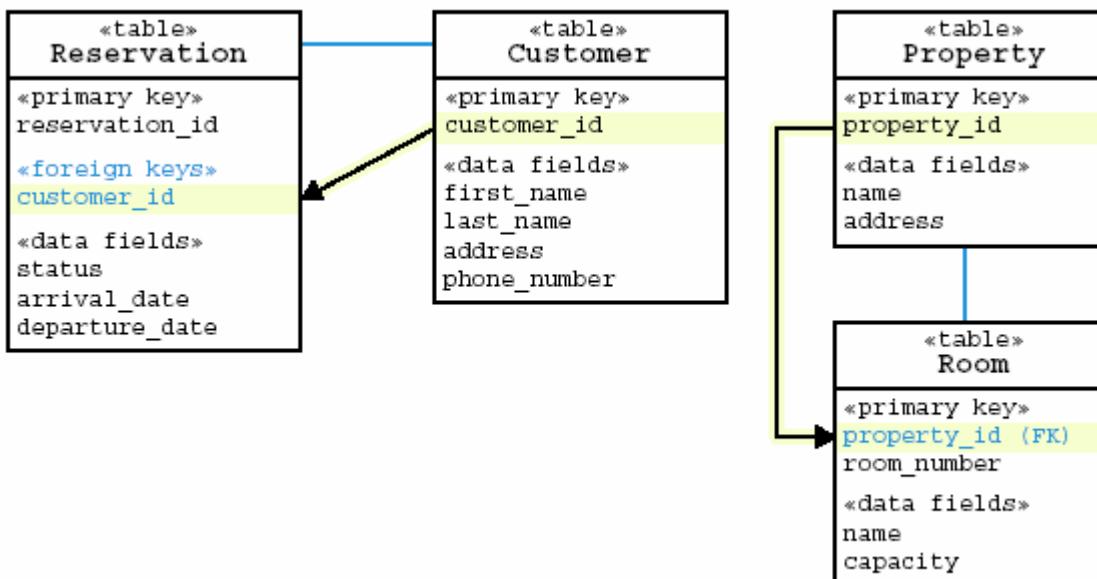
외래키는 연관된 다른 테이블의 유일키입니다.

예를 들어, 도메인 모델에서 한 예약이 한 고객과 연관되어 있으면 이것을 **ERD**로 표현할 때 **Reservation** 테이블이 **Customer** 테이블의 유일키 필드를 갖게 하면 됩니다.

다음은 테이블 간의 관계를 외래키를 통해 표현한 그림입니다.

[이미지]

n Step 3 – Creating One- to- Many Relationships



Step 3 – 다대다 관계 표현

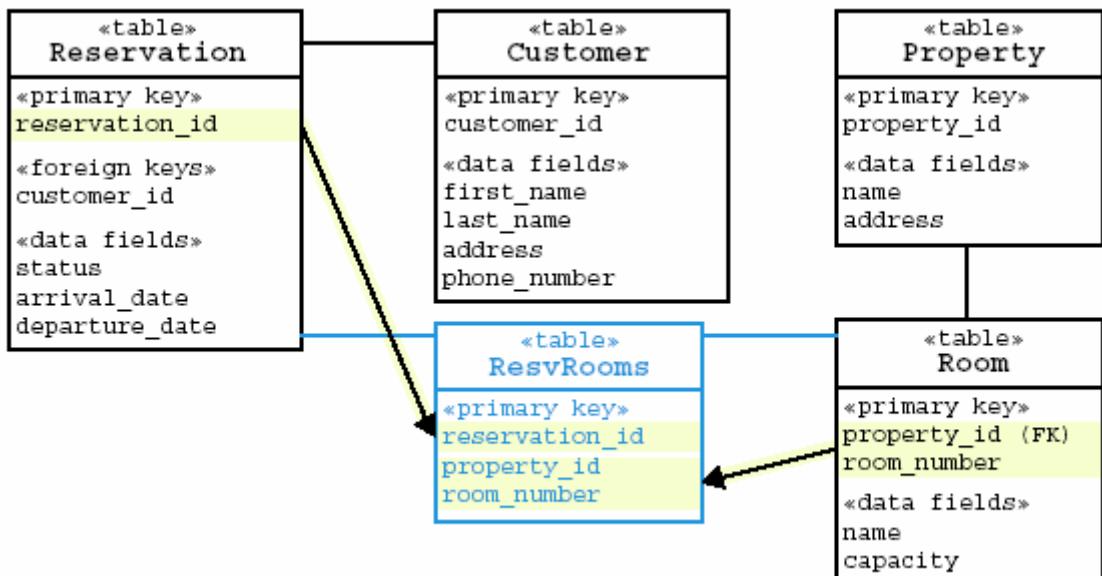
Reservation과 **Room**의 관계 같은 경우에 한 예약에 여러 방이 예약되어 질 수 있고, 한 방이 여러 예약에 속해 질 수 있다는 것을 생각하면 이것은 다대다 관계로 볼 수 있습니다.

이 경우에는 ‘예약객실’이라는 별도의 테이블을 만들어서 어떤 예약에 어떤 방이 속해있는지를 표현 할 수 있습니다.

다음 그림에서 **ResvRooms**는 그러한 테이블을 보여줍니다.

[이미지]

n Step 3 – Creating a Many- to- Many Resolution Table



ResvRooms는 **Reservation**의 유일키와 **Room**의 유일키를 외래키로 가지고 있어야하는데 **Reservation**의 유일키는 **reservation_id** 하나밖에 없지만 **Room**의 유일키는 **property_id**와 **room_number** 두개가 있습니다. 이 경우 두개 모두 **ResvRooms**의 외래키가 되어야 합니다.

2. Resource Tier의 아키텍쳐 모델 생성

Resource Tier의 컴포넌트 모델을 생성한 후에 아키텍쳐 모델의 산출물 들을 얻을 수 있습니다.

다음은 **Resource Tier**의 아키텍쳐 모델을 생성하기 위한 과정입니다.

1. 상세 배치 다이어그램에 **Resource Tier**의 컴포넌트를 추가합니다.
2. 상세 배치 다이어그램으로부터 아키텍쳐 템플릿을 얻습니다.
3. **Resource tier**의 기술과 컴포넌트를 표현한 **Tier & Layer Package Diagram**을 얻습니다.

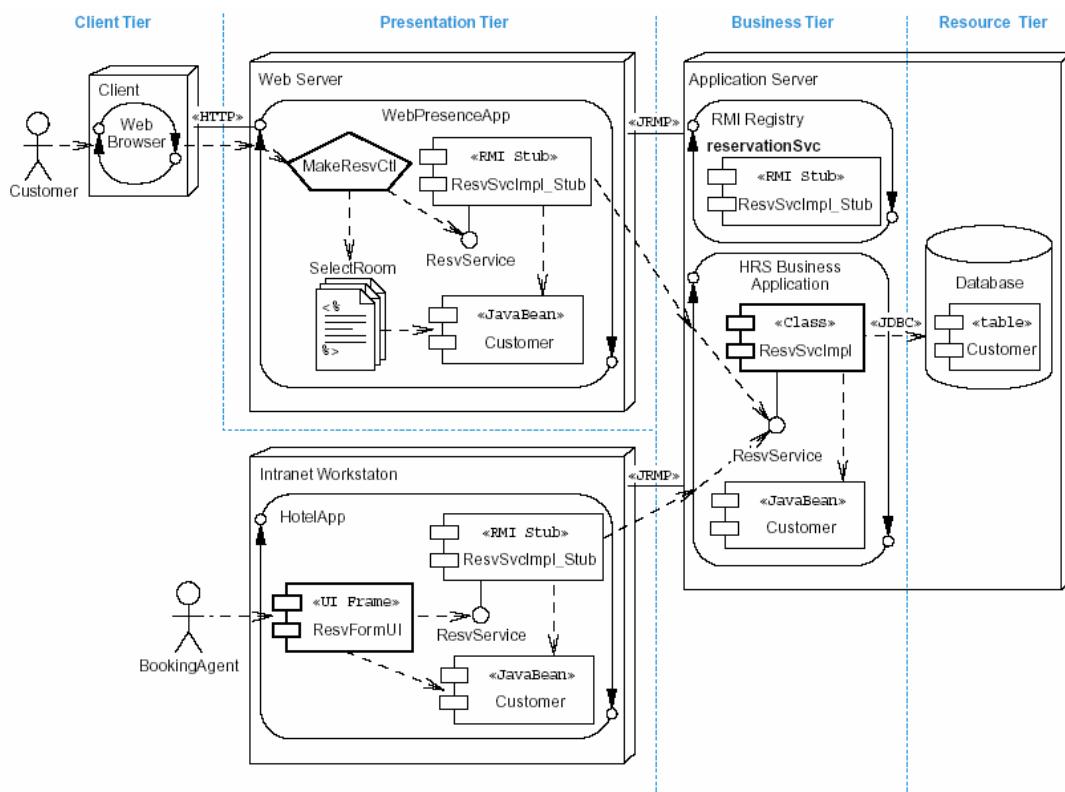
1) 상세 배치 다이어그램 참조

호텔 예약 시스템의 경우 데이터 스토리지를 **RDBMS**로 결정했습니다. 또한 비즈니스 티어에서의 도메인 엔티티를 테이블로 매핑하는 작업을 마쳤습니다.

그렇다면 이제 그것을 다음과 같이 도식화 할 수 있습니다.

[이미지]

n Resource Tier를 가진 호텔 예약 시스템의 상세 배치 다이어그램



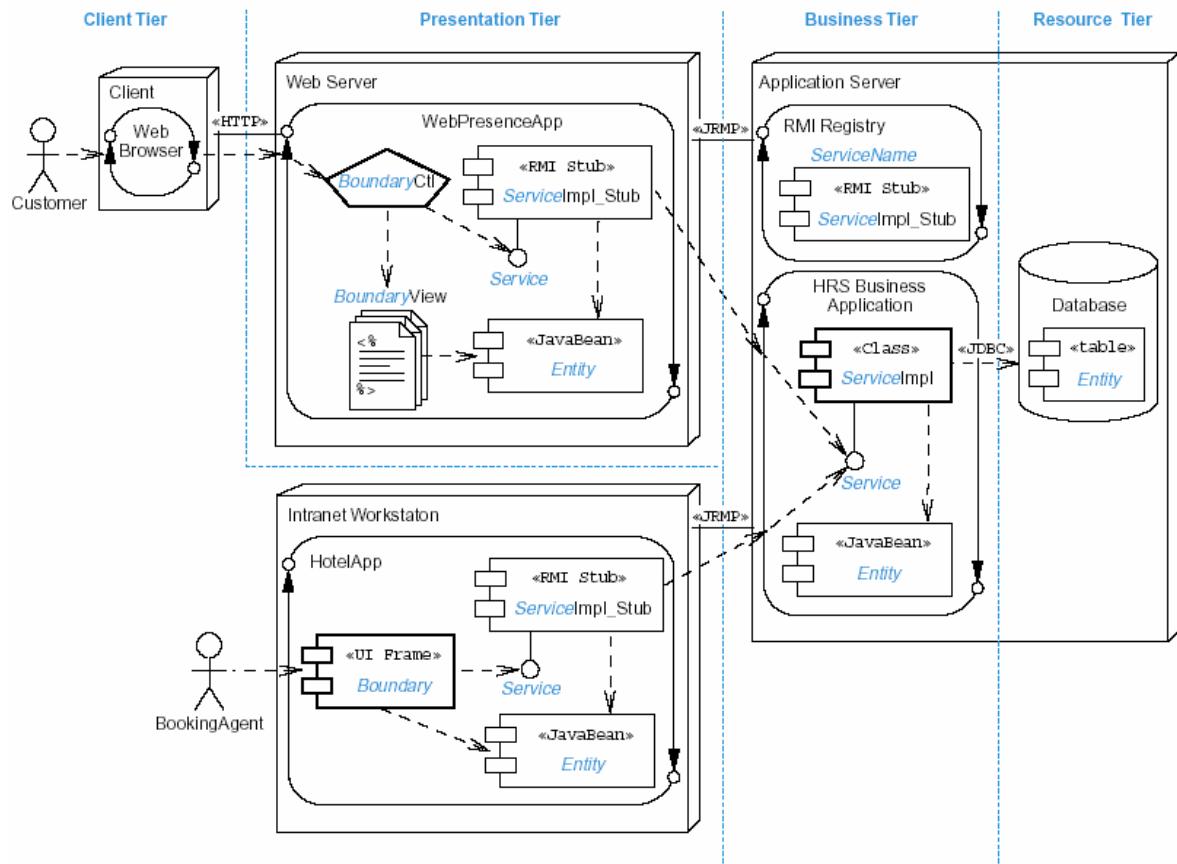
2) 아키텍쳐 템플릿 생성

상세 배치 다이어그램으로부터 아키텍쳐 템플릿을 얻습니다.

아키텍쳐 템플릿에는 각 컴포넌트의 이름 대신 타입이 표현되어 있습니다.

[이미지]

n Resource Tier를 가진 호텔 예약 시스템의 아키텍쳐 템플릿



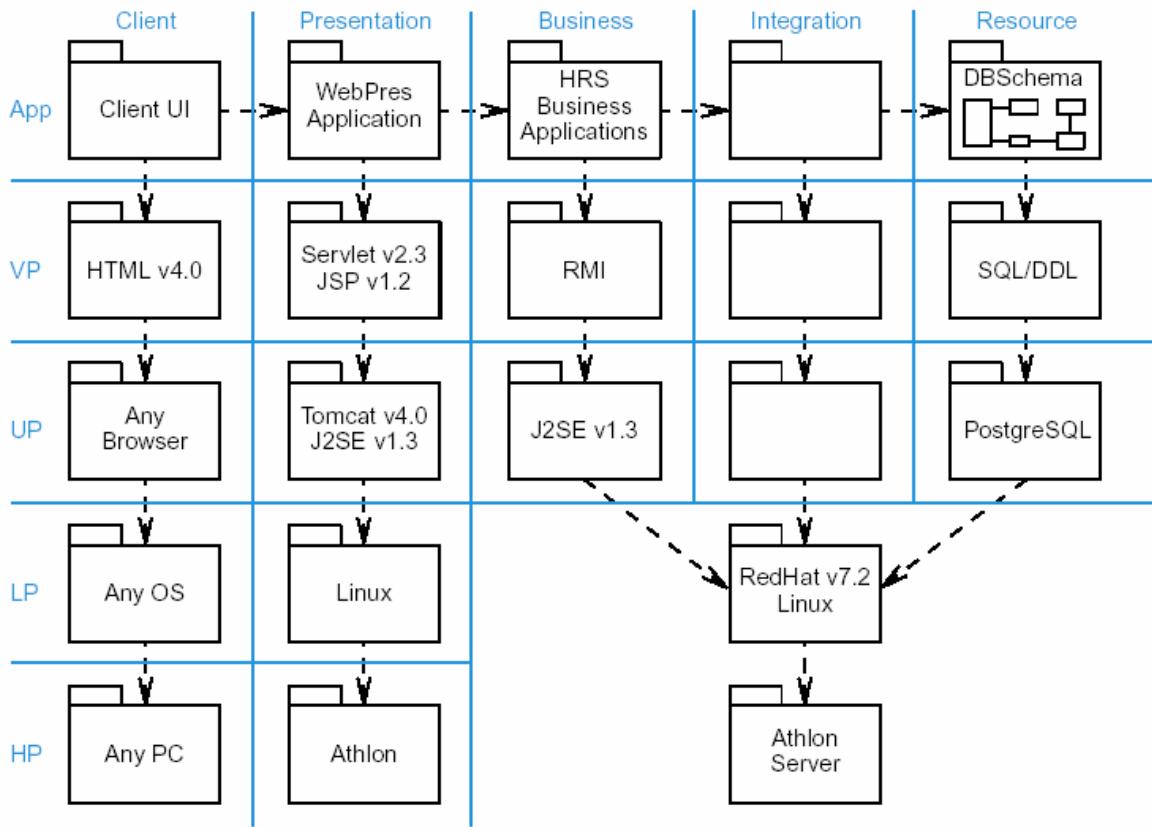
3) Tier & Layer Package Diagram 생성

Resource tier의 기술과 컴포넌트를 표현한 Tier & Layer Package Diagram을 생성합니다.

아직 Integration tier는 채워지지 않습니다.

[이미지]

n Resource tier를 가진 호텔 예약 시스템의 Tier & Layer Package Diagram



그림에서 보면 비즈니스 티어와 **Integration** 티어 그리고 **Resource** 티어가 같은 하드웨어 플랫폼에 놓여 있는 것을 알 수 있습니다.

또한 데이터 스토리지로 **PostgreSQL DBMS server**를 사용하고 있는 것을 알 수 있습니다

3. Persistence Integration Mechanism의 이해

Resource 티어의 기술과 컴포넌트가 선택되면 비즈니스 컴포넌트에서 **Resource** 티어로 도메인 엔티티를 저장하는 메커니즘이 필요하게 됩니다.

그것을 **Integration(통합)**이라고 합니다.

통합 기술에는 여러 가지가 있습니다.

다음에 간략히 소개 될 통합 기술들을 통해 여러분들의 시스템에 적합한 하나를 선택하실 수 있습니다.

1) Integration Tier Mechanism의 이해

SunTone Architecture Methodology에서의 **Integration tier**란 비즈니스 티어와

Resource 티어의 모든 리소스가 연결되는 소프트웨어 집합을 의미합니다.

리소스는 ‘통합’을 필요로 합니다.

다음은 **integration** 기술입니다.

| Data sources

Data Source는 영속적인 객체 저장 메커니즘입니다. 이것은 이 모듈의 1. **Object Persistence**의 이해에서 다룬 개념입니다.

| EIS(Enterprise Information Systems)

EIS란 이미 존재하는 소프트웨어 시스템을 의미합니다.
보통 전사적인 개념으로 모든 정보 시스템을 통합하는 것을 일컫습니다.

| Computational libraries

이것은 수학적 기능의 소프트웨어 시스템이나 시뮬레이션 기능을 가진 시스템을 통합시키는 기술입니다.

| Message Services

이것은 비동기적인 메시지 커뮤니케이션을 가능하게 하는 통합 기술입니다.

| Business to Business(B2B) services

이것은 여러 회사의 2개 이상의 어플리케이션이 서로 통합하게 하는 기술입니다.

이 모듈에서는 오직 **Data source**만 다루게 됩니다.

2) DAO Pattern

DAO 패턴은 **Data Source**를 사용할 때 적용될 수 있는 디자인 패턴입니다.

비즈니스 티어와 리소스 티어 사이에서 커뮤니케이션을 위한 통합 티어를 **Data Source**로 선택했다면 이를 구현하는 컴포넌트를 이 패턴을 적용해서 디자인 할 수 있다는 것입니다.

이 패턴을 적용할 때의 특징은 다음과 같습니다.

| 어플리케이션 티어로부터 **CRUD** 기능의 구현을 분리시킵니다.

간단히 말해서 비즈니스 컴포넌트와 통합 컴포넌트를 완전히 분리시키자는 것입니다.

이것은 데이터 스토리지가 어떤 타입으로 변했을지라도 비즈니스 컴포넌트에 직접적인 변경을 초래하지 않는다는 장점을 가지고 있습니다.

| 각 엔티티에 대해 **CRUD** 기능을 가진 **DAO** 컴포넌트를 하나씩 제공합니다.

각 도메인 엔티티에 하나씩의 **DAO** 컴포넌트를 만들라는 것은 예를 들어, **Customer** 엔티티에 **CustomerDAO**라는 컴포넌트를 만들고 이 컴포넌트가

CRUD 기능을 수행하도록 하라는 것입니다. 그러면 엔티티와 **DAO** 컴포넌트가 분리될 수 있습니다.

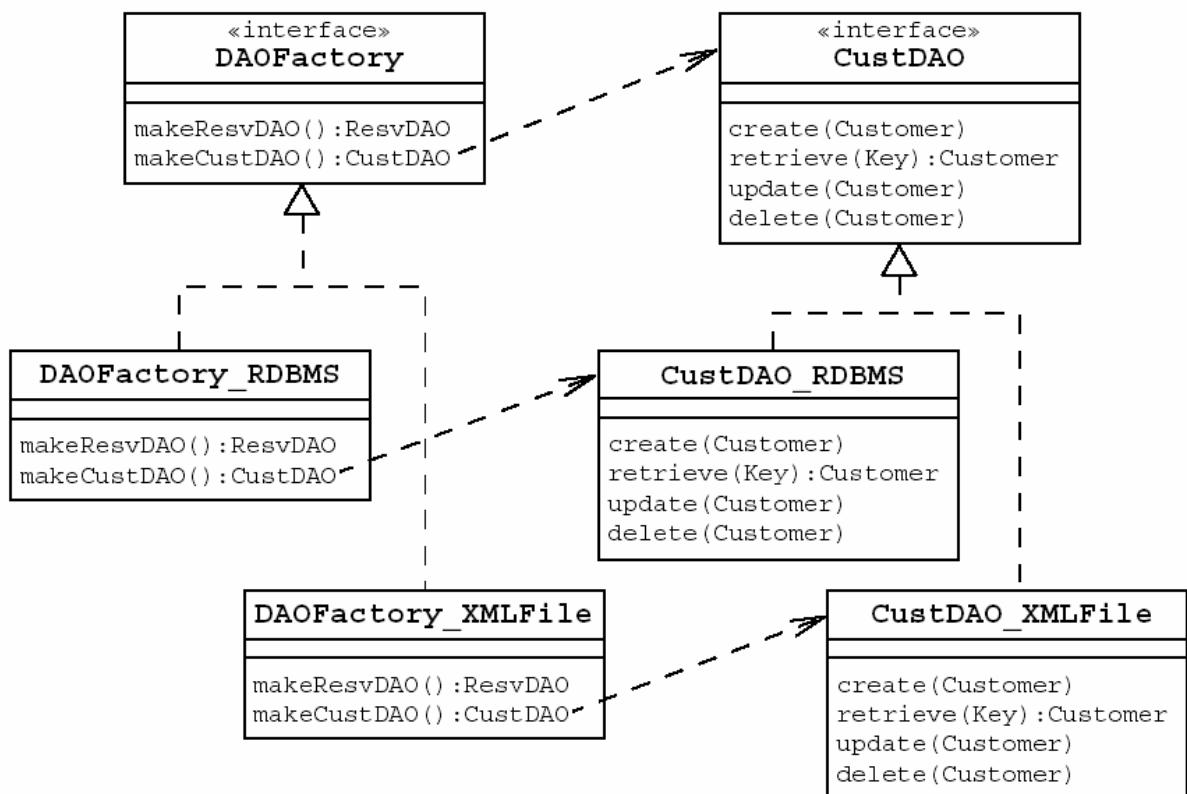
I **DAO** 컴포넌트의 추상적 인터페이스를 하나 생성합니다.

예를 들어, **CustomerDAO** 컴포넌트를 생성해 낼 수 있는 추상 메소드를 가진 **CustomerDAO interface**를 만들라는 것입니다.

다음 그림이 이해를 도울 것입니다.

[이미지]

n **DAO Pattern**



[참고하세요]

n **Factory Pattern**

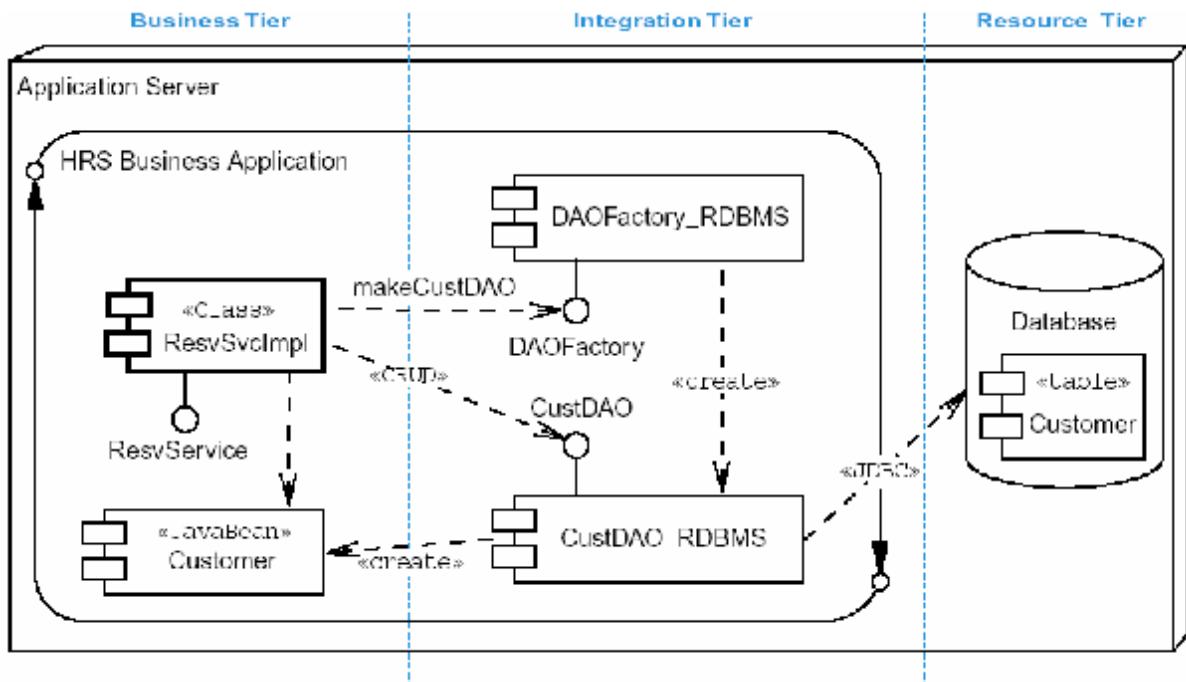
여러 구현 객체를 일괄적으로 생산해 낼 수 있는 인터페이스를 가진 디자인 유형을 **Factory Pattern**이라고 합니다.

Factory Pattern은 실행 시에 좀 더 유연한 객체 생성을 유도 할 수 있습니다.

다음은 **Integration tier**에서 **DAO Pattern**을 적용한 상세 배치 다이어그램의 일부입니다.

[이미지]

n DAO가 적용된 상세 배치 다이어그램



4. Integration Tier의 아키텍쳐 모델 생성

Integration tier의 아키텍쳐 모델의 산출물은 다음과 같이 얻습니다.

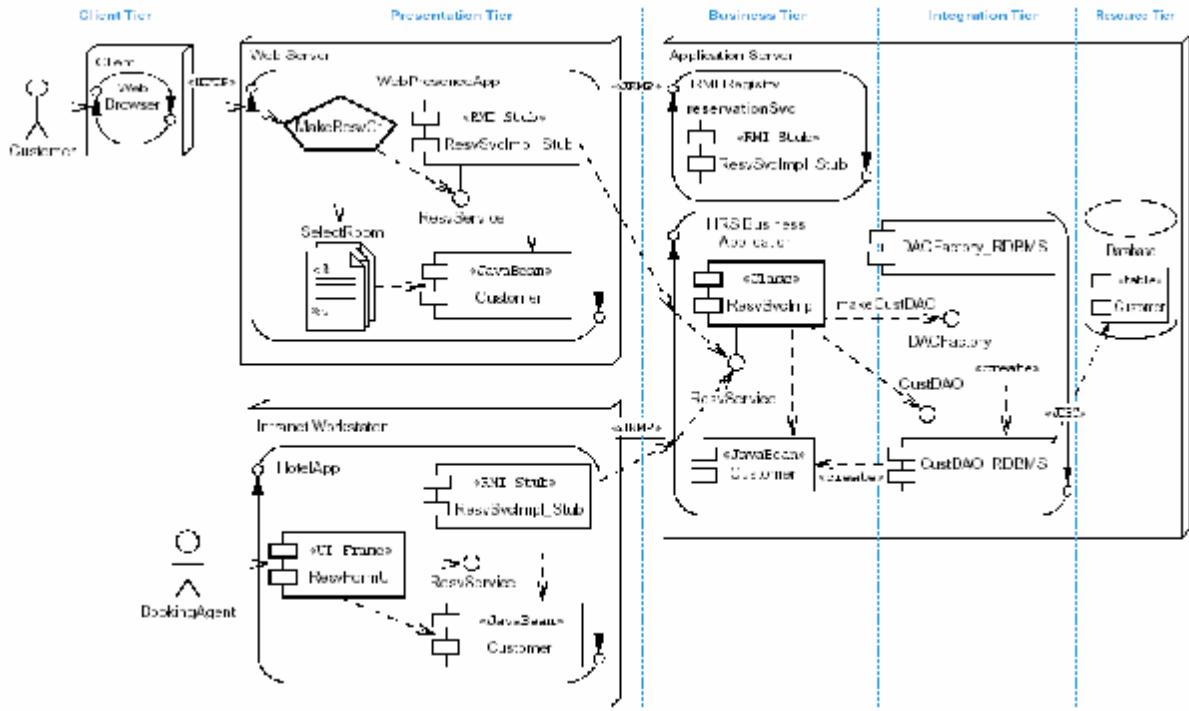
1. Integration tier의 컴포넌트를 표현한 상세 배치 다이어그램을 그립니다.
2. 상세 배치 다이어그램을 통해 아키텍쳐 템플릿을 얻습니다.
3. Integration tier의 기술과 컴포넌트를 표현한 **Tier & Layer Package Diagram**을 얻습니다.

1) 상세 배치 다이어그램 참조

Integration tier에서 DAO 패턴을 적용한 것을 표현한 호텔 예약 시스템의 상세 배치 다이어그램은 다음과 같습니다.

[이미지]

n Integration 티어를 가진 상세 배치 다이어그램

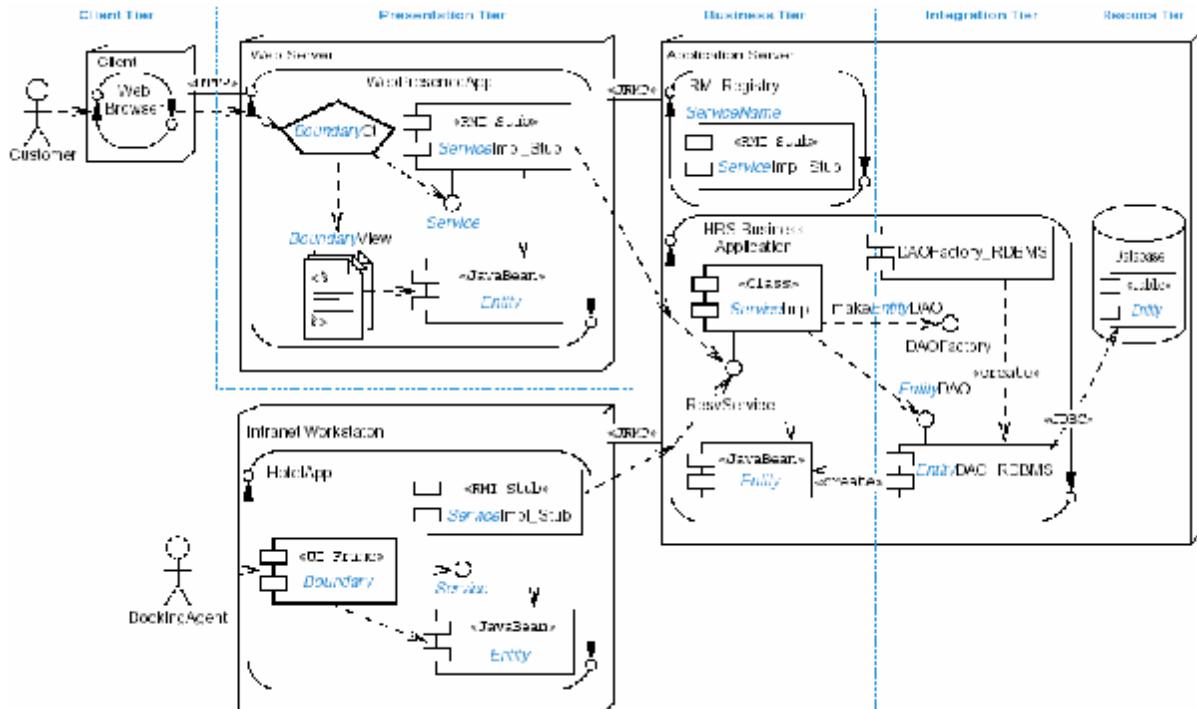


2) 아키텍쳐 템플릿 생성

상세 배치 다이어그램을 통해 아키텍쳐 템플릿을 얻습니다. 이때는 템포넌트의 이름 대신 컴포넌트의 타입을 표현합니다.

[이미지]

n Integration tier를 가진 아키텍쳐 템플릿

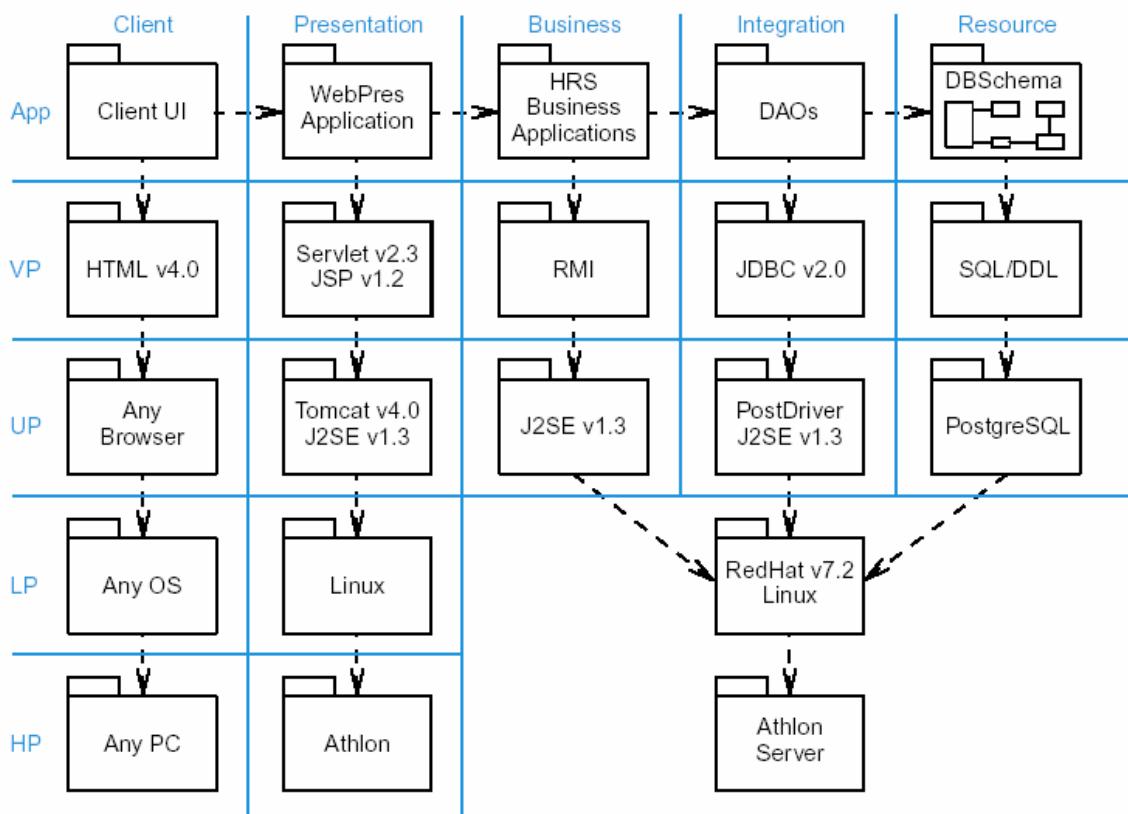


3) Tier & Layer Package Diagram 생성

Integration tier에서 Data source를 사용한 DAO 패턴을 적용했습니다. 이 기술과 컴포넌트를 Tier & Layer Package Diagram으로 문서화합니다.

[이미지]

n 완전한 호텔 예약 시스템의 Tier & Layer Package Diagram



이 그림에서 보면 호텔 예약 시스템의 데이터 통합 티어에는 **Data Source**로 **JDBC 2.0**을 사용하고 있는 것을 알 수 있습니다. 또한 **JDBC API**를 사용한 컴포넌트는 **DAO** 패턴으로 디자인 되어 있다는 것을 알 수 있습니다.

과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원6 : 솔루션 모델의 구축

모듈 1 : 솔루션 모델 생성

담당강사 : 전은수

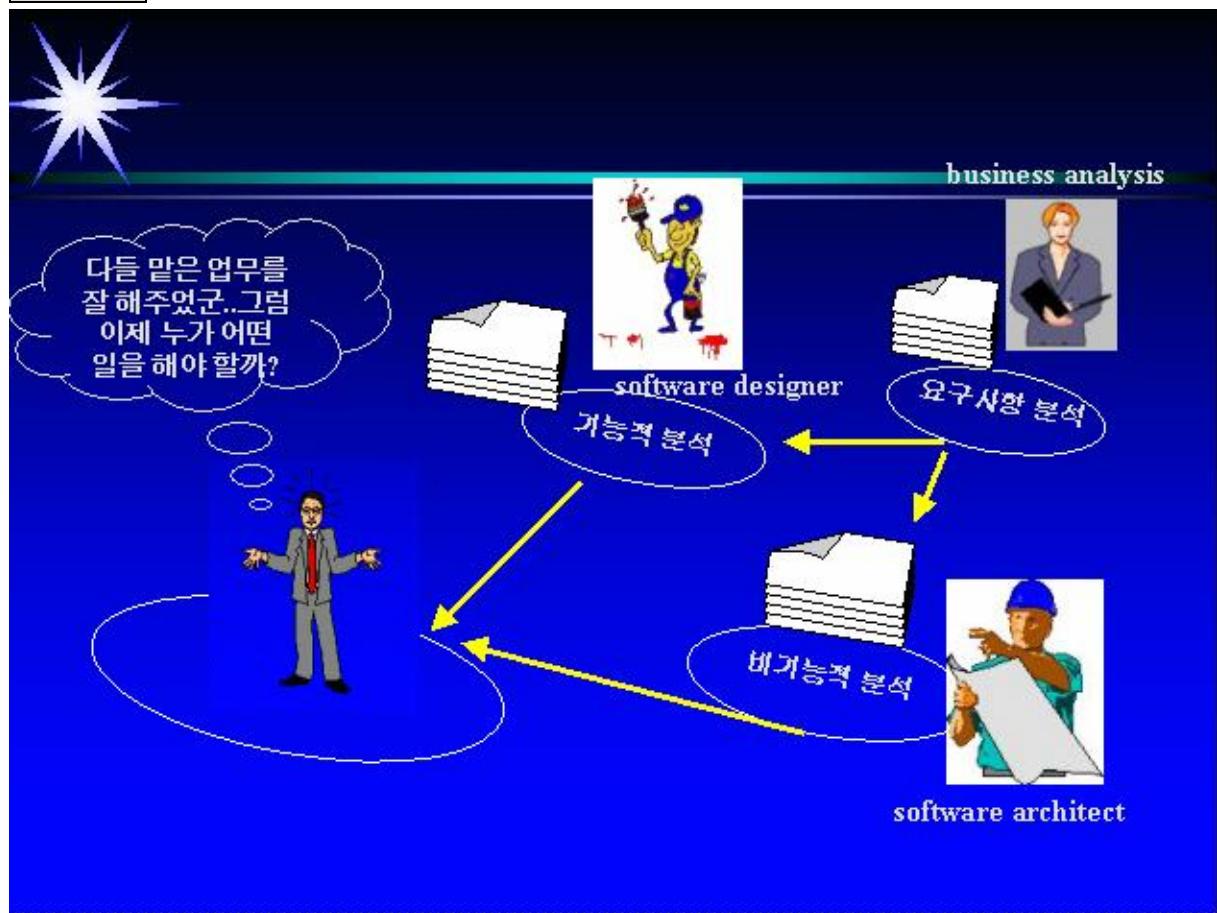
■ 생각해봅시다 ■

호텔 예약 시스템의 경우 의뢰인의 요구 사항과 프로젝트 참가자들의 요구 사항을 전부 수집한 뒤 **Business analysis**가 초기 분석을 마치면 **software designer**가 시스템의 기능적 요구 사항에 집중하여 강력 분석을 하게 되고 디자인 모델을 생성하게 됩니다. 또한 시스템의 비기능적 요구 사항에 대해서는 아키텍트가 분석, 정리하여 아키텍쳐 모델을 생성하게 되었습니다.

이제 시스템 구축에 한발 더 다가서게 된 것입니다. 그렇다면 이 시점에서는 누가 어떤 일을 해야 할까요? 모델링 된 내용을 시스템에 반영하기 위한 과정은 있을까요? 이것을 솔루션 모델이라고 합니다.

이 단원에서는 솔루션 모델의 생성과 구축에 대해 알아보겠습니다.

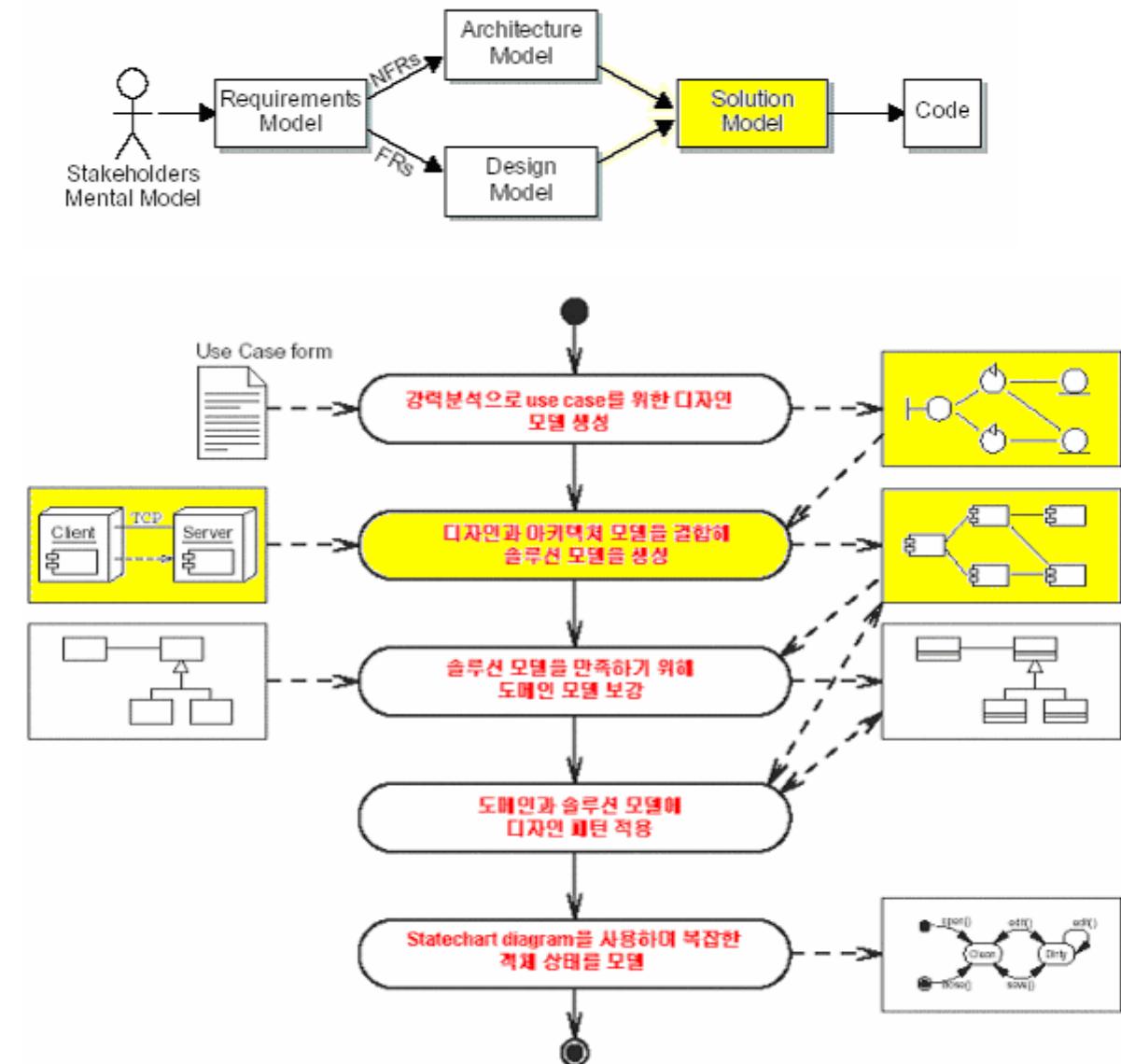
애니메이션



■ 학습하기 ■

참고하세요

n 노란색 부분이 이 모듈에서 학습하는 내용입니다.

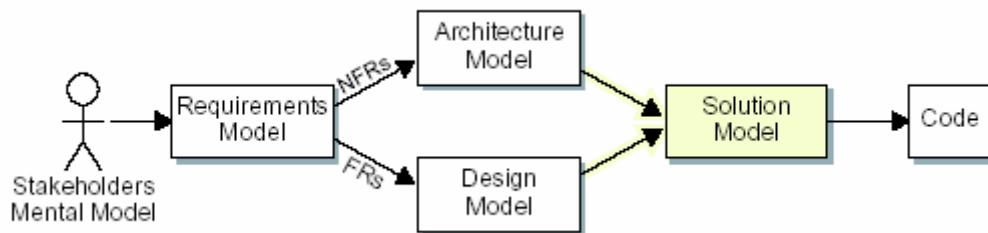


1. 솔루션 모델이란

1) 솔루션 모델의 개념

솔루션 모델은 개발팀이 시스템 솔루션의 코드를 생성하기 위한 기초가 됩니다.

다음 그림에서 보는 것처럼 디자인 모델을 아키텍쳐 모델 속으로 합해 어플리케이션 코드가 만들어 질 수 있는 전단계의 모델링을 하는 과정입니다.



개념적으로 말하면, 솔루션 모델이란 하나의 유즈케이스를 해결하기 위한 초기 소프트웨어 컴포넌트를 표현한 것을 말합니다.

이 모델은 실제 코드에 아주 근접해 있기 때문에 양이 많을 수 있습니다.

완벽한 솔루션 모델을 생성하는 것이 항상 좋은 것은 아니지만 개발팀내의 초보자들에게는 매우 중요한 모델이 되기도 합니다.

어떤 경우에는 실제 코드가 솔루션 모델이 되기도 합니다. 아키텍쳐 팀이 아키텍쳐 베이스라인을 생성했다면 여기에 디자인 모델의 내용을 더해 실제 코드로 솔루션 모델을 생성할 수도 있습니다.

이 과정에서는 솔루션 모델이 상세 배치 다이어그램으로 표현 될 것입니다.

2. GUI 어플리케이션을 위한 솔루션 모델

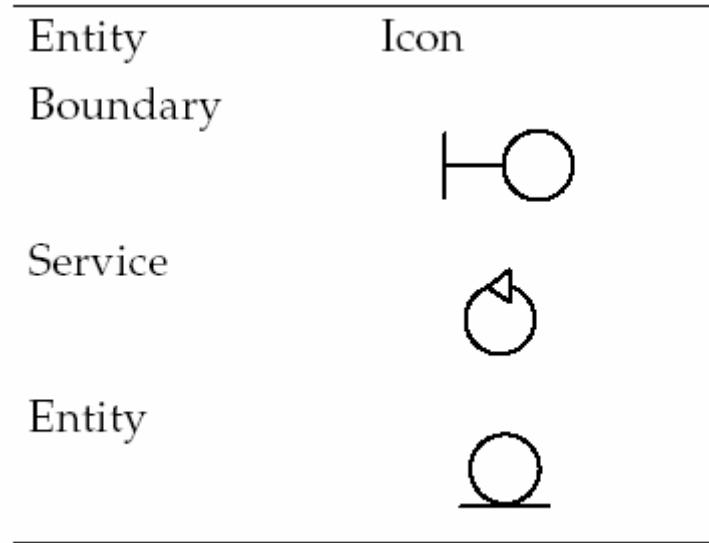
1) 개요

GUI 어플리케이션의 디자인 모델(분석 모델이라고도 합니다)은 디자인 컴포넌트의 유형으로 표현되었습니다.

다음은 디자인 컴포넌트의 유형입니다.

[이미지]

- n 디자인 컴포넌트의 유형

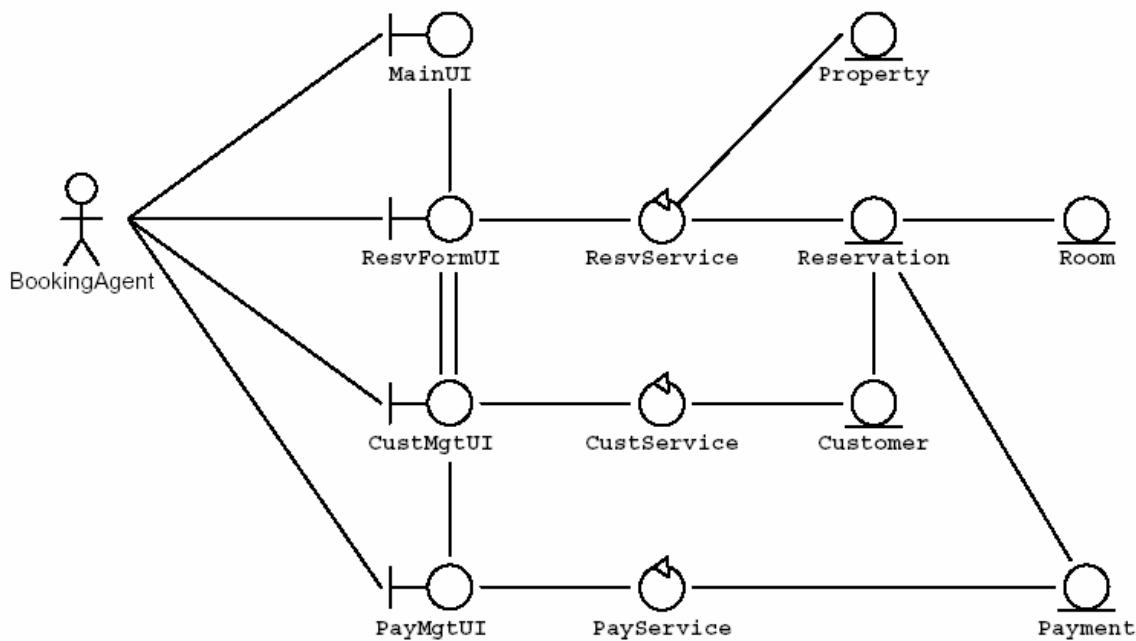


솔루션 모델을 생성하기 위해서는 디자인 모델이 반드시 필요합니다.

다음은 호텔 예약 시스템의 디자인 모델입니다.

[[이미지](#)]

n 호텔예약시스템의 디자인 모델

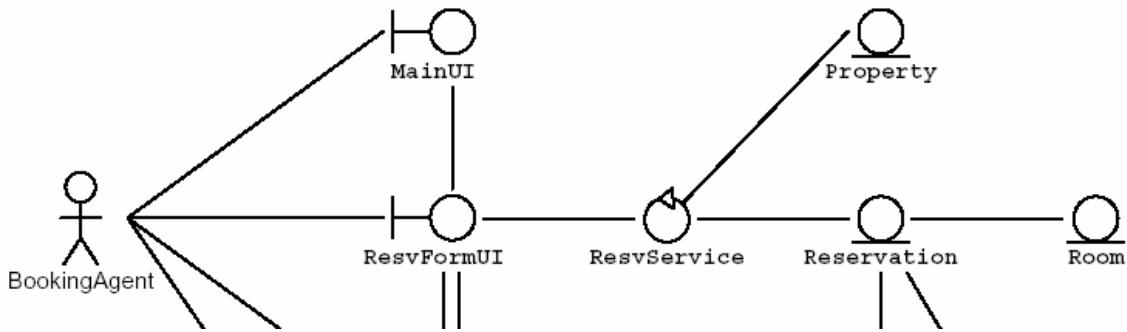


2) 호텔 예약 시스템의 디자인 모델 파악

(1) <예약>에 관한 디자인 모델

[이미지]

n 예약에 관한 디자인 모델



BookingAgent는 호텔 내 예약 담당 직원입니다.

이 직원이 시스템을 사용할 때 처음 접하는 화면이 **MainUI**입니다.

초기 화면인 **MainUI**로부터 **ResvFormUI**로 넘어가면 예약을 할 수 있는 화면이 보입니다. 이것은 **PAC 패턴**으로 구성되어 있습니다.

PAC 패턴이란 어떤 UI 컴포넌트가 **Presentation, Abstraction, Control** 컴포넌트로 구성되어 있는 것을 말합니다.

따라서 **ResvFormUI**는 **Presentation** 컴포넌트이고 **ResvService**가 **Control**, **Reservation**은 **Abstraction** 역할을 하는 컴포넌트가 됩니다.

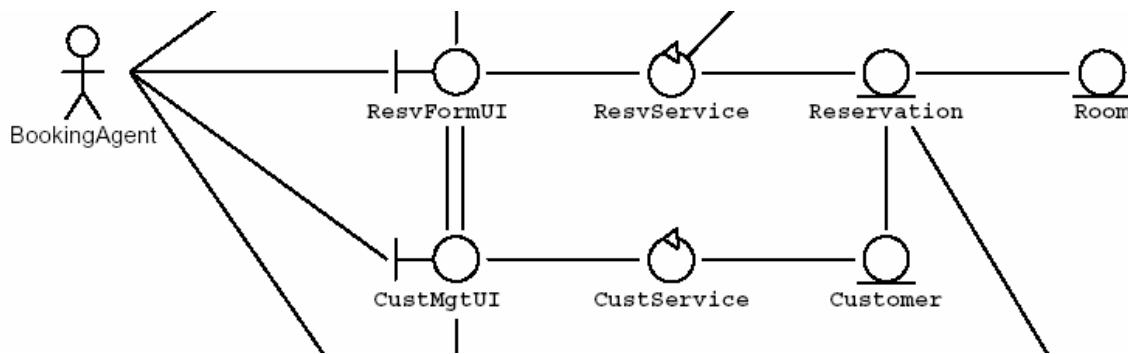
예를 들어, **BookingAgent**가 새 예약을 하나 하고자 한다면

- | 화면에 새 예약 버튼을 누르게 되고 ↳ **ResvFormUI**
- | 누가 어떤 객실을 언제 사용할 것인지에 대한 정보가 ↳ **Reservation**
- | 만들어지면서 이것이 새 예약으로 처리되어집니다 ↳ **ResvService**

(2) <고객관리>에 관한 디자인 모델

[이미지]

n 고객관리에 관한 디자인 모델



예약을 성립 시키기 위해서는 고객의 정보를 알아야 합니다.

따라서 **Reservation**은 **Customer**라는 **Abstraction** 컴포넌트와 관계가 있습니다.

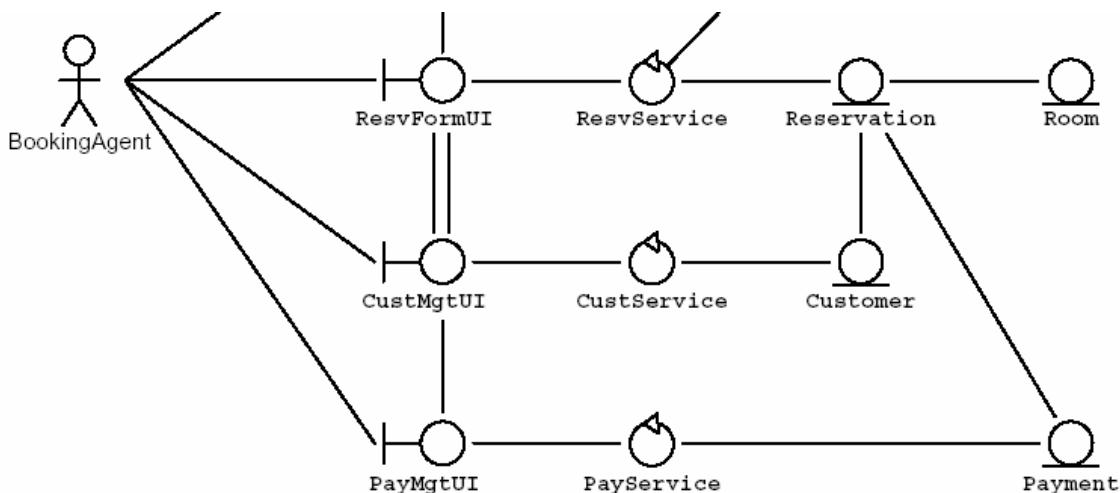
만약 이 고객이 신규 고객이라면 **BookingAgent**는

- | 고객관리 화면으로 들어가서 고객 등록 버튼을 누르고 ↳ **CustMgtUI**
- | 고객의 여러 정보를 입력한 뒤 ↳ **Customer**
- | 새 고객으로 등록시킵니다. ↳ **CustService**

(3) <결제>에 관한 디자인 모델

[이미지]

n 결제에 관한 디자인 모델



예약을 위해서는 결제 시스템도 거쳐야 합니다.

BookingAgent는 새 예약에 대한 결제를 위해

- | 결제 화면으로 들어가서 ↳ **PayMgtUI**
- | 어떻게 얼마를 결제할 것인지를 입력하고 ↳ **Payment**
- | 결제가 처리되도록 합니다 ↳ **PayService**

이제 간단하게나마 예약에 관한 디자인 모델을 살펴 보았습니다.

이것은 이미 [단원 4. 기능적 요구 사항 설계]에서 자세히 배운 내용들입니다.

그렇다면 호텔 예약 시스템의 아키텍쳐 모델에 이 디자인 모델만 융합 시킨다면 솔루션 모델은 완성될 것입니다.

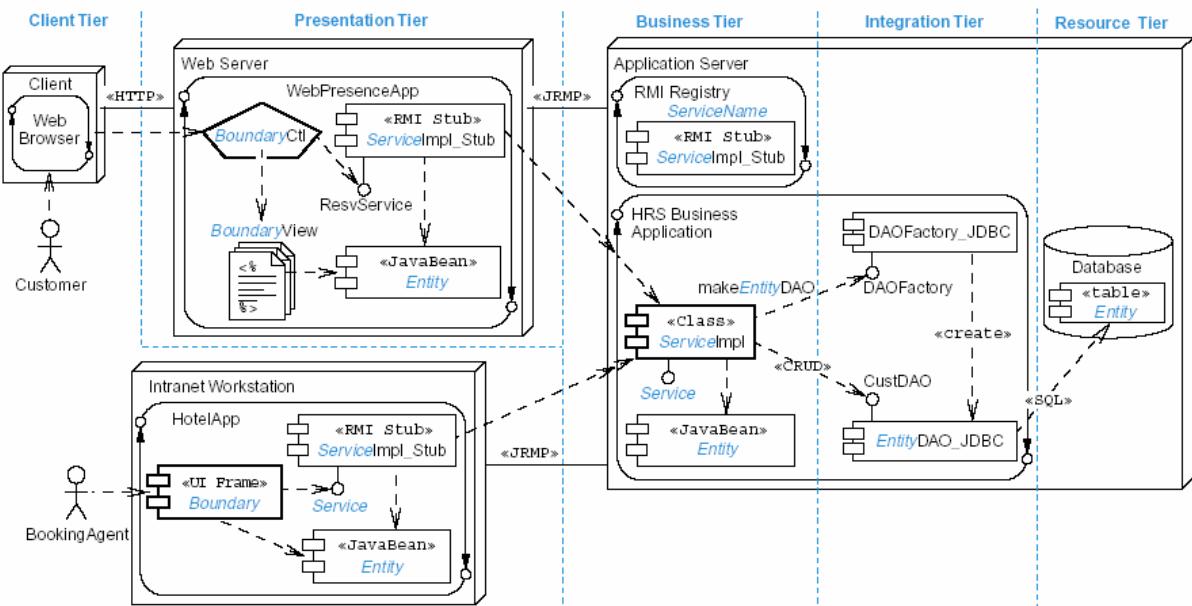
다음은 아키텍쳐 모델을 리뷰한 것입니다.

3) 호텔 예약 시스템의 아키텍쳐 모델 파악

호텔 예약 시스템의 아키텍쳐 모델의 여러 산출물 중에 아키텍쳐 템플릿은 다음과 같습니다.

[이미지]

n 호텔 예약 시스템의 아키텍쳐 템플릿



이것은 **BookingAgent** 뿐만 아니라 **Customer**까지도 이 시스템에 접근했을 때의 구조를 전체적으로 보여 주고 있습니다. **Customer**가 웹으로 이 시스템에 접근하는 것은 이 모듈의 [3. Web UI 어플리케이션을 위한 솔루션 모델]에서 다를 것입니다.

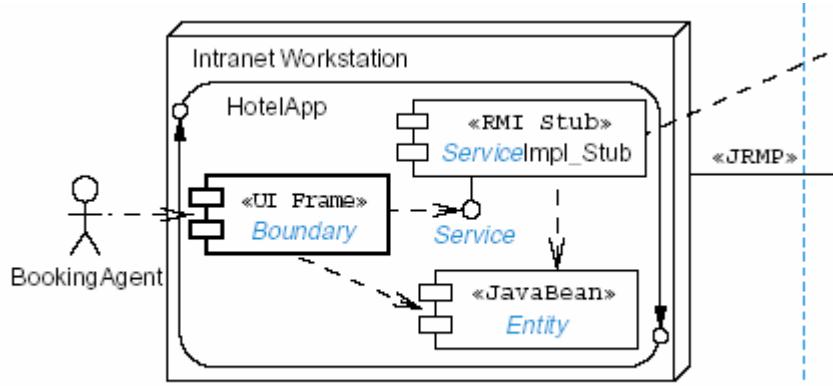
BookingAgent는 **GUI**를 통해 시스템에 접근하고 있으므로 클라이언트 티어에 아키텍쳐 템플릿을 구성합니다.

(1) Client Tier의 파악

위 전체 그림의 왼쪽 하단만 확대하면 다음과 같은 **Client Tier**의 **GUI** 어플리케이션 아키텍쳐 템플릿을 볼 수 있습니다.

[이미지]

n GUI 어플리케이션의 Client Tier



BookingAgent는 호텔 내 **Intranet Workstation**을 통해 호텔 예약 시스템에 접근합니다.

Boundary(바운더리) 컴포넌트는 화면을 구성하고 있는 컴포넌트들을 통칭하는 것으로 여기에는 초기 화면, 예약 화면, 고객관리 화면, 결제 화면 (**MainUI**, **ResvFormUI**, **CustMgtUI**, **PayMgtUI**) 등이 포함 될 것입니다.

바운더리 컴포넌트를 통해 어떤 서비스를 요청하면 그것은 **Service(서비스)** 컴포넌트에 전달되고 처리되어집니다.

그런데, 이 서비스 컴포넌트의 스테레오 타입이 <<**RMI Stub**>>으로 되어 있는 것에 주목하셔야 합니다.

이것은 서비스 구현 컴포넌트가 이 웍스테이션에 존재하지 않고 **RMI** 원격 객체로 존재한다는 것을 의미합니다.

따라서 비즈니스 티어를 참조해야 실제 서비스 컴포넌트가 어떻게 구성되어 있는지를 자세히 파악할 수 있을 것입니다.

RMI 프로토콜은 **JRMP**라고 했습니다. 이 프로토콜을 통해서 파라미터가 전달 될 때는 두 가지 경우로 나뉩니다.

파라미터가 **Primitive data**인 경우는 그 값이 직접 전달됩니다.

파라미터가 **Reference data**인 경우는 **Serialization(직렬화)**됩니다.

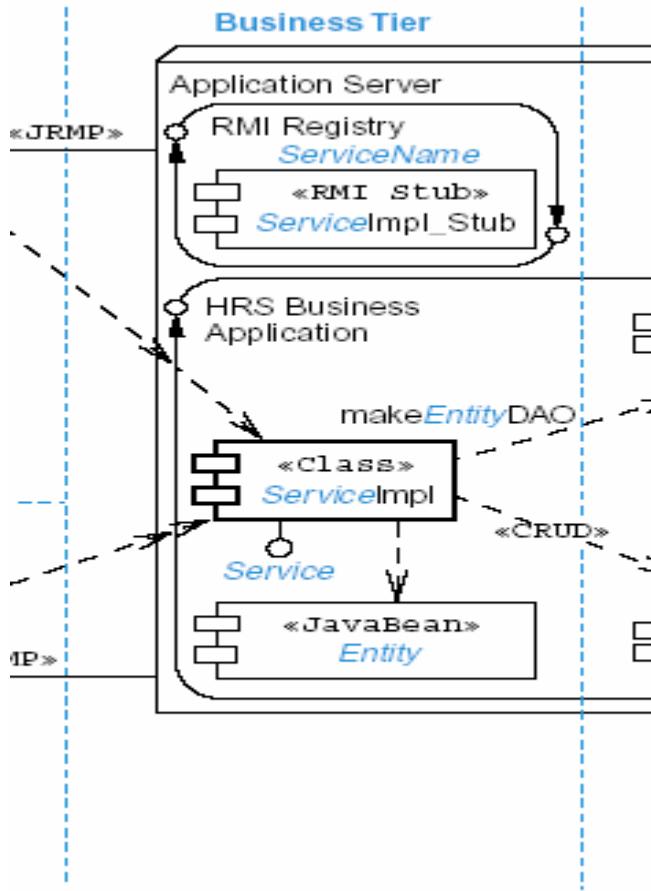
예약에 관한 서비스를 요청했을 때는 어떤 고객이 어떤 객실을 언제 예약 했는지에 관한 ‘정보’가 전달되어야 하는데, 바로 이 정보가 **Entity(엔티티)** 컴포넌트가 되는 것입니다. 이것은 직렬화 되어서 비즈니스 티어의 어플리케이션으로 전달되게 됩니다.

(2) Business Tier의 파악

호텔 예약 시스템의 비즈니스 티어를 확대해 보도록 하겠습니다.

[이미지]

n 호텔 예약 시스템의 Business Tier



비즈니스 티어는 **RMI**기술을 사용하고 있습니다.

1. **RMI** 레지스트리를 통해 서비스 구현 객체를 등록 시키고
2. 클라이언트 티어의 바운더리 컴포넌트가 이 서비스 스텁 객체를 찾고
3. 스텁 객체를 통해 원하는 서비스 메소드를 호출하면 이것이 실제 구현 객체로 전달되어 처리되어집니다.

ResvService라는 디자인 컴포넌트는 바로 이 **RMI** 구현 객체가 될 것입니다.

그런데 예약 정보는 저장되어져야 합니다.

따라서 비즈니스 티어의 서비스 컴포넌트는 데이터 스토리지에 이 정보를 저장하는 기능을 가지고 있어야 합니다.

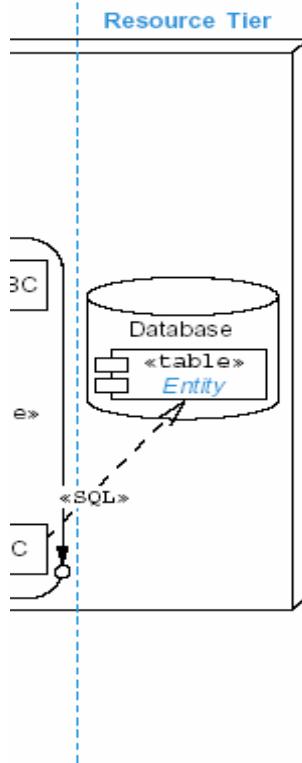
그렇다면 이제 **Resource tier**를 살펴봐야 합니다.

(3) Resource Tier의 파악

호텔 예약 시스템의 **Resource** 티어를 확대해 보도록 하겠습니다.

[이미지]

n 호텔 예약 시스템의 Resource Tier



리소스 티어는 간단하게 표현되어 있습니다.

많은 어플리케이션에서 **Data Storage**로 **RDBMS**(관계형 데이터베이스 시스템)을 사용하고 있습니다.

호텔 예약 시스템도 **Database** 시스템을 선택했습니다.

이 데이터베이스에는 저장하고자 하는 정보가 테이블 단위로 취급됩니다.

호텔 예약 시스템에서의 정보는 **Reservation**, **Customer**, **Room** 등의 엔티티 컴포넌트로 구현되어 있습니다.

따라서 이 엔티티 컴포넌트들이 테이블화 되어져야 합니다.

구성된 테이블의 한 행은 엔티티 컴포넌트의 인스턴스(객체) 하나에 매핑됩니다.

객체를 저장, 삭제, 읽기, 변경(**CRUD**) 하기 위해서는 데이터베이스 시스템에 **CRUD** 오ペ레이션을 명령하는 어플리케이션이 필요합니다.

만약 이 오ペ레이션들을 비즈니스 티어의 서비스 컴포넌트에 둘다면 데이터베이스 시스템이 변경될 때마다 서비스 컴포넌트의 변경도 불가피해 질 것입니다.

어플리케이션의 유연성을 위해서 우리는 **CRUD** 기능을 하는 컴포넌트를 별도로 구성할 것입니다. 이것을 **Integration Tier**라고 합니다.

Key Point

n 엔티티의 테이블화

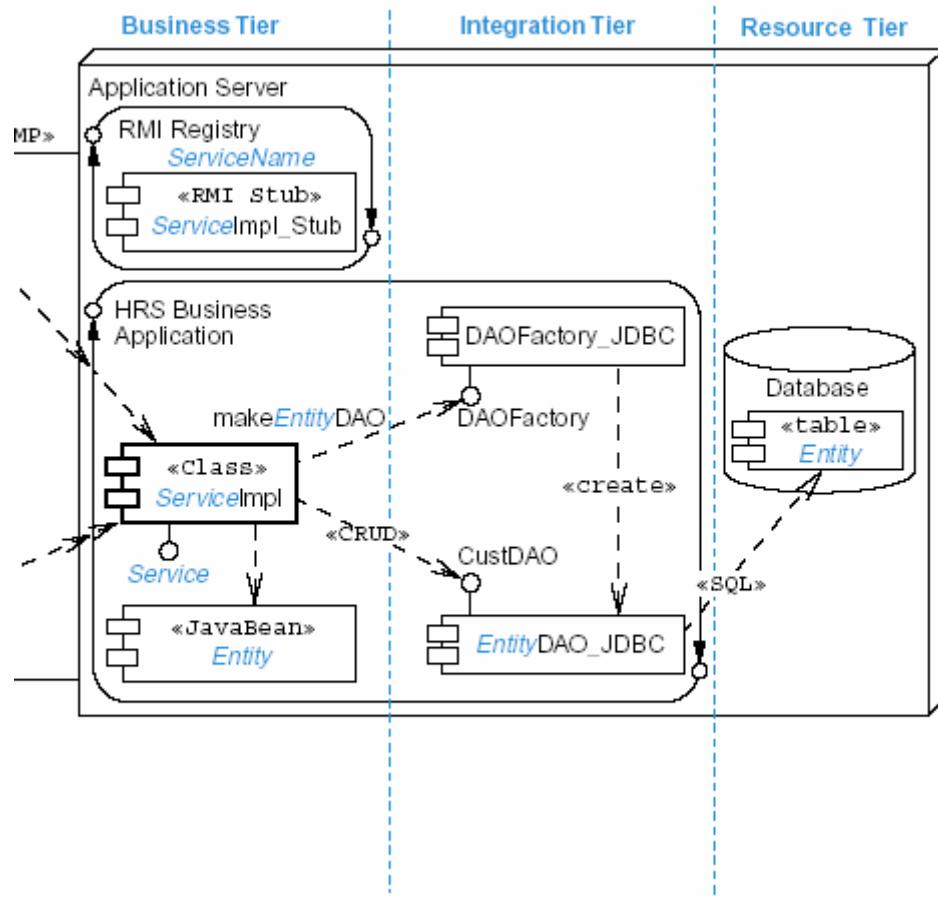
비즈니스 티어에서의 엔티티 컴포넌트들이 데이터베이스 시스템에서 테이블화 되어져야 합니다.

(4) Integration Tier의 파악

호텔 예약 시스템의 **Integration** 티어를 확대해 보도록 하겠습니다.

[이미지]

n 호텔 예약 시스템의 Integration Tier



Integration Tier에서의 기술은 **JDBC**를 사용하는 것입니다.

JDBC는 Java 어플리케이션이 데이터베이스 시스템에 쉽게 연결될 수 있도록 해주는 컴포넌트 Set입니다.

JDBC API를 사용하여 **CRUD(Create,Read,Update,Delete)**를 하는 컴포넌트는 **EntityDAO_JDBC**입니다.

이것은 각 엔티티 컴포넌트에 하나씩 대응되도록 만들어진 컴포넌트입니다.

Entity가 많으면 이 **DAO** 컴포넌트도 많아지므로 일관적인 관리를 위해서 **Factory** 패턴을 적용했습니다.

DAOFactor가 각 **DAO** 컴포넌트의 생성과 접근을 가능하게 하는 인터페이스입니다. 따라서 비즈니스 티어의 서비스 컴포넌트는 **CRUD** 기능을 직접 구현하지 않아도 되므로 훨씬 유연한 프로그램이 될 것입니다.

Key Point

- DAO 컴포넌트는 엔티티 컴포넌트에 하나씩 대응 되도록 만듭니다.

4) 디자인 모델 + 아키텍쳐 모델 = 솔루션 모델

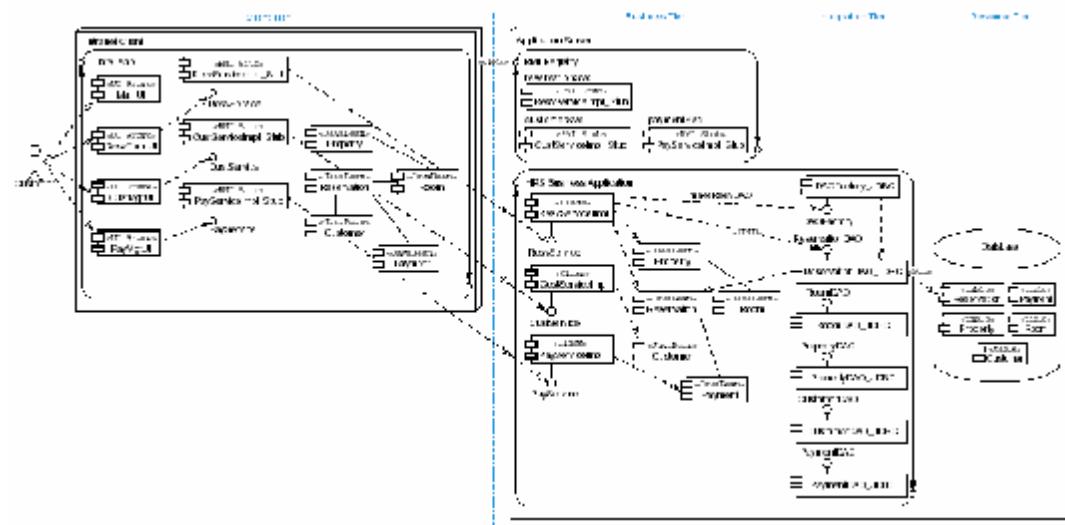
이제 디자인 모델과 아키텍쳐 모델을 융합한 솔루션 모델을 생성합니다.

유즈케이스가 많을수록 솔루션 모델의 크기가 커집니다.

작성한 아키텍쳐 템플릿은 컴포넌트 타입으로 표현되어 있습니다. 이것을 디자인 모델의 실제 컴포넌트 이름으로 대치합니다.

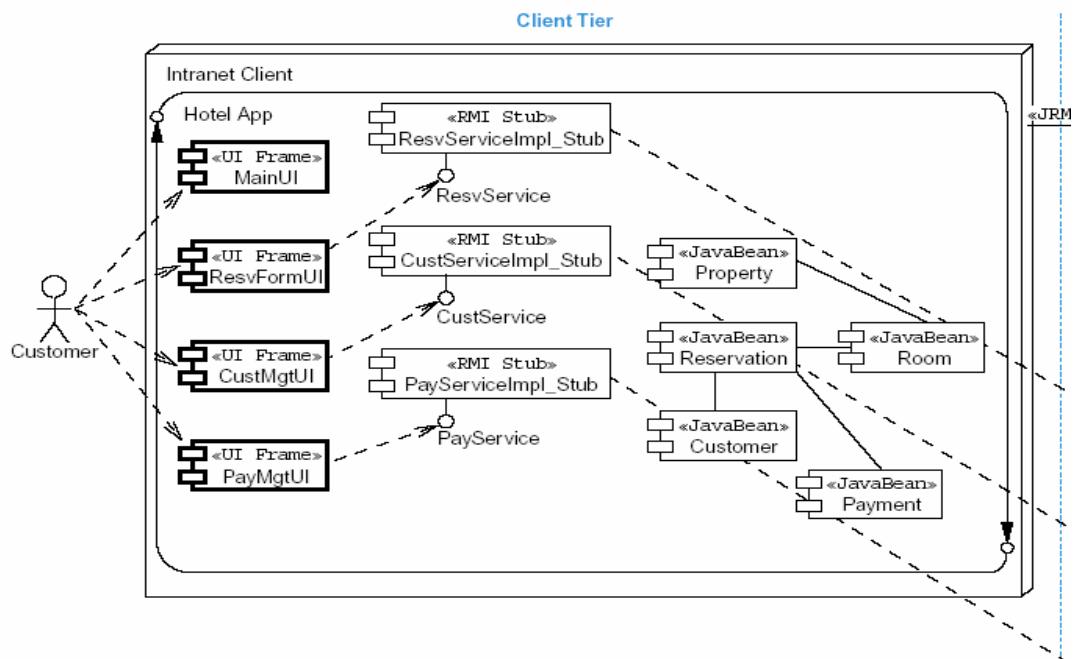
이미지

- 호텔 예약 시스템의 솔루션 모델

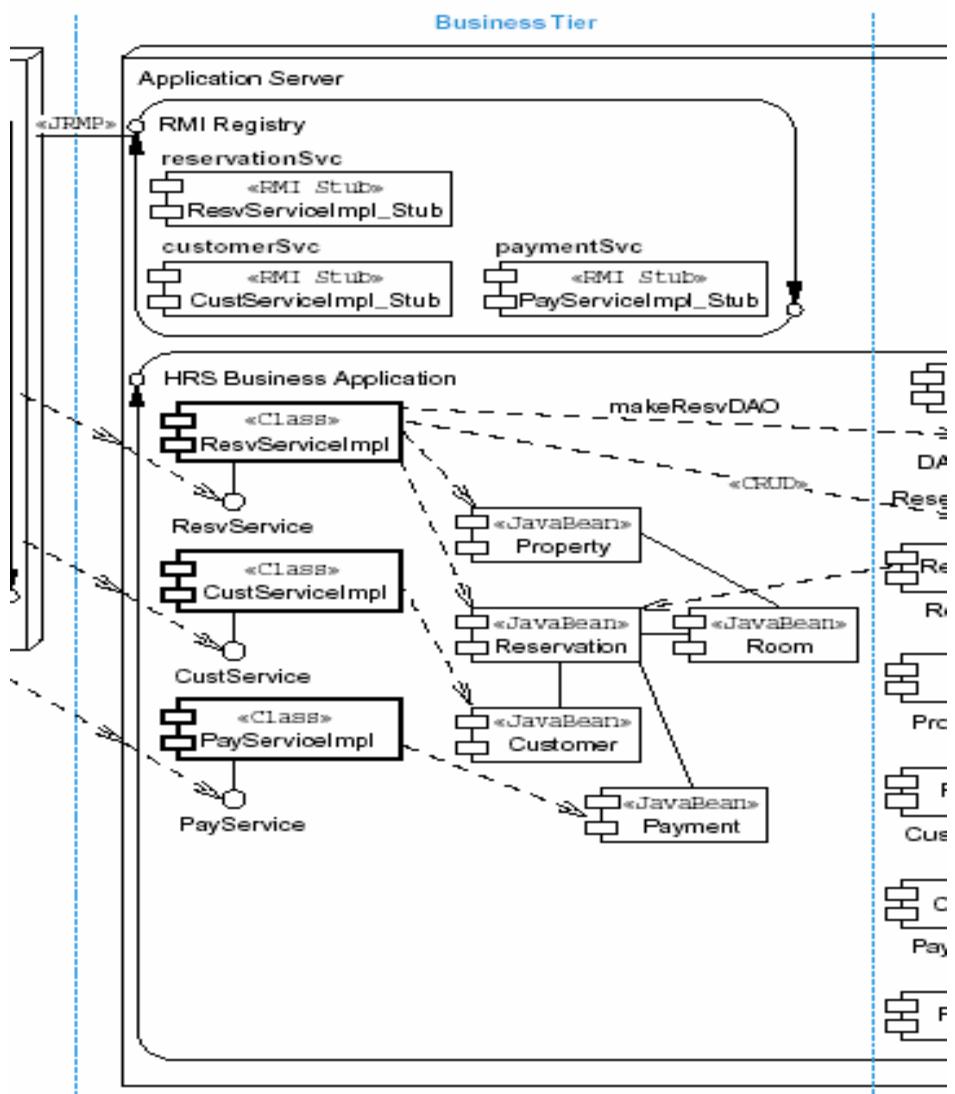


<이미지 설명>

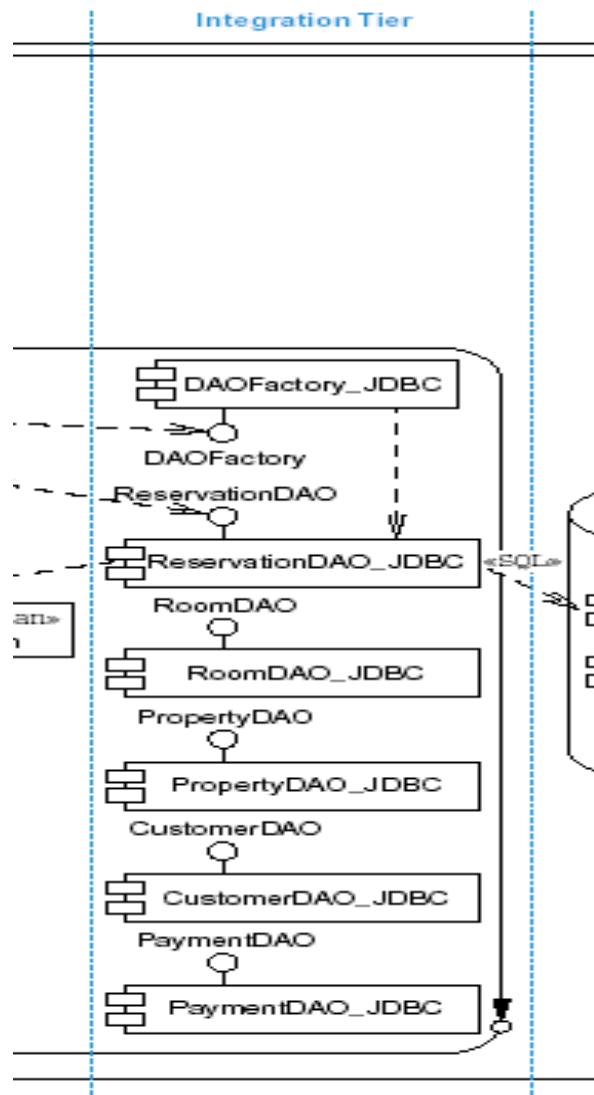
1. Client tier는 다음 그림으로 확대되어 보임



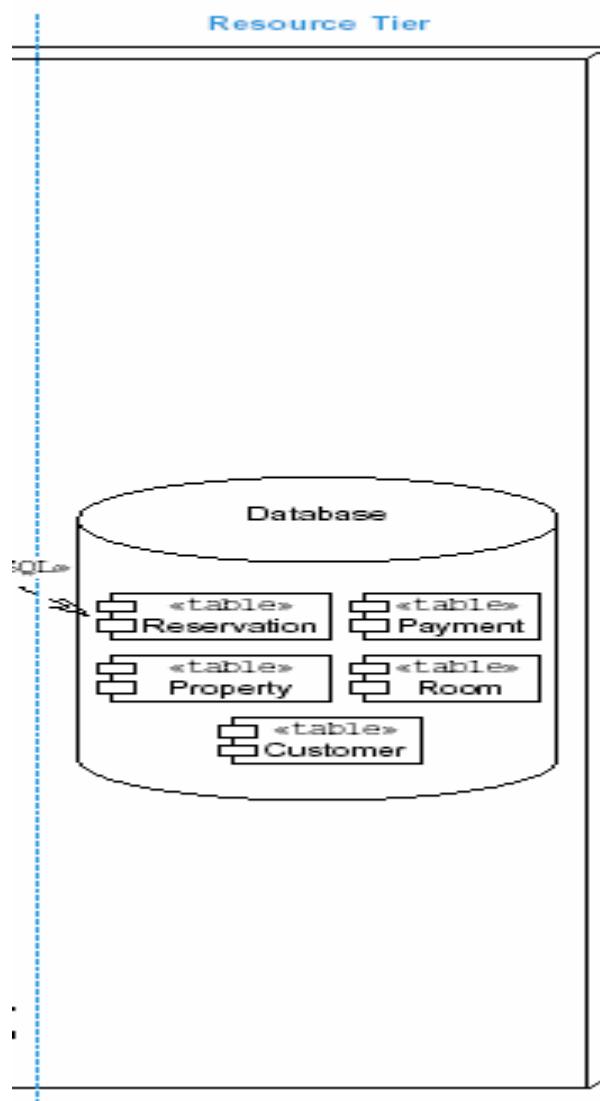
2. Business tier는 다음 그림으로 확대되어 보임



3. Integration tier는 다음 그림으로 확대되어 보임



4. Resource tier는 다음 그림으로 확대되어 보임



3. Web UI 어플리케이션을 위한 솔루션 모델

1) 개요

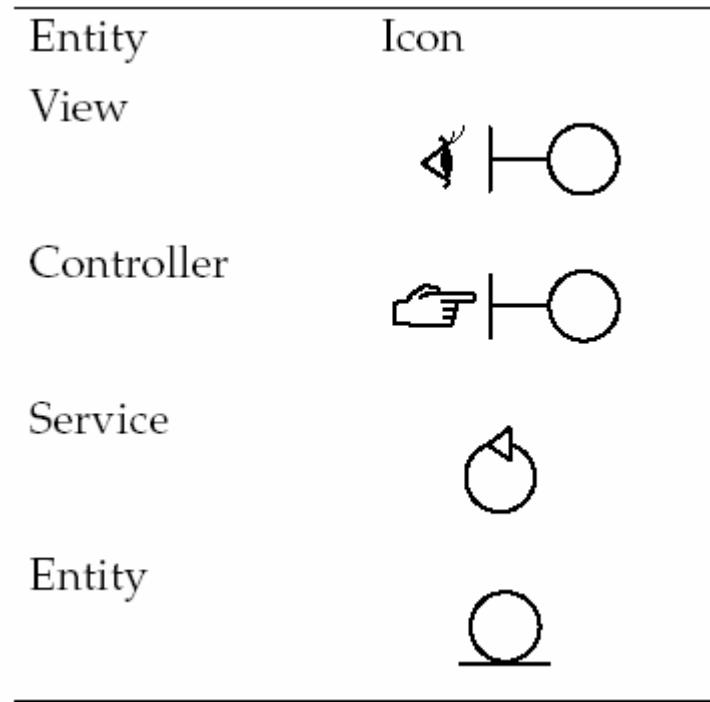
Web UI 어플리케이션의 디자인 모델은 GUI 디자인 모델과 동등하지 않습니다.

GUI 디자인 컴포넌트의 **Boundary component**는 **View** 와 **Controller** 컴포넌트로 세 분화 되어 표현되어 집니다.

따라서 **Web UI** 어플리케이션에서 사용하는 디자인 컴포넌트 유형은 다음과 같습니다.

[이미지]

- Web UI 디자인 컴포넌트의 유형

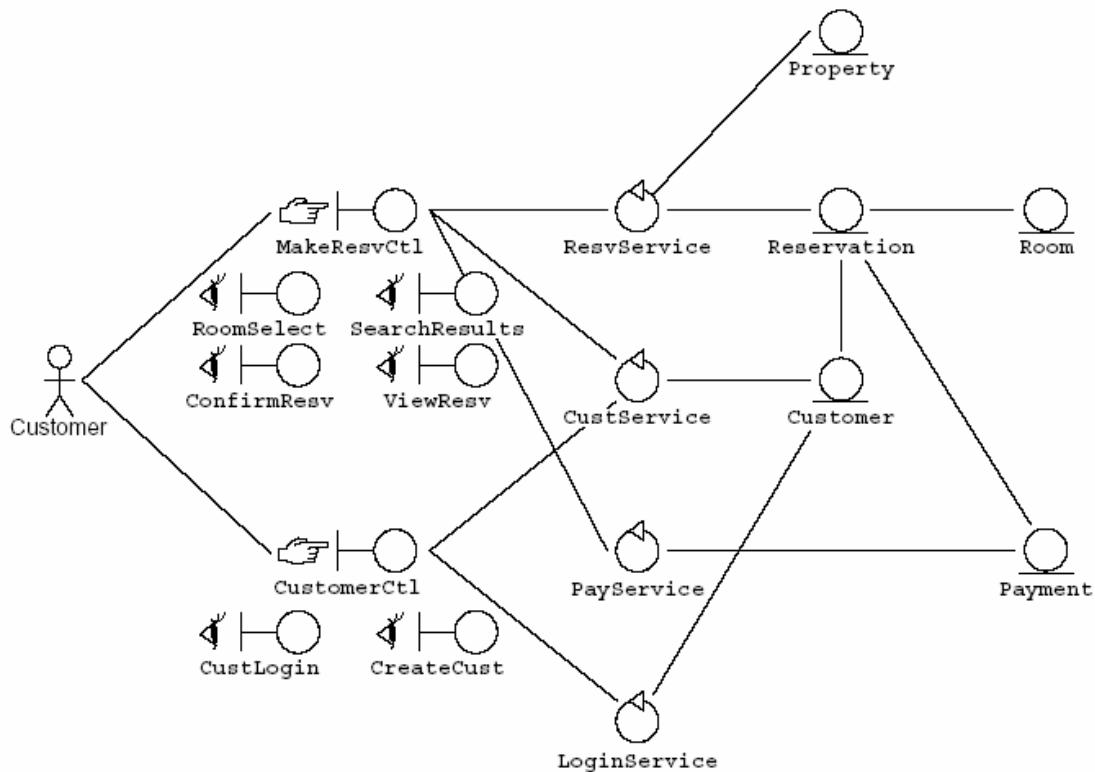


그림에서 보는 바와 같이 **View** 컴포넌트는 ‘눈’ 아이콘과 같이 그림으로써 오로지 시각적인 표현 컴포넌트임을 강조하는 반면 **Controller** 컴포넌트는 ‘손가락’ 아이콘과 같이 그림으로써 화면상에 액션을 취했을 때를 표현하고 있습니다.

다음은 호텔 예약 시스템의 **Web UI** 디자인 모델입니다.

[이미지](#)

n 호텔예약시스템의 Web UI 디자인 모델



Key Point

n GUI 어플리케이션의 Boundary Component

GUI 어플리케이션의 **Boundary Component**는 **Web UI** 어플리케이션에서의 **View Component**와 **Controller Component**를 합한 것입니다.

2) 호텔예약시스템의 디자인모델 파악

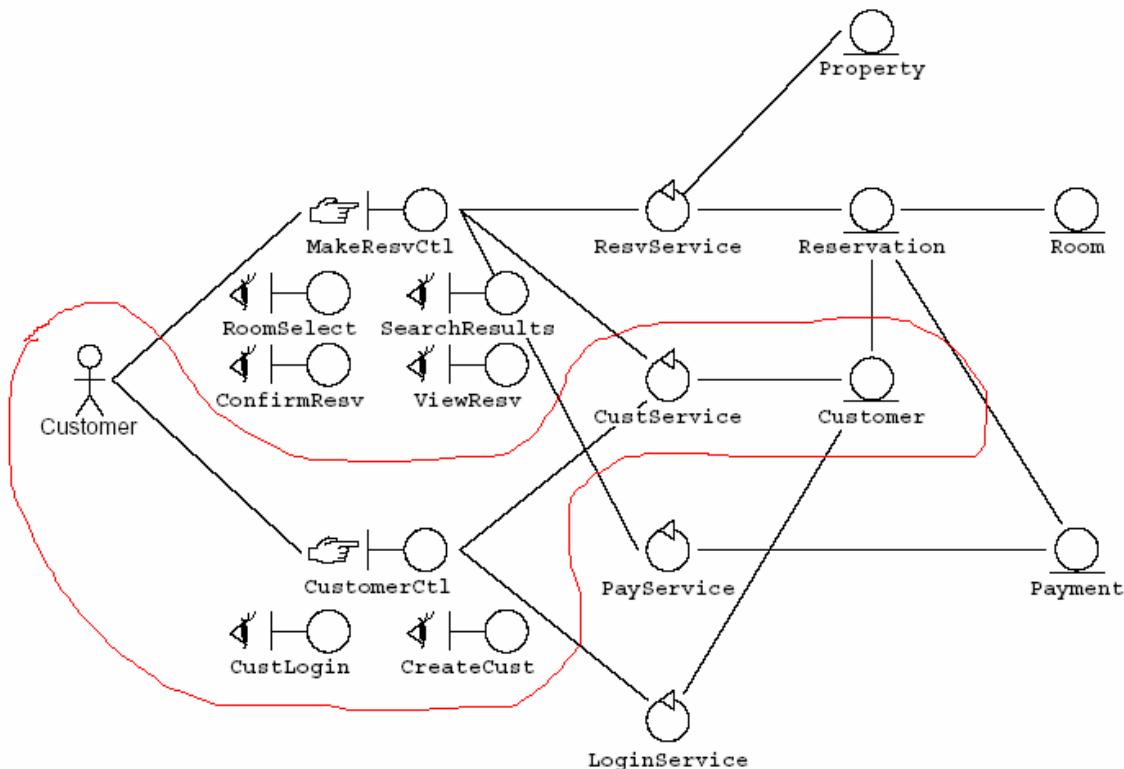
(1) <로그인>에 관한 디자인 모델

GUI 어플리케이션이 주로 호텔 내 직원이 사용하는 시스템이라면 **Web UI** 어플리케이션은 일반 고객이 인터넷으로 사용하는 시스템입니다.

따라서 액터는 **Customer**로 표현을 하는 것이 일반적이며 고객이 이 시스템을 사용할 때의 ‘인증’을 위해서 로그인 유즈케이스를 추가하였습니다.

이미지

n 로그인 부분의 디자인 모델



고객은 인터넷으로 호텔예약 시스템에 접속합니다.

고객은 회원 관리 화면을 선택해서 볼 수 있는데, 신규회원 이라면 등록화면을, 기존 회원이라면 로그인 화면을 보게 될 것입니다.

회원 등록 화면에서는 고객의 여러 정보들을 입력할 수 있는 **Form**이 제공되고 **CreateCust**

가입 버튼을 누르거나 하는 사용자 액션을 취하게 되면 **CustomerCtl**

입력된 정보를 얻어 **Customer**

가입 처리를 하게 됩니다 **CustService**

회원 로그인화면에서는 보통 고객의 **ID**와 **PW**를 입력할 수 있는 **Form**이 제공되고 **CustLogin**

로그인 버튼을 누르거나 하는 사용자 액션을 취하면 **CustomerCtl**

시스템이 알고 있는 이 고객의 기존 정보를 확인하여 **Customer**

인증처리를 하게 됩니다 **CustService**

GUI 어플리케이션과의 차이는 바운더리 컴포넌트가 **CustomerCtl**, **CustLogin**, **CreateCust**로 나뉘어 있다는 것입니다.

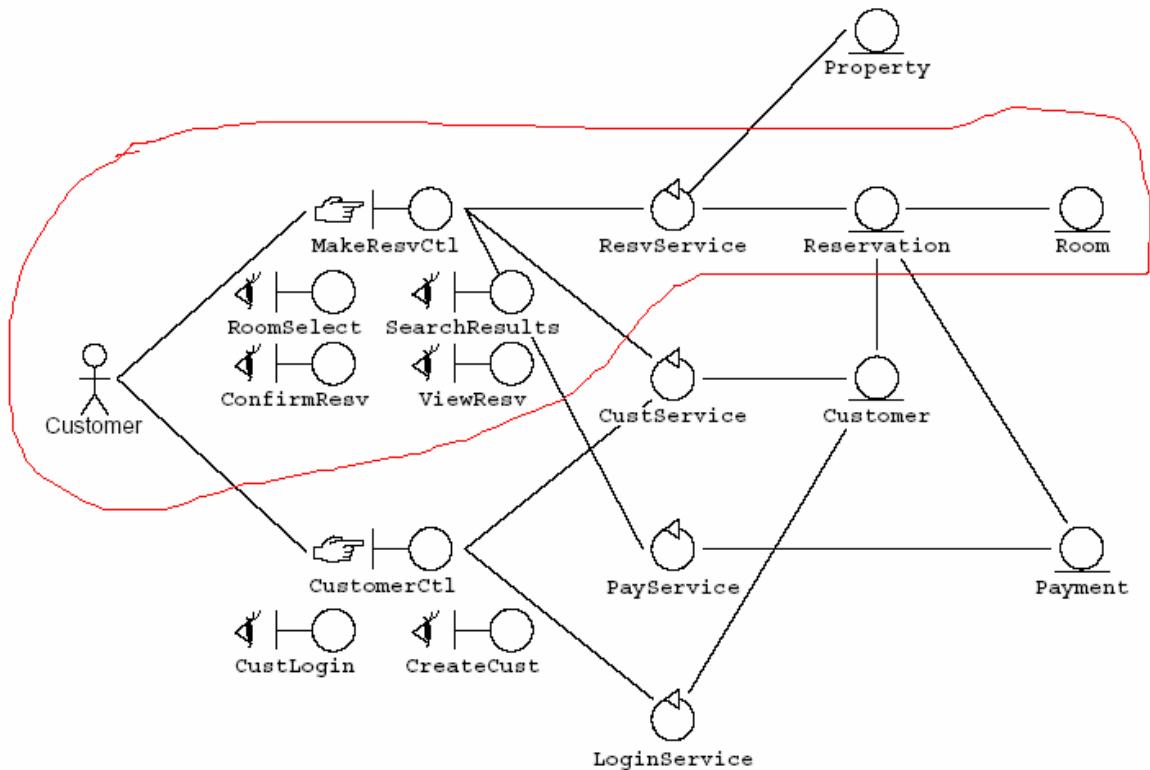
CustomerCtl이 Controller컴포넌트가 되고 **CustLogin**, **CreateCust**가 View 컴포넌트가 됩니다.

(2) <예약>에 관한 디자인 모델

고객 인증 절차가 끝났으면 고객은 ‘예약’ 유즈케이스를 사용할 수 있습니다.

[이미지]

n 예약 부분의 디자인 모델



고객은 예약하고자 하는 호텔 내 시설을 검색할 수 있습니다.

⇒ RoomSelect

만약 이미 예약 해 둔 사항이 있다면 그 내용을 다시 볼 수도 있습니다

⇒ ViewResv

이 예약을 승인할 수 있는 화면을 볼 수도 있습니다.

⇒ ConfirmResv

이런 View화면들로 예약에 관한 여러 유즈케이스를 요청하면

⇒ MakeResvCtl

필요한 정보, 예를 들어 어떤 고객이 어떤 객실을 언제 예약하는지를 정리

하여 ⇒ Reservation

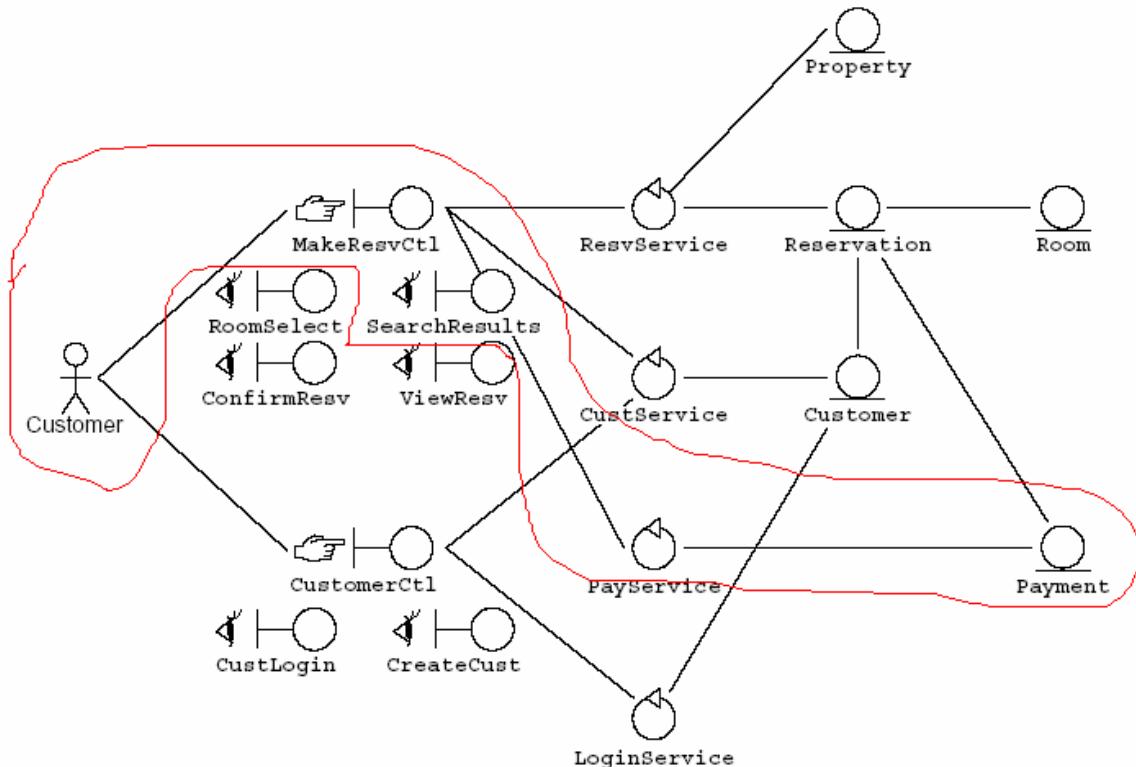
그것을 처리하게 됩니다 ⇒ ResvService

(3) <결제>에 관한 디자인 모델

예약에는 결제 시스템이 따르게 됩니다.

[이미지]

n 결제 부분의 디자인 모델



고객이 어떤 예약을 원하면 시스템은 결제 처리 서비스를 가동시켜야 합니다. ⇒ PayService

이때 누가 어떤 예약에 대해서 얼마를 결제하는지에 관한 정보가 필요합니다

다 **Payment**

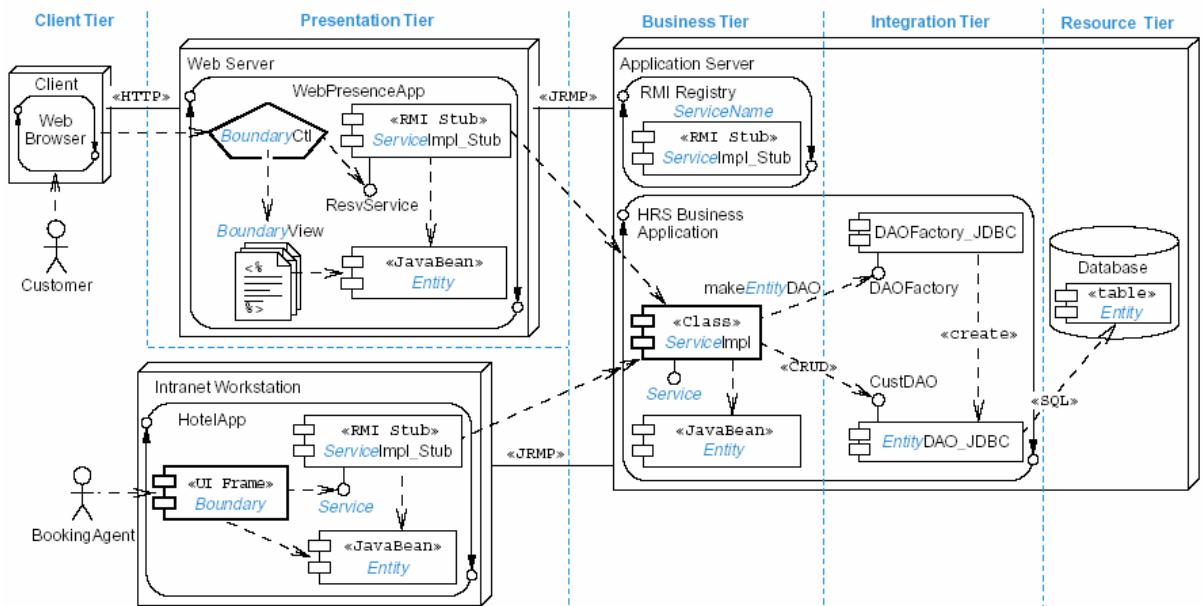
결제가 완료된 예약 처리 결과를 볼 수도 있습니다 **SearchResults**

3) 호텔예약시스템의 아키텍쳐 모델 파악

호텔 예약 시스템의 아키텍쳐 모델의 여러 산출물 중에 아키텍쳐 템플릿은 다음과 같습니다. 이것은 **GUI** 어플리케이션에서와 동일합니다.

[이미지]

■ 호텔 예약 시스템의 아키텍쳐 템플릿



Business Tier, Integration Tier, Resource Tier는 **GUI** 어플리케이션에서 이미 설명한 바 있으므로 **Client Tier**와 **Presentation Tier**에 대해서만 설명하겠습니다.

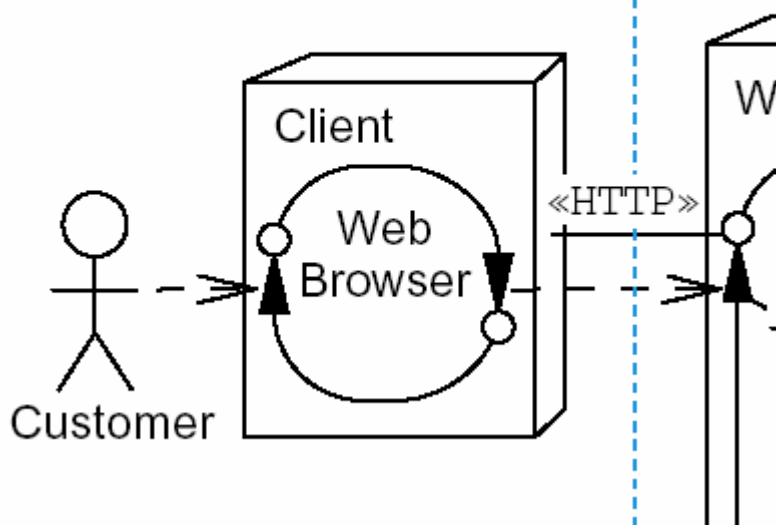
(1) Client Tier의 파악

고객은 웹 브라우저로 호텔 예약 시스템에 접속합니다.

[이미지]

■ 호텔예약시스템의 Client Tier의 아키텍쳐 템플릿

Client Tier



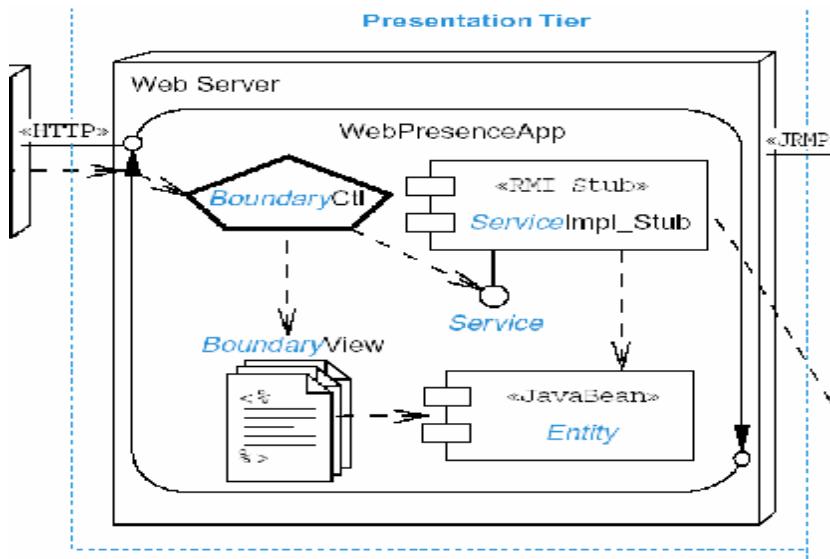
고객은 호텔예약시스템의 **Web UI**의 메인 화면으로부터 다양한 뷰를 선택해서 볼 수 있습니다. 여기서 소형 이벤트는 **JavaScript**로 처리 될 수 있고 대형 이벤트는 **Http Request** 형태로 **Web Server**로 전달될 것입니다.

(2) Presentation Tier의 파악

Web UI를 통한 다양한 뷰나 이벤트의 제공은 **Presentation Tier**에서 표현되어집니다.

[이미지]

n 호텔예약시스템의 **Presentation Tier**부분의 아키텍쳐 템플릿



웹 브라우저에서 발생한 대형이벤트는 **Http Request** 형태로 웹 서버로 전달되어 집니다.

웹 서버위의 바운더리 컴포넌트 중에 **Controller**에 해당하는 컴포넌트가 이벤트를 분석하고 이것을 처리할 서비스 컴포넌트를 찾게 됩니다.

서비스 컴포넌트는 **RMI** 원격 객체이므로 분산된 다른 어플리케이션으로 구축되어 있어 실제로는 원격 스텝 객체를 통해 요청을 전달하게 됩니다.

웹 서버위의 바운더리 컴포넌트 중에 **View**에 해당하는 컴포넌트는 고객에게 결과 화면을 보여주거나 다양한 유즈케이스 동안의 연속적인 화면을 보여주기도 합니다.

4) 디자인 모델 + 아키텍쳐 모델 = 솔루션 모델

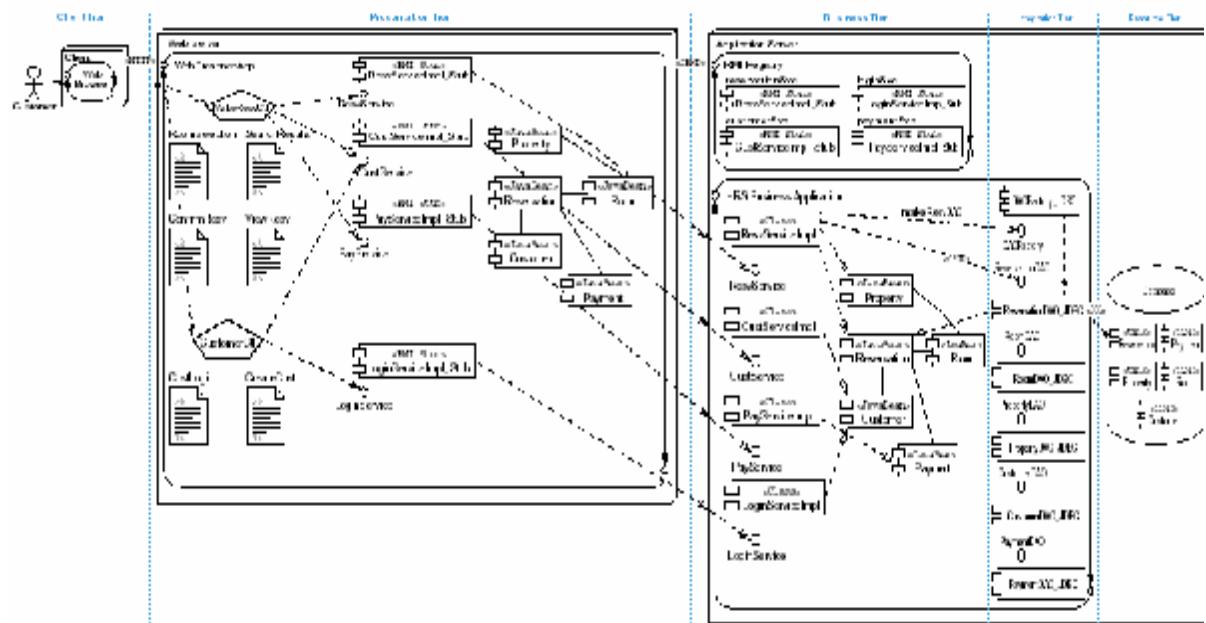
이제 디자인 모델과 아키텍쳐 모델을 융합한 솔루션 모델을 생성합니다.

이것도 **Client Tier**와 **Presentation Tier** 부분만 새로이 적용된 부분이고 나머지 **Tier**들은 **GUI** 어플리케이션의 솔루션 모델과 동일합니다.

작성한 아키텍쳐 템플릿은 컴포넌트 타입으로 표현되어 있습니다. 이것을 디자인 모델의 실제 컴포넌트 이름으로 대치합니다.

[이미지]

n 호텔 예약 시스템의 솔루션 모델



과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원6 : 솔루션 모델의 구축

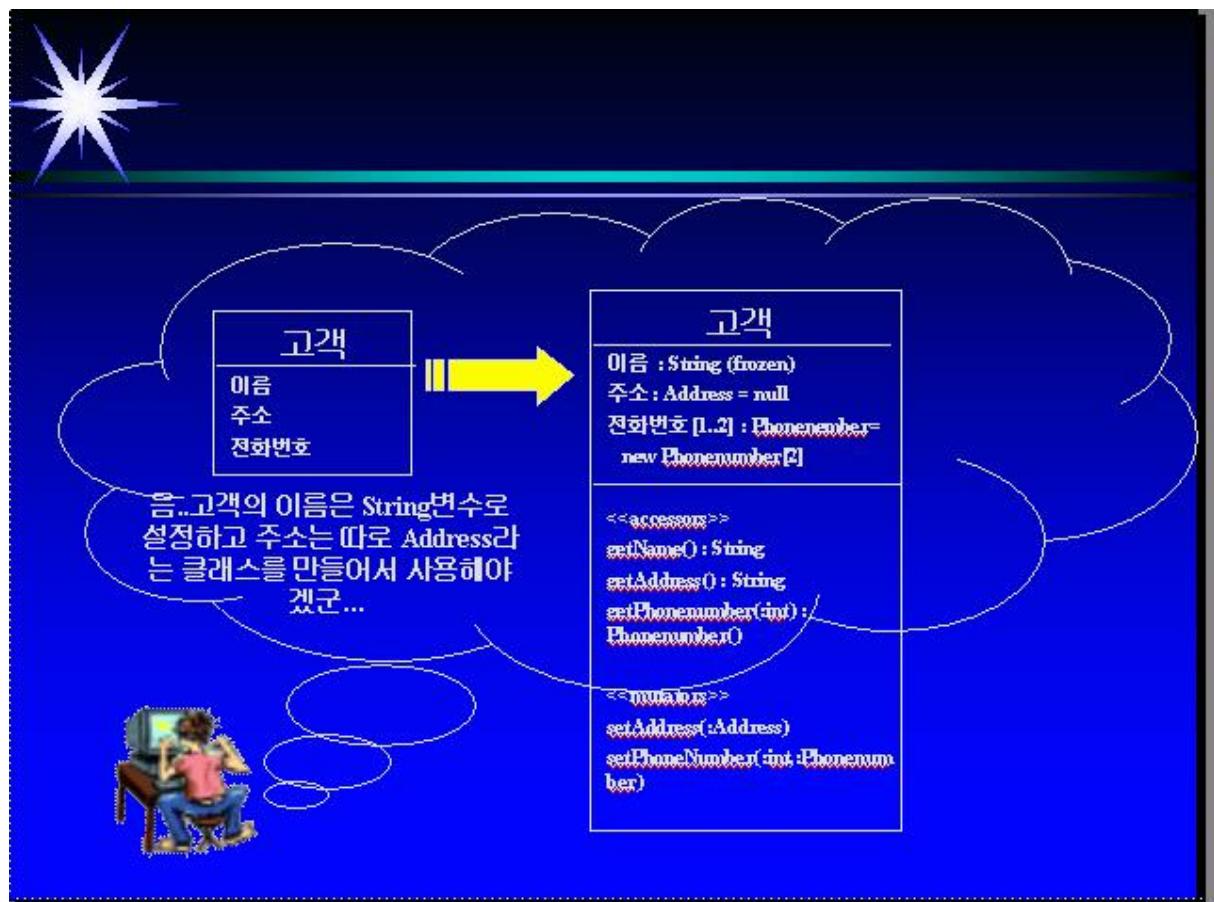
모듈 2 : 상세 Domain Model 구축

담당강사 : 전은수

■ 생각해봅시다 ■

이전 모듈(단원5.솔루션 모델의 구축- 모듈1.솔루션 모델 생성)에서는 실제 구현에 한 발 다가설 솔루션 모델을 생성했습니다. 그러나 이 솔루션 모델을 바로 코드화하기엔 아직 무리가 있습니다. 솔루션 모델은 시스템의 아키텍처와 잘 구성된 컴포넌트들이 융화되어 있긴 하지만 그 컴포넌트들의 상세 정보에 대해서는 아직 부족한 것이 많습니다. 상세 정보가 없는 솔루션 모델만으로 코드를 작성하는 것은 전체 설계도면만 보고 주먹구구식으로 집을 짓는 것과 다름 없습니다. 집을 지을 때 배선도면, 배관도면 등과 같은 여러 상세 시공도면이 필요하고 이 시공도면에는 구체적인 부자재의 종류나 이름까지 명시되어 있는 것처럼 소프트웨어 구축 시에도 이 단계에서 상세 도메인 모델을 구축하여야 합니다. 이미 [단원3.기능적 요구 사항분석 - 모듈4. **Problem Domain Model**생성]과정에서 도메인 모델이란 무엇이고 어떻게 생성하는지에 대해 배웠습니다. 그렇다면 이 도메인 모델을 상세화하는 구체적인 방법은 무엇일까요?

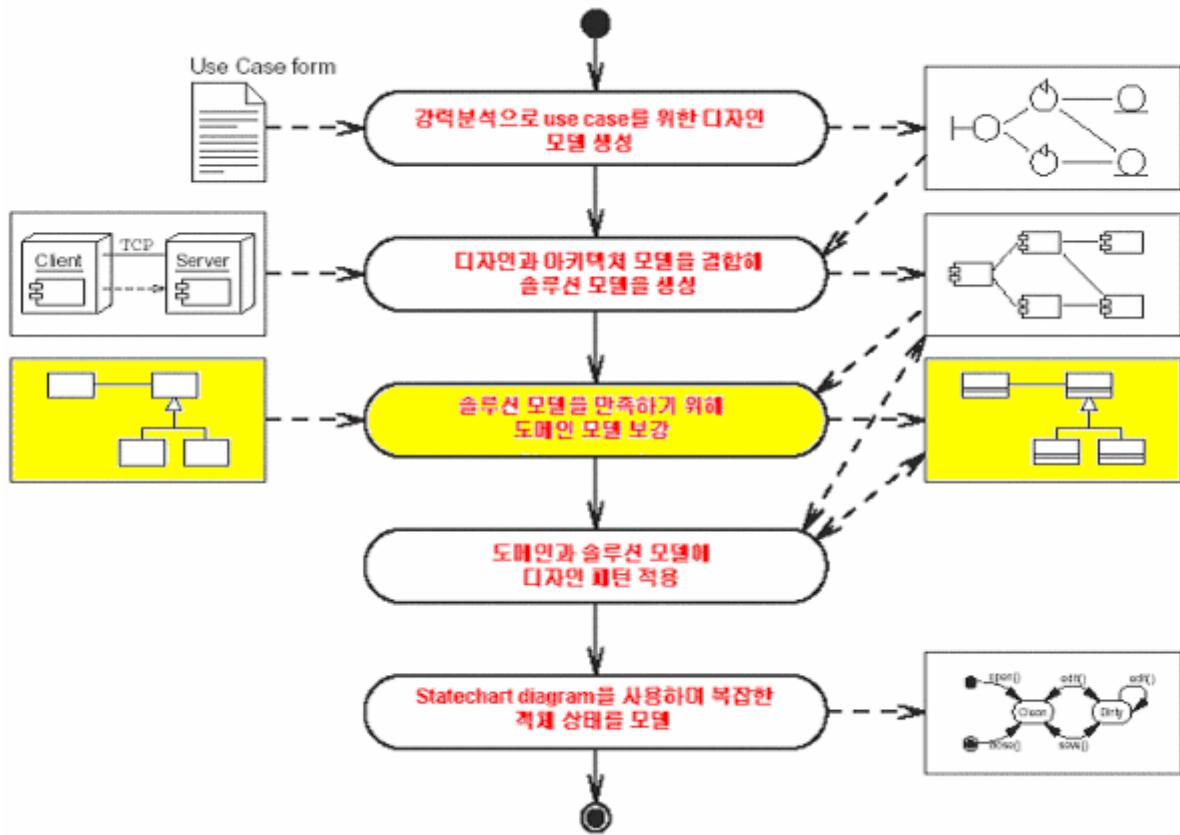
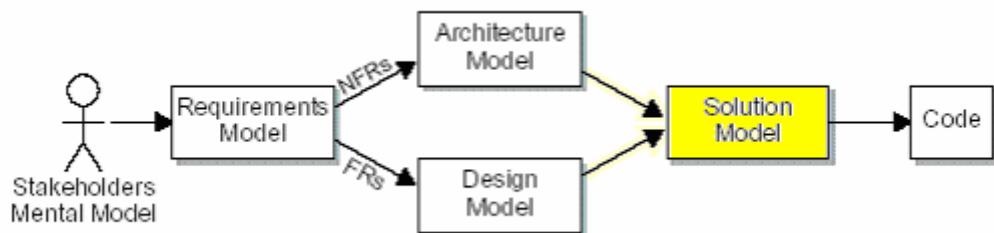
애니메이션



■ 학습하기 ■

[참고하세요]

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



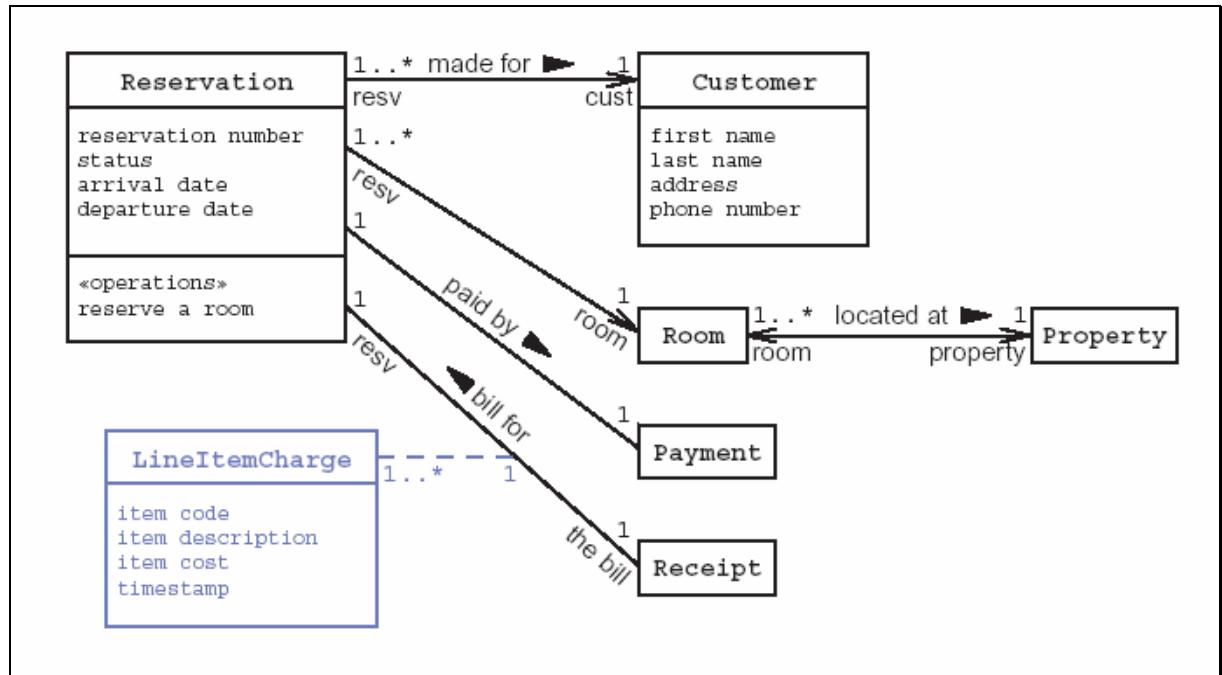
1. Domain Model에 변수 추가

1) 변수의 상세화

분석 단계에서 생성했던 도메인 모델은 **SRS**를 통해 **Key Abstraction**했던 것으로 다음과 같은 클래스 다이어그램으로 표현할 수 있습니다.

이미지

- 호텔예약 시스템의 클래스 다이어그램



그림에서 클래스의 이름은 표현되어 있지만 변수와 메소드에 대해서는 언급되지 않은 클래스도 볼 수 있습니다.

이것은 이 클래스에 대해 구체적이고 상세한 설계가 이뤄지지 않았다는 것입니다.

클래스는 이름만 가지고서 구현할 수 없습니다.

클래스의 구성 요소인 변수와 메소드를 가지고 있어야 합니다.

먼저 변수를 추가하기 위해서는 변수를 표현할 수 있는 여러 요소들에 대해서도 알아야 할 것입니다. 다음은 변수를 표현할 수 있는 요소들입니다.

(1) 변수의 Metadata

UML Class Diagram으로 표현할 수 있는 변수의 요소들은 다음과 같은 것들이 있습니다.

① Name

변수에는 반드시 이름이 있어야 합니다. 예를 들어 “**Reservation**” 클래스에서 “**reservation number**”라고 표현된 클래스의 속성이 변수의 이름이 되는

것입니다. 그러나 분석 단계에서의 “**reservation number**”는 설계단계에서는 정해진 프로그래밍 언어에 의존하여 표현되어야 합니다.
자바에서는 변수의 이름은 띄어쓰기가 없는 한단어로 표현되어야 합니다.
따라서 “**reservationNumber**” 와 같은 표현이 적당할 것입니다.

② **Visibility**

이것은 **accessibility(접근성)**라고 말하기도 합니다.
변수를 클래스 밖에서도 볼 수 있느냐는 것입니다. 다시 말해 이 변수를 다른 클래스에서도 사용할 수 있느냐는 것입니다.
변수는 데이터입니다. 어떤 데이터는 그 클래스 내부에서만 사용되어야 하며 또 어떤 데이터는 다른 모든 클래스에서도 사용될 수 있습니다.
이것을 변수에 표현할 수 있는데 다음과 같은 종류가 있습니다.

■ Table::UML Visibility Indicators

Visibility	Indicator
Public	+
Protected	#
Package private	~
Private	-

Public은 변수의 이름 앞에 + 기호로 표시하며 다른 어떤 클래스에서도 접근할 수 있다는 뜻을 가집니다.

Protected은 변수의 이름 앞에 # 기호로 표시하며 같은 패키지에 속해 있는 클래스나 상속 관계에 있는 클래스의 접근을 허락한다는 뜻입니다.

Package private은 변수의 이름 앞에 ~ 기호로 표시하며 이 클래스가 속해 있는 패키지 내의 다른 클래스에서만 접근할 수 있다는 뜻입니다.

Private은 변수의 이름 앞에 - 기호로 표시하며 이 클래스 내에서만 사용할 수 있다는 뜻입니다.

이렇게 하는 이유는 데이터를 보호하기 위함입니다.

중요한 데이터의 직접 접근을 차단하여 무분별한 변경을 막고자 할 때 적당히 취사선택하시면 될 것입니다.

③ **Type**

변수는 데이터라고 했습니다. 어떤 데이터든 타입이 있습니다.

타입이 없는 모호한 데이터는 사용할 수 없습니다.

프로그램 언어에 따라 지원하는 데이터 타입이 다른데, 대부분의 프로그램 언어에서는 기본 데이터형으로 **integers**, **floating point numbers**, **Boolean values** 등을 제공하고 특히 객체지향 프로그램에서는 **object reference**도 제공합니다.

④ Multiplicity

변수에는 특정 값이 얼마나 많이 존재 할 수 있는지를 표시할 수 있습니다.
기본적으로 아무것도 표시되어 있지 않으면 그 변수의 특정 값은 오직 하나
라는 것을 의미합니다.

⑤ Initial value

이 변수를 포함하는 클래스가 객체화 될 때 변수에 최초로 주어지는 특정
값을 변수의 초기값이라고 합니다.

⑥ Property

변수는 실행 시 여러 값으로 변경되거나 추가될 수 있습니다. 또는 어떤 값
으로도 변경되지 않게 할 수도 있습니다. 이것을 **property-string**으로 표현
할 수 있는데, **{changeable}**이라고 표현하면 변수값이 유동적이라는 것이고,
{addOnly}라고 표현하면 오직 하나의 값만 더 추가 할 수 있다는 것이며,
{frozen}이라고 표현하면 변경할 수 없음을 의미합니다.

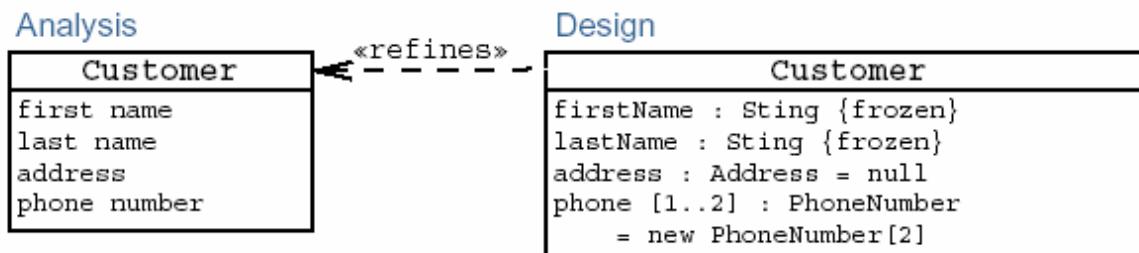
변수의 선언을 위한 **UML syntax**는 다음과 같습니다.

[visibility] name [multiplicity] [: type] [= init value] [{property- string}]

다음 그림은 **Customer** 클래스의 변수 선언부를 이 단계에서 보강된 내용으로 보여
줍니다.

이미지

I Example Refinement of Attributes



이 다이어그램에서 분석(**Analysis**)시의 **Customer** 클래스의 변수는 속성만 표현되어
있지만 설계(**Design**) 시에는 확정된 변수명으로 표현되어 있고 변수의 타입외의 여
러 요소들도 추가되어 있는 것을 볼 수 있습니다.

(2) 적당한 Data Type의 선택

앞서 **Customer** 클래스의 **firstName** 변수의 데이터 타입이 **String**으로 선언되어 있
는 것을 보셨습니다. **firstName**은 문자열로 표현되는 것이 가장 적당하기 때문입니다.

그렇다면 어떤 변수의 데이터 타입을 선택하기 위해서는 어떤 점을 고려해야 할까요?

| 얼마나 잘 표현하는가.

이것은 프로그래머가 의도하는 변수의 값을 얼마나 근접하게 표현할 수 있느냐의 문제입니다.

예를 들어 **firstName**을 숫자로 표현한다면 어떨까요?

어색할 뿐만 아니라 값을 표현하는데도 무리가 있습니다.

또 다른 예로, ‘나이’라는 변수의 데이터 타입은 어떤 것이 좋을까요?

나이는 숫자이므로 **integer** 타입으로 표현하는 것이 가장 적당할 것입니다.

| 변경시간이 얼마나 걸릴까.

어떤 데이터는 사용자 인터페이스에 표현되기 위해 그 표현이 변경되거나 **Data storage**에 저장되기 위해 표현이 변경될 경우가 있습니다.

설계 시에 이런 부분까지 고려하여 데이터 타입을 선택해야만 실행 시에 데이터 표현의 변경을 위한 시간을 줄일 수 있습니다.

| 데이터 크기는 얼마나 될까.

이 데이터를 표현하기 위한 메모리 공간을 따지는 것도 적당한 데이터 타입을 선택하는 기준이 될 수 있습니다.

cellphone이나 **PDA** 같은 작은 디바이스에서는 데이터의 크기가 큰 이슈가 되기 때문입니다.

다음 표는 **Customer** 클래스의 **phone number**라는 변수의 데이터 타입을 결정하기 위해서 고려된 내용입니다.

▣ Table::Phone Number를 위한 데이터 타입 고려 사항

Data type	설명
String	사용자 인터페이스에서의 표현과 데이터 storage 에서 표현될 내용간의 매핑이 필요할 수도 있습니다.
long	이것은 국제전화번호를 표현하고자 할 때 크기가 모자랄 수도 있습니다.
PhoneNumber	전화번호를 표현하기 위한 별도의 클래스를 만드는 것을 말합니다. 이것은 데이터가 PhoneNumber 타입임을 쉽게 나타낼 수 있지만 별도의 클래스를 만드는 수고로움이 따릅니다.
char array	String 의 경우와 유사합니다.
int array	표현이 깔끔하지 못할 수 있습니다.

(3) Derived Attributes 생성

어떤 변수의 값은 다른 변수의 유동 값을 계산하여 대입되는 경우도 있습니다.

다음 그림에서 분석 시의 **Customer** 클래스의 **age**라는 변수에 주목합니다.

[이미지]

n Example Derived Attribute



age는 고정값을 주기에 무리가 따르는 변수입니다. 예를 들어, 전은수라는 고객의 나이를 ‘**20**’이라고 주었다면 다음 해에는 ‘**21**’이라는 값으로 변경되어야 하고 또 그 다음 해에는 ‘**22**’로 변경되어야 할 것입니다. 이때는 **age**의 값을 현재시스템의 날짜- 생년월일로 계산하는 것이 적당할 것입니다.

이때 ‘생년월일’이라는 변수가 필요해지는 것입니다.

이렇게 고정된 다른 변수의 값(ex. 생년월일)과 유동 값(ex. 현재 시스템의 날짜)을 계산하여 대입되는 값을 갖는 변수를 **Derived Attribute**라고 합니다.

Derived Attribute는 변수명 앞에 “/”를 붙입니다.

(4) Encapsulation(캡슐화) 적용

앞서 데이터의 보호를 위해 여러 가지 **visibility indicator**를 쓸 수 있다는 것을 배웠습니다. 이렇게 데이터를 보호하는 방법을 **캡슐화(Encapsulation)**라고 합니다. 캡슐화를 위해서는 다음과 같은 절차를 따릅니다.

1. 모든 변수를 **private**으로 만듭니다.
2. 이 변수를 읽기 위한 메소드(**accessors**)는 **public**으로 만듭니다.
3. 이 변수의 값을 변경하기 위한 메소드(**mutators**)도 **public**으로 만듭니다.

이 과정을 거친 클래스 디자인은 다음과 같습니다.

[이미지]

n Example Use of Encapsulation

Customer
-firstName : String {frozen}
-lastName : String {frozen}
-address : Address = new Address()
-phone [1..2] : PhoneNumber = null
«accessors»
+getFirstName() : String
+getLastName() : String
+getAddress() : Address
+getPhoneNumber(:int) :PhoneNumber)
«mutators»
+setAddress(:Address)
+setPhoneNumber(:int, :PhoneNumber)

2. Class Relationship의 강화

일반적으로 소프트웨어의 분석단계에서는 이미 클래스들간의 ‘관계’에 대한 부분이 설정되어집니다. 설계단계에서는 이 관계를 좀더 보강할 수 있습니다.

그러나 클래스들간의 관계를 강화하는 것은 반드시 설계단계에서 해야 하는 것은 아닙니다.

단지 좀더 정밀한 관계 표현이 이루어졌나를 다음과 같은 것에 주목하여 살펴 볼 수 있습니다.

- | **Type : association, aggregation, composition**
- | **Direction (또는 navigation)**
- | **Qualified associations**
- | **Declaring association management methods**
- | **many- to- many associations**
- | **association class의 해결**

1) Relationship Types

클래스들간의 “관계”에 대한 유형은 다음과 같은 것들이 있습니다.

- | **Association**
- | **Aggregation**
- | **Composition**

“관계”라는 것은 해당 객체가 다른 객체와 어떻게 연관되어 있는가를 조명하는 것입니다.

심화학습

■ Dependency

“관계”의 또 다른 유형으로 **Dependency**라는 것이 있습니다. 이것은 한 객체가 일을 할 때 다른 객체를 사용하는 것을 말합니다. 그러나 사용하는 객체를 변수로 가지지는 않습

니다.

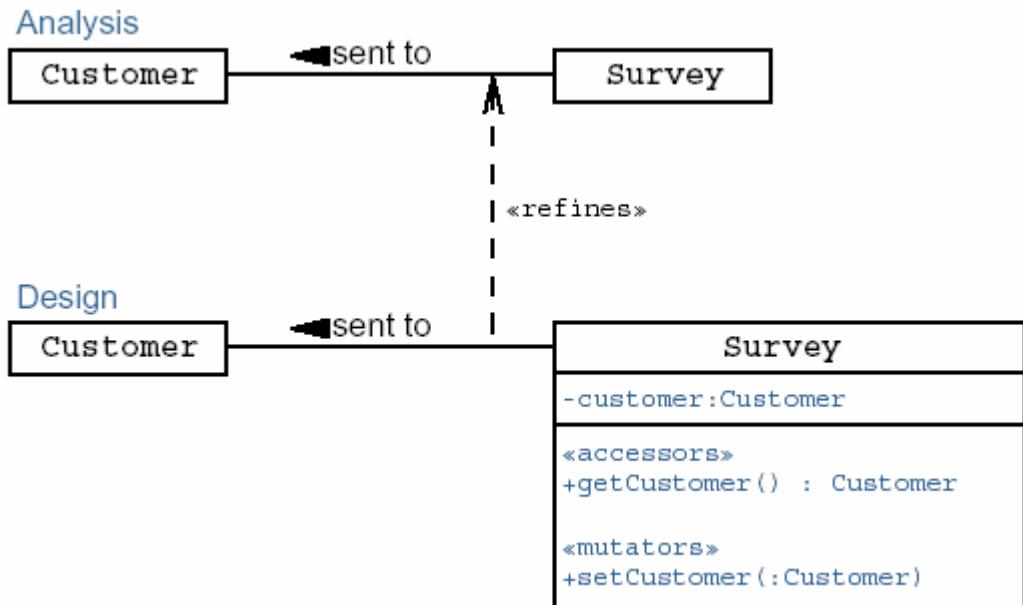
예를 들어, **Class1**이 **Class2**타입의 파라미터를 받는 메소드를 가지고 있다면 ‘**Class1**이 **Class2**에 의존적이다’라고 표현합니다.
이것을 **Dependency**라고 합니다.

(1) Association

“**Association**”은 한 객체가 다른 객체의 **instance variable**로 설정되어 있는 것을 말합니다.

[이미지]

n An Association Example



그림에서 **Survey** 클래스는 **Customer** 객체를 변수로 가지고 있습니다.

이때 **Association** 관계에 있다고 합니다.

두 클래스 사이에 실선으로 표현합니다.

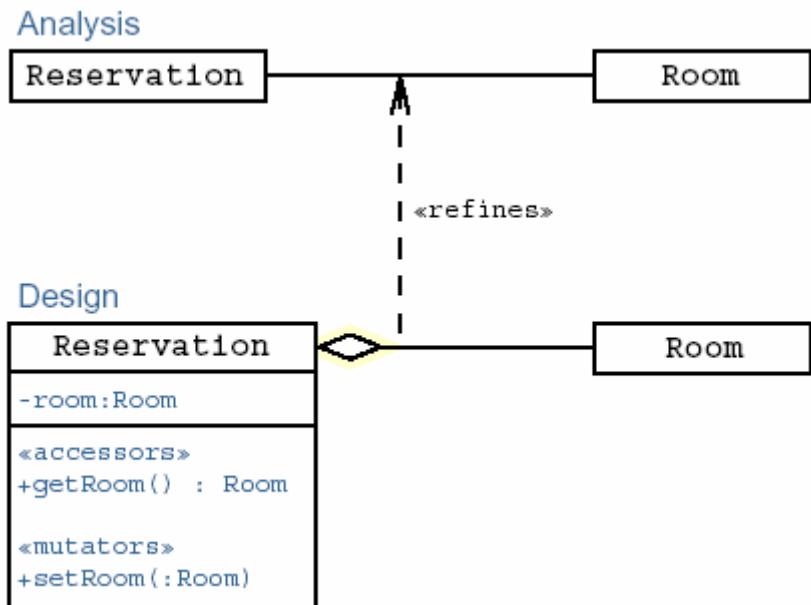
(2) Aggregation

“**Aggregation**”은 한 객체와 다른 객체간에 **whole-part** 관계가 있는 것을 말합니다.
이것은 **Association**과 의미상에서는 차이가 있지만 기능면에서는 별 차이가 없습니다.

다음의 예를 보면

[이미지]

n An Aggregation Example



Reservation에는 **Room**이 포함되어 있습니다.

포함하는(**whole**) 클래스가 **Reservation**이고 포함되는(**part**) 클래스가 **Room**입니다.

whole 클래스쪽에 **white diamond**을 가진 실선으로 표현합니다.

(3) Composition

“**Composition**”은 한 객체의 생성 때부터 다른 객체를 필요로 하는 관계를 말합니다.

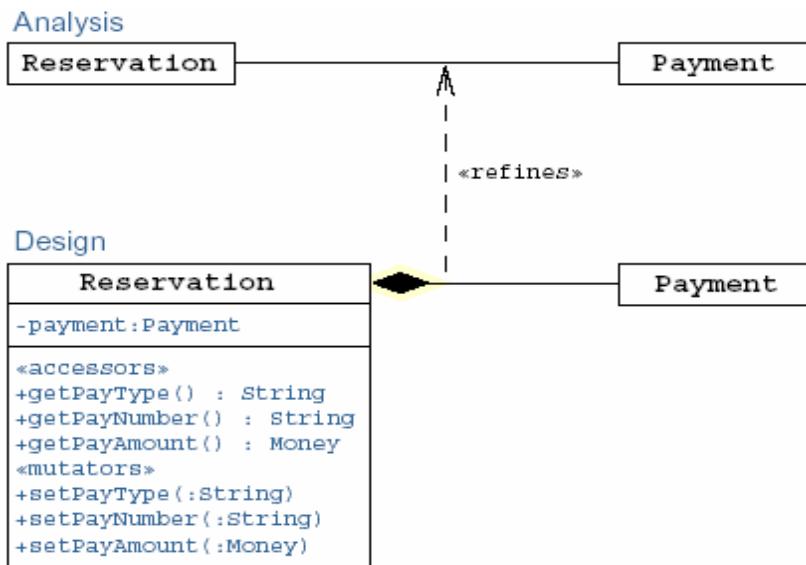
aggregation과 **composition**은 의미상으로는 유사하지만 기능면에서는 차이가 있습니다.

일반적으로 포함되는 객체(**composition**)는 포함하는 객체(**whole**) 밖에서는 사용되지 않습니다.

다음 예를 보면

[이미지]

■ A Composition Example



Reservation은 **whole object**가 되고 **Payment**가 **composition object**가 됩니다.

Reservation은 **Payment** 객체의 생성과 소멸에 관해 완전히 조절합니다.

Payment 객체의 내용을 변경하기 위해서는 반드시 **Reservation**의 메소드를 통해야만 합니다.

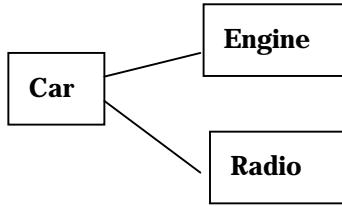
whole 클래스쪽에 **black diamond**를 가진 실선으로 표현합니다.

보충

n Relationship types 쉽게 이해하기

“자동차에는 엔진과 라디오가 있습니다”

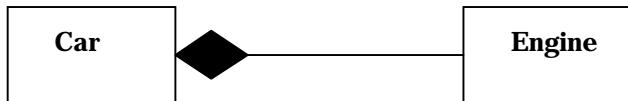
를 클래스 다이어그램으로 표현할 때



Car 클래스는 **Engine** 클래스와 **association**관계가 있습니다. 또한 **Car** 클래스는 **Radio** 클래스와도 **association**관계가 있습니다.

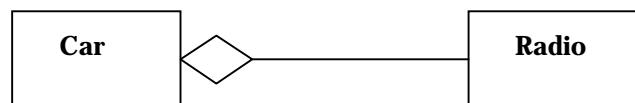
그러나 엔진은 자동차에 ‘완전히’ 조절되는 생명주기를 가지고 있습니다. 예를 들어, 차에 시동을 걸 때 엔진 객체가 생성되고 차가 멈출 때 엔진 객체가 사라지는 것입니다. 이때 **Car**와 **Engine**은 **Composition**관계가 있다고 합니다. 클래스 다이어그램

으로는 다음과 같이 표현합니다.



반면에 **Radio**는 자동차클래스에 의해 완전히 조절되는 객체가 아닙니다. 단지 실행 중 한 순간에 **Radio** 객체를 필요로 할 뿐입니다.

이때 **Car** 와 **Radio** 클래스간에 **aggregation**관계가 있다고 합니다. 이 관계는 다음과 같이 표현합니다.



2) Navigation

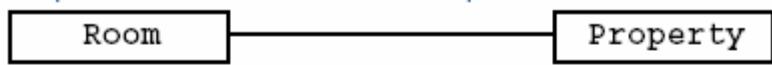
navigation arrow는 실행 시에 객체의 참조 관계의 방향을 보여 줍니다.

다음은 기본적인 **3** 종류의 **navigation**입니다.

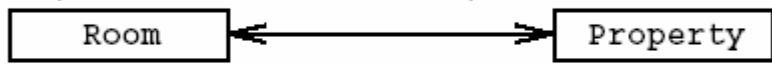
[이미지]

■ The Three Forms of Navigation Indicators

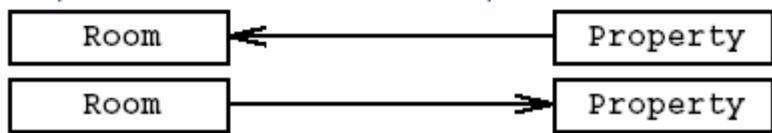
Implicit bidirectional relationship



Explicit bidirectional relationship



Explicit unidirectional relationship



일반 실선은 묵시적으로 양방향성을 의미합니다.

이것은 양쪽 끝에 화살촉을 표시함으로써 명시적 표현을 할 수도 있습니다.

한 객체가 다른 객체를 일방적으로 참조할 때는 참조당하는 쪽으로 화살촉이 향하게 그립니다.

navigation은 **accessor** 메소드를 갖는다는 것을 암시합니다.

양방향일 때는 두 클래스 모두 **accessor** 메소드를 갖습니다.

첫번째와 두번째 그림에서 **Room class**는 **getProperty()** 메소드를 갖고 **Property class**는 **getRoom()** 메소드를 갖을 것입니다.

세번째 윗그림에서는 **Property** 클래스는 **getRoom()** 메소드를 갖지만 **Room** 클래스에는 **getProperty()** 메소드가 없습니다.

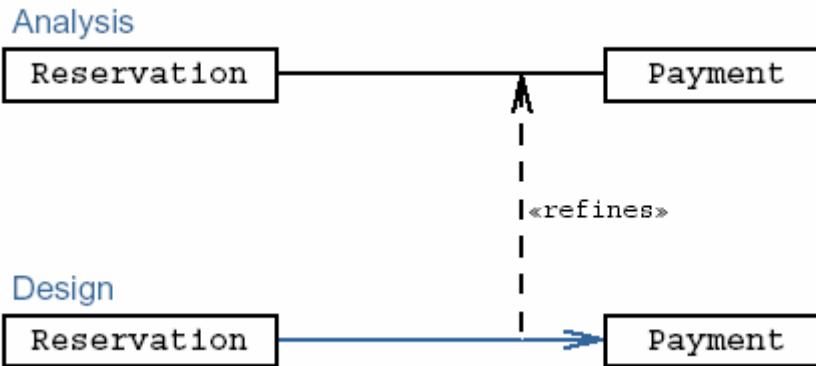
세번째 아랫그림은 반대의 경우입니다.

분석과정에서 이런 방향성에 대해 명확히 파악하지 못했다면 설계단계에서 명확히 보강 할 필요가 있습니다.

다음 그림에서처럼 분석 시 비즈니스 분석가가 **Reservation**과 **Payment**의 관계 방향성에 대해 아무런 언급도 하지 않았을 경우(일반 실선으로 표현되어 있는 경우) 설계단계에서 이것이 양방향이 아닌 것이 판단되었다면 단방향으로 명확히 표현할 수 있습니다.

[이미지]

■ An Example Navigation Refinement



3) Qualified Associations

어떤 경우에는 실행 시에 두 객체의 개수가 문제가 되는 경우도 있습니다.

일대다 경우나 다대다 경우처럼 한 객체가 다른 객체 몇 개와 대응되느냐는 문제를 표현해야 할 필요가 있다는 것입니다.

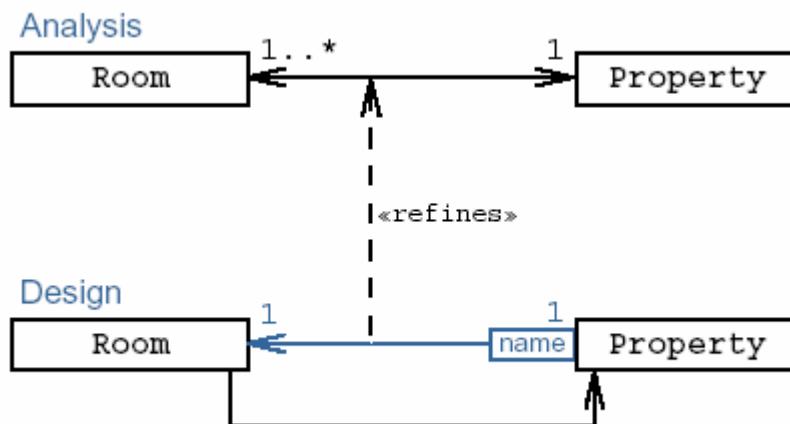
호텔 숙박시설은 여러 개의 방과 연관되어 있습니다.

이것은 분석 시에 **multiplicity**로 표현될 수 있지만 설계단계에서는 많은 방들을 어떤 기준으로 찾느냐는 것을 표현할 수 있습니다.

다음 그림에서

[이미지]

■ An Example Qualified Association



Room에는 고유 이름(**name**)이 있고 **Property**에서는 이 **Room**의 이름으로 다수의 **Room** 객체와 대응한다는 것을 보여주고 있습니다. 이때 **name**과 같은 다수의 객체중 하나를 구별할 수 있게 하는 인자를 **qualifier**라고 합니다.

4) Relationship Methods

Association method란 연관된 객체에 접근하고 변경할 수 있게 하는 메소드를 말합니다. 세 가지 경우로 나누어 설명해 보겠습니다.

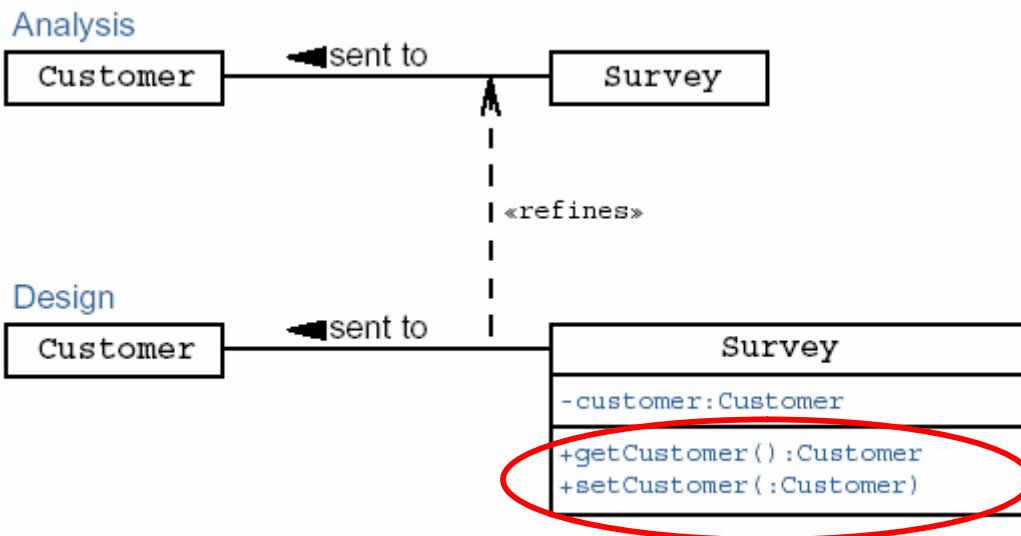
(1) one- to - one

두 객체가 일대일 대응이 되는 경우를 말합니다.

예를 들어, 호텔 관리 시스템은 고객에게 설문지를 보내려 합니다. 설문지 객체 하나는 고객 객체 하나와 연관되어 있으며 고객 객체에 접근하고 변경할 수 있는 메소드를 가지고 있습니다.

[이미지]

n Association Methods for a One- to- One Relationship



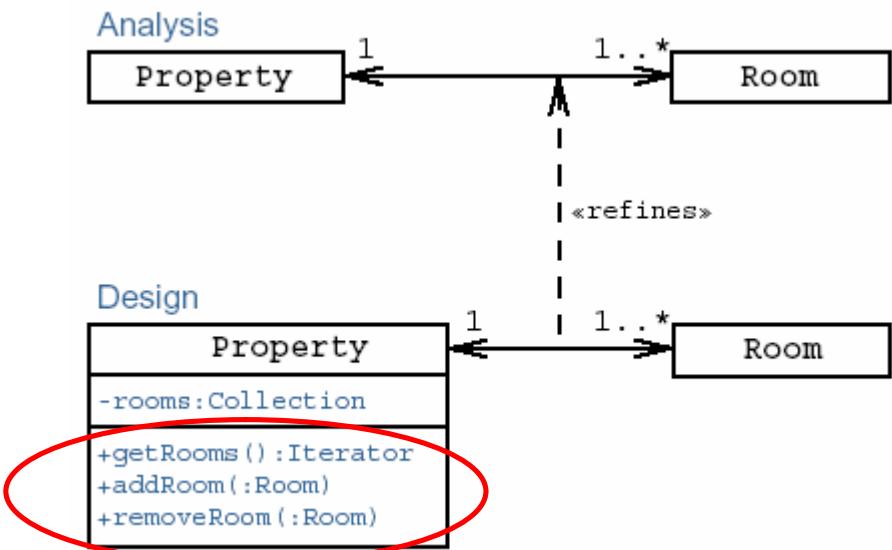
(2) one- to - many

일대다 관계는 **collection**을 사용할 필요가 있습니다.

예를 들어, 호텔 숙박시설은 여러 개의 방을 가지고 있습니다. 이때 **Property** 클래스가 여러 **Room** 객체를 가질 수 있도록 **collection** 객체를 사용합니다.

[이미지]

n Association Methods for a One- to- Many Relationship



그림에서 보면 **Property**가 다수의 **Room** 객체를 가지기 위해 **Collection**이라는 데이터 타입을 사용하고 있는 것을 알 수 있습니다.

이때 **Room** 객체 전부를 얻기 위해서는 **getRooms()** 메소드를 사용하여 **Iterator** 타입의 객체를 얻고, **Room** 객체를 추가하기 위해 **addRoom(:Room)** 메소드를 사용하며 가지고 있는 **Collection**에서 특정 **Room** 객체를 삭제하기 위해서 **removeRoom(:Room)** 메소드를 사용합니다.

이런 메소드를 “**association methods**”라고 하는 것입니다.

(3) many- to- many

다대다 관계를 해결하는 것은 앞의 두 경우보다 훨씬 힘이 듭니다.

이 과정에서는 다음 두 가지 방법으로 해결해 보겠습니다.

① 다대다 관계를 없애는 경우

다대다 관계는 때때로 설계 단계에서 필요 없음이 밝혀지는 경우가 있습니다.

예를 들어, 자동차 판매 시스템의 경우를 보겠습니다.

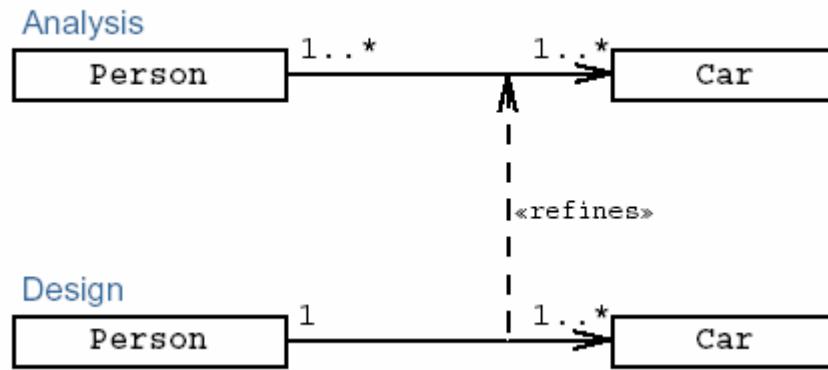
분석 시에는 한 사람이 여러 대의 차를 소유할 수 있고 한 차가 여러 사람에 의해 소유될 수 있다(다른 시간에)고 판단하여 **Person** 클래스와 **Car** 클래스를 다대다 관계로 설정하였습니다.

그런데 설계 시에 다시 내용을 검증해 보니 차를 특정 사람에게 팔게 되면 굳이 어떤 차의 소유주의 리스트를 기록하는 일이 중요하지 않음을 발견하게 되었습니다.

이때 다대다 관계를 일대다 관계로 변경할 수 있는 것입니다.

[이미지]

▪ Drop the Many- to- Many Relationship



② 다른 클래스를 사이에 두어서 직접적인 다대다 관계를 피하는 경우

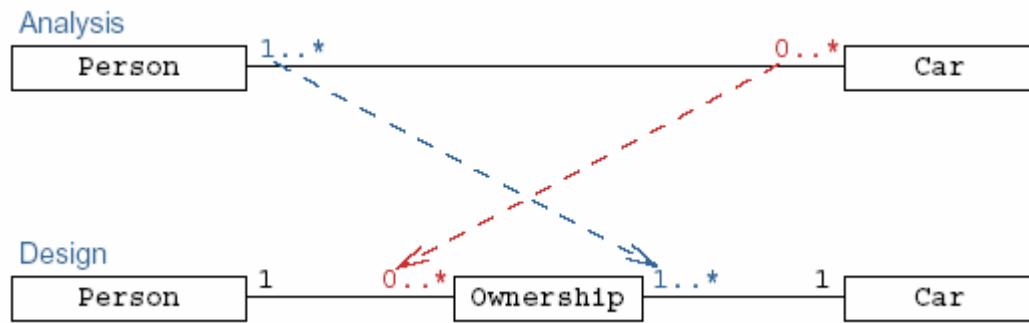
만약 다대다 관계가 유지되어야 할 경우 두 클래스 사이에 연관되는 다른 클래스를 끼워 넣음으로써 일대다 관계로 표현되게 하는 방법이 있습니다.

앞의 자동차 판매 시스템에서 어떤 차의 소유주가 누구인지를 계속 기록해야 할 경우 다대다 관계를 유지해야 합니다.

이때 소유관계를 명시할 수 있는 새로운 클래스를 만들어서 **Person**과 **Car**사이에 두어 직접적인 다대다 관계를 피할 수 있습니다.

[이미지]

Introduce an Intermediate Relationship



Ownership은 어떤 사람이 어떤 차를 소유하고 있는지를 보여주는 클래스입니다.

한 사람은 차를 한대도 소유하고 있지 않을 수도 있고 여러 대를 소유하고 있을 수도 있습니다. 이때 **Person**과 **Ownership**은 일대다 관계가 성립됩니다.

이미 팔린 한대의 차는 반드시 한 사람 이상의 소유주가 있습니다.(물론 사고 팔고가 여러 번 이루어 졌을 때 만 여러 소유주 리스트가 생깁니다) 이때 **Car**와 **Ownership**사이에도 일대다 관계가 성립됩니다.

이렇게 하여 **Person**과 **Car**사이에는 직접적인 다대다 관계가 생기지 않도록 할 수 있습니다.

5) Association Class 해결

분석 단계에서 두 클래스 사이에 연관 있는 다른 클래스를 명시해 놓은 것을 **association class**라고 합니다. **association class**는 오직 분석단계에서만 존재할 수 있습니다. 구현 시에 이를 직접적으로 코드화 할 수 있는 방법이 없기 때문에 설계 시에 반드시 이 클래스를 처리해야만 합니다.

다음과 같은 두 가지 방법이 있습니다.

(1) Case One

association class의 내용을 한쪽 클래스에서 흡수하고 **association class**는 없애버리는 방법입니다.

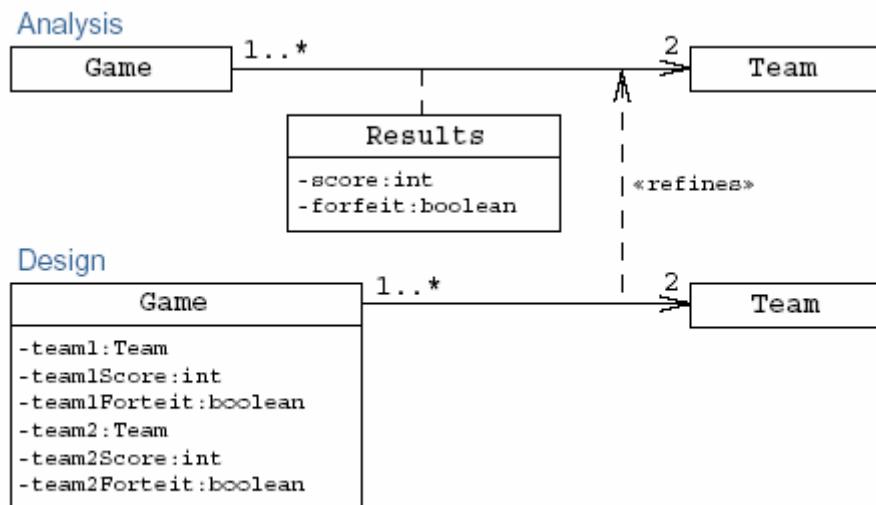
아래 그림처럼 축구 경기 어플리케이션으로 예를 들겠습니다.

한 게임에 두 팀이 경기를 합니다. 그러나 한 팀은 한번 이상의 게임을 할 수 있습니다. 각 게임마다 결과 스코어가 있고 팀에 벌점이 있는지를 알 수 있습니다. 분석 시에는 이러한 상황을 **Results**라는 **association class**로 해결 할 수가 있었습니다.

그러나 설계 시에는 이 **Results** 클래스의 모든 내용을 **Game** 클래스로 포함시키고 **Results** 클래스를 삭제해 버리기로 결정했습니다.

[이미지]

■ Resolving Association Classes: Case One



한 게임에 참여하는 팀의 수가 고정되어 있기 때문에 이 경우 이러한 방법은 꽤 유용합니다.

(2) Case Two

일대다 관계에서는 첫번째 해결책이 별로 도움이 되지 않습니다.

다음과 같은 예를 보겠습니다.

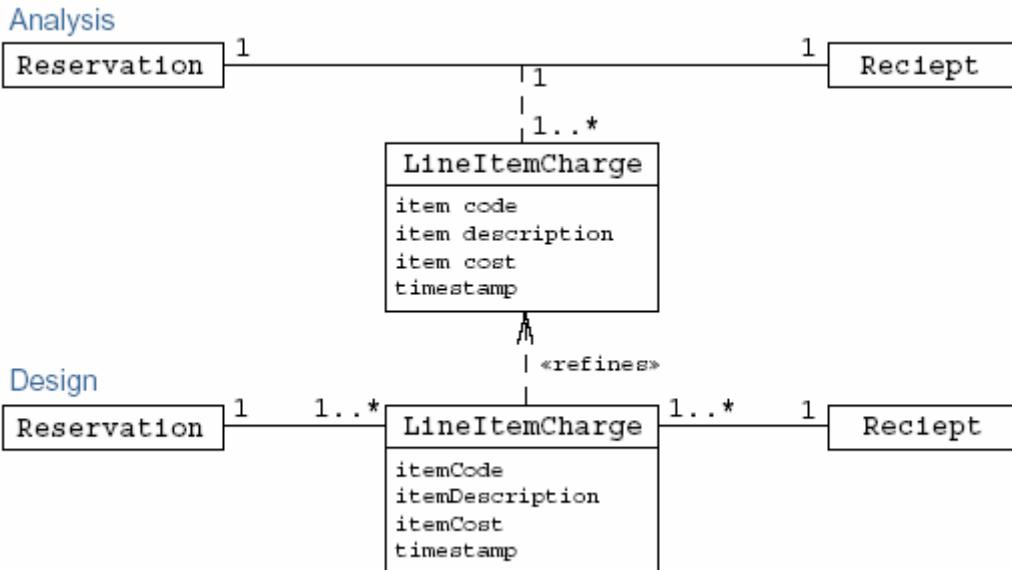
호텔 예약 시스템의 경우에 예약 한건에는 영수증이 하나 발부됩니다.

그러나 분석 시에 **LineItemCharge**라는 **association class**를 둘으로써 한 예약에 포함될 수 있는 여러 아이템(무엇을 사용했느냐는 것, 예를 들어 전화, 냉장고 속 음료나 음식, 별도로 주문한 품 서비스 등...)이 다수 있다는 것을 표현했고 영수증에도 이 아이템에 관한 다수의 내용이 관계가 있다는 것을 표현했습니다.

그런데 설계 시에 이 **LineItemCharge** 클래스의 내용은 반드시 필요하다는 결론이 내려졌고 첫번째 경우처럼 어느 한쪽 클래스에 그 내용을 포함시키기에는 무리가 있다는 판단을 하였습니다. 이 경우에는 아예 **Reservation** 과 **Receipt** 사이에 추가 시켜 버리는 것이 좋습니다.

[이미지]

▣ Resolving Association Classes: Case Two



이것은 구현단계에서 완전한 하나의 클래스로 코딩되어 집니다.

3. Domain Model에 메소드 추가

1) 분석, 설계 시에 정의될 수 있는 메소드

변수를 보강하고 클래스들 간의 관계도 더욱 명확히 하였다면 이제 메소드도 보강해야 합니다.

소프트웨어 개발 과정의 다음 워크플로우 동안 메소드가 정의 될 수 있습니다.

| CRC분석 단계에서 **responsibilities**는 메소드를 결정짓게 합니다.

다음 그림은 **Reservation**의 CRC 카드입니다.

[이미지]

n ‘예약(Reservation)’ Key Abstraction의 CRC 카드

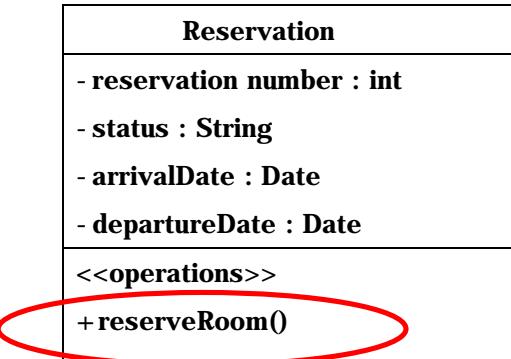
Reservation	
Responsibilities	Collaborators
<p>Reserves a Room</p> <p>status (New, Held, Confirmed) arrival date departure date form of payment reservation number</p>	Room Customer

Responsibilities 중 업무 내용에 해당하는 **Reserves a Room**(방을 예약하다)은 메소드가 될 수 있습니다.

따라서 다음과 같이 표현할 수 있을 것입니다.

[이미지]

n 메소드가 정의된 Reservation Class Diagram

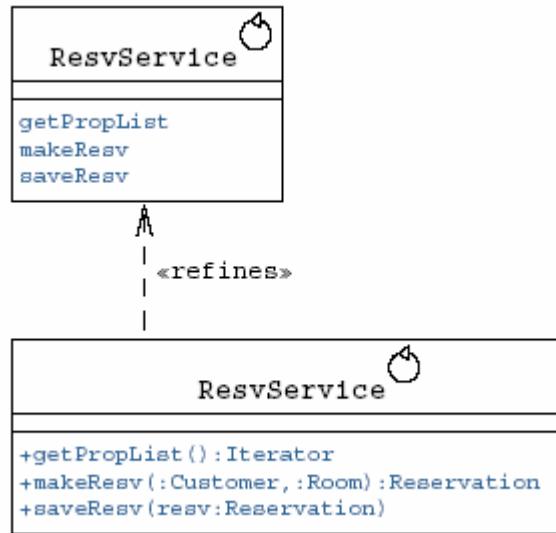


| 강력 분석단계에서 **Service class**의 메소드가 결정됩니다.

호텔예약 시스템의 강력 분석 단계에서 **ResvService**라는 클래스가 예약을 처리하는 메소드를 제공하도록 결정되었다면 다음 그림과 같이 보강될 수 있습니다.

[이미지]

n Example Refinement of Methods



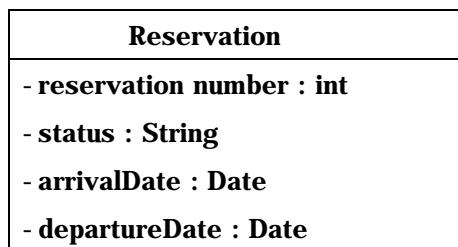
메소드 선언을 위한 **UML syntax**는 다음과 같습니다.

[visibility] name [({[param] [:type]}*)] [:return – value] [{property- string}]

- | 설계단계에서 변수와 **association**을 위한 **accessor** 메소드와 **mutator** 메소드가 결정 됩니다.
설계 단계에서 추가 될 수 있는 **accessor** 메소드는 일반적으로 **getter** 메소드라고도 불리며 메소드의 이름 앞에 **get**이 붙는 것이 상례입니다. 마찬가지로 **mutator** 메소드는 **setter** 메소드라고 불리며 메소드의 이름 앞에 **set**을 붙입니다.

이미지

- Reservation class의 **accessor**와 **mutator**의 추가



```

<<operations>>
+ reserveRoom()

<<accessors>>
+ getReservationNumber() : int
+ getStatus() : String
+ getArrivalDate() : Date
+ getDepartureDate() : Date

<<mutators>>
+ setReservationNumber(: int)
+ setStatus(: String)
+ setArrivalDate(: Date)
+ setDepartureDate(: Date)

```

2) 추가될 수 있는 메소드

이외에도 추가 될 수 있는 메소드로는 다음과 같은 것들이 있습니다.

| Object management

프로그래 언어내에서 객체의 관리에 관계하는 메소드들이 있습니다. C++ 같은 경우는 생성자를 통해 이런 일들을 합니다. 자바의 경우는 **equals** 메소드와 **toString** 메소드가 객체를 관리하는 메소드로 취급됩니다.

| Unit testing

클래스의 한 부분(**unit**)을 테스트 하기 위한 메소드도 삽입될 수 있습니다. 자바에서는 **main** 메소드로 테스트를 수행할 수 있습니다. 또한 다른 테스트 클래스를 사용하거나 **Junit** 같은 **unit test**용 프레임워크를 쓸 수도 있습니다.

| Recovery and inverse operations

복잡한 기능의 수행을 되돌리는 메소드도 추가 될 수 있습니다.

일반적으로 사용자 인터페이스 프로그램의 **Undo(되돌리기)** 기능에 해당하는 메소드를 말합니다.

3) 생성자 추가

생성자는 메소드와 유사한 구조를 갖습니다. 그러나 리턴 타입이 없습니다. 또한 자바에서는 생성자의 이름이 클래스 이름과 동일해야만 합니다. UML에서는 생성자의 표기 방법이 별도로 제공되지 않기 때문에 다른 메소드들과의 구별을 위해 **<<constructor>>**라는 스태레오 타입을 사용하는 것이 좋은 방법입니다.

n Example Constructors

Customer
<pre>-firstName : String {frozen} -lastName : String {frozen} -address : Address = new Address() -phone [1..2] : PhoneNumber = null</pre>
<pre>«constructors» +Customer(fName:String, lName:String) +Customer(fName:String, lName:String addr:Address)</pre>
<pre>«accessors» +getFirstName() : String +getLastName() : String +getAddress() : Address +getPhoneNumber(:int) : PhoneNumber</pre>
<pre>«mutators» +setAddress(:Address) +setPhoneNumber(:int, :PhoneNumber)</pre>

과정명

OO- 226

Object- Oriented Analysis and Design Using UML

단원6 : 솔루션 모델의 구축

모듈 5 : Statechart Diagram 을 통한 솔루션 모델의 보강

담당강사 : 전은수

생각해봅시다

객체가 상태를 갖는다는 것은 무엇을 의미할까요?

실세계에서 상태를 갖는 엔티티를 한번 찾아보세요. 그 엔티티의 상태는 무엇입니까? 무엇이 엔티티의 상태를 변하게 합니까?

제 주위에는 히터와 에어컨의 역할을 동시에 할 수 있는 실내 온도 조절기가 있습니다.

이 기계를 켜면 일단 ‘준비상태’가 됩니다. 온도 조절 버튼을 내리면 ‘에어컨이 가동되는 상태’가 되고 온도 조절 버튼을 올리면 ‘히터가 가동되는 상태’가 됩니다.

이를 소프트웨어 개발에 적용한다면 어떻게 될까요?

실내 온도 조절기는 객체입니다. 이 객체가 여러 상태로 변하는 것을 시스템이 알아야 합니다.

이 과정에서 바로 상태가 변하는 객체에 대해 배우게 됩니다.

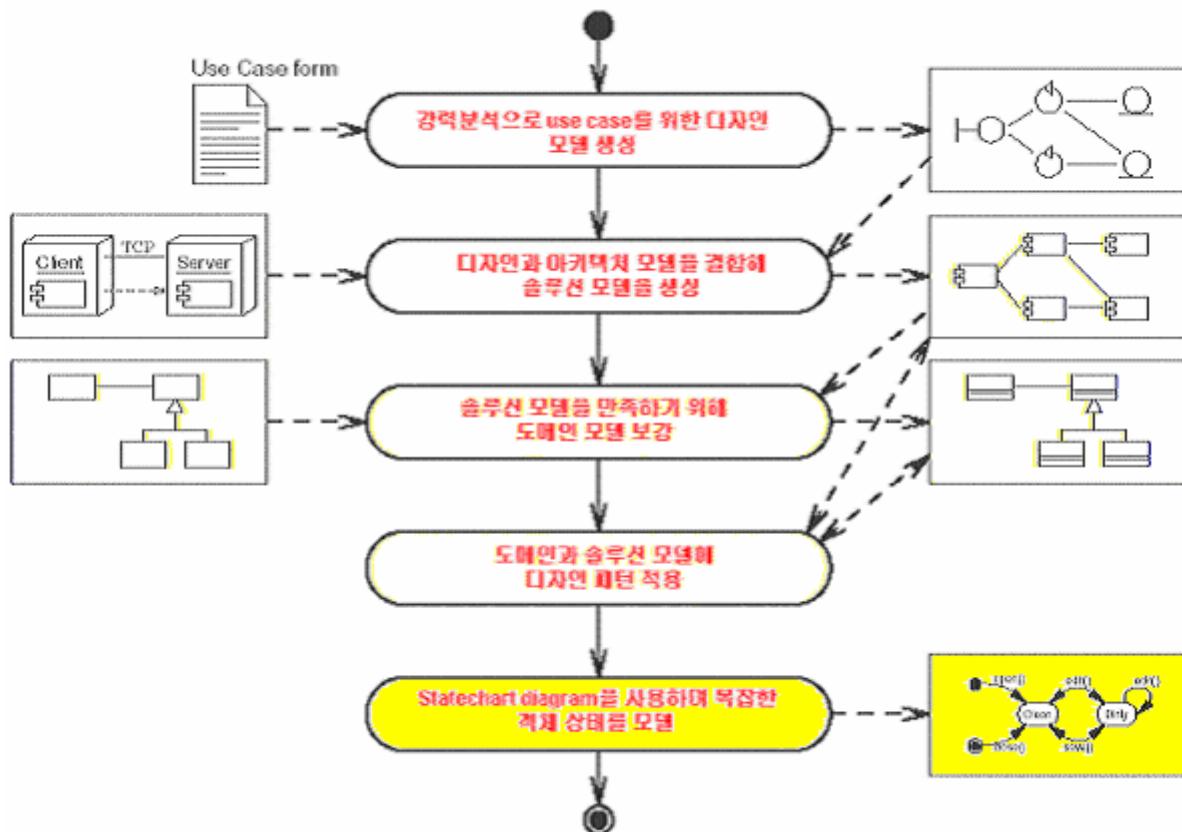
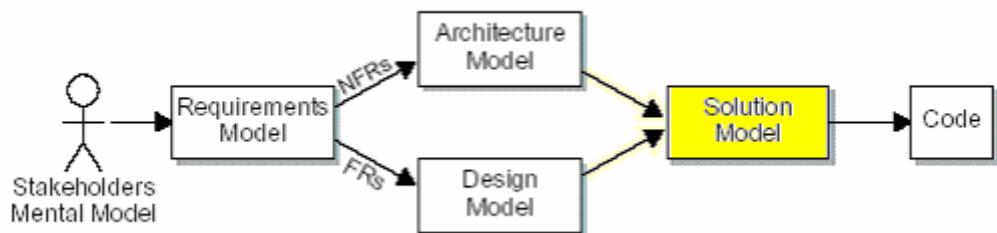
애니메이션



■ 학습하기 ■

[참고하세요]

- 노란색 부분이 이 모듈에서 학습하는 내용입니다.



1. 객체의 상태 변화

1) 객체의 상태

객체의 상태에 대해 생각하는 두 가지 방법이 있습니다.

| 변수값으로 생각

객체의 상태는 객체를 위한 특정한 변수 값의 모음으로 생각 할 수 있습니다.

일반적으로 사람들은 객체의 상태를 객체 변수의 상태로 생각하게 됩니다.

이것이 틀리지는 않지만, 객체의 행동에 관한 중요한 부분을 놓칠 수도 있으므로 유의하여야 합니다.

예를 들어, 자동 온도 조절기(**heating ventilating and air-conditioning system:HVAC**)를 생각해 본다면 우리는 **HVAC** 시스템은 일반적으로 전원이 켜져 있다는 상태만 생각하게 되고 현재 온도 범위 내에서만 생각하게 됩니다. 실제 **HVAC** 시스템이 돌아갈 수 있는 외부 자극에 대해서는 놓치고 있다는 것입니다. 따라서 객체의 상태에 대해 생각할 때는 다음 한가지를 더 고려해야 합니다.

| 외부 자극에 연관된 행동으로 생각

데이터보다 ‘행위’에 초점을 맞추어 보는 것입니다.

HVAC 시스템에서 기본 네 가지 상태를 생각해 볼 수 있습니다.

‘초기 상태’는 아직 전원이 켜지지 않은 상태입니다.

‘idle 상태’는 전원은 켜졌지만 작동은 하지 않고 있는 상태입니다.

‘Cooling 상태’는 온도를 낮추도록 가동되고 있는 상태입니다.

‘Heating 상태’는 온도를 높이도록 가동되고 있는 상태를 말합니다.

이것을 **HVAC** 시스템의 ‘동작’ 혹은 ‘행위’에 초점을 맞추어 생각해 봅니다.

상태를 변하게 하는 시작점이 있습니다.

예를 들어, **HVAC** 시스템에서는 현재 온도입니다. 현재 실내 온도가 얼마이고 기준온도가 얼마나에 따라 **Cooling** 상태로도 **Heating** 상태로도 변환될 수 있는 것입니다.

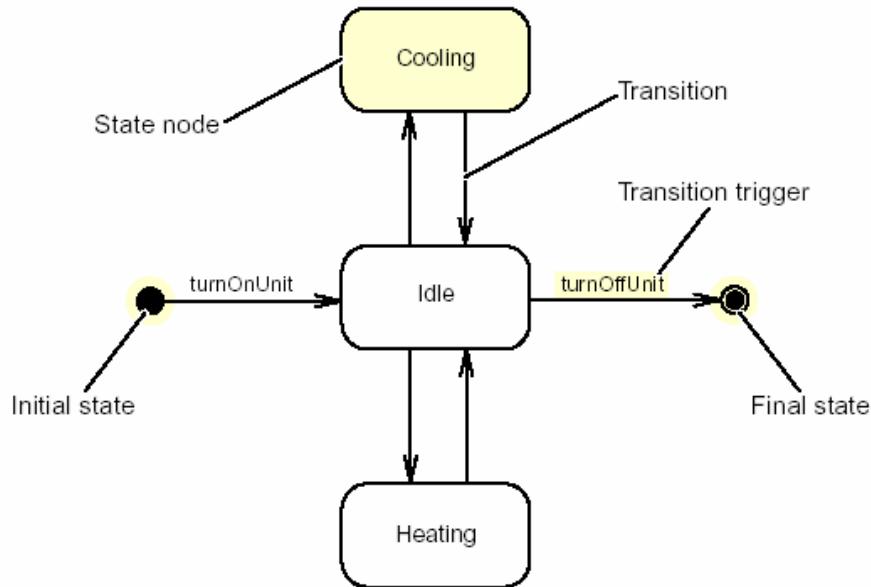
이 모듈에서는 상태가 변하는 이 같은 객체들이 소프트웨어로 표현되어져야 할 때 어떻게 모델화되는지에 관해 학습할 것입니다. **HVAC** 시스템은 계속 이 모듈의 예로 쓰이게 될 것입니다.

2) Statechart Diagram의 요소

HVAC 시스템과 같은 객체의 상태 변화를 가시화할 수 있는 **UML diagram**이 바로 **Statechart Diagram**입니다.

다음 그림이 **Statechart Diagram**의 요소를 잘 표현하고 있습니다.

■ An Example Statechart Diagram



모든 **Statechart diagram**은 **Initial state**(초기 상태)와 **Final state**(종료 상태)를 표현해야 합니다.

일반적으로 이 노드는 연관된 행위를 가지지 않습니다.

한 상태에서 다른 상태로의 ‘전이(transition)’는 화살촉을 가진 실선으로 표현됩니다.

전이는 연관된 ‘trigger(이하 트리거)’를 갖습니다.

이 트리거는 일반적으로 객체에 대한 외부 자극입니다.

HVAC 시스템의 경우 초기 상태에서 **Idle** 상태로의 전이는 전원을 켜는 트리거를 통해서 이루어집니다.

이 다이어그램에서 트리거가 없다면 상태 변화에 대한 설명이 불완전할 수 있습니다.

이제 **Statechart diagram**의 요소를 자세히 살펴 보겠습니다.

(1) State Transitions

상태의 ‘전이’는 실행 시의 상태의 변화를 의미합니다.

상태 전이는 일반적으로 한 상태에서 다른 상태로 변화하게 만드는 트리거를 포함하고 있습니다.

HVAC 시스템에서 **Idle** 상태에서 실제 온도를 높이는 상태로 전이되거나 온도를 낮추는 상태로 전이됩니다.

전이는 또한 “**guard condition**”과 “**행위**”들을 가집니다.

HVAC 시스템이 **Idle** 상태에서 **Cooling** 상태로 가기 위한 “**guard condition**”은 이 시스템이 에어컨을 장착하고 있다는 것입니다. 즉, “**guard condition**”은 전제되는 요

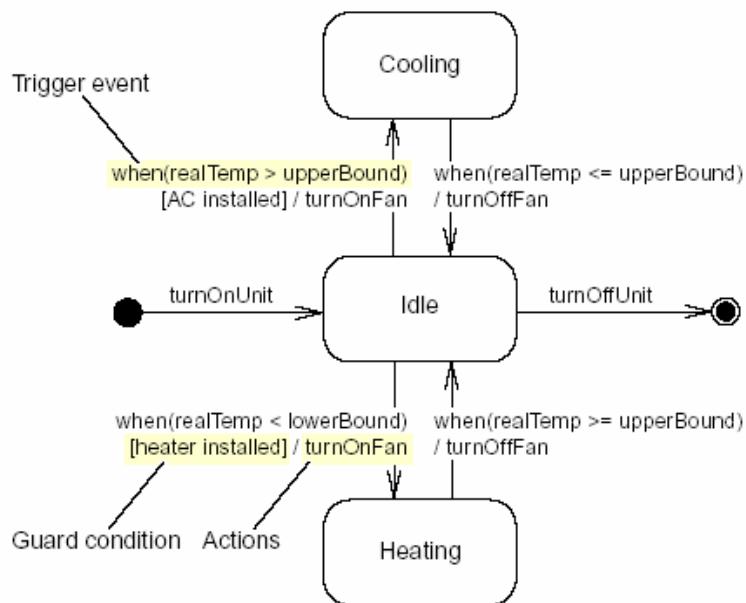
소라는 것입니다.

일단 에어컨 기능이 있고 실내 온도가 기준 온도보다 높다면 fan이 돌아가는 행위가 따를 것입니다.

이것을 **Statechart Diagram**으로 그리면 다음과 같습니다.

이미지

n HVAC Statechart Diagram With Transitions



(2) State Nodes

State node(이하 상태 노드)는 실행 시 한 객체의 상태를 표현합니다.

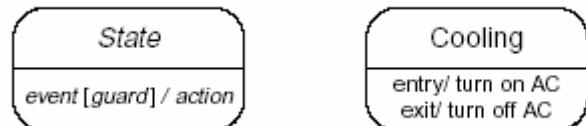
상태 노드의 내부에는 경우에 따라 발생하는 특별한 동작이 표현될 수 있습니다.

이 동작을 **event(이하 이벤트)**라고 합니다.

다음은 상태 노드의 그림입니다.

이미지

n Internal Structure of State Nodes



이벤트의 종류는 다음과 같습니다.

Entry – 그 상태로 들어 오게 한 특별한 ‘진입 행동’

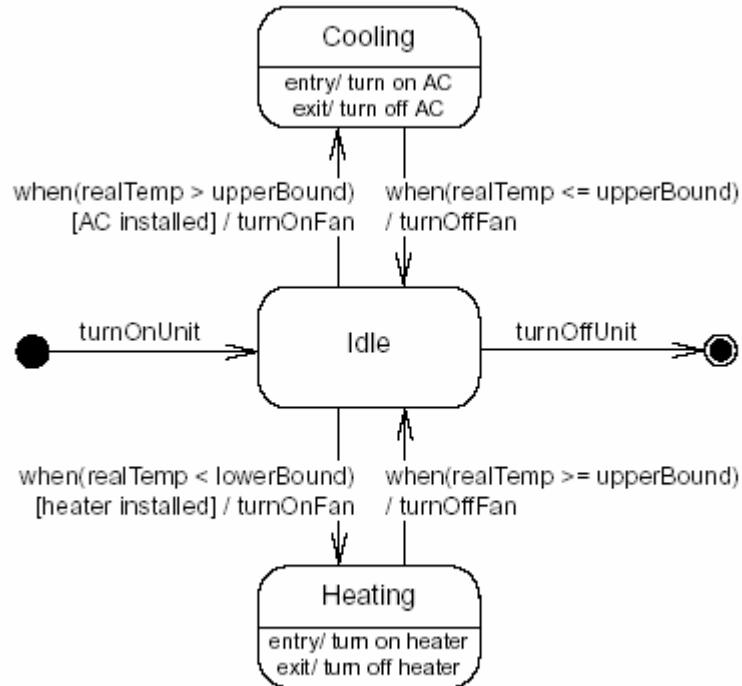
Exit – 그 상태를 빠져 나가게 할 특별한 ‘탈출 행동’

Do – 그 상태를 이어나가는 “유지 행동”

이벤트를 표현한 HVAC 시스템의 **Statechart diagram**은 다음과 같습니다.

[이미지]

n Complete HVAC Statechart Diagram



3) Complex Object를 위한 Statechart Diagram 생성

복잡한 객체를 심도 있게 이해하기 위해서는 **Statechart diagram**이 유용합니다.

이제 어떤 복잡한 객체를 분석해 나가면서 이를 **Statechart diagram**으로 그리는 방법을 공부해 봅니다.

일단, **Statechart diagram**을 그리는 전체 순서는 다음과 같습니다.

1. **Initial state**와 **Final state**를 그립니다.
2. 객체가 안정적일 때의 상태를 그립니다.
3. 그 객체의 수명동안 상태가 어떻게 달라지는지 순서를 파악합니다.
4. 객체의 상태가 전이되는 트리거와 이벤트를 표현합니다.
5. 상태 노드 안에 액션을 기술합니다.

(1) Step 1 – 초기 상태와 마지막 상태를 그림

Step 1은 쉽습니다.

다이어그램 안에 **Initial state**와 **Final state**를 표현합니다.

그러나 이것을 잊어버려서는 안됩니다. 모든 **Statechart diagram**에는 반드시 **Initial state**와 **Final state**가 있어야 합니다.

다음 그림처럼 표현합니다.

[이미지]

n Step 1 – Start with the Initial and Final States



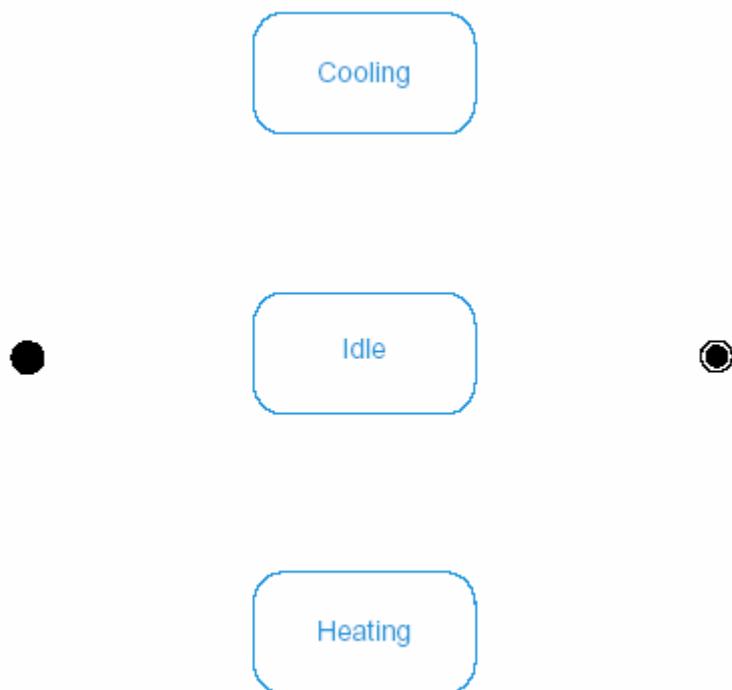
(2) Step 2 – 상태 노드 표현

객체의 안정적인 상태를 결정 짓습니다.

이것을 상태 노드로 표현합니다.

[이미지]

n Step 2 – Determine Stable Object States



안정적인 상태란 객체가 의미있는 일정 시간만큼 가지게 되는 조건과 행위의 세트를

말합니다.

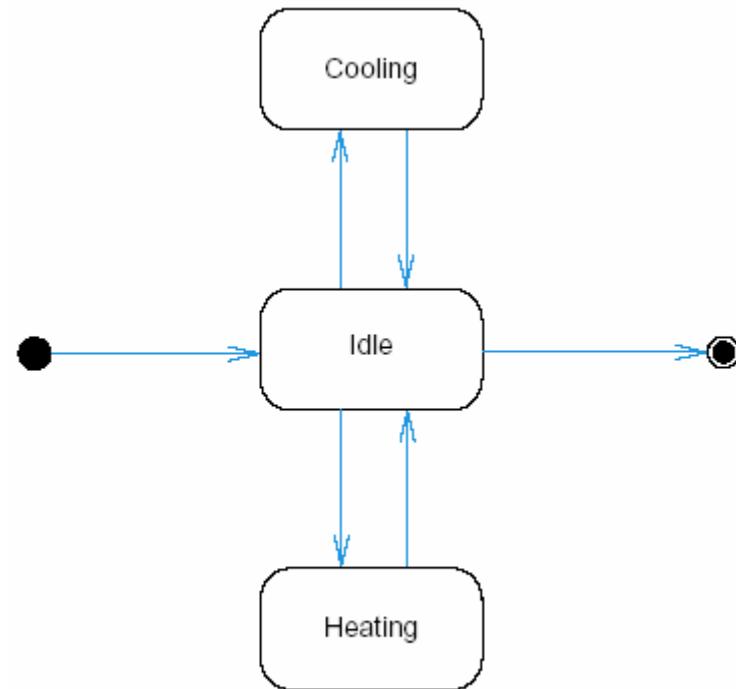
HVAC 시스템에서 전원은 켜져 있지만 아무것도 하지 않고 있는 상태를 ‘**Idle** 상태’로 결정했다면 현재 온도가 기준 온도보다 높은 조건과 온도를 낮추는 행위를 하고 있는 상태를 ‘**Cooling** 상태’라고 결정할 수 있습니다.

(3) Step 3 – 상태의 변화 순서나 방향을 표시

상태가 옮겨지는 단계가 있습니다. 예를 들어, HVAC 시스템에서는 **Idle** 상태에서 **Cooling** 상태가 되거나 **Heating** 상태가 됩니다. 그러나 **Cooling** 상태에서 바로 **Heating** 상태가 되거나 **Heating** 상태에서 바로 **Cooling** 상태가 될 수는 없습니다. 이를 화살촉을 가진 실선으로 나타냅니다.

[이미지]

n Step 3 – Specify the Partial Ordering of States



(4) Step 4 – 상태를 전이 시키는 특별한 이벤트나 행위를 기술

‘트리거’와 ‘**guard condition**’, 경우에 따라서 ‘행위’도 표현합니다.

트리거가 없으면 전이는 조건 없이 즉시 발생하는 것입니다.

“**guard condition**”이 없을 때는 트리거가 발생하는 시점에 전이가 일어납니다.

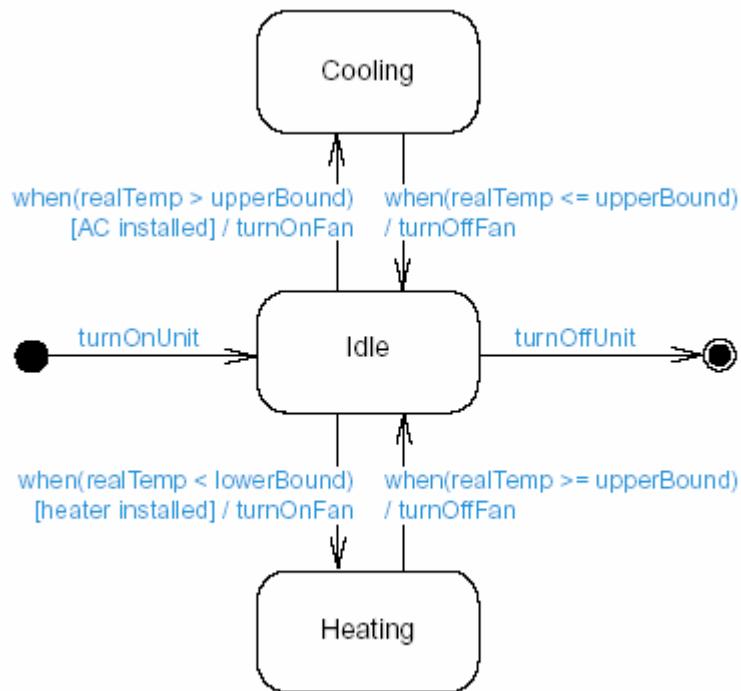
“**guard condition**”은 트리거가 발생했을 지라도 전이를 중지시킬 수 있습니다.

또, 어떤 ‘행위’도 없다면 객체에 별다른 영향을 끼치지 않고 전이가 됩니다.

만약 어떤 ‘행위’가 있다면 객체가 새로운 상태로 들어오기 전에 어떤 특별한 영향을 미칩니다.

[이미지]

n Step 4 – Specify the Transition Events and Actions



(5) Step 5 – 상태 노드 안에 특별한 행위 기술

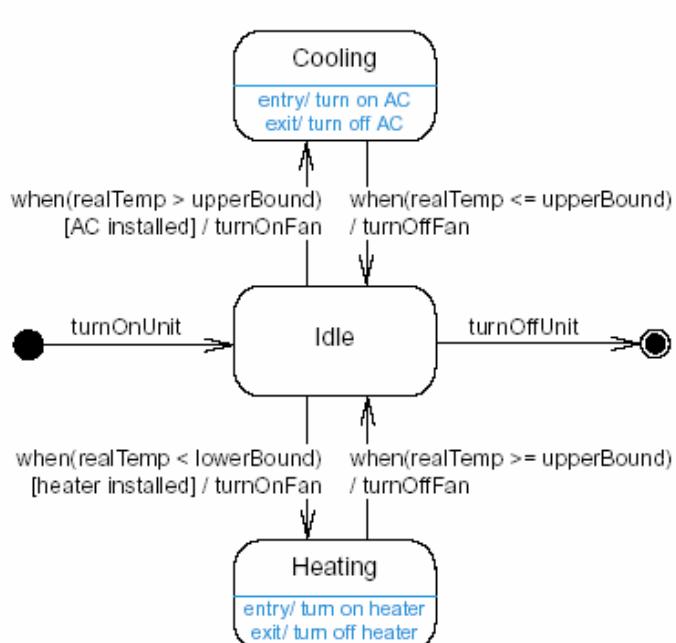
마지막으로 상태 노드 안에서 일어 날 수 있는 특별한 이벤트를 기술합니다.

예를 들어, HVAC 시스템에서 **Cooling** 상태는 에어컨이 켜지는 이벤트에 의해 이 상태로 진입될 수 있고 에어컨이 꺼지면서 이 상태를 탈출할 수 있습니다.

이것을 상태 노드 안에 표현합니다.

[이미지]

n Step 5 – Specify the Actions Within a State



4) Complex Object의 구현

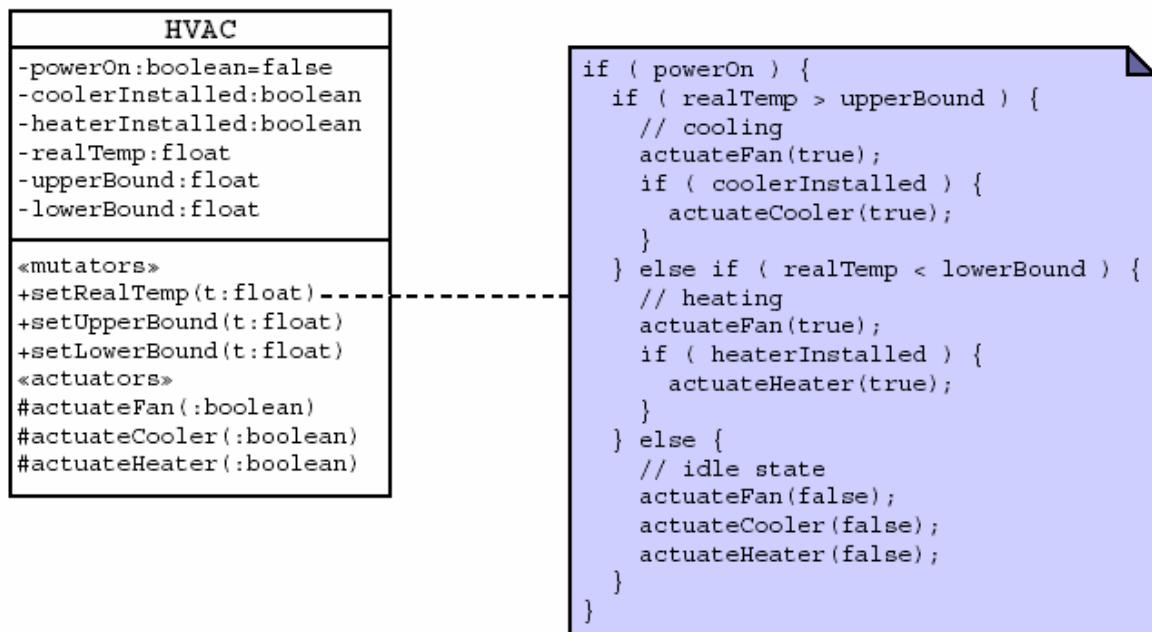
Statechart diagram은 복잡한 상태 변화를 갖는 객체를 모델화한 것입니다.

그렇다면 이렇게 모델링 된 객체는 어떻게 구현할 수 있을까요?

만약 여러분들이 특별한 패턴을 사용하지 않는다면 다음과 같이 구현될 수 있을 것입니다.

[이미지]

n HVAC Class With Complex State Code



먼저, **HVAC**라는 이름의 클래스를 만들었습니다.

이 클래스의 변수로는 전원의 상태를 아는 **powerOn**이 있고 에어컨이 장착되어 있는지에 관해 아는 **coolerInstalled**라는 변수와 히터가 장착되어 있는지에 대해 아는 **heaterInstalled**가 있습니다.

여기에는 현재 온도 값을 나타내는 **realTemp**, 최저기준온도를 나타내는 **lowerBound**, 최고기준온도를 나타내는 **upperBound**가 있습니다.

setRealTemp라는 메소드를 구현하고자 한다면 오른쪽 노트의 내용처럼 **if**를 사용한 조건 체크가 될 수 일 것입니다.

이런 조건절을 가진 코드는 작성하기 어려울 뿐만 아니라 객체의 상태를 정확하게 표현하는데도 문제가 있을 수 있습니다.

예를 들어, 코드의 **else**절에서는 **idle** 상태로 되돌아가기 위해 세가지 메소드를 호출합니다.

이는 시간과 비용이 많이 드는 작업입니다.

좀 더 쉽고 정확하게 코딩하는 방법이 있을까요?

또 다른 문제는 새로운 상태를 포함하도록 하는데 유연한가 하는 문제입니다.

어떤 객체에 새로운 상태가 도입되어야 한다면 이미 작성된 이 코드를 변경, 추가하는 것은 쉽지 않습니다.

이런 문제들을 해결하기 위하여 적당한 패턴을 적용해 보도록 하겠습니다.

1. Statechart Diagram을 통한 솔루션 모델의 보강

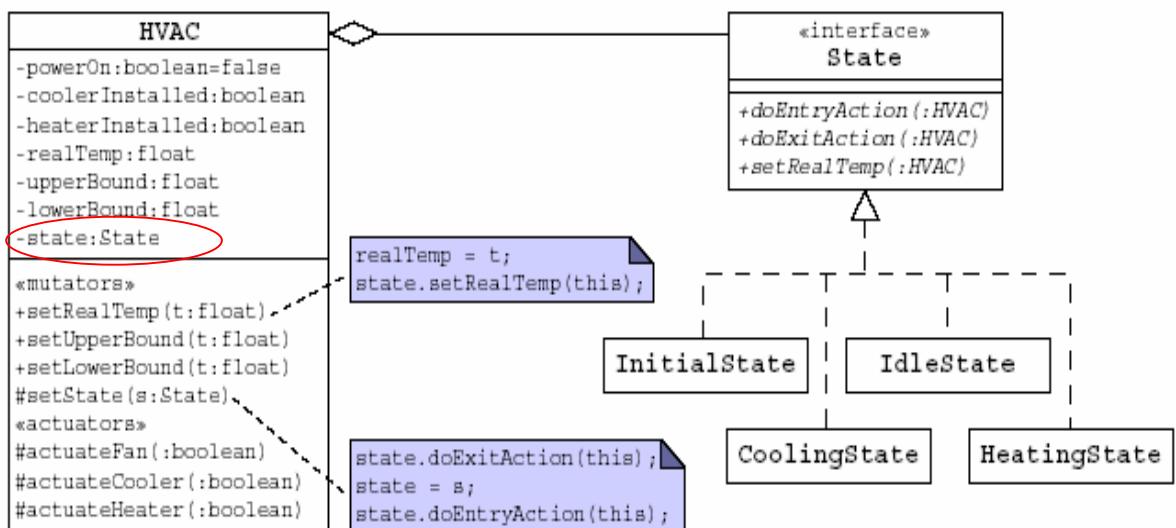
1) State Pattern

상태가 복잡한 객체를 구현하는데 가장 적합한 패턴이 **State Pattern**입니다.

먼저 HVAC 시스템을 **State Pattern**을 적용한 다이어그램으로 살펴 보겠습니다.

[이미지]

n HVAC Class Using the State Pattern



HVAC 클래스는 변수에 **state**를 하나 더 추가하고 있습니다. 이 변수는 **interface State** 타입의 객체입니다. 이 변수의 값은 **setState**라는 메소드를 통해 실행 시에 동적으로 획득됩니다.

State라는 **interface**에는 **doEntryAction**, **doExitAction**, **setRealTemp**라는 세 가지 메소드가 있습니다.

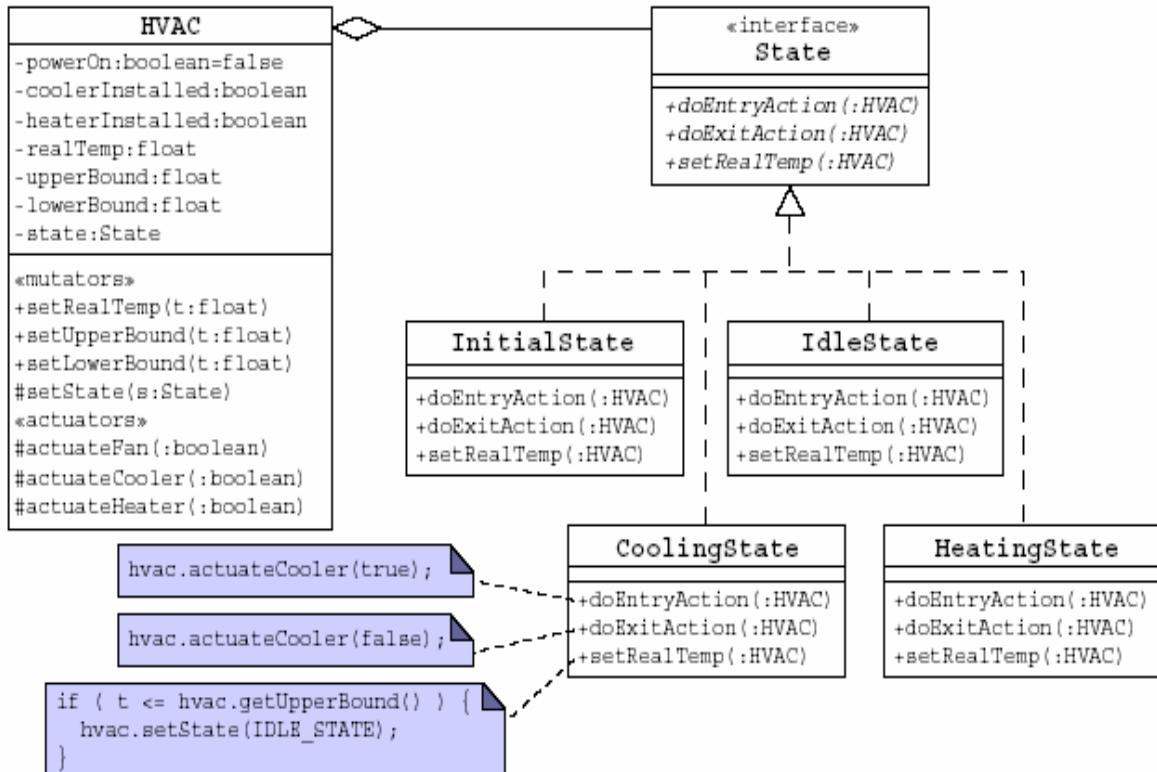
InitialState, **IdleState**, **CoolingState**, **HeatingState**는 모두 이 메소드를 구현한 서브 클래스입니다.

예를 들어, **CoolingState**라는 클래스의 **doEntryAction** 메소드는 에어컨을 켜는 행위를 포함하게 구현되어야 하고 **doExitAction** 메소드는 에어컨을 끄는 행위를 구현해야 합니다.

다음이 이 내용을 도식화한 것입니다.

[이미지]

n HVAC Class Using the State Pattern



(1) State Pattern : Problem

State Pattern^o 적용될 수 있는 ‘문제 상황’은 다음과 같습니다.

- | 객체가 실행 시에 상태에 의존하는 행위를 할 때
 객체가 복잡한 상태 변화를 갖는 경우를 말합니다.
- | 객체의 상태에 의존하는 함수가 크고 많은 조건절을 가질 때
 객체의 상태 변화를 코딩할 때 **if/else**를 사용하는 많은 조건절은 작성하기 어렵고 변경하기도 어렵습니다.

(2) State Pattern : Solution

이런 문제들을 해결하기 위하여 다음과 같은 내용을 적용합니다.

- | 객체의 상태를 기반으로 하는 메소드를 가지는 **interface**를 만듭니다.
- | 이 **interface**를 구현한 **concrete class**를 만듭니다.
- | 현재 객체 상태를 표현하고 있는 객체에게 상태 기반 메소드를 대행하게 합니다.

보충

n State Pattern VS. Strategy Pattern

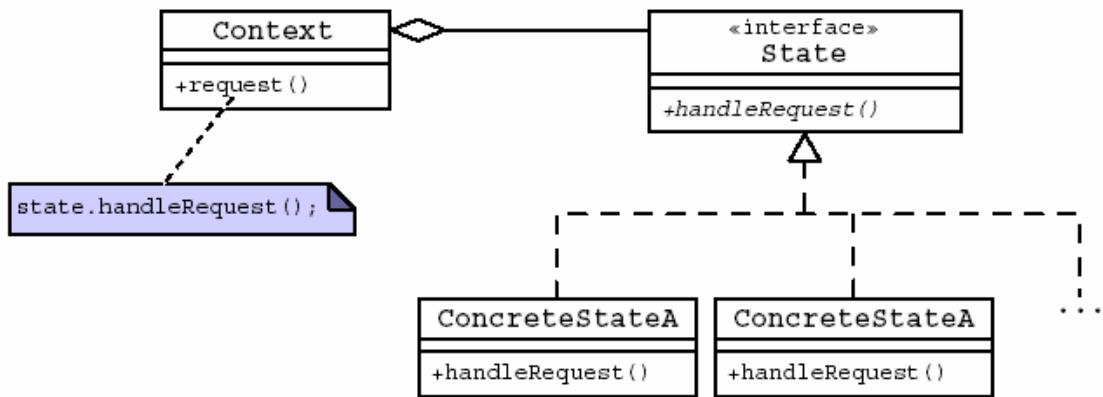
State Pattern은 Strategy Pattern과 유사합니다.

그러나 Strategy는 알고리즘에 관계하는 메소드가 각 Concrete class마다 다르게 구현되어 있고 실행 시에 Context에 연관된 concrete Strategy 객체에 의해 특별한 알고리즘이 실행되는 패턴입니다.

반면 State는 상태에 관계하는 메소드가 각 Concrete class마다 다르게 구현되어 있고 실행 시에 Context에 연관된 concrete State 객체에 의해 특별한 상태가 표현되는 것입니다.

이미지

n GoF Solution For the State Pattern



(3) State Pattern : Consequences

State Pattern을 적용한 결과

| 상태 특정적 행위를 국소화 시킵니다.

이 패턴의 개발은 아주 간단합니다.

이 패턴은 새로운 상태를 추가하는 것도 쉽습니다.

이 패턴을 적용하면 Context 클래스에서 상태에 대한 특정적 행위에 대한 구현부가 줄어들고 각각 별도의 클래스로 국소화 되어서 표현됩니다.

| 조건절을 감소시킵니다.

Context 클래스 안에서 상태 변화를 얻고 세팅하기 위한 if/else 절을 가질 필요가 없습니다. 이것은 코드의 가독력도 좋게 만듭니다.

| 상태전이를 명백하게 만듭니다.

이 패턴을 사용하지 않으면 객체가 한 상태에서 다른 상태로의 변화를 자세히 기술하지 않을 수도 있습니다.

이 패턴을 사용하면 실행의 어떤 순간에라도 이 객체의 상태를 확인할 수 있으며 그 전이 상태를 명확히 알 수 있습니다.

| 그러나 객체의 수는 증가 됩니다.

이 패턴은 추가되는 서브 클래스가 많습니다.

그러나 이 클래스들이 실행 시에 전부 객체화 되는 것은 아닙니다.

또한 필요한 상태 객체라 할지라도 매번 생성되지 않도록 만들 수 있습니다.(**Singletone pattern** 을 사용하여...)

이것은 이과정의 이슈가 아니므로 다루지 않습니다.

그러나 실행 시 객체의 수를 고려하여야 합니다.

| 상태 객체와 컨텍스트 객체 사이에 오버헤드가 발생합니다.

상태 객체가 컨텍스트 객체의 자세한 정보를 알아야 한다면 오버헤드가 발생할 수도 있습니다. 이를 위해서 상태 객체를 컨텍스트 객체의 **inner class**로 만드는 방법을 사용하기도 합니다. 그러나 이것도 이 과정의 이슈가 아니기 때문에 다루지 않습니다.

심화학습

n divide and conquer

이것은 “분할해서 정복하라”는 프로그램 원리입니다.

복잡하고 규모가 큰 프로그램의 경우 문제를 작게 세분화 해서 풀어나가라는 원리입니다.

State Pattern의 적용 결과 ‘상태 특정적 행위가 국소화 된다’고 했는데 바로 이것이 원리를 따른 결과입니다.

국소화라는 것이 작게 세분화 되었다는 말과 일맥 상통합니다.

상태를 광범위하게 다루려고 하지 않고 ‘어떤 상태’라는 구체적이고도 세분화된 별도 단위로 쪼개서 다루는 것이 바로 이 원리의 핵심인 것입니다.