Introduction to OOAD using UML tools

Team Report

2010 년 10 월 27 일

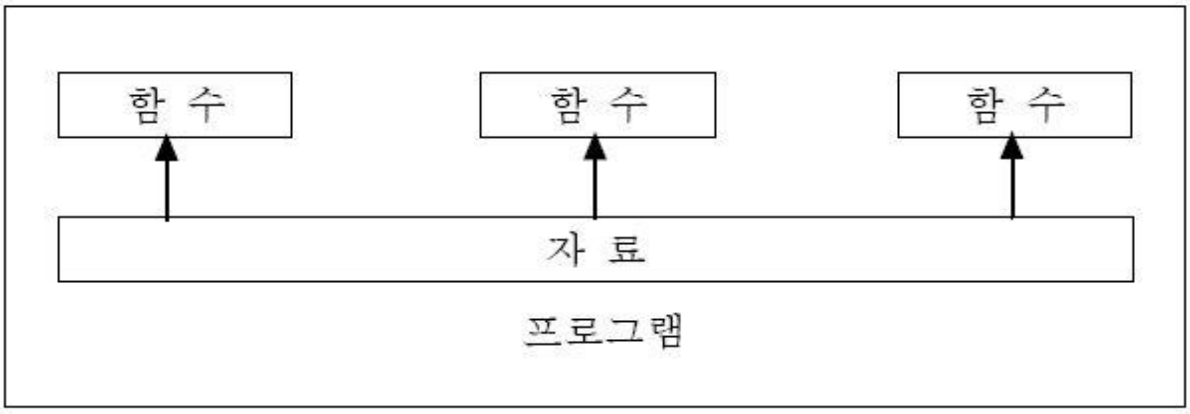
Team 6
200611499 이낙원
200611521 최정명
200911411 이상규
200611520 진경훈

목차

1. 들어가는 말

1) OOAD 란?

최근에 컴퓨터의 이용이 증가함에 따라 소프트웨어에 대한 수요의 증가로 소프트웨어의 개발, 유지보수 비용은 급속한 속도로 증가하고 있다. 품질 좋은 소프트웨어를 개발하기 위한 소프트웨어 방법론이 많이 연구되고 있다. 현재 많이 이용되는 절차중심 개발 방법론은 소프트웨어 재사용이나 모듈화, 개발된 소프트웨어를 유지보수 하는 데는 효과적이지 못하다는 결점을 가지고 있다. 그러므로 적은 규모의 소프트웨어를 개발하는 데는 적합하지만 대규모 소프트웨어를 개발하는 데는 적합하지 못하다.

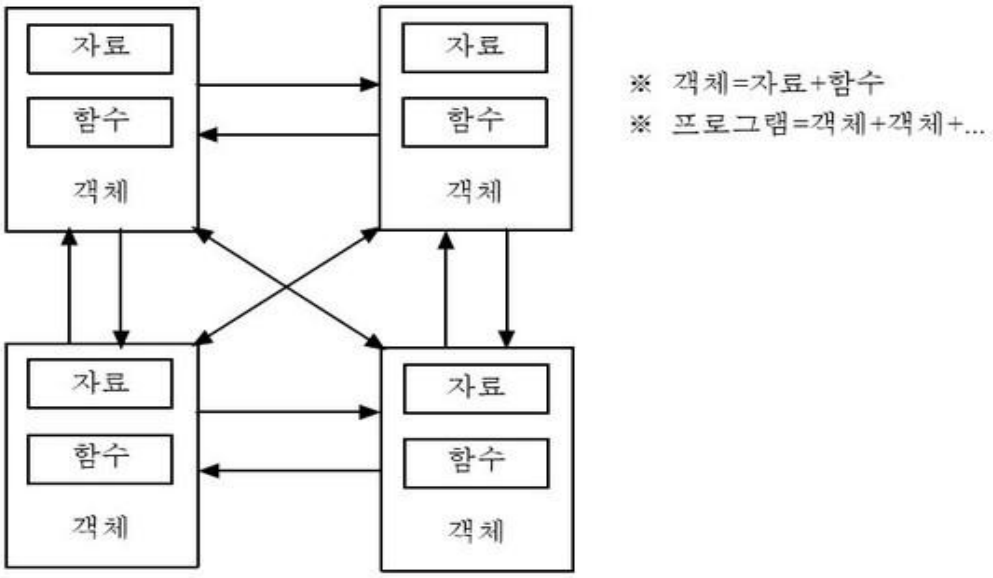


(그림 1-1) 절차중심 개발 프로그램

조직개발 방법론의 대안으로 객체 지향 개발 방법론이 대두되었다. 종래의 구조적 개발 방법론은 소프트웨어의 동적인 기능 측면에 초점을 맞추고 있는데 비해, 객체 지향 개발 방법론은 소프트웨어의 정적인 데이터 측면과 동적인 기능 측면을 모두 중시한 시스템을 개발할 때 요구되는 여러 가지 강력한 모델링 개념과 캡슐화 개념 등을 잘 지원하고 있다. 이러한 객체 지향 방법론은 구조적 개발 방법론에 비해 소프트웨어 분석, 테스트, 유지보수, 확장 등을 쉽게 해주며, 분명하면서도 잘 이해할 수 있는 설계 결과를 만들어 낸다. 즉 클래스는 자연스럽게 소프트웨어 모듈의 한 단위가 되고 객체 지향 설계 과정은 그러한 객체 지향 사이의 복잡도를 적절히 관리해 준다.

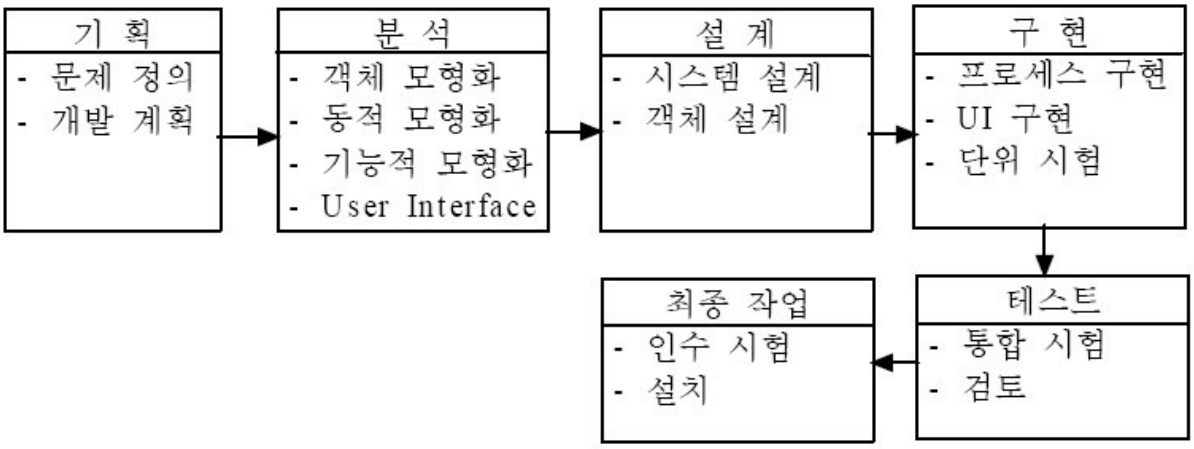
객체지향이란 실 세계의 개체(Entity)를 속성(Attribute)과 메소드(Method)가 결합된 형태의 객체(Object)로 표현하는 개념이다. 이때 시스템은 객체간의 메시지를 주고 받는 형태로 구성한다.

객체지향 개발 방법론이란 객체지향 기법이 적용된 소프트웨어 개발 방법 및 절차, 객체지향 개발 도구 등이 실무적인 관점에서 하나의 체계로 묶어진 것이다. 처음으로 분석을 통해 객체와 클래스를 찾고, 각 클래스의 특성(자료, 함수)을 정의하고, 각 클래스의 특성을 구체화한다(자료구조와 알고리즘 등등).



(그림 1-2) 객체지향 개발 프로그램

객체지향 개발 모형은 기획단계에서 문제를 정의하고 개발 계획을 설정하고, 분석단계에서 요구사항을 분석해서, 객체, 실행동작, 기능을 모형화 하고, 설계단계에서 시스템과 객체를 설계하고, 구현단계에서 프로세스와 User Interface를 구현하고, 테스트 단계를 거쳐 완성하게 된다.



(그림 1-3) 객체지향 개발 모형

객체지향 개발 방법론의 장점은

1. 단일 패러다임 – 하나의 언어(UML)를 사용자, 분석자, 설계자, 구현자들이 사용하기 때문에 상호 이해가 쉽다.
2. 모델들을 보다 현실에 가깝게 반영이 가능하다.
3. 유연하게 아키텍쳐(Architecture)와 코드를 재사용 할 수 있다.
4. 빠르고 쉬운 이해와 유지보수성이 향상된다.
5. 안정성이 증대된다 – 요구변경에 의한 작은 변경은 시스템 내부적으로 큰 변화를 주지 않는다.

2) UML 이란?

객체지향 언어가 등장(1970년대 중반 1980년대 후반 사이) 이후, 객체지향 언어로 소프트웨어 개발을 위한 방법론들이 나오기 시작하고, 1990년대에는 수십 개의 방법론들이 자신들이 최고라고 주장하면서 등장 했다. 하지만 소프트웨어 개발에 있어서 서로 다른 방법론을 사용하는 조직간에 모델 정보를 공유해야 한다면, 사용하는 모든 방법론의 표현기호에 대해 이해해야 하고, 서로 다른 표현 기호들 사이에 mapping 방법을 알아야 한다. 이 때 동일한 개념에 대해 서로 다른 표현들을 읽고, 이들 간의 mapping 방법을 배우는 일은 매우 소모적이다. 또한, 표현 기호 간의 mapping이 100% 정확히 된다는 보장도 없기 때문에 모델 이해 시 의미적인 변질이 생길 수도 있다. 이를 해결하기 위한 방법 중 최선은 동일 표현기호, 즉 동일 모델링 언어를 사용하는 것이다. 이러한 이유로 모델링 언어를 통합하려는 노력을 시작하게 되었고, UML이 탄생하게 되었다.

UML은 소프트웨어 시스템을 개발하는데 쓰이는 시각적 언어로서, 객체지향 시스템을 분석하고 설계하는 사람들이 소프트웨어 시스템의 가공물을 가시화, 구축, 문서화하고, 그 시스템을 사용하는 조직의 업무를 모델링 하는 방법을 제공한다. 우리는 OOAD소프트웨어 개발 방법론에 입각하여 UML을 사용한 개발 과정을 프리웨어 UML툴인 StarUML을 사용하여 설명하였으며, 주요 diagram인 Use Case diagram, Class diagram, Sequence diagram, Collaboration diagram, State Chart diagram, Component diagram, Deployment diagram에 대하여 자세히 알아보았다.

2. OOA

1)요구사항 분석

(1)요구사항 분석이란?

시스템이 필요로 하는 요구사항들을 찾아내고 정의하여 요구사항 명세서를 도출해내는 과정이다. 최종 사용자가 직접 현 시스템의 문제점이나 구축 되어야 할 업무 시스템에 대한 요구사항을 작성하거나, 시스템 분석 설계자가 사용자와 면담, 설문, 기존 시스템 분석을 통해 도출한다. 구축 되어야 할 업무 시스템의 영역이 명확히 정의되어야 하며, 사용자가 이를 검증해야 한다.

(2) Use case diagram

a. 정의

사용자 시각에서 소프트웨어 시스템의 범위와 기능을 설명하고 정의한 모델로서, 소프트웨어 시스템과 연관된 모든 사람들이 이해할 수 있는 방법으로 requirement를 표현한다.

b. 사용목적

프로젝트의 시작인 요구사항 분석 단계에서 필수적으로 사용되는 diagram이다. 소

프트웨어 프로젝트의 개발범위를 정의할 때, 요구사항을 정의할 때, 세부기능 분석 및 소프트웨어가 아닌 업무영역을 이해하고 분석하기 위해 사용된다. 시스템 사용자, 업무 영역 전문가, 시스템 개발자 간의 의사소통 수단을 제공하며 "누가 시스템을 사용할 것인가?", "시스템이 사용자를 위해 무엇을 해야 하는가?", "사용자와 상호작용을 위해 시스템이 제공해야 하는 인터페이스는 무엇인가?" 에 대한 사항을 파악하는데 유용하다.

c. 사용법

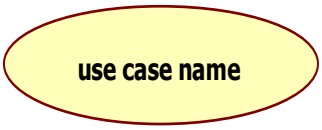
• Actor

시스템을 사용하는 어떤 대상으로 사람 혹은 다른 시스템이 될 수 있다. 행위자의 이름은 사람형태의 그림 아래에 위치한다. 시스템 사용자의 역할을 의미하며 반드시 시스템 외부에 존재한다.

(그림2-1) Actor

• Use case

시스템이 제공하는 서비스 혹은 기능을 의미한다. 시스템이 actor에게 제공하는 기능단위이며, actor와 상호작용한다. 사용자의 관점에서 정의가 필요하며 따라서 "출력 미리 보기, 문서출력" 같은 최종적인 기능 보다는 "출력 허용" 과 같이 일반적으로 표현한다. 타원 안에 use case의 이름을 적는다.



(그림2-2) Use case

• Relationship

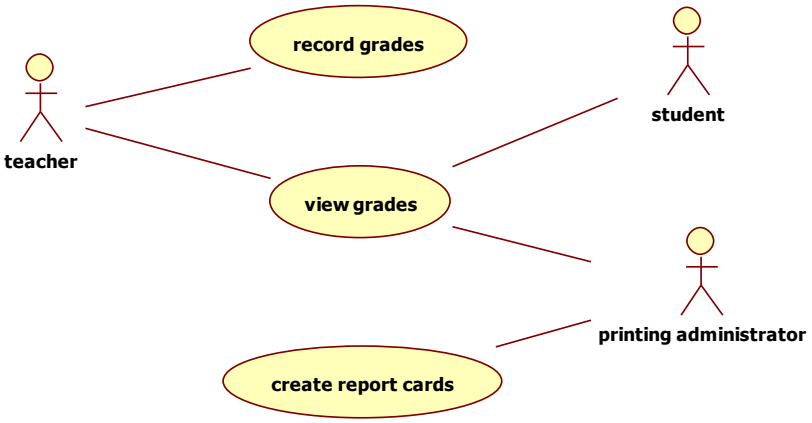
actor와 actor, use case와 use case, 또는 actor와 use case간에 존재하는 관계를 설명하기 위한 표현 방법이다.

Communication

actor와 use case간에 정의되는 관계이다. 일반 상호작용 관계가 존재함을 의미하며, actor는 정보를 요구하거나 통보 받고, use case는 정보를 제공한다. 일반 실선

으로 표시하면 상호 교류 관계, 화살표로 표시되면 한쪽이 커뮤니케이션을 받음을 의미한다. 다중의 actor는 하나의 use case에 연결될 수 있으며, 다중의 use case가 하나의 actor에 연결될 수 있다.

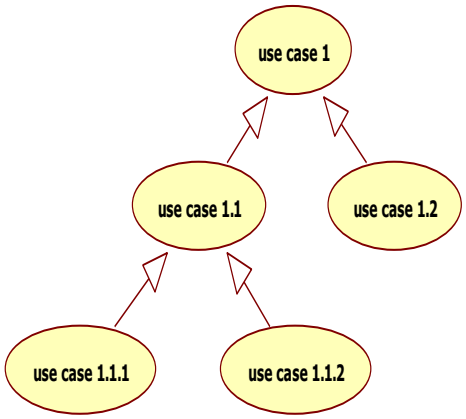
(그림2-3) Communication



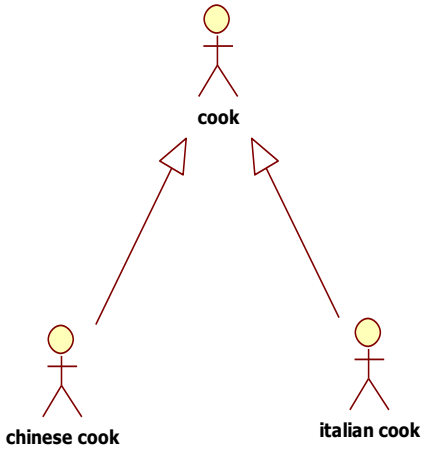
(그림2-4) 다중 actor와 다중 use case관계

Generalization

actor와 actor, use case와 use case사이에 정의 된다. 상속성을 표현하는 기법으로 보다 보편적인 것과 구체적인 것 사이의 관계로 표현된다(is-a관계).



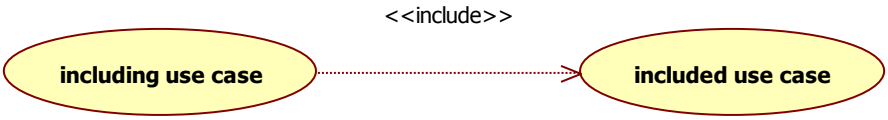
(그림2-5) Use case의 일반화



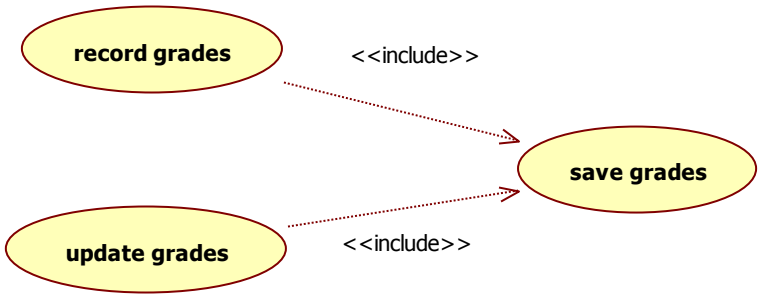
(그림2-6) Actor의 일반화

Include

use case와 use case사이에 정의된다. 한 use case가 자신의 서비스 수행 중에 다른 use case의 서비스 사용이 필요할 때 정의된다. 서비스가 반드시 이용 되어야 한다. 포함 되는 use case는 공통 서비스를 가진 존재이다.



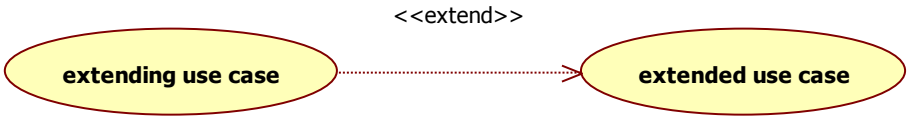
(그림2-7) Include



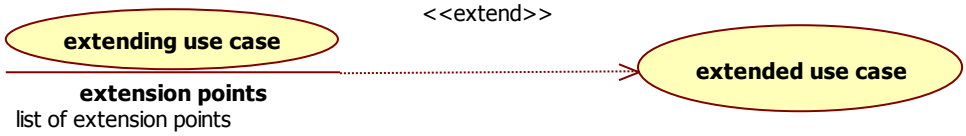
(그림2-8) Include의 예시

extend

include관계와 동일하게 서비스 수행을 요청하지만 include관계와는 달리 서비스가 수행되지 않을 수 있다. 수행 요청 조건을 extension point라고 한다.



(그림2-9) Extend



(그림2-10) Extension points

d. 작성순서

ⓐ 시스템의 actor식별

모든 사용자의 역할, 상호작용하는 타 시스템, 정보를 주고받는 하드웨어 장치를 식별한다.

ⓑ use case식별

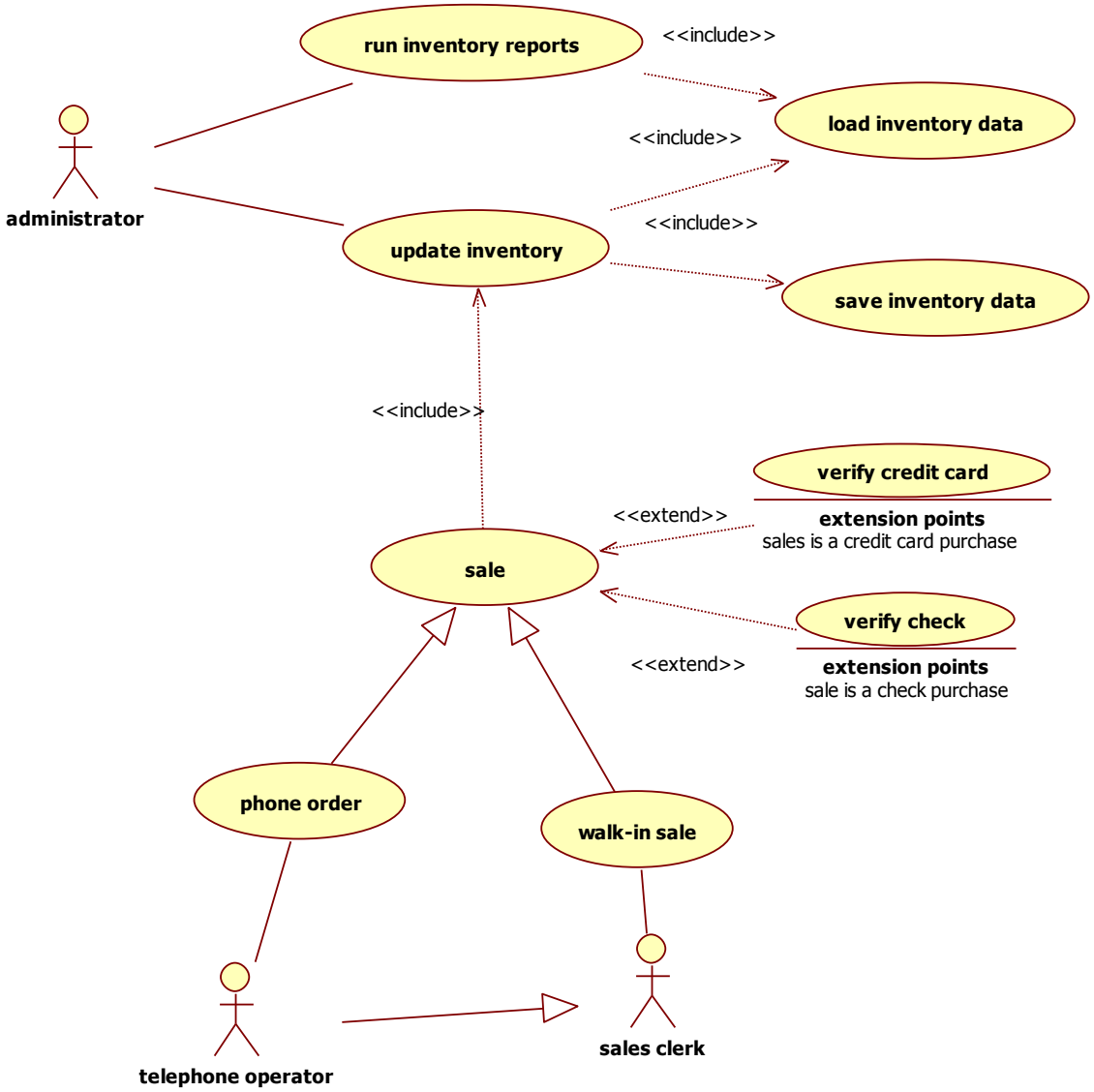
actor가 요구하는 서비스, actor가 요구하는 정보, actor가 시스템과 상호작용하는 행위를 찾는다.

ⓒ 관계의 정의

actor와 actor간의, actor와 use case간의, use case와 use case간의 관계를 분석한다.

ⓓ use case diagram 구조화

다수의 use case에 존재하는 공통 서비스를 추출하고 추출된 서비스를 use case로 정의한다. 포함, 확장, 일반화를 통해 적절한 모형화를 수행한다.



(그림2-11) Use case diagram의 사용 예

(3) Activity diagram

a. 정의

activity diagram은 state chart의 특별한 종류이다. 처리 로직이나 조건에 따른 처리 흐름을 순서에 따라 정의한 모델. 사용자에게 시스템 실행의 예측과 여러 조건과 자극에 대하여 시스템이 어떻게 반응하는지 보여준다.

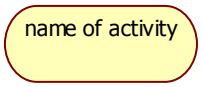
b. 사용목적

activity diagram은 사용자에게 시작과 끝을 명확하게 보여주는 특성 때문에 use case의 작업 흐름을 모형화하거나, use case내부의 작업경로를 표현하는데 유용하다. 모델링 과정을 통해 새로운 use case의 발견 또한 가능하다. 주로 비즈니스 프로세스의 정의 목적으로 가장 많이 사용되며, 처리 순서의 표현, 프로그램 로직 정의, use case 실현을 위해 사용한다.

c. 사용법

Activity

activity는 모퉁이가 둥근 사각형으로 표현되며 activity수행 또는 이벤트 흐름의 단계를 표시한다. 이름을 정할 때는 발생하는 행위를 정확히 묘사하는 단어를 선택한다.



(그림2-12) Activity

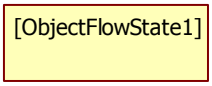
• State

현재 도달된 상태를 표시하는 단어나 구로서 식별한다. 시작상태는 속이 꽉 찬 점으로, 끝 상태는 검은 점 주위에 원이 그려진 형태로 그려진다. 각 activity diagram은 시작 상태는 하나만 가질 수 있으나, 종료상태는 여러 개를 가질 수 있다. 각 상태에 또한 레이블을 붙일 수 있다.

(그림2-13) start state와 end state

• Object flow

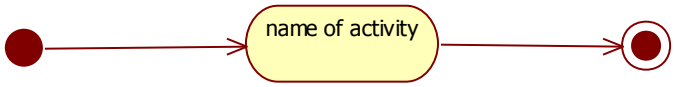
State 간에 전달되는 자료를 객체로 표현하고 전달 방향을 객체 흐름으로 표현한다.



(그림2-14) Object flow

• Transition

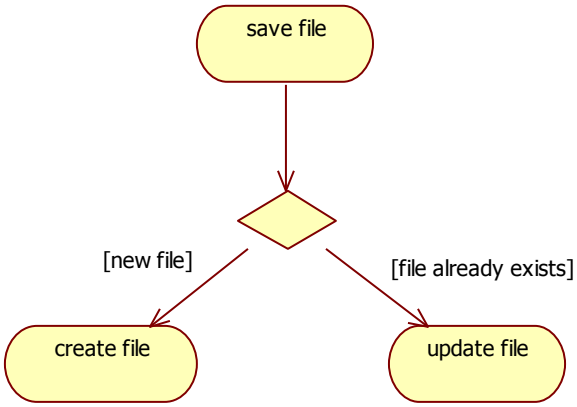
하나의 state나 activity에서 다른 상태로 제어의 흐름을 나타내는 선이다. 방향성을 나타내기 위해 열린 화살표를 이용 할 수도 있다.



(그림2-15) Transition

• 조건, 선택점

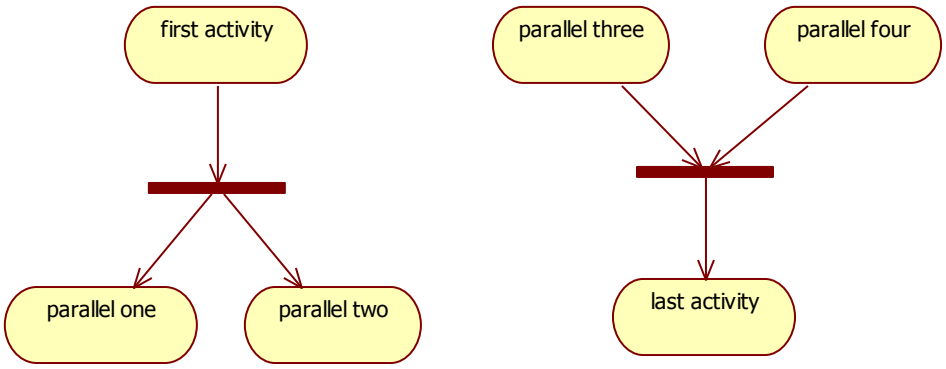
다이아몬드 형태로 표현되는 조건 선택점은 흐름의 분기를 위해 사용된다. 특정 조건에 의한 시점에 사용되며, 이때 각 조건은 [ ]로 감싸 각 활살표에 표시해 준다. 아래의 예시와 같이 파일을 저장할 때 새로운 파일명이면 새 파일을 만드는 activity로, 이미 파일명이 존재하면 기존의 파일을 업데이트 하는 activity로 이동하는 방식이다.



(그림2-16) 조건, 선택점

• Synchronization bar

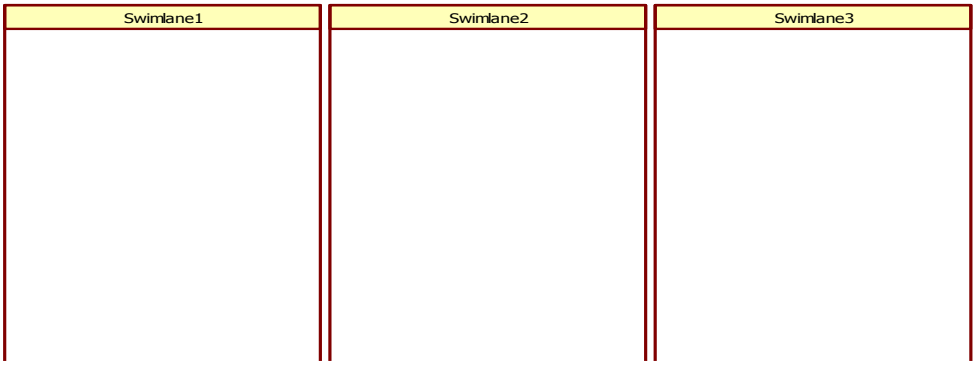
처리 과정 중에 특정 활동이 동시에 같이 실행 되거나, 동시에 실행 중이던 활동이 하나로 모이는 경우가 발생한다. 이때 동시경로를 사용하며 긴 직사각형의 선으로 나타낸다. 분기는 하나의 입력에서 둘 이상의 여러 출구를, 연결은 둘 이상의 입력에서 하나의 출구를 가진다.



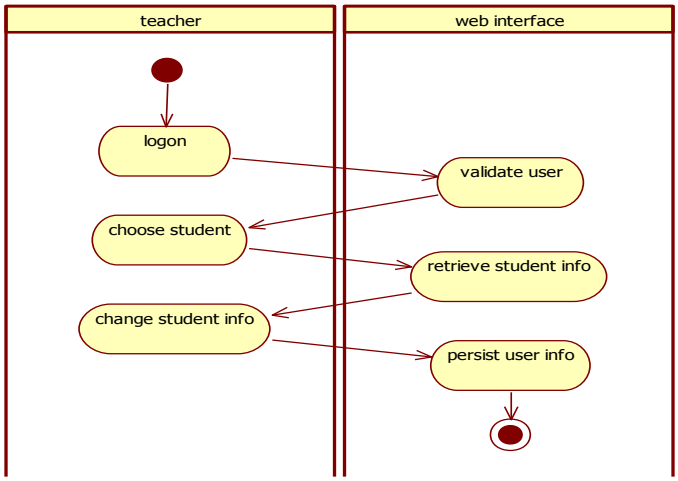
(그림2-17) 동시경로

• Swim lane

활동 다이어그램의 가독성을 증가시키며 누가 그 활동을 하는가를 명확하게 구분할 수 있는 방법을 제공한다. diagram을 여러 구획으로 나눠 각 구획 상단에 객체 이름이나, 도메인 명, 즉 활동의 주체를 표시한다. 유영경로는 반드시 표현하지는 않아도 되나, 각 수행역할을 책임지는 개체를 알고 싶다면 유용한 표기 방법이 된다.



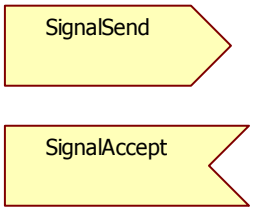
(그림2-18) 유영경로



(그림2-19) 유영경로 표현 예

• Signal

자주 사용되는 요소는 아니지만, 각 활동이 처리되는 과정에서 신호를 보내고, 받는 과정이 가능하다. 비 동기적인 흐름을 나타내고 싶을 경우, 각 활동(단계) 간의 이동 중 발생하는 상황을 보다 명확히 하고 싶을 경우에 사용 가능하다. signal은 발신과 수신으로 오각형의 형태로 표현한다.



(그림2-20) Signal

d. 작성순서

ⓐ activity diagram에 요구되는 use case를 식별한다.

앞 단계에서 작성된 use case diagram에서 무엇을 모형화 할지 결정해야 한다.

ⓑ 각 use case를 위한 주요 경로를 모형화한다.

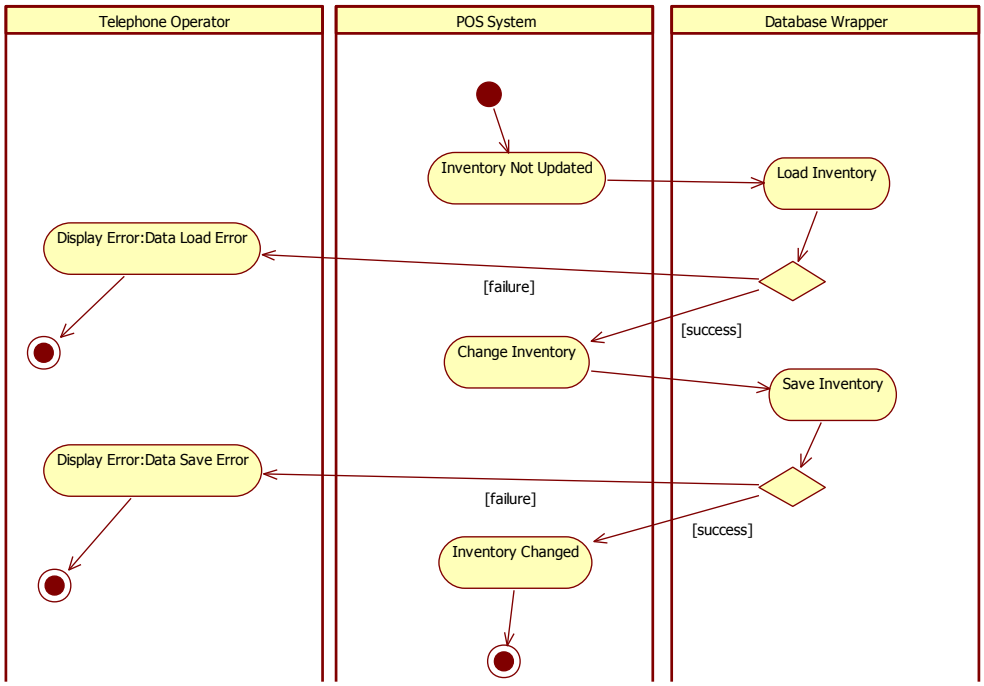
선택된 use case의 가장 분명한 경로, 완수하기 가장 쉬운 주 경로를 모형화한다.

ⓒ 각 use case에 대한 대안경로를 모형화한다.

주 경로 이외의 오류 제어나, 다른 활동들을 수행할 경로들을 모형화한다.

ⓓ 유영경로를 추가한다.

가독성을 높이기 위해 유영경로를 추가한다.



(그림2-21) Activity diagram 사용 예

2) Use case Analysis

(1) 목적

use case안의 서로 연관되어 작용하고 있는 객체들을 추출하고 객체들간의 주고 받는 메시지를 정의한다.

(2) 클래스와 객체 (Class & Object)

a. 정의

객체

현실 세계에 존재하는 모든 것. 특정 domain에서 명확한 범위와 의미를 가지는 실체. 구현을 위한 실질적 기반을 제공한다.

클래스

객체를 생성하기 위한 설계도의 개념으로 객체의 클래스의 정의에 따라 객체가 구현된다.

b. 특징

객체

상태와 행위를 가진다. 상태는 객체가 가지는 속성의 구체적인 값을 의미하며, 행위는 객체의 행동방식, 다른 객체의 요청에 대한 반응으로 표현된다.

클래스

객체의 상태를 나타내는 변수와 행위를 나타내는 메소드의 선언을 포함한다.

추상화

추상화는 객체 지향의 일반적인 특성으로 객체는 추상화의 대상이며 일반화된 형태로 표현된다. 일례로 자동차는 스포츠카, 트럭, 버스, 승용차등 다양한 종류가 있으나 모두 공통적인 속성 바퀴, 의자, 핸들, 엔진 등을 가지고 있으며, 운행 주차 등의 동작을 가지고 있다. 이런 공통의 특성에 대하여 추상화가 가능하다.

캡슐화

블랙박스는 기능성을 가지고 있지만 그 기능을 수행하는 세부사항을 외부에 공개하지 않는다. 캡슐화는 이와 유사한 개념으로 객체의 속성은 외부에 대하여 숨겨져 있고 오로지 입력에 대한 결과물 만을 다른 객체에 공개하는 개념이다. 이런 캡슐화는 소프트웨어를 사용하기 쉽고 읽기 쉽게 만드는 역할을 한다. 캡슐화를 적용하지 않은 설계 시 모든 것이 쉽게 노출된다. 프로그램의 규모가 커지고 함수간 의사소통의 제어가 어려워지면 이런 노출된 사항에 접근 함으로서 전체 로직을 무시하게 되고 이런 특성은 소프트웨어의 갱신을 어렵게 만든다. 이렇게 엉킨 코드를 "스파게티 코드"라고 부른다.

상속

상속은 여러 클래스가 같은 인터페이스를 가지면서 각각 자신만의 능력 또한 가질 수 있게 만드는 것으로, 코드의 재 사용성을 높인다. 이런 기능은 코드의 크기를

줄이고 명확하게 하는 장점을 가진다.

(3) Sequence diagram

a. 정의

순차 다이어그램은 두 종류의 interaction diagram중 하나이다. 문제 해결에 필요한 객체를 정의하고, 객체간의 동적인 상호관계를 시간 순서에 따라 정의한다.

b. 사용목적

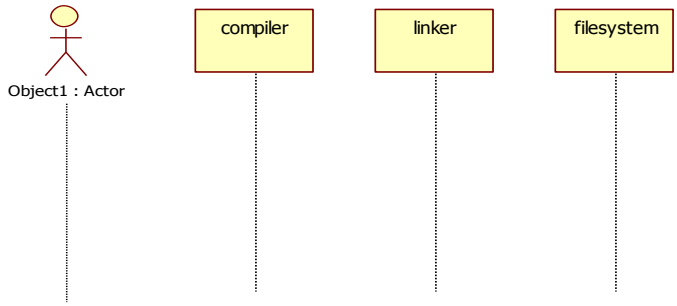
객체간의 동적 상호작용을 표현하는 것이 주된 목적이다. 다소 차이는 있지만 activity diagram과 유사한 역할을 수행한다. 경로뿐만 아니라 시스템 연산의 기능성에 대한 상위 수준의 이해를 돕는데도 이용된다. use case중 일부는 순차 다이어그램에 의해서 더 명확해지고 분명해진다.

c. 사용법

각각의 use case별로 작성한다. 하나의 use case에서 다양한 이벤트의 흐름 별로 작성한다. actor또한 이 다이어그램에 포함될 수 있다. 다이어그램의 위에서부터 아래로 진행되면서 시간의 흐름이 표현된다.

• Activity object

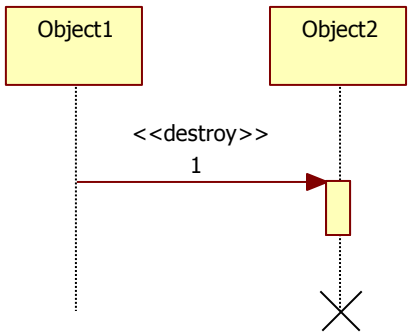
클래스의 정의와 유사하게 직사각형에 객체의 이름을 적고 밑줄을 그어서 표현한다. 각 객체들은 다이어그램 상단에 왼쪽에서 오른쪽으로 배치된다. actor또한 객체와 같은 위치에 표현되어 메시지를 주고 받을 수 있다.



(그림2-22) Sequence diagram 객체 표현

• Life line

각 활성객체 아래에 긴 점선을 그리는데 이것이 생명선이다. 생명선은 객체가 존재하고 있음을 의미하며, 상호간의 교류를 위해 이용된다. 생명선의 끝에 X자를 붙이면 그 시점에 객체가 소멸됨을 의미한다.



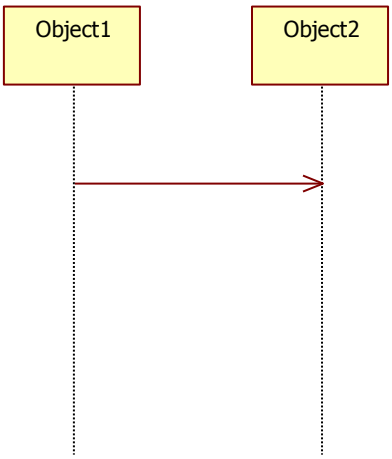
(그림2-21) Life line

• Message

순차 다이어그램의 서로 다른 활성 객체간의 의사소통을 묘사하는데 이용된다. 객체가 다른 객체의 처리를 유도하거나, 다른 객체에게 정보를 제공할 때 사용된다. 메시지는 한 객체의 생명선에서 다른 객체의 생명선까지의 화살표로 이루어진다. 각 화살표 위에 보내는 메시지를 표시한다.

Flat message

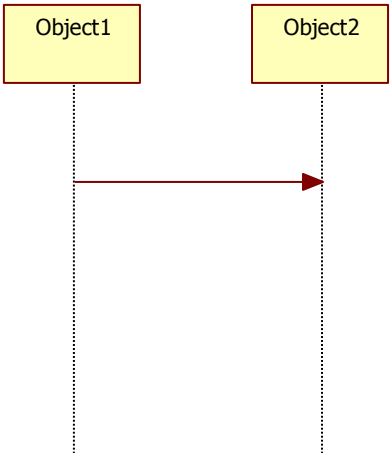
가장 일반적인 메시지로 동기와 비 동기 구분이 없으며 열린 화살표와 실선으로 표시된다.



(그림2-22) Flat message

Synchronous message

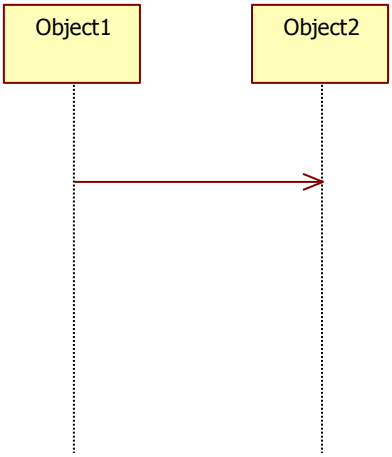
메시지의 중첩 시, 메시지가 모두 돌아와야 다음 처리를 진행한다. 안이 채워진 화살표와 실선으로 표시한다.



(그림2-23) Synchronous message

Asynchronous message

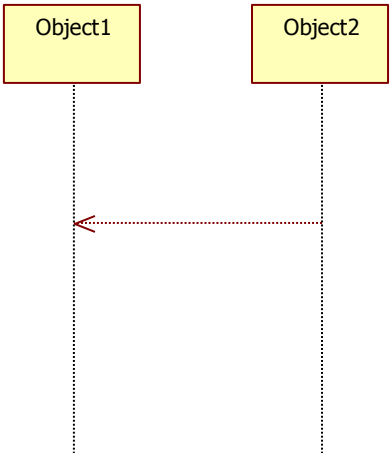
활성 객체가 응답을 기다리지 않고 전송되는 메시지이다. 반이 열린 화살표와 실선으로 표시된다. StarUML에서는 flat message와 같이 표현한다.



(그림2-24) Asynchronous message

Return message

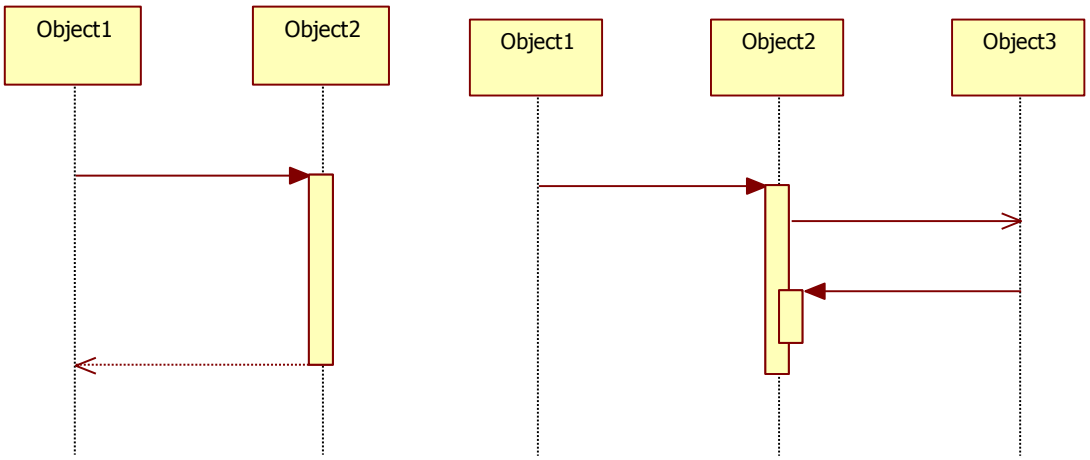
제어흐름이 호출했던 활성화 객체로 반환됨을 보여준다. 이는 동기 메시지가 그 임무를 다 했음을 호출한 객체에게 전달하는 것이다. 반환이 필요한 경우에 사용된다.



(그림2-25) Return message

• Activation

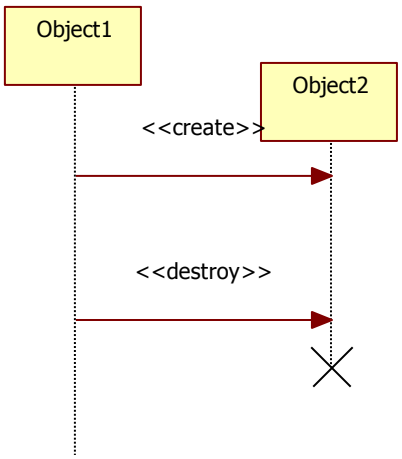
객체가 활성화 되어 있음을 의미한다. 생명선에 길고 좁은 직사각형으로 표시한다. 생명선에 이 표시가 위치하는 시간 동안 메시지 순차에 참여하고 있음을 표현한다. 아래와 같이 activation 내에 또 다른 activation을 위치시키는 것도 가능하다.



(그림2-26) Activation

• Object의 생성과 소멸

객체는 반드시 가장 상위에만 위치하는 것은 아니다. 필요에 따라 순차 다이어그램의 흐름 중에 생성도 가능하다. 이렇게 생성된 객체 또한 생명선을 가지며 소멸되기까지 역할을 수행한다.



(그림2-27) Object생성과 소멸

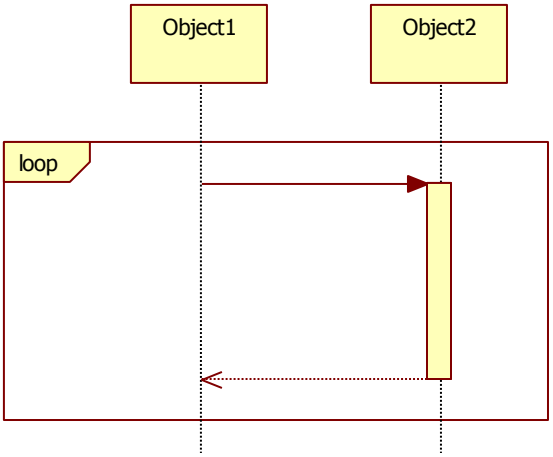
• 루프, 조건문 등

루프는 프레임을 통해 표현한다. 반복이 수행되는 부분을 프레임으로 묶고 프레임의 왼쪽 상단공간에 loop, alt, opt 등의 operator를 적는다. 아래 그림은 프레임에 들어가는 여러 operator에 대한 설명이다.

Table 4.1 *Common Operators for Interaction Frames*

Operator	Meaning
alt	Alternative multiple fragments; only the one whose condition is true will execute (Figure 4.4).
opt	Optional; the fragment executes only if the supplied condition is true. Equivalent to an alt with only one trace (Figure 4.4).
par	Parallel; each fragment is run in parallel.
loop	Loop; the fragment may execute multiple times, and the guard indicates the basis of iteration (Figure 4.4).
region	Critical region; the fragment can have only one thread executing it at once.
neg	Negative; the fragment shows an invalid interaction.
ref	Reference; refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	Sequence diagram; used to surround an entire sequence diagram, if you wish.

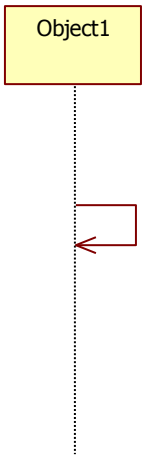
(그림2-28) Interaction frame operator



(그림2-28) 프레임을 이용한 loop표현

• 재귀호출

메시지를 보내는 생명선이 다시 받는 형태로 표현된다.



(그림2-29) 재귀호출

d. 작성순서

ⓐ 모형화할 작업 흐름도를 결정한다.

use case를 선택하고 use case의 이벤트의 흐름을 찾는다.

ⓑ 왼쪽에서 오른쪽으로 객체를 배치한다.

ⓒ 각 작업흐름도를 생성하기 위해서 메시지와 조건들을 포함시킨다.

오류조건을 포함하지 않는 기본적인 작업 흐름부터 모형화해 나간다. 적절한 메시지 유형의 선택이 필요하다.

ⓓ 시간과 반복의 모형화, 메시지 설명 추가

활성시간, 반복문의 모형화 각 메시지에 설명을 추가하는 등 세부적 사항을 적용

한다.

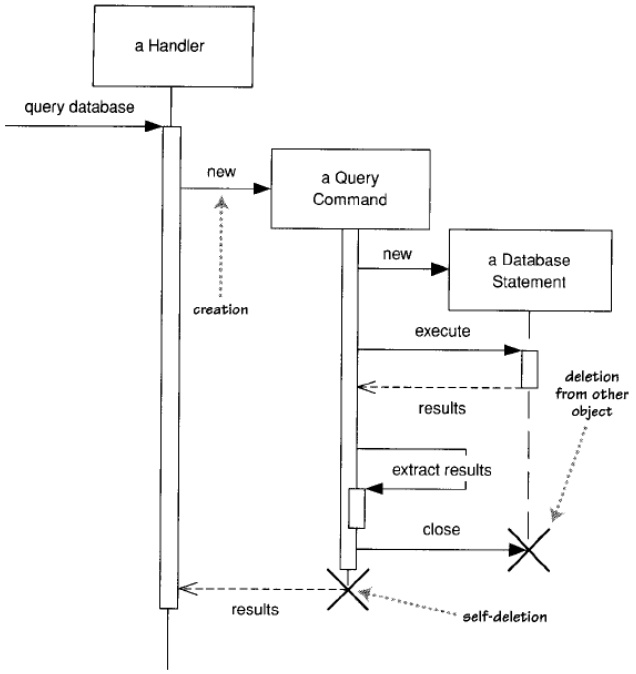


Figure 4.3 *Creation and deletion of participants*

(그림2-30) Sequence diagram 사용 예

```
procedure dispatch
  foreach (lineitem)
    if (product.value > $10K)
      careful.dispatch
    else
      regular.dispatch
    end if
  end for
  if (needsConfirmation) messenger.confirm
end procedure
```

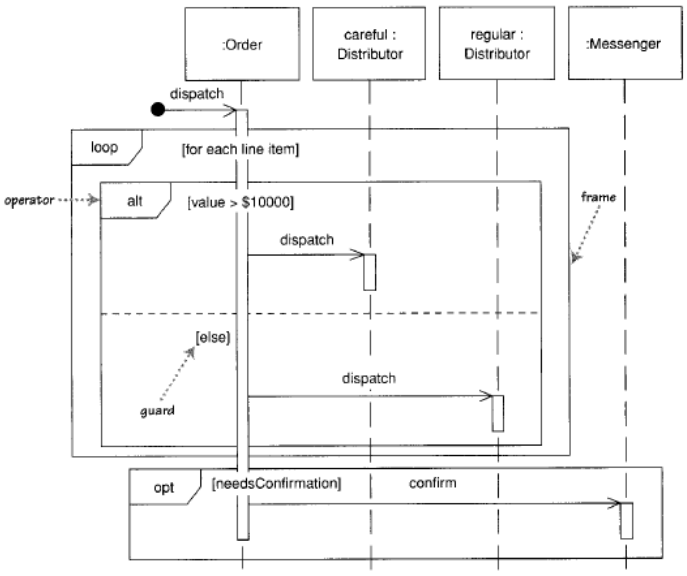


Figure 4.4 *Interaction frames*

(그림2-30) Sequence diagram 사용 예2

(4) Collaboration diagram

a. 정의

sequence diagram과 함께 interaction diagram의 한 종류이다. collaboration diagram은 문제해결에 필요한 객체를 정의하고 객체간 동적 상호관계를 시간순서에 따라 정의해놓은 것이다.

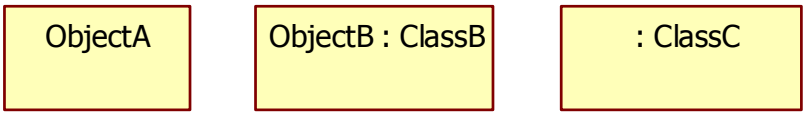
b. 사용목적

collaboration diagram의 목적은 객체간 동적 상호작용을 구조적 측면을 중시하여 작성하고, 객체를 더욱 상세히 정의하고, use case를 실현하고, 프로그래밍 사양을 정의 하기 위해 사용한다. 모델링 공간에 제약이 없어 구조적인 면을 중시할 수 있다. collaboration diagram과 sequence diagram의 차이점은 sequence diagram은 시간에 따른 행위의 흐름에 중점을 두고 표현하지만, collaboration diagram은 객체들 사이의 정적인 구조에 더 중점을 두고 있다.

c. 사용법

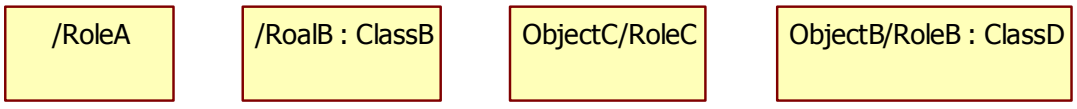
• Object & Role

object instance의 표현방법은 아래와 같이 맨 왼쪽부터 객체의 클래스를 알 수 없거나 중요하지 않은 객체로 취급됨을 의미하는 형태, 객체 이름과 클래스 이름을 포함하는 유형, 객체 이름이 명명되지 않은 형태가 있다.



(그림2-30) Object의 표현

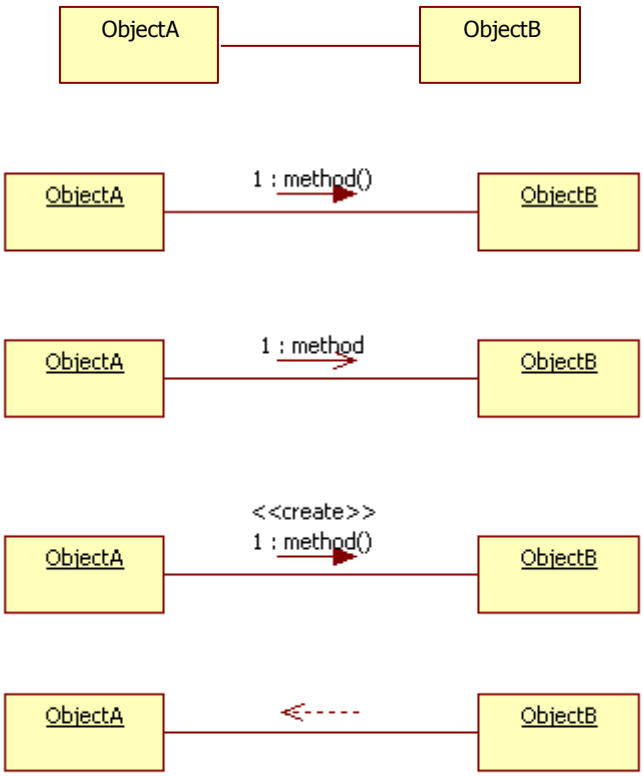
아래는 각 instance가 수행하는 역할을 표현하는 형태로 객체 명 뒤에 역할을 표기 함으로서 나타낸다.



(그림2-31) 역할 표현의 추가

• Link

객체를 연결하며 이 연결에 메시지를 추가 함으로서 객체간 이벤트의 흐름을 표현한다. sequence diagram과 마찬가지로 flat, synchronous, return 등의 메시지 표현이 모두 가능하며, 메시지 앞에 ID번호를 붙여서 메시지의 순서를 표현할 수 있다.



(그림2-32) Link와 message의 표현

d. 작성순서

ⓐ diagram에 속하는 구성요소들을 식별한다.

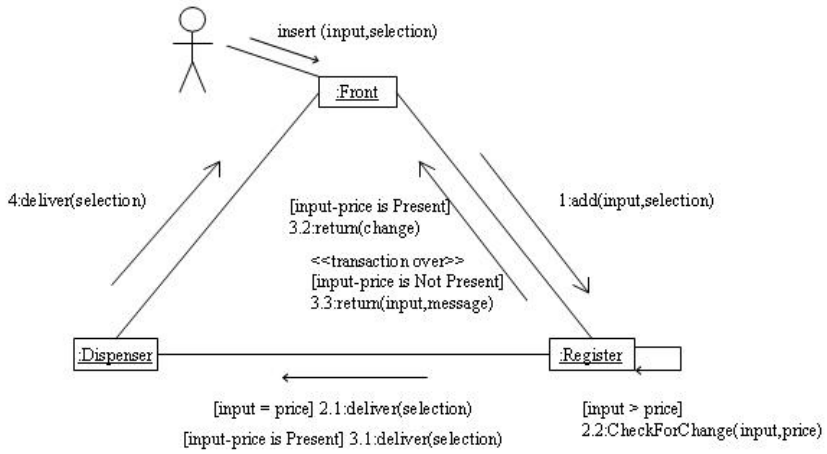
어떤 class들이 필요한지 구성요소들을 찾는다.

ⓑ 각 요소간의 구조적 관계를 모형화 한다.

이 구성요소들 간의 관계를 찾아 link로 연결한다.

ⓒ instance 수준의 diagram을 모형화한다.

각 link에 구체적인 massage를 표기하여 모형화 한다.



(그림2-33) Sequence diagram 사용 예

(5) 분석 클래스

a. 정의

시스템 내부에서 책임과 행위를 가지는 것들에 대한 개념적 모델이다.

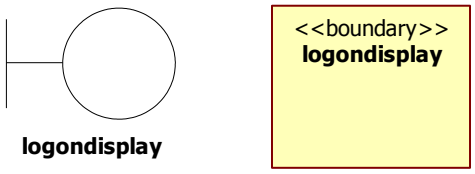
b. 사용목적

object들 간의 상호작용을 표현한 diagram을 통해 클래스를 추출하고 이렇게 추출한 클래스를 모형화하기 위해 사용한다.

c. 사용법

• Boundary class

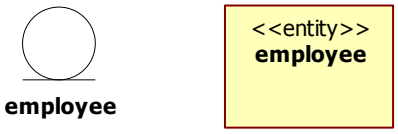
사용자나 시스템, 하드웨어의 인터페이스를 나타내는 클래스이다.



(그림2-34) Boundary class

• Entity class

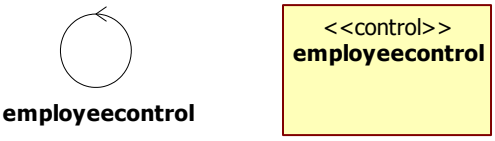
오랫동안 지속되는 정보를 저장하는 역할을 하는 클래스. 시스템의 key concept이다.



(그림2-35) Entity class

• Control class

다른 클래스를 제어하는 클래스이다. 이벤트 간의 순서 같은 논리적 제어를 하고, 트랜잭션을 처리하고, 여러 개 entity class의 내용을 사용하거나 변경하는 클래스이다.



(그림2-36) Control class

d. 작성순서

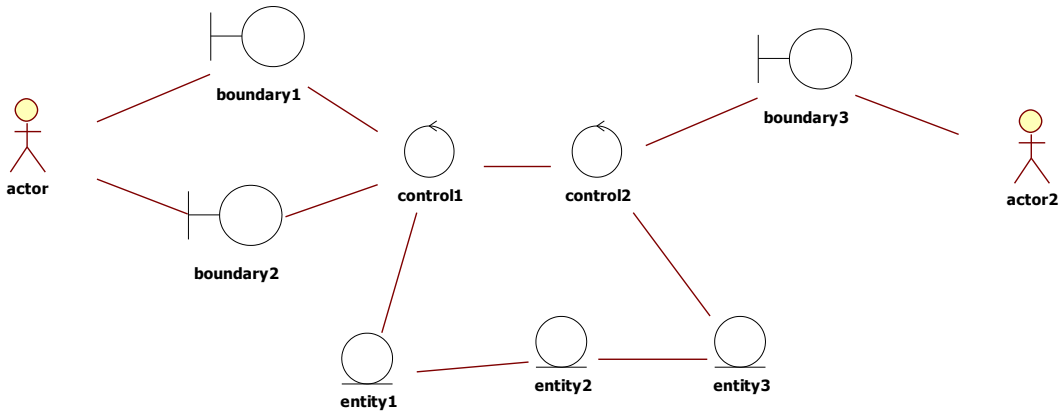
ⓐ actor와 scenario를 비교, 하드웨어를 포함한 시스템들의 의사소통을 파악해서 boundary class를 추출한다.

ⓑ scenario에서 명사구 중심으로 판별하여 entity class를 추출한다.

찾은 객체들이 유사한 구조나 행위를 가지는지 파악하여 그룹핑한다.

ⓒ control class는 use case마다 하나 이상 존재한다.

ⓓ actor는 boundary class를 통해 시스템과 상호작용하고, boundary class는 control class를 통해 entity class에 접근한다.



(그림2-37) 분석 클래스의 예

3. OOD (Object-Oriented Design)

1) Architecture Design

(1) Architecture Design 이란?

Architecture Design은 Preliminary Design(예비설계) 또는 상위 수준 설계(high-level design)이라고 부른다. 시스템의 전체 구조, 구성 요소물 및 동적 행위물을 결정하고 구조, 행위, 인터페이스 설계를 담당한다.

설계시 설계 기준이 되는 것은 다음과 같다.

절차기반 분해 방법	소프트웨어에 요구된 기능, 자료 처리 과정, 알고리즘 등을 중심으로 시스템을 분해하여 설계하는 방법
자료위주 분해 방법	소프트웨어에서 저장 및 처리되어야 할 자료의 구조들을 중심으로 시스템을 분해하여 설계하는 방법
객체지향 분해 방법	자료와 자료에 요구되는 오퍼레이션들을 분리하지 않고 하나의 모듈단위로 보고 시스템을 분해하여 설계하는 방법

(표 3-1) Architecture Design 기준

Architecture Design시에 중요한 부분은 추상화와 상세화, 결합도와 응집도를 이해하는 것이다.

(표 3-2) Architecture Design의 특징

추상화	다른 몇 가지 성분의 집합에서 상세한 개개의 성분보다 단순한 공통적 개념을 생성하는 활동
상세화	보다 일반적인 성분의 가능한 실현으로 상세한 부분으로 요구하거나 선택하는 과정
결합도	한 module(모듈)과 다른 모듈간의 상호의존도 또는 두 모듈 사이의 연관도의 관계를 말한다. 모듈이란 상세화이다. 전체 시스템을 구성하는 하나의 단위로 이 모듈 상호간에 낮은 결합력을 갖는 것이 바람직하다. 모듈의 낮은 결합도는 체계가 잘 분할되어 서로 관계없는 모듈은 분리되어 존재한다는 의미가 된다. 낮은 결합도의 장점은 한 모듈내의 에러가 다른 모듈에 영향을 미치는 파급효과의 최소화가 가능하며, 한 모듈의 변경이 다른 모듈에 큰 영향을 미치지 않고 모듈의 유지보수 및 변경이 가능하다. 또한, 특정모듈의 내부 사항을 자세히 알지 못해도 그 와 관련된 다른 모듈의 효과적 취급이 가능
응집도	한 모듈 내부의 처리 요소들간의 기능적 연관도를 나타내며 모듈 내부요소는 명령어, 명령어의 모임, 호출문, 특정 작업 수행 코드 등 체계를 모듈 단위로 얼마나 잘 분할 하는지를 알려주는 지침이 되며 모듈간의 결합도를 최소함으로써 체계내의 모든 모듈의 양호한 응집도를 성취

(2) Subsystem & Package

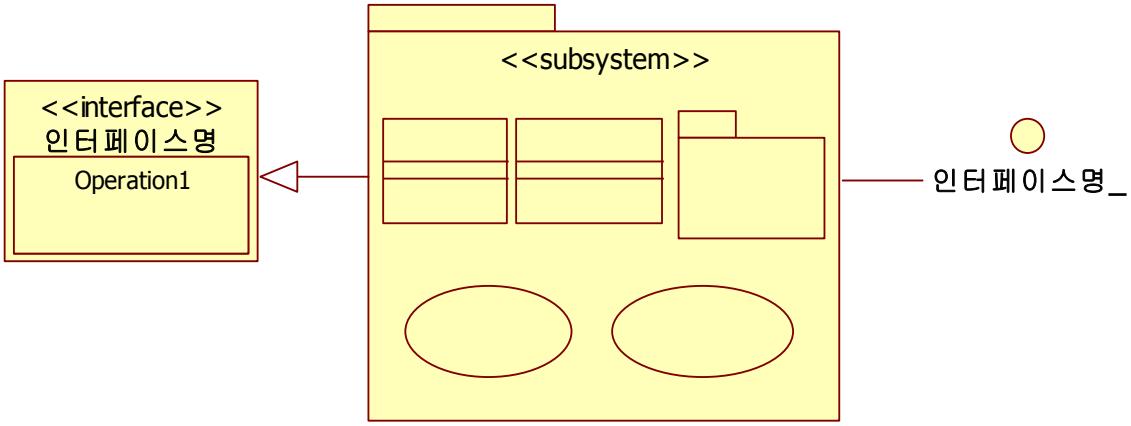
a. 정의

Subsystem - 물리적인 시스템 내의 행위적인 단위를 나타낸 것이다.

Package - 관련 있는 모델요소들을 분류하기 위한 Mechanism이다.

b. 특징

Subsystem - 인터페이스(Interface)와 오퍼레이션(Operation)을 제공하고 재사용 컴포넌트, 외부시스템, 공통 유틸리티 등을 표현 한다.

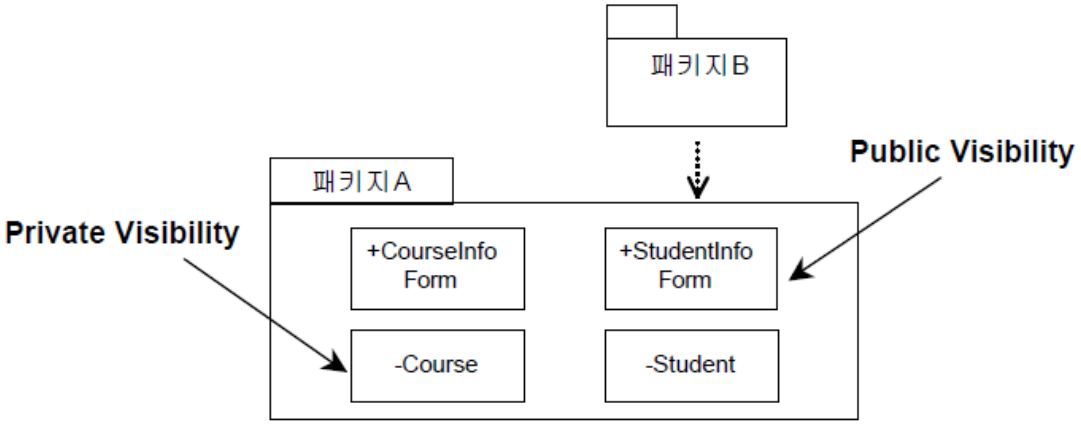


(그림 3-1) Subsystem

Package – 패키지 안에 다른 패키지가 포함 가능하고 패키지 간의 관계와 패키지의 Visibility를 표현한다.

Package Visibility

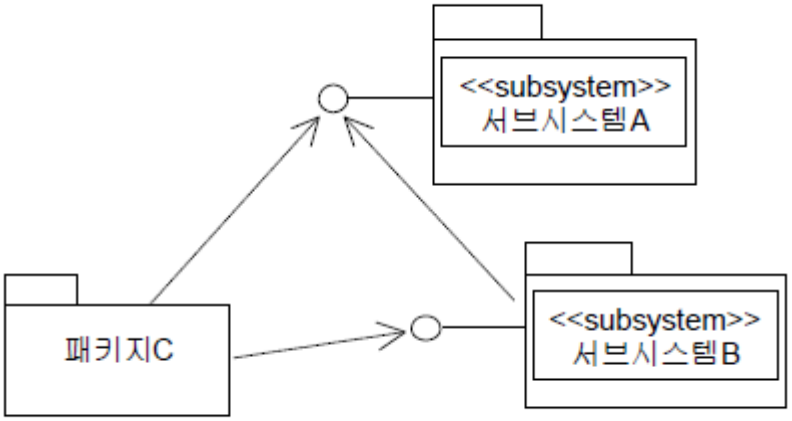
- Public : 다른 패키지의 모델 요소에서 참조 가능
- Protected : 패키지의 모델 요소를 상속받는 서브 패키지의 모델요소에 의해서만 참조 가능
- Private : 패키지 내 모델 요소에 의해서만 참조 가능



(그림 3-2) Package

c. Subsystem과 Package

- Subsystem은 행위를 포함하고 인터페이스를 갖는다.
- Package는 단순히 모델 요소들을 의미상으로 그룹화 한 것이다.



(그림 3-3) Package & Subsystem

2) Class Design

(1) Class Design 이란?

(2) Class diagram

a. 정의

Class Diagram은 객체지향 시스템에서 시스템의 기초가 되는 클래스를 보여준다. 메시지 전달을 통한 클래스 간 협력이 이들 클래스 간 관계에 나타난다. 지금까지 과정을 통해 use case를 분석하고, 이를 토대로 class와 객체를 간략하게 정의 했으며, 다시 이들을 통해 시스템이 수행할 활동과 상호관계를 activity diagram과 sequence diagram을 통해 모델링 했다. 이렇게 만들어진 다이어그램을 통해 클래스가 가지는 속성과 연산을 명확하게 표현할 수 있다. 또 클래스 다이어그램의 갱신된 부분은 다시 순차 다이어그램에 적용 시스템의 설계를 진행시켜 나간다. 이와 같이 각 과정은 단계적으로 진행 되기 보다는 다음단계의 결과가 다시 이전단계에 영향을 미치는 형태로 반복적인 과정을 취하게 된다.

b. 사용목적

클래스다이어그램은 개발 과정 전반에 걸쳐 사용된다. 클래스 다이어그램은 시스템 또는 서브시스템을 구성하는 클래스의 정적인 구조를 나타낸다. 클래스의 정적인 구조는 클래스들과 이들 클래스다이어그램의 특성, 즉 속성과 오퍼레이션 그리고 그들 간 관계를 나타낸다. 시스템 또는 서브시스템의 클래스는 시스템 기능 요구의 일부를 충족시킨다. 클래스다이어그램은 클래스 모델의 다른 구성요소들이 어떻게 서로

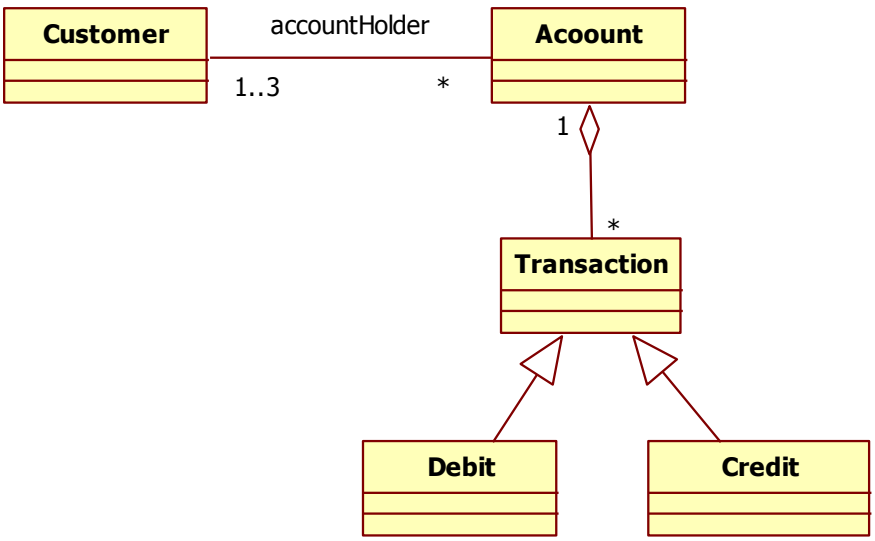
상호작용 하는가를 보여주지는 않는다. 그것은 상호작용 Sequence Diagram에서 다루어 진다. 클래스 다이어그램은 각 클래스의 행동측면 및 데이터 관리 책임과 이러한 책임이 클래스 모델 전체에 걸쳐서 어떻게 위임되는가를 보여준다. 클래스다이어그램은 시스템 최종 사용자의 관점에서 볼 때, 시스템의 기능측면 요구를 보여주지 않는다 그런 역할은 Use Case Diagram이 담당한다. 클래스 다이어그램을 만드는 주요 목적은 다음과 같다.

- 시스템이나 서브시스템을 구성하는 클래스를 문서화하는데 사용된다.
- 클래스 간 연관, 일반화, 집합연관 관계를 나타내는 데 사용된다.
- 클래스의 특성, 특히 클래스의 속성과 오퍼레이션을 나타내는 데 사용된다.
- 문제 영역 내 클래스에 대한 명세에서 제안된 시스템 구현모델에 이르기까지 개발 생명주기 전체에 걸쳐서 그 시스템의 클래스 구조를 보여주는 데 사용 된다.
- 특정 시스템의 클래스가 기존 클래스 라이브러리와 어떻게 상호작용하는지를 문서화 한다.
- 클래스 구조에서 개별 객체 인스턴스를 보여 주는 데 사용된다.
- 주어진 클래스가 지원하는 인터페이스를 보여준다.

c. 사용법

- 클래스

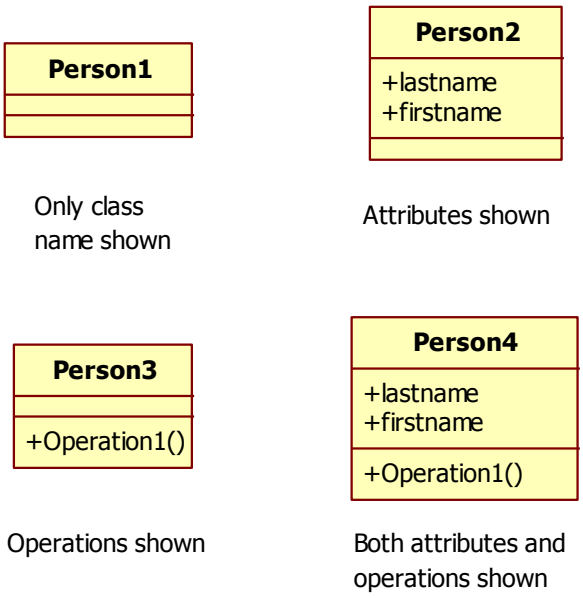
클래스다이어그램의 기본요소는 클래스이다. 클래스는 직사각형으로 나타내고, 그 이름을 사각형 중앙에 쓴다. 클래스 이름은 영어의 경우 대문자로 시작한다. 관습상 클래스 이름은 이름을 이루는 단어 사이에 공란이 없되, 각 단어는 대문자로 시작된다. 예를 들면 Bank account가 아니라 BankAccount이고, Customer Invoice가 아니라 CustomerInvoice이어야 한다. 다음 그림은 클래스를 가장 간단히 나타낸 것이다.



(그림 3-4) 은행 시스템의 클래스도 예

각 클래스 표기에는 속성 및 오퍼레이션을 나타내는 list compartment가 있고, 추가로 다른 특성을 나타내는 별도로 정의된 구획이 없거나 또는 하나 이상 있을 수 있다. 구획은 그 구획에 포함되는 클래스 특성을 열거하는 곳이다. 속성과 오퍼레이션을 나타내는 구획은 생략되어도 된다. 달리 표현하면 클래스를 다음과 같이 표시 할 수 있다.

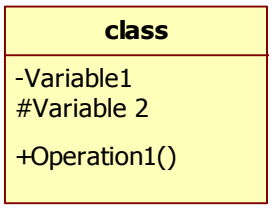
- 클래스 이름만 표시
- 클래스 이름과 속성 목록
- 클래스 이름과 오퍼레이션 목록
- 클래스 이름, 속성 목록과 오퍼레이션 목록



(그림 3-5) 클래스를 표현하는 네가지 방법

- 접근 제한자

접근제한자를 사용해 정보를 은닉한다. –는 private을 #은 상속 관계에만 정보를 공개하는 protected +는 어디에서나 접근이 가능한 public을 의미한다. 접근 제한은 속성과 메소드 모두에게 적용된다.

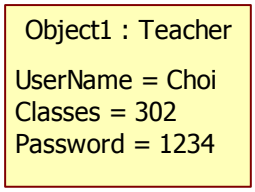


(그림 3-6) 메소드와 변수의 접근제한자 표현

• 객체

객체는 소프트웨어 시스템의 요소들로 실세계 요소들을 반영한다. 객체는 상태를 나타내는 속성과 각 상태에서 일어나는 동작들을 포함하며, 동작은 다른 객체와의 상호작용 일수도 있다. 객체는 클래스의 정의에 따라 구현되며 클래스는 속성과 동작의 선언을 포함한다.

객체는 클래스의 인스턴스로서, "인스턴스이름 : 클래스이름" 으로 객체를 표현한다. 인스턴스로 표현할 경우 속성값을 명시할 수 있다.



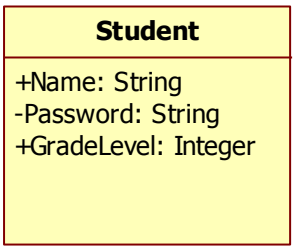
(그림 3-7) 속성 값이 명시된 객체

• 패키지

클래스들을 공통의 범주로 집단화 하는 방법이다. "패키지이름 : : 클래스명"으로 표현한다.

• 자료형

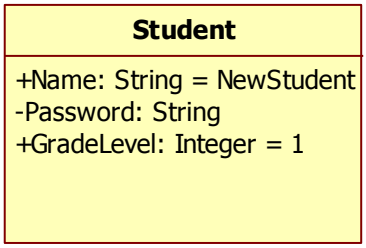
각 속성은 속성명 뒤에 : 을 붙이고 자료형을 명시할 수 있다. 이때 속성의 초기값 설정도 가능하고, 배열과 같은 개념의 다중성을 표현할 수도 있다. 메소드 또한 반환 자료형이 있는 경우 : 뒤에 반환 값의 자료형을 명시해줄 수 있다.



(그림 3-8) 자료형이 명시된 변수를 가진 클래스

• 초기값

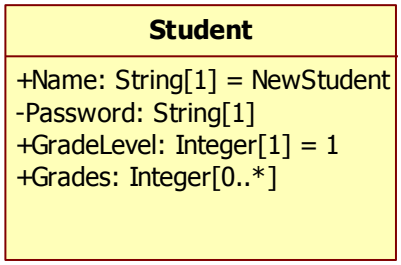
속성 이름과 자료형 뒤에 등호(=)와 값을 추가하여 속성의 기본값(default value)을 명시할 수 있다. Name과 Grade속성은 기본값이 할당되지만, Password 속성은 아무 값도 부여되지 않았다. 기본값을 정하는 것은 시스템의 요구에 달렸기 때문에 꼭 초기값을 명시할 필요는 없다.



(그림 3-9) 초기값이 명시된 변수를 가진 클래스

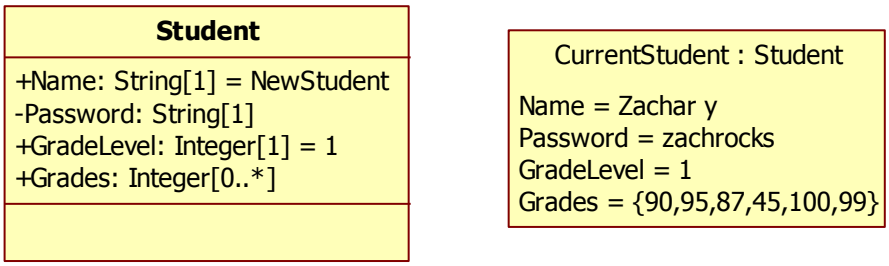
• 다중성

다중성은 클래스들 사이의 관계에 적용되는 것과 유사하게, 속성에도 적용 될 수 있다. 예를 들어, Grade속성을 갖는 Student클래스를 가지고, 이 속성이 단일 값을 갖지 않고 임의의 숫자에 해당하는 학생의 모든 등급들을 갖기 원할 수 있다. 다음의 클래스 다이어그램에서, 비록 단일 클래스의 경우에도 각 속성에 다중성을 갖도록 할 수 있다. Grade속성은 속성 범주(0에서 다수) 내의 임의의 평점 수치를 의미하는 '0...*'의 다중성을 가질 수 있다.



(그림 3-10) 다중성을 가진 속성을 지닌 클래스

클래스의 인스턴스 내에서 속성이 중괄호({and})내의 서로 다른 값들을 집합으로 갖는 다중성을 갖도록 명세할 수 있다.

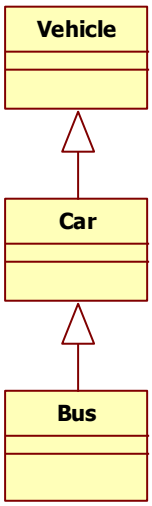


(그림 3-11) 중괄호를 사용하여 인스턴스의 다중성 표현

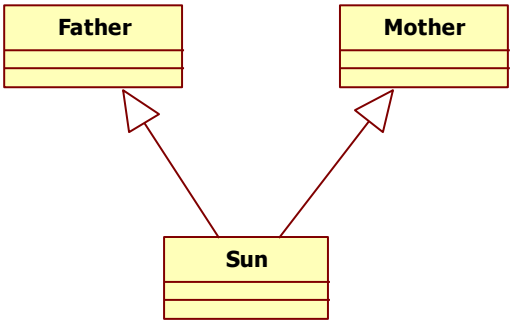
• 상속 (is a)

상속은 여러 클래스가 같은 인터페이스를 가지면서 각각 자신만의 능력 또한 가

질 수 있게 만드는 것으로, 코드의 재사용성을 높인다. 이런 기능은 코드의 크기를 줄이고 명확하게 하는 장점을 가진다.
상속은 속이 빈 화살표로 표현한다. 화살표가 시작되는 클래스는 상속을 받는 자식 클래스, 화살표를 받는 클래스는 상속을 하는 부모 클래스가 된다. 다중 상속의 표현도 가능하다.



(그림 3-12) 상속

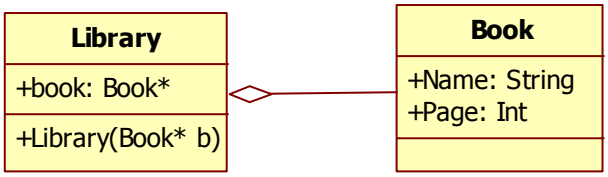


(그림 3-13) 다중 상속

• using
의존관계, 다른 클래스를 사용하는 것을 의미한다. 열린 화살표와 점선으로 표시한다.

(그림 3-14) using

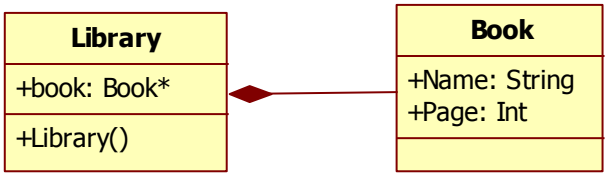
• aggregation
집합관계는 전체와 부분의 관계를 표시한다. 전체는 부분들을 포함하는 여러 클래스들로 구성되어 있다. 연관관계는 전체클래스가 소멸되어도 부분 클래스는 여전히 존재하며, 부분 클래스의 소멸 시에도 전체 클래스는 계속 존재하게 된다. 전체 클래스 앞에 빈 다이아몬드를 그린 선으로 표현된다.



(그림 3-15) aggregation

• composition

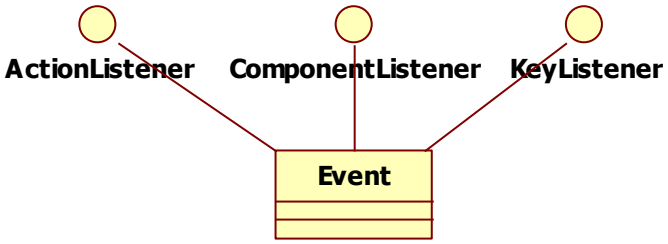
전체 클래스를 구성하는 부분 클래스가 스스로 존재할 수 없는 집합 관계이다. 이때 전체클래스가 소멸되면 부분 클래스도 없으며, 부분 클래스의 조합으로 전체 클래스가 구성된다. 조합 관계는 다중성이 함축되어 있으며 연관선과 속이 찬 다이아몬드로 표시된다.



(그림 3-16) composition

• realization

자바의 인터페이스 개념에서 사용할 수 있다. 속이 빈 삼각형과 점선으로 표시하며 인터페이스를 상속받아 구현하는 클래스임을 표현한다.



(그림 3-17) realization

d. 작성순서

모델링 과정에서 클래스다이어그램을 작성하기 위해서 다음과 같은 일련의 활동을 반복할 필요가 있다. 이들 활동은 산출물의 전체 구도와 명세에 도움이 된다.

ⓐ 클래스와 연관 찾기

클래스와 오퍼레이션은 UseCase Diagram, 정보 수집활동의 산출물, 또는 클래스 모델 자체에 대한 조사와 통찰을 통해 찾는다.

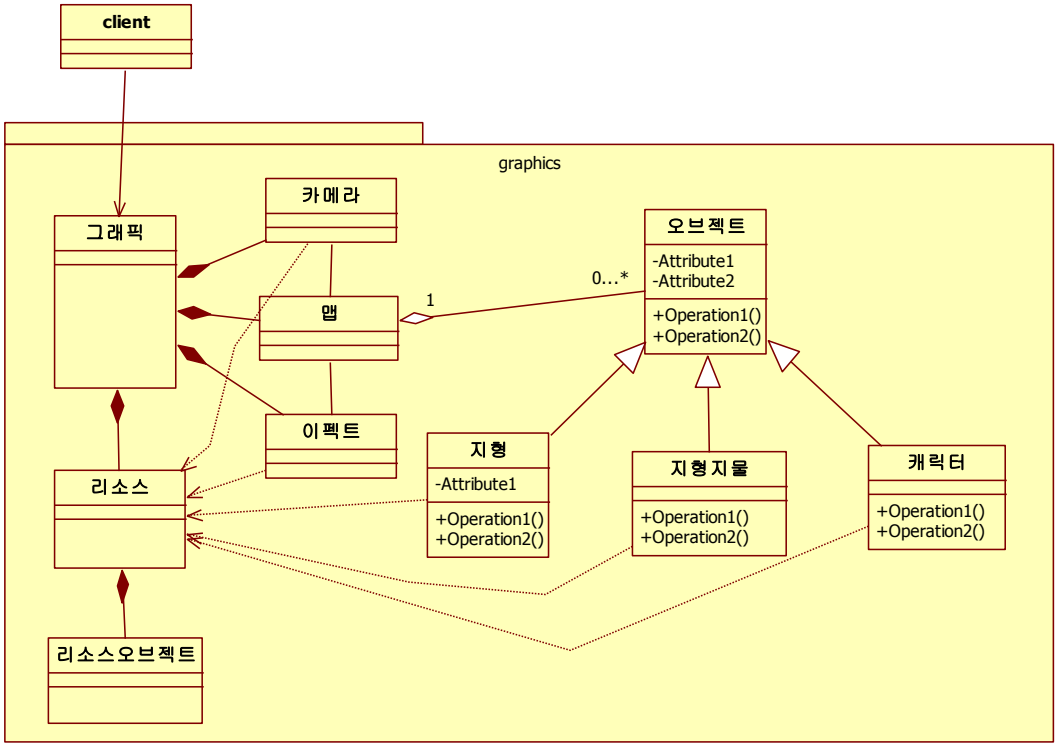
명사, 즉 사용사례와 면담 기록에 나타나는 사물의 종류 이름과 명사구는 흔히 클래스를 나타낸다. 동사구는 클래스 간 연관을 나타낸다. 어떤 경우에는 클래스 간 내재적인 논리적연관이 클래스 모델이 다음어짐에 따라 보다 더 분명해 진다. '고객이 계좌를 보유한다', '합승회원이 행선지를 등록한다', '자원봉사자가 특정 기술을 지닌다' 등은 연관으로 고려해볼 만한 구절 유형을 나타내는 예이다.

ⓑ 속성과 오퍼레이션을 식별하고 클래스에 배정하기

클래스의 각 인스턴스가 보유한 데이터 항목은 그 클래스의 속성이다. 클래스와 마찬가지로 개념적 모델링에서 속성을 입수가능한 정보원으로부터 찾아내는 데 어려움이 없어야 한다.

오퍼레이션은 클래스의 처리능력을 나타낸다. 개념적 모델링에서는 Interaction diagram상의 더 상세한 작업이 끝나기 까지는 각 클래스의 상세한 오퍼레이션을 찾아내기는 쉽지 않다. 그러나 어떤 오퍼레이션들은 입수 가능한 정보원에서 쉽게 알 수 있고 이들은 노트로 기록되어야 한다.

이런 방법으로 모델링 하는 것은 분석자가 완전한 모델이라고 만족할 때까지 계속된다. 모델은 문제 영역의 주요 클래스와 그들의 속성 및 해결책으로 필요한 오퍼레이션을 포함 해야 한다.



(그림 3-18) Class Diagram의 예

(3) State Diagram

a. 정의

State Diagram은 동적 모델 요소들의 행동을 기술하는 수단으로, Activity Diagram과 밀접한 관계가 있다. Activity Diagram이 작업 영역 간의 흐름을 기술하는 데 반해, State Diagram은 state(상태)간 흐름을 기술한다. 여기서 상태란 대상이 어느 시점에서 일정기간 처할 수 있는 조건을 의미한다. 예를 들면, 전화기는 수화기가 내려져 있는 상태이거나, 번호를 돌리는 중이거나, 통화 중이거나, 또는 연결이 끊겨진 상태일 수 있다. 어떤 것이 특정 상태에 있을 때, 일이 진행되거나 또는 그렇지 않을 수 있다. 예를 들어 전화기의 수화기가 내려져 있을 때, 전화기에는 어떠한 활동도 없다. 그러나 통화가 시작되면, 전화기에는 많은 활동이 생긴다. 따라서 이러한 상태들을 연결하여 State Diagram으로 표현함으로써, 시스템에 대한 적법한 흐름을 결정할 수 있다.

b. 사용목적

객체는 속성에 상관없이 일정한 동작을 하는 것도 있지만, 특정한 상태에 따라서 행동이 달라지는 것도 존재한다. State Diagram은 분석가, 설계자, 개발자들이 시스템 내의 객체 행위를 이해하는데 도움을 준다. 특히, 개발자는 객체들이 어떻게 행동하는지를 정확히 파악해야 객체들을 구현 할 수 있다. 따라서 객체의 상태가 명확하게 설명된 State Diagram을 가져야 요구사항에 맞는 시스템을 설계할 수 개발할 수 있다.

c. 사용법

사용법을 알아보기 전에 State와 Event의 개념을 알아보겠다.

• State(상태) : 시스템 또는 객체와 같은 시스템 내 실체들은, 상태에서 상태로 옮겨가는 것으로 볼 수 있다. 외부 이벤트들이 어떤 활동을 촉발시키고 그 활동이 시스템의 상태 및 특정 하위시스템의 상태를 변화시킨다. 예를 들어 은행시스템에서, 금액이 계좌에서 인출되고, 그 계좌가 채무(Credit) 상태에서 초과인출(Overdrawn) 상태로 옮겨 갈 수 있다. 그리고 특정 항목은 동시에 몇 가지 상태에 있을 수 있다. 예를 들어 고객은 물품인도를 기다리는 중이며, 동시에 빚진 상태에 있을 수 있고, 청구서에 대하여 분쟁 중 일수도 있다. 상태는 또한 중첩(nest)이 가능하다. 예를 들어 이자가 지급되는 은행 계좌의 신용상태는 잔고에 따라 이자율이 높은 상태 또는 낮은 상태로 이동할 수 있다.

• Event(이벤트) : 시스템은 외부 및 내부 이벤트에 의해 구동된다. 시스템은 이벤트에 반응하며, 이들 이벤트는 다른 이벤트들을 촉발시킨다. 외부 이벤트는 보통 일련의 내부 이벤트를 발생케 하며, 때로는 이들은 또한 외부 이벤트를 촉발시킨다. 예를 들

면, 자기의 저축 계좌에 수표를 저금하는 고객은 내부 이벤트를 촉발시켜서 미결 신용거래를 자기의 결제계좌와 관계하도록 하고, 외부 이벤트를 촉발시켜서 수표 발행자에게 은행 시스템에 의한 대금 이체를 요청한다. 시간 역시 이벤트의 발생 자로, 예를 들면 청구일자는 송장 인쇄 이벤트를 발생시킨다. 이벤트는 UML로 표현된 정보를 인자(argument)로 전달한다. 이벤트가 다른 이벤트들을 촉발시키므로 이 정보는 시스템을 흘러 다닌다. 이들 인자들은 이벤트 발생시 다음에 올 적절한 상태는 어떤 상태인가를 결정하는 일과 같은 의사결정의 일부로서 사용될 수 있다. 궁극적으로 이벤트는 오퍼레이션의 형태로, 객체에 의해 실현된다. (어떤 프로그래밍 언어에는 이벤트가 들어 있으며, 오퍼레이션은 event handler(이벤트 처리자)로 구현된다.

• 상태와 의사상태

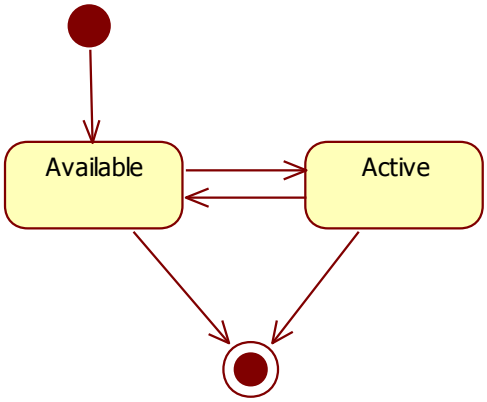
상태도의 상태는, 어떤 조건을 충족시키는 모델 요소의 생명주기의 한 지점으로, 특정 행동이 수행 중이거나 또는 어떤 이벤트를 기다리는 곳이다. 상태는 단순상태 이거나 복합상태 일 수 있다. 단순상태는 더 이상 나누어 지지 않으며, 객체의 단순한 속성 값으로 표현된다. 복합상태는 중첩된 상태도 또는 단일 도해에서 상태 안에 상태를 내장시킴으로써 더 나누어진다.

상태에는 두 가지 특별한 상태가 있는데, 초기의사상태는 흐름의 시작이다. 시작 상태는 상태머신을 시작하는 촉발자를 표기할 수 있고, 이 경우 많은 초기 상태가 있을 수 있다. 시작 상태에 이름이 없으면 상태 기계의 디폴트 시작점이다. 이는 검은 점으로 나타내며 그림 3-4처럼 이름 붙일 수 있다.

(그림 3-19) 시작 상태

• 종료 의사상태는 그림 3-5처럼 원으로 둘러싼 검은 점으로 그려진다. State Diagram에서는 종료 상태가 여러 개 존재할 수 있다. 종료상태는 정상적인 종료를 나타낸다.

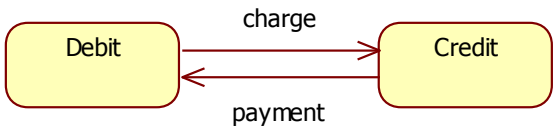
(그림 3-20) 종료 상태



(그림 3-21) State Diagram의 시작점과 종료점

• 전이

전이는 상태간 이동이다. 전이는 그림 3-7에서처럼, 상태와 상태사이의 화살표로 나타낸다. State Diagram에서 전이는 항상 어떤 이벤트에 반응하여 발생하며, 전이를 촉발시키는 이벤트 이름은 선 옆에 나타낸다.



(그림 3-22) 클래스의 상태간 전이

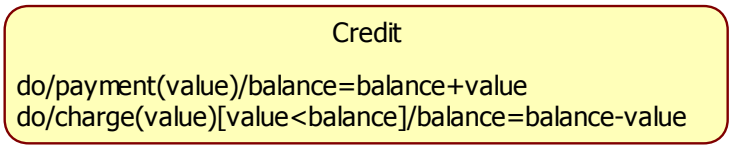
• 행동

상태는 행동을 촉발시킬 수 있다. 행동의 촉발방법은 네 가지이다.

진입 시(On Entry)	행동이 상태 진입 즉시 촉발된다.
수행 시(Do)	행동이 상태의 수명기간 중 발생한다.
이벤트 발생 시(On Event)	행동이 이벤트에 반응하여 발생한다.
종료 시(On Exit)	행동이 상태종료 직전에 발생한다

(표 3-3) 행동의 촉발방법

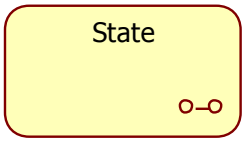
전이 또한 행동 촉발을 할 수 있다. 구문규칙은 상태의 이벤트 행동에 대한 것과 같고 전이와 함께 표기한다. 상태로부터의 전이가 하나라면 행동은 상태에 관한 종료행동(exit action)으로 나타내는 것이 가장 좋지만, 상태로부터의 종료(exit)가 복수인 경우 행동들이 다르므로, 이들 행동을 관련 전이에 첨부할 필요가 있다.



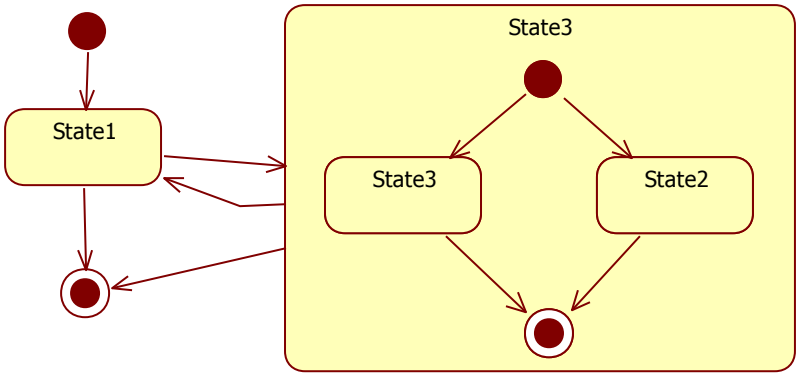
(그림 3-23) 상태의 행동

• 복합 상태

복합상태는(Composite state) 여러 하위상태(Substate)로 추가 분해될 수 있다. 하위 상태들은 상태 내에 또는 별도 도해에 나타낼 수 있다. 그림 3-9에 있는 클래스의 상태를 보자



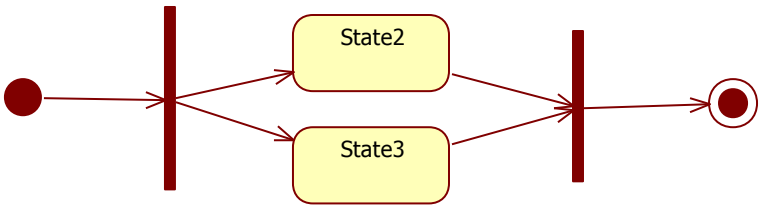
(그림 3-24) 상태에 세부흐름을 나타내는 표기법



(그림 3-25) 복합상태에 그려진 세부흐름

• 포크와 조인(Fork and Join)

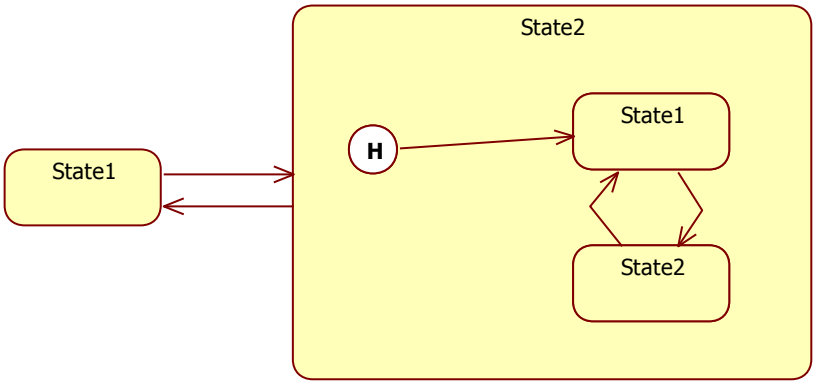
복합상태는 여러 병행 하위 상태들로 이루어 질 수 있다. 이때에는 포크를 써 전이를 다중 경로로 나누거나 또는 조인을 써서 다중 경로를 단일 전이로 결합한다. 포크와 조인은 굵은 선으로 나타낸다. 포크는 하나의 진입전이와 둘 이상의 종료전이가 있고 조인은 많은 진입전이와 하나의 종료 전이가 있다. Workflow가 병행 흐름들로 나뉠 때, 이들 흐름은 같은 도해에서 재결합 되어야 한다.



(그림 3-26) 포크와 조인을 사용한 병렬상태

• 이력상태(History State)

보통은 복합 상태에 진입하면, 복합 상태는 초기상태(또는 전이가 직접 그 하위 상태로 진입하면 그 지시된 하위상태)에서 시작한다. 그러나 때로는 복합 상태를 마지막으로 떠난 지점의 복합 상태로 재 진입하는 것이 필요하다. 이것을 나타내려면 안에 H자를 가진 원으로 표시되는 이력 상태를 추가하면 된다. 그림에서와 같이 표현되는데 복합 상태들 자체가 복합 상태이면, 그들이 중지된 지점에서의 중첩된 상태 되찾기가 필요할 경우도 있다. 이력 상태 표시에 H 대신 *H가 표시되면, 이것은 깊은 이력 상태로 부리며 하위상태는 그들이 떠났던 지점에서 다시 시작된다. 재시작은 중첩된 상태들 아래로 전체에 걸쳐 계속된다.



(그림 3-27) 복합상태에서의 이력 상태

d. 작성순서

ⓐ 복잡한 행동을 하는 실체들을 식별한다

분석 수준에서 모델링 되는 실체들은 시스템과 상호작용 하는 비즈니스 행위자, 또는 변경되는 상태와 상태 간 복잡한 관계를 갖는 계좌, 정책 및 협정과 같은 실체 클래스들이다. 설계 수준에서는 보통 많은 복합 클래스들이 도입 된다. 스크린을

나타내는 경계(Boundary) 클래스, 거래를 관리하는 통제(Control)클래스가 그 예이다.

ⓑ 그 실체의 초기 및 최종 상태를 결정한다.

실체가 어떻게 생성되고 어떻게 소멸되는지 결정한다.

ⓒ 그 실체에 영향을 주는 이벤트들을 식별한다. use case를 이용해 객체에 관한 이벤트들을 추적한다. use case는 이벤트의 그룹으로 시스템에게 의미 있는 기능을 제공하며, 일련의 협력하는 객체들에 의해 실현 된다.

ⓓ 초기 상태에서 시작하여, 이벤트의 영향을 추적하고 중간 상태들을 식별한다.

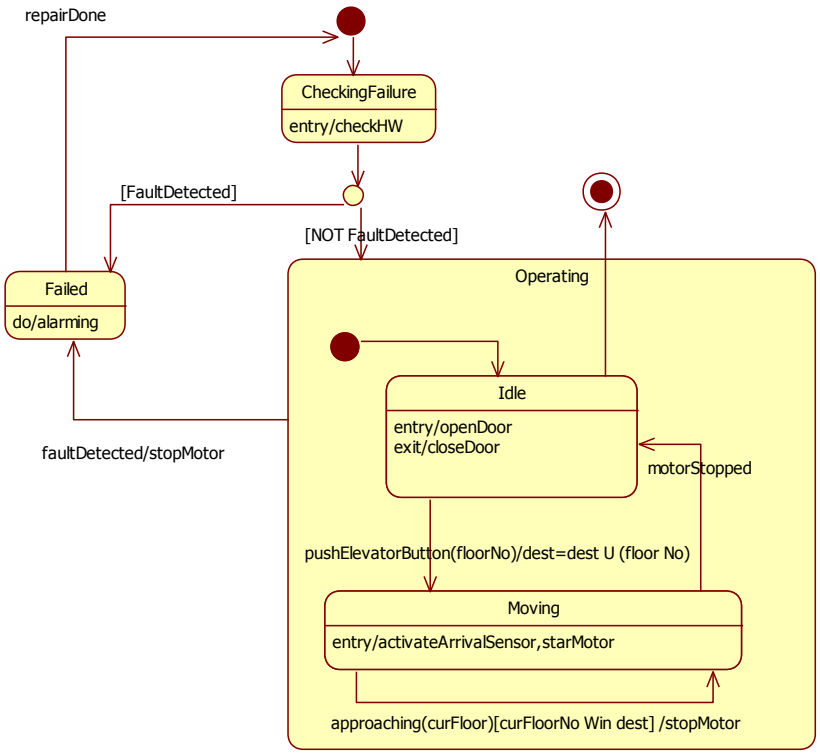
초기 상태에서 시작하여 어떤 이벤트들이 초기 상태에 영향을 주는지를 확인하고, 추적하면서 중간상태들을 확인한다.

ⓔ 상태에 관한 진입 및 종료 행동들을 식별한다.

상태에 진입할 때 어떤 행동을 취할 것이며, 종료할 때 어떤 행동을 취할 것인 지를 식별한다.

ⓕ 필요 시 하위상태를 이용하여 상태들을 확장한다.

상태 내 행동들이 클래스의 오퍼레이션 및 관계에 의해 지원되는지를 점검한다. 실체가 클래스일 경우, 상태 내 행동들이 클래스의 오퍼레이션 및 관계에 의해 지원되는지를 점검하여, 그렇지 않을 경우 클래스를 확장한다.



(그림 3-28) State Diagram의 예

3) Component 설계 및 배치

(1) Component diagram

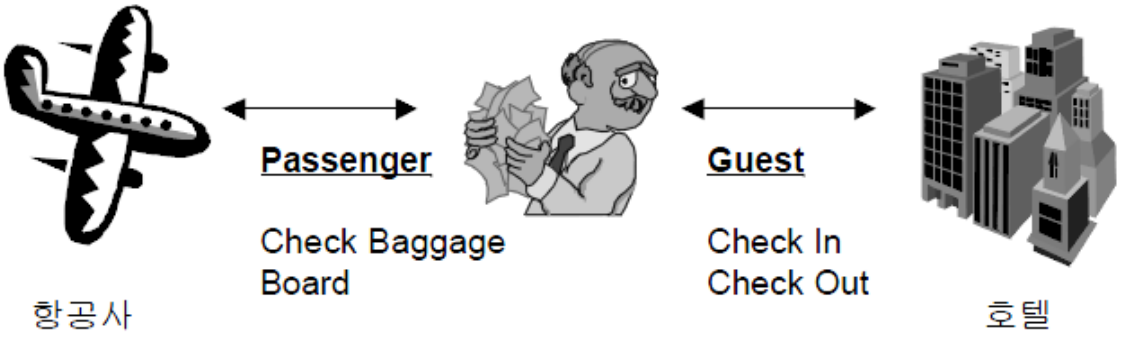
a. Component& Interface

Component

소프트웨어 컴포넌트는 시스템을 이루는 물리적 요소이다. 컴포넌트는 컴퓨터 내에 있으며, 분석가의 마음속에는 절대로 없다. 컴포넌트는 다른 컴포넌트에 인터페이스를 제공한다. 컴포넌트는 시스템의 기능을 정의하며, 한 개 이상의 클래스를 구현하여 하나의 컴포넌트를 만든다.

Interface

인터페이스는 서비스(service)또는 오퍼레이션(operation)의 집합이다. 인터페이스 모델링이 컴포넌트 재사용을 위한 컴포넌트 설계의 핵심이며 객체(또는 컴포넌트)는 많은 상호작용을 하며, 객체의 역할은 다를 수 있다. 컴포넌트는 여러 개의 인터페이스를 구현한다.



(그림 3-29) 인터페이스 예

b. 정의

컴포넌트 다이어그램은 컴포넌트, 인터페이스로 구성되어 있으며, 각각의 관계가 설정되어 있는 다이어그램이다.

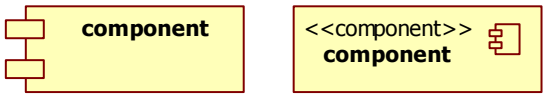
c. 사용목적

프로젝트 초기에 파악된 소프트웨어 컴포넌트와 그들 간 인터페이스를 모델링 하고, 그 결과로 상세 설계에 대한 고려를 보다 적절한 시기로 연기시키는 수단을 제공한다. 내부의 세세한 사항에 대한 명세를 숨기는 대신 컴포넌트 간 인터페이스로 나타나는 컴포넌트 간 관계에 초점을 맞추게 한다. 이미 검증된 컴포넌트들이 새 시스템 설계에서 어떻게 통합되는지를 보여준다.

d. 사용법

• 컴포넌트

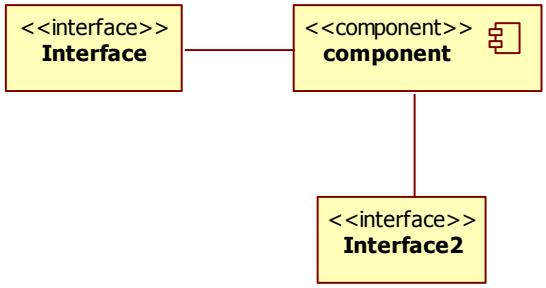
컴포넌트는 컴포넌트 이름을 기입한 사각형으로 나타낸다. 최근의 UML버전에서는 사각형으로 단순화한 것을 이용하지만 우리가 쓰게될 StarUML에서는 1.X의 표기법을 사용하고 있다.



(그림 3-30) 컴포넌트의 예

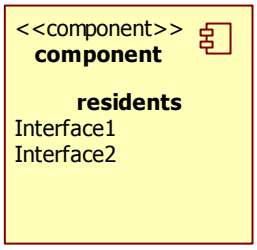
• 컴포넌트 인터페이스

컴포넌트의 사용과 모델링의 핵심 개념은 각 컴포넌트가 제공 하거나 요구하는 인터페이스 명세를 준수해야 한다는 점이다. 당연히 컴포넌트를 그 경계를 정의하는 인터페이스와 연결시킬 수 있어야 한다.



(그림 3-31) 컴포넌트의 제공 및 요구 인터페이스

컴포넌트가 따라야 하는 인터페이스는 의존 스테레오타입 그룹에 따라 리스트 형식으로 나타낼 수도 있다.



(그림 3-32) 그룹화 된 리스트로 나타난 컴포넌트 인터페이스

e. 작성순서

컴포넌트 다이어그램을 작성할 때에는 다음의 활동을 반복적으로 행해야 한다.

ⓐ 컴포넌트와 의존관계 찾기

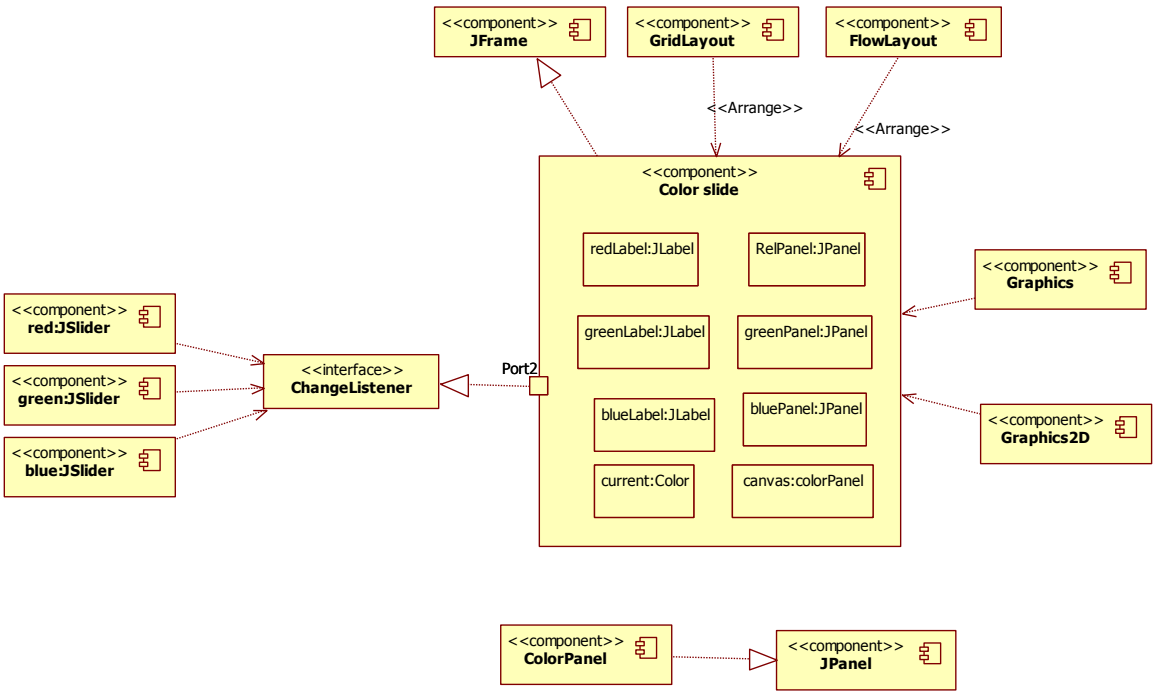
컴포넌트와 의존관계는 개발 Lifecycle의 초기 단계에서 얻은 정보에서 찾을 수 있다. Use Case가 정의되고 Activity Diagram이 작성됨에 따라, 분석가는 파악된 시스템을 구성하는 핵심 컴포넌트와 의존 관계를 찾는다.

ⓑ 서브컴포넌트를 찾고 레벨을 정하기

모델의 한 컴포넌트 내의 중요한 서브컴포넌트가 주요 컴포넌트로 될 수 있는지를 찾는 일이다. 역으로 다른 한 컴포넌트와만 상호작용을 하는 컴포넌트도 있을 수 있다. 이 컴포넌트는 다른 컴포넌트의 서브컴포넌트로 포함될 수 있다.

ⓒ 컴포넌트 간 인터페이스를 명시하기

위의 예비 모델을 토대로, 다른 컴포넌트들 간 인터페이스를 명확하게 한다. 컴포넌트는 제공하거나 요구하는 인터페이스에 의해 연결됨을 기억할 필요가 있다.



(그림 3-33) Component Diagram의 예

(2) Deployment Diagram

a. 정의

- 시스템 하드웨어에 인공물이 어떻게 배포되는지를 보여 준다. 그리고 하드웨어가 다른 하드웨어와 어떻게 연결되는지도 보여준다. 주요 하드웨어 아이템은 노드이다.

b. 사용목적

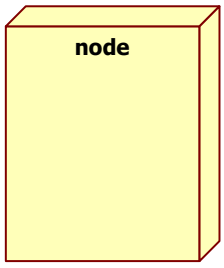
시스템의 물리적 구현 측면을 모델링 한다. Deployment Diagram은 시스템을 구성하는 하드웨어 요소(또는 노드)의 구성(configuration)을 모델링 하는 데 이용된다. 여기에는 컴퓨터(클라이언트와 서버), 임베디드 프로세서, 센서(sensor)와 주변장치 같은 장치들이 포함된다. 또한 배치도는 런타임(run-time)시스템에서 소프트웨어 컴포넌트가 위치하는 노드를 나타내는데 사용된다. 배치도는 다음 목적에 이용된다.

- 물리적 하드웨어 컴포넌트 및 이들 간 통신 경로를 모델링 하는 데 이용된다.
- 시스템 아키텍쳐를 계획하는 데 이용된다.
- 하드웨어 노드에 소프트웨어 컴포넌트의 배치를 문서화 하는 데 이용된다.

c. 사용법

- 노드

가장 기본적 형태로, Deploy Diagram은 통신 경로에 의해 연결된 노드를 보여준다. 노드에 대한 표기법은 다음 그림과 같다.

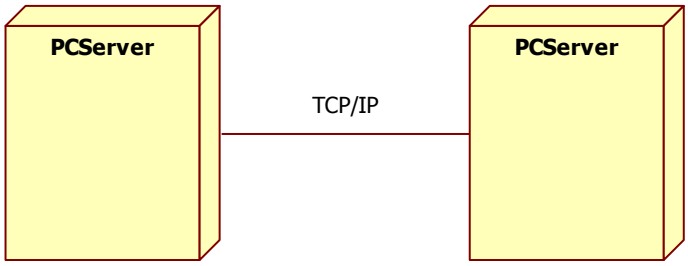


(그림 3-34) 노드 표기

노드는 시스템의 처리 자원을 나타내며, 처리능력과 메모리를 가진 컴퓨터가 그 전형적 예이나 센서, 주변장치 또는 임베디드 시스템을 나타내는 데도 사용한다.

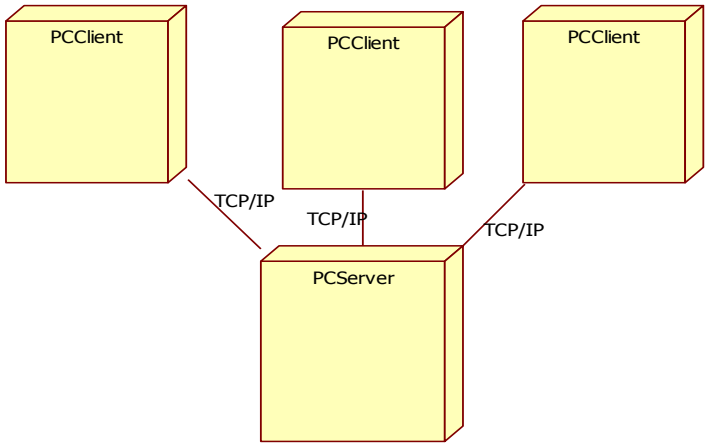
- 통신경로

노드는 통신 경로에 의해 연결되며, 통신 경로는 연결된 노드들 간 통신의 특성을 나타내도록 스테레오타입화 될 수 있다. 다음 그림은 TCP/IP 네트웍 프로토콜을 이용하여 연결된 클라이언트와 서버를 나타낸다.



(그림 3-35) 클라이언트와 서버간 통신경로

Deploy Diagram은 위의 그림처럼 노드 타입을 나타내거나, 또는 아래 그림처럼 인스턴스를 나타내는 데 사용된다.. 아래 그림을 구현 될 시스템의 클라이언트와 서버를 나타낸다.



(그림 3-36) 배치도 상의 인스턴트들

마지막 노드는 <<device>>로 스테레오타입화 되거나 <<execution environment>>의 특정 타입으로 스테레오타입화 될 수 있다.

• 산출물(Artefact)

앞서 배치도는 소프트웨어가 하드웨어에 배치되는 것을 나타낸다고 언급했다. 시스템의 하드웨어 모델링도 설명했다. 이제는 배치될 코드 모듈에 대해 설명한다.

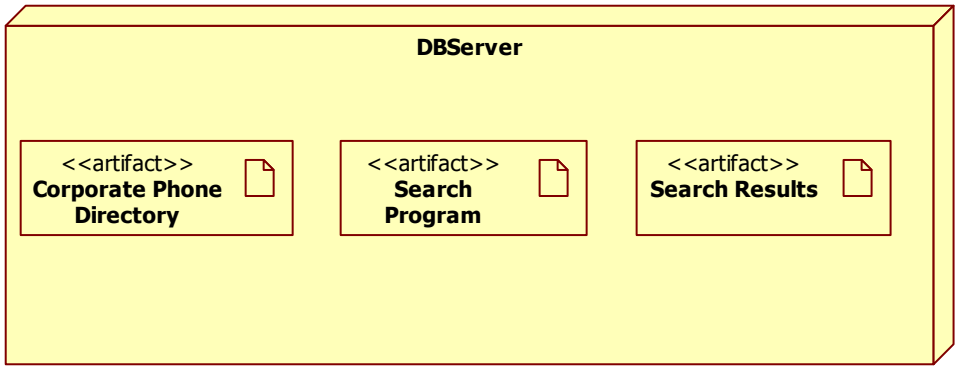
산출물의 UML표기법은 다음 그림과 같다.

(그림 3-37) Artifact표기법

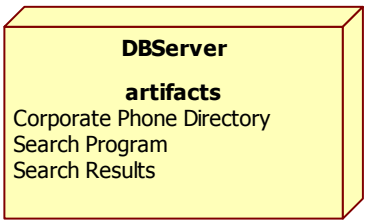
산출물은 노드에 배치되는 응집력 있는 기능을 가진 원시코드의 집합체이다. 산출물은 컴포넌트에 배치되는 집단을 나타낸다. 이는 아래와 같은 표기법으로 나타낸다.

• 산출물의 배치(Deployment of Artifact)

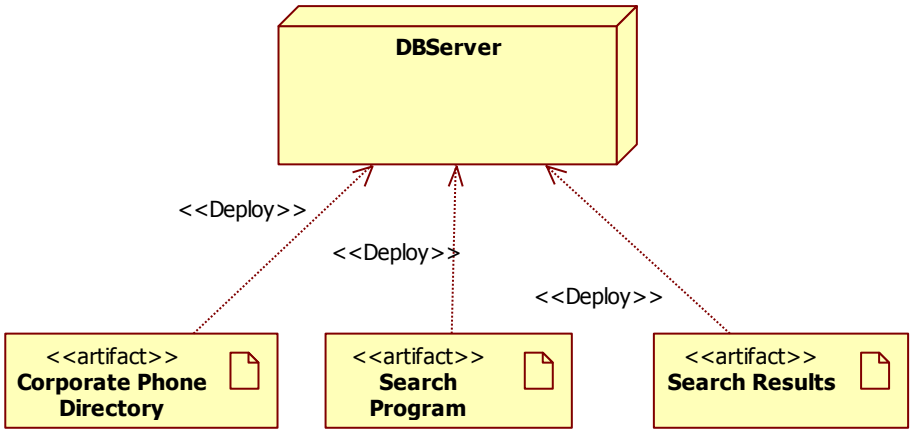
UML에서 노드에의 산출물 배치는 세가지 방법으로 나타낸다 첫 번째 방법은 도형 상 포함관계를 쓰는 것이고, 두 번째 방법은 노드에 배치된 산출물의 텍스트 목록으로 나타내는 것이며 마지막으로 <<deploy>> 스테레오 타입을 쓰는 것이다.



(그림 3-38) 도형 상 포함관계를 쓴 Artifact 배치



(그림 3-39) 텍스트 목록을 쓴 Artifact 배치



(그림 3-40) <<deploy>> 스테레오타입을 쓴 Artifact 배치

d. 작성순서

ⓐ 목적을 결정한다

첫 단계는 Diagram의 목적을 결정하는 일이다. Diagram이 노드만을 모델링 하는가, 아니면 산출물과 배치명세를 모델링 하는데 사용되는가?

ⓑ 노드를 추가한다

Diagram자체를 산출하는 첫 단계는 관련된 노드를 정하는 일이다 이들은 실행코드들이 가동되는 프로세서들이다.

ⓒ 통신경로를 추가한다.

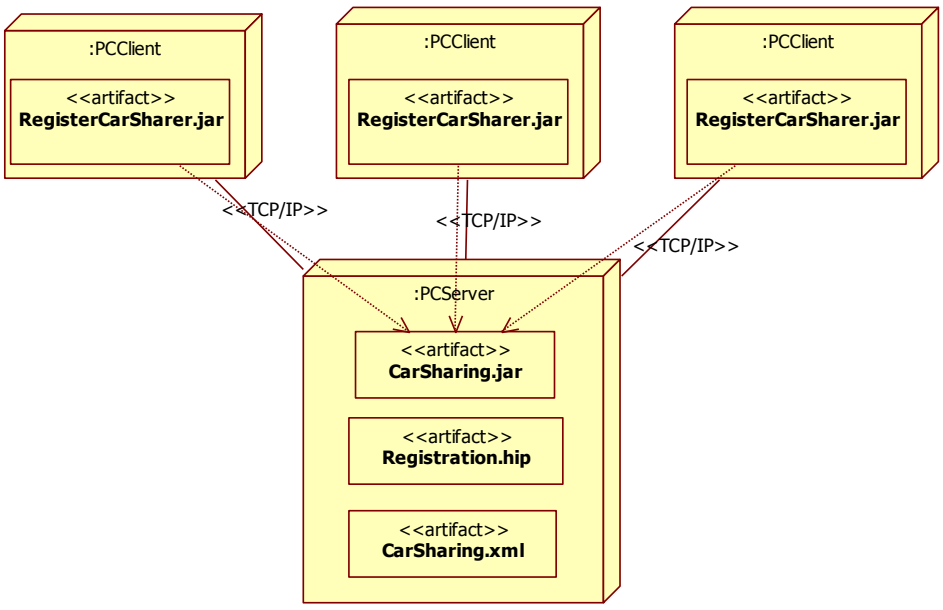
노드 쌍 간의 통신을 위한 경로들이 추가되어야 한다. 통신 프로토콜을 나타내기 위해 스테레오타입이 사용된다.

ⓓ 다른 요소를 추가한다

Diagram의 목적이 산출물과 배치명세를 시스템에서 위치하는 곳을 보여주는 것이라면 이들은 Diagram에 추가 되어야 한다.

ⓔ 의존관계를 추가한다.

산출물들 간의 의존관계를 배치도에 나타내는 것이 도움이 된다. 그러나 Diagram은 선으로 매우 어지러워질 수 있다. 이 경우 여러 Deployment Diagram을 작성하여, 그 각각이 구현 기본 구조의 다른 측면을 설명하도록 하는 것이 더 좋다.



(그림 3-41) Deployment Diagram의 예