An introduction to UML

과 목 명: 소프트웨어 모델링 및 분석
교 수 명: 유준범 교수님
제 출 일: 2016.03.16. (수)
팀 원 : 201211341 김태현
201411269 고수창
200911411 이상규

1. **UML 개요**
    a. UML 이란 무엇인가?
    b. UML 을 정의하게 된 동기
        i. 모델링을 하는 이유
        ii. 소프트웨어 산업의 경향
        iii. 산업표준이 생기기 전

2. **UML 의 목표**
3. **UML 의 범위**
    a. UML 산출물
    b. UML 의 범위 바깥

4. **UML 의 과거, 현재 그리고 미래**
    a. UML 0.8 – 0.91
    b. UML 1.0 – 1.1
    c. UML 의 현재와 미래

5. **유스 케이스 (Use case)**
6. **관계 (Relationship)**
7. **상태 (State)**
8. **배치 (Deployment)**
9. **컬레보레이션 (Collaboration)**
10. **인터렉션 (Interection)**
11. **클래스 다이어그램(Class Diagram)**
    a. 연관 (Association)
    b. 속성 (Attribute)
    c. 연산 (Operation)
    d. 일반화 (Generalization)
    e. 제약 규칙 (Constraint Rule)

12. **상호작용 다이어그램 (Interaction Diagram)**
    a. 순차 다이어그램 (Sequence diagram)
    b. 협동 다이어그램 (Collaboration Diagram)
    c. 순차 다이어그램과 협동 다이어그램의 비교
    d. 언제 상호작용 다이어그램을 사용하는가?

13. **상태 다이어그램 (State Diagram)**
14. **활동 다이어그램 (Activity Diagram)**
15. **컴포넌트 다이어그램 (Component Diagram)**

1. UML 개요

a. UML 이란 무엇인가?

Unified Modeling Language (UML)은 소프트웨어 시스템, 더 나아가 업무 모델링, 기타 소프트웨어가 아닌 시스템의 산출물을 규정하고 시각화하며 구현하고 문서화하는 언어이다 (The UML is a language for specifying, visualizing, constructing, and documenting the artifacts of a software systems, as well as for business modeling and other non-software systems). UML 은 복잡한 대형 시스템을 모델링하는데 성공적으로 증명된 공학적 기법들을 모아 제시한 것이다.

UML 은 80 년대 후반에서 90 년대에 이르는 기간 동안 나타난 객체지향 분석 설계 방법론의 흐름을 이어받은 객체지향 모델링 언어이다. 가장 영향력이 있던 Booch 와 Rumbaugh, Jacobson 의 방법론을 직접적으로 통합하였으며 이외 방법론의 장점들 을 모아서 표준화시킨 것이다. 99 년 3 월 현재 UML 은 OMG(Object Management Group)에 의해서 표준으로 확정되었으며 개정된 1.1 판이 제시된 상태이고 1.3 판의 개정 작업이 이루어지고 있다.

UML 은 모델링 언어로서 방법론의 일부이다. 방법론의 다른 부분인 공정(process)은 표준화되어 있지 않으나 어떠한 방법론을 사용하든지간에 통일된 표기법을 제시하 는 것이 UML 의 역할이다.

b. UML 을 정의하게 된 동기

- **모델링을 하는 이유**

소프트웨어 시스템을 구축하거나 혁신하기 전에 모델을 개발하는 것은 건물을 지을 때 청사진을 그리는 것과 마찬가지로 필수적인 일이다. 좋은 모델은 아키텍처를 건 전하게 하고 프로젝트 팀의 의사소통을 원활히 하는 데에 있어서 필수적이다. 복잡 한 시스템의 모델을 만드는 이유는 그러한 시스템을 한번에 통째로 이해할 수 없기 때문이다. 시스템의 복잡성이 커질수록 좋은 모델링 기법의 중요성도 커지게 마련 이다. 프로젝트의 성공을 위한 요소들이 많이 있지만 엄격한 모델링 언어의 표준화 는 그 중 필수적인 요소이다. 모델링 언어는 다음과 같은 사항을 포함해야 한다.

- **소프트웨어 산업의 경향**

많은 회사에서 소프트웨어의 전략적 가치가 증가함에 따라 산업계는 소프트웨어의 생산을 자동화할 수 있는 기법을 찾게 되었다. 그 기법은 품질을 향상시키고 비용 과 시간을 절감할 수 있는 기법들로서 컴포넌트 기술, 시각적 프로그래밍, 패턴, 프 레임웍 등을 포함한다. 또한 시스템이 범위나 스케일이 증가하는 경우에도 그 복잡 성을 다루어 낼 수 있는 기법이 필요했다. 특히, 반복해서 나타나는 아키텍처에 관 련된 문제, 예를 들어, 물리적인 분산, 동시성, 복제, 보안, 로드 밸런싱, 고장 방지 능력(fault tolerance) 등을 해결하기 위한 필요성을 인식하게 되었다. 최근의 월드와이 드 웹을 위한 개발경향은 몇 가지를 단순화시켰지만 아키텍처 문제는 더욱 심화되 었다.

복잡성은 응용프로그램의 영역과 공정의 단계에 따라 다르게 마련이다. UML 개발 자들에게 다가온 핵심적인 동기부여 중의 하나는 모든 영역에 걸쳐 모든 규모의 복 잡한 아키텍처를 적절하게 처리할 수 있는 의미와 표기법을 창조해내는 것이었다.

- **산업표준이 생기기 전**

UML 이전에는 뚜렷이 선도하는 모델링 언어가 없었다. 사용자들은 대동소이한 많 은 모델링 언어 중에서 하나를 선택해야만 하였다. 대부분의 모델링 언어들은 많은 개념들을 공통적으로 사용하면서도 그것을 약간씩 다른 의미와 표기법을 통해 표현

하였다. 이러한 통일성의 부족은 새로운 사용자들이 객체지향 모델링을 선택하는데 꺼리는 요소가 되었다. 사용자들은 일반적으로 사용되기에 적합한 산업계의 표준이 등장하기를 기다려 왔다. 즉 모델링을 위한 공통어를 원했던 것이다.

업체들 역시 비슷하면서 약간씩 다른 여러 모델링 언어를 지원해야 할 필요성 때문 에 객체 모델링 분야에서 어려움을 느끼고 있었다. 따라서 많은 업체들이 통일된 모형화 언어로서의 UML 의 개발을 지지하게 되었다.

UML 은 프로젝트의 성공을 보장하는 것은 아니지만 많은 일들을 개선시킨다. 예를 들어, 프로젝트나 조직을 바꿀 때에 새로 교육하는 비용을 감소시킨다. 또한 여러 가지 도구나 공정, 영역사이의 통합을 이끌어 낼 수 있다. 가장 중요한 것은, UML 이 개발자들로 하여금 업무 가치를 생산하는데 집중할 수 있게 하고, 그것을 성취할 수 있는 패러다임을 제공한다는 데 있다.

c. UML의 목표

- UML의 저자들이 UML을 설계하면서 주안점을 두었던 목표는 다음과 같다.
    1. 사용자들이 의미있는 모델을 만들고 교환할 수 있도록 사용하기 쉽고 표현이 풍부한 시각적 모형화 언어를 제공한다.
    2. 핵심 개념을 확장하기 위한 메커니즘을 제공한다.
    3. 특정 프로그래밍 언어나 개발 공정에 종속되지 않아야 한다.
    4. 모델링 언어를 이해하기 위한 공식적 기준을 제공한다.
    5. 객체지향 도구 시장의 성장을 장려해야 한다.
    6. 고수준의 개발 개념들, 예를 들어 협동(collaboration), 프레임웍, 패턴, 컴포넌트 등의 개념들을 지원한다.
    7. 산업계의 검증된 최상의 경험들을 통합한다.

d. UML의 범위

- Unified Modeling Language (UML)은 소프트웨어 시스템의 산출물을 규정하고 시각화 하며 구현하고 문서화하는 언어이다.
- 첫째로, UML 은 Booch, OMT, OOSE 의 개념을 융합하여 널리 사용될 수 있는 공통된 단일 모델링 언어를 만든 것이다.
- 둘째로, UML 은 기존 방법론들로 할 수 있었던 일의 영역을 확장시켰다. 예를 들어, UML 의 저자들은 분산 병렬 시스템의 모델링을 목표로 삼았다.
- 셋째로, UML 은 표준 공정이 아니라 표준 모델링 언어에 초점을 맞추었다. 물론 UML 은 어떤 공정의 문맥 안에서 적용되어야 하겠지만 경험상으로 보면 조직과 문 제영역의 차이에 따라 다른 공정이 요구되기 때문이다. 그러므로, 의미를 통일시키 는 공통 메타모델과 그 의미를 표현할 수 있게 하는 공통 표기법을 개발하는데 집 중하였다. UML 의 저자들은 사용사례 중심, 아키텍처 중심, 점진 반복적인 개발 공 정을 권장한다.
- UML 은 객체지향 공동체의 일치된 의견을 핵심 모델링 개념에 통합한 모델링 언어 이다. 그 확장 메커니즘에 따라 문제영역에 맞게 재단하여 사용할 수 있다.

i. UML 산출물

- 모델링은 관련된 부분에 집중하고 나머지는 무시하는 추상화를 통해 이루어진다. 모델의 특성에는 다음과 같은 것이 있다.
    - 복잡한시스템은모델의독립적인뷰의집합으로표현될수있다. 하나의뷰 만으로는 충분하지 않다.
    - 모든모델은상세함의정도가다른여러차원으로표현될수있다.
    - 좋은 모델은 실재를 잘 반영한다.
- UML 은 모델의 뷰라는 용어를 사용하여 다음의 그래픽 다이어그램을 정의한다. 다 음 중 밑줄 친 8개의 다이어그램이 실제 산출물의 이름이다.

- 사용사례 다이어그램 use case diagram : 사용사례와 사용자의 관계를 표현하는 다이어그램
- 클래스 다이어그램 class diagram : 클래스의 관계를 표현하는 다이어그램
- 행위 다이어그램 behavior diagrams
    - 상태차트 다이어그램 statechart diagram : 객체의 생명주기를 나타내며, 이 벤트에 의해 변화하는 객체의 상태를 표현하는 다이어그램
    - 활동 다이어그램 activity diagram : 객체에 작용하는 활동의 흐름을 표현하는 다이어그램
    - 상호작용다이어그램interactiondiagrams
    - 순차 다이어그램 sequence diagram : 객체들간의 상호작용을 시간적 순서를 강조하여 표현하는 다이어그램
    - 협동 다이어그램 collaboration diagram : 객체들간의 상호작용을 공간 적 협조 체계를 강조하여 표현하는 다이어그램
- 구현 다이어그램 implementation diagrams
    - 컴포넌트 다이어그램 component diagram : 컴포넌트들의 관계를 표현하는 다이어그램
    - 배치 다이어그램 deployment diagram : 시스템을 구성하는 물리적 객체나 장치들의 관계를 표현하는 다이어그램

- 위 다이어그램들은 분석 또는 개발 중인 시스템에 복합적인 관점을 제공한다. 모델 은 이러한 관점들을 통합하여 일관성 있는 시스템이 개발될 수 있게 한다. 이 다이어그램들과 설명적인 문서들이 주된 산출물이 된다.

ii. UML의 범위 바깥

- 프로그래밍 언어
    - UML 은 모델링 언어로서 프로그래밍 언어는 아니다. 복잡한 알고리즘 같은 것은 프로그래밍 언어로 표현하는 것이 나을 것이다. UML 은 객체지향 언어와 긴밀하게 연결되어 있으므로 둘을 동시에 활용할 수 있어야 한다.

- 도구
    - 언어의 표준화는 필연적으로 도구와 공정을 위한 기반이 된다. UML 이 제공하는 의미와 표기법은 도구의 개발과 호환성에 도움이 된다.

- 공정
    - UML 은 공정에 독립되어 공통어로 사용된다. 공정은 프로젝트의 성공을 좌우하는 중요한 요소이지만 조직과 문화, 그리고 주어진 문제 영역에 맞추어 재단되어야 한다.
    - Booch, OMT, OOSE 등 많은 방법론들은 잘 정의된 공정을 가지고 있으며 UML 은 대 부분의 방법론을 지원할 수 있다. 개발 공정에 대한 상당한 수렴이 있었지만 아직 표준화에 대한 합의에는 이르지 못했다. 아마도 최상의 경험들을 융합하여 개별적 인 공정을 만들어낼 수 있는 공정 프레임웍이 도출될 가능성이 있다. UML은 특정 한 공정을 지정하지는 않지만 사용사례 중심, 아키텍처 중심, 점진 반복적인 공정을 권장한다.

e. UML의 과거, 현재 그리고 미래

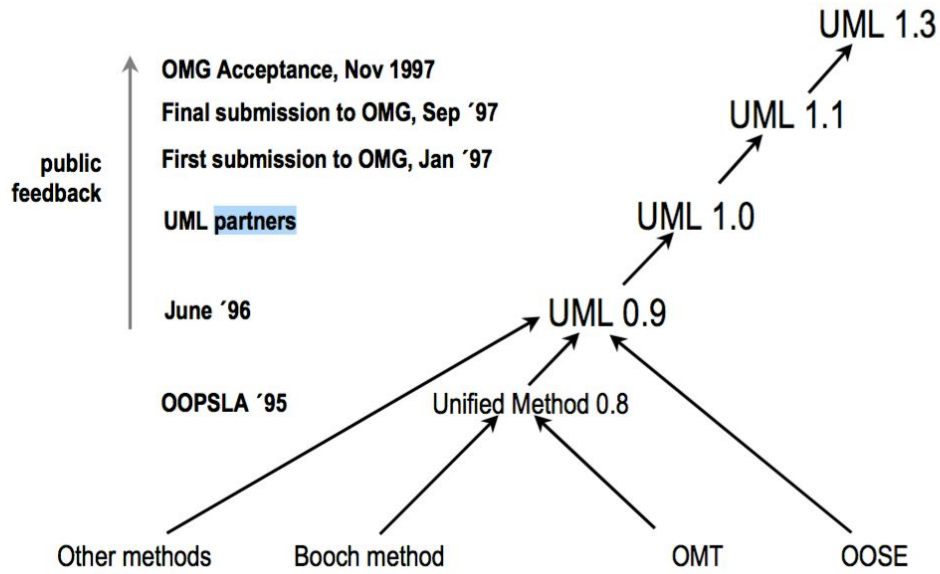
- UML 은 Rational 과 그 협력사들에 의해 개발되었다. 이는 Booch, OOSE/Jacobson, OMT 와 그 외의 방법론에서 제시하던 모델링 언어를 계승

발전시킨 것이다. 많은 회사들이 UML 을 표준으로 받아들여 업무 모델링, 요구사항 관리, 분석 설계, 프로 그래밍, 시험에 걸친 모든 범위에서 사용하고 있다.

- UML 이전
    - 1970 년대 중반부터 1980 년대 후반에 걸쳐 몇 개의 객체지향 모델링 언어가 개발되어 객체지향 분석과 설계에 관한 다양한 접근방식을 제시하였다. 1994 년에 이르러 서는 모델링 언어가 50 개가 넘게 되었다. 이들은 어느 하나만을 가지고는 완전한 만족감을 줄 수가 없었으며 “방법론 전쟁”이라는 말이 나돌 정도였다. 그 와중에서 Booch ’93, OMT, Fusion 의 방법론이 주목을 끌기 시작했고 이들은 서로 상대방의 기 법을 융합하여 OOSE, OMT-2, Booch ’93 의 이름으로 두드러지기 시작했다. 이들 각각은 나름대로 완전성을 갖추고 특정한 영역에 장점을 가진 것으로 인식되었다. 간단히 말하면, OOSE 는 사용사례를 중심으로 하여 업무 엔지니어링과 요구분석에 탁월하였고, OMT-2 는 분석과 자료 중심적인 정보시스템에 특히 풍부한 표현력을 가지 고 있었다. Booch ’93 은 설계와 구현 단계에 특히 유용하고 공학 중심적인 응용프로그램 쪽에서 많이 사용되었다
- Booch, Rumbaugh, Jacobson 이 힘을 합치다
    - UML 의 개발이 시작된 것은 1994 년 10 월, Grady Booch 와 Jim Rumbaugh 가 Rational 사에서 자신들의 두 방법론을 통합하는 작업을 시작한 때이다. 이미 두 방법론이 세계적으로 가장 선도적인 위치에 있었기 때문에 이 작업은 통일의 큰 가능성을 보 여주었다. 1995년 10월에 이 작업의 초안 0.8이 발표되었다. 1995년 가을에 Ivar Jacobson 과 그의 회사가 Rational 에 합류하여 UML 은 OOSE(Object-Oriented Software Engineering)까지 통합하게 되었다.

    - 이들은 다른 사람들의 피드백을 받아들이며 1996년 6월과 10월에 UML 0.9와 0.91 을 발표하였다.

- UML 1.0 - 1.1
    - 1996 년에 Object Management Group(OMG)는 객체지향 모델링 언어의 표준화를 위한 제안요청서를 발행하였다. 이에 따라 Rational 은 컨소시엄을 결성해 UML 1.0 을 생 산하고 이를 1997년 1월에 OMG에 제출하였다. 이 때 IBM & ObjecTime을 중심으 로 한 다른 컨소시엄에서 따로 제안된 것이 있었으며 이 제안이 역시 UML 에 통합되어 1997년 UML 1.1로 개정되었다. OMG에서는 1997년 11월에 UML 1.1을 표준으로 승인하였다.



- UML의 현재와 미래
  - OMG는 현재 UML 1.3을 확정하기 위해 노력하고 있으며 1999년 중반에 발표될 예 정이다. UML은 독점되지 않으며 모두에게 개방되어 있다. 이미 전반적 차원에서 산업계의 표준으로 받아들여지고 있다.

  - 비록 UML 이 정밀한 언어를 정의하고 있지만 그렇다고 해서 추후 개선의 여지가 없는 것은 아니다. 새로운 기법들이 차후 버전에 추가될 수 있을 것이며 현재의 UML 은 그 핵심을 기반으로 확장해 나갈 수 있도록 되어 있다.

  - 많은 도구들이 UML을 기반으로 함으로써 도구의 통합이 쉬어지고 구현을 위한 표준들이 가능해질 것이다. UML 은 많은 개념들을 통합시켜 왔으며 이에 따라 객체 지향의 사용이 더욱 가속화될 것이다. 컴포넌트 기반의 개발은 객체지향 기술과 맞 물려 있다. 최근 컴포넌트 기반의 재사용이 널리 확산되고 있지만 그렇다고 객체지 향 기술을 대체하는 것은 아니다. 컴포넌트와 클래스는 의미에 있어서 단지 미세한 차이만을 갖고 있을 뿐이다.

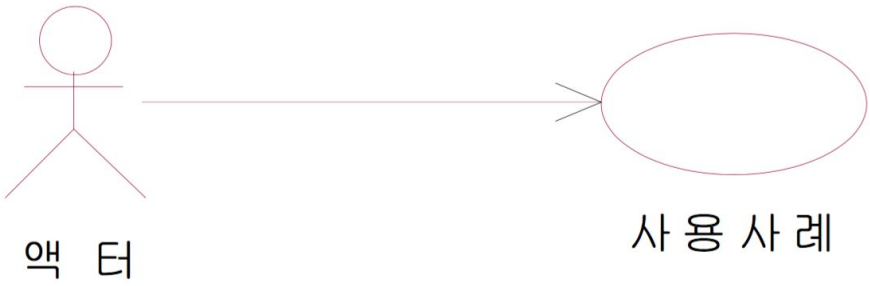
2.사용 사례

a. 액터와 사용사례

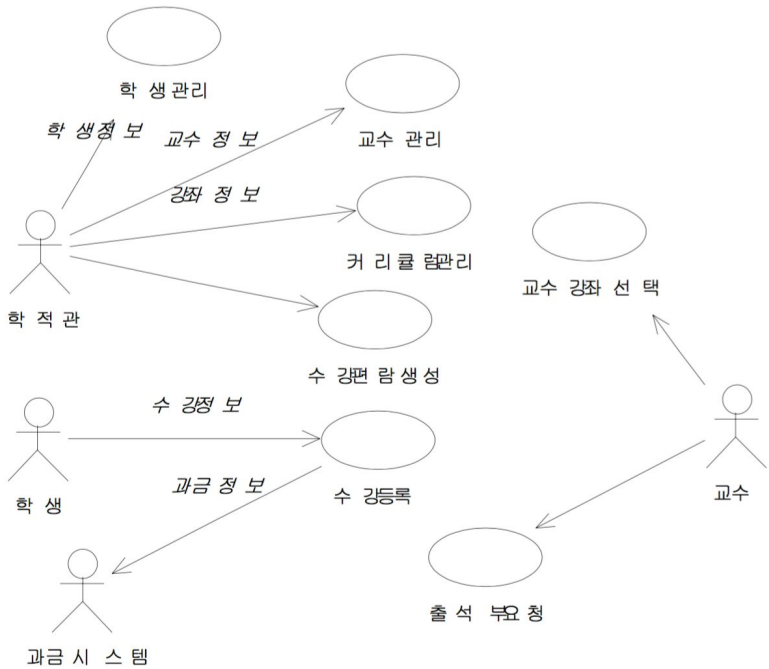
- 시스템의 요구사항은 누가 어떤 용도로 시스템을 사용하는지에 대한 명세서이고 이 를 간단하게 액터와 사용사례의 관계로 표현할 수 있다.

- 액터는 개발하려고 하는 시스템과 상호작용하는 사용자 또는 외부시스템, 장치 등을 의미한다. 사용사례란 사용자와 컴퓨터 시스템간의 전형적인 상호작용을 의미하며 시스템의 기능을 분류해 주는 역할을 한다.

b. 사용사례 다이어그램(Use case diagram)

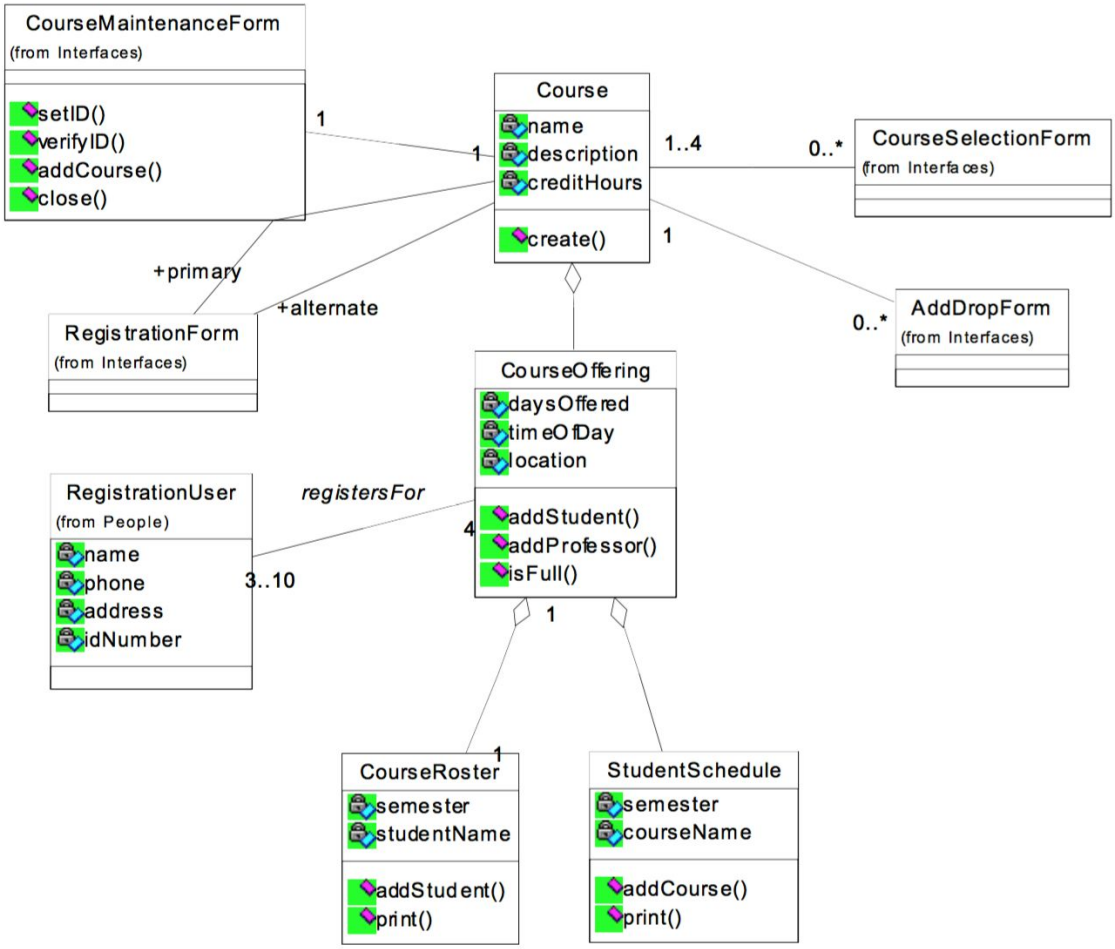


- 사용사례 다이어그램은 시스템의 요구사항을 엑터와 사용사례의 관계로 시각적으로 표현한 것이다.



3. 클래스 다이어그램

- 클래스 다이어그램은 시스템을 구성하는 객체들의 타입들과 그들간에 존재하는 다 양한 정적 관계들을 기술한다. 정적인 관계에는 연관(association)과 하위타입 (subtype)이 주로 사용된다. 연관이란 두 클래스가 관련이 있다는 것으로서 예를 들 어 수강관리시스템에서 학생이 강좌를 신청할 수 있음을 의미한다. 하위타입이란 클래스간에 상속을 받는다는 것으로서 예를 들어 학생이 등록사용자의 일종임을 의미한다. 또한 클래스 다이어그램은 클래스의 속성과 연산 및 객체들의 연결 방법에 적용되는 제약조건들을 기술한다.



a. 연관 (Association)

- 연관은 클래스간에 관계가 있음을 나타낸다. 수강관리시스템에서 학생이 강좌를 신 청한다거나 강좌에 따른 일정이 존재한다는 등의 예를 들 수 있다.

- 연관에는 관계에 참여하는 객체의 수를 멀티플리시티로 표시한다. 그림 4.1 에서 보 면, RegistrationUser 는 여러 개의 CourseOffering 을 신청할 수 있으며 한 CourseOffering 에는 3 명에서 10 명까지의 RegistrationUser 가 등록할 수 있음을 의미한다

b. 속성 (Attribute)

- 클래스의 속성은 클래스가 어떤 요소를 가지고 있음을 의미한다. 그림 4.1 에서, RegistrationUser 의 name 속성은 RegistrationUser 가 name 을 가지고 있음을 의미한다.

- UML 에서는 visibility name: type = defaultValue 와 같이 표기한다. 속성은 int 나 real 같 은 기본자료형일 수도 있고 string, date 등과 같은 작고 단순한 클래스일 수도 있다. 드물게는 크고 복잡한 클래스를 속성으로 가질 수도 있다.

c. 연산 (Operation)

- 연산이란 클래스가 수행하는 처리로서 메소드라고도 한다.

- UML 에서는 visibility name (parameter-list) : return-type-expression { property-string }과 같이 표기한다. visibility 에는 + (public), # (protected), – (private)의 세 종류가 있다.

d. 일반화 (Generalization)

- 일반화는 공통된 속성 또는 연산을 가진 클래스들에서 공통 요소들을 추출하여 상 위 클래스를 만들어내는 것을 의미한다. 학생, 교수 등의 클래스에서 등록사용자라 는 클래스를 추출해내는 것과 같다. 이때 등록사용자에 대한 모든 사항은 학생에게 도똑같이적용된다고볼수있다.
  공통요소를추출하는것과반대방향으로새로 운 요소를 추가하여 하위클래스를 만들어내는 것은 특수화라고 한다.

- 명세 관점에서 보면 일반화란 하위타입의 인터페이스가 상위타입의 인터페이스로부 터 모든 요소를 포함한다는 것을 의미한다. 구현 관점에서 보면 일반화는 프로그래 밍 언어의 상속과 연관되어 있다. 하위클래스는 상위클래스의 모든 메소드와 속성 을 상속받고 상속된 메소드를 재정의(override)할 수 있다.

e. 제약 규칙 (Constraint Rule)

- 클래스 다이어그램에는 많은 제약사항을 기록한다. 그림 4.1 은 한 CourseOffering 에 는 3 명에서 10 명까지의 RegistrationUser 가 등록할 수 있음을 나타내고 있다. 연관, 속성, 일반화 등을 통해서 위와 같은 중요한 제약조건들을 규정할 수 있다.

- 그 외의 제약조건들은 중괄호 {} 사이에 기술한다. 특별한 문법은 정해져 있지 않으며 읽기 쉬운 자연어를 쓰든가 좀 더 명확한 논리적 기술 또는 단편적인 프로그 램 코드를 사용해도 좋다.

4. 상호작용 다이어그램 (Interaction Diagram)

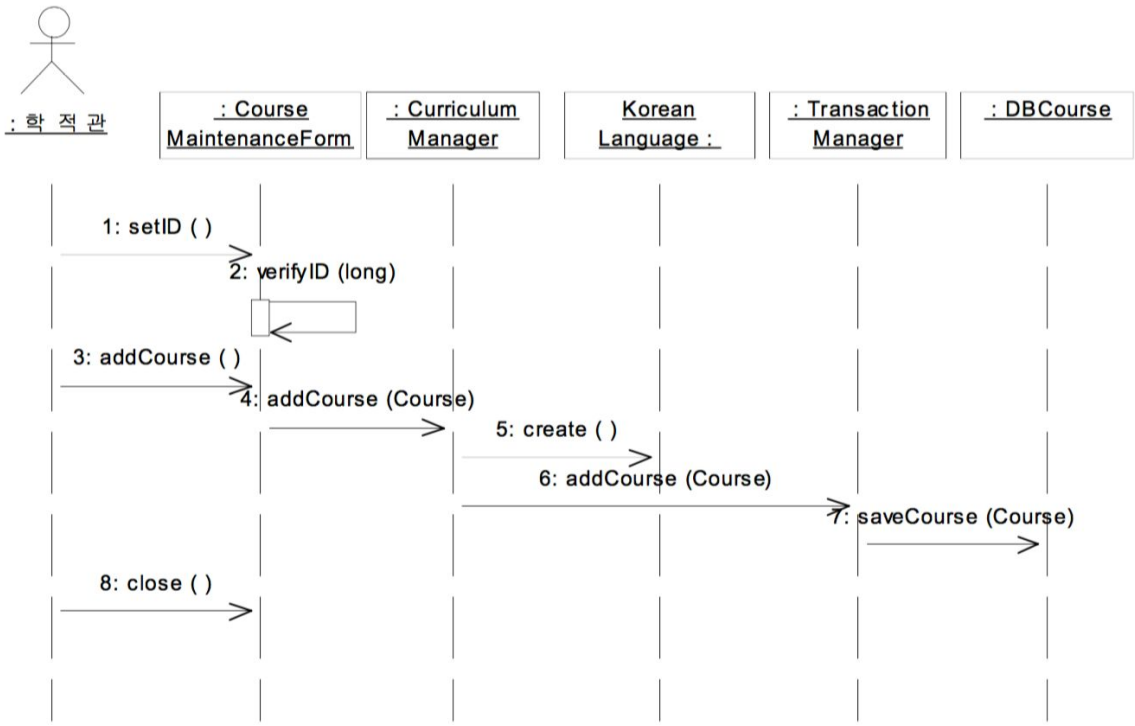
- 상호작용 다이어그램은 여러 객체들이 어떤 일을 처리하기 위해서 협동하는 행동양 식을 기술하는 모델요소이다.

- 전형적으로 하나의 상호작용 다이어그램은 사용사례 하나의 행동양식을 포착하여 나타낸다. 다이어그램에는 사용사례에 관련된 객체들과 그들간에 주고받는 메시지 가 표현된다.

- 상호작용 다이어그램에는 순차 다이어그램(sequence diagram)과 협동 다이어그램 (collaboration diagram)의 두 종류가 있다.

a. 순차 다이어그램 (Sequence diagram)

- 순차 다이어그램에서는 객체를 수직쇄선 위에 상자모양으로 표시한다.



- 이 수직선을 객체의 생존선(lifeline)이라고 하며 상호작용 동안의 객체의 생존을 나타낸다. 각 메시지는 두 객체 생존선간의 화살표로 표현한다. 이 메시지가 발생하 는 순서는 위에서 아래로 순차적으로 표시한다.

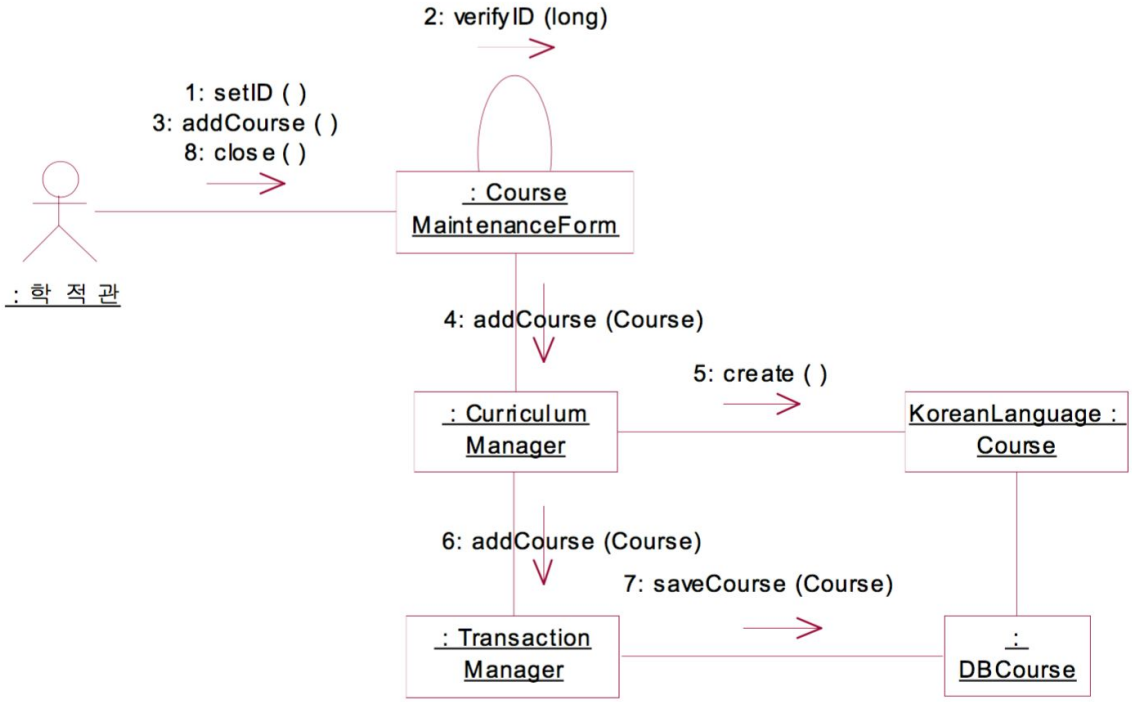
- 그림에서 보듯이 순차 다이어그램은 무척 단순하고 즉각적인 시각적 매력을 갖고 있으며 따라서 자주 사용된다.

b. 협동 다이어그램 (Collaboration Diagram)

- 협동 다이어그램은 순차 다이어그램과 같이 상호작용을 나타내는 또다른 표현기법 이다. 순차 다이어그램에서와 같이 객체들은 상자모양 아이콘으로 표시한다. 객체 들이 주고받는 메시지 역시 객체간의 화살표로 표시한다. 사용사례를 구현하기 위 한 메시지의 순서는 메시지에 번호를 매겨 표시한다.

- 다음 그림은 앞의 순차 다이어그램과 같은 내용을 협동 다이어그램으로 다시 표현 한 것이다.

- 메시지에 번호를 매기는 것은 순차 다이어그램보다 순서를 보기에 불편한 점이 있 다. 반면에 객체들을 공간적으로 배치시킴으로써 아키텍처 등 다른 중요한 정보를 강조할 수 있다.



c. 순차 다이어그램과 협동 다이어그램의 비교

- 순차 다이어그램과 협동 다이어그램은 같은 내용을 다르게 표현하는 기법이다. 엄 밀히 말하면 협동 다이어그램은 자료의 반환 흐름(data return flow)을 표현할 수 있다 는 점이 다르다. 두 그림은 상황에 맞추어, 개인적인 취향에 따라, 바꾸어 사용할 수 있다.

d. 언제 상호작용 다이어그램을 사용하는가?

- 상호작용 다이어그램은 하나의 사용사례 안에서 객체들의 행동양식을 표현할 때에 사용한다. 행동양식의 정밀한 정의를 표현하기에는 적절하지 않다.

- 여러 사용사례에 걸친 한 객체의 행동양식을 표현할 때에는 상태전이 다이어그램을 사용한다. 여러 사용사례에 걸쳐 있거나 쓰레드가 많은 행동양식을 표현할 때에는 활동 다이어그램을 사용한다.

5. 유스케이스 (Use Case)

- UML에서는 사용자의 목적 달성을 위해 시스템이 제공해야 하는 서비스와 서비스를 제공하기 위한 과정을 유스케이스로 표현한다.

a. Use Case의 정의

- 유스케이스란 개발될 시스템의 개개 액터가 시스템 사용 목적을 잘 달성할 수 있도록 개 발될 시스템이 제공해야 하는 서비스이다. 유스케이스는 타원 아이콘의 아래에 유스케이스 이름을 작성한다.

b. Use Case 이름

- 유스케이스 정의를 통해 우리는 유스케이스가 액터의 시스템 사용 목적 단위로 작성 되 어야 함을 알 수 있다. 따라서 유스케이스는 액터의 시스템 사용 목적으로 식별하고, 이름은 액터의 시스템 사용 목적에 서비스를 붙여서 작성한다.

c. Use Case와 시나리오

- 개발될 시스템이 액터에게 제공해야 하는 유스케이스를 찾고 난 후 우리는 어떻게 하면 그 유스케이스를 액터에게 가장 효율적으로 제공할 것인가를 고민해야 한다. 액터 가 시스템을 사용하면서 벌어질 여러 가지 상황들을 가정해서 일련의 시나리오들을 작성해야 한다.

d. 이벤트 플로어

- 액터의 시스템 사용 시나리오를 잘 작성하기 위해서는 시나리오 작성 방법에 대해 알아야 하며 그러기 위해서는 액터와 시스템의 역할과 책임에 대해서 알아야 한다.
- 액터의 책임
    - 시스템에게 서비스 수행을 시작하도록 요청해야 한다.
    - 시스템이 요구하는 정보를 제공해야 한다.
    - 시스템과의 상호 작용과 관련된 의사결정을 한다.
- 시스템의 책임
    - 액터로부터 제공 받은 정보나 시스템의 서비스 수행 상태를 기록해야 한다.
    - 서비스를 시작하거나 시스템이 요청한 행위를 수행하는데 액터가 필요로 하는 정보를 제공해야 한다.
    - 서비스를 제공해야 한다.

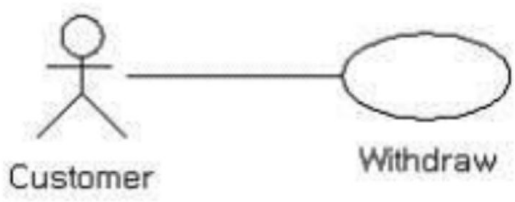
- 액터의 시스템 사용 목적을 달성하기 위해 액터와 시스템에 의해 수행되는 행위들의 흐름은 결국 사건의 흐름에 따라 진행된다. 사건의 흐름을 영어로는 이벤트 플로어(Event Flow)라고 한다.

6. 관계 (Relationship)

- 액터와 시스템 같이 목적 달성을 위해 고안되고 시도되는 커뮤니케이션을 목적 지향적인 커뮤니케이션이라고 한다. 즉 액터와 시스템은 목적 지향적인 커뮤니케이션을 하고 있는 것이다.

a. 엑터와 유스케이스의 관계

- 액터와 유스케이스 사이에 작성되는 연관(association)은 액터가 시스템의 어떤 기능을 사용하기 위해서 시스템과 상호작용하는가를 나타낸다. 액터와 유스케이스 사이에 실선으로 작성된다.



b. 엑터와 엑터의 관계

- 액터들 사이에는 단지 일반화 관계만이 존재한다. 액터의 일반화 관계가 의미하는 것은 하위 액터가 상위 액터의 모든 역할과 책임을 상속받는 것을 의미한다.

c. 유스케이스와 유스케이스의 관계

- 유스케이스와 유스케이스 사이에는 추가적으로 포함(include)과 확장(extend)이 작성된다. 공통부분에 해당하는 유스케이스와 이를 포함하는 유스케이스의 관계에는 include, 확장되는 유스케이스와 확장하는 유스케이스의 관계에는 extend 키워드를 갖는 의존관계 기호로 표현한다.

d. 엑터와 관련된 문제들

- 액터가 없는 유스케이스
  - 시스템이 존재하는 이유는 액터의 시스템 사용 목적을 달성하도록 하기 위해서다. 따라서 액터가 없는 유스케이스는 존재할 수 없다.
  -
- 잘못된 유스케이스 이름
  - 유스케이스를 찾을 때 중요하게 거론되는 것이 액터에게 측정 가능한 '가치(value)'를 제공해야 한다는 것이다 여기서 가치라는 단어를 잘못 이해하게 되면 유스케이스 이름이 너무 추상적으로 작성된다.
    '가치'라는 단어를 생각할 때는 반드시 잊지 말아야 할 것이 시스템을 사용해서 얻는 가치라는 것이다.

e. 관계와 관련된 문제들

- 일방향 연관으로 표현된 액터와 유스케이스
  - 액터와 유스케이스의 상호 작용은 항상 액터의 요청에 의해 시작된다고 하였기 때문에 커뮤니케이션의 시작을 표현하기 위한 액터에서 유스케이스로의 화살표 표시는 무의미하다.
- 단일 유스케이스에 포함되는 유스케이스
  - 포함 관계는 유스케이스의 이벤트 플로어들에 공통된 부분이 발견되었을 때 작성하는 관 계이고, 공통된 부분이 발견되었다는 것은 두 개 이상의 유스케이스들에서 공통부분을 갖고 있다는 것을 의미한다.

포함 관계가 나타나면 항상 두 개 이상의 유스케이스들이 공통 부분에 해당하는 유스케이스를 포함해야 한다.

7. 상태 (State)

a. 이벤트 (Event)

- 객체의 서비스 수행을 요청하는 일들과 시간의 경과, 조건의 변경

b. 이벤트 종류

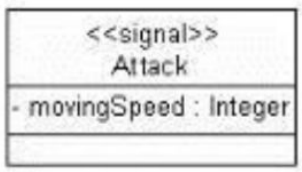
- 오퍼레이션을 직접적으로 호출하는 이벤트를 오퍼레이션 호출 이벤트라고 하고, 시그널을 전송하는 이벤트를 시그널 전송 이벤트라고 한다. 시간 경과를 나타내는 이벤트를 시간 이벤트(Time Event)라고 하고, 외부조건의 변경을 변경이벤트(Change Event)라고 한다.

c. 오퍼레이션 호출 이벤트

- 오퍼레이션 호출 이벤트는 직접적으로 객체의 오퍼레이션을 호출할 때 발생하는 이벤트이다. 오퍼레이션 호출 이벤트는 객체의 상태를 변화시킬 수도 있고, 상태를 변화시키지 않을 수도 있다. 객체의 상태를 변화시키는 이벤트가 상태 머신에 표현된다. 이벤트 이름은 오퍼레이션 이름과 오퍼레이션의 매개변수들을 사용해서 작성한다.

d. 시그널 전송 이벤트

- 시그널 전송 이벤트에 의해 송신 객체에 시그널이 보내진다. 시그널은 화재경보장치 등 에서 보내는 경보음과 같이 비동기적으로 전달되는 신호를 나타낸다. 시그널은 <<signal>> 키워드를 갖는 분류자 기호로 표현한다. 그림은 외부침입이 발생했다는 신호를 표현한 것으로 속성으로 침입자의 이동속도를 갖는다.



- 시그널 전송 이벤트는 시그널 이름을 이벤트 이름으로 사용하고, 시그널 속성을 이벤트 의 매개변수로 사용해서 'attack(10)'과 같이 작성한다. 객체의 행위는 오퍼레이션으로 명세되고, 객체는 오퍼레이션 호출 이벤트에 대응하는 오퍼레이션을 수행한다. 시그널 전송 이벤트에 대응하는 행위는 <<signal>> 키워드를 갖는 오퍼레이션 기호로 나타낸다.

e. 변경 이벤트

- 변경이벤트는 하나 이상의 속성들이나 링크들의 변화로서 불린 식이 참이 될 때 발생 하는 이벤트를 명세합니다. 변경이벤트는 일반적으로 'when 현재날짜>주문일자+14'와 같이 키워드 when 을 사용해서 표현할 수 있습니다.

f. 시간 이벤트

- 시간이벤트는 설정된 시간만큼 시간이 경과하거나 설정한 시간이 되면 발생하는 이벤 트입니다. 시간의 경과는 시간 측정을 시작했을 때를 기준으로 경과되는 시간을 나타 내기 위해 'after(5 초)'와 같이 after 라는 키워드를 사용합니다. 특정 시간을 설정할 때 는 '2003 년 11 월 5 일 오후'와 같이 일반적인 시간표현방법을 사용합니다

g. 상태 머신 (State Machine)

- 상태머신다이어그램은 상태머신을 명세하는 다이어그램입니다. 상태머신은 객체의 상태 를 명세하기 위해서도 사용되지만 객체가 수행하는 역할들을 명세하는 인터페이스의 상 태를 명세하기 위해서도 사용될 수 있습니다.

h. 프로토콜 상태 머신 (Protocol State Machine)

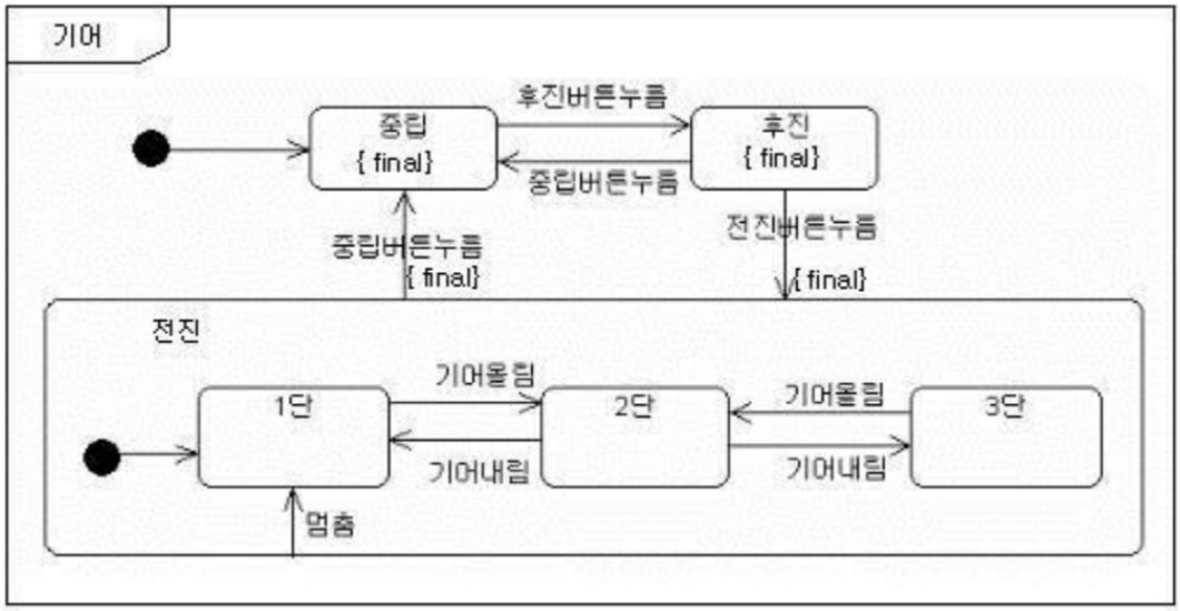
- 인터페이스의 상태머신을 객체의 상태머신이 지켜야 하는 규약(계약)이라는 의미에서 프 로토콜상태머신이라고 합니다. 인터페이스는 명세로서 내부 구현을 갖지 않기 때문에 구현부분에 해당하는 액션들을 표 현할 수 없습니다. 따라서 프로토콜상태머신에 표현되는 상태들은 진입액션과 탈출액션 과 액티비티를 갖지 않습니다. 또한 상태를 저장할 수 없기 때문에 얕은이력과 깊은이력 을 가질 수 없습니다.

i. Protocol Conformance 관계

- . Protocol Conformance 는 일반화 관계나 실현 관계에 사용되는 프로토콜 상태 머신들 사이 의 관계로 상위 프로토콜 상태 머신의 명세 내용을 하위 프로토콜 상태 머신이 따라야 합니다.

j. 상태 머신 확장

- 상태 머신은 일반화 가능하고, 하위 상태 머신은 상태나 전이를 추가하거나 재 정의함으로서 상위 상태 머신을 확장합니다. 상태머신은 확장을 고려해서 상태나 전이가 확장될 수 있는지의 여부를 명세해야 합니다. 다른 상태머신에서 상태나 전이가 확장되지 않는다는 것을 표현하기 위해서 그림과 같이 {final}이라는 키워드를 사용합니다.



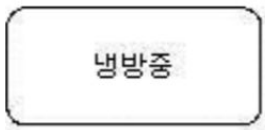
- 확장하는 상태머신은 {extended}라는 키워드를 사용한다.
- 확장관계나 실현관계를 표현하기 위해서 상태머신은 <> 키워드를 갖는 분류자로 표현할 수 있습니다.

k. 상태

- 상태란? 몇몇 불변 조건(invariant condition)들이 유지되는 동안의 상황을 모델링하는 것이다. 객체는 대응해야 하는 이벤트가 발생되기를 기다리는 정적인 상태와 액티비티를 수행 하고 있는 동적인 상태를 가질 수 있다. 일반적으로 정적인 상태는 Connect 나 Connected 와 같이 표현한다. 자신의 상태를 수동적으로 표현할 때는 과거형을 사 용한다. 동적인 상태는 Connecting 과 같이 액티비티가 수행하고 있음을 나타내기 위해서 진행형으로 표현한다.

l. 상태 종류

- 객체의 상태는 다른 상태를 포함하는지의 여부에 따라 단순상태와 복합상태로 구분됩 니다. 단순상태는 다른 상태를 포함하지 못하는 것이고, 복합상태는 다른 상태를 포함 하는 것으로 포함되는 상태를 하위상태라고 부릅니다. 상태는 액티비티 기호와 동일하게 모서리가 둥근 사각형 기호를 사용합니다. 그림에서 냉방중이라는 진행형으로 작성된 상태이름을 통해 냉방을 위한 어떤 액티 비티가 계속적으로 수행되고 있다는 것을 알 수 있습니다.



- 의사상태
  - 의사상태는 전이에 대한 제어구조를 표현하는 것.
  - 상태들의 전이에 대한 제어를 위해 entryPoint, exitPoint, initial, deepHistory, shallowHistory, join, fork, junction, terminate, choice 와 같은 의사상태 (PseudoState)가 존재합니다.
  - 사용자들은 의사상태를 상태전이에 대한 제어구조로 생각하면 됩니다.

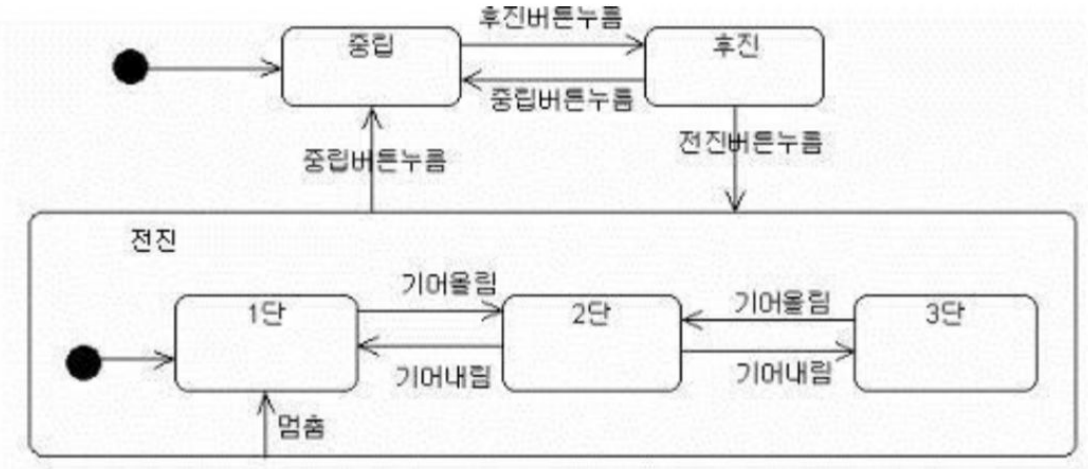
- 종료상태 (Final State)
  - 종료상태는 상태 머신의 모든 전이가 완료되었다는 것을 의미합니다. 복합상태 에서 종료상태가 사용되고, 종료상태로 전이가 되었다는 것은 자신이 포함하는 모든 상태들의 전이가 완료되었다는 것을 의미하고, 해당 상태에서 빠져 나오 게 됩니다. 종료상태는 액티비티 다이어그램의 종료 점과 동일한 기호를 사용 합니다.

m. 전이 (Transitions)

- 객체의 한 상태에서 이벤트에 의해 다른 상태로 이동하는 것을 전이라고 합니다. 객체는 상태를 변화시킬 수 있는 이벤트가 발생하고, 명세된 경계조건변화를 만족시킬 때 전이이전의 상태에서 명세된 액션들을 수행하고, 전이되는 상태로 들어갑니다. 전이는 '이벤 트이름(인자리스트)[경계조건]/액션리스트'와 같은 형식으로 작성됩니다.

n. 복합 상태

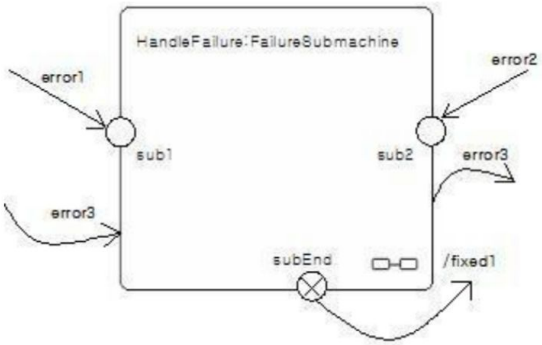
- 복합상태는 다른상태를 포함하는 상태이다.



- 그림에서 전진이 상위상태이고, 1 단과 2 단과 3 단은 하위상태입니다. 하위상태는 상위상태의 전이와 액션들을 상속받기 때문에 전진의 하위상태들은 모두 중립버튼누름 이벤트에 의해 중립으로 전이됩니다. 중립에서 전진버튼누림 이벤트가 발생하면 전진의 시작상태인 1 단이 됩니다. 멈춤에 의한 이벤트에 의한 전이는 전진상태와 연결되어 있기 때문에 1 단과 2 단과 3 단 모두가 이 전이를 상속받습니다. 따라서 전진의 어느 단계에 있든지 멈춤이라는 이벤트에 의해 1 단상태가 될 수 있다. 전진상태는 추상의 의미를 갖고, 실제 구체적인 상태는 1 단과 2 단과 3 단이 됩니다.

o. 서브 머신 상태

- 컴포넌트 기반 개발과 같이 부분들에 해당하는 상태 머신. 서브머신 상태는 다른 많은 서브머신에서 참조되어 재사용될 수 있기 때문에 특정 서브 머신에 종속되도록 표현하면 안됩니다. 컴포넌트의 인터페이스와 같이 외부 상태에서 전이될 수 있는 통로만을 제공하면 된다. 서브머신 상태는 서브머신 상태로 들어올 수 있도록 진입 점(entry point)을 제공하고, 서브머신 상태에서 나갈 수 있도록 탈출 점(exit point)을 제공합니다. 그림은 서브머신 상태가 상태 머신에서 참조되는 방법을 설명합니다. 서브머신 상 태가 상태머신에서 사용될 때의 역할은 에러를 처리하는 것이기 때문에 HandleFailure 로 이름이 작성되었다.



8. 배치 (Deployment)

a. 액티브 클래스 (Active Class)

- 액티브 클래스의 객체들은 능동적인 객체들로서 객체가 생성되면서 자신의 행위를 시작 하고, 행위가 완료되거나 자신의 제어흐름을 가지고 있는 외부적 객체에 의해 중단될 때 까지 멈추지 않고 실행합니다. 액티브 클래스의 객체를 액티브 객체라고 합니다. 적합한 의미를 갖는 우리말로는 '활성클래스'와 '활성객체'가 있습니다. 액티브 클래스는 그림과 같이 양쪽에 수직선을 갖는 분류자 기호로 표현합니다.

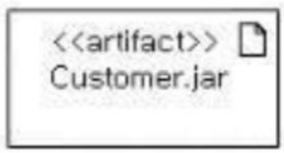
- 액티브 클래스를 사용해서 프로세스와 쓰레드를 모델링할 수 있습니다. 병행처리를 계획 해야 하는 시스템의 경우 액티브 클래스를 사용해서 제어흐름에 대한 이름을 작성할 수 있기 때문에 프로세스와 쓰레드의 계획과 관리를 용이하게 합니다.

b. 배치 (Deployment)

- 시스템 개발에 있어 배치에 대한 모델을 작성하는 것은 시스템이 실행에 대한 요구사항 에 따라 의사결정을 하는 것으로 시스템의 실행아키텍처라고 할 수 있습니다. 요구사항 과 '강하게 결합된 컴포넌트들은 어떤 것인지?, 많은 자원을 요구하는 컴포넌트는 어떤 것인지? 변화가 예상되는 것들은 어떤 것인지?'등의 질문에 따라 구현된 컴포넌트를 배 치하기 위한 계획을 세웁니다. 잘못된 배치계획은 필요로 하는 공간보다 적은 공간 안에 서 자신의 역량만큼 책임을 수행하지 못하는 컴포넌트를 만들거나, 필요하지 않는 공간 을 차지하면서 시스템 자원을 낭비하는 컴포넌트를 만들 수 있습니다.
- 시스템 배치모델은 노드들과 노드들을 연결하는 커뮤니케이션 경로와 노드에 배치되는 아티팩트들로 구성되어 있으며 배치다이어그램에 작성됩니다.

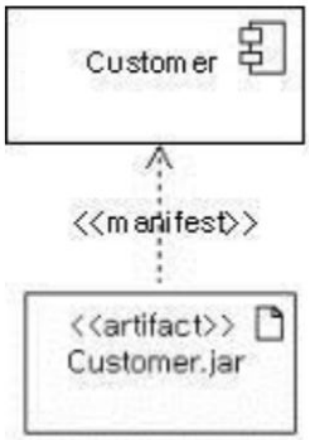
c. 아티팩트 (Artifact)

- 아티팩트는 시스템 개발에 관련된 물리적인 형태를 갖는 모든 정보들을 의미합니다. 아 티팩트의 예로는 모델파일, 소스파일, 스크립트와 바이너리의 실행파일과
- 데이터베이스의 테이블과 개발 산출물들과, 워드프로세스 문서나 메일 메시지 등이 있다. 아티팩트는 그림 11.2 와 같이 <>라는 키워드를 갖는 분류자 기호로 표현합니 다. 오른쪽 상단에 선택적으로 노트 기호와 같은 아이콘을 사용할 수 있습니다.



d. Manifestation 관계

- Manifestation 은 추상(Abstraction)관계의 특수한 형태로, 목록이라는 의미를 가지고 있으 며, 아티팩트에 포함되는 모델요소들을 나타내기 위한 관계입니다. Manifestation 관계는 그림과 같이 <>라는 키워드를 갖는 의존관계로 표현합니다.



e. 표준 스테레오 타입

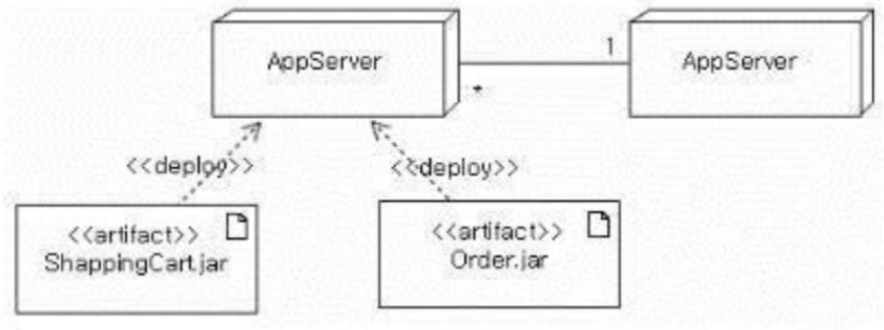
- 아티팩트는 개발되는 시스템 범위와 관련된 물리적 파일을 나타내는 <>과 파일의 특수한 형태들을 나타내는 <<document>>, <<source>>, <<library>>, <<executable>> 을 스테레오타입으로 갖습니다.
- <<document>> : 문서와 같은 일반적인 파일을 나타낸다.
- <<source>> : 실행 가능한 파일로 컴파일 될 수 있는 파일
- <<library>> : 정적 또는 동적 라이브러리를 나타낸다
- <<executable>> : 컴퓨터 시스템에서 실행되어질 수 있는 프로그램파일을 나타낸다.

f. 노드

- 노드는 실행시간에 존재하고, 컴퓨터자원(computational resource)을 나타내는 물리적인 요소이다. 노드는 일반적으로 컴포넌트가 배치되어지는 프로세서나 장치를 나타낸다. 노 드들은 네트워크 구조를 정의하기 위한 커뮤니케이션 경로를 통해서 상호 연결되어질 수 있다. 노드는 육각형기호로 표현합니다. 그림은 AppServer라는 노드의 인스턴스를 표현하 고 있습니다.

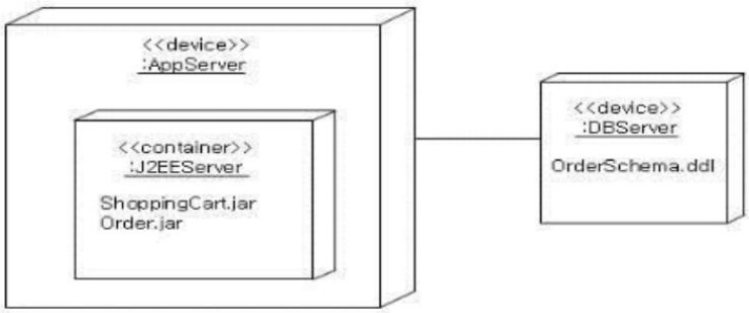
g. 커뮤니테이션 경로 (Communication Paths)

- 노드들 사이의 연결은 커뮤니케이션경로라는 특별한 연관관계에 의해서 표현됩니다. 그림은 DBServer 가 여러 개의 AppServer 와 메시지를 교환한다는 것을 표현하 고 있습니다.



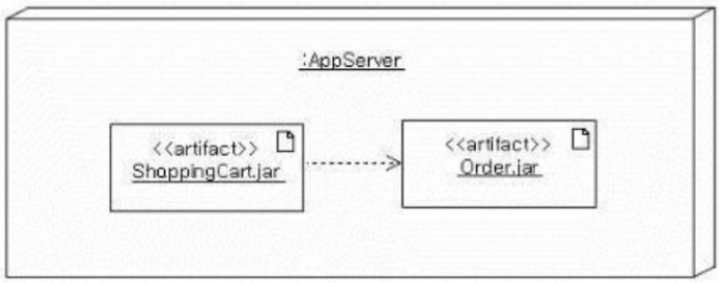
h. 장치 (Devices)

- 아티팩트가 실행되기 위해 배치될 수 있는 프로세싱 기능을 갖는 물리적 컴퓨터 자원 이다. 장치는 특별한 형태의 노드로, 다른 장치들을 포함할 수 있으며 <>라는 키 워드를 갖는 노드 기호로 표현할 수 있습니다.



i. 배치 관계

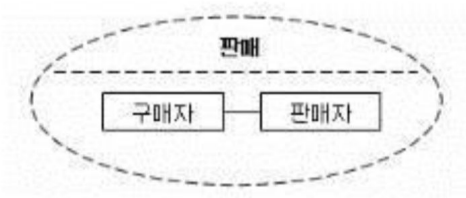
- 배치는 의존관계의 특수한 형태로, 배치대상(노드)에 아티팩트를 할당하는 관계입니다. 노드와 아티팩트는 분류자의 특수한 형태이기 때문에 배치관계는 분류자의 범위에서 작성될 수도 있고, 인스턴스 범위에서 작성될 수도 있습니다. 배치관계는 그림과 같이 노드에 아티팩트를 직접 포함하는 것과 아래 그림과 같이 노드에 아티팩트 이름을 리스트하는 방법이 있습니다.



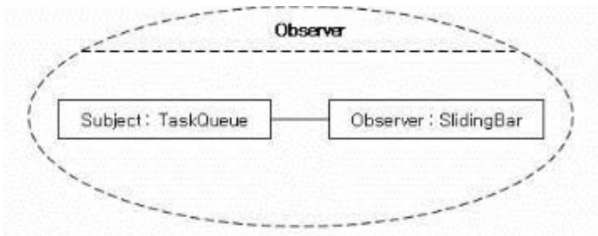
9. 컬레보레이션 (Collaboration)

a. 기호

- 컬레모레이션의 표현은 그림과 같이 컬레보레이션 이름을 포함하는 점선으로 된 타원 기호를 사용한다. 컬레보레이션 내부에는 컬레보레이션에 참여하는 역할과 역할을 연결하는 커넥터가 작성된다.
- 그럼에는 판매가 이루어지기 위해서는 구매자라는 역할과 판매자라는 역할이 필요하고, 그들 사이에 협력이 요구됨을 보여주기 위해서 구매자와 판매자를 커넥터로 연결한다.



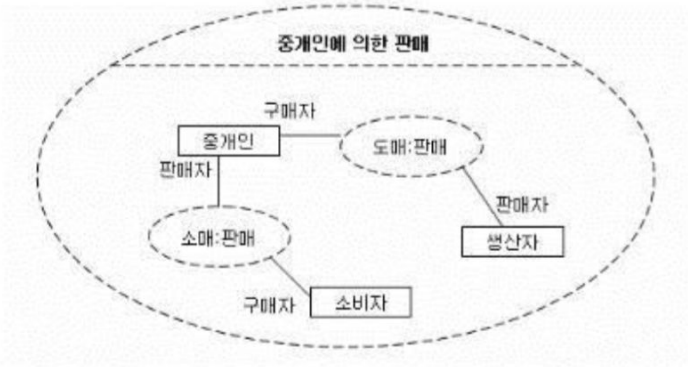
- 역할을 구현하는 클래스를 명시하고자 할 때는 역할 이름 뒤에 ":"와 클래스 이름을 작성한다.
- 아래 그림은 구성요소의 클래스가 TaskQueue와 SlidingBar이고, 이들의 구성요소들이 수행하는 역할이 Subject와 Observer임을 보여주고 있다.

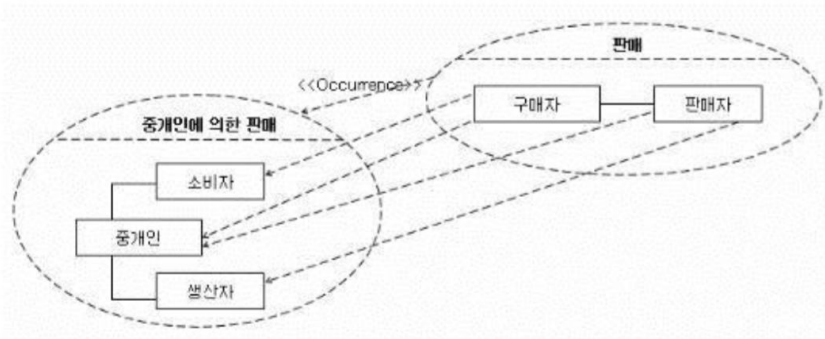


b. 어커런스

- 컬레보레이션에 대한 특별한 상황에 대한 적용을 컬레보레이션 어커런스라고 한다. 컬레보레이션 어커런스는 특수한 상황에서의 협력을 나타내기 때문에 이것 또한 컬레보레이션이다. 쉽게 말해서 구체화된 컬레보레이션이다.
- 예를 들어, 특정 항로를 따라 대상물들을 목적지까지 데려가야 하는 운항에는 이동수단을 조종하는 역할과 대상물들을 보호하는 역할과 대상물의 역할이 있다.
- 운항은 대상물을 목적지까지 데려가야 하는 목적을 가지고 있고, 그 목적을 달성하기 위 해서는 여러 역할들이 필요하고, 그 역할들의 협력이 요구되는 컬레보레이션이다. 운항의 특수한 예로는 항공기운항, 선박운항과 같은 것들이 있습니다. 항공기운항의 경우 이동수단을 조종하는 역할은 조종사이고, 대상물들은 승객과 화물이며, 대상물을 보호하는 역할은 승무원이다. 항공기운항은 운항이라는 컬레보레이션의 특수한 사용에 해당하는 컬레보레이션 어커런스이다. 또한 항공기운항은 조종사와 승객과 화물과 승무원 역할들의 협력을 나타내는 컬레보레이션이다.
- 컬레보레이션은 여러 가지 구체적인 상황에서 발생하는 협력들을 추상화한 것으로 행위 의 재 사용을 위한 매우 중요한 요소다. 만약 여러분이 대부분의 전자상거래에서 필 요한 예약이나 주문에 대한 컬레보레이션을 잘 명세한다면, 이 컬레보레이션은 많은 시 스템 개발에서 컬레보레이션 어커런스로 재사용 될 것이다. 컬레보레이션을 정확히 표현하기 위해서는 구조적인 부분과 행위적인 부분을 모두 표현해야 한다. 어떤 요소들이 협력하는가는 구조적인 부분으로,

협력을 위해 어떻게 상호 작용하는 가는 행위적인 부분으로 표현되어야 한다. 일반적으로 컬레보레이션의 구조 적인 부분은 컴포지트 스트럭처 다이어그램(Composite Structure Diagram)과 같은 스트럭처 다이어그램(Structure Diagram)으로, 행위적인 부분은 시퀀스 다이어그램(Sequence Diagram)과 커뮤니케이션 다이어그램(Communication Diagram)와 같은 인터렉션다이어 그램(Interaction Diagram) 들로 표현한다. 컬레보레이션 어커런스는 그림과 같이 표현된다. 대안적인 표현으로 아래 그림과 같이 나타낼 수도 있다. 개념에 대한 실체 표현과 같이 컬레보레이션 어커런스는 이름 다음에 ':'을 작성하고, 어 커런스되는 컬레보레이션 이름을 작성한다. 아래 그림 판매 컬레보레이션의 두 개의 컬레보레이션 어커런스들로 중개인에 의한 판매 컬레보레이션을 구성한 것을 설명하고 있다. 중개인이라는 역할은 구매자의 역할과 판매자 역할 둘을 모두 수행하고 있다.





10. 인터렉션 (Interaction)

a. 메시지 표현

- 메시지의 이름은 서비스 요청내용을, 입출력 매개변수들은 책임과 권한들로 구성된다. 서비스를 찾는 단계에서 작성되는 메시지는 어떤 정보들이 입력, 출력 또는 리턴으로 사용되는지에 대한 의미만을 표현하면 된다. 요청메시지는 'Multiply(n1, n2)', 응답메시지는 'Multiply(_):resultValue' 로 표현된다. 시뮬레이션이나 테스트 단계에서는 요청메시지로 'Multiply(3, 5)', 응답메시지는 'Multiply(_):15' 와 같이 표현 된다.

b. 메시지 분류

- 메시지들은 수행하는 일의 성격에 따라 서비스를 요청하는 요청메시지, 요청에 대한 결과를 리턴해주는 응답메시지, 구성요소를 생성하는 생성메시지, 구성요소를 소멸하는 소멸 메시지로 나눠진다.
- 또한, 메시지를 보낸 이벤트와 받은 이벤트가 알려졌는지에 따라 Complete Message, Lost Message, Found Message 로 나눠진다. 동기적인 방법에서는 서비스 요청자가 서비스를 요청하고 나면 제어가 서비스 제공자로 넘어간다. 서비스 제공자가 서비스 수행을 완료한 후에야 다시 제어가 서비스 요청자에게 넘어와서 다음 작업을 수행하게 된다.
- 비동기적인 방법으로는 서비스 수행여부에 상관없이 제어권을 잃지 않고 바로 다음 작업을 수행할 수 있다.
- 전자와 같은 방법으로 서비스 요청을 전달하는 메시지를 Synchronous Message
- 후자와 같은 방법으로 서비스 요청을 전달하는 메시지를 Asynchronous Message 라 한다.
- 동기적인 방법에서 요청된 결과가 있을 때 결과를 넘겨주는 메시지를 응답 메시지라 한다.
- 시스템 개발 초기에는 인터렉션에 작성되는 메시지가 동기인지 비동기인지 정확하게 결정하기가 어려우므로 간단한 메시지 표현으로 비동기 메시지의 표현과 같은 머리부분이 열려있는 화살표를 사용한다.
- 오퍼레이션 호출은 서비스 요청자가 직접적으로 서비스 제공자에게 서비스를 요청하는 것이고, 시그널 송신은 서비스 요청자가 특정한 신호를 보내어 간접적으로 요청하는 것 응답메시지는 결과를 제공해주기를 원하는 서비스 요청 메시지에 대응되는 것으로 특별한 이름을 사용해야 하는 경우를 제외하고는 서비스 요청 메시지의 이름을 따른다. 예를 들어 foo(3) 에 대한 응답메시지는 foo(_):15 와 같이 작성된다.
- Complete Message 는 보낸 이벤트와 받는 이벤트 모두 알려진 경우 , Lost Message 는 보낸 이벤트는 알려져 있으나 받는 이벤트는 알려지지 않은 경우, Found Message 는 받는 이벤트는 알려졌으나 보낸 이벤트가 알려지지 않은 경우 이다. Lost Message 는 메시지가 목적지에 도착되지 않았다는 것을 의미하고, Found Message 는 메시지의 시작이 메시지를 기술하는 범위 밖에 있다는 것을 의미한다.

c. 생명선

- 참여자의 이름은 '구성요소/역할이름:구성요소클래스' 와 같은 형태로 작성된다. 컬렉션에서 포함된 부분을 찾는 표현식을 Selector 라 한다.

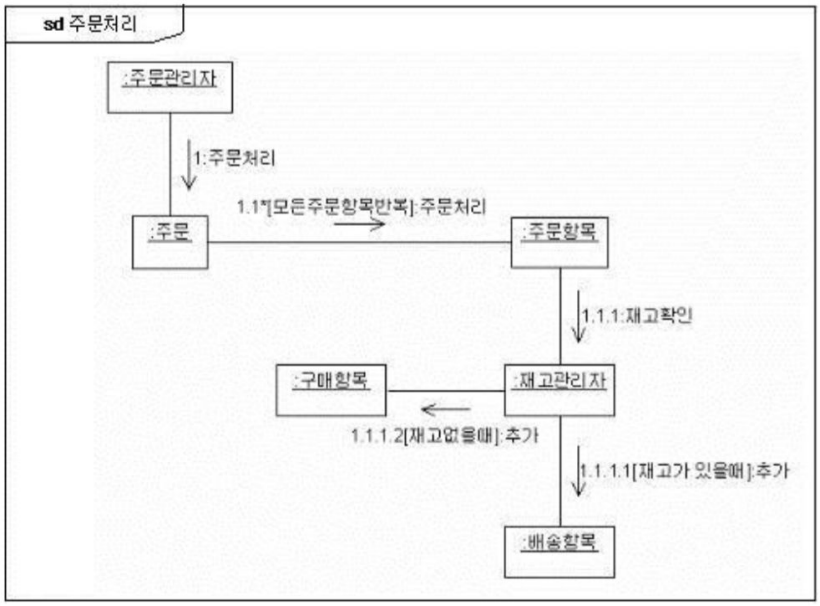
d. 인터렉션 다이어그램

- 인터렉션 다이어그램은 인터렉션에 대한 그래픽 표현입니다. 인터렉션은 매우 복잡한 모델요소이고, 다양한 관점을 가지고 작성할 수 있습니다. UML 2 는 인터렉션의 다양한 관점에 따른 다이어그램들을 제공합니다. 인터렉션 다이어그램에는 구성요소들의 상호작용에 있어 시간 제약을 중요시 하는

시퀀스 다이어그램, 관계 제약을 중요시하는 커뮤니케이션 다이어그램, 시간 제약을 중요시하는 타이밍 다이어그램, 전체적인 개괄을 보고자하는 인터렉션 오버뷰 다이어그램이 있습니다. 모든 인터렉션 다이어그램은 시퀀스 다이어그램과 같이 sd 라는 키워드를 갖습니다. 지금까지는 인터렉션에 대해서 시퀀스 다이어그램을 주로 사용해서 설명하였습니다.

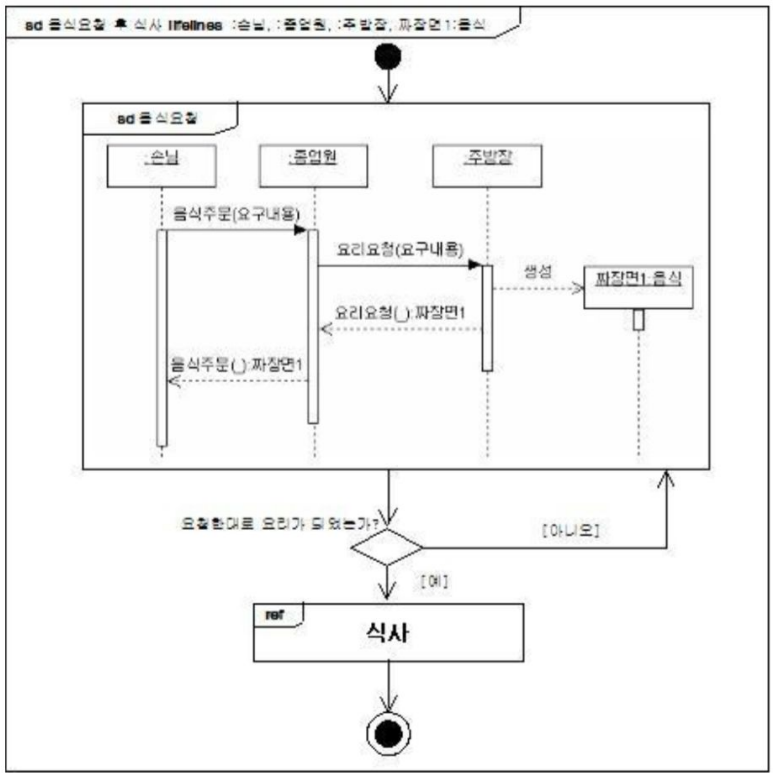
e. 커뮤니케이션 다이어그램

- 커뮤니케이션 다이어그램은 구성요소들이 상호작용하는데 있어 어떻게 관계를 맺는지에 중점을 둔 다이어그램입니다. 따라서 커뮤니케이션 다이어그램에는 구성요소들을 표현할 수 있는 생명선과 그들의 연결과 메시지로 표현됩니다.
- 시퀀스 다이어그램은 시간적인 순서로 메시지들의 전송 순서를 작성하기 때문에 메시지 전송 순서를 분명하게 표현하지 않아도 되지만, 커뮤니케이션 다이어그램은 하나의 생명 선에서 여러 개의 메시지들을 주고 받기 때문에 메시지 전송의 선후행관계를 알기 어렵 습니다. 따라서 커뮤니케이션 다이어그램에 표현되는 메시지는 '1.1:myMessage'와 같이 메시지 전송순서를 명확히 표현해야 합니다.
- 번호의 작성은 제어의 계층으로 작성됩니다.
- 한 제어 안에서 동시적으로 처리되어야 하는 메시지들을 포함하고 있다면, '1.1.a:myMessage1, 1.1.b:myMessage2'와 같이 일반적으로 알파벳을 사용하여 작성합니다.
- 어떤 메시지는 반복적으로 전송되거나, 또는 조건에 따라 전송여부가 결정되거나, 병행처 리될 수 있습니다.
- 반복표현은 '*[i:=1..n]'과 같이 별표 다음에 반복횟수를 대괄호 안에 작성합니다.
- 조건표현은 [a>b]와 같이 대괄호 안에 조건식을 작성합니다.
- 병행처리는 "*||"와 같이 별표 다음에 두개의 수직선으로 작성합니다.
- UML2.0 에서는 반복이나 조건의 표현식에 대해서는 표식 형식을 제시하지 않고 있기 때 문에, 가상코드나 프로그래밍코드를 사용할 수 있습니다.
- 그림은 주문관리자가 주문에게 주문처리를 요청하면, 주문은 모든 주문항목들에게 주문처리를 요청합니다. 모든 주문항목에 대해서 반복하는 메시지이기 때문에 반복기호 별표를 하고 대괄호 안에 반복조건을 작성하였습니다. 각각의 주문항목은 재고관리자에 게 해당하는 물품이 있는지 확인을 요청합니다. 재고관리자는 재고가 없으면 구매항목에 해당물품을 추가하고, 재고가 있으면 해당물품을 배송항목에 추가합니다. 이 메시지들은 조건에 의해 처리되기 때문에 대괄호 안에 조건을 작성하였습니다. 메시지에 작성된 화 살표는 메시지의 방향을 나타냅니다. 만약 주문항목들에 대해서 병행처리를 하고 싶다면 '1.1*||[모든주문항목반복]:주문처리'와 같이 작성할 수 있습니다.


sd 주문처리
:주문관리자
1:주문처리
1.1*[모든주문항목반복]:주문처리
:주문
:주문항목
1.1.1:재고확인
:구매항목
:재고관리자
1.1.1.2[재고없을때]:추가
1.1.1.1[재고가 있을때]:추가
:배송항목

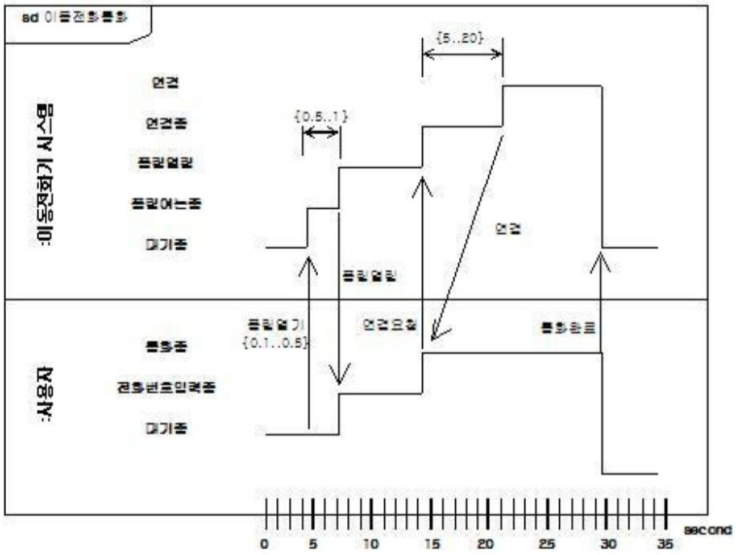
f. 인터렉션 오버뷰 다이어그램

- UML 이전 버전에서의 시퀀스 다이어그램은 시나리오의 제어흐름을 표현하기가 어려웠습니다. UML 2.0 에서 여러 가지 연산자들이 추가되었지만 여전히 제어흐름을 표현하기 는 어렵습니다. 제어흐름을 가장 잘 표현할 수 있는 것은 액티비티 다이어그램입니다. 하지만 액티비티 다이어그램은 액티비티들을 중요시 하기 때문에 어떻게 구성요소들이 상호작용하는지는 표현하기가 힘듭니다. UML 2.0 에서 두 다이어그램의 장점들을 취합해 서 만든 것이 인터렉션 오버뷰 다이어그램입니다.



g. 타이밍 다이어그램

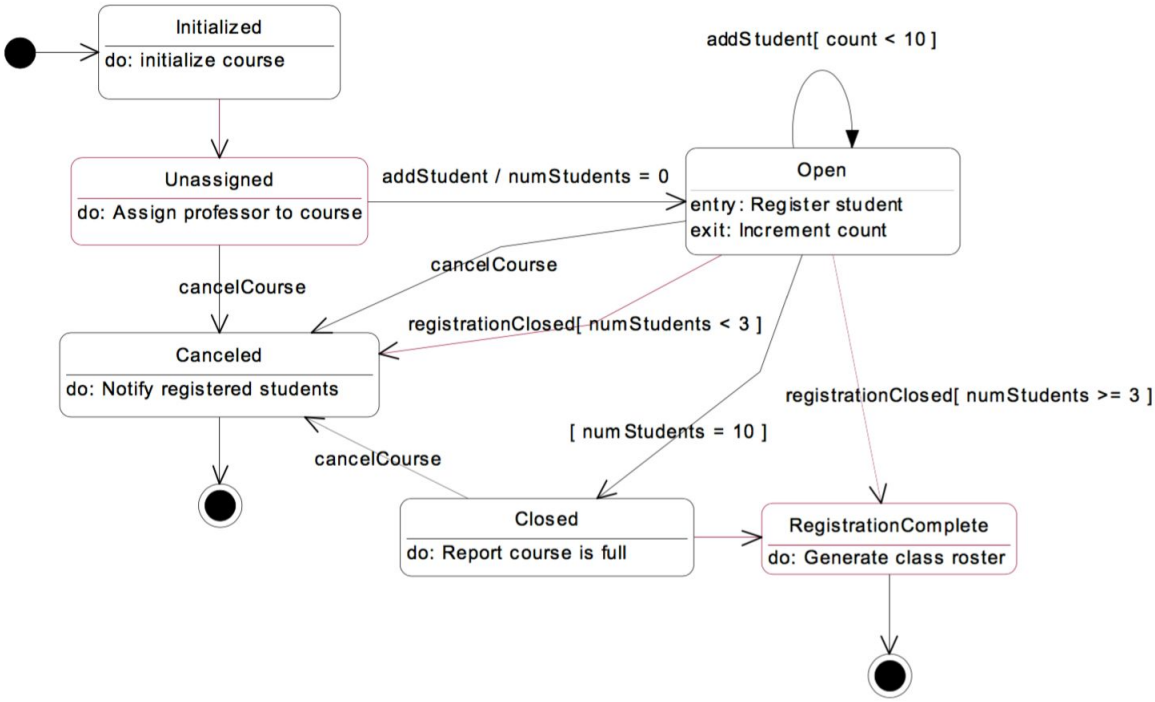
- 타이밍 다이어그램은 리얼타임시스템과 같이 시간에 대한 제약이 중요하게 다루어져야 할때 사용되는 다이어그램입니다. 타이밍 다이어그램은 가로축에 시간을 표시하고, 세로 축에 상태를 표시하여 시간이 지남에 따라 구성요소들의 상태의 변화를 보여줄 수 있습 니다. 상태다이어그램이 개개의 구성요소에 대한 상태를 표현하는 반면, 타이밍 다이어그 램은 구성요소들 사이의 상호작용과 상태변화를 동시에 보여줄 수 있습니다.



- 그림은 사용자가 이동전화를 이용하여 전화통화에 대한 타이밍 다이어그램입니다. 생명선인 ':사용자'와 ':이동전화기시스템'은 다이어그램의 가장 왼쪽에 작성되며, 생명선 은 서로 다른 구획으로 분리됩니다. 사용자는 대기중 상태에서 이동전화기의 플립열기버 튼을 누르면 이동전화기는 플립을 자동으로 엽니다. 이때 플립열기라는 메시지의 전달로 플립이 열리기 시작하는 시점까지의 시간은 최소 0.1 초, 최대 0.5 초 안에 이루어져야 합 니다. 또한 플립이 완전히 개폐되는 시간은 최소 0.5초, 최대 1초의 시간이 걸려야 합니 다. 플립이 다 열리고 나면 이동전화기 시스템은 플립열림 상태가 되고, 플립이 다 열렸 다는 메시지를 사용자에게 보냅니다. 사용자는 플립열림 메시지를 받고 전화번호를 입력 합니다. 사용자가 전화번호를 다 입력하고 난 후에는 연결버튼을 눌러, 이동전화기에 연결요청메시지를 보냅니다. 연결요청을 받은 이동전화기는 해당하는 번호로 연결을 시도 합니다. 이때 소요되는 시간은 최소 5 초, 최대 20 초여야 합니다. 연결이 완료되면 이동 전화기는 연결메시지를 사용자에게 보냅니다. 연결메시지를 받은 사용자는 통화를 시작 합니다. 통화가 완료된 후에는 사용자와 이동전화기는 모두 대기상태가 됩니다.

11. 상태 다이어그램 (State diagram)

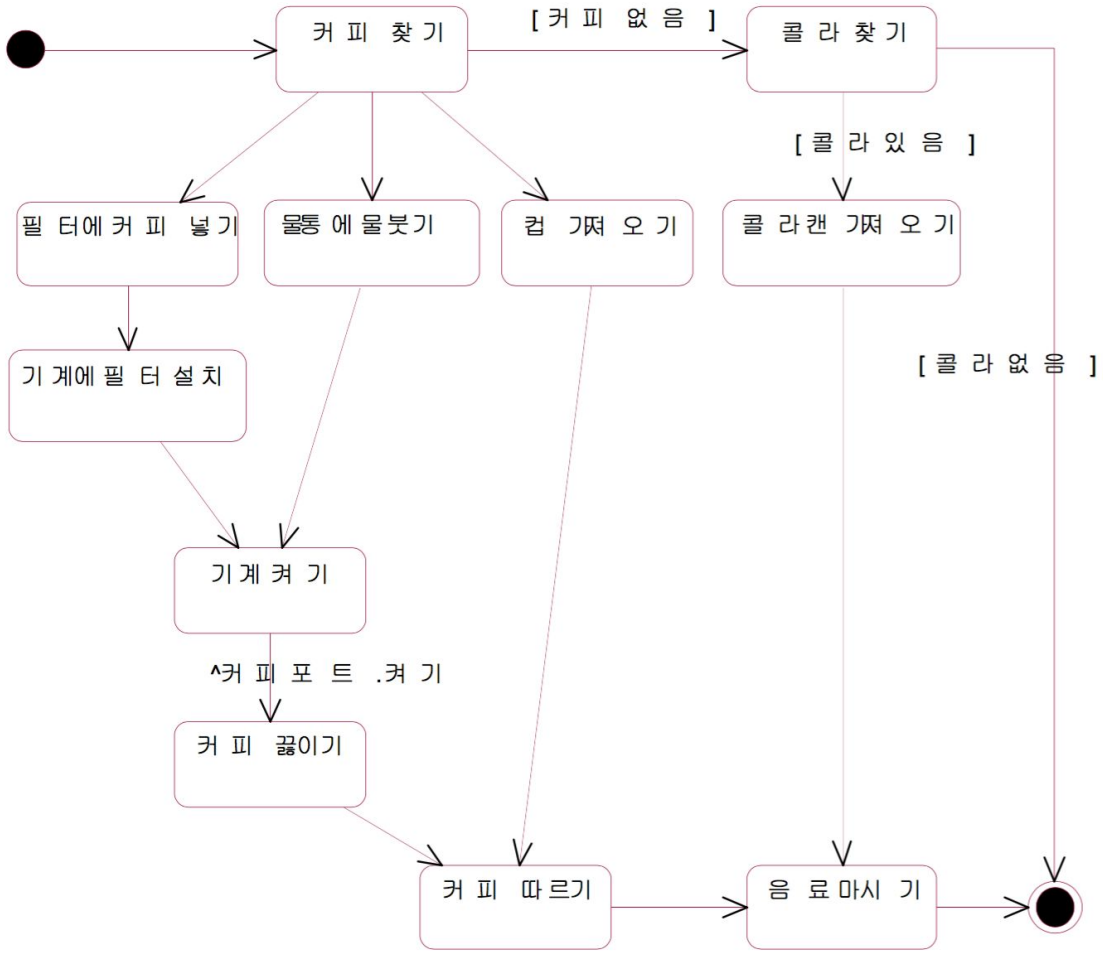
- 사용사례와 시나리오는 시스템의 행동양식 즉 객체들의 상호작용을 기술하는 기법 이다. 때로는 한 객체의 행동양식을 기술할 필요가 있다. 상태전이 다이어그램은 한 객체가 자신의 생명주기 안에서 취할 수 있는 상태들과 그 상태간 전이를 일으키는 이벤트들, 그리고 상태간 변화에서 발생하는 작용들을 표현한다.



- 상태 다이어그램은 시스템의 모든 클래스에 대해 그릴 필요는 없으며 의미있는 행동양식을 보여주는 주요 클래스들에 대해서 그린다. 가능한 상태나 이벤트 역시 필요에 따라 간단하거나 복잡한 수준으로 표현한다.

12. 활동 다이어그램 (Activity Diagram)

- 활동 다이어그램은 작업흐름과 연계되어 병행 처리가 많은 행동양식을 기술하기에 특히 유용한 여러 기법들을 조합한 것이다.



13. 컴포넌트 다이어그램 (Component Diagram)

- 컴포넌트 다이어그램은 시스템을 구성하는 실제 소프트웨어 컴포넌트간의 구성체계 를 기술하므로 아키텍처를 표현하기에 좋다. 컴포넌트란 용어는 시스템의 물리적 구조를 구성하는 소스 코드 단위로부터 부시스템 같은 실행 프로그램에 이르기까지 다양하게 지칭한다. 예를 들어 개별 클래스의 헤더와 구현파일로부터 그들을 조합 하여 만들어진 EXE, DLL 등까지 컴포넌트로 표시할 수 있다. 이러한 다양한 수준 의 컴포넌트를 사용하여 시스템의 아키텍처를 표현한다.

- 컴포넌트 다이어그램에는 각 컴포넌트를 그리고 컴포넌트간의 의존성 관계를 화살 표로 나타낸다.