Introduction to UML

2015/03/13

김용현 201011320
김태호 201111347
손준익 201111360

목 차

1. UML이란 ?

1.1 개요

통합 모델링 언어(UML, 영어: Unified Modeling Language)는 소프트웨어 공학에서 사용되는 표준화된 범용 모델링 언어이다. 이 표준은 UML을 고안한 객체 관리 그룹에서 관리 하고 있다.

UML은 소프트웨어 집약 시스템의 시각적 모델을 만들기 위한 도안 표기법을 포함한다. 통합 모델링 언어는 객체 지향 소프트웨어 집약 시스템을 개발할 때 산출물을 명세화, 시각화, 문서화할 때 사용한다. UML은 아래와 같은 사항을 포함하여 시스템의 구조적 청사진을 시각화 하는 표준안을 제공한다.

- 행위자 (UML)
- 비즈니스 프로세스
- (논리적) 부품 (UML)
- 행위 (UML)
- 프로그래밍 언어 구문
- 데이터베이스 스키마
- 재사용할 수 있는 소프트웨어 부품

UML은 데이터 모델링(개체-관계 다이어그램)과 비즈니스 모델링(업무 흐름), 객체 모델링, 부품 모델링의 최선의 기술을 조합한다. UML은 소프트웨어 개발 공정뿐만 아니라 다른 구현 기술의 모든 공정에서 사용될 수 있다. UML은 Booch 방법론의 객체 모델링 기법(OMT)와 객체 지향 소프트웨어 공학(OOSE)을 광범위하게 사용할 수 있는 단일한 공통 모델링 언어로 통합한다. UML의 목표는 동시적 분산 시스템을 모델링 하는 표준 언어다. UML은 산업의 실질적 표준으로서, 객체 관리 그룹(OMG)에 의해 개선되고 있다. 초기에 OMG가 엄격한 소프트웨어 모델링 언어를 만들기 위해 객체 지향 방법론적인 통지를 요청했고, 많은 산업 선구자가 UML 표준 제작을 돕기위해 진지하게 응답하였다.

UML 모델은 객체 관리 그룹이 지원하는 QVT와 같은 변환 언어 등을 이용해 다른 표현(예를 들면 자바)으로 자동적으로 변환된다. UML은 확장할 수 있으며 커스터마이제이션을 위한 메커니즘인 프로파일 (UML), 스테레오타입 (UML)을 제공한다. 프로파일을 이용한 확장의 의미는 UML 2.0에서 개선되었다.

1.2 역사

UML은 그래디 부치(Grady Booch), 제임스 럼버(James Rumbaugh), 이바 야콥슨(Ivar Jacobson)의 머리에서 태어났다. 최근 “쓰리 아미고(Three Amigos-3인방)”이라고 불리는 이 세 사람은 80년대 전반 부터 90년대 초반까지 객체지향 분석 설계 분야에서 각자의 영역에서 방법론을 연구해 왔었다. 그들이 발표한 방법론은 동일한 분야의 다른 경쟁자들보다 항상 탁월한 위치에 있었으며, 세 사람은 90년대 중반에 이르러 각자의 아이디어를 교환하기 시작하였고, 결국 각자의

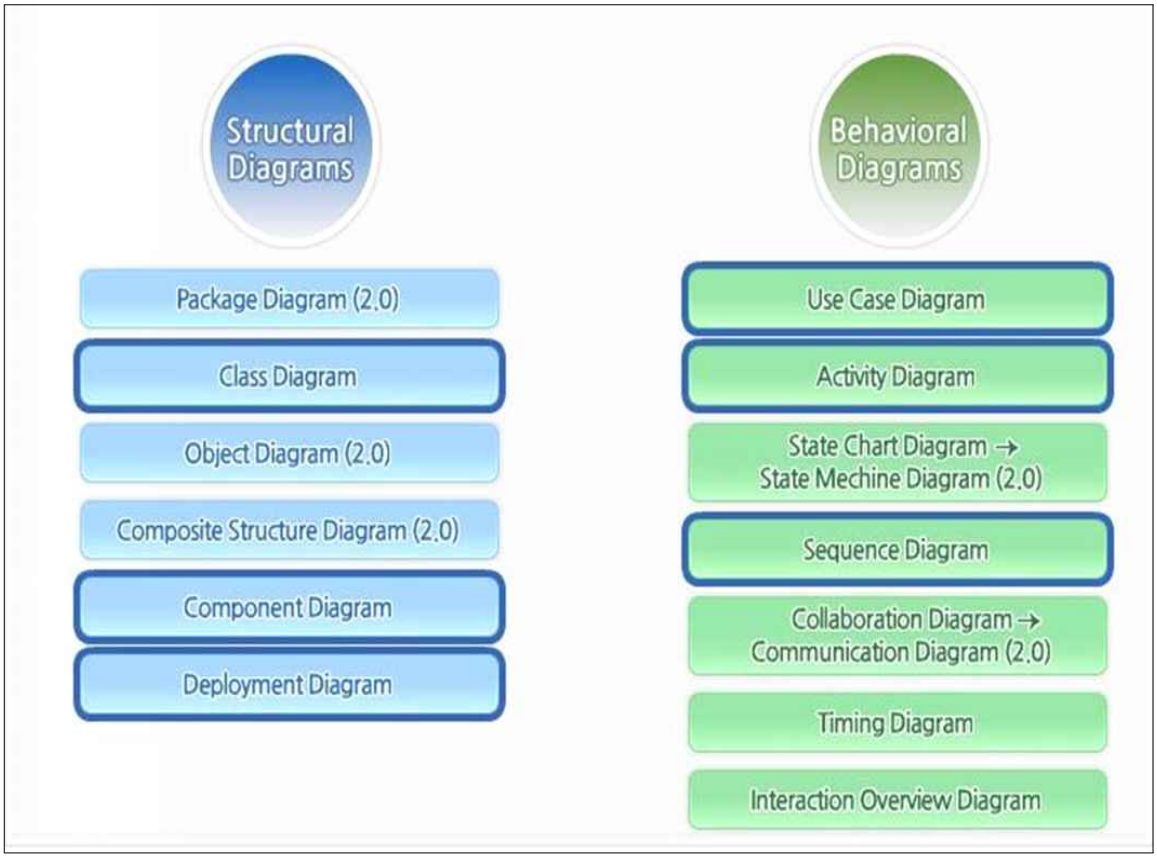
방법을 하나로 모아 합치기에 이른다.

1994년, 럼버는 부치가 세운 래셔널 소프트웨어(Rational Software Corporation)에 영입 되었고, 야콥슨은 그로부터 1년 후에 래셔널 사에 들어가게 된다. 나머지는 그들이 말하듯이, 역사(History)라고 말할 수 있다. UML의 초안(draft) 버전은 소프트웨어 업계를 뒤흔들기 시작했고, 그 결과로 돌아온 피드백은 바로 변경점에 반영되었다. “UML은 우리들의 전략에 딱 맞는다”라고 인식해 가는 회사가 늘어남에 따라 그 결과로 UML 컨소시엄도 발족하게 되었다. UML 컨소시엄의 멤버로는 디지털(DEC), 휴렛 팩커드(HP), 인텔리캅(Intellicorp), 마이크로소프트, 오라클, 텍사스 인스트루먼트(Texas Instruments), 래셔널 소프트웨어 등이 있었다. 1997년 UML 컨소시엄은 UML 버전 1.0을 만들어 내었고, 객체 관리 그룹(OMG:Object Management Group)이 표준 모델링 언어의 제안서를 내라는 요구에 맞추어 이것을 제출 하였다.

UML 컨소시엄은 계속 발전하였으며, OMG에 다시 상정된 UML 1.1d은 1997년 말에 표준 모델링 언어로 채택되었다. OMG는 UML의 관리 기법을 받아들여 1998년에 새로운 수정안을 발표하였다. UML은 소프트 웨어의 업계 명실 상부한 표준이 되었으며, 계속 수정 보안되고 있다. 버전 1.3과 1.4 그리고 1.5가 나와 있고, 최근에는 버전 2.0이 OMG에 의해 승인된 상태이다. 이전 버전들, 즉 버전 1.X는 현존하는 대부분의 모델 및 UML 모델링 책의 기본이 되어 왔다.

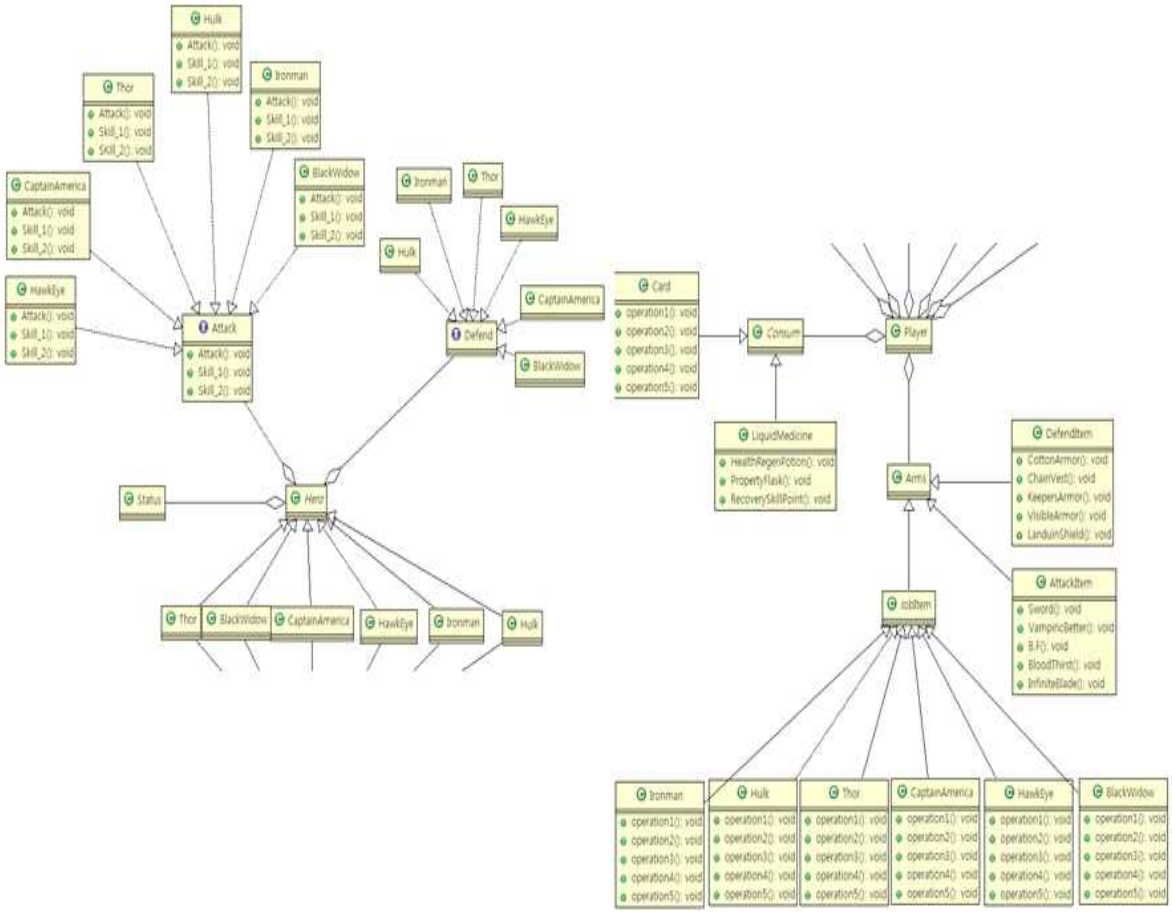
2. UML Diagram

2.1 UML Diagram



통합 모델링 언어(UML)를 사용하여 시스템 상호 작용, 업무 흐름, 객체 간의 메시지 전달, 시스템의 구조, 컴포넌트 관계 등을 그린 도면. 이에는 요구 분석 과정에서 시스템과 외부와의 상호 작용을 묘사하는 Use Case 다이어그램, 업무의 흐름을 모델링하거나 객체의 생명 주기를 표현하는 Activity 다이어그램, 객체 간의 메시지 전달을 시간적 흐름에서 분석하는 Sequence 다이어그램, 객체와 객체가 주고받는 메시지 중심의 작성 동적 다이어그램인 Collaboration 다이어그램, 시스템의 구조적인 모습을 그리는 Class 다이어그램, 소프트웨어 구조가 그리는 Component 다이어그램, 기업 환경의 구성과 컴포넌트들 간의 관계를 그린 Deployment 다이어그램 등이 있다.

2.2 Class Diagram



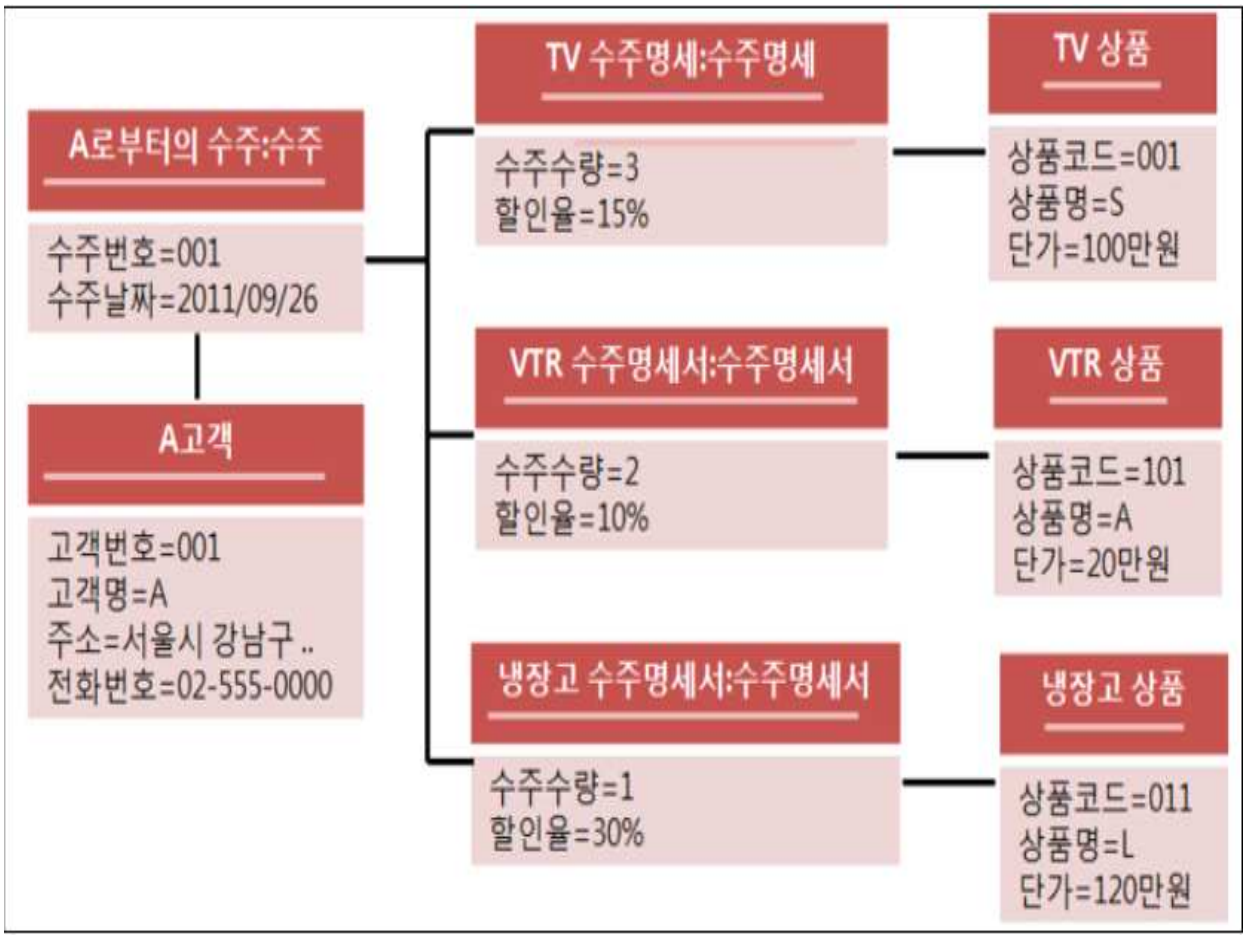
대부분의 사물은 자기만의 속성과 일정한 행동 수단을 지니고 있다. 이러한 행동을 오퍼레이션의 집합으로 생각할 수 있다. UML에서는 두 단어 이상으로 이루어진 클래스의 이름은 단어 사이의 공백을 없애고 각 단어의 처음 문자를 모두 대문자로 한다. 속성과 행동의 이름 또한 마찬가지이지만, 가장 앞 단어의 처음 문자는 소문자로 한다.

클래스 다이어그램은 시스템의 정적인 정보구조를 나타내는 정보 모델로서, 시스템에 필요한 클래스들과 이들 사이의 관계를 나타내는데 사용된다.

각 클래스는 해당 객체의 특성을 나타내는 여러 가지 속성들과 오퍼레이션으로 구성된다.

클래스, 클래스의 속성, 클래스들 간의 관계를 찾아내기 위해서는 선행단계에서 작성된 문서들을 활용할 수 있는데, 문제기술서(요구사항분석)나 UseCase 시나리오가 클래스 도출에 사용될 수 있다.

2.3 Object Diagram



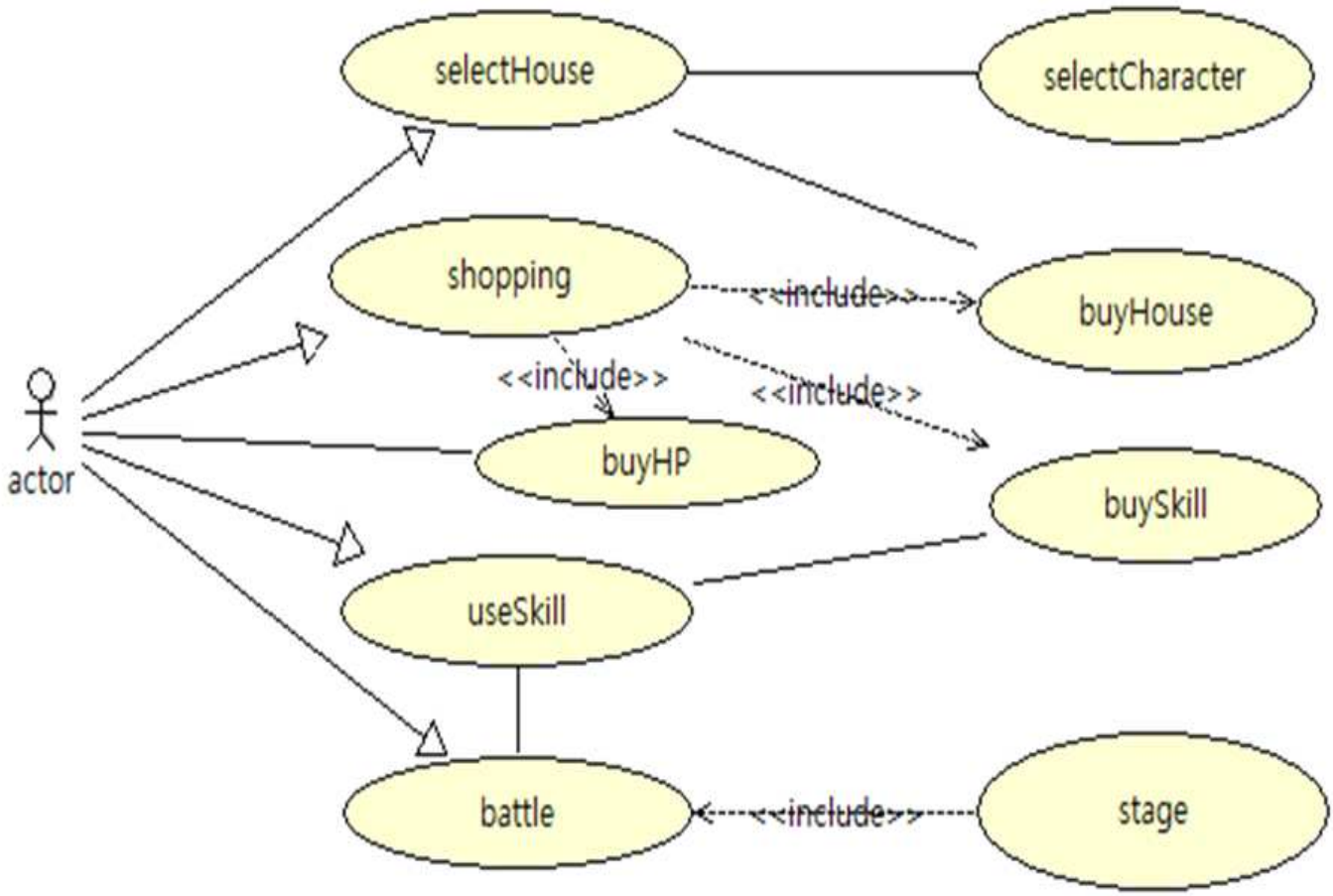
객체 다이어그램이란 객체사이의 정적인 관계를 파악하기 위해 작성한다. 객체다이어그램은 클래스 다이어그램을 구체화 시킨 것으로, 시스템의 어떤 한 시점에서의 상태를 나타낸다.

객체는 현실세계의 사물을 표현한다. Composit 객체는 객체가 모여 하나의 사물을 구성하고 있는 경우, 그 전체 객체를 가리킨다. 링크는 객체사이에 관계가 존재함을 표현한다.

객체란, 클래스의 인스턴스이다. 값이 매겨진 속성과 행동을 가지고 있는 개별적인 개체를 일컫는다.

다이어그램에 나타내는 그림 자체는 클래스와 같이 사각형이지만, 이름 밑에 밑줄이 그어져있다. 인스턴스의 이름은 콜론의 왼편에 쓰며, 클래스의 이름은 콜론의 오른편에 쓴다. 인스턴스의 이름은 소문자로 시작하고, 익명의 객체도 가능하다. 객체가 속해있는 클래스를 보여주는 것에만 중점을 둠으로써, 특정 이름을 지정해 주지 않은 것이다.

2.4 UseCase Diagram

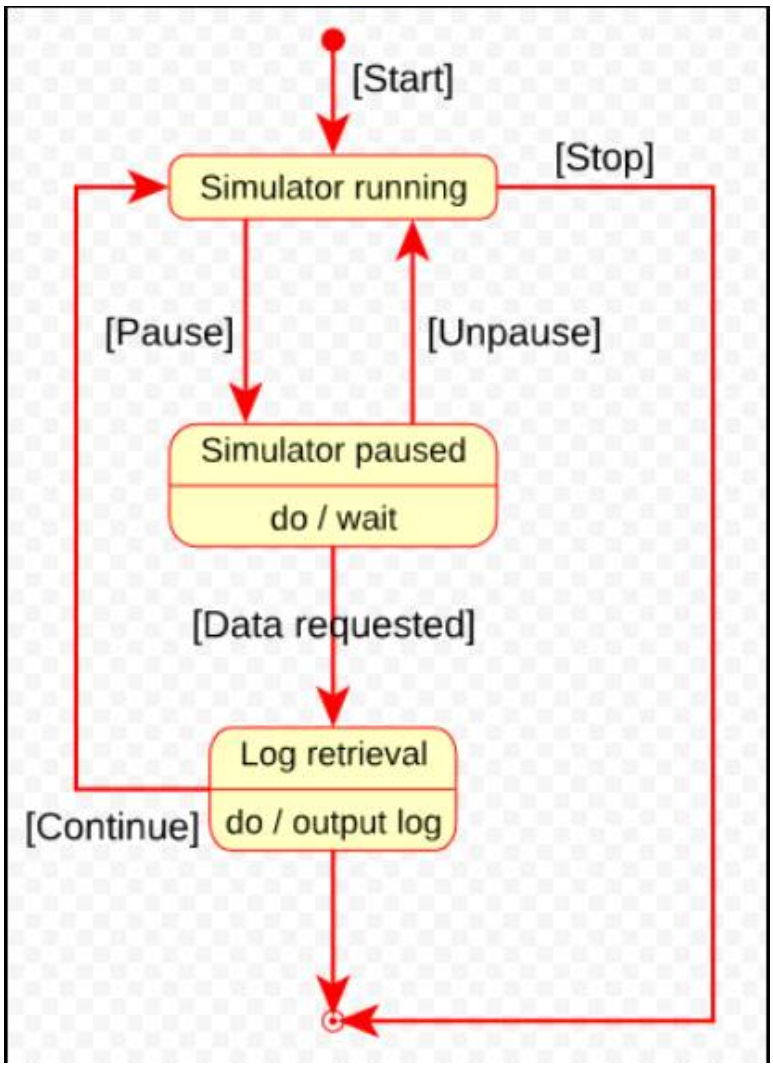


Use Case 다이어그램이란 사용자의 시각에서 소프트웨어 시스템의 범위와 기능을 설명하고 정의한 모델이다. 소프트웨어 시스템의 기능적 요구사항에 대한 기초를 나타낸다.

Use Case 다이어그램의 작성시기로는 소프트웨어 프로젝트의 개발범위를 정의하는 단계, 소프트웨어에 대한 요구사항을 정의하는 단계, 소프트웨어의 세부기능을 분석하는 단계, 소프트웨어가 아닌 업무영역을 이해하고 분석하는 단계에서 수행한다.

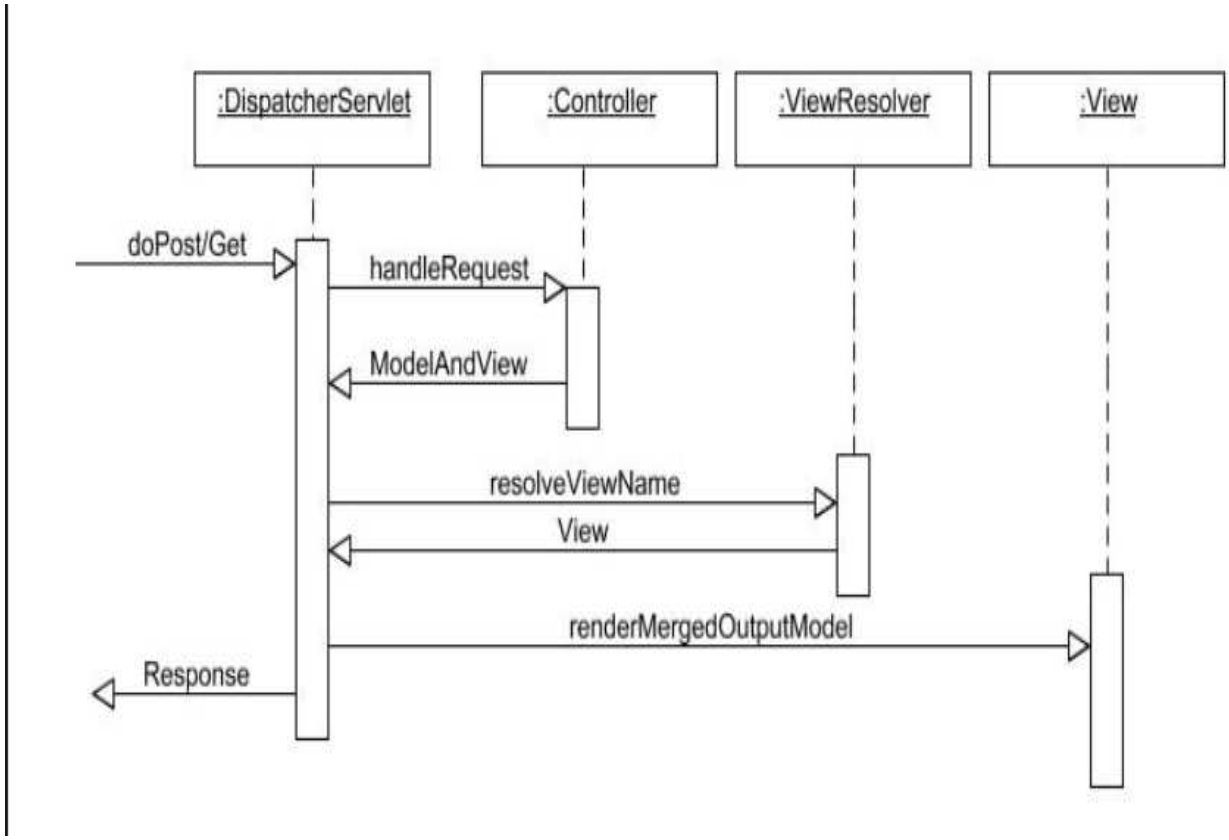
막대인간의 그림을 행위자(Actor)라 한다. 타원은 유스 케이스를 나타낸다. 행위자(유스 케이스와 대화를 시작하는 개체)는 사람 혹은 다른 시스템이 될 수 있다. 또한 유스 케이스가 시스템을 의미하는 사각형 내에 있고, 행위자는 사각형 바깥에 있다.

2.5 State Diagram



하나의 객체가 생성되어 소멸될 때까지 가질 수 있는 가능한 모든 상태를 분석하고 표현하는 Diagram이다.
상태, 전이, 사건, 활동으로 구성되며 한 객체가 가지는 상태와 사건에 따라 순차적으로 발생하는 행동에 중점을 두고 작성한다.
하나의 상태전이 다이어그램에서 객체의 초기상태를 나타내는 시작상태는 오직 하나만 존재하며, 객체의 마지막 상태인 종료상태는 여러 개 존재할 수 있다.
※ 상태, 전이, 이벤트 설명
상태 : 객체가 존재할 수 있는 조건이나 상황, 상태들은 의미적인 차이가 잘 드러나도록 추상화해야한다.
전이 : 다음상태를 결정한다.
이벤트 : 전이를 트리거하는 외부 또는 내부의 사건이다.

2.6 Sequence Diagram

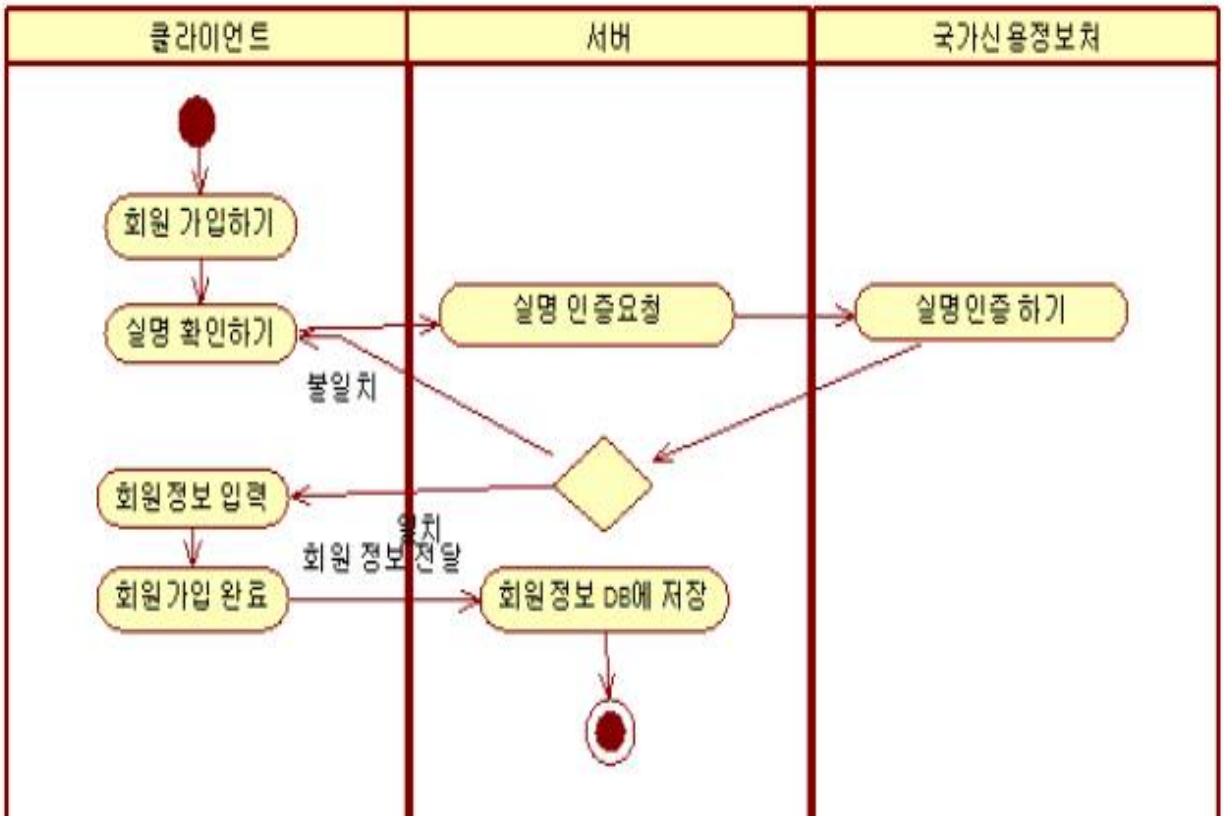


Sequence 다이어그램은 특정 사용 케이스에 대한 상세한 흐름이나 심지어는 특정 사용 케이스의 일부분 까지도 보여준다. 대부분이 설명을 포함하고 있다. Sequence에서 다른 객체들 간의 호출관계를 보여주고, 다른 객체들로부터 호출까지 상세히 볼 수 있다.

Sequence 다이어그램은 2차원으로 그려진다. 수직차원은 발생시간 순서로 메시지/호출 Sequence를 보여주고 있다. 수평차원은 메시지가 전송되는 객체 인스턴스를 나타내고 있다.

Sequence 다이어그램의 상단에 각 클래스의 객체를 박스 안에 놓아 각각을 구분한다. 박스 안에 클래스 객체 이름과 클래스 이름을 “ : ”으로 분리시킨다. 클래스 객체가 메시지를 또 다른 클래스 객체로 보내면 메시지를 받는 곳을 가리키는 화살표를 긋는다.

2.7 Activity Diagram



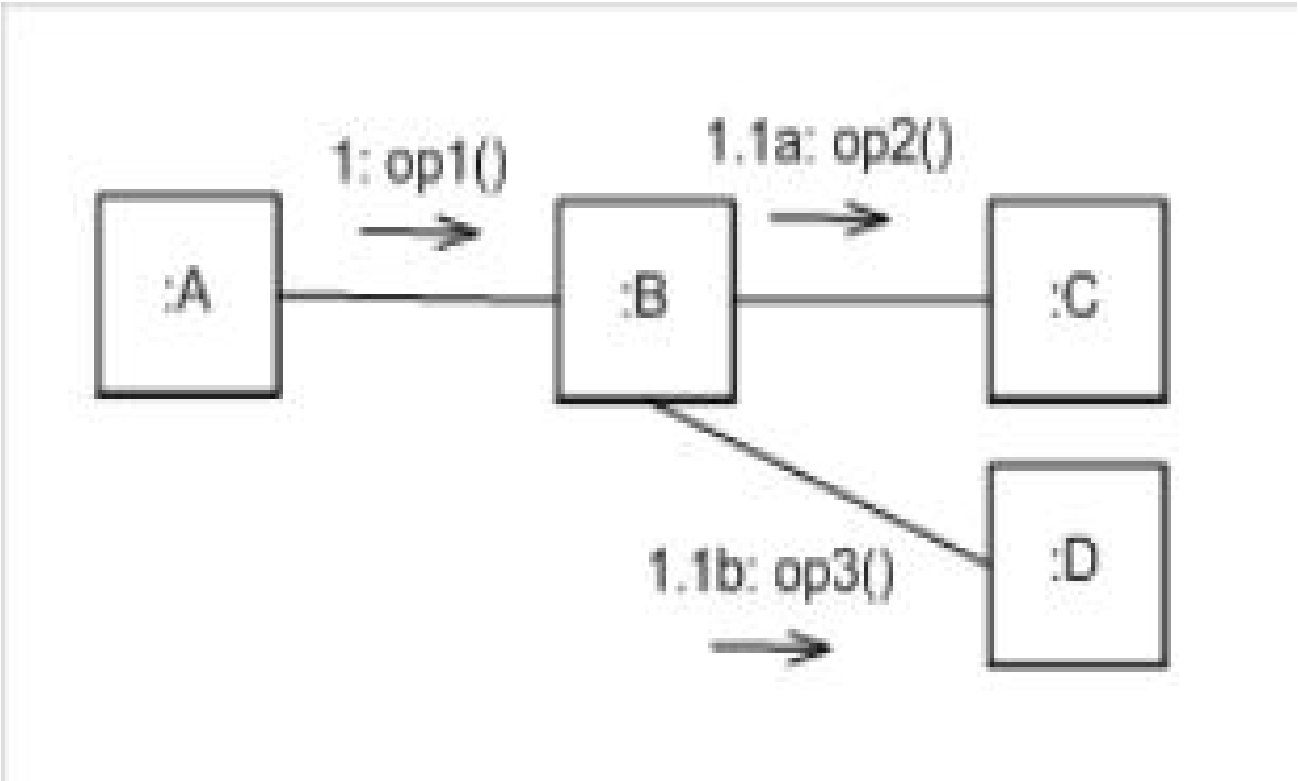
Activity 다이어그램은 활동을 처리하는 동안 두 개 이상의 클래스 객체들 간 제어흐름을 보여준다. Activity 다이어그램은 비즈니스 단위 레벨에서 상위 레벨의 비즈니스 프로세스를 모델링하거나 저 수준 내부 클래스 액션을 모델링 하는데 사용된다.

Activity 다이어그램은 기업이 현재 어떻게 비즈니스를 수행하는지, 또는 어떤 것이 비즈니스에 어떤 작용을 하는지 등의 고차원 프로세스를 모델링 할 때 적합하다.

Activity 다이어그램은 시퀀스다이어그램 보다 덜 기술적이기 때문에 비즈니스 마인드를 가진 사람들이 빠르게 이해 할 수 있는 장점이 있다.

유스 케이스 내부 혹은 객체의 동작중에 발생하는 활동은 대개 시퀀스내에서 발견할 수 있다.

2.8 Communication Diagram



하나의 시스템을 구성하는 요소들은 다른 요소들과 손발을 맞추면서 시스템 전체의 목적을 이루어 나가는 것이 보통이기 때문에, 모델링 언어는 이것을 표현할 수 있어야 한다. 앞서 말한 Sequence 다이어그램이 그것이고, Communication 다이어그램 또한 이러한 목적을 위하여 디자인된 것이다.

Communication 다이어그램은 상호작용 다이어그램의 일종이다. Communication 다이어그램은 여러 객체, 컴포넌트들 사이의 상호작용을 표현하기 위해 사용된다.

Communication 다이어그램은 시스템 개발의 여러 단계에서 작성될 수 있다.

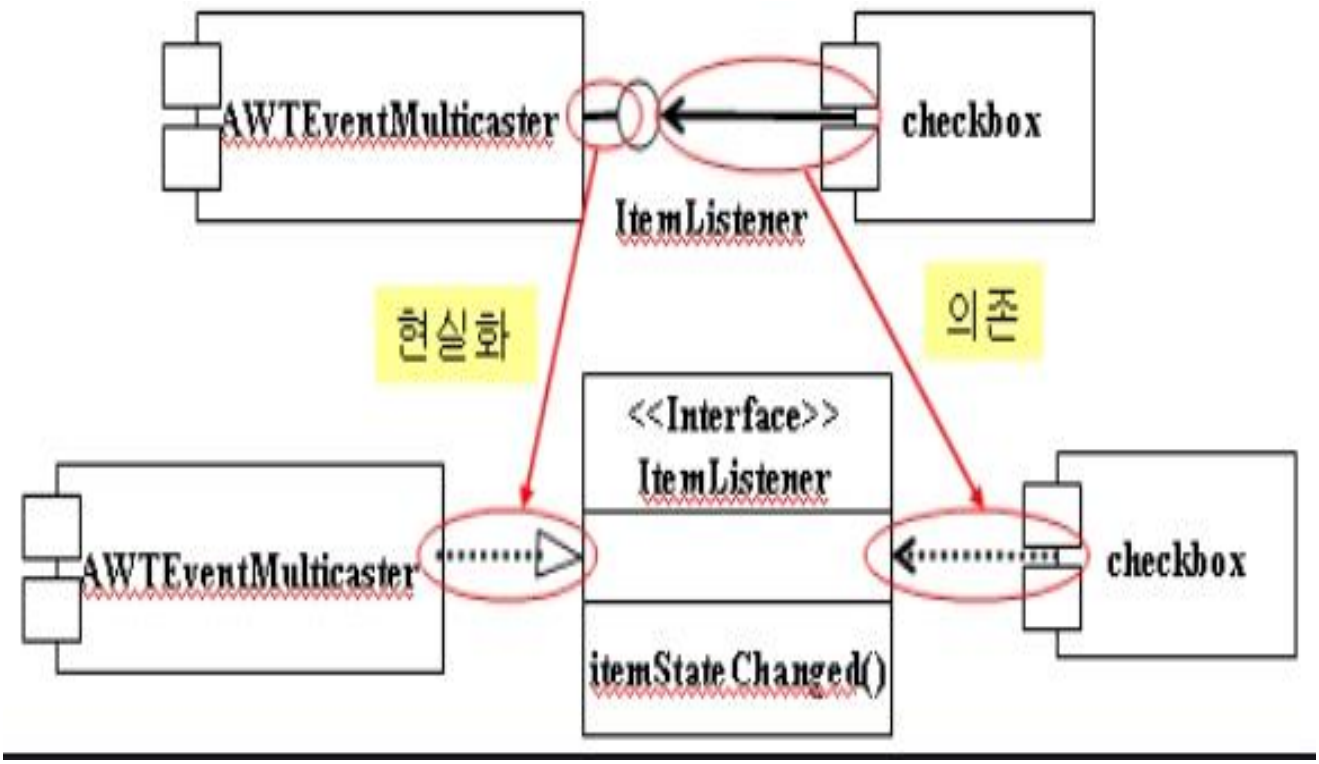
시스템 개발 초기에, 클래스 다이어그램을 개발하는 과정의 일부로서 작성한다. 이 경우 각 사용사례에 참여하는 객체는 Communication 다이어그램에 모델링 된다. 객체는 클래스에 할당된다. Communication 다이어그램의 클래스들은 합쳐져서 첫 번째 클래스 다이어그램이 된다.

Communication 다이어그램은 시스템의 클래스를 파악한 후, 사용사례 실현을 위해 객체들 간의 발생할 상호작용을 나타내기 위해 작성한다. 복잡한 형태를 가진 오퍼레이션의 실현을 나타내기위해 작성한다.

Sequence 다이어그램과 Communication 다이어그램은 객체 사이의 상호관계를 나타내는 다이어그램이기 때문에 UML에서는 이 둘을 Interaction 다이어그램이라고 부른다.

Communication 다이어그램이라는 명칭은 UML 2.0부터 등장하기 시작하였고, 이전 1.X에서는 Collaboration 다이어그램이라고 불렀다.

2.9 Component Diagram

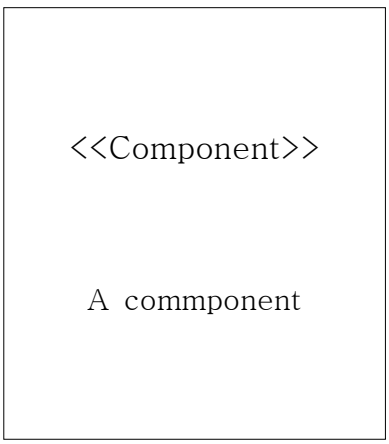


현대의 소프트웨어 개발 추세는 컴포넌트 중심으로 되어가고 있다. 팀 단위 프로젝트라면 특히 중요한 것이 컴포넌트이다.

Component 다이어그램은 시스템을 물리적으로 볼 수 있도록 한다. Component 다이어그램의 목적은 소프트웨어가 시스템의 다른 소프트웨어 Component들에 대해 소프트웨어가 가지고 있는 종속관계를 보여준다.

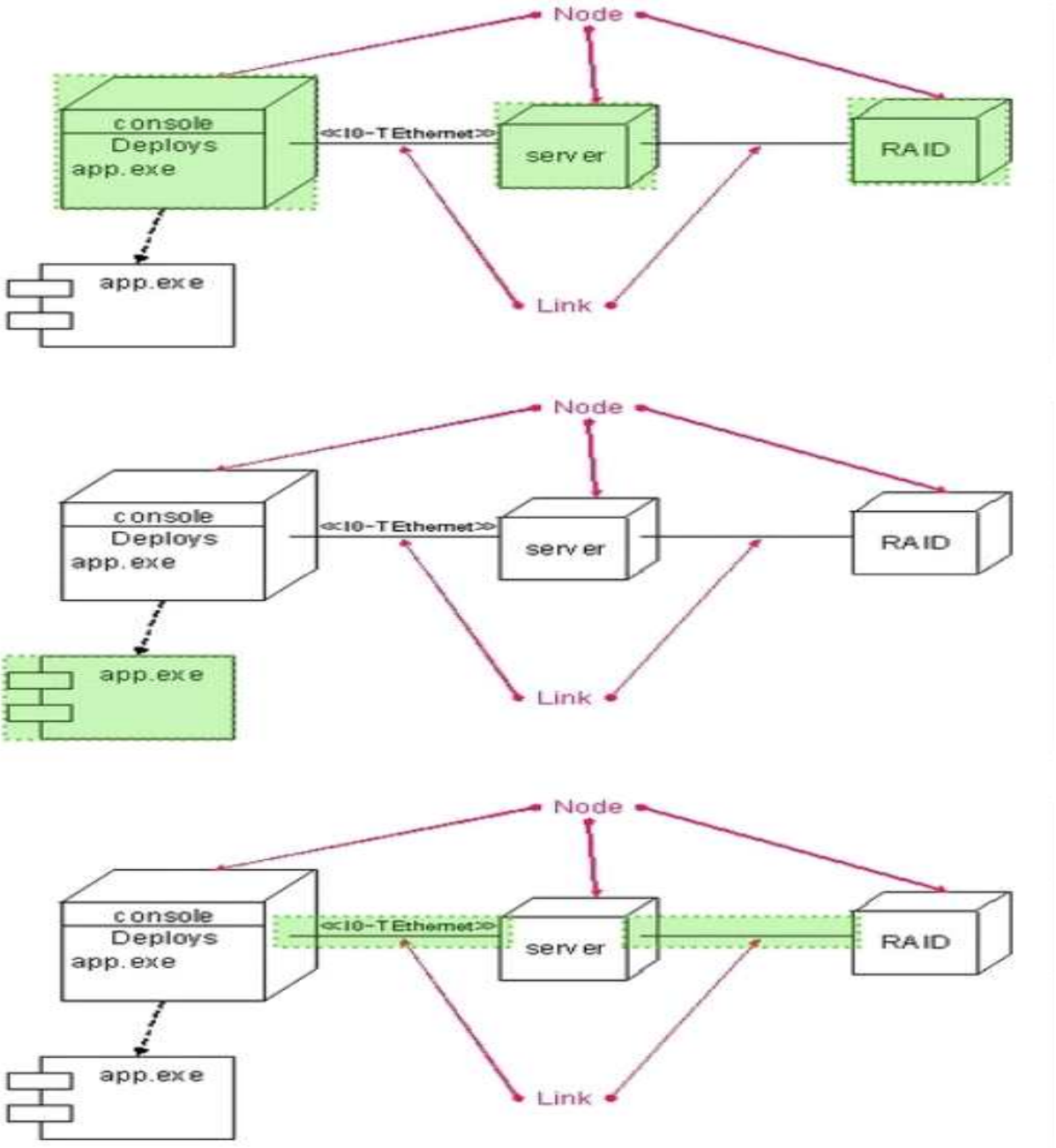
Component 다이어그램은 고급레벨에서 볼 수 있거나, Component 패키지 레벨에서 확인 할 수 있다.

위 그림에 나타난 기호는 어색하다는 반응이 많아. UML 2.0 부터는 스테레오 타입으로 표기법을 바꿨다.



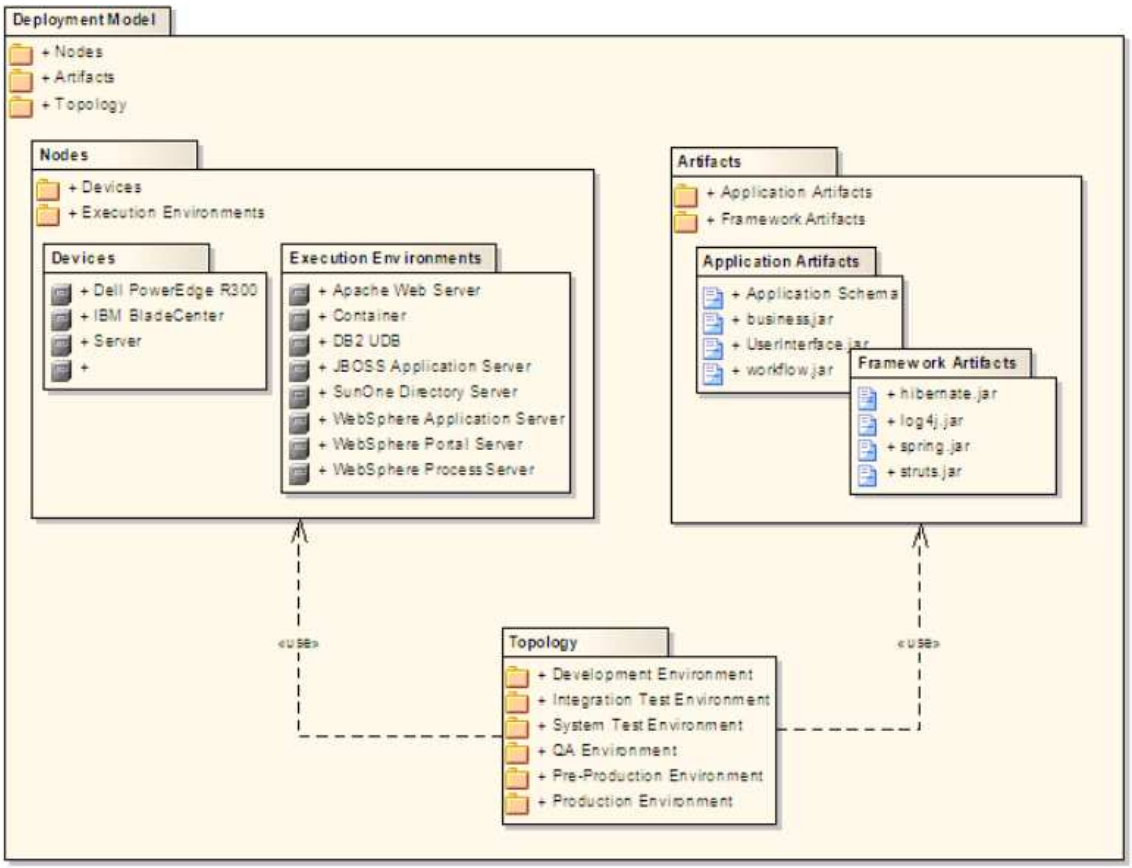
[UML 2.0 Component Diagram 표기법]

2.10 Deployment Diagram



Deployment 다이어그램은 시스템의 물리적 레이아웃을 표현하기 위해 작성한다. Deployment 다이어그램의 표기 사항은 다음과 같다. 어떤 소프트웨어 부분이 어떤 하드웨어 상에서 실행되는가를 표기한다. 소프트웨어의 배치 및 실행될 하드웨어 자원 등을 표현한다. 소프트웨어 컴포넌트가 어떤 하드웨어 자원에 탑재되어 실행되는지를 표현한다. 하드웨어 자원의 물리적인 구성을 표현한다.
Deployment 다이어그램은 시스템의 설계 단계 마지막에 작성한다.
구성요소로는 노드, 컴포넌트, 의존관계, 연관관계가 있다.

2.11 Package Diagram



패키지는 요소들을 그룹으로 조직하기 위한 범용 메커니즘으로, 모델의 요소들을 조직하고 이해할 수 있도록 해준다. 패키지에 담기는 것은 클래스뿐만 아니라, 유스케이스, Activity 다이어그램 등과 같은 다이어그램, 패키지도 담을 수 있다.

패키지를 구성할 때에는 여러 사람이 동의 할 수 있는 형태로 구성되어야 하며, 패키지의 구성과 이름체계는 개발자들이 쉽게 이해하고 사용할 수 있어야한다. 보통 클래스는 소속된 패키지와 함께 사용되거나 참조된다.

패키지에 스테레오 타입을 적용하여 표기한다.

<<facade>> : 다른 패키지에 뷰를 제공해주는 패키지

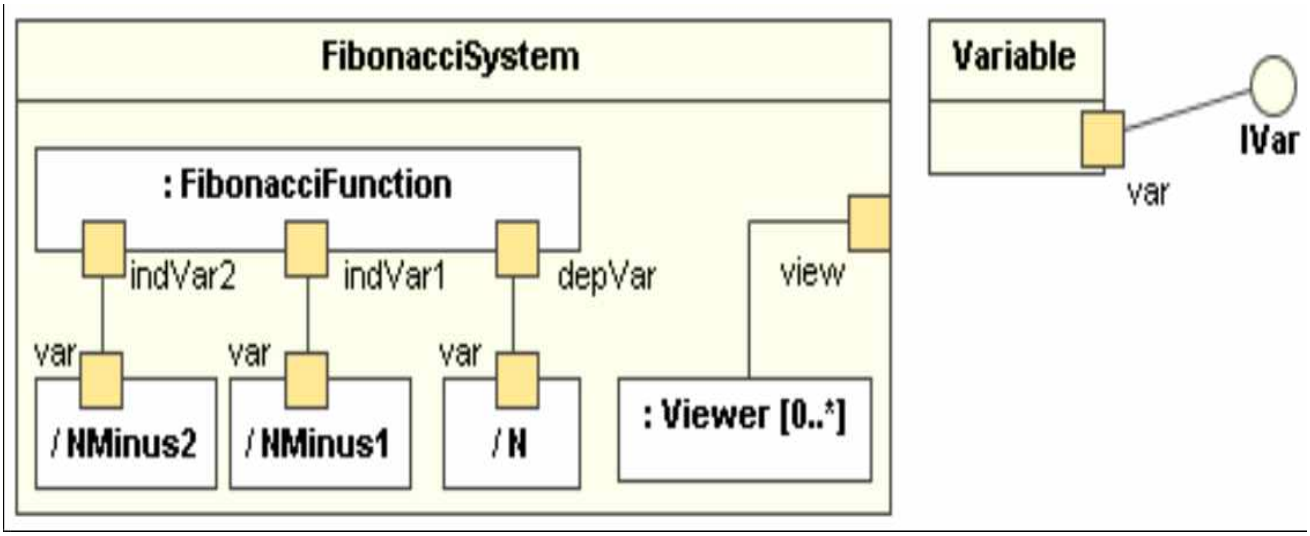
<<framework>> : 주로 패턴으로 구성된 패키지

<<stub>> : 다른 패키지의 공용 내용물에 대한 대리자 역할을 수행하는 패키지

<<subsystem>> : 전체 시스템의 독립된 일부분을 나타내는 패키지

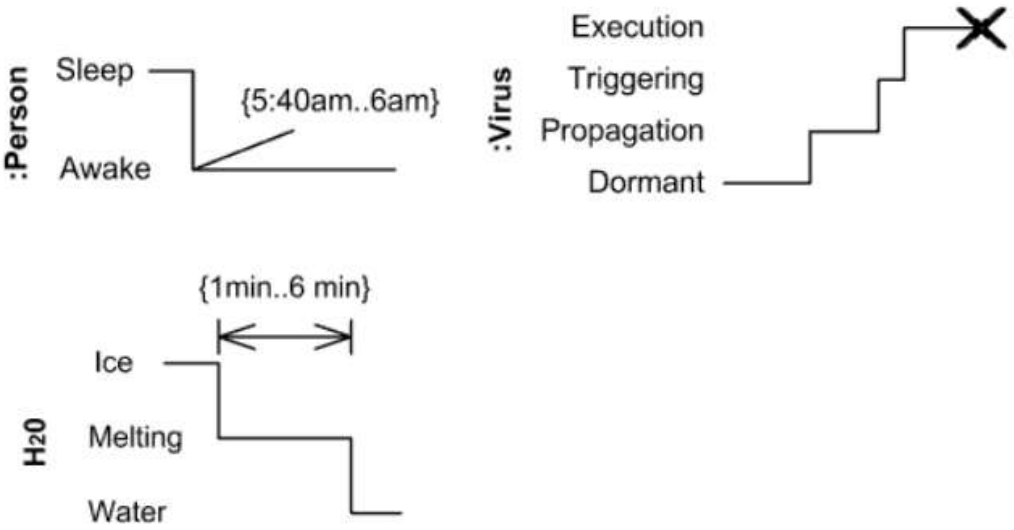
<<system>> : 전체 시스템을 나타내는 패키지

2.12 Composite structure diagram



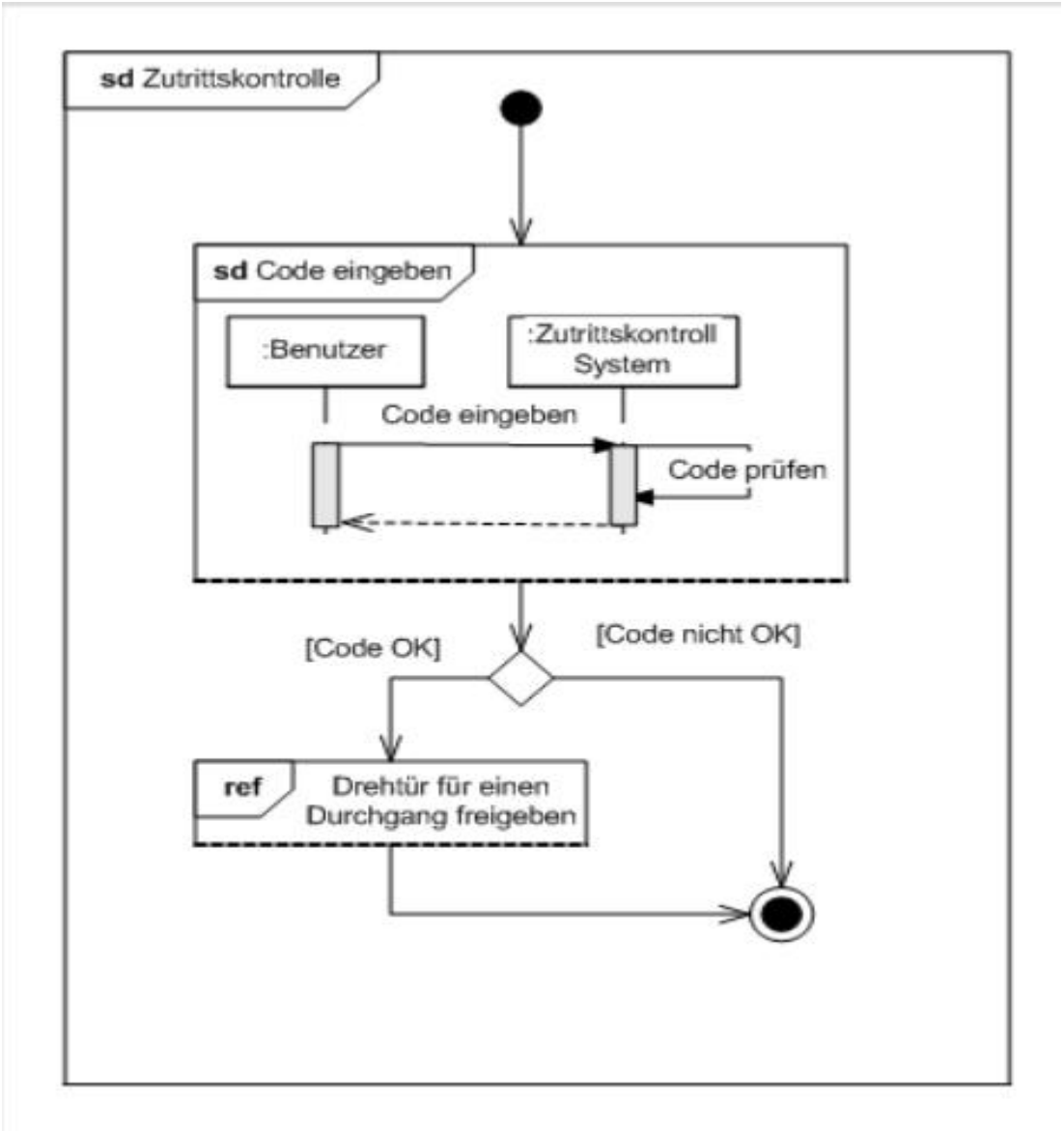
복합체 구조 다이어그램(Composite structure diagram에는 각 컴포넌트 클래스를 전체 클래스 안에 위치시킴으로써 넓이의 개념을 포함한다. 클래스로만 국한되던 시야를 전체 구조로 넓히는 것이다.

2.13 Timing diagram



Sequence 다이어그램에서 표현하지 못했던 걸리는 시간에 대한 내용을 다루는 표기방법이다. 한 상태에서 객체가 얼마나 오랜 시간을 지체하는지를 명시하는데 Timing 다이어그램을 사용한다.

2.14 interaction overview diagram

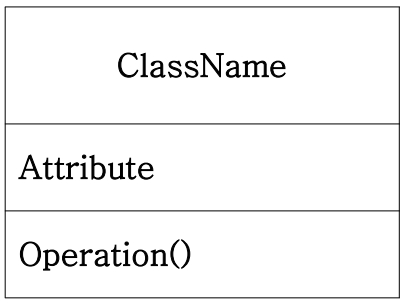


Activity 다이어그램에서 하나의 단계가 하나의 활동이었다. Interaction overview 다이어그램은 각 활동마다 객체사이에 시간의 흐름을 갖는 메시지를 표현한다.

3. UML 표기법

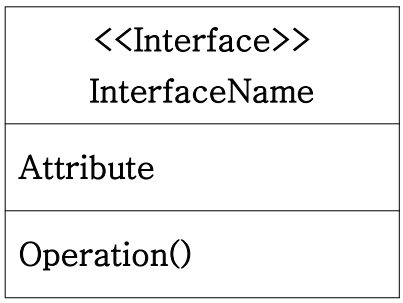
3.1 Object

3.1.1 Class

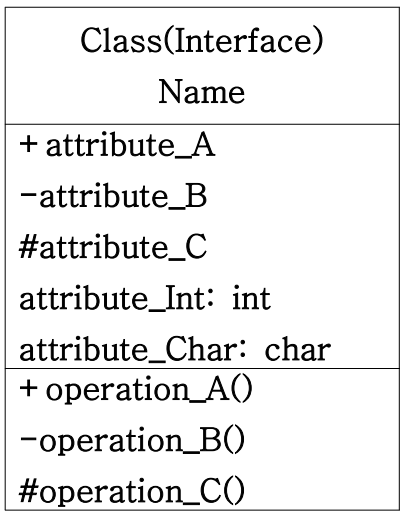


ClassName : 클래스의 이름
Attribute : 멤버 변수
Operation : 멤버 함수

3.1.2 Interface

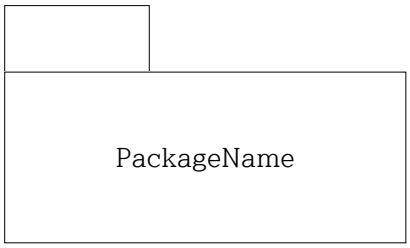


Interface는 Class와 달리 상단에 <<Interface>>라는 문구를 가지고 있다.



Class와 Interface는 모두 Attribute와 Operation을 contents로 가지고 있다. 이 두 요소의 이름 앞에는 +, -, #이 붙을 수 있는데, 이는 가시성 수정자를 의미한다. +는 public, -는 private, #은 protected 이다. 또한 Attribute에는 colon(:)을 사용하여 Attribute의 type를 표기하는 것이 가능하다.

3.1.3 Package

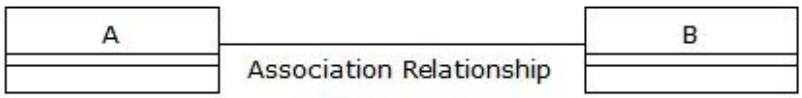


Class(Interface) 등의 집합으로 이루어진 표현이며, 기본적으로 directory구조를 기반으로 한다. PackageName은 해당 Package의 Name을 의미하는데, 이 Name의 표기는 directory구조 가지는 방향으로 가는 것이 바람직하다.

※Package는 특정한 Component를 의미하기도 한다.

3.2 Relationships

3.2.1 Association

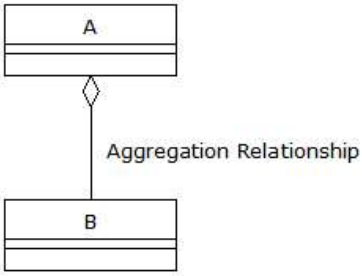


객체간의 연관관계를 의미한다. 객체 A와 B사이에 특정한 관계가 성립하는 경우에 사용한다.

Sample)

SalseMan은 SalesTeam의 구성원이기에 연관이 있다고 볼 수 있다.
따라서 SalseMan과 SalesTeam은 'Work' 라는 Association Relationship을 가지고 있으므로, 위 그림과 같이 표현이 가능하다.

3.2.2 Aggregation

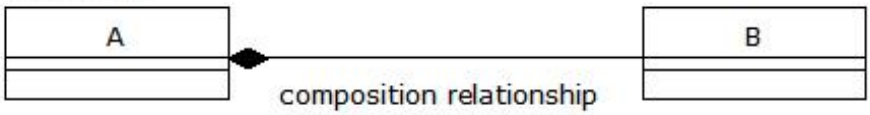


객체간의 포함관계를 의미한다. 객체 A는 객체 B를 포함하고 있음을 의미한다.
하지만, 이 관계에서 A와 B는 같은 생명주기를 갖지 않는다.

Sample)

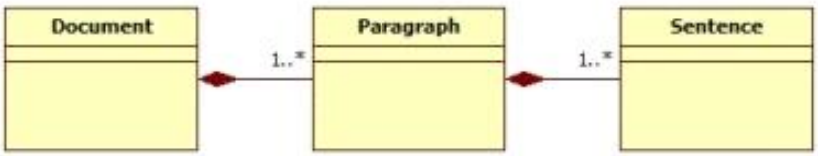
하나의 부서는 회사의 한 부분이다. 따라서 부서는 회사에 포함된다고 볼 수 있으므로, Aggregation관계로 표현 가능하다.

3.2.3 Composition



객체간의 포함관계를 의미한다. 객체 A는 객체 B를 포함하고 있음을 의미한다. 이 관계에서는 A와 B가 같은 생명주기를 갖는다.

Sample)

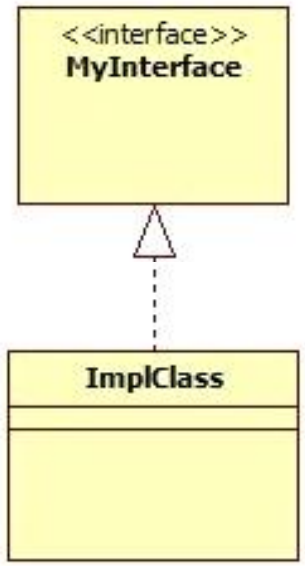


Sentence가 모여 Paragraph가 되고, Paragraph가 모여 Document가 된다. 각각의 요소는 그 상위단계에 포함된다. Aggregation과 다른 점은 그 관계가 생명주기를 같이 하는지에 대한 여부이다. Composition은 Aggregation의 한 형태이며, 각 요소 간에 더 강한 소속감을 가질 경우에 사용한다.

3.2.4 Implementation

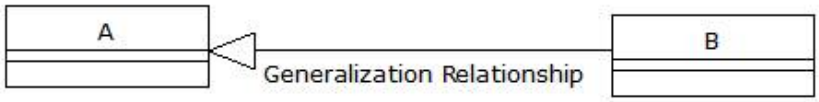
A가 B를 구체화시킴을 의미한다. B는 불완전한 Operation을 가지고 있어야 하며, A는 해당 Operation을 구체화시켜 구현한다.

Sample)



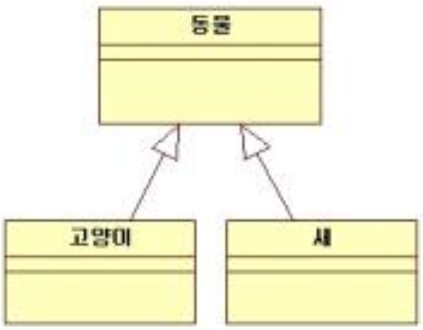
MyInterface를 ImplClass에서 구현하였음을 의미한다. Interface를 여러 개의 클래스에서 구현하는 것 또한 표현이 가능하다.

3.2.5 Generalization



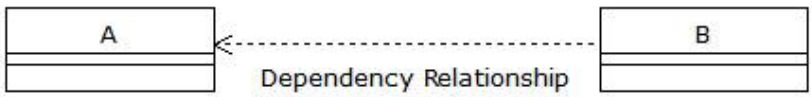
상속관계를 의미한다. A는 Parent, B는 Child이다.

Sample)



고양이와 새는 동물이다. OOP의 상속관계를 나타내는 표현이므로 동물은 Parent, 고양이와 새는 Child이다.

3.2.6 Dependency



의존적인 관계를 말한다. B는 A에 존재하는 어떠한 요소를 참조한다. 참조되는 대상이 return type이거나 argument type인 경우.

4. UML Modeling Tools

4.1 UML Modeling Tools

현존하는 대부분의 UML Modeling(Design 혹은 CASE) Tool들의 목록은 List of UML tools 페이지에서 확인할 수 있다. 제품의 가격별로 목록이 정리되어 있는 페이지도 있다.

4.2 상용 제품

몇가지 대표적인 상용 도구들을 뽑아보면 다음 목록 정도로 정리할 수 있다.

- IBM RSA (Rational Software Modeler/Architect)
- Together Architect/Designer/Developer (Borland)
- MagicDraw (No Magic)
- Visual Paradigm for UML (Visual Paradigm)
- EA (Enterprise Architect)
- Power Designer
- Visio

역사적으로 볼 때 가장 대표적인 도구이며 UML 툴의 대명사라고 할 수 있는 제품이 Rational 사의 ROSE인데, IBM으로 넘어가면서 기존 Rose와 함께 RSA(Rational Software Architect)라는 이름으로 발전을 거듭하고 있다.

Borland의 Together 역시 대표적 UML 도구의 하나이며, MagicDraw, Visual Paradigm도 많이 알려지고 사용되는 도구들이다.

Power Designer는 Data Modeling, BPM(Business Process Modeling), UML등을 지원하는데, CA ERwin Data Modeler와 함께 UML 보다는 Data Modeling Software Tool로서 더울 잘 알려져 있다.

그리고 Visio의 경우 UML 도구라기보다는 Business Graphics Software로 분류하는 것이 더 적합했었지만, Visio 2000이상 버전부터 UML 1.2를 지원하고 있으며, Visio 2007의 경우 UML 2.0을 지원하며 지속적으로 UML 관련 기능을 강화하고 있어 현재 여타 UML 툴들과 비교하여 손색이 없는 듯 하다.

마지막으로 EA(Enterprise Architect)는 고가인 다른 UML 도구들에 비해 상대적으로 저렴한데다 기능적으로도 손색이 없어 가격대비 성능이 우수하다고 알려져 있다.

4.3 오픈소스 제품

상용 제품들과 달리 Open Source 형태의 UML 도구들도 많이 있는데 대표적인 것들은 다음과 같다.

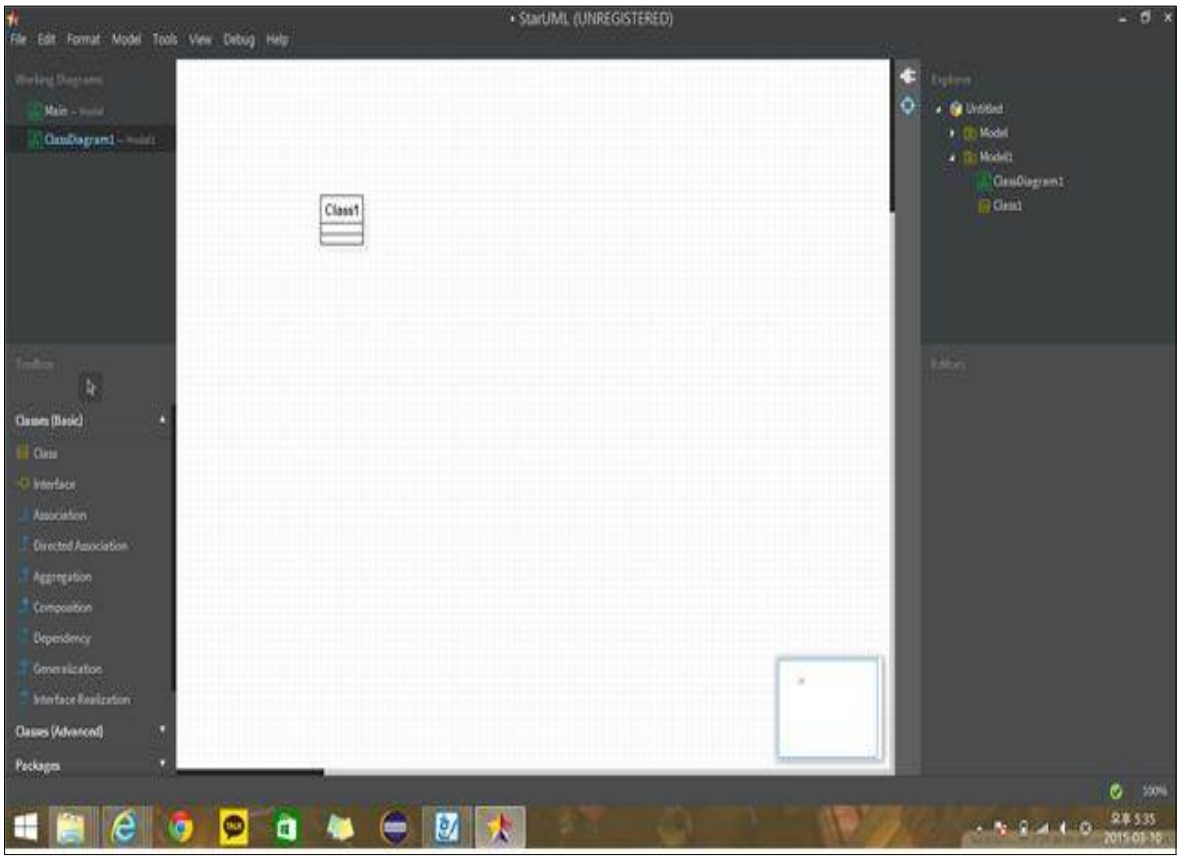
- StarUML
- ArgoUML
- JUDE/Community

• TOPCASED-UML2 (TOPCASED Modeling Framework Open Source Project)

• MDT-UML2Tools

이들 이외에도 개발도구인 Eclipse나 NetBeans를 위한 plug-in 형태로 제공되는 UML툴들이 많이 있다.

4.4 StarUML



4.4.1 개요

StarUML™은 UML(Unified Modeling Language)을 지원하는 소프트웨어 모델링 플랫폼이다. UML 버전 1.4에 기반을 두고 있으며, UML 버전 2.0의 표기법을 적극적으로 지원한다. 총 11가지의 다양한 종류의 다이어그램을 제공할 뿐만 아니라 UML 프로파일 개념과 템플릿 기반의 문서 및 코드 생성을 지원하여 MDA(Model Driven Architecture) 접근방법을 적극적으로 지원한다. 또한 고객의 환경에 대한 맞춤 능력이 우수하고 기능에 대한 확장성이 매우 뛰어난 것이 장점이다.

StarUML™은 고객의 환경에 최대한 적응할 수 있도록 설계되어 있다. 따라서, 고객의 소프트웨어 개발 방법론, 프로젝트의 플랫폼, 언어 등에 모두 적응할 수 있는 커스터마이징 변수들을 제공한다.

소프트웨어 아키텍처는 향후 10년 이상 내다보는 매우 중요한 작업입니다. OMG에서는 MDA 기술을 통해서 플랫폼에 독립적인 소프트웨어 모델을 구성하고 그것으로부터 플랫폼에 의존적인 모델이나 코드 등을 자동으로 얻을 수 있도록 하는 것을 지향하고 있습니다. StarUML™은 UML 1.4 표준 메타모델과 2.0 표기

법을 최대로 준수하면서 UML Profile 개념을 제공하여 플랫폼에 독립적인 모델을 작성할 수 있도록 지원하며, 간단한 템플릿 문서 작성만으로 고객이 원하는 산출물을 쉽게 얻을 수 있다.

StarUML™은 놀라운 유연성과 확장성을 제공한다. 도구의 기능을 확장하기 위한 Add-In 프레임워크를 제공하고, COM Automation을 통한 모델/메타모델 및 도구의 모든 기능에 접근할 수 있으며, 메뉴 및 옵션 항목까지도 확장할 수 있도록 설계되어 있다. 또한 고객의 방법론에 맞도록 접근법(Approach) 및 프레임워크(Framework)를 직접 추가 작성할 수 있고 어떠한 외부 도구와도 통합이 가능하다.

4.4.2 StarUML에서 생성할 수 있는 다이어그램의 종류

다이어그램 종류	설명
Class Diagram	클래스 다이어그램(Class Diagram)은 클래스관련 요소들의 여러 가지 정적인 관계를 시각적으로 표현한 것입니다. 클래스 다이어그램은 클래스(Class) 뿐만 아니라 인터페이스(Interface), 열거형(Enumeration), 패키지(Package) 및 여러 가지 관계들 뿐만 아니라 인스턴스(Instance)와 그것들의 연결(Link) 등도 포함할 수 있습니다.
Use Case Diagram	유스케이스 다이어그램(Use Case Diagram)은 특정 시스템 혹은 개체내의 유스케이스(Use Case)들과 그 외부의 액터(Actor)들 간의 관계를 표현한 것입니다. 유스케이스는 해당 시스템의 기능을 표현하며 그것들이 어떤 외부 액터들과 상호작용하는지를 나타냅니다.
Sequence Diagram	시퀀스 다이어그램(Sequence Diagram)은 인스턴스들이 어떻게 상호작용을 하는지를 묘사합니다. 하나의 협동-인스턴스집합에 포함된 인스턴스(Instance)들 상호간에 주고받는 자극(Stimulus)들의 집합인 상호작용-인스턴스집합(InteractionInstanceSet)을 직접적으로 표현합니다. 시퀀스 역할 다이어그램(Sequence Role Diagram)은 역할(ClassifierRole) 중심의 관점을 반영한 반면, 시퀀스 다이어그램(Sequence Diagram)은 인스턴스(Instance) 중심의 관점을 반영한 것입니다.
Sequence Diagram(Role)	시퀀스 역할 다이어그램(Sequence Role Diagram)은 역할 개념들이 어떻게 상호작용을 하는지를 묘사합니다. 하나의 협동(Collaboration)에 포함된 역할(ClassifierRole)들 상호간에 주고받는 메시지(Message)들의 집합인 상호작용(Interaction)을 직접적으로 표현합니다. 시퀀스 다이어그램(Sequence Diagram)은 인스턴스(Instance) 중심의 관점을 반영한 반면, 시퀀스 역할 다이어그램(Sequence Role Diagram)은 역할(ClassifierRole) 중심의 관점을 반영한 것입니다.
Collaboration Diagram	협동 다이어그램(Collaboration Diagram)은 인스턴스들이 어떻게 협동하는지를 묘사합니다. 하나의 협동-인스턴스집합에 포함된 인스턴스(Instance)들의 협동 모델을 직접적으로 표현합니다. 협동 역할 다이어그램(Collaboration Role Diagram)은 역할(ClassifierRole) 중심의 관점을 반영한 반면, 협동 다이어그램(Collaboration Diagram)은 인스턴스(Instance) 중심의 관점을 반영한 것입니다.

다이어그램 종류	설명
Collaboration Diagram(Role)	협동 역할 다이어그램(Collaboration Role Diagram)은 역할 개념들이 어떻게 협동하는지를 묘사합니다. 하나의 협동(Collaboration)에 포함된 역할(ClassifierRole)들의 협동 모델을 직접적으로 표현합니다. 협동 다이어그램(Collaboration Diagram)은 인스턴스(Instance) 중심의 관점을 반영한 반면, 협동 역할 다이어그램(Collaboration Role Diagram)은 역할(ClassifierRole) 중심의 관점을 반영한 것입니다.
Statechart Diagram	상태 다이어그램(Statechart Diagram)은 특정 개체의 동적인 행위를 상태(State)와 그것들간의 전이(Transition)를 통해 묘사합니다. 일반적으로 클래스의 인스턴스에 대한 행위를 묘사하는데 사용되지만 그 밖의 요소들에 대해서도 얼마든지 사용될 수 있습니다.
Activity Diagram	액티비티 다이어그램(Activity Diagram)은 상태 다이어그램의 특별한 형태로써, 활동들의 수행 흐름을 묘사하는데 적합합니다. 일반적으로 작업흐름(Workflow)을 표현하기 위해 많이 사용되며, 클래스, 패키지 혹은 연산 등의 개체에 대해 주로 사용됩니다.
Component Diagram	컴포넌트 다이어그램(Component Diagram)은 소프트웨어 컴포넌트 사이의 의존관계를 묘사합니다. 소프트웨어 컴포넌트를 구성하는 요소들과 그것들을 구현하는 요소들도 모두 표현될 수 있습니다.
Deployment Diagram	디플로이먼트 다이어그램(Deployment Diagram)은 물리적인 컴퓨터 및 장비 등의 하드웨어 요소들과 그것에 들이 배치되는 소프트웨어 컴포넌트, 프로세스 및 객체들의 형상을 묘사합니다.
Composite Structure Diagram	복합구조 다이어그램(Composite Structure Diagram)은 분류자(Classifier)의 내부 구조를 표현하는 다이어그램입니다. 여기에는 Classifier가 시스템의 다른 부분들과의 상호작용하는 지점 등을 포함합니다.