Introduction to UML

Software Modeling & Analysis

소프트웨어 모델링 및 분석

보고서 #1

Team. T1

201111388 조연호

201211374 이창오

201211379 장종훈

201314196 양동혁

목차

1. 개요

UML(Unified Modeling Language)은 시스템 디자인을 시각화하는 표준화된 방법을 제공하기 위한 소프트웨어 공학 분야의 모델링 언어이다.

객체 관련 표준화 기구인 OMG에서 1997년 11월 객체 모델링 기술(OMT; Object Modeling Technique), OOSE 방법론 등을 연합하여 만든 통합 모델링 언어로 객체 지향적 분석 · 설계 방법론의 표준 지정을 목표로 하고 있다. 요구 분석, 시스템 설계, 시스템 구현 등의 과정에서 생길 수 있는 개발자 간의 의사 소통의 불일치를 해소할 수 있다. 모델링에 대한 표현력이 강하고 비교적 모순이 적은 논리적인 표기법(Notation)을 가진 언어라는 장점이 있다. 따라서 개발자 간의 의사 소통이 쉬워지며 생략되거나 불일치 하는 모델링 구조에 대한 지적도 용이하고, 개발하려는 시스템 규모에 상관없이 모두 적용 가능하다.

유스 케이스 다이어그램(Use Case Diagram), 클래스 다이어그램(Class Diagram) 등 8개의 다이어그램을 기반으로 객체지향 소프트웨어를 개발하기 위한 풍부한 분석 및 설계 장치를 제공하고 있어 향후 상당 기간 동안 산업계의 표준으로 활용될 것이라 예상된다. UML을 가장 잘 적용할 수 있는 소프트웨어 개발 프로세스는 1998년 11월 미국 래셔널(Rational)사에서 개발한 통합 프로세스(Unified Process) 5.0이다. 이 프로세스는 웹 애플리케이션(Web Application) 개발에 효율적이고 개발팀의 생산성을 극대화하며 UML의 장점을 최대한 살릴 수 있도록 고안된 실무형 개발 프로세스이다.

2. 소개

A. 배경

UML은 Grady Booch의 방법론과 James Rumbaugh의 OMT(Object-Modeling Technique), Ivar Jacobson의 OOSE(Object-Oriented Software Engineering)의 표기법에 기초하였으며 후에 하나의 언어로 통합되었다.

1994년 Booch가 세운 Rational Software Corporation에 Rumbaugh가 합류하고, 1년 후 Jacobson이 합류하면서 이들의 연구는 하나로 결집되어 UML 드래프트 버전이 탄생하였다. 이들은 UML을 OMG(Object Management Group)에 표준화 제정을 위해 HP, DEC, IBM, Microsoft 등 유수의 멤버로 결성된 UML 컨소시엄을 발족하였다.

1997년 UML 컨소시엄은 UML 버전 1.0을 만들어내고 이를 OMG에 제출하여 그 해 말에 OMG는 이를 수정한 UML 1.1을 표준 모델링 언어로 채택하게 된다.

B. 목적

기존의 객체 지향 방법론과 함께 제안되어 모델링 언어 표기법의 표준화를 하기 위함이다.

C. 특징

i. 가시화 언어

UML은 각 심볼(Symbol)에 명확한 저의가 존재하여 개념 모델 작성에 있어서 오류 없이 전달이 가능하다. 또한 개발자들 사이 원활한 의사소통이 가능한 그래픽 언어이다.

ii. 명세화 언어

명세화란 정확하고 명백하며 완전한 모델을 만드는 것을 의미한다. UML은 소프트웨어 개발 과정인 분석, 설계, 구현 단계의 각 과정에서 필요한 모델을 명세화 할 수 있는 언어이다.

iii. 구축 언어

UML로 명세화된 설계모델은 JAVA, C++, VB 등 다양한 언어의 소스 코드로 변환하여 구축할 수 있다. 반대로 구축되어있는 소스코드를 UML로 변환하여 분석하는 역공학도 가능하다.

iv. 문서화 언어

UML은 시스템 아키텍처(Architecture)와 이에 대한 모든 상세 내역에 대한 문서화를 다루며, 요구사항을 표현하고 시스템을 테스트하는 언어도 제공한다.

D. 용도

실제 시스템 개발의 현장에서는 UML을 주로 3가지의 용도로 쓰이는 경우가 많다.

i. 모델링

'무엇을 만들까?'를 의식해 유저의 요건을 묻기 위해서 모델링이라고 하는 기법을 사용해 시스템의 전체상을 그리는 작업을 하는 일이 있다. 이 작업을 실시하는 사람을 '모델러'라고 부르고 모델링에 의해서 작성하는 그림을 '개념 모델'이라고 부른다.

개념 모델 자체는 어떠한 표현을 해도 상관없지만, 일반적으로는 UML 의 클래스 다이어그램을 사용하는 경우가 많다. 그 이유 중 하나는 설계자가 이해하기 쉽기 때문이다.

또 클래스 다이어그램의 읽는 법이 몇 가지는 결정되어 있긴 하지만, 기억할 것은 많지 않다. 직감적으로 판단할 수 있기 때문에 UML 에 익숙하지 않은 유저도 이해하기 쉬어서 개념 모델(Class Diagram)에 대한 평가는 대체로 높다. 전체상을 시각적으로 이해할 수 있어 유저 자신이 깨닫지 못했던 과제도 발견할 수 있다는 장점이 있다.

ii. 설계

요건 정의 국면에 UML 을 사용해 작성한 그림은 설계에서 보다 구체화된다. 예를 들면, 개념 모델(Class Diagram)이나 State Machine Diagram 으로부터 데이터베이스의 논리 설계를 행하거나 실제 이미지에 접근하기 위해서 클래스의 상세화를 한다.

설계에서 Class Diagram 을 사용하는 가장 큰 장점은 클래스 간 인터페이스를 빠른 단계에서 명확하게 할 수 있는 점이다. 설계에서 쓰는 Class Diagram 에는 클래스의 속성이나 관계뿐만 아니라 조작도 나타내게 되어 있다.

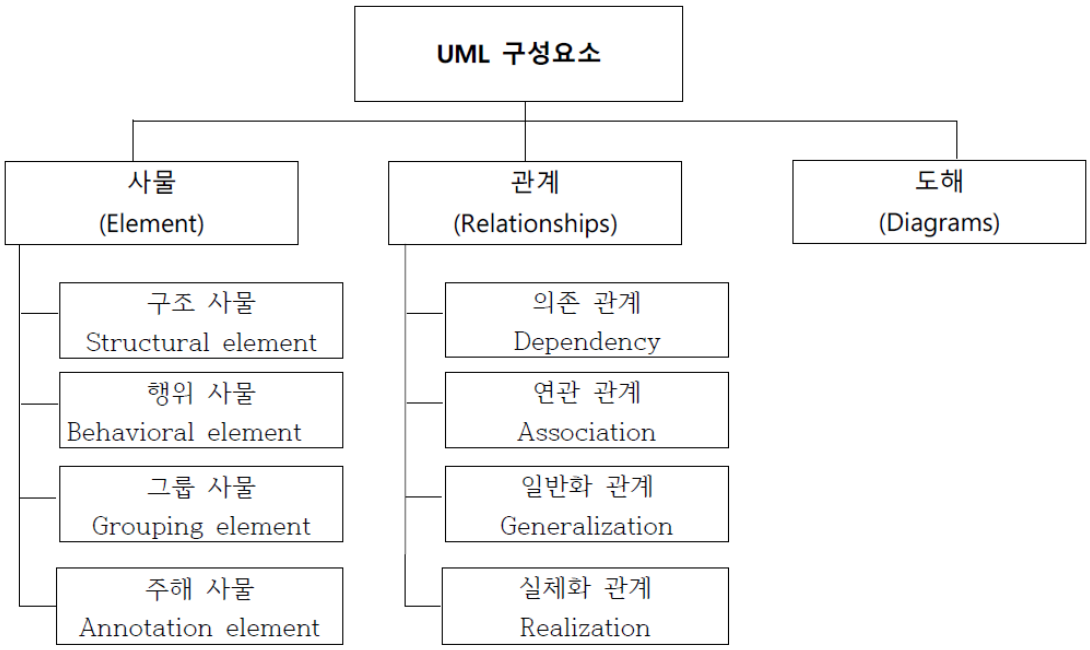
덧붙여 설계 시에 Class Diagram 이나 순서도 등을 만드는 것은 필수가 아니고, 비교적 소규모의 시스템으로 개발 멤버 사이에 미리 설계 방법이 공유되어 있는 경우에는, 필요에 따라서 코딩제의 프로그램으로부터 역공학 하여 Class Diagram 등을 작성해, 설계서를 나중에 작성하기도 한다. 역공학 기능은, 이후 설명하는 툴로 지원되고 있다.

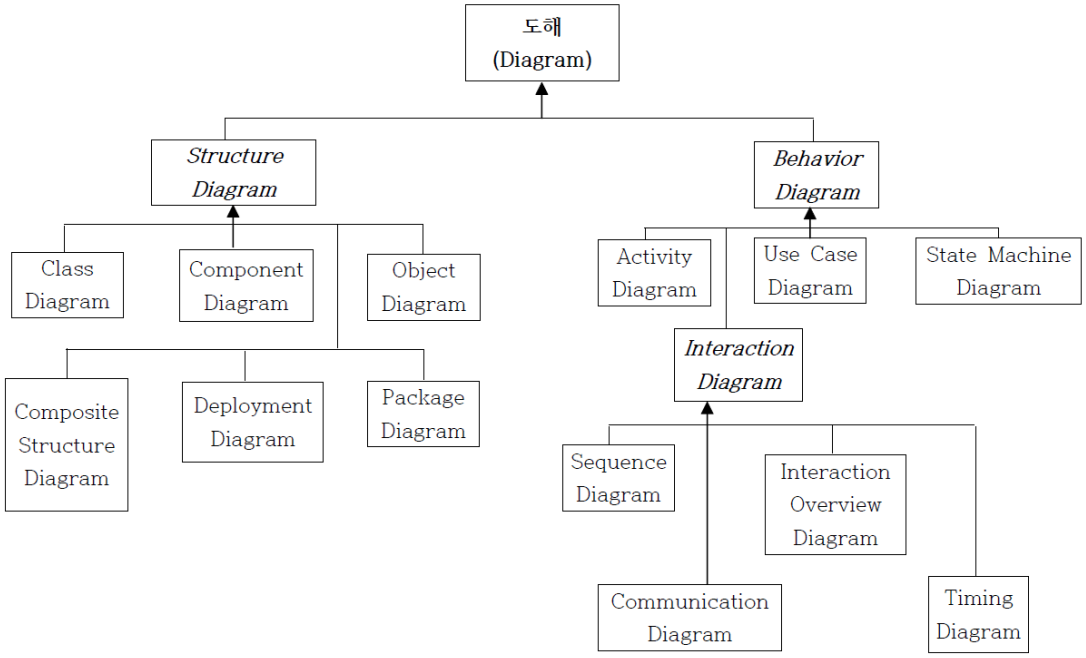
iii. 프로그래밍

실행 환경에 의존하지 않는 UML 모델로부터 툴을 사용해 실제로 움직이는 프로그램으로 변환하는 기술을 MDA(Model Driven Architecture)라고 부른다. MDA 에 준거하면, 모델링→설계→프로그래밍으로의 변환을 모두 UML 만으로 할 수 있게 된다.

그러나 실제 현장에서는 MDA 는 거의 보급되어 있지 않고, 변함없이 프로그래머가 설계서를 보면서 손으로 코딩을 하는 스타일이 여전히 계속 되고 있다. 현시점에서는 툴 자체가 지원하고 있는 기술이 미성숙하기 때문에 실제 현장에서 MDA 가 적극적으로 사용되지 않고 있다.

3. 구성요소





A. 사물(Element)

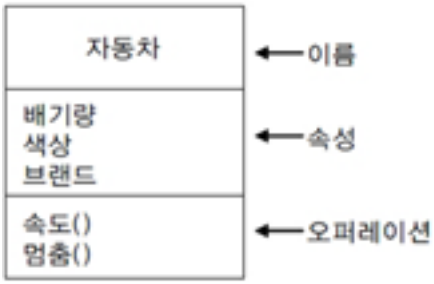
사물(Element)은 추상적 개념으로 모형 구성의 기본 요소이다.

i. 구성요소

1. 구조 사물(Structural Element)

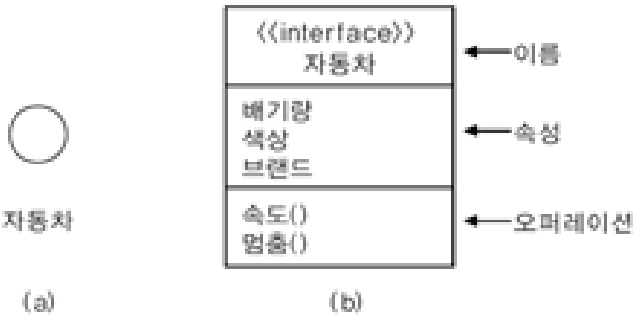
구조 사물(Structural Element)은 모형의 명사형으로 정적인 부분이다. 개념적이거나 물리적 요소를 표현한다.

- **Class**



Class는 같은 종류의 객체 집합으로, 그 특성을 표현한다.

- **Interface**



Interface는 클래스나 컴포넌트의 서비스를 명세하기 위한 것으로, Operation의 집합이다.

- **Communication**

Communication은 교류를 정의하며, 서로 다른 요소와 역할들의 집합이다.

- **Use Case**

Use Case는 시스템이 수행하는 활동들을 순차적으로 기술하여 표현한다.

- **Component**

Component는 시스템의 물리적이고 대체 가능한 부분이다. 서로 다른 논리 요소를 물리적으로 패키지화한 것이다.

- **Active Class**

Active Class는 동작 상태와 활동 상태로 두 가지가 존재한다. 동작 상태는 더 이상 분할되지 않지만, 활동 상태는 다른 제어 흐름을 가지는 활동 또는 동작 상태로 분해가 가능하다.

- **Node**

Node는 실행할 때에 존재하는 물리적 요소이다.

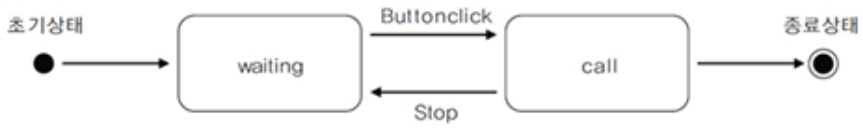
2. 행위 사물(Behavioral Element)

행위 사물(Behavioral Element)은 모델의 동사형으로 동적인 부분을 뜻한다. 시간과 공간에 따른 행동 요소를 표현한다.

- **Interaction**

Interaction은 어떠한 목적을 위해 객체들 간 주고받은 메시지로 구성된다.

- **State Machine**



State Machine은 상태의 순서를 지정하는 행동이다.

3. 그룹 사물(Grouping Element)

그룹 사물(Grouping Element)은 UML 모형을 조직하는 부분으로, 모델을 분해하여 재구성화 할 수 있는 단위 상자이다.

- **Package**

Package는 요소를 그룹으로 묶는 역할을 한다.

4. 주해 사물(Annotation Element)

주해 사물(Annotation Element)은 UML 모형을 설명하는 부분이다. Comment로 모형 요소를 설명하고 표현하는 도구이다.

- **Note**

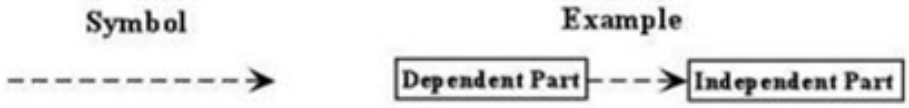
Note는 제약과 주석을 나타내기 위해 사용된다.

B. 관계(Relationship)

관계(Relationship)은 구성 요소간의 의미 있는 연결이다.

i. 구성요소

1. 의존 관계(Dependency)



의존 관계(Dependency)는 두 사물간의 의미적인 관계로, 한쪽 사물의 변화가 다른 사물에 영향을 주는 관계를 뜻한다.

2. 연관 관계(Association)

연관 관계(Association)는 객체 간 연결의 집합이다. 집단 연관 관계를 표현한다.

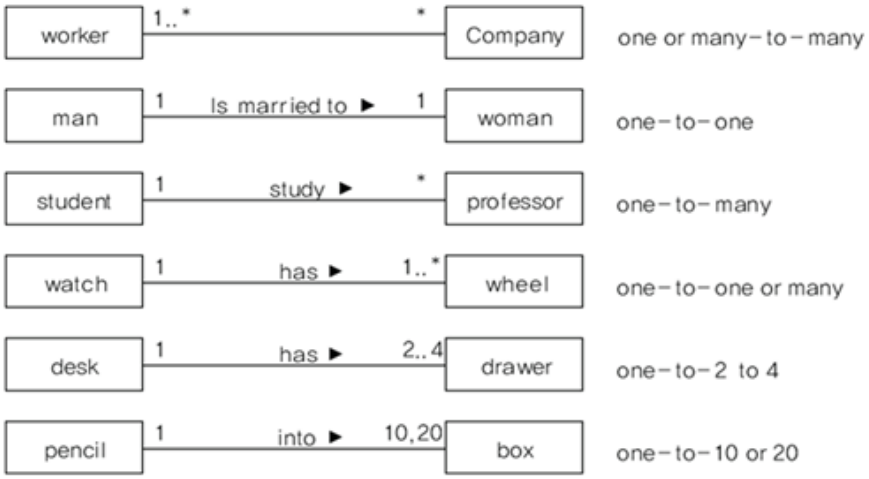
- **이름**

이름은 관계의 의미를 설명하기 위해 사용된다.

- **역할**

역할은 클래스가 연관에 참여하면 그것이 수행해야 하는 특별한 역할을 의미한다.

- **다중성**



다중성은 한 연관에 참여하는 하나의 객체에 몇 개의 객체가 연결되었는지 일컫는다.

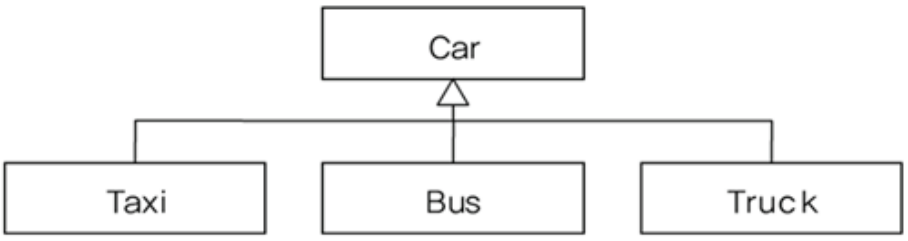
- **집합 연관**

집합 연관은 전체-부분 관계로, 전체는 부분을 참조한다. 이들은 서로 독립적으로 생성 및 소멸된다.

- **복합 연관**

복합 연관은 전체-부분 관계로, 전체는 부분을 포함한다. 부분은 생성 및 소멸을 전체와 함께 한다.

3. 일반화 관계(Generalization)



일반화 관계(Generalization)는 일반화된 사물과 특수화된 사물 사이의 관계(is-a-kind-of 관계)이다.

4. 실체화 관계(Realization)

실체화 관계(Realization)는 객체들 사이의 의미적 관계로, 한 객체가 다른 객체의 계약을 지정한다.

실체화의 간단한 표현은 아래의 그림과 같다.

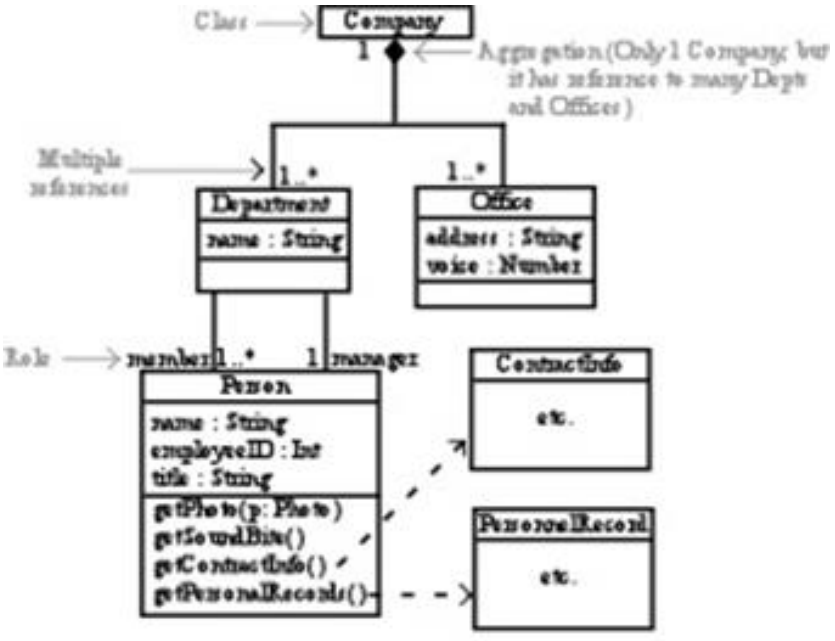
C. 다이어그램(Diagram)

다이어그램(Diagram)은 구성 요소들의 그래픽 표현이다.

i. 구성요소

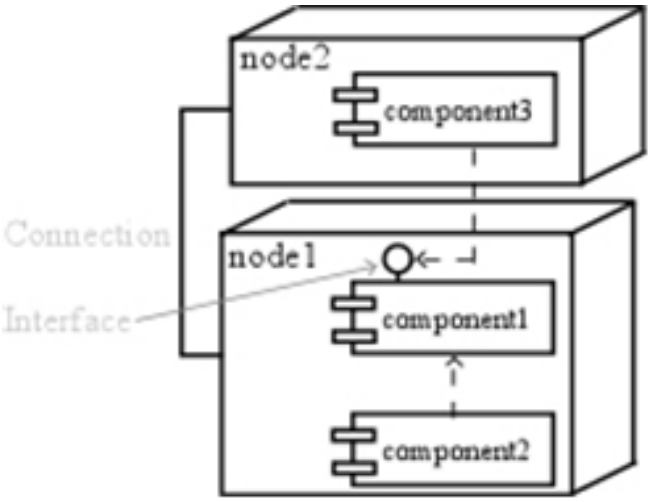
1. 정적 다이어그램(Structure Diagram)

- **Class Diagram**



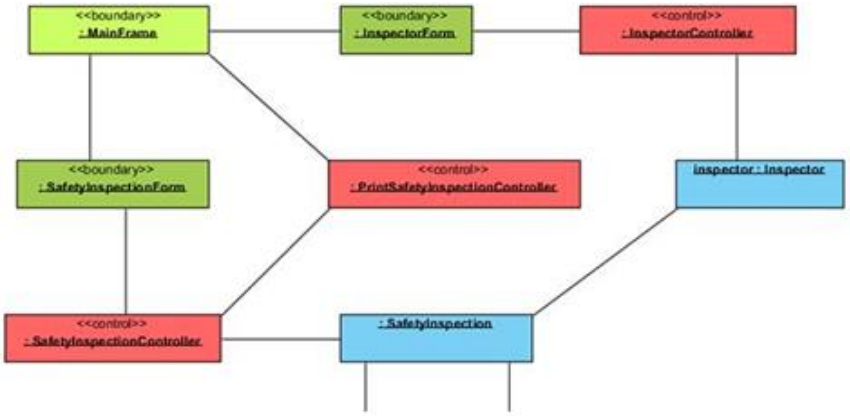
Class Diagram은 클래스와 인터페이스, 통신 그리고 이들의 관계를 나타낸다. 또한 시스템의 정적 설계 View를 다룬다.

- **Component Diagram**



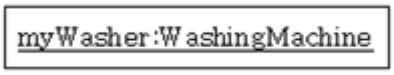
Component Diagram은 컴포넌트 사이의 구성과 의존을 나타낸다.

- **Object Diagram**

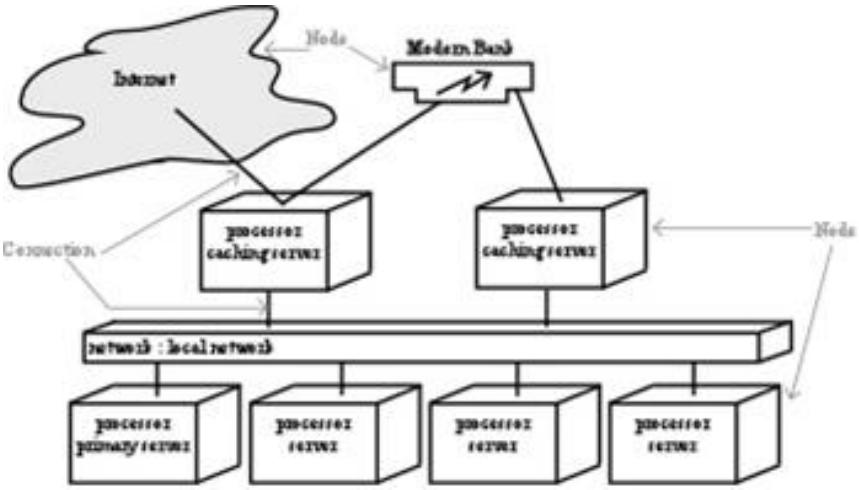


Object Diagram은 객체와 객체들 사이의 관계를 나타낸다. 또한 특정 시점의 객체들의 구조적 상태를 표현한다.

표기법은 아래의 그림과 같다.

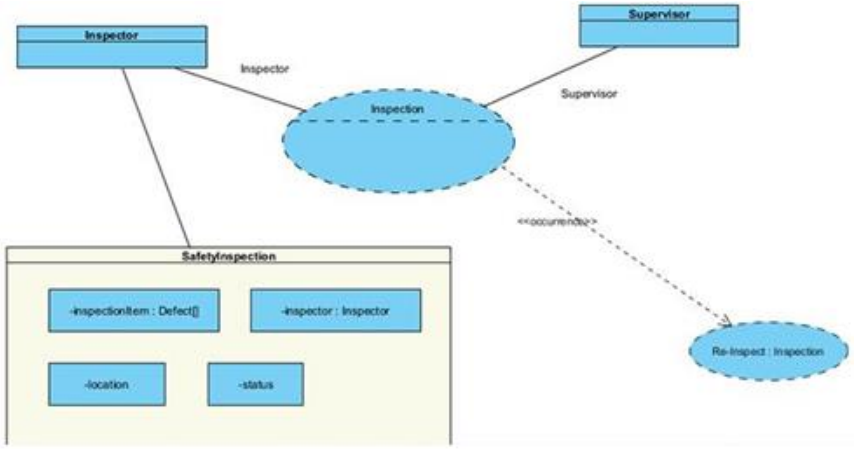


- **Deployment Diagram**



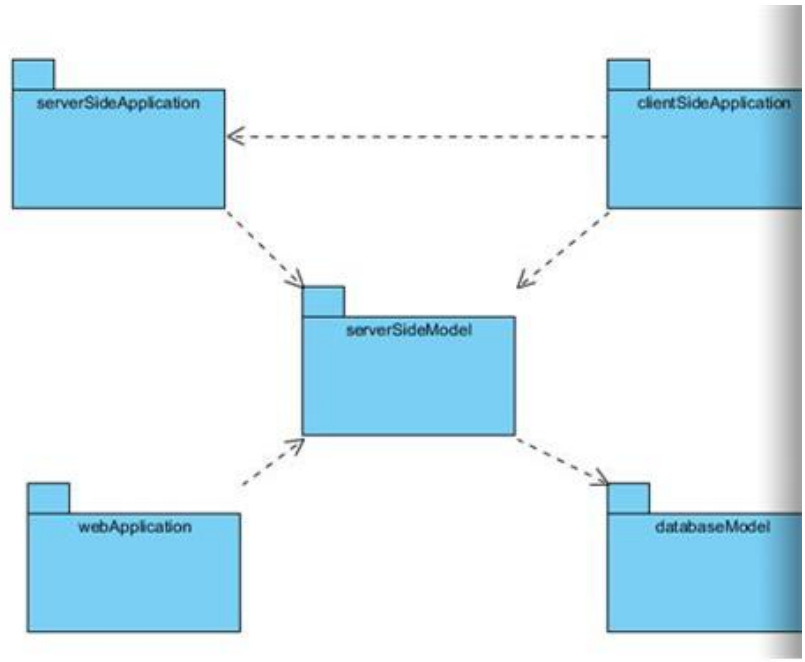
Deployment Diagram은 시스템을 구성하는 노드와 그 노드에 있는 컴포넌트를 구성한다.

- **Composite Structure Diagram**



Composite Structure Diagram은 컴포넌트의 내부 구조를 표현한다.

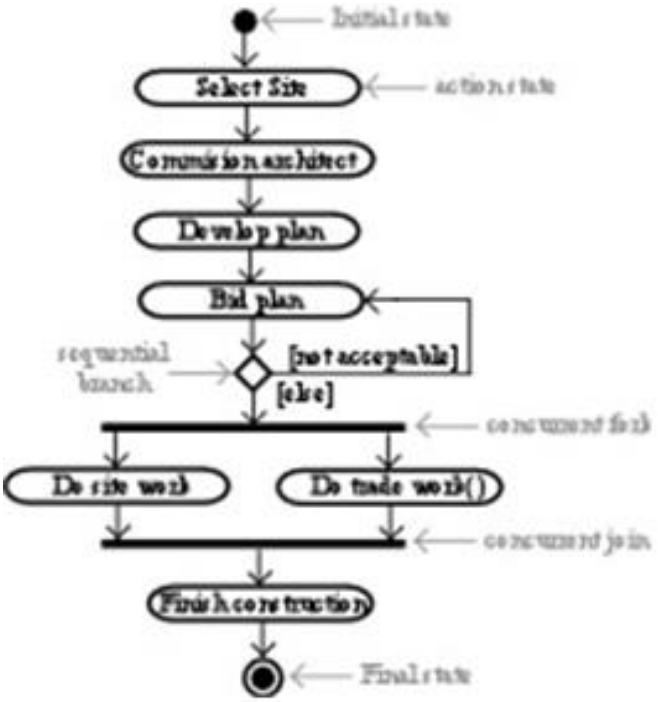
- **Package Diagram**



Package Diagram은 다이어그램의 요소를 조직화하여 패키지 형태로 나타낸다.

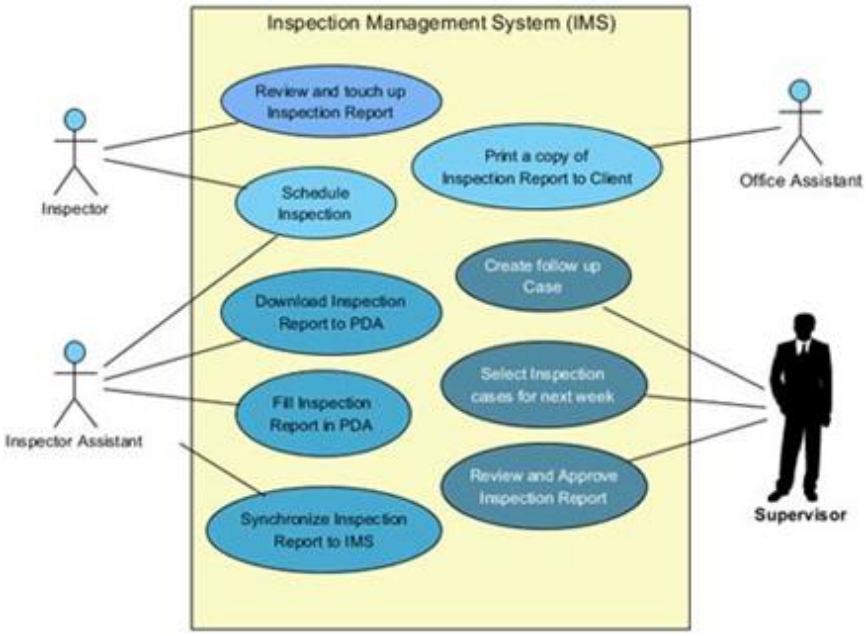
2. 동적 다이어그램(Behavior Diagram)

- **Activity Diagram**



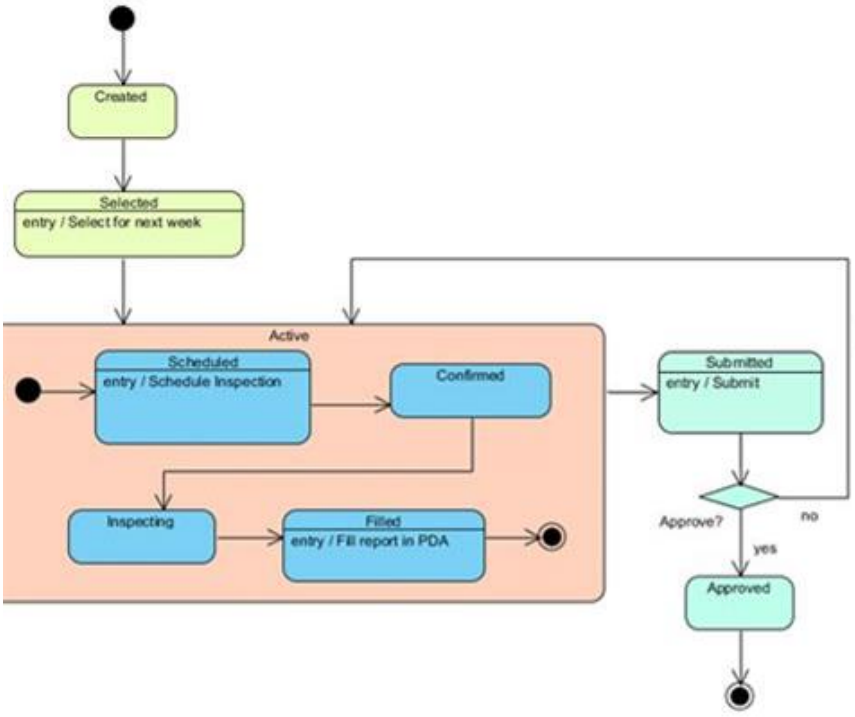
Activity Diagram은 시스템 내부에 있는 활동의 흐름을 표현한다.

- **Use Case Diagram**



Use Case Diagram은 Use Case와 행위자의 관계를 구조적으로 나타낸다.

- **State Machine Diagram**



State Machine Diagram은 오브젝트의 상태 변화를 표현한다.

- **Interaction Diagram**

3. 교류 다이어그램(Interaction Diagram)

- **Sequence Diagram**

  Sequence Diagram은 시스템 외부 이벤트를 처리하기 위하여 시스템 내부 객체 간 주고받는 동적 메시지를 시간 흐름에 따라 표현한다.

- **Communication Diagram**

  Communication Diagram은 Sequence Diagram과 동일한 내용을 객체 상호 관계의 관점에서 표현한다.

- **Interaction Overview Diagram**

  Interaction Overview Diagram은 객체 사이에 시간의 흐름을 갖는 메시지가 존재한다면 몇몇 Sequence Diagram과 Communication Diagram으로 바꿔야 하는 것을 나타낸다.

- **Timing Diagram**

  Timing Diagram은 객체, 컴포넌트간의 상태 변화, 메시지에 대한 정확한 시간 정보를 표현한다.

4. 발전

A. UML 1.0의 한계

객체지향 모델링과 더불어 널리 사용하게 된 모델링 언어인 UML 1.0은 사용범위가 넓어지면서 여러 가지 한계를 드러냈다. 문제로 지적된 것은 크기가 지나치게 크고, 복잡하며, 의미(Semantics)가 명확하지 않고, 구현과 관련된 부분이 부족하여 커스터마이징(Customizing)이 제한되어 있다는 것 등이다. 뿐만 아니라 컴포넌트 기반의 개발 방법을 제대로 지원하기 어렵고, 모델 다이어그램을 교환할 방법이 없다는 것도 또 다른 문제점으로 지적되고 있다.

i. 복잡성

UML 1.0이 지나치게 크고 복잡하다는 것은 이미 오래 전부터 언급되던 것이다. 크고 복잡하면 배우기 어렵고, 적용이나 구현에 있어서 접근성에 영향을 미치기 때문에 이를 간소화하는 것은 매우 중요한 이슈가 된다.

ii. 낮은 이해성

UML 규격의 의미(Semantics)나 표기법(Notation)의 상세 내용에 대해 이를 정확하게 이해하기 어렵다는 것도 중요한 문제이다. 의외로 많은 UML 사용자가 UML의 의미(Semantics)를 제대로 이해하지 못하고 있으며, 여기에 따른 문제로 어려움을 겪는다.

iii. 낮은 간결성

UML의 크기를 줄이는 가장 좋은 시작점은 언어 자체를 정확하고 간결하게 정의하는 것이다. 이렇게 함으로써 UML은 쉽게 배울 수 있고, 구현하기도 쉬워질 것이다. 그리고 벤더나 사용자에 의해 언어를 쉽게 커스터마이징(Customization)할 수 있게 함으로써 서로 다른 도메인(예를 들어 금융 서비스, 보건 의료, 통신 등)과 서로 다른 플랫폼(J2EE, .NET, CORBA)에 효과적으로 대처할 수 있도록 변형될 수 있다.

간결하고 명확한 언어는 UML 구현이 규격에 부합되도록 하는 데도 일조할 것이다. 사실 UML 1.1이 1997년에 채택되었지만, UML 언어의 전체 규격을 아직도 완벽하게 구현하지 못하고 있다.

iv. 컴포넌트 개념 미지원

컴포넌트 기반 개발을 지원하는 컴포넌트의 개념이 지원되지 않는다. 현재의 개발 프로세스의 변화 추이에 따라 이를 위한 컴포넌트 패러다임의 지원이 필요하다.

v. 모델 교환 미지원

지금까지는 Vendor들 사이의 모델 교환이 이루어지지 않았다. 때문에 실질적으로 서로 다른 모델링 도구들 사이의 모델을 효과적으로 공유하는 것은 사실상 불가능하다.

vi. 아키텍처(Architecture) 설계 미지원

아키텍처(Architecture) 설계를 위한 다이어그램이 제공되지 않는다. 때문에 Package Diagram이나 Class Diagram을 변형해서 사용할 수 밖에 없다.

vii. 모델-코드 간 불일치성

실질적으로 모델의 Behavior 부분을 기술할 수 없었기 때문에 모델과 코드가 일치하지 않는 경우가 많으며, 이를 해결하기 위해 Round-Trip Engineering은 잘 적용되지 않고 있다.

B. UML 2.0으로의 발전

i. Composite Structures를 통한 Component-based Development4 지원

Structured Classifiers(Classes, Components)는 Parts, Ports, Connectors를 통해서 계층적으로 분해되고 결합될 수 있다. 이는 SDL(Specification & Description Language)의 Block Diagram과 유사한 방법으로 작성된다.

ii. Structure와 Behavior의 계층적 분해 지원

Structural Constructs인 Classes와 Components뿐만 아니라 Interactions, State Machines, Activities와 같은 주요 Behavioral Constructs의 계층적 분해도 지원한다.

iii. Structure와 Behavior의 Cross4 integration

Class의 내부 구조를 보여주기 위하여 Composite Structure Diagram에서 사용된 동일한 Parts는 내부 Structures가 서로 Communicate하는 방법을 보여주기 위해 Sequence Diagram에서 사용될 수 있다.

iv. Interaction의 향상된 표현

기존의 MSC(Message Sequence Chart)의 개념을 받아들여 Sequence Diagram에 매우 향상된 Semantics을 제공한다.

v. State Diagram의 향상된 표현

기존의 State Diagram에 Inheritance 기능 등 여러 요소의 추가뿐만 아니라 SDL의 Syntax를 가져와서 Transition-Oriented State Machine Diagram을 제공한다.

vi. Behavioral Constructs를 갖는 Action Semantics의 통합

UML Actions은 Simulation과 Code Generation을 위한 실행 가능한 모델을 정의할 수 있도록 Programming Language의 Actions처럼 상세히 정의될 수 있다.

vii. 점증적 구현과 Compliance Testing을 편리하게 하기 위한 Layered Architecture

UML 2.0 Packages는 Vendor들이 표준을 보다 효과적으로 보다 편리하게 구현할 수 있도록 하기 위해 3개의 Layers(Basic, Intermediate, Complete)로 구성된다.

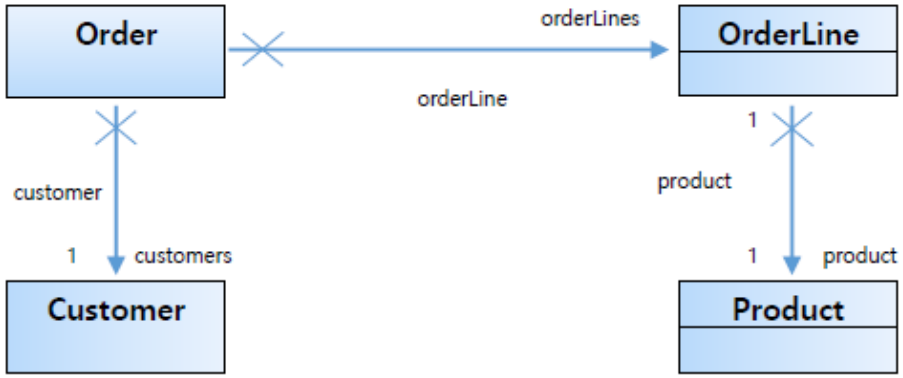
viii. Fully Design Systems

UML 2.0은 전체 시스템을 디자인하는 것을 가능하게 한다. UML 2.0을 사용해서 Architecture, Data, Interaction, Behavior를 기술할 수 있다. 이것은 시스템 모델로부터 완전한 코드를 생성해낼 수 있게 해준다. UML 2.0을 사용하여 요구사항을 기술하는 것도 가능하다.

C. UML 2.0의 특징

i. 관계 표현

1. Association

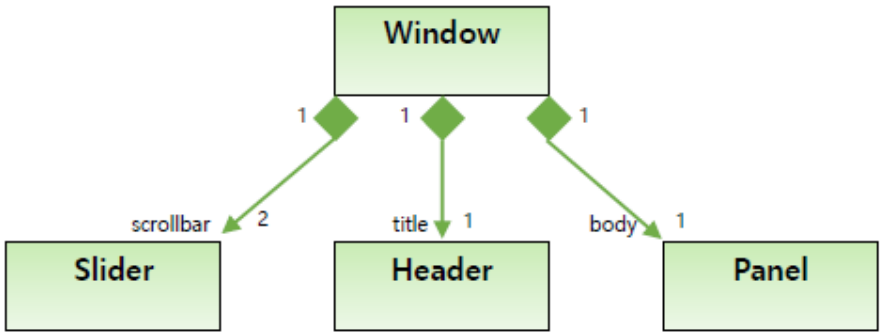


- 클래스 간에 표현되는 개념적 관계로 객체간 연결을 나타내는 구조적 관계를 표현한다.
- 각각 클래스마다 Role Name 및 Association Name 표시가 가능하다.
- Association 중 Whole/Part 관계를 표현하는 것으로 Aggregation 과 Composition 이 있다.

2. Aggregation

- Whole 클래스가 하나 이상의 Part 클래스로 구성되는 경우 Whole과 Part간의 생명주기 관계가 없다.

3. Composition



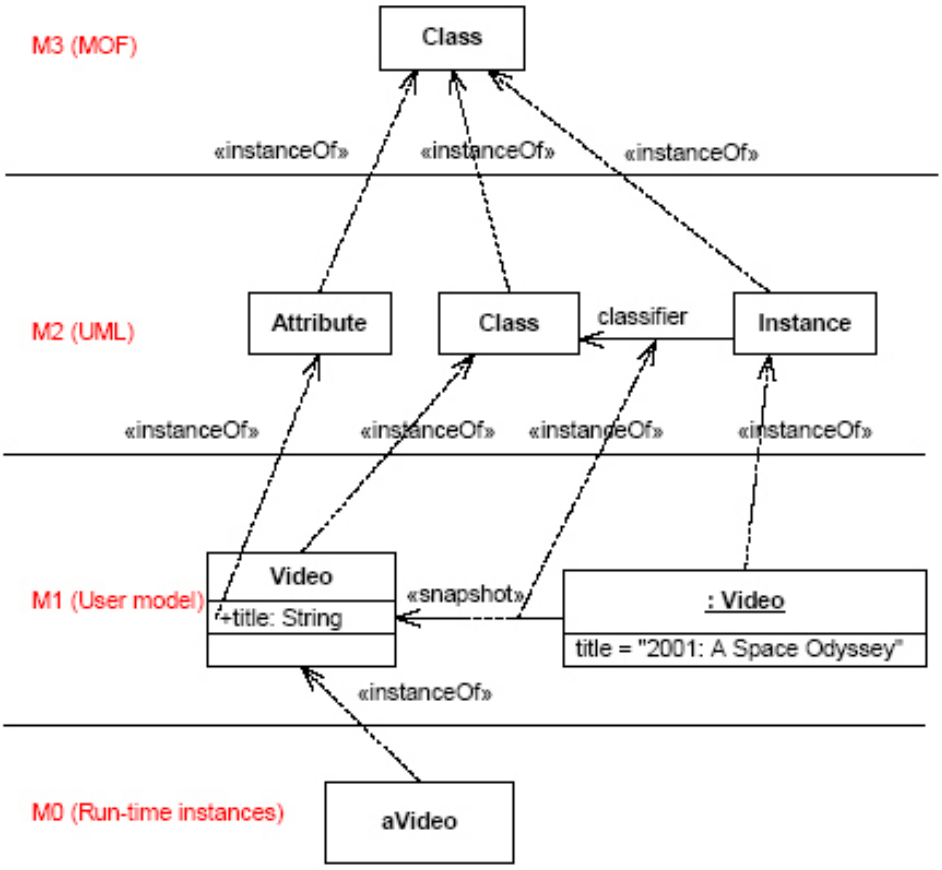
- Whole 클래스의 객체가 없어질 때, Part 클래스의 객체도 함께 없어진다. (Whole과 Part의 생명주기가 같다)
- Part 단독으로 존재하는 것은 의미가 없으며, Part 클래스의 Instance는 다른 Whole 클래스에 공유될 수 없다.

ii. 표준의 구성

4개의 UML 명세는 메타 모델로 작성되어 있고 개념이 까다롭기 때문에 제대로 파악하는 일은 매우 어렵다. UML로 작성된 산출물을 이해하거나 UML을 이용하여 산출물을 작성하기 위해서는 상부구조의 각종 다이어그램의 요소를 파악하고 다이어그램을 이해하는 정도로 충분하다.

Part	설명
상부구조(Superstructure)	13개의 다이어그램과 그 다이어그램에 등장하는 요소들에 대한 명세(행위형 다이어그램)
하부구조(Infrastructure)	상부구조에 대한 기본이 정의(메타 모델)
Object Constraint Language	객체 제약 언어
Diagram Exchange	UML 도구들이 다이어그램을 교환하기 위해 필요한 명세

iii. 4계층 구조



계층	설명
M3	MOF(Meta Object Facility)는 M2 수준에 속한 메타 모델을 정의하는 메타메타 모델
M2	UML 기반의 설계를 가능케 하는 Attribute, Class, Instance 등과 같은 모델 요소를 정의하는 메타 모델 UML 2.0의 하부구조는 4계층 메타 모델 관점에서 M2 수준의 UML 메타 모델
M1	시스템 분석가나 설계자들이 일반적인 모델링 케이스 도구를 통해 특정 도메인 시스템을 설계한다고 했을 때의 메타 모델 수준 사용자 모델을 도식하게 되는 수준
M0	모델이 만들어낸 코드 실행수준의 단계

D. UML 2.0의 효과적 활용방안

i. 비즈니스 도메인의 철저한 분석 선행

UML 2.0을 효과적으로 활용하기 위해서는 설계하고자 하는 해당 도메인에 대한 철저한 분석이 선행되어야 한다. 철저한 분석 없이는 일정한 추상화 수준을 유지하기 어렵기 때문에 유기적인 모델을 만들어 낼 수 없다는 사실을 기억해야 한다.

ii. 모델의 추상화 수준 확립

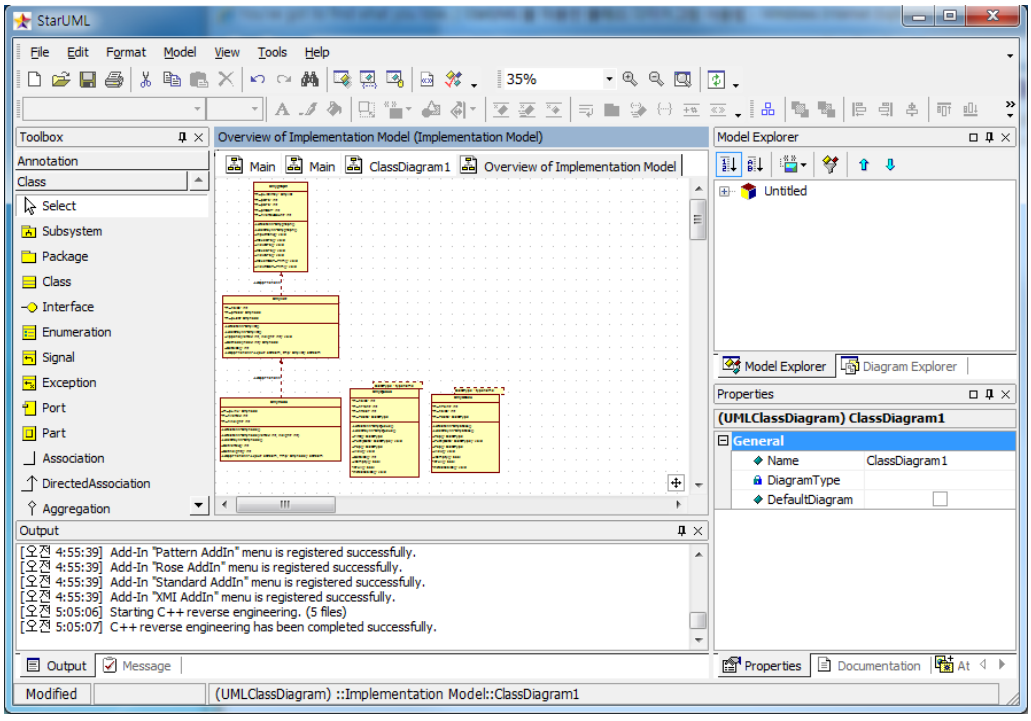
모델을 추상화하는 과정에서 다이어그램의 핵심 표기들 간 추상화 수준이 일관된 원칙(Principle)에 따라 정립될 수 있도록 설계 작업을 수행하여야 UML 2.0을 효과적으로 활용할 수 있다.

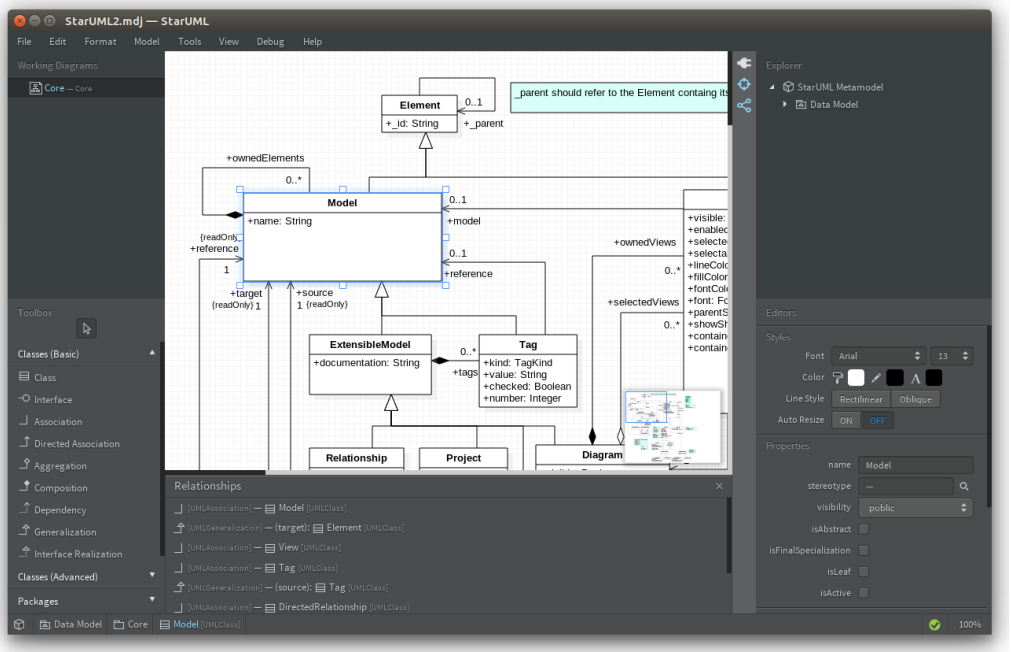
iii. 모델 자체의 높은 완성도 추구

UML 2.0의 강력한 표현력(Semantic Expressiveness)과 섬세함(Elements Precision), 다이어그램 간의 유기적 연결성 지원(Support for Diagram Interchange) 기능으로 모델 결과물 자체에 대한 완성도 검증(Self Verification)이 가능하므로 모델 자체의 높은 완성도를 추구하여 UML 2.0을 최대한 활용할 수 있어야 한다.

5. UML 툴

A. StarUML





StarUML은 국내 소프트웨어 업체 Plastic Software에서 개발된 가장 대중적으로 사용되고 있는 무료 UML 툴이다. UML 1.4에 기반을 두고 있으며, UML 2.0 표기법을 적극적으로 지원하고 있다. 심플한 인터페이스로 학습 및 사용이 용이하여 개발 효율을 높일 수 있다.

i. 특징

1. 정확한 UML 표준 모델

StarUML은 OMG에서 제정한 UML의 표준 명세에 따라 소프트웨어 모델을 작성할 수 있도록 도와준다. 특히 UML 1.4 표준 구문과 의미의 준수를 극대화하고, 견고한 메타모델의 기반에서 UML 2.0의 표기법을 적극적으로 수용하여 설계한 정보의 지속성을 넓혔다.

2. 개방적 소프트웨어 모델 포맷

독자적인 포맷으로 작성하는 외국산 UML 툴과는 달리 StarUML은 세계 표준인 XML 포맷으로 구성된다. 표준 이에 따라 사용자들이 쉽게 식별할 수 있으며, 누구든지 XML 파서를 이용하여 포맷을 원하는 형태로 변환하여 사용할 수 있다.

3. MDA 지원

StarUML은 UML 프로파일을 완벽하게 지원하여 UML의 확장성을 극대화 시켰다. 따라서 어떠한 영역의 어플리케이션과도 모델링이 가능하며, 각종 문서나 실제 실행 가능한 코드(Executable Code)를 자동으로 생성할 수 있다.

4. 방법론 및 플랫폼의 적응성

StarUML은 접근법(Approach)이라는 개념을 도입하여 어떠한 방법론 또는 프로세스에도 적응할 수 있는 환경을 만들 수 있다.

5. 뛰어난 확장성

StarUML 도구의 모든 기능이 Microsoft의 COM 자동화가 되어 있어 Visual Basic Script 또는 Java Script, VB, Delphi, C++, C#, VB.NET, Python 등과 같은 COM 지원 언어에서도 StarUML을 제어하고 통합된 추가 모듈을 개발할 수 있다.

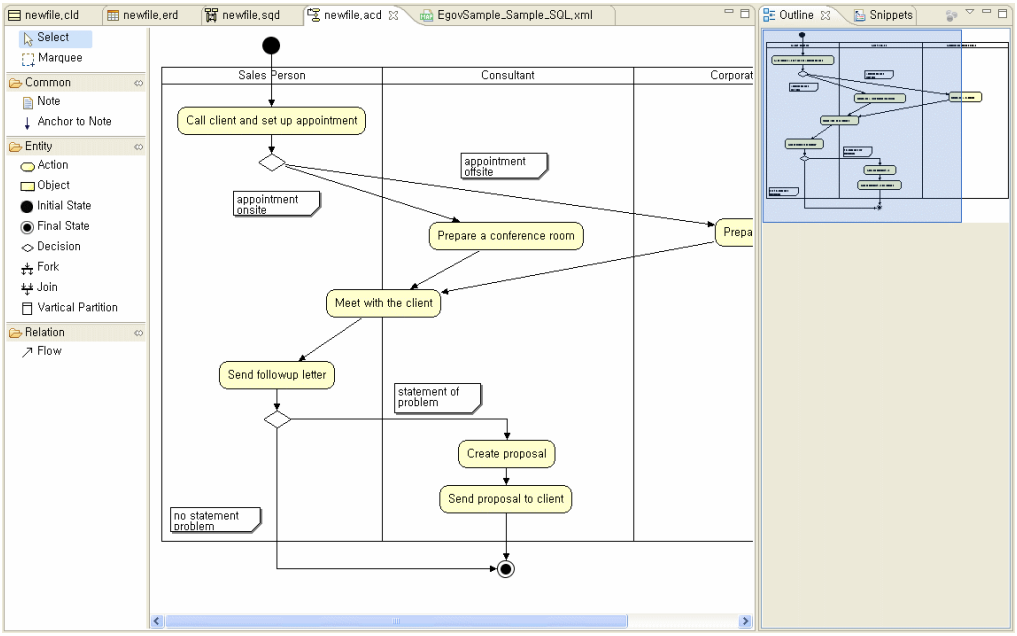
6. 소프트웨어 모델 검증 기능

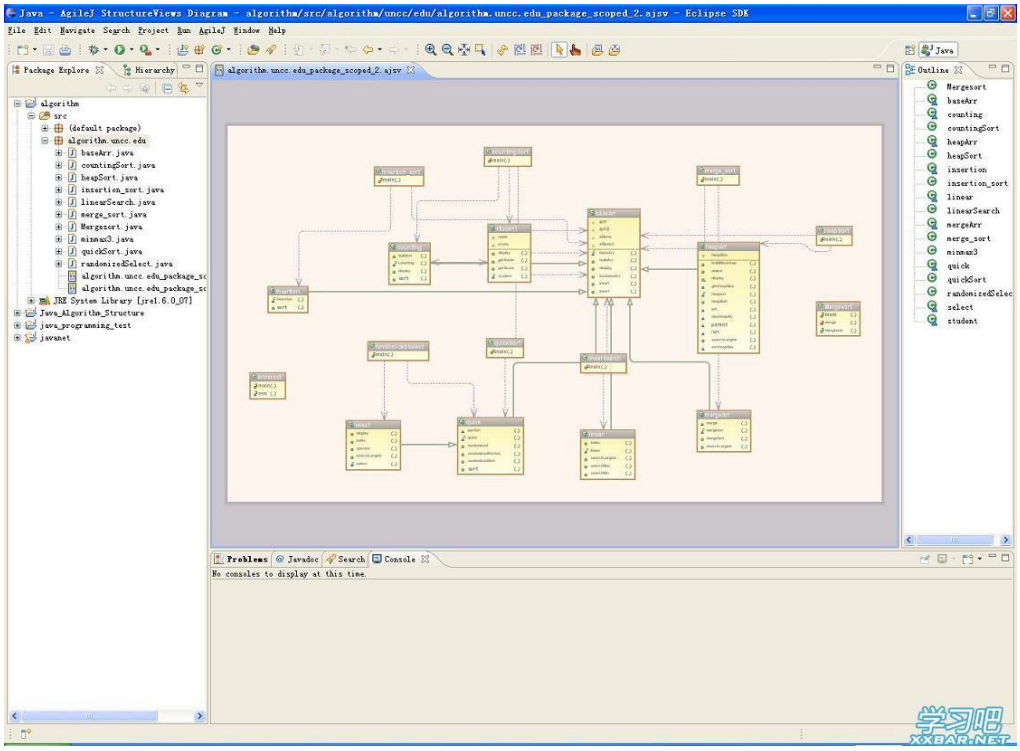
사용자가 소프트웨어 모델링을 수행하는 동안 범하게 되는 실수를 방지할 수 있도록 StarUML은 사용자가 개발한 소프트웨어 모델을 자동으로 검증(Verification)하여 사전에 오류 발생을 발견하게 만들어 준다. 따라서 사용자는 더욱 견고하고 완벽한 소프트웨어 설계를 수행할 수 있다.

7. 유용한 Add-In 제공

StarUML은 모델링한 정보의 재사용성, 생산성, 가용성, 상호 운용성을 높일 수 있는 유용한 Add-In을 빌트인(Build-In)으로 제공한다. 제공하는 Add-In으로는 프로그래밍 언어의 소스코드를 생성하거나 소스코드를 모델로 변환하는 기능을 제공하는 다수의 언어 Add-In들과 Rational Rose 파일 읽기, XMI를 통한 도구간 모델링 정보 교환, 그리고 디자인 패턴 지원 등이 있다.

B. AmaterasUML





AmaterasUML는 일본에서 만들어진 오픈소스 UML 툴로 Eclipse의 Plug-In으로 연결하여 사용한다. 설치가 용이하여 무료로 사용할 수 있어 특히 JAVA 환경의 개발자들이 많이 사용하고 있다. AmaterasUML을 사용하기 위해서는 EMF나 GEF, UML, EMTF 등이 Eclipse에 설치되어 있어야 한다.

i. 특징

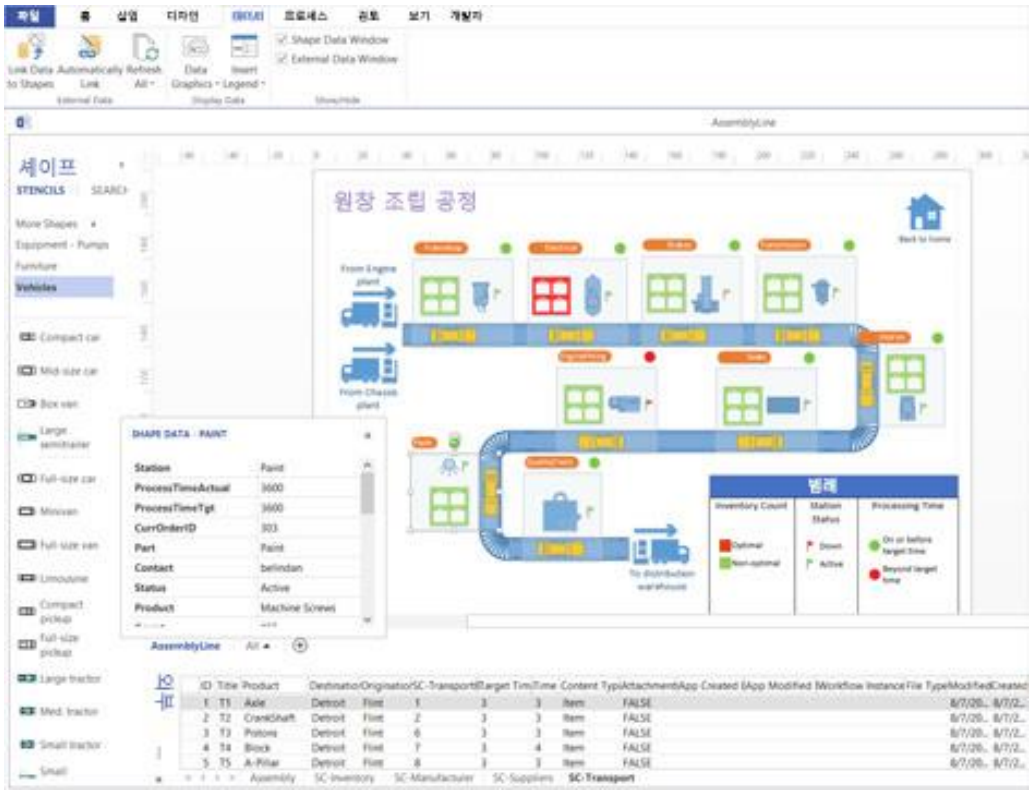
1. Eclipse 연동 기능

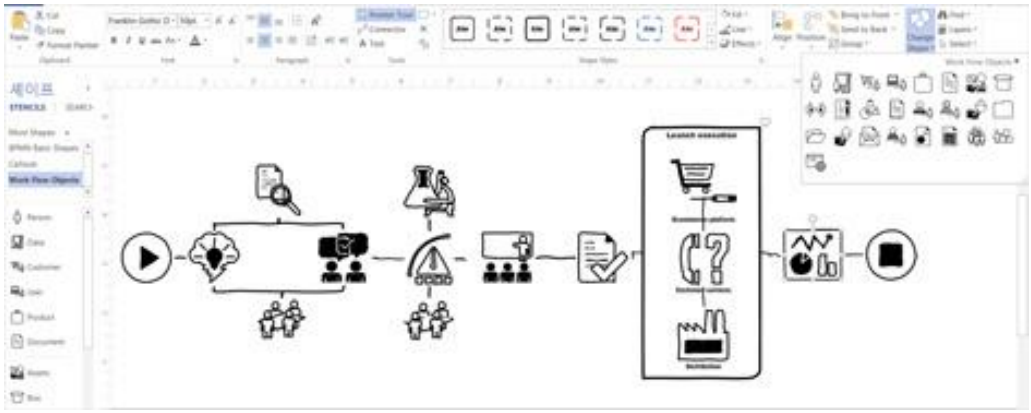
AmaterasUML의 가장 큰 장점은 Eclipse와 직접 연동되어 UML을 그린 후 해당 UML을 실제 실행 가능한 코드로 자동 생성해준다는 점이다. 이러한 기능은 개발자에게 편리함을 제공해주며 디자인한대로 코드를 작성할 수 있도록 도와주는 역할을 한다.

2. 낮은 안정성

AmaterasUML은 저장 시 문제가 발생하면 어떠한 경고 문구 없이 지금까지 작업한 Diagram이 사라지는 낮은 안정성을 보이고 있다. 특히 Diagram 수정 시 하나를 고치면 다른 변수의 이름이나 자료형이 마음대로 바뀌는 버그도 종종 발견된다.

C. MS Visio





MS Visio는 Microsoft의 유료 UML 툴로, 복잡한 정보를 단순화하여 전달하는 원스톱(One-Stop) 다이어그램 작성 솔루션이다. MS Visio는 도형을 이용하여 문자나 숫자만으로 곤란한 표현을 명확하고 간결하게 표시해주고, 효율적인 커뮤니케이션이 가능하도록 도와준다.

i. 특징

1. 빠른 다이어그램 작성

MS Visio는 재빨리 다이어그램 집합을 활용할 수 있도록 엄선된 다이어그램 집합을 사전에 제공하고 있다. 사전 제작되어 있는 다이어그램 집합은 UML 2.4와 BPMN 2.0, IEEE 규정 준수를 비롯하여 업계 표준을 충족하기 때문에 믿고 사용할 수 있다.

또한 상황에 맞는 팁과 요령을 사용하게끔 도와주어 사용자가 다이어그램을 쉽고 빠르게 작성할 수 있도록 도와준다. 특히 스마트 셰이프를 사용하여 생산성을 높이고, 새로운 테마와 효과를 활용하여 전문적인 다이어그램을 빠르게 완성할 수 있다.

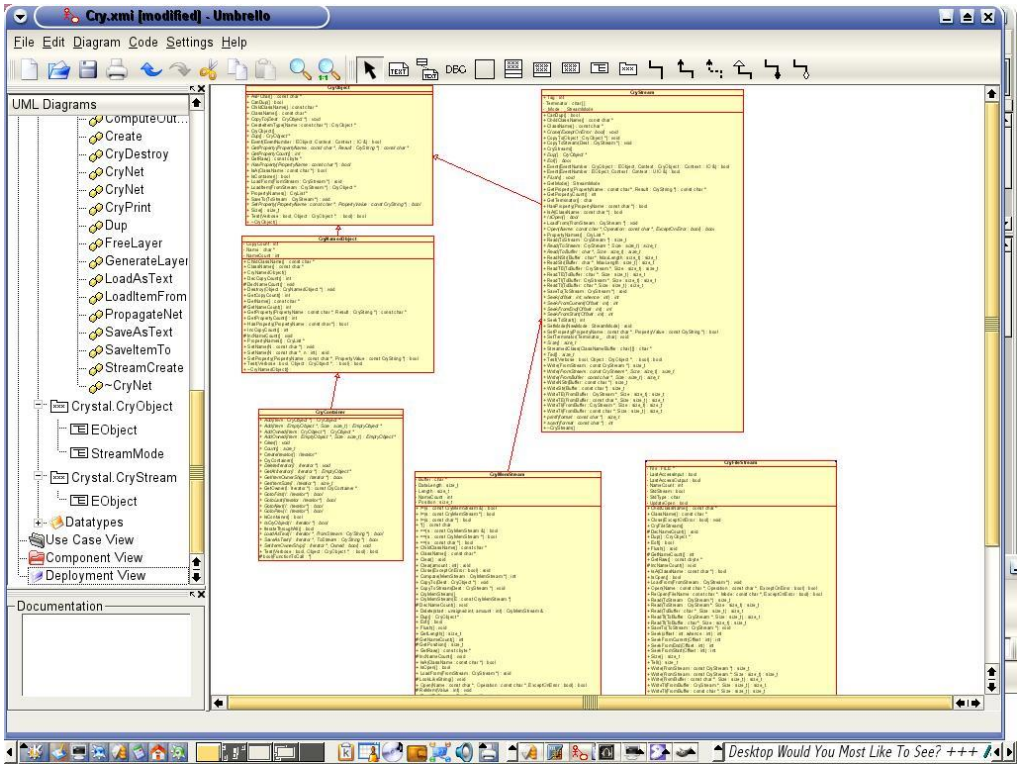
2. 데이터 연결 기능

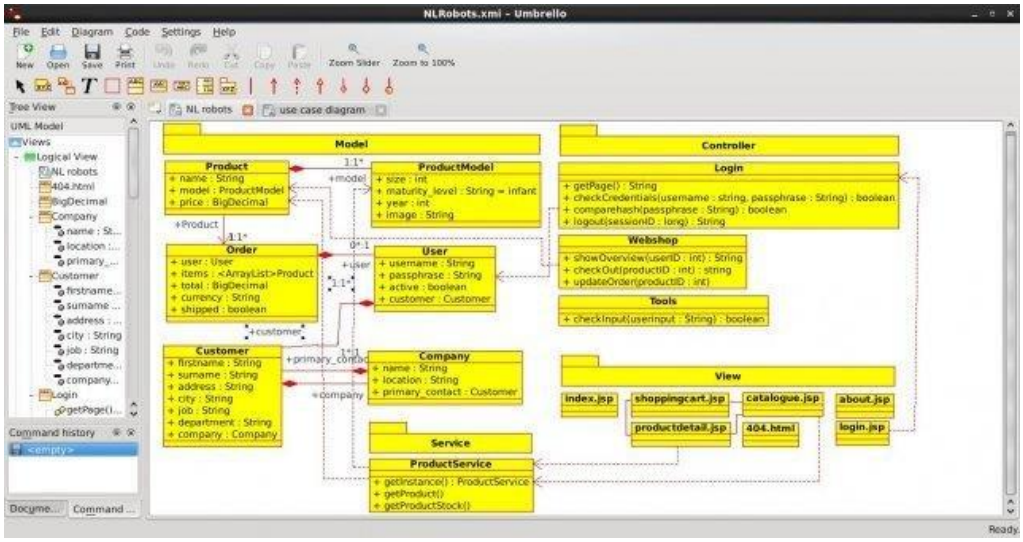
MS Visio를 사용하면 조직 구조 또는 IT 네트워크, 제조 설비, 복잡한 비즈니스 프로세스의 시각화에 데이터를 연결하여 성능을 한 눈에 알 수 있다. 사용자는 Microsoft Excel과 같은 데이터 원본에 MS Visio 셰이프를 연결하여 아이콘과 색, 텍스트 등의 데이터 그래픽을 사용할 수 있기 때문에 복잡한 정보를 간소하게 시각화할 수 있다.

3. 쉬운 공유

브라우저를 통해 Office 365 또는 SharePoint의 Visio Services를 사용하여 회사 전체에 순서도나 일정, 프로세스 맵, 조직 구조, IT 아키텍처, 평면도를 전달하고 공유할 수 있다. 만약 기본 데이터가 업데이트되면 브라우저의 다이어그램도 업데이트된다.

D. Umbrello





Umbrello는 Unix 계열 플랫폼에서뿐만 아니라 Microsoft Windows에서 기본적으로 사용 가능한 무료 UML 툴이다. 여러 데스크톱 및 프로그래밍 환경에서 잘 작동하는 특징을 가지고 있다.

i. 특징

1. 다양한 언어 지원

Umbrello는 모든 표준 UML 다이어그램의 종류를 처리한다. 이것은 C++이나 JAVA, IDL, Pascal, Delphi, Python 등으로 작성된 코드를 리버스 엔지니어링을 할 수 있을 뿐만 아니라 PHP나 펄코드 등 외부 도구로 생성된 XMI 파일을 불러오고 다양한 프로그래밍 언어로 내보낼 수 있는 기능을 지원한다.

2. 공동 개발 능력 극대화

Umbrello는 DocBook과 XHTML 형식으로 내보내게 함으로써 모델 컨텐츠의 전달을 가능하게 한다. 이것은 팀 구성원들이 Umbrello에 직접 액세스할 수 없거나 모델 컨텐츠가 웹사이트에 게재되어야 할 경우 공동 개발을 도와주는 큰 장점을 가지고 있다.

3. 적응력이 빠른 오픈소스 모델링 도구

오픈소스 프로젝트의 공통적인 특성이지만, Umbrello 또한 사용자가 곧 개발자이기 때문에 사용시 필요로 하는 기능들이 빠르게 추가된다는 사실을 알 수 있다. 필요에 의해 추가된 기능들은 전세계에 있는 사용자들로부터 사용되고, 쉽게 적응할 수 있다는 특징이 있다.

6. 레퍼런스

A. 도서

i. **UML ROSE RUP : 객체지향 분석설계 그리고 소설같은 실천 프로젝트 이야기**
서윤준 저 | 가남사 | 2004. 8. 15

ii. **LOG ON UML : 쉽게 배우는 UML과 객체지향 설계**
Jason T. Roff 저 | 이기오 역 | 사이텍미디어 | 2003. 7. 3

iii. **UML 모델링의 본질**
Kiminobu Kodama 저 | 김성훈 역 | 성안당 | 2005. 6. 1

B. 웹사이트

i. **OMG(Object Management Group**
http://www.omg.org/

ii. **StarUML**
http://staruml.io/

iii. **AmaterasUML**
http://amateras.osdn.jp/cgi-bin/fswiki_en/wiki.cgi?page=AmaterasUML

iv. **MS Visio**
https://products.office.com/ko-kr/visio/flowchart-software

v. **Umbrello**
https://umbrello.kde.org/