<Software Modeling & Analysis>

Introduction to UML

Team 1.

200911388 박미관

200911391 박준모

200911412 이영준

2013/03/22

Contents.

1 장 시스템과 시스템 언어

1. UML 정의

UML이란 소프트웨어시스템의 산출물을 가시화하고, 명세하고, 구축하고, 문서화하기 위한 그래픽 언어이다. UML은 소프트웨어 시스템을 모델링하기 위한 언어로 개발되었다. UML을 사용함으로써 소프트웨어 개발자들은 시스템에 대한 모델을 만들 수 있게 되었고, 이에 따라 소프트웨어 개발과 관련된 산출물들을 가시화하고 명세할 수 있게 되었다. UML은 모델링 수준에서 멈추지 않고 소프트웨어 설계와 구현에 관련된 사항들을 언어에 반영하여 모델을 설계와 구현에 사용 할 수 있도록 하였다.

2. UML의 구성

UML은 시스템 언어로, 시스템 구성 요소와 구성 요소들 사이의 연결 관계에 대한 개념들과 시스템 개발 과정에 필요한 개념들을 가져야 하며, 또한 이들 개념들과 개념들 사이의 관계를 표현할 수 있는 다이어그램을 가져야 한다.

개념은 시스템의 구조를 표현하는 것, 행위를 표현하는 것, 여러 개념들을 그룹화하는 것, 개념들에 대해 부가적인 설명을 하기 위한 것들이 있다. 구조는 UML에서 Use-case, Class등으로 표현되며, 행위를 표현하는 것은 Interaction, State Machine, 그룹화 하는 것에는 Package, 개념들을 부가적으로 설명하는 것에는 Note 등이 있다.

관계는 개념 실체들의 상호작용을 표현하는 연결 관계를 추상화한 것으로 의존(Dependency) 관계, 실현(Realization) 관계, 연관(Association) 관계, 일반화(Generalization)이 있다.

다이어그램은 시스템의 구조와 행위를 표현하고자 할 때 사용한다.

3. 시스템 정의

시스템이란 여러 개의 부분들이 모여 일련의 과정을 통하여 상호 작용하는 집단이다. 상호 작용하는 각 부분들이 합쳐져 새로운 성질을 보여주는 전체를 시스템이라고 한다. 전체가 부분의 합 이상이 되기 위해서는 부분들의 관계가 형성되어야 한다.

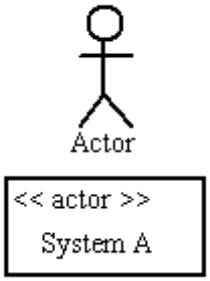
4. 시스템 특징

소프트웨어 시스템 개발은 존재하지 않는 시스템을 개발하는 것과 이미 존재하는 시스템을 변경하는 것으로 나누어진다. 소프트웨어 시스템을 개발하거나 변경하기 위해서는 지원하는 비즈니스 시스템에 대해서 알아야 하고, 또한 변경 시 에는 기존의 소프트웨어 시스템에 대해서도 알아야 한다. 따라서 소프트웨어 시스템과 비즈니스 시스템의 특징은 관찰과 개발의 관점에서 살펴보아야 한다.

관찰 관점에서의 시스템 특징

- 무엇인가를 하는 사용자가 존재한다
- 시스템은 사용자의 행위에 반응을 한다
- 시스템 내부에는 서로 다른 레벨을 갖고 서로 상호 작용하는 구성 요소들이 존재한다.

사용자 : 사용자들은 시스템을 사용해서 자신의 일을 처리하려는 목적을 가지고 있고, 그 목적은 시스템이 제공하는 기능에 의해서 달성된다. UML에서는 사용자의 역할을 액터라고 한다.

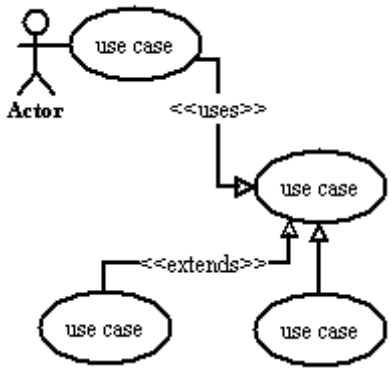


사용자의 예

시스템 기능 : 시스템은 사용자의 목적과 사용자가 원하는 방식에 따라 기능을 제공해야 하는 책임을 가진다. 시스템의 기능은 사용자의 관점에서는 제공되는 것으로 표현되고, 시스템의 관점에서는 사용자의 요구를 처리해야 하는 것으로 표현되어야 한다. UML에서는 사용자가 시스템을 사용해서 달성하고자 하는 목적을 위해 시스템이 제공하는 기능을 Use-case라고 하고 타원 기호로 표현한다. Use-case는 사용자 관점에서 작성 되어야 하기 때문에 서비스 관점에서 기술 되어야 한다.

Use-case의 예

사용자와 시스템 기능 : UML은 어떤 사용자의 역할과 어떤 시스템의 기능이 관련되는지를 유스케이스 다이어그램에 액터와 유스케이스의 관계로 표현한다.



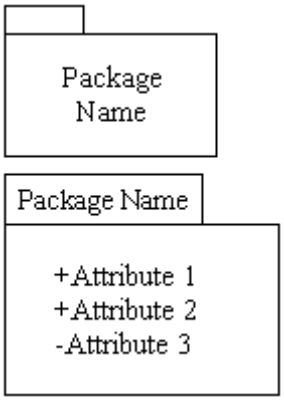
액터와 유스케이스의 관계의 예

액터가 시스템 사용 목적을 달성하기 위해서 시스템을 사용하는 과정을 사용 시나리오라고 한다. 액터가 시스템을 사용하는 방법에 따라 사용 시나리오가 달라지기 대문에 액터의 목적에 대한 여러 시나리오들은 유스케이스 이벤트 플로(Event flow)로 추상화되고 하나의 기술서로 작성된다.

시스템 구성 요소 : 시스템 내부의 구성 요소들이 시스템이 제공하는 기능을 실현하기 위해서 서로 상호작용을 통해 협력한다. 구성요소들은 서로 서비스를 주고받기 위해 상호 작용한다. 시스템 구성 요소들은 시스템의 기능 실현에 참여하기 위해 존재하는 것으로 협력에 참여하지 않는 구성요소는 존재 할 수 없다. UML에서는 사용자가 시스템을 사용해서 달성하고자 하는 목적을 역할들의 협력으로 표현하는 것이 컬레보레이션(Collaboration)이다.

UML에서는 역할들에 해당하는 구성 요소들이 협력을 위해 메시지를 전달하면서 상호작용하는 것을 인터랙션(interaction)이라고 하고, 시퀀스 다이어그램, 커뮤니케이션 다이어그램 등으로 나타낸다.

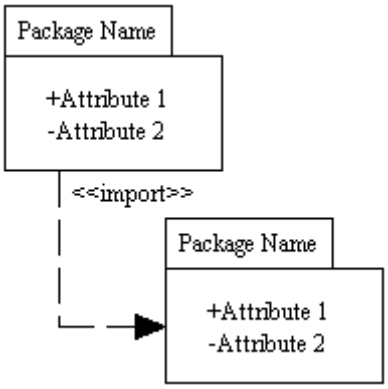
구성 요소들은 포함 개념의 추상 수준에 따라 계층화 됩니다. 계층화에서 하위 계층에 속한 구성 요소들은 그룹화 되어 상위 계층을 구성한다. UML에서는 구성요소들을 그룹화하는 것을 패키지라고 하고 패키지 다이어그램으로 표현한다.



패키지의 예

그룹들은 자신의 책임을 수행하기 위해 다른 그룹의 서비스를 필요로 하기도 한다. 이들

그룹은 서비스를 제공하는 그룹에 의존하기 때문에 서비스를 제공하는 그룹의 변화에 영향을 받게 된다. UML에서는 그룹들 간의 의존을 의존 관계로 표현한다.



의존 관계의 예

그룹 간에 그룹의 구성 요소들이 다른 그룹에서 이용 할 수 있는 지를 가시성(Visiblity)으로 표현한다. 다른 그룹이 접근할 수 있는 요소들에 대해서는 가시성을 public으로, 접근할 수 없는 요소들은 private로 설정한다.

가시성의 예

구성 요소에는 능동적으로 작업을 수행해서 사용자가 원하는 서비스를 제공하는 것과 그러한 작업을 위해 사용되는 수동적인 구성 요소가 있다. 능동적인 구성 요소에는 사용자와의 상호 작용을 지원하는 것과 실제 작업을 처리하는 것이 있다. 소프트웨어 시스템에서 사용자와의 상호 작용을 지원하는 능동적인 구성 요소를 바운더리(Boundary)라고 하고, 실제 작업을 처리하는 능동적인 구성요소를 컨트롤(Control)이라 하며, 수동 요소를 엔터티(Entity)라고 한다. 소프트웨어 시스템의 바운더리는 화면이나 시스템 인터페이스 혹은 장치 인터페이스이며 수동 요소는 정보들이다.

시스템 구성요소의 책임 : UML에서는 시스템 구성요소들의 역할에 따른 업무 처리를 액티비티 다이어그램을 사용하여 이들을 기술서로 가시화한다. 또한 상태 머신으로 시스템 구성 요소들의 상태를 명세하여 상태 머신 다이어그램으로 표현한다.

구성 요소의 배치 : 시스템 구성 요소들은 조직에 배치되고, 조직이 운영되는 데에 필요한 공간과 자원들을 필요로 한다. 소프트웨어 시스템의 조직에게는 조직이 위치하는 하드웨어 시스템과 조직이 운영되기 위한 프로세스가 할당된다. UML에서는 배치 다이어그램을 시스템 구성 요소들의 배치를 표현한다.

개발 관점에서의 시스템 특징

개발 관점에서의 시스템 특징에서는 관찰 관점에서 볼 수 있었던 특징들을 어떻게 개발할 것인가를 중점적으로 알아본다.

이해 관계자 : 기존에 존재하던 시스템의 사용자들은 직접적으로 시스템을 관찰함으로써 발견할 수 있다. 새로 개발되는 시스템의 사용자를 찾는 방법 중 하나는 이해 관계자를 찾는 것이다. 이해 관계자란 시스템 개발에 의해 영향을 받는 사람 또는 다른 시스템이다. 이해 관계자들은 시스템 개발에 영향을 받기 때문에 시스템 개발에 많은 관심을 갖게 된다.
이해관계 목적과 시점 : 이해 관계자들은 목적을 달성하기 위해 여러 번 시스템과 직접 또는 간접적으로 상호 작용을 원하는 시점들이 있다. 이해 관계자의 목적과 시점은 시스템의 책임을 좀더 넓은 범위에서 볼 수 있도록 하며, 사용자가 시스템을 이용하는 목적을 찾을 때에도 사용된다.
시스템 제공 기능 : 시스템은 시스템 사용자의 목적 달성을 위해 존재하기 때문에 시스템 사용자의 시스템 사용 목적을 기준으로 시스템이 제공해야 하는 기능을 명세한다.
컬레보레이션 : 시스템 구성 요소들은 시스템 책임을 수행하기 위한 협력에 참여하기 위해 존재하기 때문에 시스템 구성 요소들을 명세하기 위해서는 시스템 책임 수행을 위해 필요한 협력들이 무엇이고, 협력을 위해 필요한 역할들이 무엇인지를 알아야 한다.
인터랙션 : 협력에 참여하는 역할들은 서비스를 요청하고 제공하면서 상호 작용하기 때문에 역할들이 제공하는 서비스들을 명세하기 위해서는 유스케이스 이벤트 플로어를 통해 인터렉션을 작성해야 한다.
액티비티 : 각각의 역할은 수행해야 하는 책임이 구분된 역할 타입들에 의해서 찾을 수 있기 때문에 각각의 역할 타입들이 해야 하는 구체적인 책임은 역할 타입들이 해야 하는 책임의 종류에 따라 찾으면 된다.
시스템 구성 요소 : 시스템 구성 요소들은 시스템 책임 수행을 위한 협력에 필요한 역할을 수행하기 위해 인터랙션에 참여한다. 따라서 시스템 구성 요소들을 설계 할 때는 이러한 역할들에 명세된 서비스들을 제공하도록 해야 된다. 소프트웨어 시스템의 경우 객체지향 개발 방법이나 컴포넌트기반 개발 방법과 같이 적용되는 개발 방법에 따라 구성 요소가 달라진다. 객체지향 개발 방법에서 구성 요소는 객체이고 설계도는 클래스가 된다. 컴포넌트 기반 개발 방법의 경우 구성 요소는 실행 컴포넌트이고 설계도는 설계 컴포넌트이다.
시스템 구성 요소의 상태 : 시스템 구성 요소들의 책임 수행은 구성 요소들의 상태에 따라 달라지기 때문에 시스템 구성 요소들의 생애 주기 동안의 상태 변화를 찾고 명세해야 한다. 소프트에어 시스템의 컨트롤은 능동적인 구성요소로서 이들의 상태는 이전까지 수행한 행위가 무엇인지에 대해 영향을 받는다. 엔티티는 속성 값의 변화와 다른 자원이나 엔티티와의 연결이 만들어지거나 삭제될 때 상태가 변한다. UML에서는 객체 다이어그램을 사용해서 구성 요소들의 속성 값과 연결 관계의 변화를 표현할 수 있다.
조직화 : 시스템 구성 요소들을 효율적으로 관리하기 위해서는 시스템 구성 요소들을 조직화할 필요가 있다. 시스템 조직은 핵심 조직과 지원 조직으로 나누어 지기 때문에 컬

레보레이션을 기준으로 핵심 조직을 구성하고 이들을 지원하기 위한 지원 조직을 구성한다.
시뮬레이션과 테스트 : 개발 과정에서 발생할 수 있는 결함을 줄이고 품질을 높이기 위해서 지속적으로 시스템에 대한 단위 테스트와 통합 테스트가 병합되어야 한다.
구성 요소의 배치 : 시스템 구성 요소들의 조직에게 필요한 공간과 자원을 할당하기 위해서는 시스템 사용자의 위치, 조직에게 요구되는 책임 수행 정도, 책임 수행에 요구되는 처리 시간이 고려 되어야 한다.

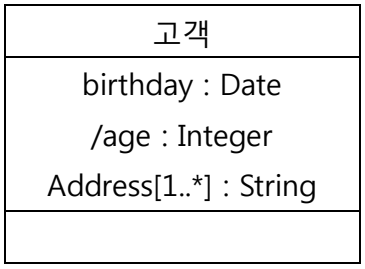
2장 공통 메커니즘

1.분류자

시스템은 추상적인 개념들에 의해서가 아니라 실체들에 의해서 운영된다. 시스템의 운영을 위해 요구되는 실체들은 매우 방대하기 때문에 이들을 개별적으로 정의하는 것은 불가능하다. UML에서는 설계도를 통해 만들어진 실체들을 인스턴스라고 하고, 실체들을 분류한다는 의미에서 설계도를 분류자라고 한다.

2. 분류자와 인스턴스

분류자는 구조적 특징과 행위적 특징을 기술하는 메커니즘이다. 분류자의 종류에는 액터, 유스케이스, 인터페이스, 언터렉션, 클래스, 데이터 타입, 상태 머신 등이 있다.



분류자의 예

분류자의 특징 : 분류자의 구조적 특징을 속성(Attribute)이라고 하고, 행위적 특징을 오퍼레이션(Operation)이라고 한다. 분류자는 속성과 자신의 인스턴스에 연결되는 인스턴스들에 대한 참조자에 대한 명세를 합쳐 특성(Property)라고 한다. UML에서 인스턴스들 사이의 연걸은 링크(Link)라고 하고 링크에 대한 명세는 연관(Association)관계라고 한다.

분류자의 속성은 다음과 같은 형식으로 작성된다. 대괄호로 작성된 것은 선택적으로 작성할 수 있다는 것을 의미한다.

[가시성][/] 속성 이름 [: 타입][다중성] [= 기본값] [{특성 문자열}]

분류자의 오퍼레이션은 다음과 같은 형식으로 작성된다.

[가시성] 오퍼레이션 이름 ([매개변수 리스트]) [: 리턴 타입[다중성]] [{특성 문자열}]

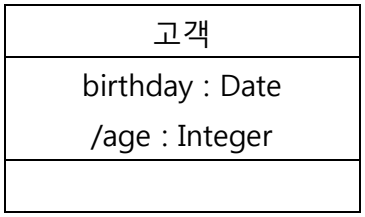
오퍼레이션의 매개변수 리스트는 다음과 같은 형식으로 작성된다.

[입출력 방향] 매개변수 이름 : 타입[다중성][=기본값][{특성문자열}]

가시성 : 가시성은 분류자의 특징을 다른 분류자가 접근할 수 있는지의 여부를 표현하는 것이다.가시성의 종류는 public, protected, private, package가 있다. Public은 다른 분류자가 아무런 제약 없이 특징에 접근할 수 있도록 하고, protected는 상속 관계에서 자식에 해당하는 분류자만이 해당 특징에 접근할 수 있도록 한다. Private는 어떠한 분류자도 접근 할 수 없도록 한다.

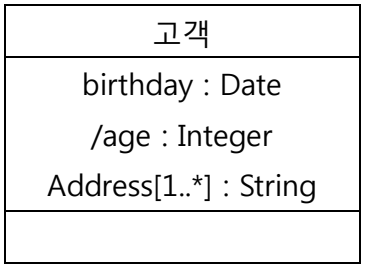
파생 특성 : 파생 특성은 다른 특성 값으로부터 자신의 값을 얻을 수 있는 것을 특성 이

름 앞에 '/' 기호로 나타낸다.



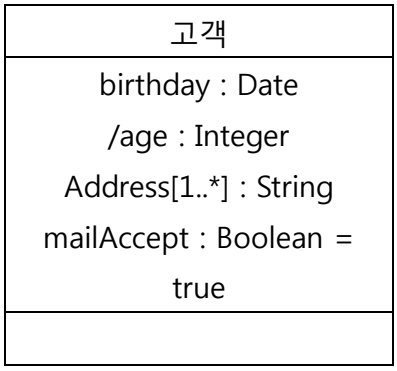
파생 속성 표현

다중성 : 분류자의 특성은 한 개의 값을 갖는 게 일반적이지만 여러 개의 값을 갖는 것도 있기 때문에 특성이 가질 수 있는 값의 범위를 규정해야 한다. UML에서는 이를 다중성이라고 한다.



속성의 다중성 표현

타입과 기본값 : 타입에는 속성의 타입에 대한 분류자를 작성한다. 기본값은 특별한 값을 성정하지 않는 경우 인스턴스가 생성될 때 기본적으로 설정되는 값이다.



기본값 표현

특성 문자열 : 특성 문자열은 분류자의 특성에 추가적인 제약을 작성하는 것이다. 특성 문자열에는 {readOnly}, {Ordered}, {Bag}, {Sequence} 등이 있다.

적용 범위 : 적용 범위는 분류자의 속성값 또는 오퍼레이션이 분류자의 범위에서 모든 인스턴스에게 적용될 것인지, 개개 인스턴스마다 특성 값을 설정할 것인지를 결정하는 것이다.

3. 일반화

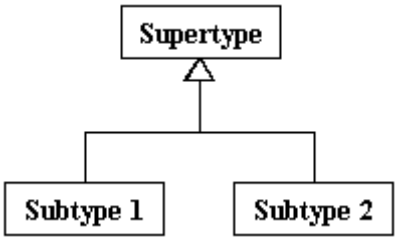
분류자는 인스턴스를 만들어 내고 분류하기 위한 설계도의 개념 외에도 개념들을 추상화 수준에 따라 계층화 한다는 개념을 가지고 있습니다.

일반화 계층과 분류자의 특징들과의 관계

추상화는 어떤 대상에 대해서 특별한 목적을 위해 필요로 하는 핵심적인 특징만을 나타내는 것이고, 인스턴스들은 뚜렷한 공통된 특징에 의해 같은 종류의 개념으로 분리된다. 따라서 같은 추상화 수준에 속하는 같은 종류의 인스턴스들이라고 하는 것은 인스턴스들이 같은 수의 뚜렷한 공통된 특징들을 갖는다는 것을 의미한다.
공통된 특징 개수와의 관계 : 개념들의 추상화 수준을 높이는 것은 일반화 한다고 하고, 추상화 수준을 낮추는 것을 구체화한다고 한다. 개념들을 점점 일반화하게 되면 공통된 특징의 수가 줄어들고, 점점 구체화하면 공통된 특징의 수가 늘어난다는 것을 알 수 있다. 추상화 수준이 올라갈수록 더 많은 인스턴스들을 동일한 개념으로 묶을 수 있게 된다.
행위적 특징 내용과의 관계 : 추상화 계층에 따라 인스턴스들을 분류할 때 주의해야 할 것은 동일한 행위라도 행위의 내용이 다른 인스턴스들의 집합이 존재한다는 것이다. 행위의 내용을 추상화 수준에 따라 계층적으로 분류할 때에도 공통된 특징의 수와 마찬가지로 추상화 수준이 올라갈수록 행위 수행의 공통적인 면이 적어진다는 것을 의미한다.

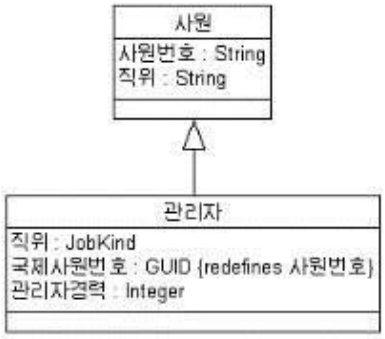
상속

추상화 수준에 따라 계층화된 분류자들에서는 하위 수준 분류자가 상위 수준 분류자의 공통된 특징들을 갖게 된다. 따라서 추상화 수준마다 공통된 특징들을 기술하는 것은 중복 작업을 발생시킨다. 이러한 분류 기준을 반복적으로 기술하는 문제를 해결하기 위한 개념이 상속이다. 상속 관계를 갖는 개념들은 하위 개념이 상위 개념의 구조적 특징과 행위적 특징을 모두 물려 받는다.
UML에서는 직접적으로 상속 관계라는 용어를 사용하지 않고 대신 일반화 관계라는 용어를 사용한다.



일반화 관계 표현

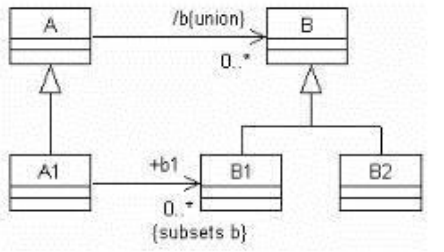
{redefines} : 하위개념에서 상위개념을 재정의할 때 redefines라는 키워드를 사용해서 제약사항을 작성한다.



redefines의 표현

{union}, {subsets} : 연관에도 일반화가 작성된다. 하위개념들 사이에 작성된 연관과 상위 개념들 사이에 작성된 연관이 일반화 관계를 갖으면 연관되는 실체들 사이에는 합집합과 부분집합의 관계가 만들어진다.

이러한 경우에 합집합에 union이라는 키워드를 부분집합에 subset이라는 키워드를 사용해서 제약사항으로 작성한다



Union과 subset의 표현

추상

추상화에 의해 개념을 계속적으로 일반화 하다 보면 인스턴스들을 만들어내는 역할보다는 단지 계층을 분류하는 역할만하는 분류자들이 나타나게 되는데, 이처럼 인스턴스를 만들지 않고 단지 곛ㅇ 분류의 역할만을 하는 분류자들을 추상 분류자라고 한다.

상위분류자가 하위 분류자들의 행위 수행 내용 중 동일 부분을 취해 자신의 행위에 정의함으로써 하위 분류자들에 대한 행위 수행 내용을 중복 작성하는 것을 막을 수 있다. 상위 분류자에 정의한 행위 수행 내용은 하위 분류자들의 행위 일부로서 불환전한 형태이기 때문에 실행은 할 수 없다. 이러한 행위적 특징을 추상 오퍼레이션이라고 한다.

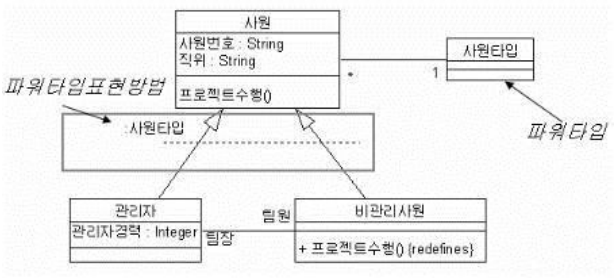
추상 분류자와 추상 오퍼레이션과 같이 불완전한 명세로 인스턴스 될 수 없는 것들을 통틀어 추상이라고 한다.

파워 타입

복잡한 계층에 이해 추상화된 분류자들은 그 분류 기준을 명시하는 것이 중요하다. UML에서는 이러한 분류 기준을 파워 타입이라고 한다.

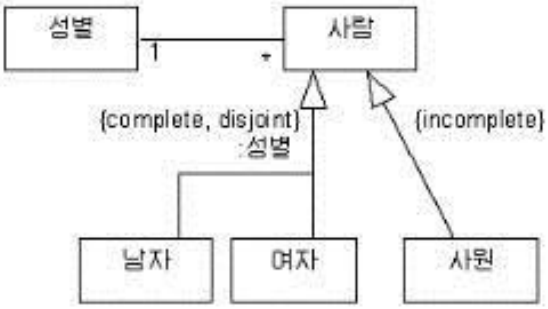
파워타입은 상위개념과 연관을 갖는 개념표현으로 작성한다. 파워타입이 적용되는 것(인

스턴스되는 것)은 상위개념과 하위개념들 사이에 점선으로 구분된 선을 작성하고 '파워 타입 인스턴스 이름 : 파워타입 이름'을 작성한다.



파워 타입 표현

점선을 작성하는 방법은 표현도 복잡하게 하고, 보기도 좋지 못하다. 이에 대한 좀 더 나은 표현은 일반화 기호를 공유하도록 하는 것이다.



공유된 일반화 표현

제약 사항 : 일반화에 대한 제약사항은 {complete}, {incomplete}, {disjoint}, {overlapping}이 있다. 동일 일반화 계층에서 하위 계층 개념들이 상위 계층 개념의 모든 실체들을 포함할 수 있다면 {complete} 되고, 그렇지 않다면 {incomplete} 된다. 동일 일반화 계층에서 상위 계층 개념의 실체가 반드시 하위 개념들 중 하나의 개념에만 해당한다면 {disjoint}가 되고, 그렇지 않다면 {overlapping}이 된다.

명세와 실현 : 분류자가 다양한 방법에 의해 구현되기 위해서는 명세와 구현이 분리되어야 한다. 구현 방법을 정의하지 않고, 명세의 역할만을 하는 분류자를 나타내기 위해서 <<specification>>이라는 스테레오 타입을 사용하고 명세를 구현하는 클래스를 나타내기 위해서 <<realization>>이라는 스테레오 타입을 사용한다.

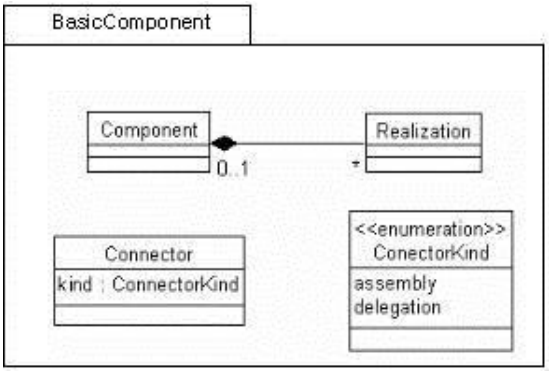
4. 조직화 메커니즘

조직화 메커니즘은 시스템 구성 요소 그룹과 시스템 모델 요소 그룹의 두 가지 관점에서 바라보아야 한다. 시스템 구성 요소 그룹 관점은 시스템의 설계, 구현, 운영이라는 관점에서 객체와 클래스를 조직화 하기 위한 도구로 패키지를 바라본다. 시스템 모델 요소 그룹 관점에서는 시스템 개발 과정에서 만들어지는 모델 요소들과 그들을 표현하는 다이어그램들을 조직화하기 위한 도구로 패키지를 바라본다.

5. 패키지

UML에서는 구성 요소 그룹을 패키지로 표현하고, 패키지에 포함되는 구성 요소들은 멤버라고 부른다.

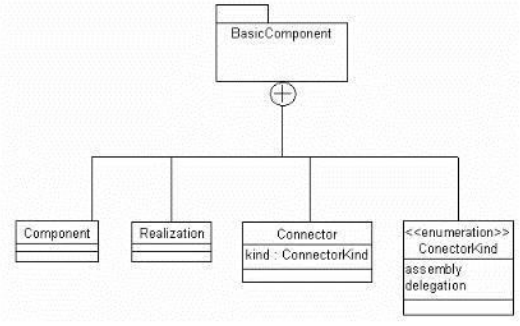
패키지 표현 : 패키지는 이름이 작성되는 탭(헤더)과 패키지되는 모델요소들(패키지 멤버)이 작성되는 사각형 영역(바디)으로 작성된다. 패키지 기호는 탭이 달린 폴더라고 불리기도 한다. 패키지 멤버들을 보여주지 않을 때는 패키지 이름을 패키지 바디에 작성한다.



패키지 표현

멤버들을 보여주지 않을 때의 패키지 표현

패키지 멤버들은 패키지의 바디에 표현할 수도 있지만, 좀더 간단히 표현하기 위해서 패키지 외부에 트리구조로 표현할 수도 있다.



트리 구조로 표현한 패키지

네임스페이스 : 패키지에 포함되는 모델요소들은 고유한 이름을 갖는 요소들이다. 패키지 또한 고유한 이름을 갖는 요소이다. 이름이 고유하다는 것은 이름 공간내에서 중복된 이름이 없다는 것이다. 패키지와 같이 고유한 이름을 갖는 모델요소들을 포함하는 것들은 이름공간(name space)이 되어주어야 한다.

패키지 내에서는 패키지 멤버를 가르킬 때는 모델요소의 이름 만으로 가능하다. 하지만 패키지 밖에서는 중복된 이름이 있을 수 있기 때문에 패키지 밖에서 특정 패키지의 패키지 멤버를 가르킬 때는 패키지 이름을 함께 사용해야 한다. 이름공간과 이름은 '::'를 사

용해서 구분한다. 이름공간과 이름을 결합한 이름을 full name 또는 qualified name이라 한다.

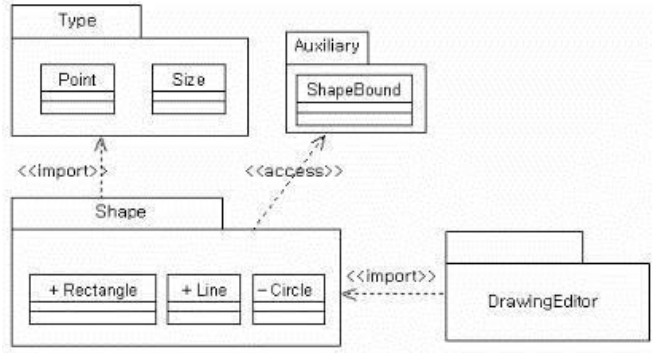
서로 다른 패키지의 패키지 멤버들끼리 상호작용을 관리하기 위해서는 어떤 멤버는 외부에서 접근할 수 있고, 어떤 멤버는 접근하지 못하도록 하는 접근성에 제한을 두어야 한다. UML은 가시성을 통해 접근성을 제약한다. 다른 패키지의 멤버들의 접근을 허용할 경우는 public으로, 접근을 허용하지 않을 경우는private으로 가시성을 설정한다.

6. 패키지 관계

패키지 멤버가 다른 패키지 멤버에 접근하기 위해서는 먼저 그룹차원인 패키지들 사이에 관계가 형성이 되어야 한다. 이들 관계는 의존관계 기호로 표현한다. 의존관계에는 패키지 임포트 관계와 패키지 머지 관계라는 좀더 특수한 의존관계가 존재한다.

패키지 임포트 관계

패키지임포트는 <<import>>와 <<access>>라는 두 개의 키워드를 가지고 표현된다. <<import>>는 키워드와 같이 다른 패키지의 멤버를 자신의 멤버처럼 사용하기 위해서 가져 온다는 의미를 갖는다. 자신의 멤버처럼 가져와서 사용하기 때문에 가져온 멤버에 대해서 멤버의 네임스페이스를 포함한 이름으로 접근하지 않아도 되고, 자신을 임포트하는 다른 패키지에서도 가져온 멤버의 이름을 직접 사용할 수 있다는 것을 의미한다.



패키지 임포트 관계 표현

<<access>>는 키워드와 같이 다른 패키지의 멤버에 단지 접근만 한다는 것이다. 따라서 자신을 직접적으로 액세스한 패키지는 멤버에 접근할 때 네임스페이스를 사용하지 않아도 되지만, 자신을 임포트하는 다른 패키지는 액세스한 멤버들에 대해서 네임스페이스를 사용하지 않고는 접근할 수 없다.

패키지 머지 관계

직접적으로 다른 패키지에 명세되어 있는 멤버들을 가지고, 새로운 패키지를 재구성하기 위해 사용하는 관계를 패키지 머지 관계라고 부르고, <<merge>> 키워드를 갖는 의존으로 표현한다. <<merge>>는 키워드의 의미대로 기존의 패키지들의 멤버들을 병합해서 새로운 자신의 패키지를 구성한다는 것을 의미한다.

패키지 머지 관계

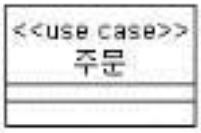
7. 확장 메커니즘

UML은 스테레오 타입, 태그를 갖는 값, 제약 사항과 같은 세 가지 방법에 의해 확장된다.

8. 스테레오 타입

스테레오타입은 기존의 모델요소와 유사하나 정확히 같은 의미를 가지지 않을 때 기존의 모델요소를 기반으로 UML의 어휘를 확장하는 방법이다.

스테레오타입 표현 : 스테레오타입은 '<<>>'기호 안에 확장하려는 어휘를 작성하는 방법으로 표현된다.

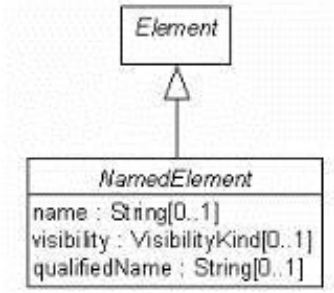


스테레오 타입 표현

스테레오 타입과 UML 버전 : 이전 UML 버전에서 새로운 UML 버전으로 갱신될 때 중요하게 다루어지는 것들 중 하나가 스테레오타입이다. 발표된 버전을 사용하면서 UML 사용자들은 많은 스테레오타입을 만들어 사용하게 되고, 이러한 스테레오 타입 중에는 일반화되는 것들도 있다. UML 사용자에 의해 만들어져 사용되는 스테레오타입을 사용자정의 스테레오타입이라고 한다. 새로운 UML 버전을 만들 때, 일반화된 스테레오타입들은 표준화된 스테레오타입으로 UML에 명세된다.

9. 태그를 갖는 값

UML에는 name, visibility, qualifiedName과 같은 속성을 갖는 NamedElement라는 클래스가 명세되어 있다. UML 모델요소들 중에서 유스케이스나 클래스나 패키지와 같이 이름을 갖는 모델요소들은 모두 이 클래스를 상속받는다.



NamedElement

스테레오 타입과 태그를 갖는 값 : 기존의 모델요소를 추가적인 의미를 가지고 확장해서 스테레오타입을 만든다는 것은 기본 요소에 대해 새로운 속성이 추가되거나 새로운 제약사항이 존재해야 한다는 것을 의미한다. 따라서 사용자정의 스테레오타입을 추가하고자 할 때는 그와 관련된 태그를 갖는 값들과 제약사항들의 집합을 함께 정의해야 한다.

10 . 제약 사항

제약사항은 모델요소에 추가적인 규칙을 부여 할 때 사용하며, 중괄호 안에 제약사항을 작성해서 표현한다. 제약사항은 일반적으로 노트 안에 작성된다. UML에는 표준으로 정해진 키워드를 갖는 제약사항들이 있다. 이들 표준 제약사항들은 모델요소에 직접적으로 표현된다.

11. 프로파일

확장모델요소들을 재사용하기 위해 하나의 그룹으로 묶어서 정의하는 것을 프로파일이라고 하고, 도구지원을 받을 수 있도록 확장관계에 대한 표현을 추상구문과 객체제약언어를 사용해서 명세한다.

3 장 유스케이스

1. 액터

고객의 조직 구성원들은 직접적으로 개발될 시스템을 사용하기도 하고, 자신들의 기존 시스템을 사용하므로 간접적으로 개발될 시스템을 사용하기도 합니다. 시스템의 입장에서는 사람이든 다른 시스템이든 시스템을 사용하는 대상은 모두 사용자입니다.

액터 정의

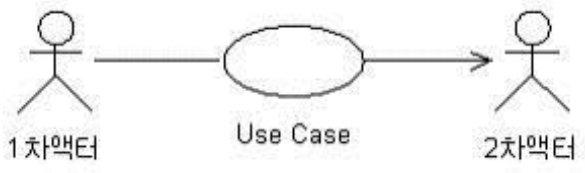
UML에서는 이러한 사용자의 역할을 액터로 표현합니다. 액터란 시스템과 상호 작용해야 하는 어떤 사람 또는 어떤 것이라고 정의합니다. 액터는 개개 사용자를 모두 표현하는 것이 아니라 사용자들의 시스템 사용 역할만을 표현하게 되므로 사용자 분석 작업을 단순하게 만듭니다.

액터 표현

액터는 막대사람 아이콘 아래에 액터 이름을 작성합니다.

액터의 표현

1차 액터(Primary Actors)와 2차 액터(Secondary Actors) : 액터가 유스케이스 사용을 마치는데 또 다른 액터의 도움이 필요할 때가 있습니다. 1차라는 것은 그 기능을 사용하는 메인이고, 2차는 보조적인 역할입니다. 일반적으로 1차 액터를 유스케이스 오른쪽에, 2차 액터를 유스케이스 왼쪽에 위치시킵니다. 좀 더 명확히 하고자 할 때, 유스케이스에서 2차 액터로의 화살표를 작성하기도 합니다.



1차 액터와 2차 액터의 표현

2. 유스케이스

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

유스케이스의 정의

유스케이스란 개발될 시스템의 개개 액터가 시스템 사용 목적을 잘 달성할 수 있도록 개발될 시스템이 제공해야 하는 서비스이다. 유스케이스는 타원 아이콘의 아래에 유스케이

스 이름을 작성합니다.

유스케이스의 표현

유스케이스 이름

유스케이스 정의를 통해 우리는 유스케이스가 액터의 시스템 사용 목적 단위로 작성 되어야 함을 알 수 있습니다. 따라서 유스케이스는 액터의 시스템 사용 목적으로 식별하고, 이름은 액터의 시스템 사용 목적에 서비스를 붙여서 작성합니다.

유스케이스와 시나리오

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

이벤트 플로어

액터의 시스템 사용 시나리오를 잘 작성하기 위해서는 시나리오 작성 방법에 대해 알아야 하며 그러기 위해서는 액터와 시스템의 역할과 책임에 대해서 알아야한다.

액터의 책임

- 시스템에게 서비스 수행을 시작하도록 요청해야 합니다.
- 시스템이 요구하는 정보를 제공해야 합니다.
- 시스템과의 상호 작용과 관련된 의사결정을 한다.

시스템의 책임

- 액터로부터 제공받은 정보나 시스템의 서비스 수행 상태를 기록해야 합니다.
- 서비스를 시작하거나 시스템이 요청한 행위를 수행하는데 액터가 필요로 하는 정보를 제공해야 합니다.
- 서비스를 제공해야 합니다.

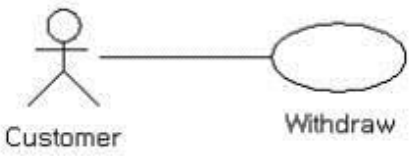
액터의 시스템 사용 목적을 달성하기 위해 액터와 시스템에 의해 수행되는 행위들의 흐름은 결국 사건의 흐름에 따라 진행됩니다. 사건의 흐름을 영어로는 이벤트 플로어(Event Flow)라고 한다.

3. 관계

액터와 시스템 같이 목적 달성을 위해 고안되고 시도되는 커뮤니케이션을 목적 지향적인 커뮤니케이션이라고 합니다. 즉 액터와 시스템은 목적 지향적인 커뮤니케이션을 하고 있는 것입니다.

액터와 유스케이스 관계

액터와 유스케이스 사이에 작성되는 연관(association)은 액터가 시스템의 어떤 기능을 사용하기 위해서 시스템과 상호작용하는가를 나타냅니다. 액터와 유스케이스 사이에 실선으로 작성됩니다.



액터와 유스케이스 관계 표현

액터와 액터 관계

액터들 사이에는 단지 일반화 관계만이 존재합니다. 액터의 일반화 관계가 의미하는 것은 하위 액터가 상위 액터의 모든 역할과 책임을 상속받는 것을 의미합니다.

유스케이스와 유스케이스 관계

유스케이스와 유스케이스 사이에는 추가적으로 포함(include)과 확장(extend)이 작성됩니다. 공통부분에 해당하는 유스케이스와 이를 포함하는 유스케이스의 관계에는 include, 확장되는 유스케이스와 확장하는 유스케이스의 관계에는 extend 키워드를 갖는 의존 관계 기호로 표현합니다.

4. 액터와 관련된 문제들

조직의 역할이나 직위로 작성된 액터

조직의 역할로 액터를 작성했을 때 발생하는 문제는 액터와 유스케이스 관계가 매우 복잡하게 작성된다는 것이다. 액터는 조직의 역할이 아닌 시스템 사용 역할로 작성해야 합니다.

5. 유스케이스와 관련된 문제들

액터가 없는 유스케이스

시스템이 존재하는 이유는 액터의 시스템 사용 목적을 달성하도록 하기 위해서 입니다. 따라서 액터가 없는 유스케이스는 존재할 수 없습니다.

잘못된 유스케이스 이름

유스케이스를 찾을 때 중요하게 거론되는 것이 액터에게 측정 가능한 '가치(value)'를 제공해야 한다는 것입니다 여기서 가치라는 단어를 잘못 이해하게 되면 유스케이스 이름이 너무 추상적으로 작성됩니다. '가치'라는 단어를 생각할 때는 반드시 잊지 말아야 할 것이 시스템을 사용해서 얻는 가치라는 것입니다.

6. 관계와 관련된 문제들

일방향 연관으로 표현된 액터와 유스케이스

액터에와 유스케이스의 상호 작용은 항상 액터의 요청에 의해 시작된다고 하였기 때문에 커뮤니케이션의 시작을 표현하기 위한 액터에서 유스케이스로의 화살표 표시는 무의미합니다.

단일 유스케이스에 포함되는 유스케이스

포함 관계는 유스케이스의 이벤트 플로어들에 공통된 부분이 발견되었을 때 작성하는 관계이고, 공통된 부분이 발견되었다는 것은 두 개 이상의 유스케이스들에서 공통부분을 갖고 있다는 것을 의미합니다. 포함 관계가 나타나면 항상 두개 이상의 유스케이스들이 공통 부분에 해당하는 유스케이스를 포함해야 합니다.

4장 컬레보레이션

1. 컬레보레이션 기호

컬레보레이션의 표현은 그림 4.2 와 같이 컬레보레이션 이름을 포함하는 점선으로 된 타원 기호를 사용합니다.

컬레보레이션 내부에는 컬레보레이션에 참여하는 역할과 역할과 역할을 연결하는 커넥터가 작성됩니다.

그림 4.2 는 판매가 이루어지기 위해서는 구매자라는 역할과 판매자라는 역할이 필요하고, 그들 사이에 협력이 요구됨을 보여주기 위해서 구매자와 판매자를 커넥터로 연결하였습니다.

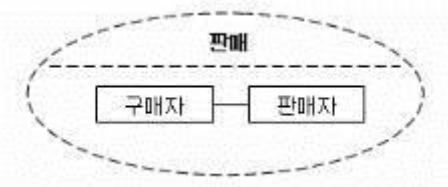


그림 4.2 컬레보레이션

역할을 구현하는 클래스를 명시하고자 할 때는

역할이름 뒤에 ":"와 클래스이름을 작성합니다.

그림 4.3 은 구성요소의 클래스가 TaskQueue 와 SlidingBar 이고, 이들의 구성요소들이 수행하는 역할이 Subject 와 Observer 임을 보여주고 있습니다. 다른 방법으로는 그림 4.4 와 같이 구성요소들의 클래스를 컬레보레이션 외부에 표현하는 방법을 사용할 수 있습니다.

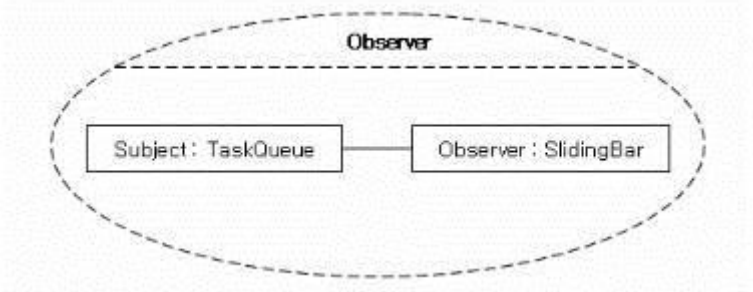


그림 4.3 클래스를 포함한 컬레보레이션 내부구조

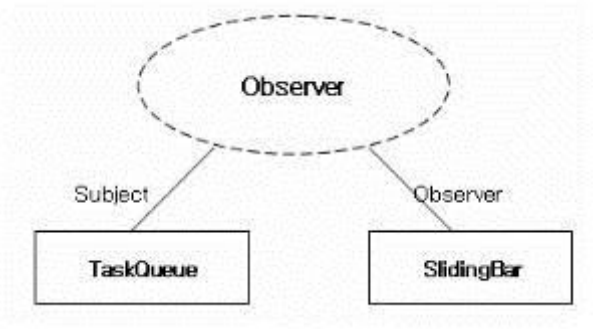


그림 4.4 클래스를 포함한 컬레보레이션 내부구조 대안적인 표현

2. 컬레보레이션 어커런스

컬레보레이션에 대한 특별한 상황에 대한 적용을 컬레보레이션 어커런스라고 합니다.
컬레보레이션 어커런스는 특수한 상황에서의 협력을 나타내기 때문에 이것 또한 컬레보레이션입니다.
쉽게 말해서 구체화된 컬레보레이션입니다.
예를 들어,
특정 항로를 따라 대상물들을 목적지까지 데려 가야하는 운항에는 이동수단을 조종하는 역할과 대상물들을 보호하는 역할과 대상물의 역할이 있습니다.
운항은 대상물을 목적지까지 데려가야 하는 목적을 가지고 있고, 그 목적을 달성하기 위해서는 여러 역할들이 필요하고, 그 역할들의 협력이 요구되는 컬레보레이션입니다.
운항의 특수한 예로는 항공기운항, 선박운항과 같은 것들이 있습니다.
항공기운항의 경우 이동수단을 조종하는 역할은 조종사이고, 대상물들은 승객과 화물이며, 대상물을 보호하는 역할은 승무원입니다.
항공기운항은 운항이라는 컬레보레이션의 특수한 사용에 해당하는 컬레보레이션 어커런스입니다. 또한 항공기운항은 조종사와 승객과 화물과 승무원 역할들의 협력을 나타내는 컬레보레이션입니다.
컬레보레이션은 여러 가지 구체적인 상황에서 발생하는 협력들을 추상화한 것으로 행위의 재 사용을 위한 매우 중요한 요소입니다. 만약 여러분이 대부분의 전자상거래에서 필요한 예약이나 주문에 대한 컬레보레이션을 잘 명세한다면, 이 컬레보레이션은 많은 시스템 개발에서 컬레보레이션 어커런스로 재사용되어질 것입니다.
컬레보레이션을 정확히 표현하기 위해서는 구조적인 부분과 행위적인 부분을 모두 표현해야 합니다. 어떤 요소들이 협력하는가는 구조적인 부분으로, 협력을 위해 어떻게 상호작용하는 가는 행위적인 부분으로 표현되어야 합니다. 일반적으로 컬레보레이션의 구조적인 부분은 컴포지트 스트럭처 다이어그램(Composite Structure Diagram)과 같은 스트

럭처 다이어그램(Structure Diagram)으로, 행위적인 부분은 시퀀스 다이어그램(Sequence Diagram)과 커뮤니케이션 다이어그램(Communication Diagram)와 같은 인터렉션다이어그램(Interaction Diagram) 들로 표현합니다.
컬레보레이션 어커런스는 그림 4.5 와 같이 표현되어집니다. 대안적인 표현으로 4.6 와 같이 나타낼 수도 있습니다.
개념에 대한 실체 표현과 같이 컬레보레이션 어커런스는 이름 다음에 ':'을 작성하고, 어커런스되는 컬레보레이션 이름을 작성합니다. 그림 4.5 는 판매 컬레보레이션의 두 개의 컬레보레이션 어커런스들로 중개인에 의한 판매 컬레보레이션을 구성한 것을 설명하고 있습니다. 중개인이라는 역할은 구매자의 역할과 판매자 역할 둘을 모두 수행하고 있습니다.

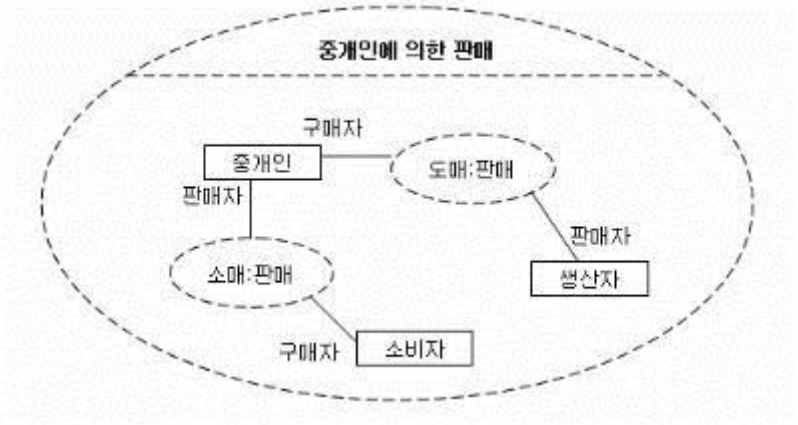


그림 4.5 컬레보레이션 어커런스

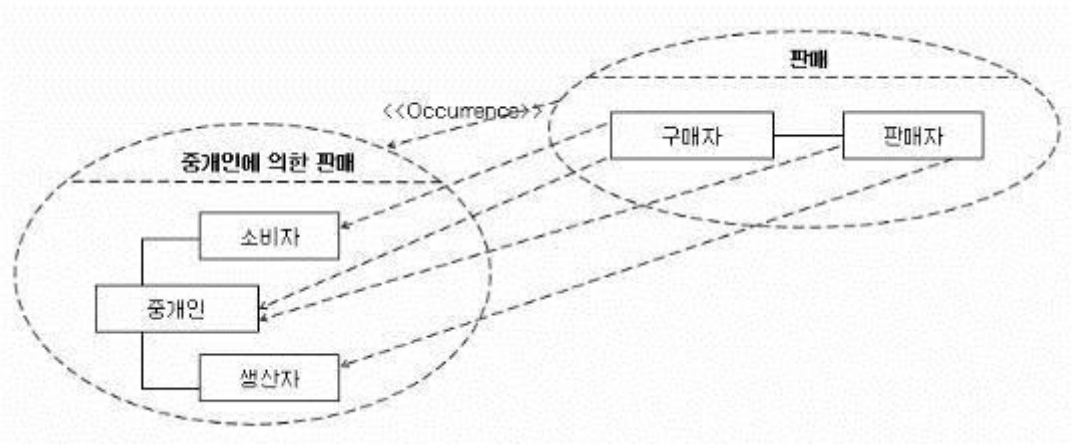


그림 4.6 컬레보레이션 어커런스의 대안적 표현

5장 인터렉션

1. 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.

2. 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.

3. 생명선

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

4. 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.

5. 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 생명선에서 여러 개의 메시지들을 주고 받기 때문에 메시지 전송의 선후행관계를 알기 어렵습니다. 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, 병행처리될 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 때문에, 가상코드나 프로그래밍코드를 사용할 수 있습니다.
그림 5.17 은 주문관리자가 주문에게 주문처리를 요청하면, 주문은 모든 주문항목들에게 주문처리를 요청합니다. 모든 주문항목에 대해서 반복하는 메시지이기 때문에 반복기호 별표를 하고 대괄호 안에 반복조건을 작성하였습니다. 각각의 주문항목은 재고관리자에게 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 화살표는 메시지의 방향을 나타냅니다. 만약 주문항목들에 대해서 병행처리를 하고 싶다면 '1.1*||[모든주문항목반복]:주문처리'와 같이 작성할 수 있습니다.

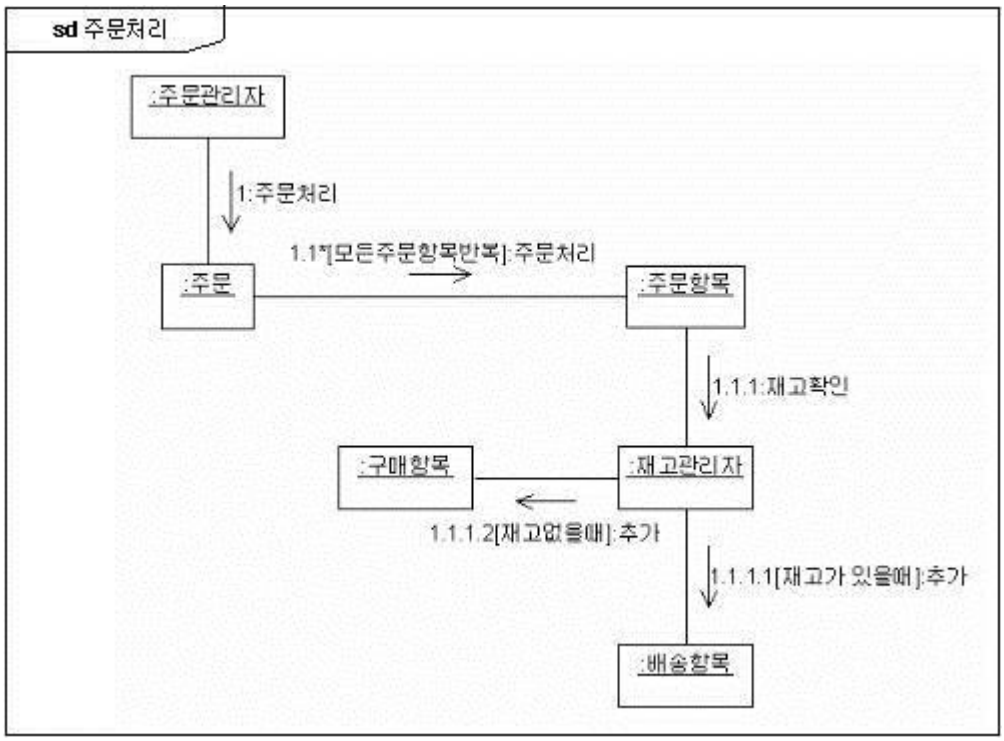


그림 5.17 커뮤니케이션 다이어그램

6. 인터렉션 오버뷰 다이어그램 다이어그램

UML 이전 버전에서의 시퀀스 다이어그램은 시나리오의 제어흐름을 표현하기가 어려웠

습니다. UML 2.0 에서 여러 가지 연산자들이 추가되었지만 여전히 제어흐름을 표현하기는 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 취합해서 만든 것이 인터렉션 오버뷰 다이어그램입니다. *액티비티 다이어그램에 대한 자세한 사항은 6 장 액션과 액티비티를 참조하시기 바랍니다.*
인터렉션 오버뷰 다이어그램의 주요 구성요소는 인터렉션과 인터렉션에 대한 참조와 액티비티 다이어그램에서 사용되는 모델요소들입니다. 인터렉션에 참여하는 생명선들은 lifelines 라는 키워드 다음에 열거합니다. 그림 5.18 은 음식점에서 음식을 요청하고, 요청한대로 음식이 나오지 않는 경우에 계속적으로 재요리를 요청하고, 요청한대로 나오면 먹는 시나리오입니다. 인터렉션 오버뷰 다이어그램을 사용함으로 처음부터 시나리오를 시퀀스 다이어그램에 상세적으로 작성하지 않고, 간략하게 작성하고 점점 상세하게 작성할 수 있는 잇점을 얻을 수 있습니다.

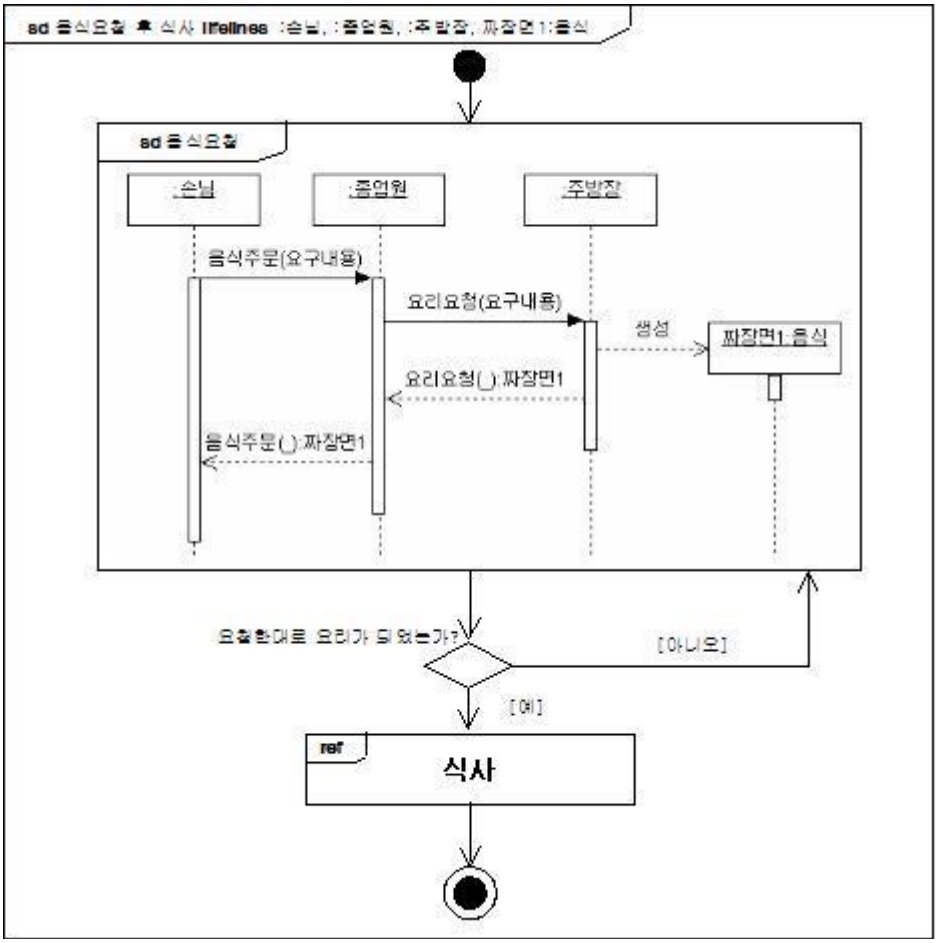


그림 5.18 인터렉션 오버뷰 다이어그램

7. 타이밍 다이어그램

타이밍 다이어그램은 리얼타임시스템과 같이 시간에 대한 제약이 중요하게 다루어져야 할때 사용되는 다이어그램입니다. 타이밍 다이어그램은 가로축에 시간을 표시하고, 세로축에 상태를 표시하여 시간이 지남에 따라 구성요소들의 상태의 변화를 보여줄 수 있습니다. 상태다이어그램이 개개의 구성요소에 대한 상태를 표현하는 반면, 타이밍 다이어그램은 구성요소들 사이의 상호작용과 상태변화를 동시에 보여줄 수 있습니다. *상태는 10 장에서 자세하게 설명됩니다.*

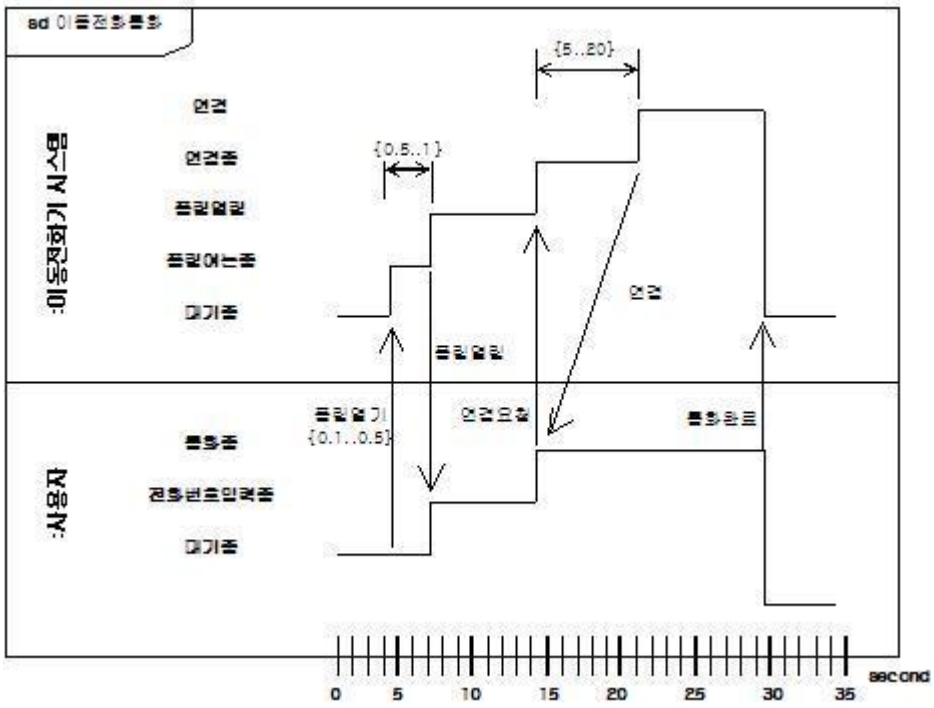


그림 5.19 타이밍 다이어그램

그림 5.19 는 사용자가 이동전화를 이용하여 전화통화에 대한 타이밍 다이어그램입니다. 생명선인 ':사용자'와 ':이동전화기시스템'은 다이어그램의 가장 왼쪽에 작성되며, 생명선은 서로 다른 구획으로 분리됩니다. 사용자는 대기중 상태에서 이동전화기의 플립열기버튼을 누르면 이동전화기는 플립을 자동으로 엽니다. 이때 플립열기라는 메시지의 전달로 플립이 열리기 시작하는 시점까지의 시간은 최소 0.1 초, 최대 0.5 초 안에 이루어져야 합니다. 또한 플립이 완전히 개폐되는 시간은 최소 0.5 초, 최대 1 초의 시간이 걸려야 합니다. 플립이 다 열리고 나면 이동전화기 시스템은 플립열림 상태가 되고, 플립이 다 열렸다는 메시지를 사용자에게 보냅니다. 사용자는 플립열림 메시지를 받고 전화번호를 입력합니다. 사용자가 전화번호를 다 입력하고 난 후에는 연결버튼을 눌러, 이동전화기에 연

결요청메시지를 보냅니다. 연결요청을 받은 이동전화기는 해당하는 번호로 연결을 시도합니다. 이때 소요되는 시간은 최소 5 초, 최대 20 초여야 합니다. 연결이 완료되면 이동전화기는 연결메시지를 사용자에게 보냅니다. 연결메시지를 받은 사용자는 통화를 시작합니다. 통화가 완료된 후에는 사용자와 이동전화기는 모두 대기상태가 됩니다.

타이밍 다이어그램에 메시지들이 많이 나타날 경우에는 어떤 화살표가 어떤 메시지에 대한 것인지 혼란스러운 경우가 발생합니다. 이를 방지하기 위해서 메시지는 그림 5.20 과 같이 레이블로 표현할 수도 있습니다.

그림 5.20 레이블로 표현된 메시지

타이밍 다이어그램에 명시적으로 메시지 전송 순서를 표현하고자 할 때는 그림 5.21 과 같은 기호를 사용할 수 있습니다. UML 2.0 에서는 이를 General Ordering 이라고 합니다. General Ordering 의 화살표는 선행메시지와 후행메시지를 나타냅니다. 화살표가 가르키는 메시지가 후행메시지가 됩니다.

그림 5.21 General Ordering

그림 5.22 는 좀더 간략화된 타이밍 다이어그램의 표현입니다. 타이밍 다이어그램은 상호작용을 표현하기 관련된 구성요소 모두를 표현할 수도 있고, 개개 구성요소 별로 작성할 수도 있습니다.

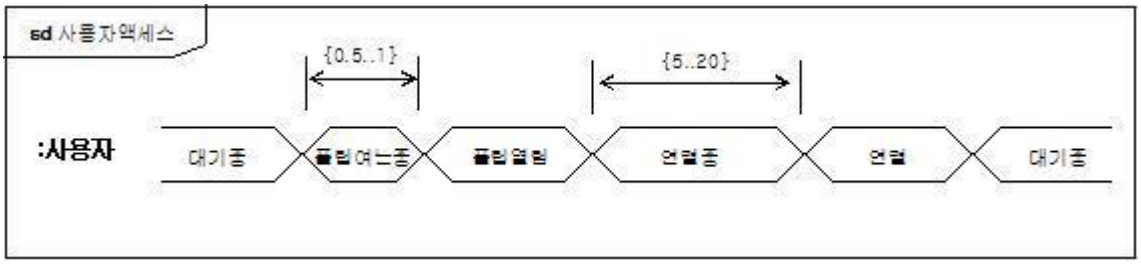


그림 5.22 타이밍 다이어그램의 간략화된 표현

6 장 액션과 액티비티

1. 액션

사람들에 의해 수행되는 실세계의 액션들을 정의하고자 하는 시도가 되고는 있지만, 우리가 사람을 창조한 하나님이 아니기 때문에 이는 매우 어려운 일입니다. 하지만 우리가 만든 객체지향프로그래밍 언어에 있어서의 액션을 정의하는 것은 가능한 일입니다. 따라서 UML 에서는 객체지향프로그래밍에 맞추어서 액션들을 정의하고 있습니다. 이러한 액션들 중에서 서비스를 요청하고 제공하기 위한 메시지를 전달하고 받는 것은 실세계에 동일하게 적용될 수 있습니다.

객체지향프로그래밍에 있어 가장 많이 수행되는 최소단위의 행위는 객체를 생성하고, 객체의 속성 값을 설정하거나 객체들 사이의 링크를 만드는 일과, 속성 값을 변경하거나 링크를 삭제하는 일입니다. 객체들은 서비스를 주고, 받기위해서 메시지를 보내거나 받아야 합니다. 서비스를 제공하기 위해서 계산을 수행해야 하며, 예외처리를 위해 예외상황에서 예외를 발생시켜야 합니다. 최소단위의 행위로 평가될 수 있는 계산은 산술, 불린, 문자와 관련된 기능들과 같이 입력된 값들만을 가지고 계산이 수행되는 것으로 상태변화 등의 부작용이 발생하지 않습니다. 이런 함수를 기본함수라고 합니다.

속성 값을 설정하거나 링크를 만드는 행위를 읽기액션이라고 하고, 속성 값을 변경하거나 링크를 삭제하는 행위를 쓰기액션이라고 합니다. 객체들이 메시지를 보내거나 받는 행위를 커뮤니케이션액션이라고 하고, 기본함수를 수행하는 행위를 기본함수수행액션이라 하고, 예외상황에서 예외를 발생시키는 행위를 예외발생액션이라고 합니다.

2. 커뮤니케이션 액션

5 장 인터렉션의 메시지 부분에서 설명했듯이, 메시지는 요청방식에 따라 오퍼레이션 호출과 시그널 송신으로 나누어지고, 메시지를 보내는 것이나 받는 것은 하나의 액션입니다.

서비스를 제공 받기 위해서 다른 역할에게 메시지를 보내는 액션을 오퍼레이션호출액션(Call Operation Action) 또는 시그널송신액션(Send Signal Action)이라고 합니다. 오퍼레이션호출액션은 수신자에게 전달되는 메시지가 오퍼레이션 호출이고, 시그널송신액션은 시그널 발생입니다. 메시지 전달이 하위 레벨의 역할에서 상위 레벨의 역할에게 전달될 경우, 상위 레벨의 역할의 이름을 직접적으로 언급하지 않고, 역할이 수행하는 서비스를 직접적으로 언급합니다. 서비스는 하나의 행위이기 때문에 이러한 방식으로 메시지를 전달하는 액션을, 직접적으로 행위를 호출한다고 해서 행위호출액션(Call Behavior Action)이라고 합니다.

메시지를 받는 액션은 서비스를 제공할 역할을 명시적으로 언급하든, 언급하지 않든 동일한 서비스 수행을 요청받는 것이기 때문에 호출방식에 상관 없이 호출수용액션(Accept Call Action)이라고 합니다.

오퍼레이션이나 행위를 호출하는 이벤트를 호출이벤트라고 하고, 신호를 발생시키는 이벤트를 시그널이라고 하고, 시간의 변화에 따라 발생하는 이벤트를 타임이벤트(Time Event)라고 하고, 구성요소의 속성 값이나 링크의 변화에 따라 발생하는 이벤트를 변화이벤트(Change Event)라고 합니다. 호출이벤트를 제외한 이벤트들을 수용하는 액션을 이벤트수용액션(Accept Event Action)이라고 합니다. 이벤트에 대한 좀더 자세한 내용은 11 장 상태에서 설명됩니다.

오퍼레이션 호출 메시지가 동기적인 방식으로 전달될 때 응답메시지를 보내야 합니다. 이 응답메시지를 보내는 것 또한 하나의 액션입니다. 이러한 액션을 응답액션(ReplayAction)이라고 합니다.

오퍼레이션호출액션(CallOperationAction), 행위호출액션(CallBehaviorAction)

오퍼레이션호출액션은 그림 6.2 와 같이 액션이름에 오퍼레이션 이름과 해당 오퍼레이션의 클래스 이름을 추가적으로 작성합니다. 액션이름이 호출되는 클래스의 오퍼레이션의 이름과 같은 경우는 (클래스::)와 같이 오퍼레이션 이름을 생략할 수 있습니다.

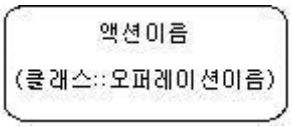


그림 6.2 호출액션

액티비티의 여러 액션들 중에 행위호출액션을 구분하기 위해서 그림 6.3 과 같이 액션의 오른쪽 밑 부분에 갈퀴 기호를 사용합니다. 행위호출액션이 호출하는 액티비티이고, 이 액티비티 또한 여러 액션들로 구성된 복잡한 내부구조를 가질 것입니다. 갈퀴 기호는 좀더 복잡한 내부구조를 감추는 방법으로 사용됩니다. 행위호출액션은 갈퀴 기호를 사용하지 않고, 직접적으로 해당 액티비티로 표현될 수 있습니다.

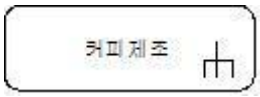


그림 6.3 행위호출액션

시그널송신액션(SendSignalAction)

시그널송신액션은 시그널발생 메시지를 수신자에게 보내는 행위입니다. 시그널은 시그널을 받는 수신자의 상태 전이를 일으키거나 액티비티를 실행시킵니다. 시그널발생메시지는 비동기적인 메시지로 어떠한 응답메시지도 송신자에게 전달되지 않습니다. 시그널에 대한 좀더 자세한 내용은 11 장 상태에서 설명됩니다. 시그널송신액션은 그림 6.4 와 같이 한쪽면이 볼록한 사각형으로 표현됩니다.

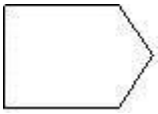


그림 6.4 시그널송신액션

이벤트수용액션(AcceptEventAction)

이벤트수용액션은 그림 6.5 와 같이 한쪽면인 오목한 사각형으로 표현합니다. 이벤트가 타임이벤트인 경우는 모래시계 기호를 사용할 수 있습니다.

그림 6.5 이벤트수용액션

자판기의 예에서 사용자가 자판기의 커피종류버튼을 클릭하면, 커피종류버튼의 시그널송신액션은 커피종류를 선택했다는 시그널발생 메시지를 중앙제어장치로 보냅니다. 시그널발생메시지는 중앙제어장치의 이벤트대응액션에 의해 받아들여지고, 이벤트대응액션은 중앙제어장치의 커피를 만드는 행위를 호출합니다. 이에 대한 표현은 그림 6.6 과 같이 할 수 있습니다. 시그널송신액션과 이벤트대응액션은 동일한 시그널에 대해서 보내고 받는 역할을 하기 때문에 기호 또한 하나는 한쪽면이 볼록나오게 다른 하나는 한쪽 면이 들어가 있어 둘이 결합될 수 있는 모습을 가지고 있습니다.

그림 6.6 시그널송신액션과 이벤트대응액션

만약 자판기가 출력기능을 가지고 있어서 매월 판매량을 보고한다면 그림 6.7 과 같이 타임이벤트에 대한 이벤트대응액션으로 표현할 수 있습니다.

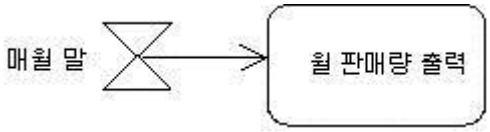


그림 6.7 타임이벤트에 대한 이벤트대응액션

기본함수 수행액션

시스템은 책임수행과정에서 많은 계산을 수행합니다. 이런 계산들 중, 산술계산과 같이 입력값 만으로 직접적인 계산을 처리할 수 있어 단일 액션으로 고려될 수 있는 최소단위의 함수를 기본함수(Primitive functions)라고 합니다. 기본함수는 입력값만으로 계산을 수행하기 때문에 어떠한 다른 구성요소에도 영향을 주지 않습니다. UML 2 에서는 특별한 기본함수를 정의하지는 않고 있지만, 수학과 같은 특별한 도메인에 대해서는 확장해서 정의할 수도 있습니다. 일반적인 기본함수는 산술, 불린, 문자와 관련된 기능들을 포함합니다.

예외발생액션

액티비티 내부에서 예외적인 상황이 일어나면, 예외적인 상황을 처리할 수 있도록 예외를 발생시켜야 합니다. 예외를 입력으로 받아 예외를 발생시키는 액션을 예외발생액션이라고 합니다. 예외발생액션이 수행되면 예외가 발생된 지점에서 액티비티 수행을 즉각적으로 멈추고, 예외를 처리할 수 있는 예외처리자를 찾습니다.

일반적으로 예외처리는 예외가 발생할 가능성이 있는 영역을 설정하고, 그 영역에서 예외가 발생하면 예외를 잡고, 예외에 해당하는 예외처리자(Exception Handler)가 이를 처리합니다. 이렇게 설정된 영역을 예외에서 보호되고 있다는 의미에서 보호(protected)점이라고 합니다. 보호점은 그림 6.8 과 같이 설정영역에 해당하는 액션을 포함한 액션기호로 표현합니다. 예외처리자는 예외액션에 의해 전달된 예외를 입력으로 받습니다. 예외처리자는 예외를 입력으로 갖는 액션기호로 표현합니다.

그림 6.8 은 고객을 추가하는 액션에서 추가할 수 있는 고객수를 넘을 수가 있기 때문에 보호영역으로 설정하는 것을 보여주고 있습니다. 고객을 추가할 때, 고객수가 설정된 인덱스를 넘어서면 예외를 발생시키고, 발생된 예외를 Index Overflow 예외처리자에게 넘깁니다. 예외처리자는 예외를 기록합니다. Index Overflow 예외처리자가 입력으로 받는 예외는 예외발생액션이 발생시킨 예외타입이거나 그 상위 예외타입이어야 합니다. 고객수가 설정된 인덱스를 넘지 않으면 전체고객수를 셉니다. 보호점에서 예외처리자로의 전이는 번개 기호를 사용하여 나타냅니다.

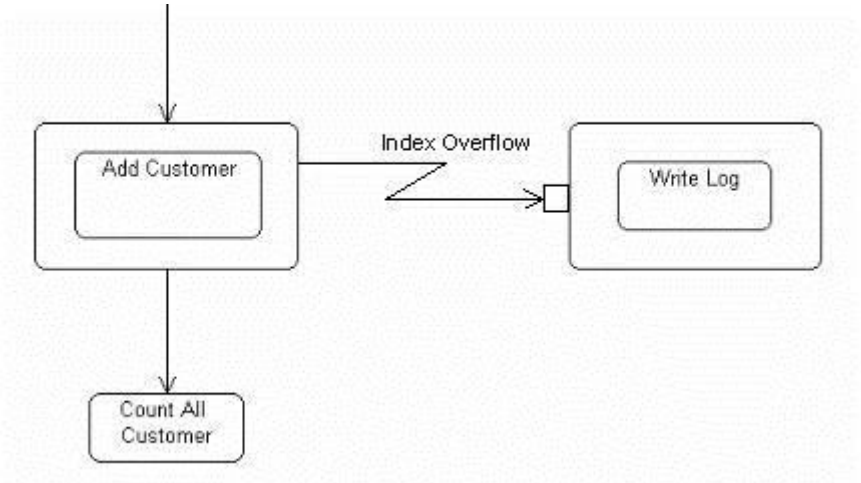


그림 6.8 예외처리

3. 액티비티

UML 2 명세서는 액티비티에 대해서 다음과 같이 정의하고 있습니다.

액티비티란? 일련의 액션들로 구성된 매개변수화 된 행위 명세이다.

서비스를 제공하는 행위인 액티비티 또한 액션과 마찬가지로 계약의 원리를 따릅니다. 따라서 액티비티는 서비스 제공을 위한 입력정보와 출력정보를 작성해야 하며, 서비스제

공에 대한 선행조건과 후행조건을 작성해야 합니다. 위의 액티비티의 정의에서 매개변수화(parameterized) 되었다는 의미는 바로 입력과 출력을 가진다는 것을 의미합니다.
액티비티 표현은 그림 6.9 와 같이 내부의 왼쪽 상단에 이름을 갖는 모서리가 둥근 사각형 기호를 사용합니다. 사각형 내부에는 액션들과 흐름들이 작성되며, 매개변수들은 사각형의 경계위에 작성되고, 매개변수의 타입은 액티비티 이름 밑에 작성됩니다. 선행조건과 후행조건은 <<precondition>>, <<postcondition>>키워드로 작성됩니다.

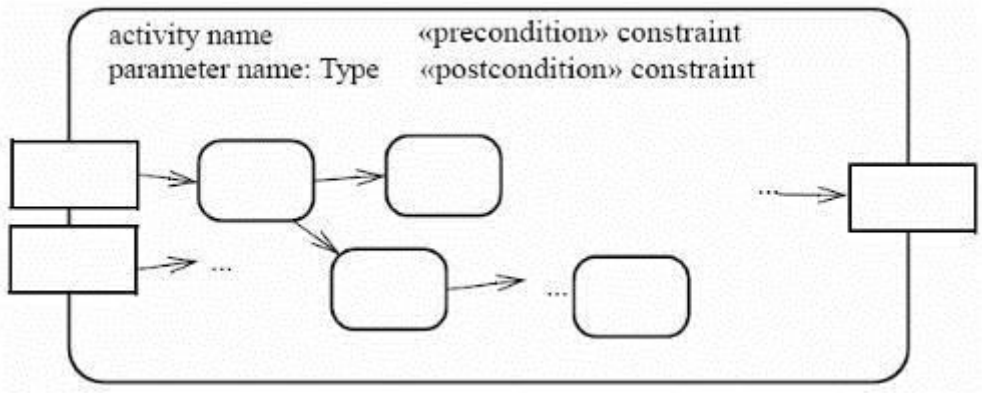


그림 6.9 액티비티

시스템은 제한된 자원으로 시스템 사용자들의 욕구를 만족시켜 주어야 하기 때문에, 서비스의 결과도 중요하지만, 서비스 실행 과정의 효율성도 중요합니다. 따라서 누가, 언제 시작하고, 중단이나 취소가 있었다면 그 이유는 무엇이며, 누구에게 서비스 결과를 보고할 것이며, 어느정도의 시간과 비용으로 어떤 품질의 서비스를 제공했는가에 대한 서비스 성과측정이 요구됩니다. 서비스에 대한 성과측정을 나타내기 위해서 액티비티는 그림 6.10 과 같이 <<activity>> 키워드를 갖는 분류자 기호를 사용할 수 있습니다.

그림 6.10 분류자 기호로 표시된 액티비티

메시지는 송신자가 전달하는 내용을 설명하는 것이고, 서비스는 수신자가 실제 실행해야 하는 행위이기 때문에, 메시지 이름은 송신자의 입장에서 상대방에게 보내는 요구내용으로 작성하였지만, 서비스 이름은 실제 실행하는 수신자의 입장에서 작성합니다. 따라서 서비스의 이름을 작성하기 위해서는 수신자의 입장에서 "메시지로 전달된 송신자의 요청을 달성해 주려면 어떤일을 해야 하는가?"에 대한 답을 구해야 합니다. 이 답을 통해서 '자재운반'과 같이 '행위의 대상 또는 목적 + 행위'의 형태로 작성합니다. 운반은 활동의

행위이고, 자재는 운반이라는 행위의 대상을 나타냅니다. 액션 또한 실제 실행해야 하는 행위로 이와 동일한 방법으로 이름을 작성합니다.

4. 액티비티 다이어그램(Activity Diagram)

액티비티 다이어그램은 비즈니스 프로세스 공학에서의 조직모델링과 워크플로어를 명세하기위해 가장 많이 사용됩니다. 최근에는 시스템레벨 프로세스나 오퍼레이션의 상세로직을 설계하기 위해서도 사용됩니다.

워크플로어나 시스템레벨 프로세스에는 다양한 역할들이 참여하기 때문에 역할별로 수행하는 액션들을 그룹화 할 필요가 있습니다. 역할들에 의해 액션들을 그룹화 함으로 해당 워크플로어나 시스템레벨 프로세스를 수행하기 위해 어떤 역할이 어떤 액션들을 수행하면서 다른 역할들과 관계를 맺는지를 쉽게 파악할 수 있습니다.

역할별로 구획된 한 부분을 스윔레인(swimlane)이라고 합니다. 그림 6.35 를 보고 수영경기장을 연상해 보십시오. 수영경기장에 역할들에 해당하는 선수들이 자신의 라인번호를 받고, 해당 라인의 출발선에 서 있습니다. 수영선수들은 자신의 라인을 따라서만 수영을 해 나갈 수 있으며 그 라인을 벗어나면 실격이 됩니다. 출발 신호가 떨어지면 그 라인에 놓여 있는 액션들을 수행하면서 나아갑니다. 스윔레인은 그림 6.35 와 같이 수평적으로 작성될 수도 있고 수직적으로 작성될 수도 있습니다.

그림 6.36 은 주문처리를 고객과 지불당담자와 주문담당자에 해당하는 액션들로 그룹화해서 표현한 것입니다. 송장은 송장전송에 의해 출력되고 대금지불에 사용되기 때문에 특정한 역할에 할당되지 않습니다.

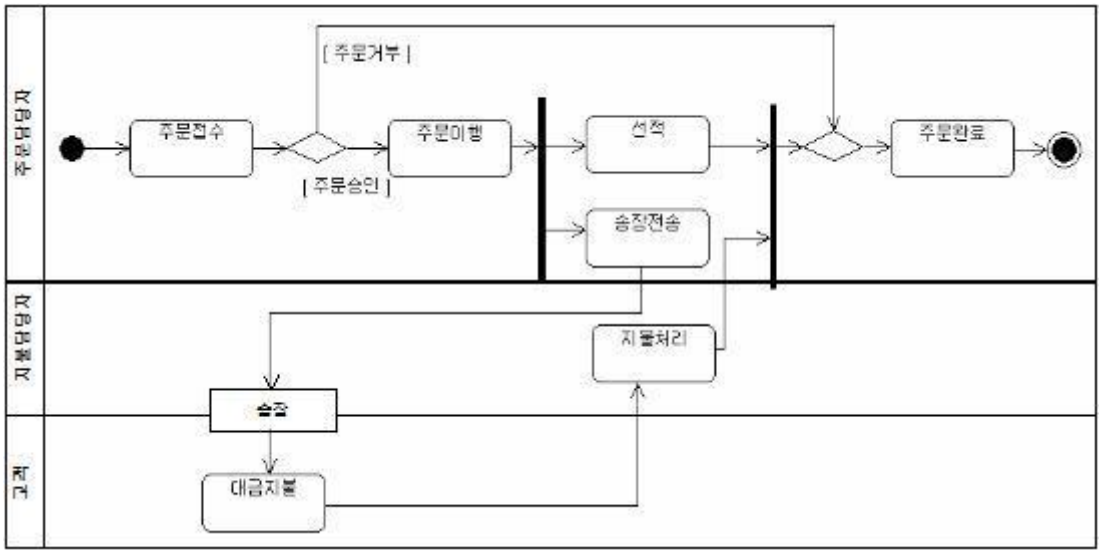


그림 6.36 스윔레인

액티비티 다이어그램에 전이가 복잡하게 얽혀 있거나 너무 길게 작성되어질 경우, 다이어그램을 간략화하기 위해 그림 6.37 과 같은 연결 기호를 사용할 수 있습니다. 같은 이름을 가진 연결기호끼리 연결됩니다.

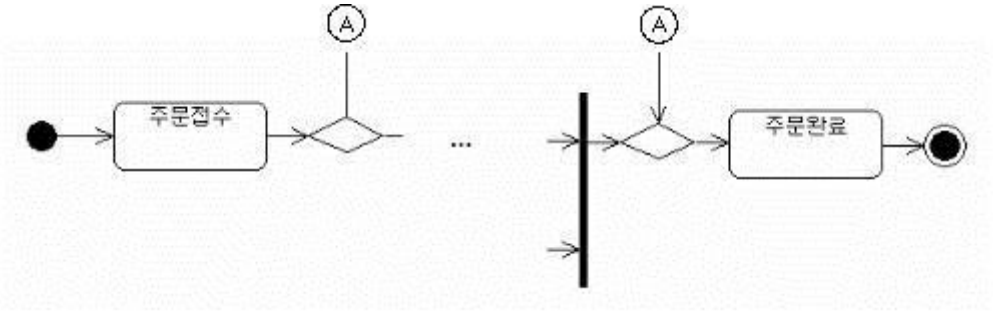


그림 6.37 연결 기호

액터와 시스템 사이에서의 시스템의 역할 수행이 시스템 내부구성요소들의 역할들에 의해서 수행되듯이, 하나의 역할은 좀더 작은 여러 역할들로 분할될 수 있습니다. 역할들의 크기에 따라 같은 레벨의 역할들로 액티비티 다이어그램을 작성하는 것이 가장 좋은 방법입니다. 하지만 많은 비즈니스 모델작성자들이 워크플로에 역할보다는 실제 기업의 조직구조로 워크플로어 작성해왔기 때문에 워크플로어는 조직구조를 반영할 수 있도록 지원하고 있습니다.

워크플로어의 작성에 있어 중요하게 다루어지는 것중 하나는 액티비티가 수행되는 위치나 사용되는 자원에 대한 것입니다. 역할도 표현되어야 하고, 위치나 사용되는 자원등을 표현하기 위해서 워크플로어는 다차원으로 작성되어야합니다. 일반적으로 2 차원으로 작성됩니다.

워크플로어에는 비즈니스 내부 조직만이 참여자가 되는 것이 아니라, 고객이나 공급자와 같은 외부조직들도 참여하게 됩니다. 내부조직과 외부조직을 구분하기 위해서 외부조직에 <<external>> 키워드를 사용하여 표현합니다. 그림 6.38 의 차원으로 작성된 도시들은 도시타입을 갖는 액티비티의 속성 값들입니다. 속성 값임을 나타내기 위해서 <<attribute>>라는 키워드를 사용해서 표현합니다.

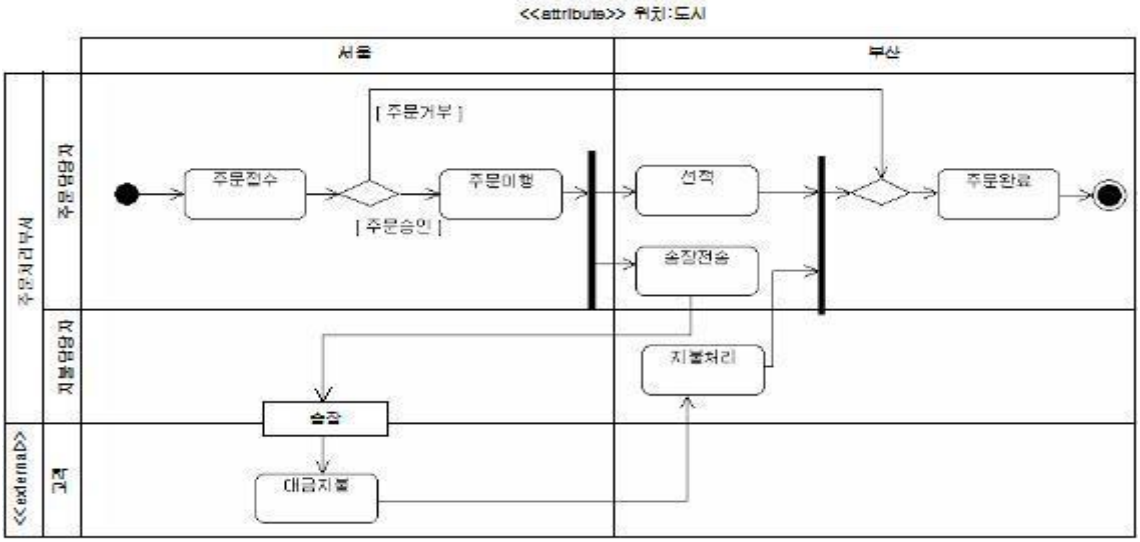


그림 6.38 다차원의 작성된 워크플로어

7장 역할 명세

1. 시스템 책임 정제

유스케이스 기술서에 작성된 시스템 책임은 사용자 관점에서 시스템이 사용자에게 제공해야하는 결과들에만 중점을 맞추어서 기술되었기 때문에 시스템이 결과를 제공하기 위해서 무엇을 해야 하는지에 대한 책임들로 정제되어야 합니다.

시스템이 해야 할 일들은 유스케이스에서 설명했듯 액터로부터 제공받은 정보나 시스템의 서비스 수행 상태를 기록해야 하고, 액터가 서비스를 시작하거나 시스템이 요청한 행위를 수행하기 위해서 필요로 하는 정보들을 제공해야 하며, 액터에게 서브시를 제공해야 하고, 서비스 제공 과정에서 필요로 하는 계산이나 의사결정 등을 해야 합니다. 여기서 액터는 일차액터와 이차 액터 모두를 포함합니다.

- 시작 조건 구성

시작 조건은 사용자가 시스템의 서비스를 시작하기 휘한 조건으로, 시스템은 시작 조건이 만족될 수 있도록 환경을 구성해야 합니다.

- 선행 조건과 후행 조건 평가

선행 조건이 만족되지 않는 경우에는 시스템의 서비스 제공 과정을 시작하더라도 진행을 계속 할 수 없기 때문에 서비스를 시작하기 전에 평가되어야 합니다. 일반적으로 올바르게 서비스가 진행되면 후행조건은 당연히 만족되어야 하는 것이기 때문에 테스트를 수행하는 경우가 아닌 경우에는 성능의 문제 등을 고려해서 후행 조건은 평가하지 않습니다.

액터로부터 제공 받은 정보 기록

액터가 제공한 정보가 이후의 서비스 제공 과정에서 요구될 경우 기록합니다. 기록된 정보들은 필요에 따라 수정이나 삭제 가능합니다.

- 액터에게 정보 제공

액터가 서비스 제공 과정에 쉽고 정확한 방법으로 참여하는 데 필요한 모든 정보를 제공해야 합니다. 정보의 형태는 표나 리스트, 또는 폼의 형태가 될 수 있습니다. 필요하다면 멀티미디어 정보 등도 제공해야 합니다.

- 서비스 수행 상태 기록

서비스 수행 상태는 사용자의 사용 중지나 예외 상황의 발생으로 인해 서비스 재공 과정이 중단되었을 때 기록됩니다. 기록된 서비스 수행 상태는 사용자가 시스템을 다시 기동

했을때 어느 단계에서부터 서비스 제공 과정을 시작할 것인지를 결정하기 위해 사용됩니다. 서비스 수행상태는 사용자의 시스템 사용 시점에 따라 결정됩니다.

- 서비스 제공 과정에 필요한 계산 수행과 의사 결정
시스템은 서비스 제공하기 위해서 계산이나 의사 결정을 수행합니다. 의사 결정내용을 상세히 기록허기 위해서는 의사결정 지점을 찾아야 합니다.

- 서비스 제공
시스템은 액터가 원하는 최종 결과인 서비스를 제공해야 합니다.

2. 역할 분류에 따라 역할 찾기
유스케이스 실현에 대한 컬레보레이션의 역할들은 역할 분류인 바운더리, 컨트롤, 엔티티에 따라 찾아 이름 붙입니다.
바운더리 유형에 해당하는 역할들은 스토리 보드를 작성했다면 스토리 보드의 구성 요소들로부터 찾을 수 있습니다. 바운더리 유형들은 유스케이스 이름에 'UI'또는 '스토리보드'라는 접미어를 사용해서 이름을 작성합니다. 바운더리 유형에 해당하는 시스템 책임은 액터로부터 정보를 받고 액터에게 정보를 제공하는 것입니다.
컨트롤 유형에 해당하는 역할들은 액터에게 서비스를 제공하기 위해서 유스케이스의 흐름에 제어하는 것으로, 유스케이스 하나에 하나의 컨트롤을 찾고 그 이름 또한 유스케이스 이름에 '관리자'라는 접미어를 사용합니다. 컨트롤 유형에 해당하는 시스템 책임은 서비스를 제공하기 위한 모든 과정을 제어하는데 필요한 의사 결정들입니다.
엔티티 타입에 해당하는 역할들은 실제 시스템 책임을 수행하는 것입니다. 대다수의 엔티티 타입들은 정보를 다루는 것으로 정보 역할 이름을 그대로 사용합니다.

3. 컬레보레이션으로 역할 찾기
하나의 역할이 다른 역할들과 협력하면서 자신의 역할을 제대로 수행하기 위해서는 자신이 협력하는 범위는 어디이고, 자신과 협력하는 역할들은 어떤 것들이며, 그들이 협력 안에서 수행하는 역할은 무엇인지를 이해해야 합니다. 역할 이름이 협력하는 범위와 협력하는 다른 역할들에 대한 이해를 반영하기 위해서 뿐만 아니라 시스템 구성 요소들이 역할을 갖는 것은 협력에 참여하기 때문이라는 점에서 컬레보레이션을 기준으로 역할들을 찾아야 합니다.

4. 개념화 하기
개념화 한다는 것은 어떤 현상에 대해 일반화하는 것을 말하며 쉽게 용어를 정의하는 것이라고 생각할 수 있습니다. 개념은 구체적인 사실들에서 귀납되어 형성되기 때문에 어

떤 현상을 개념화하기 위해서는 진행 과정에 있는 사실들이 아닌 완료된 사실들을 기준으로 해야 합니다.

5. 역할의 서비스 제약 사항

서비스의 제약 사항은 서비스가 수행되기 이전에 만족되어야 하는 조건에 해당하는 선행조건, 서비스가 수행되고 난 이후에 만족되어야 하는 조건에 해당하는 후행조건, 서비스가 완료되고 서비스 요청자에게 제공되어야 하는 것에 해당하는 실행 조건이 있습니다.

8장 객체와 클래스

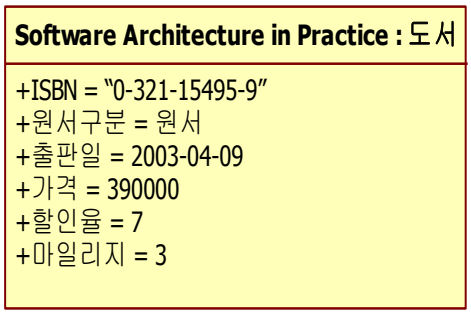
1. 객체(Objects)

정의 : 실 세계에 존재하거나 또는 개념적으로 존재하는 실체에 대한 표현이다. 객체는 트럭 또는 컴퓨터와 같은 구체적인 것을 나타낼 수도 있고 화학 작용, 은행 거래, 주문 또는 신용 이력, 이자율과 같은 개념적인 것을 나타낼 수 있다. 객체는 잘 정의 된 경계와 적용상의 의미를 가지는 개념, 추상 또는 사물이다. 시스템 내부의 각각의 객체는 상태, 행위, 정체성과 같은 세가지 특징을 가지고 있다.

인스턴스 기호로 표현되며 이름 부분과 구조적 특성 부분을 갖습니다.

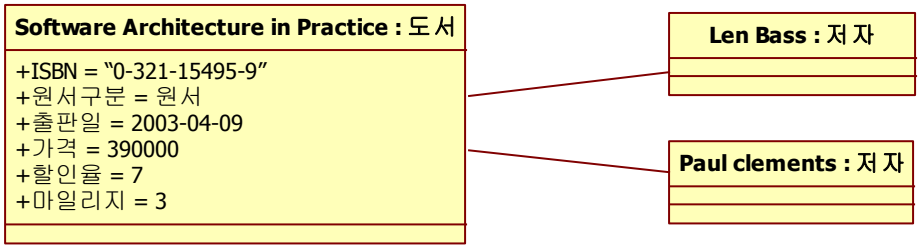
객체 이름:클래스 이름'

객체의 예



2. 링크(Links)

객체들 사이에 개념적 또는 물리적 연결. 즉, 순서를 갖는 객체 리스트이다.



3. 스냅샷(Snapshot)

정지 된 한 시점에서의 객체들과 그들의 관계들을 포착하는 것

-

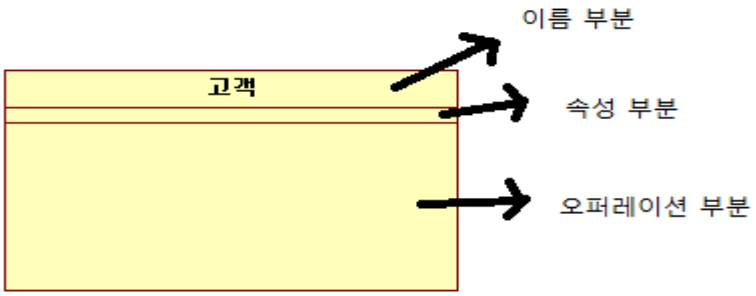
4. 객체 다이어그램(Object Diagram)

객체와 객체들 사이의 링크를 가지고 어떠한 순간에 객체에 대하여 알려진 것이 무엇인

가를 표현하는 다이어그램
어떠한 순간에 알고자 하는 객체들의 정보들을 표현하기 때문에 그 순간에 관심이 없는 객체의 구조적 특성이나 링크들은 표현할 필요가 없다.

5. 클래스(Classes)

공통의 속성들, 오퍼레이션들, 관계들, 의미들을 공유하는 개체들의 집합에 대한 기술(description)이다.



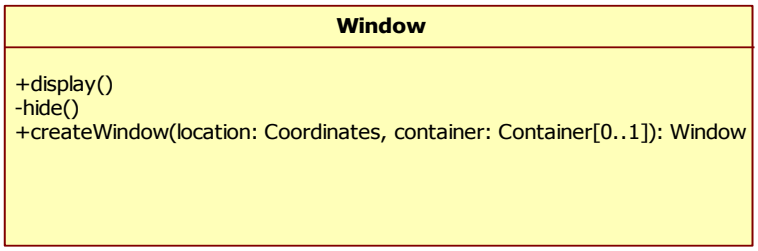
6. 속성(Attributes)

클래스의 구조적 특성에 이름을 붙인 것으로 구조적 특성에 해당하는 인스턴스가 보유할 수 있는 값의 범위를 기술한다.
[가시성] [/] [속성 이름] [:타입] [다중성] [=기본값] [{특성 문자열}]

-

7. 오퍼레이션(Operation)

이름, 타입, 매개 변수들과 연관된 행위를 호출하는데 요구되는 제약 사항들을 명세하는 클래스의 행위적 특징이다.
오퍼레이션은 섭스에 대한 명세로서 실제적인 구현 sosdyd을 담고 있지 않다. 실제적인 오퍼레이션의 구현은 메소드(Methods)에 의해서 수행되므로, 객체들이 실제 오퍼레이션을 수행하기 위해서는 반드시 메소드를 가져야 한다.
오퍼레이션의 형식
[가시성] 오퍼레이션 이름([매개변수 리스트])[:리턴 타입[다수성]] [{특성 문자열}]



8. 데이터 타입(Data Types)

오퍼레이션이 모두 순수 함수(pure function)로 되어 있는 분류자이다.

분류자로서 속성과 오퍼레이션을 가진다. 하지만 클래스와는 달리 인스턴스로 객체가 아닌 데이터 값을 가진다.

9. 기본형(Primitive Type)

미리 정의된 데이터 타입으로 내부 구조가 없다. 내부 구조가 없기 대문에 속성부분과 오퍼레이션 부분은 표현할 필요가 없다

10. 열거형(Enumeration)

열거된 값들 중 하나의 값만을 취할 수 있는 데이터 타입

11. 인터페이스(Interface)

역할을 표현하는 UML 모델 요소로서 객체들에 의해 실현됨.

역할이 제공해야 하는 서비스들은 속성과 오퍼레이션으로 작성됩니다. 인터페이스는 아래 그림과 같이 <<interface>> 키워드를 갖는 클래스 기호로 표현됩니다.

12. 구현(Implementation)

구현은 분류자와 인터페이스 사이의 특수한 형태의 실현관계입니다.

구현관계는 그림 8.15와 같이 속이 빈 삼각형 머리를 갖는 점선 화살표에 의해 표시됩니다. 실현관계를 표현할 때 많은 클래스들과 인터페이스들이 하나의 다이어그램에 작성될 경우는 키워드 만으로는 구분하기가 쉽지 않기 때문에 인터페이스를 그림 8.16과 같이 막대사탕모양의 아이콘을 사용해서 표현합니다.

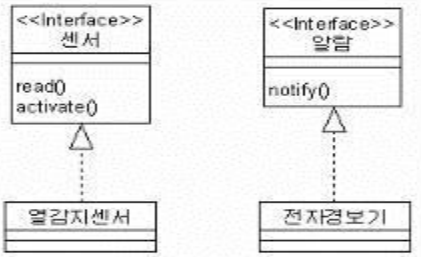


그림 8.15 실현 관계

그림 8.16 아이콘으로 표시된 인터페이스

객체가 실현하는 역할을 제공인터페이스(Provided Interface)라고 하고, 객체가 자신의 역할 수행을 위해 필요로 하는 다른 객체의 역할을 요청인터페이스(Required Interface)라고 합니다. 객체는 제공인터페이스를 통해 자신이 제공하는 서비스들이 무엇인지를 알리고, 요청인터페이스를 통해 자신이 필요로 하는 서비스들은 무엇인지를 알리는 것입니다. 예를 들어, 센서역할을 수행하는 객체에 대한 클래스는 센서인터페이스를 제공인터페이스로 알람 인터페이스를 요청인터페이스로 표현해야 합니다.

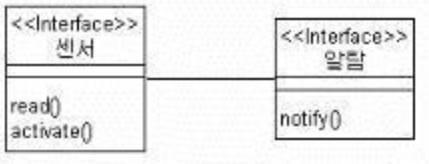


그림 8.17 감지반응 컬레보레이션을 위한 역할들

-

13. 포트(Ports)

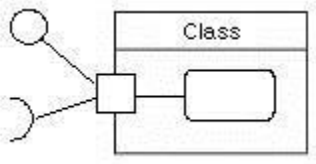
클래스와 클래스의 외부 환경 또는 클래스와 클래스의 내부 부분들 사이의 개별적인 상호 작용을 명세하는 클래스의 구조적 특징이다.

포트는 그림 8.20과 같이 포트를 포함하는 클래스의 경계에 걸친 작은 사각형 기호로 나타냅니다.

그림 8.20 엔진

포트를 포함하는 클래스가 포트를 통해 요청된 서비스를 포함부분들에게 위임하지 않고, 직접적으로 처리할 때, 이러한 포트를 행위포트라고 합니다. 행위포트(Behavior Port) 는 그림 8.25 와 같이 포트와 포트를 포함하는 클래스의 행위를 직접적으로 연결됩니다. 행위포트와 연결되는 행위는 모서리가 둥근 사각형으로 표현합니다.

-



-

그림 8.25 행위포트

14. 합성 구조(Composite Structure)

부품을 가지고 완제품을 만드는 방식을 합성이라고 하고, 이와 같은 방식으로 만들어지

는 구조를 합성구조라고 합니다.
예를 들어, 자동차를 구성하는데 앞 바퀴 두 개와 뒷 바퀴 두 개가 사용되고, 앞 바퀴들은 전방차축에 뒷바퀴들은 후방차축에 연결되어 있다면 그림 8.26과 같이 표현할 수 있습니다.

그림 8.26 자동차의 구조

그림 8.26 의 자동차는 다수성을 사용해서 그림 8.27 과 같이 좀 더 간단하게 표현될 수 있습니다.

-

그림 8.27 다수성을 사용해서 표현한 자동차의 구조

15. 컴포넌트

컴포넌트는 내부를 캡슐화하고 컴포넌트의 외부 환경 안에서 전체적으로 대체 가능한 시스템의 모듈이다.
컴포넌트기반개발의 가장 큰 장점은 재사용입니다. 기존의 부품을 재사용함으로서 개발기간과 개발비용을 줄일 수 있게 됩니다

-

-

객체지향언어에 의한 컴포넌트기반개발에 있어서 가족은 컴포넌트, 가족의 구성원은 객체에 해당합니다.

9장 관계

1. 의존관계(Dependency)

하나 이상의 모델요소들이 그들의 명세와 구현을 위해 다른 모델요소들을 요구하는 관계이다. 이것은 의존 요소들의 의미가 완벽해지기 위해서 공급자 요소에 의미적으로 또는 구조적으로 의존한다는 것을 의미한다.

의존관계는 아래 그림과 같이 의존하는 쪽에서 의존되는 쪽으로 향하는 점선으로 된 화살표 기호로서 표현됩니다.

타겟은 자신에게 의존하는 소스를 알 필요는 없지만 소스들로 추적은 가능해야 합니다. 의존관계는 바로 이러한 영향을 받는 소스 클래스들을 추적하기 위해 존재합니다.

의존관계에는 관계가 갖는 의미의 차이에 따라 사용(usage), 추상(abstraction), 허용(permission)과 같은 특수한 종류의 의존관계가 존재한다. 또한 추상에는 실현(realization)과 Manifestation 과 같은 특수한 종류의 관계로 존재하고, 실현에는 구현(implementation)과 대체(substitution)와 같은 특수한 종류의 관계가 존재한다.

2. 사용(Usage) 관계

사용관계는 일반적인 의존관계의 의미를 나타내는 것으로, 의존하는 모델요소의 명세나 구현을 위해 의존되는 모델요소를 사용하는 관계를 나타냅니다. 다른 종류의 의존관계들과 분명하게 구분해서 사용하는 경우에만, <<use>> 키워드를 사용해서 표현합니다. 그림 9.2 에서 Shape 의 draw 오퍼레이션을 명세하는데 입력매개변수로 Graphics 타입의 g 를 사용합니다. Graphics 의 이름을 Graphic 으로 바꾼다면 Shape 의 draw 오퍼레이션의 Graphics 도 Graphic 으로 바꾸어야 합니다.

3. 추상(Abstraction)

추상관계란? 같은 개념에 대해 서로 다른 추상레벨들 또는 서로 다른 관점들에서 작성된 둘 이상의 모델요소들을 연결하는 관계이다.

추상관계에는 사용되는 경우에 따른 의미적 차이를 반영할 수 있도록 <<trace>>, <<refine>>, <<derive>>와 같은 세 개의 스테레오타입을 갖습니다.

<<refine>>은 추상레벨이 높은 모델요소를 구체적으로 정제한다는 의미를 갖습니다.
<<trace>> 유스케이스와 유스케이스를 실현하는 분석클래스들과 같이 서로 다른 모델 요소들 사이에 추적가능하도록 만든다는 것을 의미합니다.
<<derive>>는 계산을 통해 타겟으로부터 소스를 얻을 수 있다는 의미를 갖습니다.

4. 실현(Realization)

실현은 소스가 되는 모델요소들의 집합이 명세가 되는 타겟 모델요소들의 집합을 구현하는 관계입니다. 실현관계는 <<realize>> 키워드를 갖는 의존관계로 표현합니다. 실현관계에는 구현과 대체라는 특수한 관계가 존재합니다.

5. 대체(Substitution)

대체는 런타임시에 상속이 아닌, 인터페이스를 구현하거나 동일 타입의 포트를 가지고 소스 인스턴스들이 타겟 인스턴스들을 대체할 수 있는 관계입니다. 대체관계는 <<substitute>> 키워드를 갖는 의존으로 표현합니다. 그림 9.4는 SizableWindow의 인스턴스들이 Window의 인스턴스들을 런타임 시에 대체할 수 있다는 것을 나타냅니다.

-

6. 허용(Permission)

허용관계는 타겟이 소스에게 자신이 포함하고 있는 모델요소들에 접근할 수 있도록 허용하는 관계를 나타냅니다. 허용관계는 <<permit>>라는 키워드를 갖는 의존관계로 표현합니다. 그림 9.5 는 Executive 는 Employee 의 name 이나 mobilePhone 을 액세스 할 뿐 아니라, 가시성이 private 으로 설정된 homePhone 이나 salary 까지도 액세스 할 수 있다는 것을 나타냅니다.

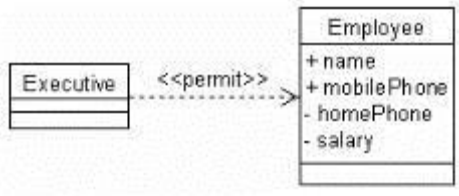


그림 9.5 허용관계 표현

-

7. 연관(Association)

- 인스턴스들 사이의 구조적 관계
- 인스턴스들을 참조하는 값을 가진(tuple)의 집합을 기술한다. 하나의 링크는 연관된 타입들의 개개 인스턴스들의 값으로 구성된 하나의 튜플이다.

10장 상태

1. 이벤트(Events)

객체의 서비스 수행을 요청하는 일들과 시간의 경과, 조건의 변경

2. 이벤트 종류

오퍼레이션을 직접적으로 호출하는 이벤트를 오퍼레이션호출이벤트라고 하고, 시그널을 전송하는 이벤트를 시그널전송이벤트라고 합니다. 시간 경과를 나타내는 이벤트를 시간 이벤트(Time Event)라고 하고, 외부조건의 변경을 변경이벤트(Change Event)라고 합니다.

3. 오퍼레이션호출이벤트

오퍼레이션호출이벤트는 직접적으로 객체의 오퍼레이션을 호출할 때 발생하는 이벤트입니다. 오퍼레이션호출이벤트는 객체의 상태를 변화시킬 수도 있고, 상태를 변화시키지 않을 수도 있습니다. 객체의 상태를 변화시키는 이벤트가 상태 머신에 표현됩니다. 이벤트 이름은 오퍼레이션 이름과 오퍼레이션의 매개변수들을 사용해서 작성합니다.

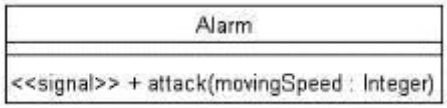
4. 시그널전송이벤트

시그널전송이벤트에 의해 송신 객체에 시그널이 보내집니다. 시그널은 화재경보장치 등에서 보내는 경보음과 같이 비동기적으로 전달되는 신호를 나타냅니다. 시그널은 <<signal>> 키워드를 갖는 분류자 기호로 표현합니다. 그림 10.1는 외부침입이 발생했다는 신호를 표현한 것으로 속성으로 침입자의 이동속도를 갖습니다.

그림 10.1 시그널 표현

시그널전송이벤트는 시그널 이름을 이벤트이름으로 사용하고, 시그널 속성을 이벤트의 매개변수로 사용해서 'attack(10)'과 같이 작성합니다.

객체의 행위는 오퍼레이션으로 명세되고, 객체는 오퍼레이션호출이벤트에 대응하는 오퍼레이션을 수행합니다. 시그널전송이벤트에 대응하는 행위는 <<signal>> 키워드를 갖는 오퍼레이션 기호로 나타냅니다.



-

- 그림 10.2 시그널전송이벤트에 대응하는 행위 표현

-

5. 변경이벤트

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

6. 시간이벤트

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

-

7. 상태 머신(State Machines)

상태머신다이어그램은 상태머신을 명세하는 다이어그램입니다. 상태머신은 객체의 상태를 명세하기 위해서도 사용되지만 객체가 수행하는 역할들을 명세하는 인터페이스의 상태를 명세하기 위해서도 사용될 수 있습니다.

8. 프로토콜상태머신(Protocol State Machine)

인터페이스의 상태머신을 객체의 상태머신이 지켜야 하는 규약(계약)이라는 의미에서 프로토콜상태머신이라고 합니다.

인터페이스는 명세로서 내부 구현을 갖지 않기 때문에 구현부분에 해당하는 액션들을 표현할 수 없습니다. 따라서 프로토콜상태머신에 표현되는 상태들은 진입액션과 탈출액션과 액티비티를 갖지 않습니다. 또한 상태를 저장할 수 없기 때문에 얕은이력과 깊은이력을 가질 수 없습니다.

9. ProtocolConformance 관계

. ProtocolConformance 는 일반화관계나 실현관계에 사용되는 프로토콜상태머신들 사이의 관계로 상위프로토콜상태머신의 명세 내용을 하위프로토콜상태머신이 따라야 합니다.

10. 상태머신 확장

상태머신은 일반화 가능하고, 하위 상태머신은 상태나 전이를 추가하거나 재 정의함으로

서 상위 상태 머신을 확장합니다. 상태머신은 확장을 고려해서 상태나 전이가 확장될 수 있는지의 여부를 명세해야 합니다. 다른 상태머신에서 상태나 전이가 확장되지 않는다는 것을 표현하기 위해서 그림 10.7 과 같이 {final}이라는 키워드를 사용합니다.

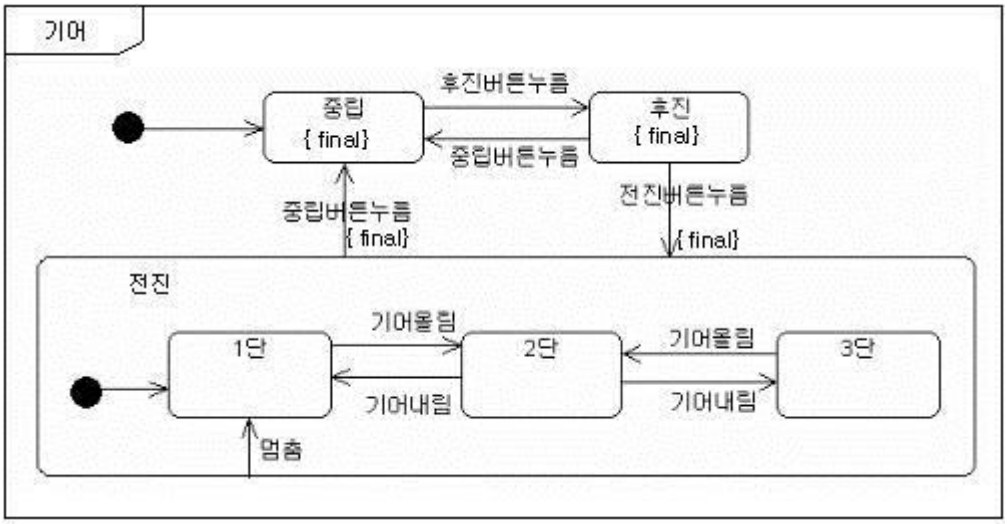


그림 10.7 {final}이 사용된 상태머신 다이어그램

확장하는 상태머신은 {extended}라는 키워드를 사용합니다.

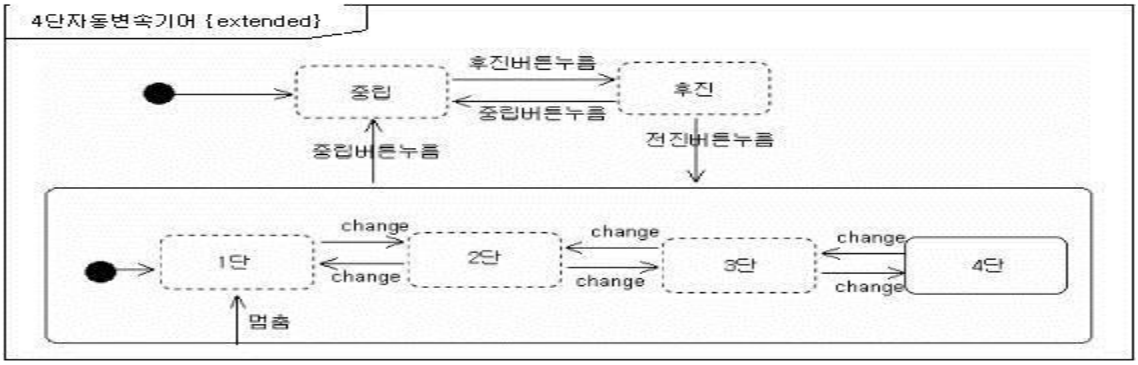


그림 10.8 기어를 확장한 4 단자동변속기어의 상태머신 다이어그램

확장관계나 실현관계를 표현하기 위해서 상태머신은 <<statemachine>> 키워드를 갖는 분류자로 표현할 수 있습니다.

11. 상태

상태란? 몇몇 불변 조건(invariant condition)들이 유지되는 동안의 상황을 모델링하는 것이다.

객체는 대응해야 하는 이벤트가 발생되기를 기다리는 정적인 상태와 액티비티를 수행하고 있는 동적인 상태를 가질 수 있습니다. 일반적으로 정적인 상태는 Connect 나 Connected 와 같이 표현합니다. 자신의 상태를 수동적으로 표현할 때는 과거형을 사용합니다. 동적인 상태는 Connecting 과 같이 액티비티가 수행하고 있음을 나타내기 위해서 진행형으로 표현합니다.

12. 상태종류

객체의 상태는 다른 상태를 포함하는지의 여부에 따라 단순상태와 복합상태로 구분됩니다. 단순상태는 다른 상태를 포함하지 못하는 것이고, 복합상태는 다른 상태를 포함하는 것으로 포함되는 상태를 하위상태라고 부릅니다.

상태는 액티비티 기호와 동일하게 모서리가 둥근 사각형 기호를 사용합니다. 그림 10.13 에서 냉방중이라는 진행형으로 작성된 상태이름을 통해 냉방을 위한 어떤 액티비티가 계속적으로 수행되고 있다는 것을 알 수 있습니다.

그림 10.13 상태 표현

 # 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 States)

종료상태는 상태 머신의 모든 전이가 완료되었다는 것을 의미합니다. 복합상태에서 종료상태가 사용되고, 종료상태로 전이가 되었다는 것은 자신이 포함하는 모든 상태들의 전이가 완료되었다는 것을 의미하고, 해당 상태에서 빠져 나오게 됩니다. 종료상태는 액티비티 다이어그램의 종료 점과 동일한 기호를 사용합니다.

13. 전이(Transitions)

객체의 한 상태에서 이벤트에 의해 다른 상태로 이동하는 것을 전이라고 합니다. 객체는 상태를 변화시킬 수 있는 이벤트가 발생하고, 명세된 경계조건변화를 만족시킬 때 전이

이전의 상태에서 명세된 액션들을 수행하고, 전이되는 상태로 들어갑니다. 전이는 '이벤트이름(인자리스트)[경계조건]/액션리스트'와 같은 형식으로 작성됩니다.

14. 복합 상태(Composite States)

복합상태는 다른상태를 포함하는 상태입니다.

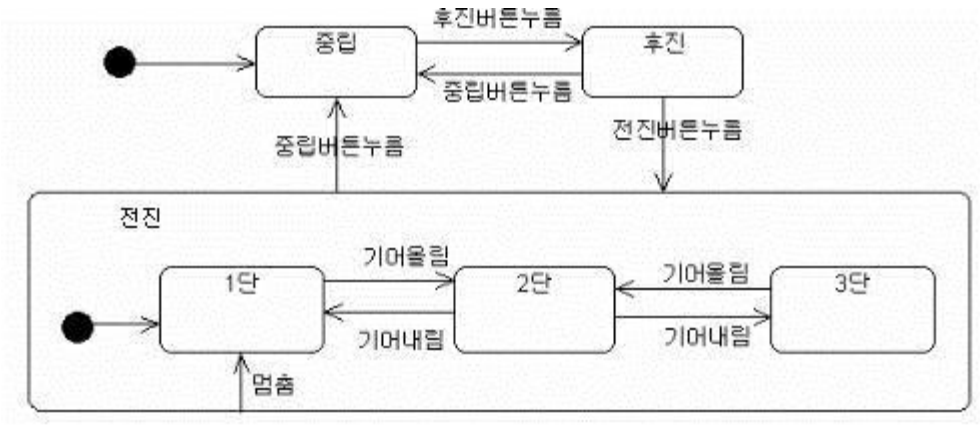


그림 10.19 기어의 상태머신

그림 10.19 에서 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.

15. 서브머신 상태(Submachine States)

컴포넌트기반개발과 같이 부분들에 해당하는 상태 머신.

서브머신 상태는 다른 많은 서브머신에서 참조되어 재 사용될 수 있기 때문에 특정 서브머신에 종속되도록 표현하면 안됩니다. 컴포넌트의 인터페이스와 같이 외부 상태에서 전이될 수 있는 통로만을 제공하면 됩니다. 서브머신 상태는 서브머신 상태로 들어올 수 있도록 진입 점(entry point)을 제공하고, 서브머신 상태에서 나갈 수 있도록 탈출 점(exit point)을 제공합니다.

그림 10.24 는 서브머신 상태가 상태 머신에서 참조되는 방법을 설명합니다. 서브머신 상태가 상태머신에서 사용될 때의 역할은 에러를 처리하는 것이기 때문에 HandleFailure 로 이름이 작성되었습니다.

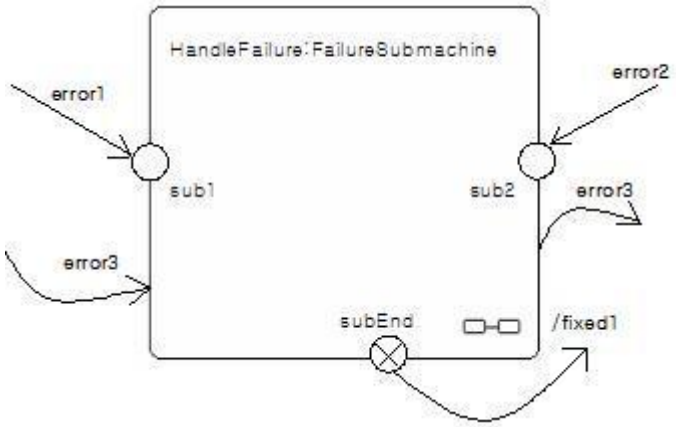


그림 10.24 상태머신에서 참조되는 서브머신상태 표현

11장 배치

1. 액티브 클래스(Active Classes)

액티브 클래스의 객체들은 능동적인 객체들로서 객체가 생성되면서 자신의 행위를 시작하고, 행위가 완료되거나 자신의 제어흐름을 가지고 있는 외부적 객체에 의해 중단될 때까지 멈추지 않고 실행합니다. 액티브 클래스의 객체를 액티브 객체라고 합니다. 적합한 의미를 갖는 우리말로는 '활성클래스'와 '활성객체'가 있습니다. 액티브 클래스는 그림 11.1 과 같이 양쪽에 수직선을 갖는 분류자 기호로 표현합니다.

그림 11.1 액티브 클래스 표현

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

2. 배치(Deployment)

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

3. 아티팩트(Artifacts)

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

그림 11.2 아티팩트 표현

4. Manifestation 관계

Manifestation 은 추상(Abstraction)관계의 특수한 형태로, 목록이라는 의미를 가지고 있으며, 아티팩트에 포함되는 모델요소들을 나타내기 위한 관계입니다. Manifestation 관계는 그림 11.3 과 같이 <<manifest>>라는 키워드를 갖는 의존관계로 표현합니다.

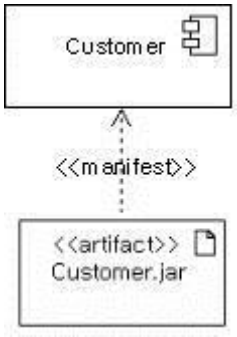


그림 11.3 Manifestation 관계

5. 표준스테레오타입

아티팩트는 개발되는 시스템 범위와 관련된 물리적 파일을 나타내는 <<file>>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.

6. 노드

노드는 실행시간에 존재하고, 컴퓨터자원(computational resource)을 나타내는 물리적인 요소이다. 노드는 일반적으로 컴포넌트가 배치되어지는 프로세서나 장치를 나타낸다. 노드들은 네트워크 구조를 정의하기 위한 커뮤니케이션 경로를 통해서 상호 연결되어질 수 있다.

노드는 육각형기호로 표현합니다. 그림 11.4는 AppServer라는 노드의 인스턴스를 표현하고 있습니다.

그림 11.4 노드 인스턴스 표현

7. 커뮤니케이션경로(Communication Paths)

노드들 사이의 연결은 커뮤니케이션경로라는 특별한 연관관계에 의해서 표현됩니다. 그림 11.5 는 DBServer 가 여러 개의 AppServer 와 메시지를 교환한다는 것을 표현하고 있습니다.

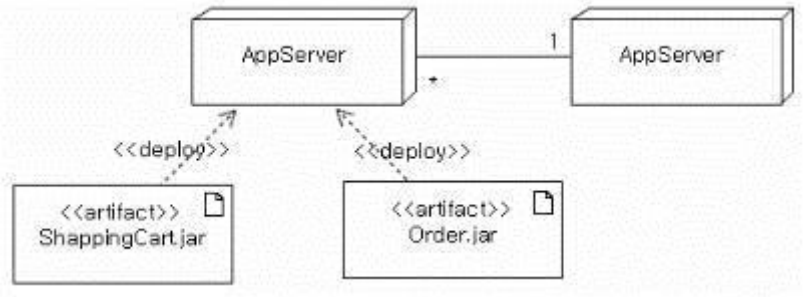


그림 11.5 커뮤니케이션경로관계 표현

8. 장치(Devices)

아티팩트가 실행되기 위해 배치될 수 있는 프로세싱 기능을 갖는 물리적 컴퓨터 자원이다.

장치는 특별한 형태의 노드로, 다른 장치들을 포함할 수 있으며 <<device>>라는 키워드를 갖는 노드 기호로 표현할 수 있습니다.

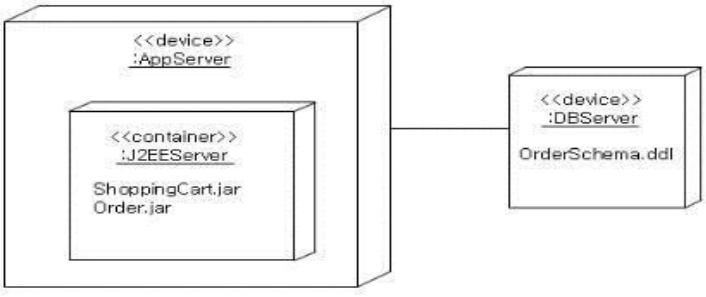


그림 11.6 장치 표현

9. 배치관계

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 그림 11.7 과 같이 노드에 아티팩트를 직접 포함하는 것과 그림 11.8 과 같이 노드에 아티팩트 이름을 리스트하는 방법이 있습니다. 또한 그림 11.5 와 같이 <<deploy>> 키워드를 사용해서 노드 외부에 표현할 수 있습니다.

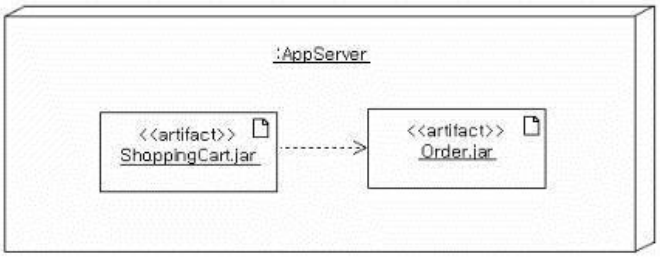


그림 11.7 아티팩트를 직접적으로 포함해서 배치관계 표현

그림 11.8 아티팩트를 리스트로 포함해서 배치관계 표현

배치명세

배치명세는 컴포넌트의 실행 시에 결정되는 변수들을 속성의 집합으로 명세한 것입니다. 일반적으로 배치명세는 특별한 종류의 컨테이너를 대상으로 합니다. 배치 명세 프로퍼티를 구현하거나 구체화하는 아티팩트를 배치 디스크립터 라고 합니다. 만약 특별한 하드웨어나 소프트웨어 기술에 배치하기 위해 공통적으로 사용해야 하는 속성들이 있다면 <<deployment spec>>이라는 키워드를 사용해서 명세할 수 있습니다.

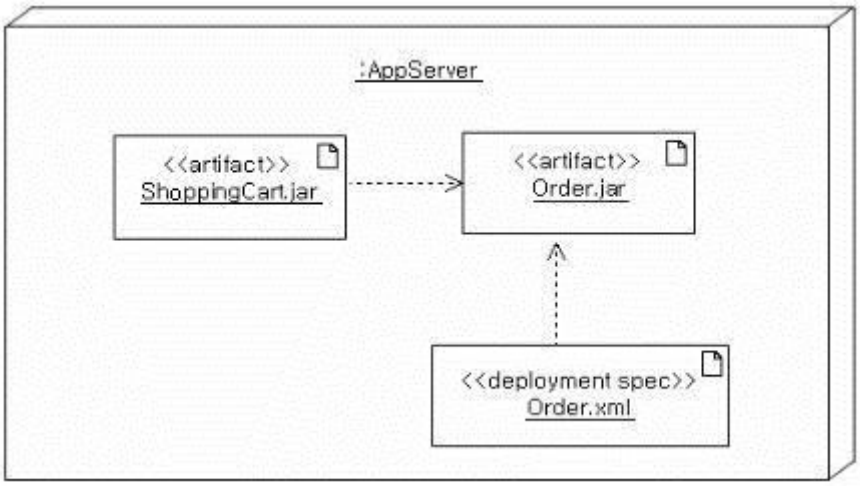


그림 11.9 배치명세 표현

- **참고문헌**

생각하며 배우는 UML 2.0(2004), 영진닷컴, 김현남 저