소프트웨어공학개론

- Introduction to OOAD using UML tools -

<A반 Team 7 >
200711423 김용호
200711436 서영주
200711451 유예근
200711458 이기석

- 목 차 -

1. 객체지향의 기본 개념

객체지향의 기본 사상은 복잡한 매커니즘의 현실 세계를 인간이 이해하는 방식으로 시스템에 적용시켜 보자는 것.

객체지향 기술에는 **객체(Object)**와 객체들의 범주를 나타내는 **클래스(Class)**, 그리고 객체간의 상호작용을 위한 **메시지(message)**가 있다.

1) 객체(Object)란?

객체 모델링에서 가장 기본적인 단위. 우리의 인식 영역 모든 것에 적용할 수 있는 개념

<table>
<tr><td colspan="5">객 체</td><td rowspan="3"><객체의 특징>
1.상태갖음
2.한클래스의 Instance
3.행동에 의해 특성 표현
4.이름 갖음
5.다른 객체와의 연관성</td></tr>
<tr><td>유형객체</td><td colspan="4">무형객체</td></tr>
<tr><td rowspan="2">실제행위를 능동적으로 수행하는 물리적인 속성을 가진 것</td><td>임무객체</td><td>사건객체</td><td>상호 작용객체</td><td>명세서 객체</td></tr>
<tr><td>역할/ 행위 수반</td><td>행위호출</td><td>객체간 통합/ 분기 결과 생성</td><td>행위의 참고 및 표준</td></tr>
</table>

※ 객체의 예

(1) 객체지향의 배경

-. 최근 발표

-. 개발의 문제점을 해결해줄 많은 장점을 보유

-. 요구사항 변경을 수용(유연성과 적응력 필요)

-. (데이터와 행위 ➔ 객체를 정의 내리고 객체를 추상화시키는 작업)

-. 기존의 데이터와 행위가 분리되었던 개발 방법의 복잡성과 통합의 어려움을 극복하려는데 있음.

2) 객체(Object)는 property, behavior, Identity들로 구성 되어 있다.

a) Property(attribute)

명사형으로 표현 되는 변수나 상수 데이터 이름을 말한다.
사람으로 치면 나이, 키 등을 의미하는 어떤 값.

Dave
Age: 32
Height: 6' 2"

Gary
Age: 61
Height: 5' 8"

b) Behavior

객체(Object)의 수행하는 기능이다. "What the Object can do"

c) Identity

사람 개개인이 이름을 가지고 있듯, Object들도 구별 할 수 있는 Unique한 식별자, 즉 Address나 ID를 가지고 있다.

2) 클래스(Class)

객체(Object)들의 properties 와 behavior를 결정한다. Class의 instance 들이 객체이다.
예) 각각 자동차는 Car Class의 Object(instance)

Class Car
Attributes
- ❑ Model
- ❑ Location
- ❑ #Wheels = 4

Operations

모든 객체는 반드시 클래스를 통해서만 정의될 수 있음.
(클래스는 여러 객체가 가지고 있는 공통적인 속성(attribute)과 메소드(method)를 가지고 있는 클래스가 먼저 정의되면, 이 클래스를 통해서 다시 객체가 정의)
예) 회사라는 조직 / 많은 사원/

속성(= 성명, 사번, 부서, 급호 등),
메소드(= 급여계산이나 업무처리)
클래스(= 사원) ---> 속성과 메소드를 갖음
이과장, 김대리 ---> 객체가 만들어짐
한 클래스에 속하는 각각의 객체를 ---> 인스턴스(instance)

- 클래스의 종류 : Super class / Sub class / Abstract class / Meta class

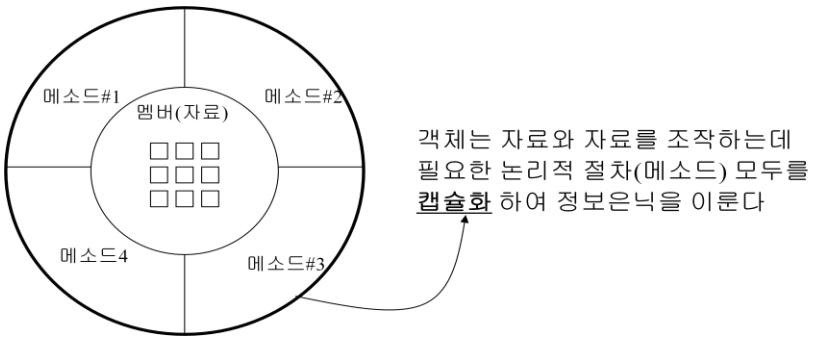
- **추상화(abstraction)**

현실세계의 사실을 그대로 객체로 표현하기 보다는 문제의 중요한 측면을 주목하여 상세 내역을 없애나가는 과정 ("프로세스 추상화(process abstraction)" 프로그램에서 자주 나오는 상세한 부분을 함수로 묶어 호출하게 함으로써 이해하기 쉽고 간단한 모양으로 만듦) "데이터 추상화(data abstraction)" integer, real, date 같은 데이터 타입을 개발자는 정수 연산의 값의 범위나 진법, 보수 형태의 치환 등을 일일이 프로그램 해 넣을 필요 없이 그저 선언만 함.

추상화는 복잡한 프로그램을 간단하게 해주고 분석의 초점을 명확히 함

- **캡슐화(encapsulation)**

객체의 상세한 내용을 객체 외부에 철저히 숨기고 단순히 메시지만으로 객체와의 상호작용을 하게 하는 것을 캡슐화(encapsulation)라고 하고 다른 말로 정보 은닉(information hiding)이라고 한다. 즉, 캡슐화는 추상화와 거의 같은 개념이지만 추상화를 지원하며 보다 구체적이고 제한적이다. (클래스를 선언하고 그 클래스를 구성하는 객체에 대하여 "public" 또는 "private" 등으로 정의해주면 "public"으로 정의된 함수 또는 데이터는 외부에서 사용이 가능하며, "private"으로 선언된 경우는 외부에서 제어할 수 없고 내부에서만 사용)



캡슐화는 객체의 내부구조와 실체를 분리함으로써 내부의 변경이 소스 프로그램에 미치는 영향을 최소화한다. (유지보수도 용이)

- **상속성(Inheritance)**

상속성(inheritance)는 객체기술의 가장 핵심이 되는 개념으로 프로그램을 쉽게 확장할 수 있도록 해주는 강력한 수단이 된다. 앞의 두 개념은 객체지향이 아닌 개발 방법에서도 흉내를 낼 수 있으나 이것은 **객체지향 언어와 개발 방법만의 특성**이다. (사원이 정규직 사원, 계약직 사원으로 구분될 때 이때 사원이라는 정보는 정규직 사원과 계약직 사원 모두 클래스의 속성을 그대로 물려받는다. 여기에 추가하여 고유의 속성을 정의하게 되는 것이다. **사원은 클래스 계층에서보면 수퍼클래스(super class)가 되고 정규직과 계약직은 서브클래스(sub class)가 된다**. 이렇게 수퍼클래스와 서브클래스간의 관계가 객체지향의 상속성의 개념)상속의 효과는 클래스를 체계화할 수 있으며, 기존의 클래스로부터 확장이 용이하다는 것이다. 함수와 변수를 서브클래스에서는 따로 정의하지 않고 상위의 클래스의 내용에다 추가적인 특성을 덧붙이기만 하면 되므로 매우 효율적이다. 또한 공통의 특성을 서브클래스마다 반복적으로 기술하지 않고 한번만 기술하기 때문에 중복을 줄여 준다.

- **다형성(polymorphism)**

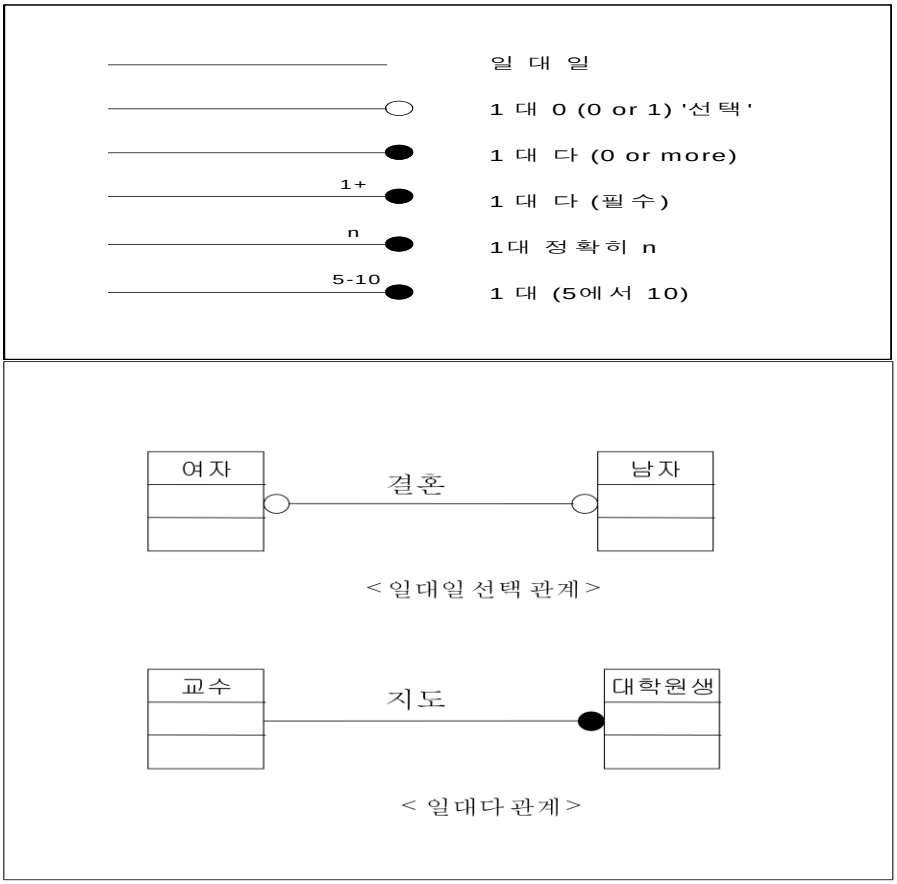
  복수개의 메소드가 구현 내용은 다르나 같은 개념의 일을 하는 경우

- **관계성**

  객체와 클래스와의 상호 연관관계(연관/분류/집단/일반)

  관계성 : 객체와 클래스 상호간의 연관관계를 표현하는 방식

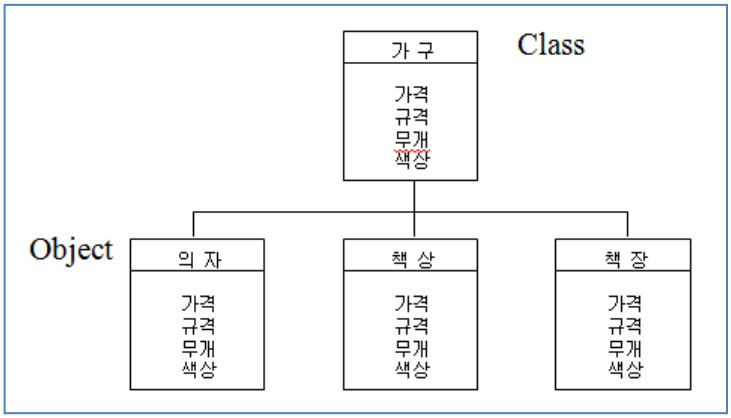
  ※ 표현 도구 (E-R다이어그램)



3) 메시지(message)

홀로 존재하는 객체는 아무런 의미가 없다.
(무언가를 실행하기 위해서 다른 객체를 필요로 함)
(상호작용은 메시지(message)를 통해서 이루어짐)
메시지는 세 부분(메시지를 받는 **수신자 객체(receiver)의 이름** /
수신자가 수행할 **메소드(method)의 이름** / 메소드 수행시 전달되는 **인자(argument)**로 구성)
(객체지향 프로그래밍에서 메소드(method)는 메시지 /
일반적으로 함수(function)와 같은 개념)

❑ 객체지향사고의 예



❑ 구조적 프로그래밍과 객체지향 프로그래밍 비교

	구조적 프로그래밍	객체지향 프로그래밍
사용언어	C나 Pascal 같은 절차적 언어 사용	C++, JAVA와 같은 객체지향 언어 사용
파라미터 전달	프로시저 호출	객체에 메시지 전달
캡슐화	데이터와 프로시저 분리 가능	데이트와 프로시저의 캡슐화
모듈화	함수 단위	객체 단위
실세계 표현	논리적인 모형고가 제어의 흐름 이용	객체를 사용

❑ 객체지향 방식의 장점

1) 잘 설계된 디자인 방법은 객체 지향 프로그래밍 언어의 장점을 최대한 발휘한다.
2) 객체 지향 설계는 좀 더 많은 재사용 가능한 뛰어난 코드를 생산한다.
3) 객체 지향 설계는 좀 더 변화에 탄력적인 시스템을 생산한다.
4) 객체 지향 방법론의 모듈화는 시스템 개발팀이 서로 독립적으로 설계의 여러 부분에 참가하여 시스템을 개발하기도 용이 하다.
5) 문제 공간을 이해하는 방법이 자연스럽기 때문에 시스템 분석가, 설계자, 최종 사용자에게도 좀 더 직관적으로 이해될 수 있다.

2. 객체 지향 모델링

1) 객체 모델링 방법론

소프트웨어 개발의 큰 문제점은 복잡성, 요구의 변화, 개발 기간의 시간적인 제약성이다. 이러한 문제 해결에 제안된 기본적인 개념으로 분해 기법과 추상화 기법을 들 수 있다.
분해 기법은 복잡한 문제를 작고 단순한 서브 프로그램들로 나눈다. 나누어진 문제는 전체 문제보다 해결하기 쉽고, 서브 프로그램들을 다시 결합하면 원래 의도한 문제를 해결할 수 있다[8].
추상화 기법은 다양하고 복잡한 개념을 단순 개념으로 사상하는 과정으로 서로 다른 많은 사항들을 무시한 채 같은 것으로 취급함으로써 복잡한 방법을 줄이는 것이다. 객체 지향 방법론은 두가지 개념을 바탕으로 소프트웨어를 개발한다[8].
객체 지향 방법론은 해 영역의 실제에 의미를 부여하고 실제를 객체로 추상화시키는 의미 객체 모델링(정적 모델) 방법과 처음부터 동적 모델을 고려한 객체 추출을 기반으로 하는 객체 모델링 방법으로 나눈다.

(a) 의미 정의에 대한 객체 모델링

정보처리 시스템은 문제 영역 P에 대한 해 영역 S를 양식으로 나타낸 것으로 P⊂S인 관계로 나타낸다. 실세계에서 문제의 의미를 분석하여 기술하고, 양식에 의한 설계를 하여 해를 기술한다. 또한 P는 의미 객체를 포함한 집합으로써 P의 객체들은 해 보다는 문제를 기술하는 개념으로 사용한다.
시스템의 의미 객체는 고객, 주문, 항목, 피고용자들로서 주어진 시스템의 실체들이 된다. S는 문제 영역 P 이외에도 인터페이스, 응용, 기본 객체들을 포함한다.
인터페이스 객체는 사용자와 정보처리 시스템 간의 인터페이스를 설명하고 문제 영역 자체를 설명하지 않는다. 즉 의미 객체에 대한 사용자의 관점에서 기술한다.
응용 객체는 정보처리 시스템에 관한 제어 매카니즘이다. 유도기능을 가지고 개발된 시스템의 기능들을 순차적으로 제어한다. 즉 시스템의 주된 모듈로서 다른 객체들에 대한 메시지와 메뉴를 전달한다[8].
기본 객체는 문제 영역이나 응용되는 독립된 객체이다.
의미 객체 모델을 바탕으로 객체 지향 분석을 하는 과정은 의미 자료 모델을 갖추어야 할 자료의 추상화, 의미 객체의 식별, 집합관계, Classification과 Instantiation, 속성의 승계, 그리고 서브 클래스와 슈퍼 클래스를 정의하여 수행한다[8].
분석 단계는 문제점을 기술함으로 의미 객체별 요구와 양식 중심으로 기술하고 설계 단계는 그 해를 기술한다. 객체 지향 분석은 문제 영역을 모델링하기 위하여 의미 객체의 집합을 식별하여 그 양식을 정하고 시스템 요구에 의해서 관련성을 기술한다. 객체 지향 설계는 해 영역을 모델링하는데 의미 클래스, 응용 및 기본 클래스를 설계한다.
결과적으로 실체를 식별하여 의미 객체로 추상화하고 객체들의 속성과 관련성, 집합관계를 기술하여 의미 자료 모델링을 하고, 여기에 객체 지향 프로그램 언어가 가진 특성, 즉 속성의 외부 서비스, 객체 간의 인터페이스 메시지, Classification 및 행위의 승계, 그리고 정보 은닉과 캡슐화를 추가하여 객체 지향 분석 및 설계를 수행한다.

(b) 객체 모델링

객체 모델링은 문제 분석과정부터 객체를 추상화시켜서 클래스로 정의하고, 관련성을 분석하여 상속성을 정의하는 분석 과정이다[8].

2) 객체 모델링의 단계

객체 지향 분석은 시스템의 시나리오에 나타난 객체를 식별하고 그들 간의 상호 작용을 파악한 다음, 각 객체에 부과된 책임(행위)을 분석한다. 객체의 행위 분석은 인터뷰하기 전부터 시작할 수 있으므로 시스템의 시나리오를 문서로 작성한 이유도 객체 추출을 쉽게 하기 위한 것이다. 여기에서 중점을 두어야 할 내용은 시스템을 개발하게 된 동기가 무엇이며, 동기별로 어떤 사건이 일어날 것인가 하는 점이다. 즉 기술 문서와 함께 객체와 그들의 속성 및 행위를 식별하는 것이 객체 지향 분석의 중요한 과제이다

(1) 분석 단계

분석 단계는 기본 단계로서 분석을 위한 계획서 작성, 행위에 초점을 맞춘 행위 분석, 객체를 추출하고 그 행위의 기술,객체들 간의 관련성 파악,동적 모델링으로 구분할 수 있다 기본 단계는 분석의 계획 단계이다. 시스템의 시나리오를 바탕으로 살펴보면 개념적 모델링, 외부 양식, 구현의 과정을 정하고 시스템의 시나리오에 나타난 목표를 설정하고, 시스템의 범위와 문제 공간의 기술, 분석을 위한 자료의 선정, 그리고 분석 단계의 계획을 세운다. 시스템의 범위와 문제 공간의 기술은 시스템의 환경과 조건을 파악하고 문제 분석의 공간 영역이 어디까지인가를 결정한다. 분석을 위한 자원 식별은 사용자의 파악과 영역을 설명할 수 있는 전문적인 지식 그리고 적당한 참고문서 등을 찾아낸다. 이상과 같이 분석과정의 상위 단계에서 나타난 행위를 설명하고 다음에 연결되는 1 단계부터 4 단계까지의 분석을 위한 계획을 세우고, 모델의 개념을 정립한다[8]. 기본 단계에서 기술되는 문서는 분석계획, 현 시스템의 목표, 요구 정의, 품질 목표, 면담내용의 기술로서 시스템의 시나리오와 비교하여 모순점이 없는가를 평가할 수 있는 문서와 객체 모델링의 전과정에 관한 계획문서이다. 요구정의와 품질목표, 면담 등은 인터뷰와 참조문서를 바탕으로 해서 이루어 지는데 객체들의 활동을 설명하는 시나리오, 에러조건과 처리방법, 모델에 포함된 서비스, 서비스의 수행 책임자인 행위자의 식별의 내용을 포함해야 한다. 시스템의 목표와 계획이 세워지면 개념적인 모델링 단계에 들어가는데 이 과정으로 문제분석, 객체의 추출, 그리고 객체의 관련성 정의에 들어간다[8].문제 분석 단계에서는 시스템의 기능을 정하고 그 기능을 누가 수행하며, 누구의 도움을 받아서 집행할 것인가를 결정한다. 시스템 기술에서 나타난 동기와 사건을 중심으로 야기되는 문제를 파악하고 그 해답 영역을 찾는다. 그리고 나서 해당 영역별로 어떤 활동이 일어날 것인가를 예측하고, 활동할 객체를 선정한다. 선정된 객체는 어떤 역할을 담당할 수 있는가를 분석하여 그 객체의 행위와 속성을 정의한다. 끝으로 행위자의 활동을 중심으로 하여 객체들의 행위를 결정하여 기술 문서를 작성한다. 기술 문서의 내용은 객체와 속성, 그리고 요구자의 행위와 행위자의 서비스를 기술한다. 활동 기술서는 요구자와 행위자의 객체가 수행하는 활동을 의미한다. 객체가 식별되면 요구자의 행위는 행동, 행위자의 행위는 서비스이다.

(2) 활동 기술 단계

객체 추출과 활동의 기술 단계는 문제 분석에서 추출된 모든 객체의 역할을 정하고, 객체의 속성과 행위를 비교 분석하여 미비점과 모순점을 찾아낸다. 그 다음에 객체들을 구분하여 요구자인지, 행위자인지를 식별하고 각각의 행위를 기술한다. 끝으로 트레이스의 방법을 기술하면 객체의 기초적인 모델링은 끝난다. 객체의 속성은 논리적인 특성이기 때문에 객체활동에 관련된 성질을 정의하면 된다. 객체의 행위는 객체의 책임이 무엇인가를 파악하고, 그 객체가 제공할 서비스와 다른 객체에 요청할 서비스를 구분하여 정해나간다. 모든 참여자가 객체로 나타나는데 참여자는 동기를 부여하고 서비스를 요청하는 요구자와 서비스를 수행할 행위자로 나눌 수 있다. 그러나 한 개의 객체가 요청과 수행의 두가지 기능을 같이 가질 수 있다. 이럴 때는 객체 기능은 두 가지로 구분해서 설명할 수 있는 활동을 정의해야 된다. 객체는 요구자와 행위자가 되는데 요구자의 활동은 능동적인 행동을 의미하고, 행위자의 활동은 수동적인 서비스를 의미한다. 본 단계의 활동 기술서에서는 활동 기술서를 구별하여 설명하는 과정이 중요하다.

(3) 관련성 기술 단계

객체의 관련성 기술 단계에서는 객체의 관련성을 추상화에 의한 일반화, 특성화에 의한 집합관계, 그리고 링크와 결합에 의한 연결관계를 찾아서 객체들을 그룹으로 나누는 단계이다.

추상화는 객체에 어떤 기능을 부여할 것인지를 기술함으로써 외부적 관점에서 설명하고자 하는 것으로 구현에 의해서도 그 기능이 변하지 않은 캡슐화를 시도할 수 있다. 또 공통된 특성과 행위를 특성화시켜서 다른 객체에 승계시켜 줄 수 있도록 일반화시키고 추상화된 객체를 새로운 객체(새로운 특성과 행위를 추가시켜서)로 재정의하는 특성화를 통해 보다 차원 높은 추상화를 유도해 낼 수 있다. 즉 추상화된 객체를 일반화와 특성화를 통해서 재차 추상화시킴으로써 승계를 위한 차원 높은 추상화 객체에 대해서 집합관계를 설계할 수 있다.

집성화 관계는 a-part-of나 partwhole로 표현되는 객체의 그룹화를 위한 관계 정의를 하는 분석과정이다. 자원관리의 BM구조와 같이 제안된 방법으로 설계할 수 있는 DB에서 쉬운 예를 찾아볼 수 있다. 링크와 결합을 사용자가 정의하는 내부 레코드의 관계와 DB에서 parent-child 관계와 같이 정의한다. 관계의 비중을 나타내는 차수와 대응수, 그리고 종속성이 정의된다. 객체 모델에서 일반화,집합관계 및 링크와 결합관계는 매우 중요하다.

(4) 동적 모델링 단계

동적 모델링 단계에서는 앞 단계에서 설명한 문제분석, 객체 추출과 행위 기술, 그리고 객체들 간의 관련성을 중요하게 생각하는 분석이 정적 모델링이라고 한다면, 시스템 전체의 생명주기를 모델링하는 것을 동적 모델링이라고 할 수 있다. 동적 모델링에서는 객체의 상태가 변화되어 가는 과정을 중요시한다. 객체의 상태는 기술문서에 나타난 전후 조건들로부터 파악된다. 정적 모델링이 객체의 구조만 관심을 두고, 시간 조건을 배제한 환경에서 설계된다면, 동적 모델링은 시간에 따라서 변하고 시스템의 관점에서 객체를

관찰하게 된다. 따라서 두 가지 모델링의 관점은 서로간에 균형을 유지할 수 있도록 설계되어야 한다. 동적 관점의 설계 내용은 각 객체에 대한 상태를 정의하고 동적 행위를 가진 객체의 상태가 객체의 생명에 어떤 영향을 줄 수 있는가를 정의한다.

3) 여러가지 객체 모델링의 비교 분석

(1) Edwards

분석결과에서 C++로 코딩할 수 있고 정적 및 정적 차원에서 지원할 수 있으나 승계 정의를 고려하지 않았다. 정적 차원의 지원에서는 클래스, 인터페이스, 관계, 기능(관계)이 있고, 동적차원의 지원에서는 다중 객체로 일어난 이벤트를 중심으로 분석하고, 단순 이벤트는 STD에 의해 분석한다. 이벤트의 5개 영역은 객체 생성, 용어의 정의, 클래스 확장, 클래스 재구성, 객체의 재분류 등이 있다.

여기서 클래스의 확장과 재구성, 객체 분류는 객체를 클래스로 구분하는 과정이다.

(2) Page와 Jones의 이벤트 영역 분류

이벤트의 영역 분류를 통해서 객체를 모델링하는 과정은 다음과 같다.

① 객체 생성
② 용어 정의
③ 객체 정의와 분류에 의한 클래스 정의
④ ③ 단계와 반복 순서
⑤ 재분류
⑥ 관계의 재정의
⑦ 서로 다른 객체의 표준화와 동향

(3) Shlaer와 Mellor의 OOA

객체의 상호 관련은 이벤트를 통해서 정의되고 이때 활동을 통해서 이벤트를 생성한다. 전이는 이벤트가 발생할 때 일어난다.

활동을 확장할 때 DFD를 채용하며, DFD는 외부 속성을 표현한다. STD에 의해서 객체의 라이프 싸이클을 결정한다.

(4) Rumbaugh의 OMT

Rumbaugh는 활동 상태와 활동을 구분하여 설명하고 있다. 활동은 전이 가능한 이벤트와 동시에 일어나고, 활동상태는 상태가 추가된 연산과정으로 생각한다. 객체의 상태는 객체가 가지고 있는 다른 객체들 간의 결합으로 모은 집합이다. 활동상태는 전이 가능한 이벤트에서 끝나고, 항상 활동으로 바꿀 수 있다. 그래서 자료 동적 및 정적 성질을 표현하는데 중점을 두고 있다.

객체 지향 분석은 다음과 같이 4가지 과정을 반복해서 이루어진다.

① 객체 모델링
② 동적 모델링
③ 기능 모델링
④ 모델 간의 관계정립

특히 Booch와 Gibson은 서브 시스템을 사용하여 객체를 정의함으로써 설계과정의 단계를 상세하게 추가하고 있다. 즉 확장 가능한 객체를 서브 시스템에 정의하고 그 자료구조를 설계하고 행위의 알고리즘을 설계하면서 새로운 객체를 식별하는 기회를 갖게 한다. 본 연구에서는 Rumbaugh, Booch와 Gibson의 방법을 참조하면서 객체 모델링 방법을 설명하였다.

4) 객체 지향 분석 설계에 대한 비교 분석

가. 객체 지향 분석 및 설계 프로세스

OOAD 프로세스에서는 객체를 의미, 인터페이스, 응용, 베이스/유틸리티 클래스들로 분류하여 나타낸다. 그리하여 OOAD프로세스에서는 클래스로 정제하는 활동을 한다. 이것은 공통된 행위에 대한 클래스의 추상화와 애트리뷰트에 대한 구조를 조사하는 것이다. 또한 설계자의 관점에서는 클래스가 완벽하고 올바르다는 것을 확인할 수 있으며 "메소드에 누락된 것이 있는가," "메소드와 애트리뷰트가 올바른 위치에 있는가"에 대해서 클래스를 정제하는 작업이다. 많은 연구자들에 의해 OOAD 프로세스에 대한 연구가 이루어졌는데 그 중에서 Booch, Coad and Yourdom, Rumbaugh, Bulmn 등이 OOAD프로세스에 대한 많은 부분을 설명하였다. 특히, Bulman은 인터페이스, 응용, 베이스, 유틸리티 객체에 중점을 두고 설명하였다.

나. OOAD 방법

표현은 크게 정적인 관점과 동적인 관점 그리고 표기법에 관한 관점으로 구분할 수 있다. 객체 지향 정적모델은 객체 지향 동적 모델보다 많은 종류의 모델이 있으며, 동적 모델 중에서 구성요소 대부분이 객체 모델에서 행위로 매핑되거나 번역되는 상태 전이도로 구성된다. 표기법에 관한 관점은 관계의 표현을 포함하는 것으로써 대부분의 표현들은 일반화 관계로 지원된다. 집단화 관계도 지원되지만 아직까지 널리 이용되고 있지 않다.

다. OOAD 복잡도 관리

몇몇 연구자들이 방대한 OOAD 복잡도를 관리하기 위한 개념적인 메커니즘을 제공하고 있는데 이 메커니즘의 대부분은 구조적인 복잡도에 대하여 처리하고 있다. 예를 들어 Coad and Youndom's의 서브젝트와 Wirfs-Brock et al의 서브 시스템이 있는데 서브젝트는 어떤 루트 구조나 서브스트락처에서 객체를 캡슐화하는 매카니즘이며 루트로부터 구조를 상속받아 구성되므로 복잡도가 감소한다. 반면에 서브 시스템은 루트구조에 기반을 두지 않고 상위 레벨 기능을 달성하기 위해 구성된 다른 서브 시스템이나 클래스로 이루어져 있다.

라. OOAD 장단점

현재의 OOAD에 접근하는 일반적인 세 가지 방법은 다음과 같다.

(1) 결합에 의한 접근

객체 지향, 기능 지향 그리고 동적 지향 기술을 단독으로 모델 구조, 기능성, 동적행위로 이용하고 다른 모델을 통합하기 위한 방법론을 제공한다. 그러나 이 결합에 의한 접근은 다른 관점을 객체 지향 설계방법으로 통합할 때 내용이 변경될 수 있다.

(2) 적용될 수 있는 접근

기존의 기술을 객체 지향에 이용하거나 객체 지향 기술을 포함하여 확정한다.

(3) 순수한 객체 지향 접근

객체구조, 기능성, 동적행위를 모델화하기 위해 새로운 기술을 이용한다.

OOAD를 지원하기 위한 정형화된 방법은 아직 표준화되어 있지 않으나 지금까지 기술한 OOAD에 대한 장담점은 <표 1>과 같다.

<표1> OOAD 연구의 장.단점

장점	단점
-의미 클래스의 명시 -애트리뷰트와 행위의 명시 -메소드의 배치(Law of Demeter) -일반화와 집단화 구조의 표현과 명시 -정적 뷰의 표현(구조)	-인터페이스, 응용, 시스템, 클래스의 명시 -애트리뷰트, 관련성, 행위 등을 결정 -클래스 배치 -다른 종류의 관계성의 표현과 명시 -관계성을 위한 일치되고, 올바른 의미의 표현(메시지, 페싱, 제어 등) -정적, 동적 모델 통합 -추상화/Granularity의 일치 레벨의 유지

3. UML(Unified Modeling Language)

1)UML(Unified Modeling Language)의 등장

-1990년대에 들어서면서 소프트웨어의 재사용성 및 유지보수를 위한 효과적인 방법을 제시하는 객체지향 개발 방법론이 관심을 끌기 시작함.
-James Rumbaugh의 Object-Oriented Modeling Technique : 분석에 좋은 모델을 제시
-Grady Booch의 Object-Oriented Analysis and Design : 설계에 좋은 모델을 제시
-Ivar Jacobson의 Object-Oriented Software Engineering : 객체의 행위를 분석하는데 좋은 모델을 제시
-서로의 단점을 보완하면서 자신의 방법론을 발전시켰지만 고유의 표기법으로 인해 사용자의 혼란을 야기. 이러한 문제점을 해결하기 위해서 UML의 개발이 추진되었다.

2) UML(Unified Modeling Language)이란?

-시스템 개발 과정에서 객체지향 시스템의 결과물을 명세화하고, 시각화하고, 문서화하기 위하여 사용되는 모델링 언어. 객체를 추출하고 설계하는 방법을 제공함으로써 보다 쉽고 빠르게 작업을 마칠 수 있게 한다.
-James Rumbaugh, Grady Booch, Ivar Jacobson의 표기법 뿐만 아니라 다른 많은 방법론의 우수한 개념을 통합하였다.
-객체지향 방법론에 사용되는 표기법을 통합함으로써 객체지향 분석과 설계 분야에서 표준을 위한 기초를 제공하였다.
-최근에는 대규모 프로젝트를 추진하는데 있어서 가장 중요한 분야로 등장하였고, 그 중요성은 갈수록 증가하고 있다. 실제 UML은 최근에 각종 개발 툴 개발업체들이 지속적으로 업그레이드 하고 있고 새로운 기능들을 추가하고 있다. 대표적으로 Rational Rose사 등이 있다.

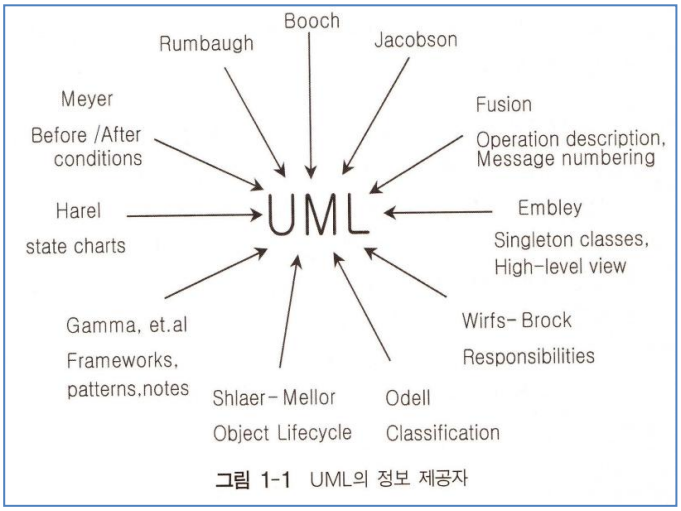


그림 1-1 UML의 정보 제공자

3) UML의 뷰 및 다이어그램

-UML의 다이어그램은 클래스, 인터페이스, 노드, 종속 관계, 일반화 등과 같은 UML의 표현 양식을 그래프 형태로 표현한 것을 말한다.
-다양한 관점에서 시스템을 가시적으로 표현하기 위하여 다이어그램을 사용한다.
-복잡한 시스템의 경우 한 관점에서 전체를 이해하기 어렵기 때문에 시스템의 다양한 국면에 독립적으로 초점을 맞출 수 있게 다수의 다이어그램을 정의할 수 있다.

(1) UML의 뷰

-소프트웨어 아키텍처를 표현하기 위해 다섯 가지의 상호 보완적인 뷰를 이용한다. 각각의 뷰는 구조 모델링(정적인 사항의 모델링)과 행위 모델링(동적인 사항의 모델링)을 포함한다.

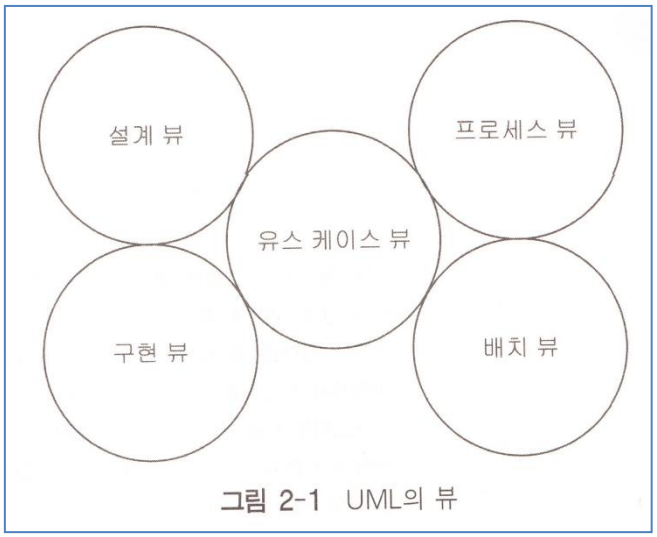


그림 2-1 UML의 뷰

※ 유스 케이스 뷰 : 외부 액터가 인식하는 시스템의 기능성을 설명한다. 사용자, 설계자, 개발자 및 테스터를 위한 뷰이다. 주로 유스 케이스 다이어그램으로 표현되고 사용자가 원하는 시스템의 용도는 유스케이스 뷰에서 다수의 유스케이스로 설명된다. 유스 케이스 뷰는 다른 뷰의 개발을 유도하기 때문에 매우 중요하다. 시스템의 궁극적인 목적은 유스 케이스 뷰에 설명된 기능성을 제공하는 것이다.
※ 설계 뷰 : 시스템의 기능성이 어떻게 제공되는지를 설명한다. 설계자, 개발자를 위한 뷰이다. 유스 케이스 뷰와는 달리 시스템의 내부를 들여다본다. 클래스, 객체, 관계 등과 같은 정적인 구조와 객체가 다른 객체에게 메시지를 전달할 때 발생하는 동적인 협동을 설명한다. 정적인 구조는 클래스 다이어그램과 객체 다이어그램에 표현되며, 동적인 모델링은 상태 다이어그램, 순차 다이어그램, 협동 다이어그램 및 활동 다이어그램에 표현된다.
※ 프로세스 뷰 : 프로세스와 프로세서로 구분되는 시스템의 분할을 설명한다. 시스템의 비기능적인 속성을 다루는 것으로서 자원의 효율적인 사용, 병행 실행 및 비동기 이벤트의 처리 등을 허용한다. 또 병행 실행되는 스레드 간의 통신과 동기화를 다룬다. 개발자

와 시스템 통합자를 위한 것으로 상태 다이어그램, 순차 다이어그램, 협동 다이어그램, 활동 다이어그램과 같은 동적인 다이어그램과 컴포넌트 다이어그램, 배치 다이어그램과 같은 구현 다이어그램으로 구성된다.
※ 배치 뷰 : 시스템의 물리적인 배치를 보여준다. 개발자, 시스템 통합자 및 테스터를 위한 것으로 배치 다이어그램으로 표현된다. 컴포넌트가 물리적인 아키텍처에 어떻게 배치되는가를 보여주는 매핑을 포함하여야 한다.
※ 구현 뷰 : 시스템이 어떻게 구현되는지를 설명해준다.

(2) UML의 다이어그램

-관심있는 UML요소를 조직화하기 위하여 다이어그램을 사용할 수 있다. UML은 9가지의 다이어그램을 정의한다.
-다이어그램의 사용 방법은 크게 두 가지 방식으로 나뉜다. 첫째는 실행 가능한 시스템을 구축하기 위한 모델을 지정하기 위한 것이고, 둘째는 실행 가능한 시스템으로부터 모델을 재구성하기 위한 것이다.
-UML에서 정의하는 다이어그램은 해당 다이어그램에서 가장 많이 등장하는 UML요소에 의하여 이름이 부여된다. 클래스와 관계를 표현하려면 클래스 다이어그램, 컴포넌트를 표현하려면 컴포넌트 다이어그램을 사용한다.
-사용자는 정적인 부분을 표현하기 위해 다음의 네 가지 다이어그램 중 하나를 사용한다.(구조 다이어그램)
-소프트웨어 시스템의 정적인 부분은 클래스, 인터페이스, 컴포넌트, 노드 등의 존재와 배열을 의미한다.
-클래스 다이어그램, 객체 다이어그램, 컴포넌트 다이어그램, 배치 다이어그램

※ 클래스 다이어그램 : 일련의 클래스, 인터페이스 및 협동과 그들간의 관계를 보여준다. 객체지향 시스템의 모델링에서 가장 많이 작성하는 다이어그램이며 사용자는 시스템의 정적인 설계 뷰를 설명하기 위하여 클래스 다이어그램을 사용한다.
※ 객체 다이어그램 : 일련의 객체와 그들간의 관계를 보여준다. 사용자는 클래스 다이어그램에 존재하는 클래스 및 인터페이스 등에 대한 자료구조 등을 설명하기 위하여 객체 다이어그램을 사용한다. 클래스 다이어그램과 마찬가지로 시스템의 정적인 설계 뷰 혹은 정적인 프로세스 뷰를 설명하지만 보다 사실적인 관점에서 설명한다.
※ 컴포넌트 다이어그램 : 일련의 컴포넌트와 그들간의 관계를 보여준다. 사용자는 시스템의 정적인 구현 뷰를 설명하기 위하여 컴포넌트 다이어그램을 사용한다. 컴포넌트는 하나 이상의 클래스, 인터페이스 또는 협동과 대응되기 때문에 컴포넌트 다이어그램은 클래스 다이어그램과 관련이 있다.
※ 배치 다이어그램 : 일련의 노드와 그들간의 관계를 보여준다. 사용자는 아키텍처의 정적인 배치뷰를 설명하기 위하여 배치 뷰를 사용한다. 노드는 하나 이상의 컴포넌트를 포함하기 때문에 배치 다이어그램은 컴포넌트 다이어그램과 관련이 있다.

클래스 다이어그램 – 객체 다이어그램
컴포넌트 다이어그램 – 배치 다이어그램

-사용자는 동적인 부분을 표현하기 위해 다음의 다섯 가지 추가적인 다이어그램을 사용한다.(행위 다이어그램)
-소프트웨어의 동적인 부분이란 변화하는 부분을 표현하는 것으로써 시간에 따른 메시지의 흐름이나 네트워크를 통한 컴포넌트의 물리적인 이동 등을 의미한다.
-유스 케이스 다이어그램, 순차 다이어그램, 협동 다이어그램, 상태 다이어그램, 활동 다이어그램

※ 유스 케이스 다이어그램 : 일련의 유스 케이스와 액터 및 그들간의 관계를 보여준다. 사용자는 시스템의 정적인 유스 케이스 뷰를 설명하기 위하여 유스 케이스 다이어그램을 사용한다. 시스템의 행위를 모델링하고 구성하는 과정에서 가장 중요한 다이어그램이다.
※ 순차 다이어그램 : 메시지의 시간 순서를 강조하는 다이어그램이다. 일련의 객체와 이들 객체간에 송, 수신되는 메시지를 보여준다.
※ 협동 다이어그램 : 메시지를 주고 받는 객체의 구조적인 구성을 강조하는 다이어그램이다. 일련의 객체와 이들 객체간의 링크 및 이들 객체간에 송, 수신 되는 메시지를 보여준다.
순차 다이어그램과 협동 다이어그램은 의미적으로 동등하다. 서로 변환 가능하며 UML이 이런 동등한 다이어그램을 제공하는 이유는 동시에 하나 이상의 관점에서 문제에 접근할 수 있도록 하기 위함이다.
※ 상태 다이어그램 : 상태 머신, 상태, 변환, 이벤트 및 활동을 보여준다. 인터페이스, 클래스 혹은 협동의 행위를 모델링하는 과정에서 가장 중요한 다이어그램이다.
※ 활동 다이어그램 : 시스템 내의 활동간의 흐름을 보여준다. 일련의 활동, 활동간의 순차적인 흐름 혹은 분기적인 흐름, 흐름에 관련되는 객체를 보여준다.
상태 다이어그램과 활동 다이어그램은 의미적으로 동등하다. 정보의 손실 없이 서로 변환 할 수 있다.

유스 케이스 다이어그램
순차 다이어그램 = 협동 다이어그램
상태 다이어그램 = 활동 다이어그램

4) 클래스

-동일한 애트리뷰트, 오퍼레이션, 관계 및 의미를 공유하는 일련의 객체 집합을 설명.
-클래스와 객체는 시스템에 무엇이 존재하는지를 모델링하며, 이들간의 관계는 이들이 어떻게 구조화 되어있는지를 보여준다.
-클래스는 하나 이상의 인터페이스를 구현한다.
-UML은 직사각형으로 클래스에 대한 그래픽을 제공한다. 클래스에 대한 그래픽 표현은 추상화의 가장 중요한 특성인 클래스의 이름, 애트리뷰트 및 오퍼레이션을 강조할 수 있도록 세 개의 구획으로 구성된다.

-클래스의 모든 애트리뷰트와 오퍼레이션을 한번에 전부 나타낼 필요는 없다. 사용자의 목적에 관련되는 애트리뷰트와 오퍼레이션만을 선택하여 표현하면 된다.

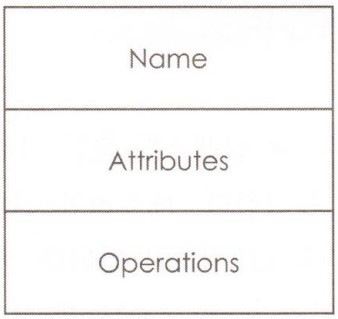


그림 3-1 클래스

(1) 클래스의 이름

-다른 클래스와 구별되는 이름을 가져야 한다. 일반적으로 짧은 명사나 명사구로 이루어진다.

-클래스의 이름만을 쓰는 경우는 단순 이름이라고 하며 패키지의 이름을 붙이는 경우는 경로 이름이라고 한다

(2) 애트리뷰트(변수)

-클래스의 모든 객체가 공유하는 어떠한 특성을 표현한다.

-일반적으로 클래스 내부의 변수로 데이터 혹은 상태의 추상화이다.

(3) 오퍼레이션(함수)

-다른 객체가 요청할 수 있는 서비스를 구현한다.

-일반적으로 클래스 내부의 함수로 객체에 대하여 수행할 수 있는 일에 대한 추상화이다.

5) 관계

-객체지향 모델링에서 대부분의 클래스는 다양한 방식으로 상호 협동한다. 따라서 시스템을 모델링할 때는 시스템의 범위를 결정하여야 할 뿐만 아니라 시스템 구성요소간의 관계도 모델링하여야 한다.

-객체지향 모델링에서는 종속, 일반화, 연관의 세 가지 중요한 관계가 존재한다.

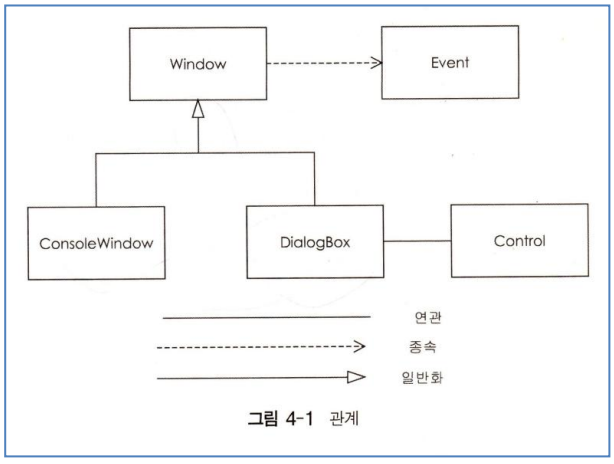


그림 4-1 관계

(1) 종속

-클래스간의 사용 관계를 표현한다. 사용되는 클래스로 향하는 방향성을 갖는 점선으로 표현된다. UML에서 종속은 대부분의 경우 클래스간에 적용되지만 노트 혹은 패키지 등에도 적용된다. 모델 내 종속이 많아서 각각을 구분할 필요가 있을 때에는 종속에 이름을 부여할 수도 있다.

(2) 일반화

-일반적인 클래스(수퍼클래스)와 세부적인 클래스(서브클래스)간의 관계이다. 일반화는 "is-a-kind-of" 관계라고도 한다. 부모-자식 관계를 표현할 때 일반화를 사용한다. 자식에서 부모로 향하는 방향성을 갖는 실선으로 표현된다. 실선의 끝에는 삼각형의 화살표를 붙인다.

-일반 상속, 다중 상속 등을 나타내는데 쓸 수 있다.

(3) 연관

-객체와 객체를 연결하는 구조적인 관계. 동일한 클래스 혹은 서로 다른 클래스를 연결하는 실선으로 표현된다.

-연관은 이름, 역할, 다중성, 집단화 등과 같은 네 가지의 추가적인 표현 사항을 갖는다.

※ 연관의 이름 : 관계의 특성을 설명한다. 이름에 방향을 지시하는 삼각형을 붙임으로써 이름을 읽고자 하는 방향을 나타낼 수 있다.

※ 연관의 역할 : 사용자는 역할 이름을 사용하여 연관에서 클래스가 수행하는 역할을 명시적으로 지정할 수 있다. 하나의 클래스는 서로 다른 연관에서 동일한 역할을 수행할 수도 있고, 다른 역할을 수행할 수도 있다.

※ 연관의 다중성 : 다양한 모델링 상황에서 얼마나 많은 객체가 연관에 연결되어 있는가를 표현하는 것이 중요하게 간주된다. 이와 같은 객체의 수를 역할의 다중성이라고 하며 특정한 값의 범위를 나타내는 표현식이나 정확한 값으로 표현한다. 1 (1), 0 혹은 1 (0..1), 다수 (0..*), 하나 이상 (1..*) 등과 같이 표현한다.

※ 연관의 집단화 : 하나의 클래스는 보다 큰 집단을 표현하고 또 다른 클래스는 작은 집단을 표현하는 whole/part관계를 모델링할 필요가 있을 때 사용한다. 집단화는 연관의 특수한 유형으로서 연관의 전체 부분에 흰색 다이아몬드를 붙여서 표현한다.

(4) 클래스 다이어그램

-시스템의 정적인 설계 뷰를 모델링하는데 사용된다. 일반적으로 클래스, 인터페이스, 협동, 종속, 일반화, 연관 등의 관계를 표현한다. 클래스 다이어그램은 컴포넌트 다이어그램과 배치 다이어그램의 기초가 된다.

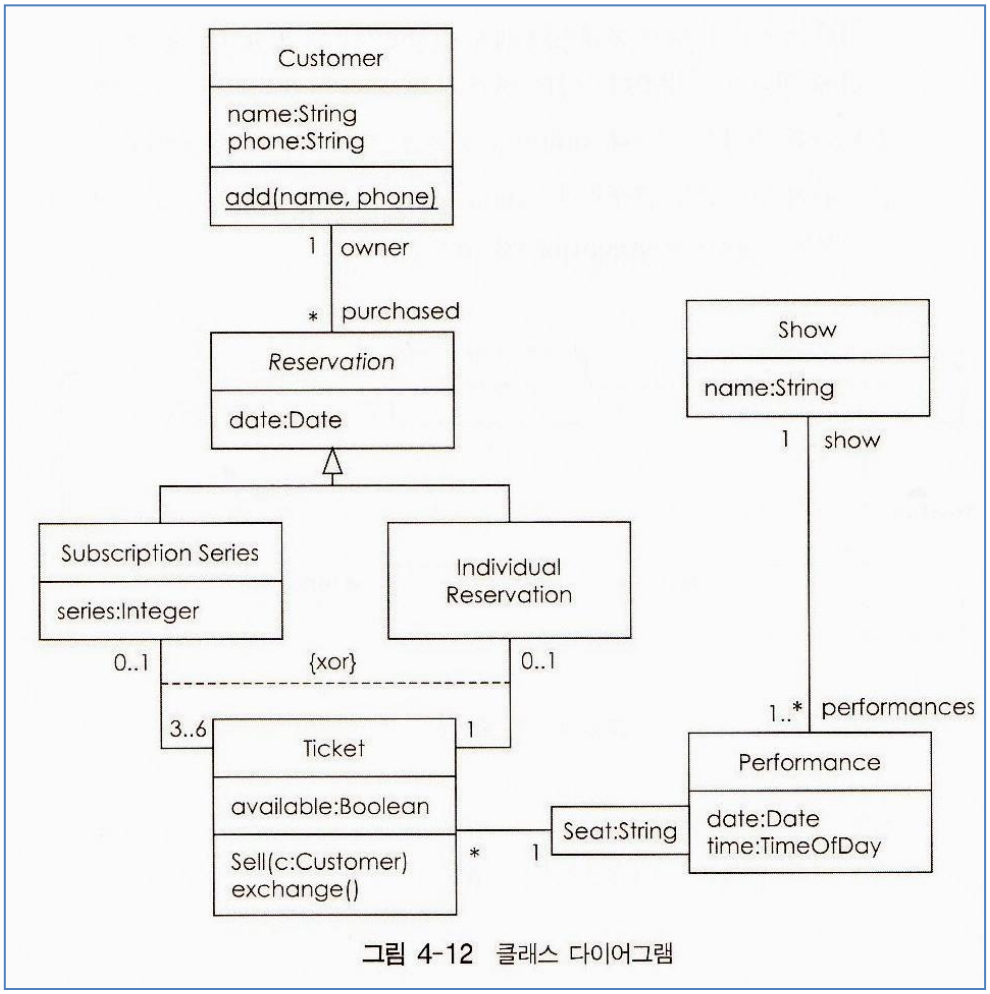


그림 4-12 클래스 다이어그램

6) 확장 메커니즘

-모델링 언어의 표현 능력을 확장시킬 수 있는 기능이다. UML의 추가적인 표현 방법이다.
-스테레오타입, 태그값, 제약조건, 노트 등이 있다.

(1) 스테레오타입

-UML의 어휘를 확장시켜 사용자가 자신의 문제에 적합한 새로운 유형의 표현 양식을 기존의 양식으로부터 유도할 수 있도록 한다. <<>>로 둘러싸인 이름으로 표현되며 다른 요소의 이름 위에 위치한다.

(2) 태그값

-UML표현 양식의 속성을 확장시켜 표현 양식의 규격에 새로운 정보를 포함시킬 수 있도록 한다. {}로 둘러싸인 스트링으로 표현되며, 다른 요소의 이름 아래 위치한다. 보편적으로 코드 생성이나 구성관리에 관련되는 속성을 지정하는데 쓰인다.

(3) 제약조건

-UML표현 양식의 세멘틱스를 확장시켜 기존의 표현 양식을 수정하거나 새로운 규칙을 추가할 수 있도록 한다. {}로 둘러싸인 스트링으로 표현되며, 관련되는 요소의 근처에 위치시키거나 종속 관계를 이용하여 해당 요소에 연결시킨다. 제약조건은 모델을 위하여 반드시 만족되어야 하는 조건을 지정한다.

(4) 노트

-기존의 모델에 주석이나 부가적인 정보를 추가시킬 수 있도록 한다. 산출물의 표현, 자유로운 형식의 메모, 설명 등을 표현할 수 있다. 의미적으로 모델에 아무런 영향을 미치지 않는다.

7) 고급 클래스

-UML은 모델링 요소의 구조와 특성을 기술하는 분류자를 지원한다. 클래스는 이러한 분류자의 한 유형이다.

-UML분류자에는 클래스 이외에도 인터페이스, 데이터타입, 신호, 컴포넌트, 노드, 유스 케이스, 서브시스템이 포함된다.

-일반적으로 인스턴스를 가질 수 있는 모델링 요소를 분류자라고 한다. 분류자는 구조적인 특성(애트리뷰트)과 행위적인 특성(오퍼레이션)을 갖는다.

(1) 가시성

-클래스의 애트리뷰트나 오퍼레이션을 다른 클래스가 사용할 수 있는지의 여부를 지정한다. 접근 권한에 대한 표기를 나타낸다. 특정 기호를 붙이지 않은 경우에는 public으로 간주한다.

※ public : 애트리뷰트와 오퍼레이션의 이름 앞에 +기호를 붙인다.
※ protected : 애트리뷰트와 오퍼레이션의 이름 앞에 #기호를 붙인다.
※ private : 애트리뷰트와 오퍼레이션의 이름 앞에 -기호를 붙인다.

(2) 범위

-애트리뷰트와 오퍼레이션이 클래스의 각 인스턴스에 나타는지, 혹은 단 하나의 인스턴스만이 존재해서 그것을 공유하는지를 지정하는 것이다. C++에서의 클래스 내 static변수/함수나 멤버 인스턴스 변수, 클래스 인스턴스 변수와 같은 개념으로 생각할 수 있다.
-instance, classifier의 두 가지 유형이 있다. Instance의 경우는 따로 표시하지 않고, classifier의 경우에는 애트리뷰트와 오퍼레이션에 밑줄을 그어 나타낸다.
※ instance : 클래스의 각 인스턴스는 각자의 고유한 애트리뷰트, 오퍼레이션을 가진다.
※ classifier : 클래스의 각 인스턴스는 하나의 애트리뷰트, 오퍼레이션을 가진다.

(3) 추상클래스, 루트, 잎, 다형적 요소

-UML에서는 추상 클래스를 표현할 시 클래스의 이름을 이탤릭체로 표기해서 나타낸다.
-자식을 갖지 않는 클래스를 잎 클래스라고 하며 클래스의 이름 밑에 leaf라는 속성을 표기한다. 또한 부모를 갖지 않는 클래스는 루트 클래스라고 하며 클래스의 이름 밑에 root라는 속성을 표기한다.

(4) 다중성

-클래스가 가지는 인스턴스의 수를 제한할 필요가 있을 경우를 위한 것이다. 클래스가 갖는 인스턴스의 수를 클래스의 다중성이라고 하며 UML에서는 클래스 아이콘의 오른쪽 상위 모서리에 숫자로 나타낸다.
-애트리뷰트에도 적용된다. 애트리뷰트의 다중성은 이름 뒤에 []속에 표기함으로써 지정한다.

(5) 애트리뷰트

-일반적으로는 이름만을 기록하지만 이름 외에도 가시성, 범위, 다중성, 타입, 초기값, 특성 등을 추가로 지정할 수 있다.
-특성에는 changeable, addOnly, frozen이 있으며 애트리뷰트의 값 변경과 관련이 있다.
-완전한 구문은 이와 같다.
[visibility] name [multiplicity] [:type] [=initial-value] [{property-string}]

사용 예)

※ origin : 이름
※ + origin : 가시성, 이름
※ origin: Point : 이름, 타입
※ head: *Item : 이름, 복합타입
※ name [0..1]: String : 이름, 다중성, 타입
※ origin: Point = (0, 0) : 이름, 타입, 초기값
※ origin {frozen} : 이름, 특성

(6) 오퍼레이션

-일반적으로는 이름만을 기록하지만 이름 외에도 가시성, 범위, 매개변수, 반환타입, 병행성 세멘틱스 등을 추가로 지정할 수 있다.

-방향은 in, out, inout이 있으며 특성에는 isQuery, sequential, guarded, concurrent 등이 있다.

-완전한 구문은 이와 같다.

[visibility] name [(parameter-list)] [:return type] [{property-string}]

사용 예)

※ display : 이름

※ + display : 가시성, 이름

※ set(n : Name, s : String) : 이름, 매개변수

※ getID(): Integer : 이름, 반환타입

※ restart() {guard} : 이름, 특성

(7) 템플릿 클래스

-템플릿은 아직 완전히 지정된 클래스가 아니며, 템플릿에 대한 매개변수를 통하여 최종적인 명세가 이루어지는 클래스를 말한다. 사용자는 템플릿을 직접 사용할 수 없고, 템플릿을 사용하기 위해서는 우선 템플릿을 인스턴스화 하여야 한다.

-템플릿 클래스는 템플릿 매개변수를 열거하는 점선 사각형을 아이콘의 우측 상단 모서리에 표기함으로써 모델링한다.

-바인딩을 제공하는 이름을 갖는 클래스를 선언함으로써 암시적으로 템플릿 클래스를 인스턴스화 할 수 있다. 또는 bind라는 스테레오 타입을 갖는 종속 관계를 사용함으로써 명시적으로 인스턴스화 할 수 있다.

8) 고급 관계

-종속, 일반화, 연관 외에 다중 상속, 네비게이션, 합성, 정제 등과 같은 추가적인 사항을 모델링할 수 있으며, 실현을 사용하여 인터페이스와 클래스 혹은 컴포넌트간의 연결과 유스 케이스와 협동간의 연결을 모델링 할 수 있다.

(1) 종속

-일반적인 종속 외에 UML에서 제공하는 17개의 스테레오 타입을 추가로 사용하여 다양한 의미를 부여할 수 있다.

-bind, derive, friend, instanceOf, instantiate, powertype, refine, use, access, import, extend, include, become, call, copy, send, trace등이 있다.

(2) 일반화

-일반적인 일반화 외에 UML에서 제공하는 하나의 스테레오타입과 네 개의 제약조건을 일반화에 적용할 수 있다.

-스테레오타입 implementation, 제약조건 complete, incomplete, disjoint, overlapping등이 있다.

(3) 연관

-일반적인 연관 외에 추가로 네비게이션, 한정 기능, 인터페이스 지정자, 합성 등과 같은 기타의 특성을 갖는다.

※ 네비게이션 : 대부분의 경우 연관에 의한 네비게이션은 양방향이다. 그러나 경우에 따라서는 네비게이션을 제한할 필요가 발생한다. 네비게이션의 방향을 명시적으로 지정하기 위해서는 네비게이션의 진행 방향으로 연관에 화살촉을 표기한다.

※ 한정기능 : 연관에 참여하는 한 객체에 대하여 이 객체와 연관으로 관련되는 다른 객체를 식별하기 위해서 UML은 한정기능을 제공한다. 한정자는 연관의 애트리뷰트로 하나의 객체와 연관으로 관련되는 일련의 객체를 분할하는 값을 갖는다. 한정자는 연관을 나타내는 실선의 끝에 작은 직사각형을 붙여서 표시하며, 직사각형 안에 한정자의 애트리뷰트를 표기한다.

※ 인터페이스 지정자 : 연관의 경우에는 자신이 제공하는 인터페이스의 일부분만을 제공할 필요가 발생한다. UML에서는 이를 위하여 연관에 참여하는 클래스의 역할에 대한 타입을 명시적으로 지정할 수 있다.

※ 합성 : 연관의 특수한 경우로서 집단화의 변형이며 전체에 대한 부분으로서의 강한 소유권과 동일한 수명주기를 의미한다. 합성 집단화에서 하나의 객체는 어느 한 순간에 다른 객체의 한 부분이 된다. 합성은 연관의 전체(whole)부분에 흑색 다이아몬드를 붙여서 표현한다.

(4) 실현

-하나의 분류자가 다른 분류자의 수행사항을 지정하는 분류자간의 의미관계를 나타낸다.

-수행사항을 지정하는 분류자로 향하는 방향성을 갖는 점선으로 표시하며 끝부분에 삼각형의 화살표를 붙인다.

-의미적으로 종속과 일반화의 중간 정도에 해당하며, 표기 방법은 종속과 일반화의 표기 방법을 결합한 것이다.

9) 인터페이스 및 타입(추상 클래스)

-인터페이스는 클래스나 컴포넌트의 서비스를 지정하기 위한 오퍼레이션의 집합체이다. 인터페이스는 추상 오퍼레이션으로만 기술되며 인터페이스를 지정함으로써 사용자는 모델링하려는 추상화의 행위를 추상화의 구현과는 독립적으로 기술할 수 있다. 인터페이스는 클래스나 컴포넌트의 구현과 명세를 분리하는데 중요할 뿐만 아니라 시스템의 확장에도 중요하다.

-타입은 객체의 도메인과 객체에 적용할 수 있는 오퍼레이션을 지정하기 위한 클래스의 스테레오타입이다.

-인터페이스는 대개 원으로 표현되지만 인터페이스의 오퍼레이션과 기타의 속성을 표현하기 위하여 스테레오타입 클래스로 표현하기도 한다.

(1) 인터페이스의 이름

-일반 클래스의 이름 구별과 같다. 단순 이름은 텍스트 스트링이고 경로 이름은 인터페이스 앞에 존재하는 패키지의 이름을 붙인다.

(2) 인터페이스의 오퍼레이션

-인터페이스는 어떠한 구조나 구현을 지정하지 않는다. 따라서 인터페이스는 애트리뷰트를 포함하지 않으며, 오퍼레이션을 가질 수는 있지만 오퍼레이션의 구현을 제공하지는 않는다.

(3) 인터페이스의 관계

-클래스와 마찬가지로 일반화, 연관, 종속, 실현 등에 참여할 수 있다. 인터페이스는 클래스나 컴포넌트의 구현에 대한 언급 없이도 이들이 수행하여야 하는 사항을 지정한다. 따라서 인터페이스는 인터페이스에 정의된 오퍼레이션을 적절히 구현하는 일련의 메소드를 제공한다.

-인터페이스는 항상 1:1의 다중성을 갖는 연관으로 모델링 요소에 연결된다. 인터페이스를 사용하는 클래스는 종속 관계를 통하여 인터페이스에 연결된다.

(4) 타입과 역할

-클래스의 인스턴스는 클래스의 모든 인터페이스를 지원하여야 한다. 그럼에도 불구하고 경우에 따라서는 인스턴스가 자신과 관련된 일부의 인터페이스만을 제공할 수도 있다. 이러한 경우 각각의 인터페이스는 객체가 수행하는 역할을 나타낸다.

-UML에서 역할은 연관의 이름으로 특정한 인터페이스 이름을 사용함으로써 표현할 수 있다.

-타입의 개념은 인터페이스의 개념과 매우 밀접하다. 그러나 타입의 정의는 인터페이스와는 달리 애트리뷰트를 가질 수 있다.

-대부분의 경우 타입과 인터페이스는 상호 교환 가능한 것으로 간주된다.

(5) 인터페이스의 모델링 프로세스

-인터페이스를 사용하는 가장 일반적인 목적은 소프트웨어 컴포넌트로 구성되는 시스템에서의 결합을 모델링하기 위한 것이다.

-인터페이스를 모델링하기 위해서는 다음을 수행하여야 한다.

① 시스템에 존재하는 클래스와 컴포넌트 중에서 상호 밀접하게 결합되어 있는 클래스와 컴포넌트를 그룹으로 묶는다.

② 변화를 고려하여 이러한 그룹핑을 개선한다. 함께 변화하는 경향이 있는 클래스 혹은 컴포넌트는 협동으로 그룹화하여야 한다.

③ 일련의 클래스나 컴포넌트의 인스턴스로부터 다른 클래스나 컴포넌트의 인스턴스로 향하는 오퍼레이션과 신호를 고려한다.

④ 논리적으로 관련성이 있는 오퍼레이션과 신호를 인터페이스로 묶는다

⑤ 시스템에 존재하는 각각의 협동에 대하여 협동이 의존하는 인터페이스와 협동이 제공하는 인터페이스를 식별한다. 협동이 의존하는 인터페이스는 종속 관계로 모델링하고, 협동이 제공하는 인터페이스는 실현 관계로 모델링한다.
⑥ 시스템에 존재하는 각각의 인터페이스에 대하여 각각의 오퍼레이션에 대한 사전조건과 사후조건을 사용하고, 인터페이스에 대한 유스 케이스와 상태 머신을 사용하여 인터페이스의 동적인 특성을 문서화한다.

-워크플로우를 따르면서 역할이 변경되는 비즈니스 객체 등을 모델링할 때에는 객체 타입의 동적인 특성을 명시적으로 모델링하는 것이 유용하다. 동적인 타입을 모델링하기 위해서는 다음을 수행하여야 한다.

① 객체의 가능한 타입을 지정한다. 각각의 타입을 type 혹은 interface라는 스테레오타입 클래스로 표현한다. 클래스가 구조와 행위를 요구하는 경우에는 type을 사용하고, 행위만을 요구하는 경우에는 interface를 사용한다
② 객체가 취할 수 있는 모든 역할을 모델링한다. 클래스 다이어그램에서 객체가 연관과 관련하여 수행하는 각각의 역할을 명시적으로 지정하고 클래스 다이어그램에서 일반화 관계를 사용하여 클래스-타입 관계를 지정한다.
③ 상호작용 다이어그램에서 동적으로 타입이 부여된 클래스의 각 인스턴스를 표현한다. 인스턴스의 역할을 객체의 이름 밑에 대괄호로 표시한다.
④ 객체의 역할 변경을 표현하기 위하여 상호작용에서 객체가 수행하는 각각의 역할에 대해 객체를 표현하고, 이들 객체를 become 스테레오타입 메시지로 연결한다.

10) 패키지

-대형 시스템의 설계에는 많은 클래스, 인터페이스, 컴포넌트, 노드, 다이어그램 등이 포함되는데 패키지란 이와 같은 모델링 요소들을 그룹으로 구성하는 범용의 메커니즘이다.
-모델링 요소들을 그룹으로 처리할 수 있는 단위로 배열하기 위해서 패키지를 사용한다. 또한 사용자는 이들 요소에 대한 가시성을 통제할 수 있다.
-잘 구조화된 패키지는 느슨하게 결합되어 있으며 응집도가 매우 높다.
-UML에서는 탭이 부착된 폴더의 모습으로 표현된다.
-클래스와 비교했을 때 클래스는 실체를 가지고 있으므로 인스턴스를 가지지만 패키지는 실체를 가지지 않기 때문에 인스턴스를 가지지 않으며 실행 시스템에서는 보이지 않는다.

(1) 패키지의 이름

-클래스와 마찬가지로 표현된다. 다만 패키지의 소유 요소를 표현할 때에는 탭 부분에 패키지의 이름을 기록한다.

(2) 소유 요소

-패키지는 클래스, 인터페이스 등과 같은 다른 요소를 포함할 수 있다. 소유란 합성 관계로서 요소를 패키지 안에 선언하는 것이다. 모든 요소는 단 하나의 패키지에 의해서만 소유된다.

-패키지는 네임스페이스를 형성한다. 따라서 동일한 패키지에서 같은 이름을 갖는 두 개의 클래스는 존재할 수 없다.

-패키지는 다른 패키지를 소유할 수 있다. 하지만 이러한 패키지의 중첩 관계가 심한 것은 바람직하지 못하다.

(3) 가시성

-클래스의 애트리뷰트, 오퍼레이션에 대한 것과 마찬가지로 +, #, - 를 통해서 가시성을 나타낸다.

(4) 임포트와 익스포트

-UML에서 임포트 관계는 스테레오타입 import를 갖는 종속 관계로 모델링한다.

-상속 모델에서 패키지의 public 부분(가시성에서 +)을 패키지의 익스포트라고 한다.

(5) 패키지의 모델링 프로세스

-시스템에 존재하는 다수의 모델링 요소를 그룹화하기 위해서 수행하여야 하는 과정.

① 특정한 아키텍처 뷰에 존재하는 모델링 요소를 분석하고, 개념적이나 의미적으로 상호 연관되는 요소의 그룹을 조사한다.

② 각각의 그룹을 패키지로 묶는다.

③ 각각의 패키지에 대하여 패키지의 외부에서 접근할 수 있어야 하는 요소들을 식별하고 이를 public으로 표시, 그 외는 protectec나 private로 표시

④ import 종속 관계를 이용하여 다른 패키지를 구축하는 패키지들을 명시적으로 연결한다.

⑤ 부모-자식 관계를 갖는 패키지인 경우에는 일반화를 통하여 자식 패키지를 부모 패키지에 연결한다.

11) 인스턴스

-추상화에 대한 구체적인 표현으로써 일련의 오퍼레이션이 적용되며 오퍼레이션의 결과가 저장되는 상태를 갖는다.

-대부분의 UML요소는 추상화와 인스턴스의 형태로서 모델링된다. 인스턴스와 객체는 대부분의 경우에 동의어로 사용된다.

-UML에서는 모델링 요소의 이름에 밑줄을 그음으로써 인스턴스를 표현한다.

-대부분의 인스턴스는 클래스의 인스턴스로 이를 객체라고 한다.

(1) 인스턴스의 이름

-일반적인 클래스의 이름과 동일하지만 다수의 객체를 모델링할 때에는 익명객체 집합을 표현하는 다중 객체를 사용할 수 있다.

(2) 오퍼레이션

-객체에 대하여 수행할 수 있는 오퍼레이션은 객체의 추상화에서 선언한다. 상속관계에 따라 다형적으로 호출될 수도 있고, 호출되지 않을 수도 있다.

(3) 상태

-특정 시점에서 객체의 상태 값을 지정함을 말한다.

-UML을 사용하여 객체의 애트리뷰트 값을 표현할 수도 있고, 특정 시점의 상태머신의 상태를 표현할 수도 있다.

(4) 인스턴스의 모델링 프로세스

-구체적인 인스턴스를 모델링하기 위한 방법.

-구체적인 객체는 객체 다이어그램, 컴포넌트 다이어그램, 배치 다이어그램 등 정적인 다이어그램에서 나타난다.

-객체간의 동적인 상호작용을 모델링하기 위해서는 프로토타입 객체를 만드는데 이를 위해서는 상호작용 다이어그램이나 활동 다이어그램을 사용한다.

1 모델링하려는 문제를 표현하는데 필요한 인스턴스를 식별한다.
2 이들 객체를 UML의 인스턴스로 표현한다.
3 각 인스턴스의 스테레오타입, 태그값, 애트리뷰트를 표현한다.
4 객체 다이어그램이나 인스턴스 유형의 적절한 다이어그램에 이들 인스턴스간의 관계를 표현한다.

(5) 객체 다이어그램

-클래스 다이어그램에 포함되어 있는 요소의 인스턴스를 모델링한다. 객체 다이어그램은 일반적으로 객체와 링크를 포함한다.

-어느 한 시점에서의 일련의 객체와 객체간의 관계를 보여주며 복잡한 데이터 구조를 모델링하는데 매우 유용하다.

12) 유스 케이스

-시스템을 사용하는 사용자(액터)에게 가치있는 결과를 제공하기 위하여 시스템이 수행하여야 하는 일련의 활동으로 정의된다.

-구현에 대한 사항은 고려하지 않고 개발 중인 시스템의 행위를 포착하기 위하여 유스케이스를 적용한다.

-시스템이 구현됨에 따라 유스 케이스는 협동으로 실현된다.
-유스 케이스의 기본적인 목적은 시스템의 기능적인 요구사항의 정의, 시스템의 수행에 대한 일관성 있는 정의, 시스템 검증을 위한 테스트의 기반 제공, 기능적인 요구사항의 추적 능력 제공 등이다.
-시스템 외부의 대상(액터)과 시스템 자체(시스템의 추상화)간의 상호작용을 표현한다.

(1) 유스 케이스의 이름
-일반적인 클래스의 경우와 동일하다.

(2) 시스템과 액터
-유스 케이스 다이어그램에서 시스템은 사각형으로 표현된다. 유스 케이스 모델링시 시스템의 범위를 정하는 것은 용이하지 않다.
-액터는 일반적으로 인간, 장비, 혹은 다른 시스템 등이며 시스템과 상호작용한다.
-액터는 시스템의 외부에 존재하며 연관 관계에 의해서만 유스 케이스와 연결될 수 있다.

(3) 이벤트 흐름 및 시나리오
-이벤트 흐름을 그래픽으로 표현하기 위하여 상호작용 다이어그램을 사용할 수 있다.
-대부분의 경우 주요 흐름을 설명하기 위해 하나의 순차 다이어그램을 사용하고, 예외적인 흐름을 설명하기 위해 순차 다이어그램의 변형을 사용하게 된다.
-유스 케이스는 하나의 순서를 설명하는 것이 아니고 일련의 순서를 설명하며 각각의 순서를 시나리오라고 한다.
-시나리오는 유스케이스의 행위를 설명하는 순서의 활동이다. 시나리오는 기본적으로 유스 케이스의 인스턴스가 된다.

(4) 유스 케이스와 협동
-함께 작업하는 일련의 클래스와 요소들 간의 정적, 동적인 구조를 UML에서는 협동으로 모델링한다.
-대부분의 경우에 하나의 유스 케이스는 정확히 하나의 협동으로 실현되기 때문에 관계성을 명확히 모델링할 필요는 없다.

(5) 유스 케이스의 관계
-유스 케이스의 관계에는 확장, 포함 및 그룹핑의 세 가지 유형이 존재한다. UML에서는 클래스의 일반화와 마찬가지로 삼각형의 화살표가 부착된 실선으로 표시한다.

◆ 포함 관계
-포함 관계에서 포함되는 유스 케이스는 결코 독립적으로 존재할 수 없으며, 자신을 포함하는 기본 유스 케이스의 일부분으로만 존재한다.
-기본 유스 케이스가 다른 유스 케이스의 행위를 끌어오는 것이라고 생각하면 된다.

-동일한 이벤트 흐름을 반복적으로 표현하는 것을 피하기 위하여 사용한다.
-포함 관계는 include 스테레오타입을 사용하여 종속 관계로 표현한다.

◆ 확장 관계

-기본 유스 케이스는 확장 지점이라는 특정한 위치에서만 확장될 수 있다.
-선택적인 행위를 필수 행위로부터 분리시켜 일부분을 모델링하기 위하여 사용된다.
-확장 관계는 exclude 스테레오타입을 사용하여 종속 관계로 표현한다.

◆ 그룹핑

-그룹핑이란 클래스의 패키지 구성과 마찬가지로 유스 케이스도 패키지를 이용하여 그룹핑할 수 있음을 말한다.

(6) 유스 케이스의 모델링 프로세스

1 시스템 요소와 상호작용하는 액터의 식별
2 액터의 일반적인 역할이나 특수화된 역할을 식별함으로써 액터를 구조화한다.
3 액터가 시스템 요소와 상호작용하는 기본적인 방식을 고려한다.
4 이러한 행위를 유스 케이스로 구성한다.

(7) 유스 케이스 다이어그램

-시스템의 행위를 표현하는 것으로 시스템이 외부에 제공하는 가시적인 서비스를 지원한다.
-시스템의 정적인 유스 케이스 뷰를 모델링할 때 일반적으로 시스템의 범위를 모델링하기 위해 또는 시스템의 요구사항을 모델링하기 위해 사용한다.
-UML에서는 유스 케이스 다이어그램을 사용하여 시스템을 둘러싸고 있는 액터를 강조하면서 시스템의 범위를 모델링할 수 있다.

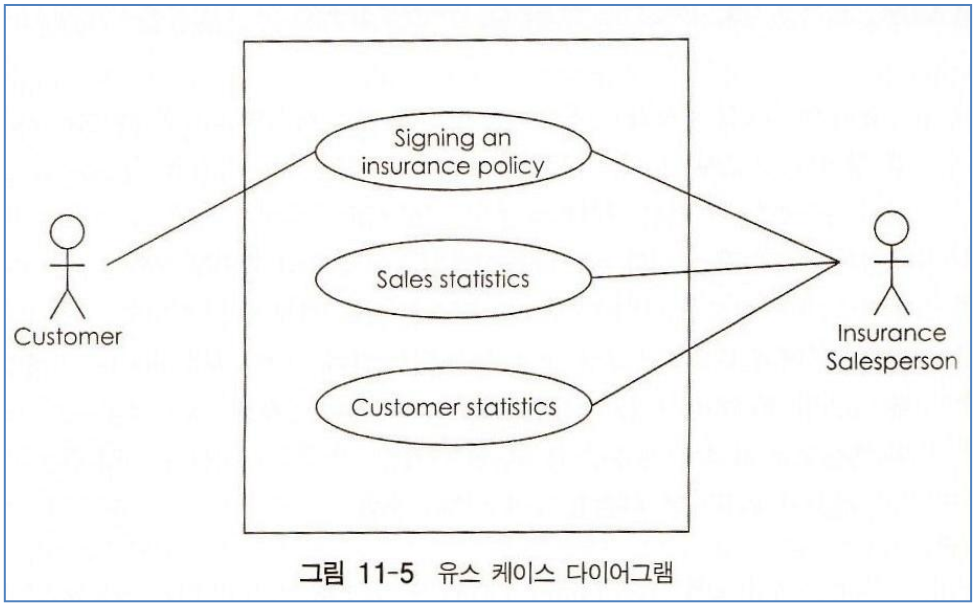


그림 11-5 유스 케이스 다이어그램

13) 상호작용

-UML에서 시스템의 동적인 면을 모델링하는데에는 상호 작용을 사용한다.
-상호작용은 특정한 목적을 달성하기 위하여 일련의 객체간에 교환되는 메시지로 구성되는 행위이다.
-대부분의 경우 메시지는 오퍼레이션의 호출이나 신호의 전달, 객체의 생성, 소멸 등을 포함한다.
-UML내의 표기는 메시지의 가장 중요한 부분인 이름, 매개변수, 순서를 강조할 수 있도록 한다. 실선으로 표현되며 거의 대부분의 경우에 오퍼레이션의 이름을 포함한다.

(1) 메시지

-메시지는 메시지의 송, 수신자간에 화살표를 갖는 선으로 표현된다. 각각의 화살표 유형은 서로 다른 메시지의 타입을 나타낸다.
※ 단순(simple) : 단순 제어 흐름을 나타낸다. 세부 사항을 설명하지 않고 객체간에 제어가 어떻게 전달되는지 보여준다.
※ 동기(synchronous) : 오퍼레이션 호출로 구현되는 제어의 절차적인 흐름이나 중첩 흐름을 나타낸다.
※ 비동기(asynchronous) : 일반적으로 객체가 병행 실행되는 실시간 시스템에서 사용된다.

(2) 메시지 순서

-메시지의 순서를 표현하기 위하여 메시지의 앞에 순서 번호를 붙일 수 있다. 대부분의 경우 사용자는 절차적 제어 흐름이나 중첩 제어 흐름을 사용한다.
-예를 들어 표현식 D5 : openDoor(3)같은 경우 오퍼레이션 openDoor가 D라는 이름을 갖는 프로세스 혹은 스레드가 루트인 순서에서 다섯번째 메시지로 전달됨을 나타낸다.

(3) 상호작용의 표현

-상호작용의 모델링에서 객체와 메시지는 순차 다이어그램과 협동 다이어그램의 두 가지 방식으로 시각화할 수 있다.
-메시지의 시간 순서를 강조하는 것을 UML에서 순차 다이어그램이라고 한다.
-메시지를 주고 받는 객체의 구조적인 구성을 강조하는 것을 UML에서 협동 다이어그램이라고 한다.

14) 상호작용 다이어그램

-순차 다이어그램과 협동 다이어그램은 시스템의 동적인 부분을 모델링하는 다이어그램으로 상호작용 다이어그램이라고도 한다.
-상호작용 다이어그램은 특정한 객체 집합의 동적인 특성을 표현하기 위하여 독자적으로 활용될 수도 있고, 유스케이스의 특정한 제어 흐름을 모델링하기 위하여 사용될 수도 있다.

(1) 순차 다이어그램

-메시지의 시간 순서를 강조한다. 다시 말해 다수의 객체간에 메시지가 전달되는 순서를 표현한다.

-X축을 따라 다이어그램의 최상부에 상호작용에 참여하는 객체를 위치시킨다. 일반적으로 상호작용을 일으키는 객체를 왼쪽에 위치시키고, 종속적인 객체들을 오른쪽에 위치시킨다.

-y축을 따라 객체간에 주고받는 메시지를 위에서 아래로 증가하는 시간에 의하여 위치시킨다.

※ 순차 다이어그램의 특징 1 : 순차 다이어그램에는 객체의 수명선이 존재해서 스테레오타입 create가 부착된 메시지를 받으면 시작되고 스테레오타입 destroy가 부착된 메시지를 받으면 종료되며 X자를 이용해 표시한다.

※ 순차 다이어그램의 특징 2 : 순차 다이어그램에는 제어 초점이 존재하는데 이는 객체가 활동을 수행하는 기간을 보여주는 얇은 직사각형으로 표현된다.

(2) 협동 다이어그램

-상호작용에 참여하는 객체의 구성을 강조한다. 상호작용에 참여하는 객체를 그래프의 노드로 표현한다. 이들 객체를 연결하는 링크는 그래프의 선으로 표현한다. 그리고 객체간에 주고받는 메시지를 링크에 표현한다.

-일반적으로 순차적인 제어 흐름을 모델링하지만 반복, 분기, 조건등을 모델링할 때에는 [i:=1..n]등의 반복표현이나 [x>0]같은 조건식을 메시지의 순서 번호에 붙인다. UML자체적인 특정 표현 양식은 없다.

※ 협동 다이어그램의 특징 1 : 협동 다이어그램에는 경로가 존재한다. 객체의 연결을 표시하기 위해 경로 스테레오 타입으로 local, parameter, global, self등으로 표현된다.

※ 협동 다이어그램의 특징 2 : 협동 다이어그램에는 순서 번호가 존재한다. 메시지의 시간 순서를 표시하기 위해 메시지 앞에 번호를 붙인다.

15) 협동 및 패턴

-협동은 유스 케이스와 오퍼레이션의 실현을 지정하고, 시스템의 중요한 디자인 패턴을 모델링하는데 사용된다.

-협동은 구조적인 요소와 행위적인 요소를 포함하는 시스템의 개념적인 블록에 이름을 붙인다.

-UML에서는 점선으로 이루어진 타원으로 협동에 대한 그래픽 표현을 지원한다.

(1) 협동의 이름

-일반적인 클래스의 이름과 동일하다.

(2) 협동의 구성

-협동은 구조적인 부분과 행위적인 부분을 갖는다.

-협동의 구조적인 부분의 표현은 클래스 다이어그램을 사용하고 행위적인 부분은 상호작용 다이어그램을 사용해서 표현한다.

-구조적인 부분은 협동을 수행하기 위한 클래스, 인터페이스, 컴포넌트, 노드 등 기타의 요소들을 지정한다

-행위적인 부분은 이들 요소들이 어떻게 상호작용 하는지에 대한 동적인 사항을 지정한다.

-협동은 패키지나 서브시스템과 달리 자신의 구조적인 요소들을 소유하지 않는다. 대신에 다른 곳에서 선언된 구조적인 요소들을 단순히 참조하거나 사용한다.

-협동에 대해서는 협동과 협동이 실현하는 대상간의 관계와 협동관의 관계의 두 가지 유형의 관계를 고려해야 한다.

※ 협동과 협동이 실현하는 대상의 관계 : 유스 케이스나 오퍼레이션과 협동간의 관계는 실현 관계로서 모델링된다.

※ 협동간의 관계 : 협동은 다른 협동을 정제할 수 있는데 이러한 관계는 유스 케이스간의 정제 관계와 같이 표현할 수 있다.

(3) 패턴

-시스템의 아키텍처를 구체화하는 설계 패턴과 아키텍처 패턴을 지정하기 위하여 패턴을 사용한다.

-패턴은 공통적인 문제에 대한 공통적인 해결책을 제공한다.

-설계 패턴은 일련의 객체에 적용되는 패턴이고 아키텍처 패턴은 응용을 위한 템플릿을 제공하는 패턴이다. 설계 패턴은 일련의 객체에 대한 구조 및 행위를 지정하고, 아키텍처 패턴은 전체 시스템의 구조와 행위를 지정한다.

-설계 패턴은 협동으로 펴현되며, 아키텍처 패턴은 스테레오타입 패키지로 표현한다.

(4) 설계 패턴 및 아키텍처 패턴

-설계 패턴은 공통의 행위를 수행하기 위하여 함께 작업하는 일련의 추상화를 지칭한다. 설계 패턴은 평범한 협동으로서 모델링한다.

-협동의 내부를 들여다보면 클래스 다이어그램으로 표현되는 구조적인 부분과 상호작용 다이어그램으로 표현되는 행위적인 부분을 볼 수 있다.

-또한 설계 패턴은 공통의 행위를 수행하기 위하여 함께 작업하는 일련의 추상화를 위한 템플릿을 지칭한다. 이러한 설계 패턴은 매겨변수 협동 혹은 템플릿 협동으로서 모델링한다.

-아키텍처 패턴은 스테레오타입 패키지로 모델링하며 패키지 내부에는 시스템 아키텍처의 다양한 뷰에 존재하는 설계 패턴이 있다.

16) 활동 다이어그램

-활동 다이어그램은 시스템의 동적인 부분을 모델링하는 UML의 다이어그램으로서 활동 간의 제어 흐름을 보여주는 일종의 플로우 차트이다.
-활동 다이어그램은 상호작용을 표현하는 또 다른 방법으로서 행동이 어떻게 이루어지며, 무엇을 수행하고, 언제 발생하며, 어디에서 발생하는지를 설명한다.
-활동 다이어그램을 이용하면 제어 흐름의 시점에서 객체의 흐름, 오퍼레이션의 제어 흐름을 모델링할 수 있다.

(1) 행동 상태, 활동 상태 및 변환

-활동 다이어그램은 활동 상태, 행동 상태, 변환, 객체를 포함한다.
-객체에 대한 오퍼레이션 호출, 객체에 대한 신호 송신, 객체의 생성 및 소멸 등과 같은 실행 가능한 처리들을 행동이라고 한다.
-행동 상태는 양쪽이 둥근 직사각형을 이용하여 표현하며, 그 내부에는 표현식을 기록할 수 있다.
-활동 상태는 표현상으로는 행동 상태와 구분이 되지 않는다. 하지만 활동 상태는 입구 행동과 출구 행동 같은 추가적인 부분을 포함한다.
-변환은 하나의 행동 혹은 활동 상태로부터 다른 행동 혹은 활동 상태로의 경로를 보인다.
-UML에서는 방향성을 갖는 단순한 선으로써 변환을 표시한다.
-제어 흐름에는 시작과 종료 상태가 있어야 하는데 시작 상태는 검은 원으로 표현하며, 종료 상태는 검은 원을 내부에 갖는 이중 원으로 표현한다.

(2) 분기, 포크 및 조인

-부울 조건에 따라 변환 경로를 선택할 수 있는 분기는 다이아몬드 형태로 표시된다. 분기는 하나의 진입 변환 경로와 두 개 이상의 진출 변환 경로를 갖는다.
-포크는 하나의 제어 흐름이 두 개 이상의 병행적인 제어 흐름으로 분할되는 것을 의미한다. 포크는 하나의 진입 변환 경로와 두 개 이상의 진출 변환 경로를 갖는데 진출 경로의 각각은 독립적인 제어 흐름을 나타낸다.
-조인은 두 개 이상의 병행적인 제어 흐름의 동기화를 의미한다. 조인은 두 개 이상의 진입 변환 경로와 하나의 진출 변환 경로를 갖는다.
-UML에서 병행적인 제어 흐름의 포크와 조인은 굵은 수직손 혹은 수평선의 동기화 바를 사용햐여 표시한다.

(3) 스윔레인 및 객체 흐름

-활동 다이어그램상의 활동 상태들을 그룹으로 분할할 필요가 있을 시 사용한다.
-UML에서 각각의 그룹은 인접하는 그룹과 수직선으로 나누어지기 때문에 스윔레인이라고 한다.

-객체는 활동 다이어그램과 관련된 제어 흐름에 포함될 수 있는데 제어 흐름에 포함되는 객체는 객체를 변경하는 활동이나 변환에 종속 관계를 사용하여 연결된다. 이와 같은 종속 관계와 객체의 사용을 객체 흐름이라고 한다.
-객체의 상태는 객체 이름 밑에 대괄호를 사용하여 표시하며, 객체의 애트리뷰트 값은 객체 이름 밑에 별도의 구획을 이용하여 표현한다.

17) 이벤트 및 신호

-UML에서 이벤트는 시간과 공간에서의 위치를 갖는 사건을 표현한다.
-이벤트에는 신호, 호출, 시간 경과 혹은 상태 변화 등이 포함된다.
-이벤트의 모델링은 프로세스나 스레드의 모델링에 포함된다.

(1) 이벤트 유형

-이벤트는 외부 이벤트 또는 내부 이벤트로 나뉜다. 외부 이벤트는 시스템과 액터간에 전달되는 이벤트를 말하고 내부 이벤트는 시스템 내부에 존재하는 객체 간에 전달되는 이벤트를 말한다.
-UML에서는 신호, 호출, 시간 경과 및 상태 변화 등과 같은 네 가지 유형의 이벤트를 모델링할 수 있다.
※ 신호 : 비동기적으로 객체간에 전달되는 명명된 객체를 말한다. 클래스와 유사하며 애트리뷰트와 오퍼레이션을 가질 수 있다. 신호는 상태 머신에서 상태 변화의 활동으로 전송되거나 상호작용에서 메시지의 송신으로 전송될 수 있다. UML에서 오퍼레이션과 오퍼레이션이 보낼 수 있는 이벤트와의 관계는 스테레오타입 send를 이용하여 종속 관계로 표현한다. UML에서 신호는 스테레오타입 클래스로 모델링된다.

※ 호출 : 오퍼레이션의 전송을 표현한다. 객체가 상태 머신을 갖는 다른 객체에 대한 오퍼레이션을 호출하면 제어가 송신자에서 수신자에게 전달되고 이벤트가 일어나며 오퍼레이션이 종료되고, 수신자는 상태가 변환되며 송신자에게 제어가 다시 돌아간다. 호출 이벤트의 모델링은 신호 이벤트의 모델링과 구분이 되지 않는다. 신호 이벤트와 호출 이벤트는 최소한 두 개의 객체를 포함한다.

※ 시간 경과 : UML에서 키워드 after와 시간 간격을 나타내는 표현식을 사용하여 모델링된다.
※ 상태 변화 : 변화 이벤트는 UML에서 키워드 when과 부울 표현식을 이용하여 모델링된다. 표현식은 절대 시간이나 표현에 대한 지속적인 검사 등이 될 수 있다.

(2) 시간 및 공간

-실시간 시스템과 분산 시스템의 핵심적인 요소이다.
-UML의 타이밍 마크, 타임 표현식, 제약조건, 태그값 등을 사용할 수 있다.

-타이밍 마크는 이벤트가 발생하는 시간에 대한 표시이다. 타임 표현식은 시간의 절대값 혹은 상대값에 대한 문장이다. 제약조건은 {}로 둘러싸인 스트링으로 종속 관계에 의해 해당 요소에 연결된다. 태그값은 노드에서의 컴포넌트에 대한 배치를 표현하기 위하여 사용된다. {}로 둘러싸인 스트링으로 요소의 이름 밑에 놓는다.

18) 프로세스 및 스레드

-프로세스 뷰에 포함되며 시스템의 병행성과 동기화 메커니즘을 구성한다. UML에서는 각각의 독립적인 제어 흐름을 능동 객체로 모델링한다.

(1) 능동 객체

-클래스-객체의 관계처럼 능동 객체는 능동 클래스-객체의 관계다. 능동 클래스는 병행성을 모델링하는데 사용된다.
-UML에서 능동 클래스와 능동 객체에 대한 표현은 일반 클래스, 객체의 표현에서 두꺼운 선을 갖는 직사각형으로 표현된다.
-능동 객체는 시스템의 프로세스 뷰를 모델링하는데 중요한 역할을 하며 기본적으로 시스템의 성능, 확장성 및 처리량을 언급한다.

(2) 동기화

-하나의 객체에 하나 이상의 제어 흐름이 동시에 존재한다면 문제가 발생한다.
-객체지향에서는 이를 해결하기 위해 객체를 임계 영역으로 처리하는데 클래스의 오퍼레이션에 동기화 속성을 첨가시킨다. UML에서는 이를 위한 순차, 가드, 병행의 세 가지 접근 방법을 모두 모델링 할 수 있다.

19) 상태 머신

-개별적인 객체의 행위를 모델링하기 위해서 사용된다.
-상태머신은 객체가 여러 이벤트에 응답하면서 자신의 수명 동안에 지니게 되는 상태의 순서를 지정하는 행위이다.
-시스템의 동적인 부분을 모델링하기 위하여 사용되며 이벤트가 발생하면 객체의 현재 상태에 따라 활동이 일어난다.
-비동기적인 자극에 반응하여야 하는 객체나 현재의 행위가 과거의 행위에 의존하는 객체에 이용하면 좋다.
-시스템 외부의 액터로부터의 신호에 반응해야 하는 시스템의 행위를 모델링하는 데에도 사용할 수 있다.

(1) 객체의 상태

-상태는 객체가 이전에 수행한 활동의 결과이며 객체의 애트리뷰트 값과 다른 객체에 대한 링크에 의하여 결정된다.

-객체는 상태 머신에 의하여 시작 상태, 종료 상태의 두 가지 특별한 상태를 정의할 수 있다. 시작 상태는 상태 머신의 잠정적인 시작 위치를 나타내며 검은 원으로 표현한다. 종료 상태는 상태 머신의 실행이나 상태가 종료되었음을 나타내며, 검은 원을 내부에 갖는 이중 원으로 표현한다.

(2) 상태 변환

-객체가 행동을 수행하고 이벤트를 발생시켜 다음 상태로 들어간 것을 나타낸다. 소스 상태, 이벤트 트리거, 가드 조건, 행동, 타깃 상태의 다섯 부분을 가진다.

-일반적인 구문은 다음과 같다. Event-signature [guard condition] / action-exoression

- 소스 상태에서 타깃 상태로 향하는 방향을 갖는 실선으로 표현된다.

(3) 부분 상태

-UML의 상태 머신은 복잡한 행위의 모델링을 단순화 시켜주는 부분상태 개념을 제공한다. 부분 상태는 다른 상태의 내부에 중첩되는 상태이다.

-단순 상태는 부분 상태를 갖지 안는다. 그러나 복합 상태는 부분 상태, 즉 중첩 상태를 갖는다. 복합 상태는 병행 부분상태 혹은 순차 부분상태를 포함할 수 있다. UML에서 복합 상태는 단순 상태와 똑같이 표현되지만 중첩 상태 머신을 표현하는 그래픽 구획을 갖는다.

-경우에 따라서는 복합 상태를 떠나기 이전의 마지막 부분 상태를 기억하여야 하는 객체를 모델링할 필요가 발생하는데 UML에서는 이를 이력 상태를 이용하여 처리한다. 이력 상태는 기호 H를 포함하는 작은 원으로 표현한다.

-병행 부분상태는 동시에 실행되는 두 개 이상의 상태머신을 지정할 때 사용되며, 각각의 부분상태는 점선으로 구분된다.

(4) 상태 다이어그램

-시스템의 동적인 부분을 모델링하는 UML의 다이어그램으로 객체가 어떻게 이벤트에 반응하며, 내부상태를 어떻게 변화시키는지를 보여준다. 상태 다이어그램은 상태 머신을 표현한다.

-활동 다이어그램은 활동간의 제어 흐름을 보이고, 상태 다이어그램은 상태간의 제어 흐름을 보인다.

-대개의 경우 반응 객체의 행위에 대한 모델링이 포함된다. 상태 다이어그램은 개별적인 객체의 동적인 면을 표현하기 위하여 클래스, 유스 케이스, 혹은 전체 시스템이 부착될 수 있다.

-UML에서 객체의 이벤트 순서적인 행위는 상태 다이어그램을 사용하여 모델링할 수 있다. 상태 다이어그램은 상태간의 제어 흐름을 강조하는 상태 머신의 단순한 표현이다.

-상호작용은 함께 작업하는 일련의 객체의 행위를 모델링하지만 상태 다이어그램은 단일 객체의 수명 동안의 객체의 행위를 모델링한다.

20) 컴포넌트

-컴포넌트는 시스템의 물리적인 사항을 모델링하는 데 중요한 요소이다.
-컴포넌트는 일련의 인터페이스를 실현하는 시스템의 대체 가능한 물리적인 부품이다.
-컴포넌트는 실행파일, 라이브러리, 테이블, 파일 및 문서등과 같이 노드에 상주하는 물리적인 대상을 모델링하는데 사용된다. UML에서 모든 물리적인 대상은 컴포넌트로 모델링된다.
-UML은 두 개의 탭을 갖는 직사각형으로 컴포넌트에 대한 그래픽 표현을 제공한다.

(1) 컴포넌트의 이름

-일반적인 클래스의 경우와 동일하다. 컴포넌트는 세부사항을 나타내기 위하여 태그값을 이용하거나 추가적인 구획을 이용할 수 있다.

(2) 컴포넌트와 클래스

-컴포넌트는 많은 점에서 클래스와 유사하지만 이하와 같은 차이점이 있다.
※ 클래스는 논리적인 추상화를 나타내지만 컴포넌트는 물리적인 대상을 나타낸다. 즉, 컴포넌트는 노드에 의존하지만 클래스는 노드에 의존하지 않는다.
※ 컴포넌트는 논리적인 컴포넌트의 패키징을 표현하며 다른 추상화 수준에서 존재한다.
※ 클래스는 애트리뷰트와 오퍼레이션을 직접적으로 가지지만 컴포넌트는 자신의 인터페이스를 통해서만 도달할 수 있는 오퍼레이션만을 갖는다.

(3) 컴포넌트와 인터페이스

-컴포넌트와 인터페이스의 관계는 두 가지 방식으로 표현할 수 있다. 첫 번째는 단순히 아이콘 형식으로 인터페이스를 표현하는 것이고 두 번째는 인터페이스의 오퍼레이션을 보여주는 확장 형식으로 표현하는 것이다.
-어느 경우이든 간에 인터페이스를 통하여 다른 컴포넌트의 서비스에 접근하는 컴포넌트는 종속 관계를 사용하여 인터페이스에 연결된다.

(4) 컴포넌트 다이어그램

-컴포넌트 다이어그램은 객체지향 시스템의 물리적인 측면을 모델링하기 위한 다이어그램이다.
-컴포넌트 다이어그램은 시스템의 정적인 구현 뷰를 모델링하기 위하여 사용된다. 이에는 노드에 상주하는 물리적인 대상에 대한 모델링이 포함된다. 결국 컴포넌트 다이어그램은 시스템의 컴포넌트에 초점을 맞춘 클래스 다이어그램이라고 할 수 있다.

(5) 컴포넌트의 모델링

-컴포넌트는 다수의 실행체와 이에 관련된 객체 라이브러리로 구성된 시스템을 모델링하는데 사용할 수 있다.

-또 수많은 소스 코드 파일과 이들의 관계를 모델링하기 위하여 사용할 수도 있다. 이러한 경우 파일로 스테레오타입되는 작업 산출물 컴포넌트와 이들간의 종속 관계만을 포함한다.
-UML은 논리적인 데이터베이스 뿐만 아니라 물리적인 데이터베이스를 모델링하는 데에도 적합하다.

21) 배치 및 배치 다이어그램

-배치 다이어그램은 객체지향 시스템의 물리적인 측면을 모델링하기 위한 다이어그램이다. 노드와 이들이 상주하는 컴포넌트를 보여주며 시스템의 정적인 배치 뷰를 모델링하기 위하여 사용된다.
-배치 다이어그램은 시스템의 노드에 초점을 맞춘 클래스 다이어그램이라고 할 수 있다.
-배치 다이어그램은 일반적으로 노드와 종속 및 연관 관계를 포함한다.
-노드는 UML에서 입방체로 표현되며 컴포넌트나 클래스와 유사하다.
-배치 다이어그램은 하드웨어 엔지니어와 소프트웨어 개발자간의 의사소통을 원활하 하는데 유용하다. UML의 모든 확장 메커니즘은 노드에 적용되므로 특정한 유형의 노드를 지정할 때에는 스테레오타입을 사용할 수 있다.

-시스템의 토폴로지를 모델링할 때 시스템을 구성하는 프로세서와 장치에 걸친 컴포넌트의 물리적인 분산을 지정하는 것이 유용할 수 있다.

4. Object oriented Analysis(객체지향 분석)

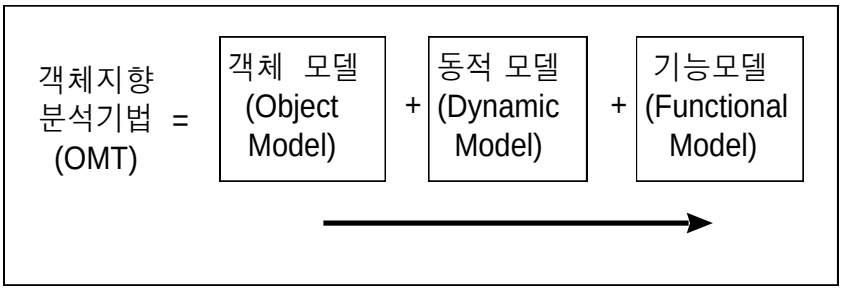
문제를 정의하고 이 정의로부터 객체 모델,동적 모델,기능 모델을 정의함으로써 실세계를 모형화한다. 시스템의 무엇(객체 모델)에서 언제(동적 모델), 무슨일(기능 모델)이 일어나는가를 분석한다.

1) 객체지향 분석의 3관점

(1) **객체 모델링**: 정보 모델링이라고도 부르며 시스템에서 요구되는 객체를 찾아내어 객체들의 특성과 객체들 사이의 관계를 규명

(2) **동적 모델링**: 객체 모델링에서 규명된 객체들의 행위와 객체들의 상태를 포함하는 라이프 사이클을 보여준다.

(3) **기능 모델링**: 각 객체의 형태 변화에서 새로운 상태로 들어갔을 때 수행되는 동작들을 기술하는데 사용



2) 객체 모델링

객체 모델링은 시스템에 요구되는 객체들을 보여줌으로써 주로 시스템의 정적인 구조를 포착하는 데 사용

객체 모델링의 복잡도를 관리하기 위하여 정보 모델링에서 언급된 **추상화, 분류화, 일반화**의 개념들이 사용

또한 **집단화** 개념이 추가되어 다양한 객체들을 모아 더 높은 단계의 객체를 추출하는 작업이 이루어진다.

객체 모델링은 시스템의 요구 사항을 기술한 문제 기술로부터 시스템에 요구되는 객체들을 추출

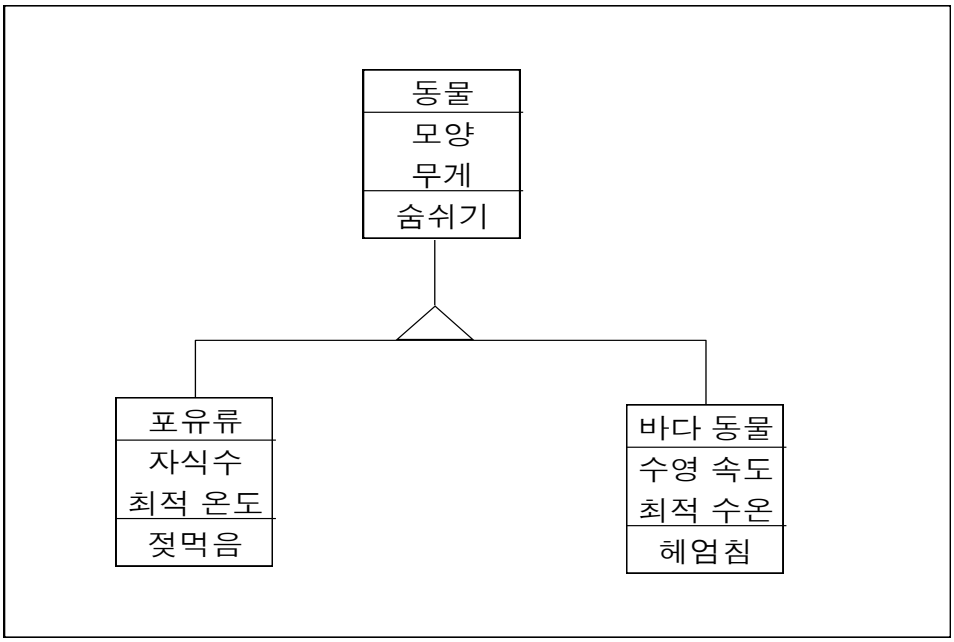
객체들이 발견된 다음 객체들 사이의 연관성을 찾아내고 객체들을 정의하기 위한 속성 추가

- 일반화(Generalization)

일반화는 유사한 클래스들 사이의 공유되는 속성과 동작을 묶어주며, 다른 한편 그들 사이의 다른 점을 보존할 수 있게 하여주는 효과적인 추상화 기법

일반화는 '**is_a**' 또는 '**kind_of**' 관계이며 각 하위 클래스의 인스턴스는 상위 클래스의 인스턴스가 된다.

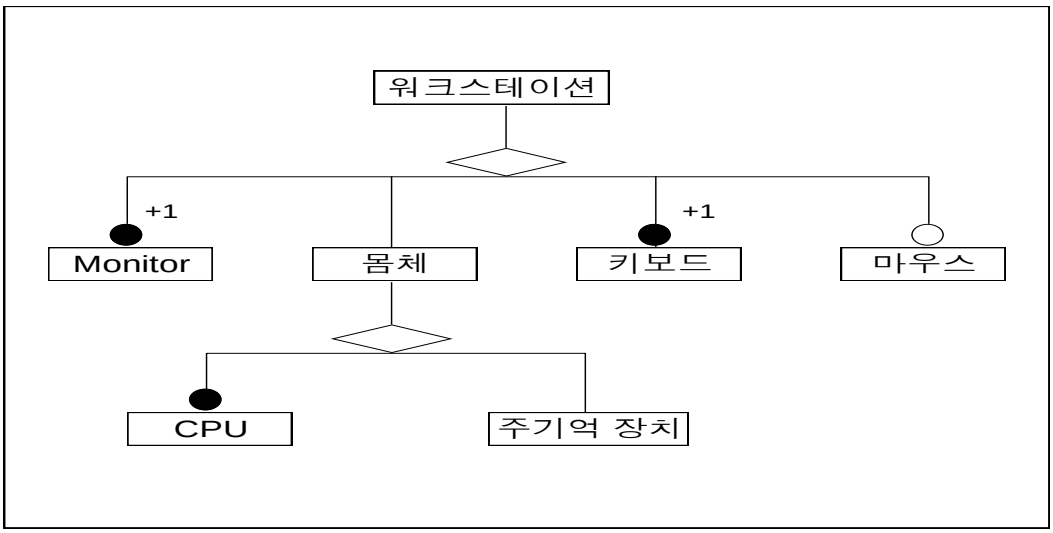
일반화를 통해 **다계층 구조**를 만들어 나갈 수 있으며 이 경우 한 하위 클래스는 이 구조상에 있는 모든 상위 클래스의 속성과 동작을 **상속**받는다.



클래스의 일반화 1

■ 집단화(Aggregation)

집단화는 클래스들 사이의 "**부분-전체**" 관계 또는 "**부분**" 관계로 설명되는 연관성을 나타낸다.집단화 기호는 조그만 마름모 기호로 표시되며 매핑 제약조건과 참여 제약조건에 대한 기호는 일반화의 경우와 동일



다계층 집단화 1

3) 동적 모델링(Dymanic modeling)

객체 모델링은 system을 구성하는 근본적인 객체들을 보여줌으로써 system을 이해하는데 최선의 접근 방법이라 인식되고 있으며, 객체지향 분석에서 가장 우선되어야할 과정이다.

그 다음 시간의 흐름에 따른 객체들과 객체들 사이의 변화를 조사하게 되며 이를 동적 모델링이라 한다. 이는 앞에서 다룬 동적 모델링과 동일하며 동적 모델링은 객체들 사이의 제어 흐름, 상호 작용, 동작의 순서 등을 다룬다. 동적 모델의 중요한 개념은 앞의 실시간 system 모델링에서 다룬 바와 같이 상태(state), 사건(event), 동작(action) 등이다. 객체지향 분석의 특징은 system에 요구되는 객체를 우선 구한 후 객체들의 동적인 면을 찾아내기 위해 동적 모델링을 적용한다는 점이다. 동적 모델은 다음에 나올 기능 모델과 관계가 있으며 system의 동작은 기능을 수행하기 위해 필요하다.

OMT 기법은 동적 모델링에 상태변화도(State Transition Diagram)를 사용한다. 상태변화도를 구하는 과정은 실시간 system의 상태변화도를 구하는 과정과 유사하다. 우선 제안서로부터 시나리오(scenario)를 만들게 되며 시나리오는 객체 또는 system의 한 실행 과정을 사건들의 흐름으로 표시한다.

4) 기능 모델링(Functional Modeling)

기능 모델링은 객체지향 분석기법에서 객체 모델링, 동적 모델링에 이어 system을 기술하는 세 번째 단계이다. 이 세 모델링은 system을 설명하는 삼각대의 세 받침 다리에 해당한다. 객체 모델링은 누구에게 이 일이 필요한지를 보여주며, 동적 모델링은 언제, 어떠한 순서로 일이 진행되어야 하는지를 나타내고, 기능 모델링은 어떤 일이 수행되어야 하는지를 보여준다.

기능 모델은 입력값으로부터 계산을 거쳐 어떤 결과가 나타나는지를 보여주며, 이것이 어떻게(how) 유도되었는지의 구현 방법은 고려하지 않는다(아직 분석 단계임!). 기능 모델은 앞에서 다룬 기능 모델링과 마찬가지로 자료흐름도(DFD: Data Flow Diagram)를 사용할 수 있다. DFD의 경우 입력 흐름은 프로세스를 통해 변환되어 출력 흐름으로 바뀌고, 다른 프로세스의 입력이나 외부로의 출력으로 작용한다.

기능 모델링은 동적 모델링에서 나타난 동작(action)들이 어떠한 기능으로 이루어져 있는지 나타내며, 자료흐름도(DFD)의 정보 흐름은 객체 모델링의 속성에 해당된다.

5. Object oriented Design(객체지향 설계)

1) 목적 : 객체지향 분석시 추출한 클래스와 서브 클래스의 인스턴스들인 객체로 프로덕트를 설계하는 것
2) OOD는 4단계로 구성
   각 시나리오에 대한 상호작용 다이어그램을 구축
   상세 클래스 다이어그램을 구축
   객체들의 클라이언트로 프로덕트를 설계
   상세 설계를 진행
   (1) 각 시나리오에 대한 상호작용 다이어그램을 구축
      UML 상호작용 다이어그램(Interaction Diagram)
      순차 다이어그램과 협동 다이어그램 지원
         같은 사물 즉 객체들과 그들 간에 전달되는 메시지
         a) 순차 다이어그램
         이벤트가 발생하는 순서가 중요한 상황에 유용
         b) 협력 다이어그램
         객체들간의 관계를 강조하고 소프트웨어 프로덕트의 구조를 이해하는데 이용되는 강력한 툴
   (2) 상세 클래스 다이어그램 구축
   ■ 클래스 다이어그램
      클래스들과 그 속성을 나타낸다
      액션(action)을 나타내지는 않는다
      메소드는 OOD 단계의 세부 클래스 다이어그램에 삽입
      프로덕트 액션 결정
         -모든 시나리오의 상호작용 다이어그램 조사 후에 수행
      액션 할당 결정 방법
         -정보은닉에 기반을 둠
      객체지향 파라다임 주요 측면
         -책임 중심 설계원리
         -일단 요청이 수행되었으면 제어는 클라이언트에게 반환
   (3) 객체들의 클라이언트로 프로덕트 설계
      객체 O에 메시지를 보내는 객체 또는 함수 C는 O의 클라이언트
      클라이언트 C로부터 객체 O에게 전송되는 메시지
      화살표로 표시
   (4) 상세 설계 진행
      main과 모든 클래스 개발

6. UML을 이용하는 객체지향 프로세스

일반적인 객체지향 방법론은 분석, 설계, 구현 및 테스트 단계로 이루어진다.

1) 요구사항 분석

요구사항 분석은 시스템의 필요한 기능을 설명하기 위하여 유스 케이스, 비즈니스 프로세스 , 혹은 텍스트 문서를 작성한다. 요구사항 분석에서 대부분의 일은 개발자와 사용자 간의 토의와 협상으로 이루어 진다.

요구사항을 분석할 때 기능적인 사항뿐만 아니라 성능, 신뢰성 등과 같은 비기능적인 사항에 대해서 고려하여야 한다.

요구사항 분석은 개발자와 사용자가 합의한 요구사항 명세서를 작성한다.

2) 분석

분석 단계는 실세계의 엔티티를 모델링 하는 클래스, 객체, 상호작용 등으로 구성되는 문제 도메인에 대한 모델을 생성한다.

분석 문서는 문제 도메인을 설명하는 모델고가 필요한 기능을 제공하기 위한 도메인클래스의 행위로 구성된다. 분석 단계에서 작성되는 UML 다이어그램은 클래스 다이어그램, 순차 다이어그램, 협동 다이어그램, 상태 다이어그램 및 활동 다이어그램이다.

3) 설계

설계는 분석 결과에 대한 기술적인 확장 및 적응이다. 분석과정에서 식별한 클래스, 관계 및 협동은 시스템을 어떻게 구현할 것인가에 초점을 맞춘 새로운 요소에 의하여 보완된다.

세부적인 설계 활동은 클래스의 구현 애트리뷰트, 클래스의 상세한 인터페이스, 오퍼레이션의 설명 등을 포함하는 모든 클래스의 명세서를 포함하여야 한다. 설계 단계에서 작성되는 UML 다이어그램은 클래스 다이어그램, 순차 다이어그램, 협동 다이어그램, 상태 다이어그램, 활동 다이어그램, 컴포넌트 다이어그램 및 배치 다이어그램이다.

추적성, 인터페이스, 성능, 단순성, 문서화등을 기억하여야 한다.

4) 구현

구현 단계에서는 실질적인 코딩을 수행한다. 설계에 대한 최종적인 결정을 내리고, 다이어그램과 명세서를 프로그래밍 언어의 구문으로 전환한다.

5) 테스트

테스트의 목적은 에러를 찾는것이다.

6.참고(Reference)

1) UML 객체지향 분석.설계 – 조완수 저 '홍릉출판사'

2) 객체지향 모델링

[1] 안중호, 고급 정보시스템 분석 및 설계, 서울, 1989.

[2] 송영제, 객체 지향 소프트웨어 설계법에 관한 연구, 대한전자공학회.

[3] 원유현, 객체 지향 언어의 주요 개념, 석사학위논문.

[4] 류형규 외 3인, UML기반 객체 지향 클라이언트/서버구축, 홍릉출판사.

[5] APPLYING UML AND PATTERNS(An Introduction to object- oreiented analysis and design and the unified procss).

[6] R.F.SINCOVEEC: Modular Software construction And Object-Oriented Design Using Ada Journal Of PASCAL/ Ada & Modula-2.

[7] R. Pricto-Diaz And P.Frecman: Classfying Software For Reusability IEEE Software.

[8] 정보과학회지, 제11권 2호(통권52호), ISSN 1015-9908.