객체지향 방법론 심층 비교·해부

분산객체지향 시스템 구축을 위하여는 일반적인 객체지향 시스템 분석/설계 이론과 원격 프로그램 처리를 할 수 있는 미들웨어 구조를 함께 이용할 수 있어야 한다. 객체지향 시스템 구축을 위한 분석/설계 방법론은 그 종류가 많아서 현재까지도, 최적의 방법론을 선정하는데 신경을 많이 쓰고 있는 형편이다. 여기서는 객체지향 분석/설계방법론과 이의 구현을 자바(Java)에 맞추되, 일반적인 객체지향 개념을 비롯해 분산객체를 지원하는 원격 객체의 설계관련 미들웨어, 대표적인 객체지향 방법론의 비교 및 그들의 특성을 설명하고자 한다.

이동진/ 한국CIM 대표이사

1. 객체지향개념

객체지향 개념은 크게 현실세계의 업무(프로세스: 프로그램을 구성함)와 자료(데이터: 파일 또는 데이터베이스를 구성함)로부터 정보시스템을 구성하는 클래스(데이터와 프로그램으로 구성됨)와 그들 간의 관계(구성관계, 유전관계 등)등을 도출하는데 있어서 가장 중요한 기술인 추상화(Abstraction)와 유전(Inheritance) 및 다형성(Polymorphism)을 사용하는 것으로써 이들은 객체지향 본래의 장점인 재사용과 프로그램 생산성 및 유지보수의 편리성을 도와준다.

이러한 개념은 객체지향 프로그래밍을 하는데 있어서 매우 쉽게 코딩이 되게 해주기 때문에, 업무 분석/설계시 반드시 반영되어야 한다.

1.1 추상화(Abstraction)

추상화는 다시 캡슐화(Encapsulation)와 정보은닉(Information Hiding)으로 나뉘어진다. 캡슐화는 데이터와 변수들에 대한 연산(Operations)이 함께 들어 있어서 한 개의 단위로 취급되는 것을 말한다. 이러한 변수들의 묶음을 클래스(Class)라고 부르고, 각각들을 객체(Object)라고 부른다.

일반적으로 객체는 하나의 존재 즉, 인스턴스(Instance)를 의미하고 동일한 특성을 지닌 인스턴스의 집합을 클래스라 부르고 있다. 즉, 클래스는 개념을 객체는 개념의 구체적인 실체를 나타낸다고 할 수 있다. 클래스의 데이터는 속성(attributes)이라고 하며 클래스를 설명하는 내재적, 기본적, 정적인 특성으로 어떤 클래스의 각 객체가 고유의 값을 갖게되는 데이터(상태정보)를 말한다.

연산은 어떤 클래스의 객체에게 요청할 수 있는 처리작업(processing)을 말한다. 요청한다는 의미에서 연산은 서비스(service) 또는 책임(responsibility)으로 이해되기도 하며 클래스의 행동적(behavior), 유동적(dynamic) 측면을 설명한다.

클래스는 외부 클래스가 접근할 수 있는 인터페이스로서 서비스할 수 있는 연산의 리스트를 정의한다. 이 연산 리스트에 정의되어 있지 않는 서비스는 어떤 방법으로도 정의되지 않는다. 즉 데이터와 연산이 캡슐화되었기 때문에 표면에 보이는 연산의 이름만이 외부 클래스가 요구할 수 있는 서비스인 것이다. 클래스의 객체가 상대방이 제공하는 연산을 호출하는 것을 메시지 전달이라고 하며 '연산이름(매개변수1, 매개변수2,...,매개변수n)'의 형태를 가진다.

연산이 수행되는 방법을 규정한 것을 매쏘드라고 하는데 하나의 연산에는 여러 개의 매쏘드가 존재할 수 있다. 아래에서 Components는 Handy-Dandy Robot라는 클래스의 속성이며, Capabilities는 클래스의 연산이다.

정보은닉(Information hiding)은 프로그램을 유지보수 하기에 편리하고, 재사용 하거나 정보사용 특권을 부여할 수 있

<캡슐화의 예>

Handy-Dandy Robot

Components : 1개의 A7 battery, 2개의 Small wheels, 1개의 Surplus computer, 1개의 Vedio camera, 1개의 Radio receiver

Capabilities : Move) Left, Right, Forward, Backward

Stop)

는 좋은 방법이다. 객체지향 프로그래밍 이전에는 프로그래머들이 많은 프로그램들에 공통으로 사용되고 있는 프로그램 단위 모듈들의 수정이 발생할 때 관련된 수많은 프로그램들을 모두 수정해야하는 번거로운 일들에 부딪혔는데, 해당하는 클래스의 모듈하나만 고쳐주면 그것을 사용하는 다른 프로그램들은 전혀 개의치 않고도, 예전의 프로그램들을 사용할 수가 있게 되었다.

아래는 정보은닉의 예를 보여준다.

main함수에서는 지역변수(local variables)가 없고 DoTheFirstThing과 DoTheSecondThing함수에서는 지역변수 a, b가 존재하는데 main함수에서는 a, b를 알 수 없는데 예를 들면, a는 DoTheFirstThing함수에서만 변경이 가능하고 DoTheFirstThing함수에서만 존재하는데 다른 함수들에서는 결코 알지를 못한다.

<정보은닉의 예>

```
main()
{
        DoTheFirstThing();
        DoTheSecondThing();
}
void DoTheFirstThing()
{
        int a = 10;}
void DoTheSecondThing()
{
        int b = 30;
}
```

1.2 유전(Inheritance)

계승이라고도 하며, 객체지향 분석/설계에서 매우 중요한 개념이다. 유전은 기존의 객체들의 특성을 확장하거나 추가할 수 있도록 프로그램 언어에서 제공하는 기능이다. 서로 다른 클래스들이 유사한 속성을 공유하기 위한 추상화 방법으로 일반화(generalization), 특수화(specialization)로 부르기도 한다.

유전은 슈퍼클래스와 특성을 유전 받는 서브 클래스로 나뉘어지는데 서브 클래스들의 공통되는 속성과 연산은 모두 슈퍼클래스에 정의하고 각 서브 클래스는 이를 공유한다. 자바에서는 이미 정의된 기존의 객체들의 특성을 수정하거나 추가할 수 있다.

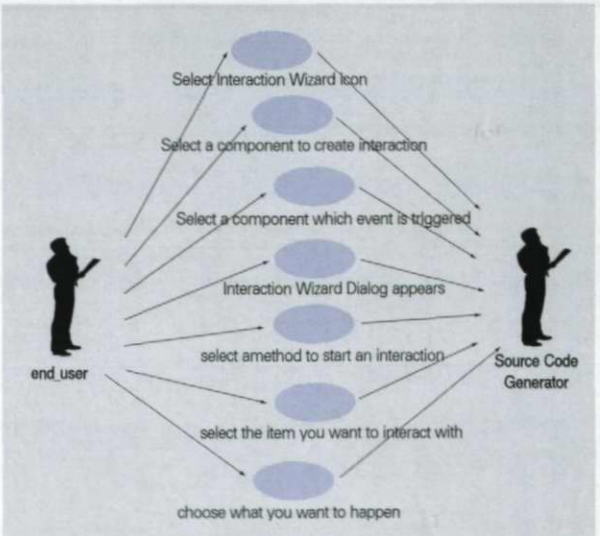
유전은 클래스와 클래스들 사이의 공통 특성들과 속성들을 공유하는 관계로서 매우 중요하다. 유전관계는 서로 관련되는 클래스사이를 삼각형으로 표시한다. 유전관계 이외에 객체지향 개념의 관계에서는 링크(Link), 일반관계(Association, Relation), 구성관계(Aggregation)를 설명하기도 하는데, 링크는 객체사이의 물리적 또는 개념적인 연결을 의미하는 것으로써 여러 유형의 객체를 의미있는 방법으로 연결하는 수단이다.

일반관계는 공통적인 의미와 구조를 가진 링크의 집합으로서 다시 말하면, 관계는 클래스 사이의 물리적 또는 개념적인 연결이다. '이동진은 한국C.I.M에서 일한다.' 와 '사람들이 회사에서 일한다.' 의 문장에서 살펴보면 전자는 링크를 나타내고 후자는 일반관계를 나타낸다.

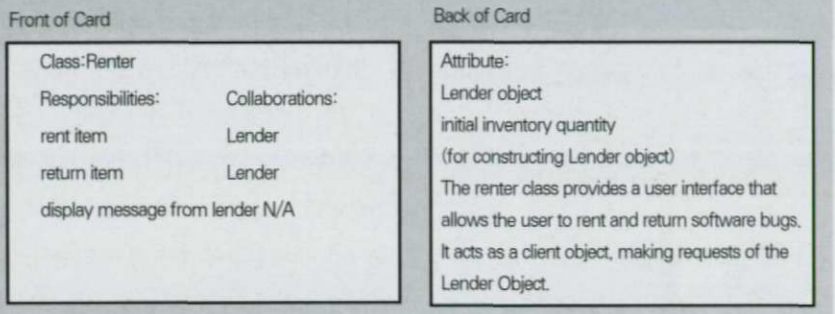
구성관계는 특수형태의 관계로 구성요소가 되는 클래스들과 이들로 조립된 클래스사이의 관계로서 부품-제품(part-whole)관계 또는 부품(part-of, consist-of)관계로 불리기도 한다. 구성관계의 중요한 특성은 전이성(transitivity)이다. 예를 들어 A가 B의 부품이고 B가 C의 부품이면 A는 C의 부품이 된다. 또한 구성관계는 비대칭이다. A가 B의 부품이면 B는 A의 부품이 될 수 없다. 구성관계는 서로 관련되는 클래스들을 다이아몬드 기호로 표시한다.

1.3 다형성(Polymorphism)

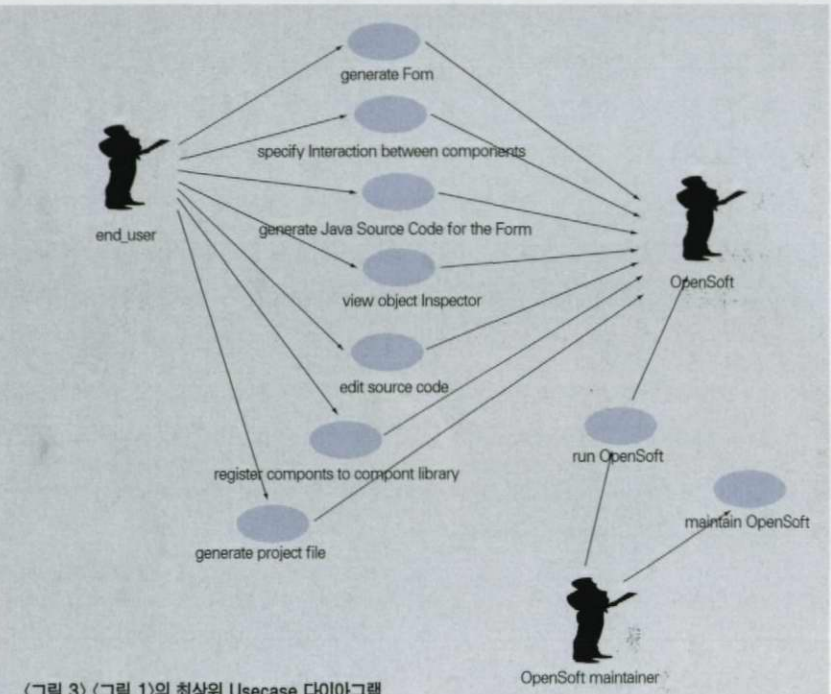
다형성이란 동일한 메시지에 대하여 서



〈그림 1〉 'Specify interaction with components' Usecase 다이아그램



〈그림 2〉 CRC 카드의 예



〈그림 3〉 〈그림 1〉의 최상위 Usecase 다이아그램

로 다른 객체들이 각각 독특하게 반응하게 하는 것이다. 예를 들면 동물 클래스에서 고양이, 개, 소 객체에게 "소리 질러!"라는 메시지를 보냈을 때, 고양이는 "야옹", 개는 "멍멍", 소는 "음매"하고 소리지르게 하는 것이다. 다형성은 한 연산에 대하여 다수의 클래스가 각각 다른 매쏘드를 구현하는 것으로 볼 수 있다.

그러나, 한 클래스내에서 동일한 이름의 연산이 각각 다른 매개변수 별로 정의되는 경우도 있다. 예를 들어 프린터 클래스는 ASCII-Code, Char, Image를 인쇄하는 서로 다른 Print연산을 가지고 있을 수가 있다.

인쇄연산은 호출될 때 주어지는 매개변수의 타입에 따라 그에 맞는 적절한 매쏘드가 수행된다. 이때 인쇄연산과 같이 동일한 이름으로 2개이상의 매쏘드를 구현하는 것을 과적(Overloading)이라고 한다.

2. 자바기반의 객체지향 분석/설계 방법론

자바를 프로그램 언어로 하는 객체지향 시스템을 구축하는데 있어서 객체지향 분석/설계 절차는 〈표 1〉과 같다. 지금까지 발표된 여러 객체지향 분석/설계 방법론 자체가 매우 다양한 기법과 사양(Specification)을 사용하고 있기 때문에 여기서도 분석/설계/구현 시에 도움이 되리라고 여겨지는 내용들을 담았다.

분석단계는 사용자 요구사항을 정의하는 기술에 초점이 맞추어진다. 사용자 요구사항에서 클래스 및 그것의 인스턴스인 객체와 클래스간의 구성 및 유전관계 등을 도출해야 하기 때문이다. 이것에 대하여 Jacobson의 OOSE방법론의 Usecase (사용사례) 모델과 RDD(Responsibility-Driven Design)방법론의 CRC(Class, Responsibility, Collaboration) 카드를 사용하여 표기하는 수가 많은데 〈그림 1〉은 Usecase의 사례를 나타내고, 〈그림 2〉는 CRC카드의 예를 보여준다.

일반적으로 사용자의 요구는 대부분이 연속적인 업무 규칙(Business Rule)의 나열인 경우가 대부분이다.

이것을 두 객체간의 또는 두 클래스간의 관계(일반관계, 구성관계, 유전관계 등)로 표현하는, 다시 말하면 복잡한 문장을 단순한 문장으로 변환시키는 기술이 필요하다.

또한 사용자에 대한 서비스를 표현하는 경우도 Usecase를 사용하여 표현할 경우가 많은데 〈그림 3〉은 〈그림 1〉의 최상위 Usecase 다이아그램이고 〈그림4〉는 〈그림1〉의 Usecase 시나리오이다.

설계단계는 Rumbaugh의 OMT (Object Modeling Technique) 방법론과

0. Shakehands를 실행한다.
1. New를 선택하여 애플리케이션 템플릿을 설정한다.
2. 폼 디자이너에 인터랙션을 설정한 컴포넌트들을 클릭 앤 드롭한다.
3. 인터랙션 위저드의 단축 아이콘(플러그)를 선택한다.
4. 이벤트를 보내는 컴포넌트를 선택한다.
5. 이벤트를 받는 컴포넌트(이벤트 리스너)를 선택한다.
6. 인터랙션 위저드 다이알로그 박스가 생성된다.
7. 다이알로그 박스에서 이벤트 트리거 동작을 선택한다.
8. 다이알로그 박스에서 인터랙션을 하고자 하는 컴포넌트를 선택한다.
9. 두 컴포넌트간의 발생할 인터랙션을 다이알로그 박스에서 선택한다.
10. 확인 버튼을 클릭한다.

〈그림 4〉 〈그림 1〉의 Usecase 시나리오

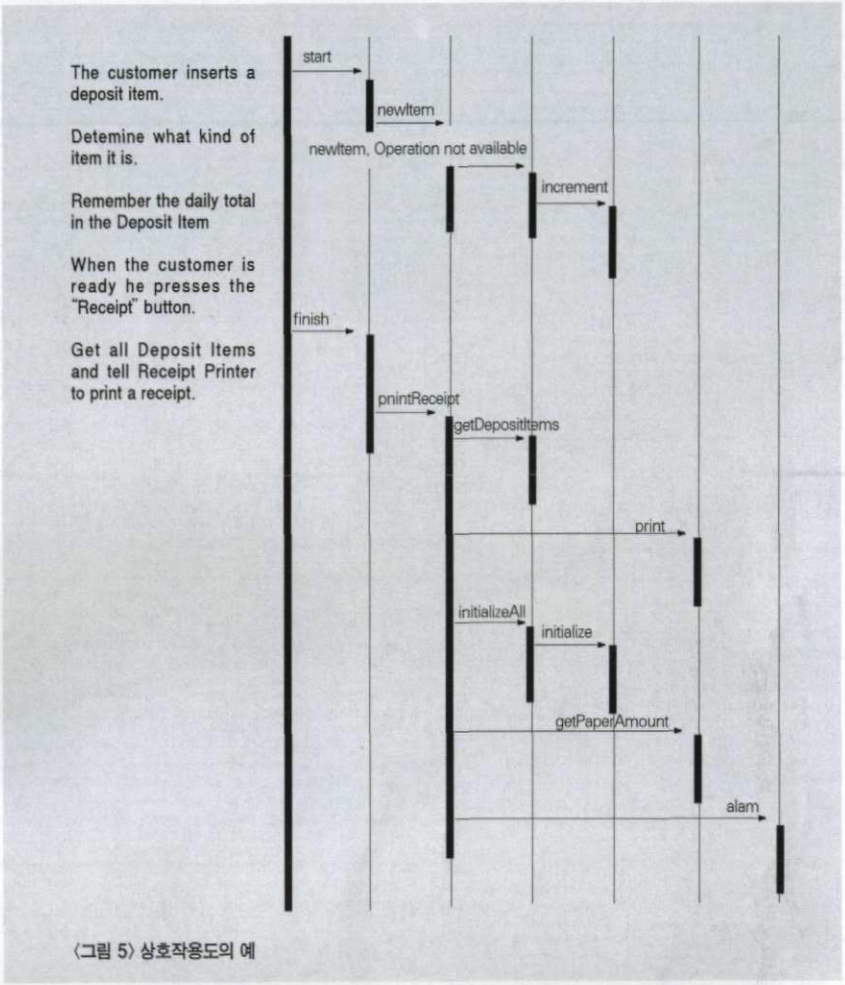
Booch의 방법론을 많이 참조하였는데, OMT방법론은 구조적 방법론인 자료흐름도(Data Flow Diagram)를 기능적 모델로 채택하고 있고, 정보공학 방법론에서 객체-관계 모델을 정적 객체모형으로 채택하였다.

또한 객체지향 방법론에서는 동적 모델인 상태전이도를 모형으로 채택하였으며 Booch의 상호작용도를 채택하고 있다. 〈그림 5〉는 상호작용도의 예를 보여주고 있고, 〈그림 6〉은 상태전이도의 예를 보여주고 있다.

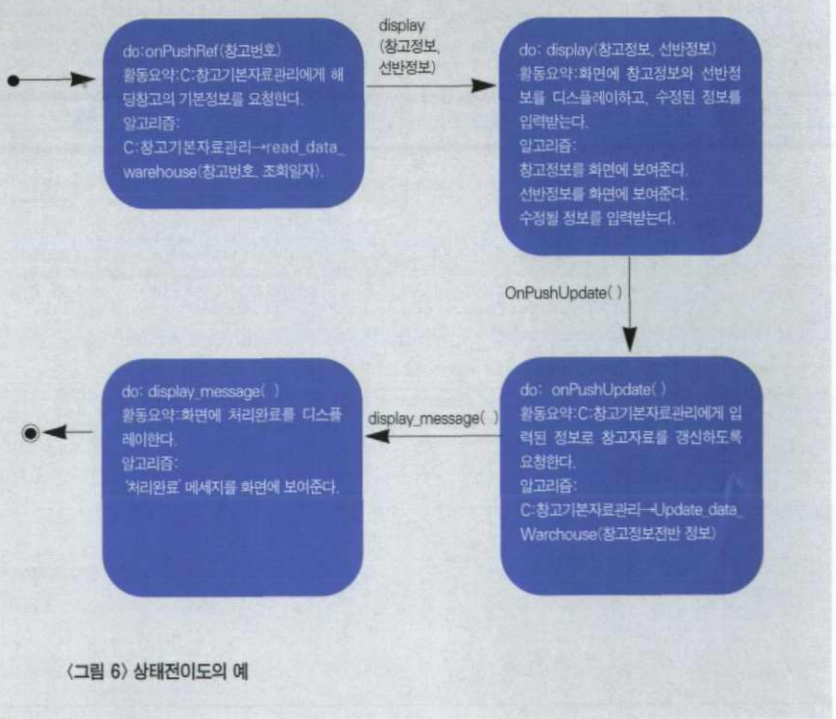
〈그림 7〉은 클래스들끼리의 관계를 나타내는 클래스 다이아그램의 예이다.

이번에는 자바의 특성인 Concurrent object의 설계와 Remote object의 설계에 초점을 맞추어서 설명하고자 한다. Concurrent object의 설계는 자바의 멀티쓰레드 개념을 인용하면 되는데, 〈그림 8〉은 여러 Object들이 동시에 작업을 하는 그림을 나타내 준다.

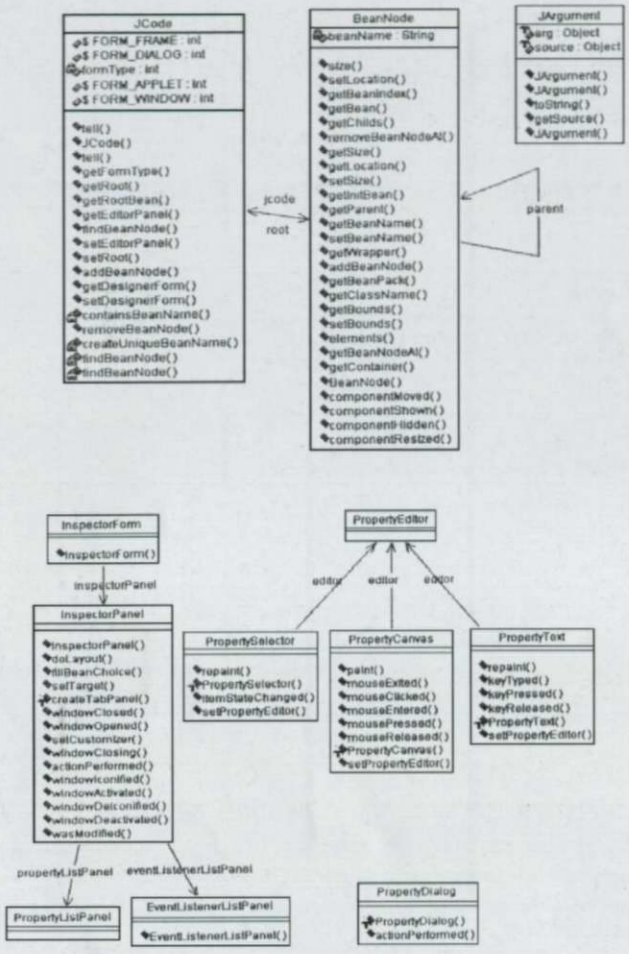
Remote object에 대하여는 Sockets, CGI(Common Gateway Interface), CORBA(Common Object Request Broker Architecture), RMI(Remote Method Invocation), DCOM(Distributed Component Object Model)을 이



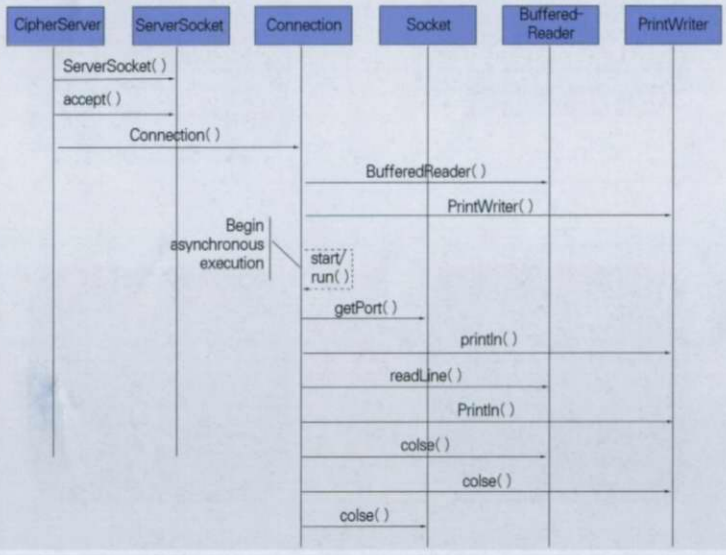
〈그림 5〉 상호작용도의 예



〈그림 6〉 상태전이도의 예



〈그림 7〉 클래스 다이아그램의 예



〈그림 8〉 멀티쓰레드를 표현하는 상호작용도

용하는 방법이 있는데, 자바에 대하여 CORBA를 사용하는 상호작용을 〈그림 9〉에서 보여주고 있다.

3. 대표적인 객체지향 방법론의 비교

객체지향 방법론 중에서 가장 널리 쓰이는 Booch, Rumbaugh의 OMT방법론, Martin/Odell, Shlaer/Mellor, Coad/Yourdon, Wirfs-Brock, Rubin/Goldbug, Jacobson의 OOSE방법론에 대하여 사용된 용어의 비교, 기반

개념에 대한 비교, 지원개념들간의 비교, Champeaux의 평가기준에 의한 비교, Fishman의 평가기준에 의한 비교를 통해 각 방법론들의 특성을 살펴기로 한다.

Rumbaugh의 OMT방법론은 객체모델로서 Entity-Relationship-Model을 채택하고, 동적모델로서는 State-Transition-Diagram을 기능모델로서 Data-Flow-Diagram을 사용하는데 가장 많이 쓰이는 방법론이지만, 개발자의 경험에 의존하고 각 모델을 구축하는 과정의 전이가 부자연스러운 단점이 있다.

Coad/Yourdon은 분석의 5계층과 설계의 4요소를 채택했는데, 분석단계에서는 Class와 Object, Structure, Subject, Attribute, 서비스의 5계층을 사용하고, 설계단계에서는 Problem domain component, Human Interaction component, Task Management component, Data Management component의 4요소를 사용한다.

단점으로는 사용자 인터페이스의 고려부분이 없고, 객체에 의해 수행되는 동적 역할을 체계적으로 표현하는 기법이 없으

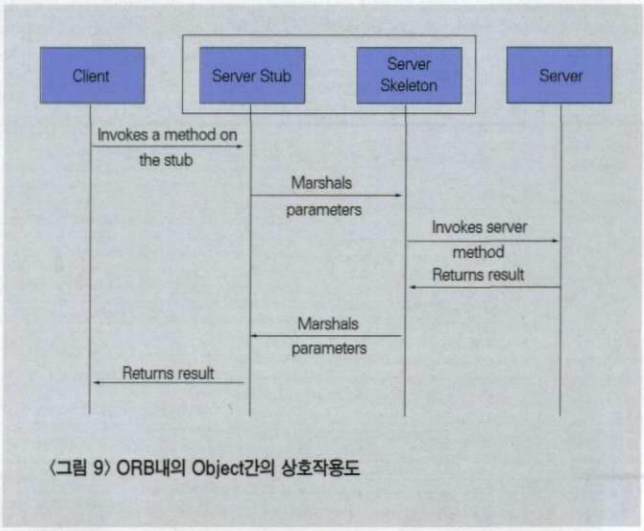
며 객체간의 통신지원기능과 전체 Life Cycle지원이 미약하다.

Booch의 OOSE방법론은 Data flow Diagram을 이용하여 객체로 분해하고 이들 객체간의 인터페이스를 찾아 Ada 프로그램으로 변환시키는 방법을 제안했다. 문제영역에서 객체와 그 특성(속성)을 식별하여 객체들간의 가시성과 인터페이스로 연결함에 의해서 객체를 구현했으며, 논리적인 관점과 물리적인 관점을 포함하는 정적모델과 동적 모델로 나누어 설계했다.

Class와 Object 구조로 표현된 논리적 설계를 통하여 Module과 Process schema의 물리적 설계로 변환한다.

클래스, 객체, 모듈, 프로세스를 사용하여 정적인 관점을 표현하고, 상태전이도(State-Transition Diagram), 시간도(Timing Diagram)를 사용하여 동적인 관점을 표현했다. Ada언어를 이용한 소프트웨어 개발에 초점을 두고 방법론이 개발되었으므로 객체, 클래스, 객체간의 교류에 대한 풍부한 개념을 지원하므로 실시간 처리에 적합하고 대형 프로젝트의 개발에 유리하나 분석단계의 취약성과 설계의 복잡성, 언어 제한적인 면 때문에 구현시 세부사항이 문제가 될 수 있다.

Shlaer/Mellor는 시스템, 도메인(Domain),서브시스템, 객체, 상태, 프로세스의 6계층을 사용하여 시스템을 표현하되 객체의 모듈화를 이용한 전통적인 구조적 방법을 이용한다. 객체와 관계가 표시된 정보모델과 사건과 상태를 나타내는 상태모델, 자료흐름도(Data Flow Diagram)를 사용한 프로세스 모델을 사용한다.



〈그림 9〉 ORB내의 Object간의 상호작용도

ADD(responsibility-Driven Design)는 탐사단계와 분석단계를 집중적으로 표현하는데, 탐사단계에서는 Class, Class의 Responsibility, Class의 Collaboration을 기술하고, 분석단계에서는 Hierarchy, Sub-system, Protocol을 표현한다.

Wirfs-Brock의 RDD방법의 분석은 설계의 일부분밖에 나타내지 못하며, 분석과 설계의 경계가 불분명한 단점이 있다.

Jacobson의 OOSE는 분석초기의 요구모델로서 사용사례(Usecase) 모델과 사용자 인터페이스의 기술, 문제 영역의 객체모델을 사용하고, 분석 모델로서는 인터페이스 객체, 엔티티 객체, 제어 객체를 이용하여 시스템을 기술한다. 설계모델은 분석모델을 지원하는 것으로서 실제의 구현환경을 고려하여 분석모델을 정련(refine)한다. 각 모델은 설계 객체인 블록(Block; 향후에 클래스로 됨)으로 구성되고, 블록구조 완성후 블록의 통신관계를 보여주는 상호작용도의 사용사례를 그린다.

구현모델로서는 블록(Block)을 구현한 소스코드와, 컴포넌트를 구현도구로 이용한다. 특징은 분석단계에 많은 비중을 둔다.

〈표 1〉 자바 기반의 객체지향 시스템 구축 절차

단계	활동사항	입력자료	출력자료
분 석	사용자 요구사항정의	o 현행업무 조사자료 o 정보전략계획 수립서 o 경영전략계획 수립서	o 사용자 요구사항기술서 o 서비스별 요구사항 정의서
	업무영역 분석	o 사용자 요구사항기술서 o 서비스별 요구사항정의서	o 업무영역 기술서
	시스템 이용절차정의	o 서비스별 요구사항정의서 o 업무영역 기술서	o 시스템 이용절차모형(관계도) o 사용자/시스템 상호작용도 o 사용자 인터페이스 정의서 o 행위자 정의서
	인터페이스 객체정의	o 시스템 이용절차모형(관계도)	o 인터페이스 객체모형(객체정의서, 다이어그램)

단계	활동사항	입력자료	출력자료
분 석	분석객체 모형화	o 사용자 인터페이스 정의서 o 시스템 이용절차모형(관계도) o 인터페이스 객체모형 (객체정의서, 다이어그램)	o 분석객체 모형 o 서브시스템 계층도 o 제어객체모형(객체정의서)
	현 기술환경 분석	o 현 전산기술 자료	o 현 기술환경 조사분석서
설 계	시스템 체계설계	o 현행업무 조사자료 o 정보전략계획 수립서 o 경영전략계획 수립서 o 분석객체모형 o 현 기술환경조사 분석서	o 시스템 개발전략서 o 시스템 구성도 o 시스템 구성요소 정의서 o 시스템 배치도 o 시스템 개발전략서
	시스템 이용절차 설계	o 시스템 구성도 o 시스템 구성요소 정의서 o 시스템 배치도 o 시스템 개발전략서 o 시스템 이용절차정의서 o 시스템 이용절차모형(관계도) o 사용자/시스템 상호작용도 o 사용자 인터페이스 정의서	o 시스템 이용절차 모형(설계) o 사용자 인터페이스 레이아웃 정의서 o 사용자/시스템 상호작용도
	객체 모형화	o 분석객체모형 o 시스템 이용절차모형(설계) o 사용자 인터페이스 레이아웃 정의서	o 정적관계 클래스 모형(정의서)
	동적 모형화	o 정적관계 클래스 모형(정의서) o 시스템 이용절차모형(설계)	o 상호작용도 o 상태전이도 o 동적관계 클래스 모형
	기능 모형화	o 상호작용도 o 상태전이도	o 객체 자료흐름도
구 현	사용자 인터페이스 구현	o 시스템 개발전략서 o 사용자 인터페이스 레이아웃 정의서 o 클래스 모형	o 인터페이스 클래스 계층도 o 사용자 인터페이스 뷰
	클래스 선언	o 인터페이스 클래스 계층도 o 사용자 인터페이스 뷰 o 시스템 개발전략서 o 클래스 모형	o 클래스 선언문 o 데이터베이스 정의서
	클래스 오퍼레이션 구현	o 클래스 선언문 o 데이터베이스 정의서 o 상태전이도	o 클래스 오퍼레이션 정의서(구현코드)
	실행 프로그램 구현	o 클래스 오퍼레이션 정의서 (구현코드) o 클래스 선언문 o 데이터베이스 정의서 o 상호작용도 o 서브시스템 계층도	o 프로그램 구성표 o 실행 프로그램 o 사용자 인터페이스 뷰

〈표 2〉 사용된 용어의 비교

Booch	Rumbaugh	Martin/Odell	Shlaer/Mellor	Coad/Yourdon	Wirfs-Brock	Rubin/Goldbug	Jacobson
객 체	객 체	객 체	객 체	객 체	객 체	객 체	객 체
클래스	객체클래스	객체타입	객 체	클래스	클래스	객 체	클래스
자 료	속 성	자 료	속 성	속 성	속성	특 성	속 성
사 용(Use)	연관관계	연관관계	관련성	인스턴스 연결	획득관계	연관관계	협력관계
포함관계	집단관계	합성관계	x	부분-전체	구설관계	x	x
상 속	일반화	서브타입	서브타입	일반화/상세화	상 속	일반화/상세화	상속/계승
메시지	사 건	사 건	사 건	메시지	자 극	행위	메시지
오퍼레이션	오퍼레이션	오퍼레이션	오퍼레이션	서비스	오퍼레이션	활동	메소드

〈표 3〉 기반 개념에 대한 비교

시스템관점	Booch	Rumbaugh	Martin/Odell	Shlaer/Mellor	Coad/Yourdon	Wirfs-Brock	Rubin/Goldbug	Jacobson
정적구조	클래스/객체구조	객체모형	정적분석	정보모형	다단계모형	정적모형	정적모형	정적모델
		의미모형	의미모형	의미모형	클래스/객체구조	클래스	클래스	클래스/객체구조
	속성	속성	속성	속성	속성	속성	특성	속성
	오퍼레이션	오퍼레이션	오퍼레이션	오퍼레이션	서비스	메소드	행위	오퍼레이션
	관련성	관련성	관련성	관련성	관련성	관련성	관련성	관련성
동적구조		동적모형	행위분석	상태모형	서비스단계	행위분석	동적모형	동적모형
	STM(Mealy)	STM(Hatel)	이벤트모형	STM(Moore)	서비스차트	CRC모형	STM(Harel)	사용사례모형
구조적관점	객체분할	기능모형	기능분할	프로세스모형	주제단계	x	x	기능분할

〈표 4〉 지원개념들간의 비교

지원개념 / 방법론	Booch	Rumbaugh	Martin/Odell	Shlaer/Mellor	Coad/Yourdon	Wirfs-Brock	Rubin/Goldbug	Jacobson
타입	O	O	O	O	O	O	O	O
패키지	O							O
단일상속	O	O	O	O	O	O	O	O
다중상속	O	O	O	O	O	O		
서브타입	O	O	O	O		O		O
추상클래스	O	O				O	O	
합성집단화	O	O	O		O	O	O	O
객체간의 상호작용	O	O	O	O				
객체의 생명주기	O	O		O				O

〈표 5〉 Champeaux에 의한 평가

평가요소 / 방 법 론	Booch	Rumbaugh	Martin/Odell	Shlaer/Mellor	Coad/Yourdon
객체의 식별과 분류	클래스/객체 다이아그램	클래스 관계 다이아그램	객체 관계성/ 일반화 다이아그램	정보 구조 다이아그램	클래스/객체 다이아그램 Layer 1
일반화/상세화 집단화/원소화 관련성	클래스/객체 다이아그램	클래스 관계 다이아그램	객체 일반화/ 구성 다이아그램	정보 구조 다이아그램	클래스/객체 다이아그램 Layer 2
다른 관련성	클래스/객체 다이아그램	클래스 관계 다이아그램	객체 관계성 다이아그램	정보 구조 다이아그램	클래스/객체 다이아그램 Layer 3
객체의 속성	클래스/객체 다이아그램	클래스 관계 다이아그램	객체 관계성 다이아그램	정보 구조 다이아그램	클래스/객체 다이아그램 Layer 4
상태와 전이 기능과 서비스에 대한 상세한 기술	상태전이/프로세스 다이아그램	클래스 속성 다이아그램	상태전이 다이아그램	객체 상태 차트 Action 자료흐름도	서비스 차트
서비스 식별	프로세스 다이아그램	상태전이도 자료흐름도	상태전이 다이아그램	Action 자료흐름도	클래스/객체 다이아그램 Layer 5
통신	클래스/객체 다이아그램	클래스 관계 다이아그램	객체흐름 다이아그램	객체통신 모형(OCM)	클래스/객체 다이아그램 Layer 5

〈표 6〉 Fishman의 평가기준에 의한 평가

지원개념 / 방법론	Rumbaugh	Martin/Odell	Shlaer/Mellor	Coad/Yourdon	Wirfs-Brock	Rubin/Goldbug	Jacobson
상 속	O	O	O	O	O		O
다중상속	O	O	O	O	O		O
속성정의	O	O	O	O	O		O
집단관계	O	O		O	O		
관련성 및 함수	O	O	O	O	O	O	O
상태전이 다이아그램	O	O	O	O			O
이벤트	O	O	O	O	O	O	O
이벤트추적	O	O	O				O
트리거	O	O					
자료흐름도	O	O	O	O			O
전체적인 병행성	O	O	O				
한 객체내에서의 병행성	O	O					
도 구			O				O
통합된 도구들의 집합							O
해당 항목수	11	12	11	8	6	2	10