□특 집□

객체지향 설계 방법론의 비교 분석

강원대학교 전자계산학과 양해술* · 조영식** · 이용근**

목 차

Ⅰ. 서 언

소프트웨어 공학의 주요한 목표중의 하나는 품질 좋은 소프트웨어를 생산하기 위한 방법론을 개발하는데 있다. 현재, 이와같은 목표에 근접하기 위한 소프트웨어 개발 방법론이 많이 연구되고 있다.

지금까지 가장 널리 이용되는 개발 방법론은 구조적 기법(structured method)이라고 할 수 있다. 그러나, 구조적 기법은 소프트웨어 재사용성이나 모듈화 또는 개발된 소프트웨어 유지 보수에 효과적인 해결 방법을 제시하지 못하는 단점이 있어, 대규모 소프트웨어의 개발보다는 소규모 내지 중간 규모 이하의 소프트웨어를 개발하는데에 적합한 방법이라고 할 수 있다. 따라서, 본 연구에서는 최근에, 관심이 고조되고 있는 소프트웨어 개발 방법 중, 많은 연구가 진행되고 있는 객체 지향 소프트웨어 개발 방법론(Object-Oriented Software Development Methodology)에 대하여 살펴보고, 현재까지 연구된 객체 지향 설계(OOD) 방법론의 비교분석과 이 방법론들의 적용 방법에 대해서 살펴보기로 한다.

또한, 객체 지향 설계의 평가 요소(critical components)를 동일하게 하여, 다양한 객체 지향 설계기술과 표현 방법(Representation)을 분석하여 객체 지향 설계 방법론의 장단점을 살펴보고, 앞으로의 객체 지향 설계에 대한 연구방향에 대하여 논의하기로 한다.

그러나, 구체적인 프레임워크나 메카니즘에 대한 상세한 설명은 피하고, 현재까지 연구된 객체 지향 설계 방법론의 비교 평가를 위한 프레임워크를 구성하며, 구성된 프레임워크는 객체 지향 설계 방법론과 다양한 연구분야를 융합하기 위한 기본적인 사항을 제공한다.

따라서, 본 연구에서는

첫째, 객체 지향 설계 방법론에서의 완전하고 응집력있는 평가 요소 식별과,

둘째, 첫번째의 요소를 기반으로 객체 지향 설계 방법론과 표현에 대한 평가와,

셋째, 객체 지향 설계 방법론의 장단점에 대하여 살펴보고자 한다.

Ⅱ. 객체 지향 설계(OOD)

지금까지 연구되어진 객체 지향 설계 방법론은 다

* 종신회원
** 준회원

음과 같이 크게 세 가지로 분류할 수 있다.

(1) Process
(2) Representation
(3) Process와 Representation

Process는 객체 지향 설계를 위한 처리절차와 방법에 대해서만 기술하고, 객체 지향 설계 다이아그램(diagrams)이나 표기법(notation)에 대해서는 기술하지 않은 것이다. Representation은 객체 지향 설계 결과를 기술하기 위한 그래픽 또는 다이아그램(diagrams)에 대한 표기법만을 기술한 것이며, "어떻게 설계되고 있는가?"에 관한 것 뿐만 아니라, "시각적인 설계 표현을 어떻게 할 것인가?"에 중점을 두고 있다. Process와 Representation은 객체 지향 설계를 위한 Process와 결과를 기술하기 위한 Representation의 내용을 포함하여 기술한 것이다.

〈표 1〉은 위의 세 요소를 이용하여 현재까지 연구된 방법론과 연구자들을 분류하여 나타낸 것이다.

다양한 시스템 개발 방법론은 전통적인 폭포수형 모델[23]과 fountain 모델처럼 다른 시스템 라이프 사이클(life cycle)과 대응되는 것을 알 수 있다. 시스템 개발 방법론은 각각의 다른 모델과 개발 방법론이 있음에도 불구하고, 그것에 대한 설계, 구현에 대응되는 모든 요소를 가지고 있다. 따라서, 본 연구에서는 구현 또는 물리적인 설계에 대한 언급을 피하고, 구성요소를 이용하여 객체 지향 시스템 개발을 위한 논리적인 설계에 중점을 두고 설명하기로 한다.

〈표 1〉 Process, Representation, Process와 Representation 분류

CATEGORY 1
Process only (no Representation) Bulman [6] Henderson-Sellers and Constantine [11] Johnson and Foote [14] Scharenberg and Dunsmore [25]
CATEGORY 2
Representation only (no technique) Ackroyd and Daum [1]-a graphical notation Beck and Cunningham [4]-Class-Responsibility Collaboration (CRC) cards Cunningham and Beck [9]-Message diagrams Page-Jones et al [22]-Uniform Object Notation (UON) Wasserman et al. [27]-OO Structured Design Notation (OOSD) Wilson [28]-Class diagrams
CATEGORY 3
Process and Representation Alabiso [2]-Transformation from Analysis to Design (TAD) Bailin [3]-Object-Oriented Requirements Specification Method Booch [5]-Object-Oriented Design Coad and Yourdon [7]-Object-Oriented Analysis (OOA) and Design (OOD) Gorman and Choobineh [10]-Object Oriented Entity Relationship Model(OOERM) Ilvari [13]-A Framework for Object identification kappel [15]-Object/Behavior Model Lieberherr et al. [18]-The Law of Demeter Meyer [19]-Object-Oriented Software Construction Rumbaugh et al [24]-Object Modeling Technique (OMT) Shlaer and Mellor [26]-Object-Oriented Systems Analysis (OOSA) Wirfs-Brock et al. [29]-Designing Object-Orientd Software

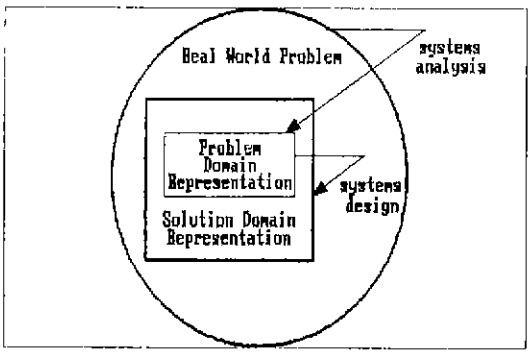
객체 지향 시스템 개발은 크게 분석과 설계로 나눌 수 있다. 분석은 문제 정의/모델링에 중점을 두고, 설계는 해결을 위한 명세/모델링에 중점을 둔다. 설계는 문제 표현(problem Representation)을 해결 표현(solution Representation)으로 변형시킨 것이다.

(그림 1)에서는 분석과 설계에 대한 관계를 나타내고 있다. 문제 영역과 해결 영역에 대한 표현 방법은 차이가 있으나, 해결 영역은 실세계의 문제보다는 작고, 문제 영역의 구성요소 모든 것을 포함하고 있으며, 해결 영역에서 요구되는 구성요소를 문제 영역에 추가한 것이다. 예를들면, 사용자 인터페이스는 문제 영역과 해결영역 모두에 포함될 수 있다.

Colter[8]과 PrColter[8]는 분석과 설계 방법론의 특성들에 대하여 많은 연구를 하였으며, 객체 지향 설계에서 객체의 활동 영역을 확장하고 지원하기 위해서 다음과 같은 특성을 포함해야 한다고 하였다.

(1) 객체 지향 설계 프로세스 : 해결 영역 모델링 기술
(2) 객체 지향 설계 표현 : 다른 추상화 레벨에 대한 구조, 기능, 제어
(3) 객체 지향 설계 추상화 복잡도와 메카니즘 관리 : 시스템의 복잡도 관리와 문제 분류

객체 지향 설계 방법론들에 대한 평가는 객체 지향 패러다임에서 분석과 설계의 구별이 명확하지 않기 때문에 올바른 평가가 이루어지기 매우 어렵다. 즉, 객체 지향 설계 연구에 있어서 문제 영역(분석)과 해결 영역(설계)을 명확하게 구분하는 것이 어렵기 때문에 실제 객체 지향 분석의 종료(end)와 객체 지향 설계의 시작(start)에 대한 구분이 매우 어렵다. 따라서, 대



(그림 1) 분석과 설계에 대한 관계

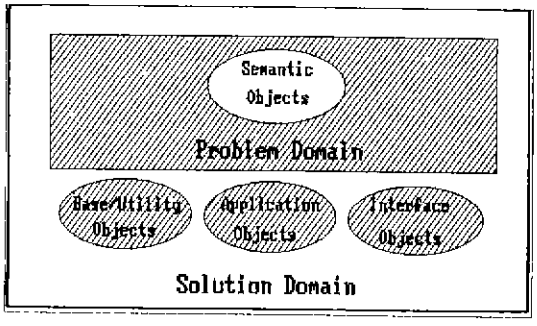
부분의 연구는 객체 지향 분석보다는 객체 지향 설계에 중점을 두고 많은 연구가 진행되어 왔다.

문제 영역 모델은 시스템 요구에 대응하기 위해 필요한 각 객체의 행위와 내부 객체간의 메세지 전송을 상세화한다. 해결 영역 모델은 문제 영역의 클래스를 검토/조사하여 재사용성과 상속의 장점을 이용함으로써 객체 지향 설계의 성능을 향상시킨다. 일반적으로 객체 지향 분석자는 객체의 구조에 관심을 가지고 있으며, 객체 지향 설계자는 객체의 동적행위(dynamic behavior)에 관심을 가진다.

〈표 2〉는 객체 지향 설계 프로세스에 대해 연구한 저자와 그들에 의해 연구된 주요 내용이다. 내용에서 공통적인 내용중의 하나는 문제 영역 객체는 식별되어야 하며, 그 의미가 정의되어야 한다는 것이다. 객체 식별을 제외하고는 문제 영역에서 포함되는 다른 기능들과는 거의 일치하지 않는다. 분석은 객체 지향 분석의 문제 영역 객체에서 처리되고, 설계는 객체 지향 설계의 해결 영역 객체에서 처리된다. (그림 2)에서는 문제영역과 해결 영역 객체의 관계를 나타내고 있다.

〈표 2〉 OOD 프로세스에 대한 예

객체 지향 설계 프로세스	
Booch [5].	Shlaer and Mellor [26].
◦ Identify objects ◦ Identify semantics of objects ◦ Identify relationships ◦ Implement and iterate	◦ Bulid system architecture design ◦ Build data structure design
Wirfs-Brock et al [29].	Bulman [6].
◦ Initial exploration (classes, resp., .) ◦ Detailed ◦ Analysis (Relationships,. .) ◦ Build subsystems	◦ Identify candidate object ◦ Specify objects ◦ Refine objects and operations ◦ Complete operations ◦ Design object oriented structure



(그림 2) 문제 영역과 해결 영역의 관계

문제 영역(분석)은 사물(thing)이나 개념(conception) 문제를 기술한 것이며, 문제 영역에서의 의미를 가지고 있기 때문에 의미 객체(semantic object)라고도 한다. 예를 들면, customer, order, item 그리고 employee와 같은 의미 클래스(semantic classes) 시스템 들이 여기에 속한다.

해결 영역은 의미 객체(semantic object)를 포함하며, 문제 영역에 대해서는 제한을 받지 않는다. 설계시에는 문제 정의에 중점을 두며, 의미 클래스(semantic classes)는 유용하게 추상화되어 찾기 쉽기 때문에 설계시 확장 가능하다. 예를 들면, 추상화 클래스 person은 customer와 empolyee의 수퍼 클래스로 구성될 수 있다. 또한, 해결영역은 Interface, Application, Base/Utility 객체들을 포함한다.

Interface 객체는 사용자 인터페이스와 관련된 것으로 문제 영역의 직접적인 일부분은 아니지만, 의미 객체에서 사용자 관점을 기술한 것이다. Application 객체는 시스템 메카니즘에 대한 감독 또는 제어가 가능하며, Application과 상위 레벨(high-level)의 기능을 순차적으로 제어 가능하게 하는 객체이고, 절차형 언어에서 메인 모듈에 해당한다. 객체 지향 설계에서 Application 클래스는 다른 객체에 초기 메세지를 전송하거나 메뉴 유도에 많이 이용된다. Base/Utility 객체는 독립적인 응용 요소이며, Base/Utility 객체의 메인 특성은 응용과 영역에서 독립적이다. Base 객체의 예로써 collection (string, arrays, etc), number 등이 있고, Utility 객체에 대한 예로 Smalltalk의 윈도우 debugger가 있다.

III. 객체 지향 설계 구성요소

〈표 3〉은 앞절에서 기술한 객체 지향 설계에 대한 평가 요소를 상세히 분류한 것이다. 이와 같은 평가 요소는 Colter/Pressman의 평가 요소를 기반으로 구성하였다. 〈표 3〉에서는 정적(구조적인 측면)과 동적(제어에 대한 측면)을 고려한 세 분야로 나누어 각각에 대하여 정의하였다. 예를 들면 (1)(a)와 (1)(b)는 구조에 대하여 정의하였고, (1)(c)는 제어에 대하여 정의하였으며, (2)는 (1)과 동일하고, (3)은 복잡도 관리에 대하여 정의하였다.

객체 지향 설계 프로세스는 의미 클래스(semantic classes), Interface, Application, Base/Utility 객체에 의해서 객체 지향 설계시 정의되며, 객체 지향 설계 프로세스에서 클래스(class), 애트리뷰트(attribute)와

〈표 3〉 객체 지향 설계에 대한 평가 요소

(1) 객체 지향 설계 프로세스
(a) Identification of: Semantic classes Attributes Behavior Relationships (e.g., generalization, aggregation, association) Interface Application Base/Utility (b) Placement of: Classes Attributes Behavior Optimization of classes (c) Specification of dynamic behavior
(2) 객체 지향 설계 표현(Representations)
(a) Static view: Object Attributes Behavior Relationships: Generalization Aggregation Other (e.g., Association) (b) Dynamic views: Communication Control/Timing (c) Constraints: On structure (attributes values, relationship cardinalities) On dynamic behavior
(3) 객체 지향 설계 복잡도 관리
(a) Mechanism for conceptually managing structural complexity (b) Mechanism for conceptually managing behavioral complexity (c) Representation of system level: Static structure Dynamic Behavior/Control

메소드(즉, behavior)를 식별하는 것은 매우 중요하다.

그리고, 배치(placement)에서 애트리뷰트와 메소드는 사전에 정의된 관계 또는 구조화된 클래스로 할당된다. 또한, 관계는 일반적으로 많은 객체 중에서 일대일 대응되는 관계를 나타내는 AKO(A-Kind-Of: generalization)와 APO(A-Part-Of: aggregation)가 있으며, 일대 다수의 관계를 나타내는 Association 관계가 있다. 그 외에도 AEO(An-Element-Of)와 AVO (A-View-Of)가 있다[1][5].

많은 객체 지향 프로그래밍 언어는 다중 상속(multiple inheritance)을 하지 않는다. 그러나 설계 단계에서 구현에 대한 독립과 다중 상속이 최선의 표현 방법이라면, Association 관계 모델을 이용하여 다중 상속 표현을 가능하게 한다. 마찬가지로, AKO는 대부분의 객체 지향 프로그래밍 언어를 지원하기 위한 표현 방법중의 하나이다. 그러나, 다른 관계는 문제 영역과 해결 영역의 의미 객체 설계를 기술하는데 활용 가능하다. Association 관계는 다중 상속이 가능한 실질적인 언어에서 구현 가능하다.

설계에서 모든 요소들은 순서에 중요한 의미를 부여하지 않는다. 클래스, 애트리뷰트, 메소드는 어떠한 순서로도 식별될 수 있으며, 어떠한 순서로 정의되어도 상관이 없다. 그러나, 분명한 것은 애트리뷰트 또는 메소드는 클래스가 정의될 때까지는 클래스에 할당될 수 없으며, 클래스들간의 관계가 정의되어 있지 않으면 관계는 성립되지 않는다.

어떤 저자들은 순차적인 프로세스 작업이 최선이라고 주장한다. 그러나, 경험에 의하면 애트리뷰트, 메소드, 클래스의 할당(allocation)과 발견(discovery)은 반복적이고, 비순서적인 프로세스이다. 따라서, 다양하고 효과적인 전략으로 분석자/설계자에 대한 문제 영역의 특성에 대한 경험과 배경에 높게 의존하는 것을 볼 수 있다. 그리고, 어떤 전략적인 장점에 의존하여 프로세스 작업을 처리하는 경우도 많다.

객체 지향 설계 표현 방법으로 Macro와 Micro 레벨에 대한 시스템의 정적(static)과 동적(dynamic)인 관점이 필요하다. 정적 표현은 객체, 관계, 애트리뷰트, 그리고 메소드에 대해서 기술하고 동적 표현은 메세지 패싱과 제어에 대하여 기술한다. 시스템에서 애트리뷰트 값, 관계 수, 오류 처리는 표기법(notation)을 이용하여 자동적으로 기술된다.

객체 지향 설계에서 복잡도 관리는 개념상 또는 시각적인 입장에서 중요하다. 시스템에 대한 다른 관점(standpoint)은 복잡한 설계 즉, Micro와 Macro 또는 정적(static)과 동적(dynamic)에 대하여 의미있고 폭넓은 관리를 도와줄 수 있도록 정의되어야 한다.

Micro 레벨은 각각의 클래스와 클래스들간의 상호 관계에 대한 내용을 포함하고, Macro 레벨은 클래스의 그룹을 포함한다. 따라서 데이타 플로우 다이아그램에서의 멀티 레벨처럼 멀티 Macro 레벨이 가능하며 Micro 레벨들은 다른 평가 요소(예를 들면, 집합화, 일반화)에 의하여 구성된다.

Macro와 Micro 레벨은 정적과 동적인 특성을 가지며, Micro 레벨 다이어그램은 명확하게 객체로 구성할 수 있는 반면에 Macro 레벨 다이어그램은 명확하게 구성할 수 없다.

복잡도 관리는 다음과 같은 것을 요구한다.

(1) Macro 레벨 다이어그램을 위한 기본적인 구성 요소가 정의되어야 한다.
(2) 구성 요소를 추출하거나 식별하기 위한 가이드라인이 제공되어야 한다.
(3) 도표 형태로 표현하려면, 정적/동적 특징과 하위 레벨로 구성된 관계를 포함하여야 한다.

이상과 같은 구성 요소로 여러 메카니즘이 개발되어 왔다. 메카니즘에 대한 예로는 Coad와 Yourdon's[7]의 subject, Wirfs-Brock *et al.*의 subsystem[29], Meyer의 clusters[19]가 있다. 이와 같은 메카니즘들은

〈표 4〉 객체 지향 설계에 대한 분석

	PROCESS AND REPRESENTATION										REPRESENTATION ONLY				PROCESS ONLY			
	Alabiso	Bailin	Booch	Coad and Yourdon	Gorman and Chocbineh	Ivari	Kappel	Lee & Carver	Rumbaugh et al.	Wirfs-Brock et al.	Ackroyd & Beck	Beck and Cunningham	Wasserman et al.	Wilson	Bulman	Henderson-Sellers and Constantine	Johnson and Foote	Scharenberg and Dunsmore
1. OOD PROCESS																		
(a) Identification of:																		
Semantic classes	√	√	√	√	√	√	√	√	√	√					√	√	√	√
Attributes			√	√	√			√										
Behavior	√	√	√	√	√	√		√	√	√					√	√	√	√
Interface classes				√		√									√			
Application classes																		
Base/Utility classes	√						√		√						√			
Relationships:																		√
Generalization			√	√	√			√	√	√					√	√		√
Aggregation			√	√					√	√								
Other			√						√	√								
(b) Placement of:																		
Classes				√	√					√								
Attributes				√														
Behavior	√	√	√				√	√	√	√					√		√	
Optimization of classes	√		√	√					√	√					√		√	√
(C) Specification of:																		
Dynamic behavior (i. e., message passing)	√		√						√									
2. REPRESENTATIONS																		
(a) Static View																		
Objects	√	√	√	√	√	√	√	√	√	√	√	√	√	√				
Attributes	√		√	√	√		√	√	√				√	√				
Behavior	√	√	√	√	√	√	√	√	√	√	√	√	√	√				
Relationships:																		
Generalization	√		√	√	√	√	√	√	√	√			√	√				
Aggregation			√	√		√	√		√									
Other			√				√		√		√	√	√					
(b) Dynamic View																		
Communication	√		√	√	√	√	√			√	√		√					
Control/Timing	√		√	√	√		√		√				√	√				
(c) Constraints																		
On structure			√	√	√				√		√		√					
On dynamic behavior			√				√		√									
3. COMPLEXITY MGT.																		
(a) For structural complexity			√	√		√			√	√							√	
(b) For behavioral complexity										√								
(c) Representation of:																		
Static structure			√	√						√								
Dynamic behavior																		

구조 복잡도를 관리한다.

〈표 4〉의 내용은 〈표 3〉의 각 구성 요소들에 대한 연구자들의 연구결과를 기반으로 구성하였다. 〈표 4〉에 표시된 부분은 평가 요소 결과에 대해서는 언급하지 않았다. 〈표 4〉는 연구 결과에 연구자들이 기술한 것이고, 빈 공간인 부분은 그 결과에 대하여 등급을 매긴 것이 아니라 단순히 그 연구 결과에 대하여 고려한 것을 살펴본 것이다.

IV. 객체 지향 설계에 대한 분석

4.1 객체 지향 설계 프로세스(Process)

앞절에서 기술한 객체 지향 설계에 대한 대부분의 분석 요소는 많은 연구자들에 의해 연구되어 왔다(〈표 4〉의 1절 참조). 그러나, 객체 지향 시스템에서 지원되는 특징중의 하나인 캡슐화에 의한 애트리뷰트의 행위를 기술하지 않았다는 것이 매우 흥미롭다. 즉, 객체 자신의 구조와 행위를 감추고 단지, 객체는 인터페이스를 통해서만 액세스 가능하다는 것이다. 이와 같은 이유 때문에 몇몇 연구자들은 애트리뷰트 이름이나 객체의 내부 구조를 그다지 중요하게 생각하지 않는다. 일반적으로 객체가 다른 객체와 인터페이스하기 위해 데이타와 행위(behavior)를 캡슐화하는 것은 좋은 방법이다. 그러나 개념 레벨에서 객체의 애트리뷰트는 객체의 의미 정의에 대한 통합적인 측면에서 매우 중요하다. 예를 들면, 고용인에 대한 논리적인 특성을 자연스럽게 표현할 수 있는 이름, 주민등록번호 등이 있다. 이것은 고용인 객체 구조의 일부분으로써 모델화가 가능하다.

객체 지향 설계 프로세스에서 객체를 의미 클래스(semantic classes), Interface, Application, Base/Utility로 분류하였다. 몇몇 연구자들은 본 연구에서 분류한 객체와 유사한 객체들에 대해서 기술하였다. 예를 들면, Wirfs-Brock *et al.*은 객체의 구성 요소중 많은 시스템에서 재사용될 수 있는 부분을 블랙박스로 묘사하였으며, 또한, Bulman은 설계와 시스템 라이브러리에 추가시 식별 가능한 새로운 추상 자료형(ADTs)에 대하여 연구하였다.

User Interface 객체는 많은 시스템 개발에서 널리 보급되어 있고 필수적이기 때문에 소프트웨어 설계 방법론의 일부분이어야 한다. 어떤 연구들은 사용자 인터페이스가 시스템의 한 구성요소라고 하지만, 인터페이스 객체를 식별하고 모델링하는 방법은 거의 제공하지 않는다. Application 객체는 앞에서 언급했듯이 특별한 응용에 대한 감독과 제어에 이용된다. 그리고, 응용 객체는 의미 클래스의 근본적인 관점인 인터페이스 객체와 Base/Utility 객체와도 구별되며, 비록 인터페이스 객체를 사용할 수 있다고 해도 인터페이스 객체의 한 종류는 아니다.

객체 지향 설계 프로세스에서 또 다른 활동은 클래스를 정제하는 것이다. 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. 클래스를 효율적으로 정제하기 위해 좀 더 추상화된 것을 찾아 심층적인 계층 구조를 생성하고, 메소드를 정제하거나 추가시키는 등 중요한 작업이 이루어진다. Bulman[6]은 다른 클래스들을 조사할 때 일반적인 특징과 행위가 발견되는 경우에 추상화와 더 나은 계층 구조를 구성할 수 있다고 지적하였다. 많은 연구자들에 의해서 객체 지향 설계 프로세스에 대한 연구가 이루어졌다. 그 중 Booch, Coad and Yourdon, Rumbaugh *et al.*, Bulman 등이 객체 지향 설계 프로세스에 대한 많은 부분을 설명하였다. 특히, Bulman은 Interface, application, Base/Utility 객체에 중점을 두고 설명하였다.

4.2 객체 지향 설계 표현(Representation) 방법

표현(Representation)은 크게 정적인 관점과 동적인 관점으로 구분할 수 있다. 먼저 정적 관점에 대하여 살펴보면, 모든 표현 모델들은 그 구조가 비슷하지만, 표기법(notation)에 있어 다양하게 이용된다. 본 연구에서는 표기법에 대한 세부적인 사항은 피하고, 몇 가지 중요한 사항에 대해서 고찰해 보기로 한다(〈표 4〉의 2절 참조).

첫째로, 객체 지향 정적 모델은 객체 지향 동적 모델보다 많은 종류의 모델이 있다. 동적 모델들 중에서 구성 요소 대부분은 객체 모델에서 행위(behavior)로 맵핑되거나 번역되는 상태 전이도로 구성된다.

동적 행위보다 정적 구조가 강조되는 한 가지 이유는 설계시 데이타 모델링으로부터 파생되는 문제가 객체 지향 설계에서 막대한 영향을 미칠 수 있기 때문이다. 데이타 모델링은 동적 행위나 제어에 중점을 두지 않고 구조에 중점을 두고 있는 반면에, 동적

모델링은 사건-처리(event-handling)에 중점을 두고 구성된다. 사건-처리란 사건(event)과 프로세스(process)의 타이밍 순서에 맞추어 순차적으로 제어하는 것이다. 동적 모델링에 대한 관심이 증가함에 따라, 객체 지향 설계에서 좀 더 많은 정적/동적 표현 방법이 증가하는 것을 알 수 있다. 예를 들면, Kappel의 객체 행위 모델(object behavior model)은 객체의 구조와 행위 모두를 모델화하는 다이어그램의 집합으로 구성하였다. 이 모델에서 많은 다이어그램들은 기존에 존재하는 동적 모델을 확장한 것이다.

둘째로, 표기법에 관한 중요한 관점은 관계(relationships)의 표현을 포함하는 것이다. 〈표 4〉에서 보는 바와 같이 대부분의 표현들은 일반화(generalization) 관계로 지원된다. Aggregation 관계도 지원되지만, 아직까지 널리 이용되고 있지 않다. 그리고, 몇 가지 다른 관계에 대해서 연구되었지만, 표현에 의해 지원되거나 많은 연구자들에게 인정을 받은 관계는 거의 없다. 일반화구조가 널리 인정받는 이유중의 하나는 일반화 구조가 객체 지향 프로그래밍 언어에서 직접 지원되기 때문이다.

대부분의 연구자들은 객체 지향 설계 표현(Representation)에 많은 관심을 가지고 연구하였다. 그중 Booch, Coad and Yourdon, Rumbaugh *et al.*, Bulman, Wasserman *et al.* 등이 객체 지향 설계 표현에 대한 많은 연구를 하였다. 특히, Wasserman은 객체 지향 설계 방법론의 표현에 대해서 집중적으로 연구하였다.

4.3 객체 지향 설계에서 복잡도 관리

몇몇 연구자들은 방대한 설계 복잡도를 관리하기 위해 개념적인 그룹 메카니즘을 제공하고 있다(〈표 4〉의 3절 참조). 이 메카니즘의 대부분은 구조적인 복잡도에 대하여 처리하고 있다. 예를 들면, 클래스를 좀 더 추상적인 'things'로 그룹화하는 방법으로 Coad and Yourdon's의 subject[7]와 Wirfs-Brock *et al.*의 subsystem[29]은 문제-분류(problem-partitioning) 메카니즘의 예제이다. subject는 어떤 루트 구조나 substructure에서 객체를 캡슐화하는 메카니즘이며, 루트로부터 구조를 상속받아 구성되기 때문에 복잡도가 감소된다. 그러나, Wirfs-brock *et al.*의 서브 시스템은 반드시 루트 구조에 기반을 두지 않는다는 점에서 subject와 구별된다. 서브 시스템은 상위 레벨 기능을 달성하기 위해 구성된 다른 서브 시스템이나 클래스로 이루어져 있다. 예를 들면 자동화된 은행 출납 기계 응용은 제정적인 서브 시스템과 사용자 인터페이스 서브 시스템으로 구성할 수 있다. 이것은 구조적 관계에 의해서 정의된 것이라기 보다는 기능적인 관계에 의해서 정의된 것이다.

서브 시스템은 복잡도를 줄이는데 있어서 유용한 메카니즘이지만, 서브 시스템의 식별에 대한 조작은 어려울 수도 있다. 이와 같은 메카니즘 연구자들은 서브 시스템을 식별하기 위한 많은 방법론을 제공하고 있지 않다.

Subject와 subsystem은 Coad and Yourdon's와 Wirfs-Brock *et al.*의 표현으로 지원된다. 그러나, 다른 연구자들은 개념적인 그룹 메카니즘을 제안하고 있으나, 이것을 표현하기 위한 툴을 제공하지는 않는다. Rumbaugh *et al.*[24]은 클래스와 관계를 그룹화하는 방법으로서 모듈을 제안하고 있으나, 모듈 레벨 관점에서의 표기법에 대해서는 제공하지 않았다.

4.4 객체 지향 설계의 장단점

현재의 객체 지향 설계 방법론에 관한 연구가 가지는 전반적인 문제는 Macro 레벨에서 Micro 레벨로의 전환시에 시스템의 다른 견해를 통합, 조정하는 포괄적인 방법론이 제시된 것이 없다는 것이다. 대부분의 객체 지향 접근 방법은 데이타 모델링 또는, 동적 모델링에 의해 단순한 기능으로 분해하는 것이다. 그러므로, 개발자들은 시스템의 구조와 동적 행위를 정의하고 표현해야 하며, 메소드에 대한 기능 분해를 수행해야 한다. 따라서, 기존의 프로세스와 표현을 결합하여 수정할 필요가 있다.

일반적으로 다른 패러다임을 결합하는 방법과 정도에 의해 결정되는, 객체 지향 설계에 접근하는 일반적인 세 가지 방법은 다음과 같다.

(1) 결합에 의한 접근 : 객체지향, 기능 지향, 그리고 동적 지향 기술을 단독으로 모델 구조, 기능성, 동적 행위로 이용하고 다른 모델을 통합하기 위한 방법론을 제공한다.
(2) 적응될 수 있는(adaptive) 접근 : 기존의 기술을 객체 지향에 이용하거나 객체 지향 기술을 포함하여 확장한다.
(3) 순수한 객체 지향 접근 : 객체 구조, 기능성, 동적 행위를 모델화하기 위해 새로운 기술을 이용한다.

결합에 의한 접근 방법은 전통적 기법을 이용하는 시스템의 관점과 매우 다르다. 전통적인 설계 기법은 구조적, 동적, 기능적 모델링을 위한 프로세스와 표현을 갖는다. 비록 표현 방법이 새롭다할지라도 구조적 표현은 데이타베이스 모델링과 특히 EROs에서 유도된다. 동적 모델은 상태 전이 다이아그램으로 구성되어 있으며 기능적 모델은 자료 흐름도와 유사하다. 이 세 가지 모델은 일반적으로 서로 독립적이며, 시스템의 서로 다른 관점에 대하여 기술한다. 그러나 이 결합에 의한 접근은 다른 관점을 객체 지향 설계 방법으로 통합할 때 내용이 변경될 수 있다. 구조적 설계는 일반적으로 객체, 애트리뷰트, 관계를 포함하지만, 행위(behavior)는 포함하지 않는다. Bailin[3], Gorman & Choobineh[10], Henderson-Sellers & Constantin[11] & Kappel은 기존의 프로세스와 표현을 객체 지향 측면과 혼합하여 적용할 수 있는 방법을 제공하였다. 예를 들면, 전통적인 자료 흐름 다이어그램과 다른 기능 지향 표현들은 시스템의 기능성보다는 객체의 기능성을 표현하는 새로운 방법으로 사용될 수 있다. Bailin의 방법은 엔터티 자료 흐름도를 이용한 것으로, 기능이 특별한 객체나 엔터티와 결합된 자료 흐름 다이어그램이다.

Gorman & Choobineh의 객체 지향 엔터티 관계 모델은 행위 모델링을 포함하기 위하여 엔터티 관계와 의미 데이타 모델을 확장한 것이다.

순수한 객체 지향 접근 방법을 연구하는 연구자들은 순수한 객체 지향 측면을 이용하여 소프트웨어 설계에 접근하였다. 그들은 문제 영역(problen domain), 프로토콜, 그리고 메세지 전송과 다른 객체와의 관계에 중점을 두고 모델링하였다. 이 기술은 결합에 의한 접근 방법처럼 번역 절차가 요구되지 않는다.

객체 지향 설계를 지원하기 위한 정형화된 방법론은 아직까지 완전하게 표준화되어 있지 않다. 따라서 객체 지향 설계 방법론에 대한 정확한 정의를 내리기는 어렵다. 〈표 5〉는 지금까지 기술한 객체 지향 설계에 대한 장단점을 간략하게 기술하였다.

〈표 5〉 객체 지향 설계에서의 장단점

장 점	단 점
◦ Identifying semantic classses	◦ Identifying interface, application and system classes
◦ Identifying attributes and behavior	◦ Determining when an attribute, relationship or behavior should be a class
◦ Placing methods	◦ Placing classes
◦ Identifying and representing generalization and aggregation structures	◦ Identifying and representing other kinds of relationships
	◦ Maintaining consistant and correct semantics for relationships
◦ Representing static views (structure)	◦ Representing dynamic views (Message passing, control,...)
	◦ Integrating static and dynamic models
	◦ Maintaining consistent levels of abstraction/granularity

V. 앞으로의 연구분야와 동향

많은 연구자들은 객체 지향 설계 시스템에 대한 몇 가지 관점을 모델화하기 위한 많은 노력을 하고 있다. 예를 들면, 구조적, 기능적 관점, 그리고 제어 관점은 모델화가 가능하며 또한, Coad & Yourdon의 subject와 Wirfs-Brock *et al.*의 subsystem과 같은 객체보다는 상위 단계의 추상화 구축이 가능하다. 그러나, 시스템의 설계와 표현을 위한 통합과 조정에 대한 단계와 관점의 명확한 정의는 매우 중요하다.

〈표 4〉의 3절에서 알 수 있듯이, 객체 지향 설계에서는 대규모 설계의 복잡도 관리를 위한 메카니즘을 갖고 있지 않다. 따라서, 이것을 해결하기 위한 방법으로 시스템의 다양한 "추상화 단계"를 정의해야 한다. 따라서, 객체 지향 설계 방법론에서 다양한 추상화 단계와 시스템의 관점 정의 및 통합을 지원하는 개발 방법론이 연구되어야 한다.

그리고, 또 하나의 연구 분야로써 객체 지향 설계에 대한 평가 모델의 개발이 있다. 현재의 객체 지향 설계 모델은 자연스런 평가보다는 일반적인 규칙에 의해 평가가 이루어지고 있다. 따라서, 객체 지향 설계 방법론에서 기술적인 측면과 의미적인 측면을 고려한 평가가 이루어져야 한다. 즉, 사용자, 분석자, 그리고 개발자가 가지고 있는 정신적 모델이 "설계에 얼마나 가깝게 반영되는가?"를 평가하기 위한 객체 지향 설계 평가 모델이 개발되어야 한다. 또한, 이외의 연구분야로서는 클래스, 행위, 관계에 대한 Typology와 객체 지향 설계 접근에 대한 Typology에 대해서도 연구되어야 한다.

VI. 결 언

객체 지향 설계 방법론은 개발기관, 대학, 그리고 연구소에서 많은 관심을 가지고 연구되고 있는 분야이다. 따라서, 본 연구에서는 객체 지향 설계 방법론에 관한 연구를 비교 평가하기 위한 프레임워크를 구성하여 현재까지 연구된 객체 지향 설계 방법론에 대

해서 살펴보았다.

개발된 많은 객체 지향 설계 방법론 중에서 Booch, Coad and Yourdon, Rumbaugh *et al.* 등은 객체 지향 설계 방법론의 구성 요소인 프로세스(Process), 표현(Representation), 복잡도 관리(Complexity MGR.)에 대한 부분을, 본 연구에서 구성한 프레임 워크를 통해서 살펴보았다. 그러나, 이와 같은 설계 방법론이 매우 우수하다고는 말할 수 없다.

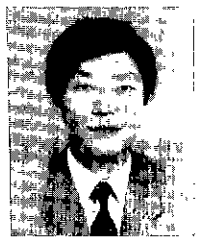
본 연구에서는 객체 지향 설계 방법론에서 프로세스, 표현, 복잡도 관리에 대한 모든 것을 완전하게 지원하는 설계 방법론이 제안되어 있지 않다. 따라서, 지금까지 연구되어진 설계 방법론을 기반으로 프로세스, 표현, 복잡도 관리 모두를 지원할 수 있는 객체 지향 설계 방법론을 개발하여, 객체 지향 방법론을 기반으로 하는 CASE 툴의 개발이 기대된다.

참 고 문 헌

1. Ackroyd, M and Daum, D. "Graphical notation for object-oriented design and programming," J. Object-Oriented Program 3(5) pp. 18~28, Jan. 1991.
2. Alabiso, B. "Transformation of data flow analysis models to object-oriented design," In proceedings of OOPSLA '88 Conference, pp. 335~353, San Diego, Calif., Sept. 1988.
3. Bailin, S. C. "An object-oriented requirements specification method," Commun. ACM 32, 5, pp. 608~623, May 1989.
4. Beck, K. and Cummingham, W. A. "Laboratory for teaching object oriented thinking," In proceedings of OOPSLA '89 Conference. pp. 1~6, New Orleans, La., Sept. 1989.
5. Booch, G. "Object-Oriented Design with Applications," Benjamine/Cunnings, 1991.
6. Bulman, D. "Refining candidate objects," Comput. Language, pp. 30~39, Jan. 1991.
7. Coad, P. and Youdon, E. "Object-Oriented Design," Prentice Hall, englewood Cliffs, N. J., 1991.
8. Colter, M. A. A. "Comparative examination of systems analysis techniques," MIS Q., pp. 51-66, Mar. 1984.
9. Cunninghamm, W. and Beck, K. A. "Diagram for object-oriented programsm," In proceedings of OOPSLA '86 Conference (Portland, Ore., pp. 39~43, Sept. 1986.
10. Gorman, K and Choobineh, J. "The object-oriented entity relationship model (OOERM)," J. Manage. Inf. Syst. 7(3), pp. 41~65.
11. Henderson-Sellers, B. and Constantine, L. L. "Object-oriented development and functional decomposition," J. Object-Oriented Prog. 33, pp. 11~17, Sept. 1991.
12. Henderson-Sellers, B. and Edwards, J. M. "The object-oriented systems life cycle," Commun. ACM 33, 9, pp. 142~169, Sept. 1990.
13. livari, J. "Object-oriented informations system analysis: A framework for object identification," IEEE Trans. on Saftware Eng., pp. 205~218, 1991.
14. Johnson, R. E. and Foote, B. "Designing reusable classes," J. Object-Oriented Program. pp. 22~35, June/July 1988.
15. Kappel, G. "Using an object-oriented diagram technique for the design of information systems," Dynamic Modelling of Information Systems. H. G. Sol and K. M. van Hee Eds. Elsevier Science Publishers, B. V., pp. 121~164, 1991.
16. Ladden, R. M. "A survey of issues to be considerded in the development of an object-oriented development methodology in Ada," Ada Letters 9, 2, pp. 78~88. Mar./Apr. 1989.
17. Lee, S. and Carver, D. L. "Object-Oriented analysis and specification: A knowledge base approach," J. Object-oriented Program., pp. 35~43, Jan. 1991.
18. Lieberherr, K., Holland, I., and Riel, "A. Object-oriented programming: An objective sense of style," In Proceedings of OOPSLA '88 Conference, San Diego, Calif., pp. 323~334, Sept. 1988.
19. Meyer, B. "Object-Oriented Software Construction," Prentice Hall, Englewood Cliffs, N. J., 1988.
20. Mrdalj, S. "Biobliography of object-oriented system development," ACM SIGSOFT Softw. Eng. Not. 15, 5, pp. 60~63, 1990.
21. Nierstrasz, O., Gibbs, S. and Tsichritzis, D. "Component-oriented software development," Commun. ACM 35, Sept. 1992.
22. Page-Jones, M., Constantine, L. L. and Weiss, S. "Modeling Object-Oriented systems: The uniform object notation," Comput. Language, pp. 70~87, Oct. 1990.
23. Pressman, R. S. "Software Engineering: A Pra-

ctitioner's Approach," Second ed. McGraw-Hill. 1987.
24. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "Object-oriented modeling and design," Prentice Hall, Englewood Cliffs, N. J., 1991.
25. Scharenberg, M. E. and Dunsmore, H. E. "Evolution of classes and objects during object-oriented design and programming," J. Object-Oriented Program. pp. 30~34, Jan. 1991.
26. Shlaer, S. and Mellor, S. J. "Object-Oriented Systems Analysis: Modeling the World in Data," Prentice Hall, 1988.
27. Wasserman, A. I., Pircher, P. A. and Muller, R. J. "The object oriented structured design notation for software design representation," IEEE Comput, pp. 50~62, Mar. 1990.
28. Wilson, D. "Class diagrams: A tool for design, documentation, and teaching," J. Object Oriented Program. pp. 38~44, Jan./Feb. 1990.
29. Wirfs-Brock, R. J., Wilkerson, B. and Wiener, L. "Designing Object Oriented Software," Prentice Hall, Englewood Cliffs, N. J., 1990.
30. Yourdon. E. and Constantine, L. L. "Structured Design: Fundamentals of a Discipline of Computer Program and Systems," Prentice Hall, Englewood Cliffs, N. J., 1979.
31. Wirfs-Brock, R. J. and Johnson, R. E. "Surveying Current Research in Object Oriented Design", Comm. ACM, Vol. 33, No. 9, pp. 104-124, Sept. 1990.
32. De. Champealus, D. and D. Fawre, "A Comparative Study of Object-Oriented Analysis Methods", J. Object-Oriented Programming, Vol. 5, No. 1, pp. 21~33, 1992.

양 해 술

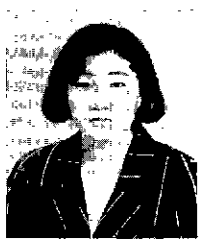


1975 홍익대학교 공과대학 졸업(학사)
1978 성균관대학교 정보처리학과 졸업(석사)
1991 日本 오사카대학 기초공학부 정보공학과 소프트웨어 공학 전공(공학박사)
1975 ~1979 육군 중앙경리단 전자계산실 근무
1984 ~1992 성균관대, 명지대 경영대학원 강사
1986 ~1987 日本 오사카대학 객원 연구원
1980 ~현재 강원대학교 지연과학대학 전지계산학과 교수
관심 분야 : 소프트웨어 공학(특히, S/W 품질보증과 평가, SA/SD, OOA/OOD/OOP, CASE), 전문가 시스템

조 영 식

1992 강원대학교 전지계산학과 졸업(이학사)
1992 ~현재 강원대학교 전지계산소 연구조교
1990 ~현재 강원대학교 전자계산학과 대학원 석사과정
관심 분야 : 소프트웨어 공학(특히, 객체지향 분석과 설계 방법론, 소프트웨이의 재이용과 CASE), 전문가 시스템(특히, 지식 추출 및 추론)

이 용 근



1988 강원대학교 전자계산학과 졸업(이학사)
1989 ~1992 강원대학교 전지계산학과 조교
1992 ~현재 강원대학교 전지계산소 연구조교
1992 ~현재 강원대학교 전지계산학과 대학원 석사과정
관심 분야 : 소프트웨어 공학(특히, 소프트웨어 품질보증과 평가, 객체 지향 분석과 설계 방법론, 객체 지향 프로그래밍)