객체 지향 설계를 위한 모듈 분해방법

허 계 범† 최 영 근††

요 약

객체 지향 설계 방법은 프로그램을 객체로 분해하고, 그들 사이의 관계를 설정하는 과정으로 기존의 시스템이 수행하는 기능 중심 방법과 달리 취급하는 객체를 중심으로 모듈을 분해하는 기법이다. 그러나 기존의 객체 지향 설계 방법은 모듈 설계 방법과 분해 기준이 모호하고 미흡하기 때문에 객체 지향 소프트웨어를 전체 시스템 단위로 구현해야 하고, 하나의 소프트웨어 컴포넌트를 이해하기 위해서는 전체적인 객체 설계 모델 및 응용 프로그램을 분석해야 하는 많은 문제점이 발생한다. 따라서 본 논문에서는 객체 지향 설계를 위한 모듈 분해 기준을 정형화 하고, 이를 이용한 객체 모듈 설계 절차를 제시하여, 규모가 크고 복잡한 시스템을 개발하는데 있어 실용성과 시스템 개발 후 유지보수를 원활하게 할 수 있도록 한다. 그리고 본 논문의 모듈 분해 기준을 실 시스템에 적용한 사례를 통하여 설계 모델과 구현 모델간의 일치성을 보이고자 한다.

A Module Decomposition Method for Object-Oriented Design

Guai Bum Heu† and Young Keun Choi††

ABSTRACT

Object-Oriented design method decomposes the program into object that establish the relationship between objects. It is the technique of object centered decomposition unlike function oriented legacy system. Since existing Object-Oriented design method doesn't fully explain of the design method of module and decomposition, and need to implement the Object-Oriented software as a total system unit. It has many problems that whole object design method and application programs should be analyzed for the understanding an Object-Oriented software component. Therefore, this study provides a procedure for Object-Oriented module design and criteria of decomposition to practically support the complex and large system development. Futhermore, it shows a consistency between design model and implementation through a case study applying these module decomposition criteria to the real system.

1. 서 론

설계 모델의 정형화된 분해 방법은 크고 복잡한 대형 시스템을 개발하는데 있어 필수적이라 할 수 있다. 즉, 큰 규모의 시스템을 몇 개의 작은 단위 모듈로 분해하는 것을 의미한다. 구조적 방법의 시스템 모듈 분해 기준은 시스템을 기능적 단위로 분해하고, 기능의 사용에 따라 계층적인 순서를 유지하며, 하위의 모듈로 분해함으로써 복잡성을 덜고 이해성을 증가시킨다는 것이다. 그러나 구조적 방법의 모듈 분해 방법은 몇 가지 문제점이 있다. 즉, 데이타 중심이 아닌 기능 중심의 설계 모델을 이용함으로써, 설계 정보와 원시코드의 불일치로 인한 시스템 구현시 막대한 비용과 시간 그리고 인력을 낭비하고, 개발 완료 후 모듈 재사용 메카니즘을 지원하지 못한다. 또한, 모듈 분해 기준에 있어서도 더 이상 함수들이 만들어지지 않으면 중단하고, 모듈간의 인터페이스가 많아 복잡성이 가중되면 중단된다 등은 그 기준이 모호하다[2, 4, 5, 18]. 객체 지향 설계 방법은 프로그램을 객체로 분해하고, 그들 사이의 관계를 설정하는 과정으로, 기존의 시스템이 수행하는 기능 중심 방법이 아닌 취급하

† 정 회 원 : 신흥전문대학 전산정보처리학과 강사
†† 정 회 원 : 광운대학교 전자계산학과 교수
논문접수 : 1994년 6월 21일, 심사완료 : 1995년 4월 17일

는 객체를 중심으로 모듈을 분해하는 방법이며, 추상화, 정보은닉, 모듈화를 기본으로 소프트웨어를 생성하는 것이 특징이다[3, 7, 8, 11, 18]. 따라서 객체 지향 설계 방법은 문제 영역에 존재하는 객체들이 해결 영역인 설계 명세서나 프로그램상에 그대로 나타나고, 각각의 객체들이 철저한 정보은닉 개념하에서 구현되어지고 있기 때문에 어떠한 변화가 발생하여도 변화의 영향을 최대한 줄일 수 있는 이점을 제공해 주고 있다[1, 3, 7, 9, 10, 11, 12, 13, 16, 18].

그러나 기존의 객체 지향 설계 방법들 역시 모듈 설계 방법과 분해 기준이 미흡하여 객체 지향 소프트웨어 구현시 전체 시스템을 구현 단위로 하고 있어 하나의 소프트웨어 컴포넌트를 이해하기 위해서는 전체적인 객체 설계 모델 및 응용 프로그램을 분석해야 하는 복잡함과 시스템 개발의 병행성을 가질 수 없는 많은 문제점이 발생한다.

따라서 본 논문에서는 규모가 크고 복잡한 시스템을 개발하는데 있어 실용성과 시스템 개발 완료 후 유지보수를 원활하게 할 수 있도록 하였다.

첫째, 상향식(bottom-up)방법으로 분석(OOA)한 객체 모델을 구현 단계로 자연스럽게 연결될 수 있도록 하며, 시스템의 다면성을 가시화할 수 있는 객체 모듈 설계 절차를 제시한다.

둘째, 시스템을 이해하기 쉬운 모듈로 분해하여 복잡성을 덜고, 이해성을 증가시킬 수 있는 객체 지향 모듈 분해 방법의 정형화된 모듈 분해 기준을 제시하며,

셋째, 위에서 제시한 두가지 방법을 실 시스템에 적용하여 객체 모델 모듈과 구현 모델간의 가시적인 원시코드 일치성을 보여줌으로서 본 논문에서 제시한 이론의 실용성을 증명한다.

또한, 본 논문에서 제시하는 모듈 분해 기준은 OMT기술을 기본으로 시스템 개발자가 객체 지향적인 개념을 이해하고 있다면, 상위 모듈에서 프로그램 구현 단위 모듈까지 시스템의 분해 절차와 기준을 단계적으로 명시하여 구현 모델에 명확하게 사상시킬 수 있는 실무 활용에 효율적인 방법이며, OMT기술의 분석 단계 객체 모델링 결과를 설계 단계에서 상속성을 이용하여 객체 모델을 재정의만 하는, 비절차적이고 기준점이 모호한 분해 기준을 해결하였다.

그리고 객체 모델 모듈 설계를 우선으로, 동적 모델 모듈, 기능 모델 모듈을 설계하고 구현 단계에 앞서 이들을 결합하여, 이러한 결합 모듈을 구현 단계에 제공함으로서 효과적인 시스템 가시화에 의한 완전한 시스템 구현을 기대할 수 있다.

본 논문의 구성은 2장에서 관련 방법들을 고찰하고, 3장에서는 개선된 모듈 설계 방법과 분해 기준을 제시하며, 4장에서는 이를 적용한 실 시스템 사례와 기존의 방법들을 비교 분석하며, 5장의 결론 및 향후 연구 과제에 대하여 논하고자 한다.

2. 관련 연구

기존 소프트웨어 개발 방법들에 대한 모듈 분해 방법을 이해하기 위해서는 이들의 설계 단계를 고찰해 볼 필요가 있다. 이에 따라 본 장에서는 기능적 분해 방법, 자료 구조 설계 방법, 구조적 설계 방법 그리고 객체 지향 설계 방법 등을 모듈 분해 중심으로 고찰하고, 이 방법들의 문제점과 모듈 분해 방법의 개선점을 모색한다.

2.1 구조적 설계 방법[2, 4, 5, 10, 18]

구조적 설계방법은 기본적으로 추상화(abstraction), 분할과 통합(divide-and-conquer), 계층적인 순서 등을 바탕으로 하고 있다. 즉, 크고 복잡한 프로그램을 여러개의 작은 서브 프로그램으로 나누고, 각 서브 프로그램은 다시 더 작은 여러개의 서브 프로그램으로 나누어 개발하는 방법이다.

그 대표적인 설계 방법으로는 하향식 구조적 설계 방법, Jackson 설계 방법, Warnier-Orr 방법, 그리고 Yourdon의 구조적 방법 등이 있으며, 이들 모두는 기능적 분해 과정을 설계에 응용하고 있다.

하향식 설계 방법은 가장 비정형적 설계 방법으로, 전체 시스템이 어떤 일을 수행 하는가를 먼저 정의한 후 점차 기능을 세분화시켜, 전체

기능을 부분 기능(sub-function)의 집합으로 분리한다. 그리고 이러한 일련의 분해 과정을 각 부분 기능에 다시 적용하여, 프로그램 코드로 변환이 가능할 때까지 분해를 계속한다.

Yourdon의 구조적 설계 방법은 하향식 설계 기법을 개량시킨 것으로, 설계 과정을 좀 더 조직화하고, 설계 결과의 타당성을 측정할 수 있는 지침을 가지고 있으며, 설계 결과로 프로시저 요소와 그들의 계층 구조 및 그들을 연결하는 데이타 구조도를 산출한다.

Jackson의 설계 방법 또한 하향식 설계 방법을 개량한 방법이나, 데이타 중심 설계 방법를 이용하여 데이타 구조에서 프로그램의 구조를 산출하는 방법으로 처리 중심 설계 방법을 이용한 하향식 방법이나 Yourdon의 방법과는 다른 면이 있다.

Warnier-Orr 설계 방법은 데이타 중심 설계 방법으로, 출력 데이타에서 입력 데이타와 프로그램 구조를 생성하는 기법이며, 입출력 데이타 구조를 모두 묶어 프로그램 구조를 생성하는 Jackson 방법과는 차이가 있다.

2.2 객체 지향 설계 방법

객체 지향 설계 방법은 프로그램을 객체로 분해하고, 그들 사이의 관계를 설정하는 과정으로 본 논문에서는 Shlaer/Meller, Coad/Yourdon, Booch, Rumbaugh 등의 방법에 관해 고찰하고자 한다[1, 3, 6, 7, 9, 10, 12, 13, 14, 17, 18].

2.2.1 Shlaer/Mellor의 OOA/OOD

Shlaer/Mellor는 하나의 시스템을 구축하는데 있어, 시스템을 6가지의 계층으로 표현한다. 즉, 시스템, 영역, 서브 시스템, 객체, 상태, 프로세스 등의 최상위 시스템에서부터 최하위 프로세스까지 하향식 방법으로 표현하는 것이 특징이다. 또한, 이들을 명세화하는 방법으로는 정보 모델(information model), 상태 모델(state model), 기능 모델(functional model) 등으로 모델링 한다.

이 방법은 대형 시스템을 여러개의 영역(domain)으로 분해하여 개발하는 방법으로, 각 영역을 좀 더 구체적으로 기술하면, 분해된 시스템의 역할에 따라 사용자로부터 시스템에 부여한 당면 과제를 나타내는 응용 시스템 영역(application system domain), 응용 시스템 영역을 지원하기 위해서 유틸리티 기능을 제공하는 서비스 영역(service domain), 자료를 관리하고 시스템을 통제하는 구조 영역(architecture domain), 프로그래밍 언어, 네트워크, 운영체제 그리고 클래스 라이브러리(class library)를 포함하는 구현 영역(implementation domains)으로 분류된다. 또한, 이 영역들은 하위 서브 시스템으로 나뉘어져 영역 차트(domain chart)와 프로젝트 매트릭스(project matrix)로 표현되게 된다.

2.2.2 Coad/Yourdon의 OOA/OOD

Coad/Yourdon의 객체 지향 분석 및 설계 방법은 객체 식별, 구조 식별, 서브젝트 정의, 속성과 연관 관계 정의, 그리고 메시지 접속 관계 정의의 5단계로 구성되어 있다.

구조적 식별 단계는 추출된 클래스를 일반화-세분화에 의한 일반-특수 구조(general-specialzation structure)와 전체-부품화에 의한 전체-부분 구조(whole-part structure)로 분류하고, 이를 이용하여 문제의 복잡성을 해결하기 위한 단계이다.

일반-특수 구조는 객체들의 공통된 속성이나 오퍼레이션을 상위 계층의 객체로 표현하고, 하위 계층의 객체는 상위 계층의 객체 속성이나 오퍼레이션을 그대로 계승함과 동시에 자신의 고유 속성이나 오퍼레이션을 지니도록 표현하는 단위이다.

또한, 전체-부품 구조는 여러개의 객체들로 구성된 전체와 전체의 구성 요소인 부품들로 구조화하는 것으로 서브젝트는 시스템 분석자가 한번에 취급하기 적당한 분량의 객체들로 분해하는 단위이다.

이와 같이 객체들이 서브젝트별로 분해되면 서브젝트 사이에 메시지를 표현하여 하나의 시스템을 형성하고, 이들의 명세화는 하향식 방법에 따라 주체 계층, 클래스 및 객체 계층, 구조 계층, 속성 계층 그리고 서비스 계층으로 나뉘어 작성된다.

2.2.3 Booch의 OOD

Booch의 객체 지향 설계 방법은 클래스와 객체의 식별, 이들에 대한 의미와 관계 식별, 그리고 구현의 4단계로 구성되어 있다. 시스템은 논리적 관점의 정적 모델(static model)과 물리적 관점의 동적 모델(dynamic model)로 크게 나뉘어 설계되어지며, 논리적 관점은 시스템의 구현과 밀접한 관계가 없는 시스템의 주요 추상체 식별과 의미 기술을 위해 사용되어지고, 물리적 관점은 구현을 위한 소프트웨어와 하드웨어 구성품의 기술에 이용되는 모듈 구조나 프로세스 구조 등을 의미한다.

그리고 문서화 단계에 있어서도 정적 모델과 동적 모델로 구분하여 다이어그램으로 표현한다. 정적인 모델은 논리적 모델의 클래스 존재 여부 및 그들과의 관계를 나타내는 클래스 다이어그램과 객체의 상호 작용에 의해서 일어나는 구조를 나타내는 객체 다이어그램이 있고, 물리적 모델은 각 클래스의 모듈이 선언되어 있는 것들을 나타내는 모듈 다이어그램이 있다. 또한, 프로세스의 할당 여부 및 프로세스가 처리할 다중 프로세스의 스케줄 등을 나타내는 프로세스 다이어그램이 있다.

동적 모델은 사건이 동적으로 일어날 경우 객체의 생성, 삭제, 메시지의 전달 등을 나타내는 상태 전이 다이어그램과 타이밍 다이어그램 등이 있다.

2.2.4 Rumbaugh의 OMT

Rumbaugh의 OMT는 모든 소프트웨어 구성요소들을 도식적인 표기법(graphical notation)을 이용하여 객체를 모델링하는 방법으로 시스템의 분석, 설계, 구현 단계 전 과정에 추상화, 캡슐화, 모듈화, 계층화 등, 일관된 객체 지향 개념을 다음과 같이 적용한다.

(1) 추상화와 캡슐화 개념은 클래스명, 속성, 오퍼레이션 등이 함께 표현된 객체 및 클래스를 추출할 때 적용한다.
(2) 모듈화와 계층화는 추출된 클래스들의 시스템 내부 구조를 표현할 때 적용된다.
(3) 계층화 개념은 분석시에는 연관화, 집단화, 일반화 등에 적용되고, 설계시에는 시스템을 수평 계층으로 분해할 때 적용된다.
(4) 모듈화 개념은 시스템의 수직 분할에 적용되며, 수직 분할은 시스템을 몇개의 독립된 시스템들로 나누는 것으로, 하위 계층은 상위 계층에게 서비스를 제공하고, 상위 계층은 하위 계층의 서비스를 이용한다.

그리고 Rumbaugh의 OMT는 객체 모델링, 동적 모델링, 기능 모델링 등의 3가지 모델링을 적용하여 분석 및 설계 단계를 객체 모델, 동적 모델, 기능 모델 등으로 가시화한다.

2.3 기존 방법의 문제점 및 개선 방향

이상과 같이 전통적 방법과 객체 지향 방법의 설계 과정에 대해 시스템 분해 방법을 중심으로 알아보았다. 전통적 설계 방법은 시스템을 기능적 단위로 분해하여 기능의 사용에 따라 계층적 순서를 유지하며, 하위의 모듈로 분해함으로써 복잡성을 줄이고, 이해성을 증가시킨다. 그러나 이러한 방법은 데이타 중심이 아닌 기능 중심의 설계 모델을 이용하므로, 객체 지향 소프트웨어 개발 방법과는 근본적인 차이를 보인다.

객체 지향 설계 방법은 시스템이 수행하는 기능 중심이 아닌, 취급하는 데이타를 중심으로 모듈을 분해하는 것이다. 그러나 기존의 객체 지향 설계 방법들은 정형화된 모듈 설계 방법과 분해 기준이 미흡하고 그 기준도 모호하다. 또한, 이 방법들은 공통적으로 일반화(generalization), 세분화(specialization), 집단화(aggregation) 개념을 적용하고 있다. 즉, Shlaer/Mellor 방법은 시스템을 여러개의 영역(domain)으로 분할하여 관리하고, Coad/Yourdon 방법은 서브젝트(subject)별로 시스템 분석자가 한번에 취급하기 적당한 분량의 객체들로 분해한다. 또한, Booch 방법은 물리적 모델이 각 클래스들의 모듈로 선언되어 있는 것들을 모듈 단위 프로그램으로 표기하고, Rumbaugh 방법은 모듈화를 시스템의 수직 분할이라는 개념에 사상시켜, 몇 개의 독립된 시스템으로 나누는 정도이다. 따라서 기존의 방법들은 대부분 모듈 분해에 대한 구체적인 언급

과 명확한 지침이 없이 모호하고 미흡한 모듈 분해 기준을 가지고 있어, 이를 이용한 객체 지향 소프트웨어 개발이 이루어질 경우 유지보수 및 재사용에 많은 문제를 발생시킬 수 있다[3, 17, 18].

따라서 객체 지향 설계 방법을 지원하는 정형화된 분해 기준의 마련이 시급하게 요구되며, 이를 바탕으로 큰 규모의 시스템 개발에 효율적으로 활용할 수 있도록 하여야 한다.

3. 객체 지향 설계를 위한 모듈 분해 방법

본 장에서는 객체 지향 설계를 위한 정형화된 모듈 분해 기준을 제시하고, 이를 토대로한 객체 지향 설계 방법을 제안한다.

3.1 객체 지향 모듈 정의

객체 지향 소프트웨어 개발 방법에 있어 설계 단계에 대한 절차와 명세화의 모호함과 미흡은 설계의 완전성, 유지보수성 등 많은 문제점을 안고 있다.

특히, 객체 지향 설계 단계가 단순하게 하나의 시스템을 잘 정의된 모듈 단위로 분해하는 과정이라고 한다면, 정형화된 모듈 분해 기준과 그 기준에 따른 모듈 정의는 객체 지향 설계 단계의 핵심적인 중요 사항이라 할 수 있다.

구조화 방법에서 모듈이란 하나의 기능을 수행하는 프로그램의 단위로 정의한다. 그러나 이러한 모듈의 정의도 소프트웨어 개발 방법론이 다르면 그에 따른 모듈 정의의 변화가 필요하다.

따라서 본 논문에서는 객체 지향 소프트웨어 개발 방법 차원의 모듈을 하나의 클래스, 또는 객체로 정의하며, 정의된 모듈은 다음과 같은 특성을 가져야 한다[4, 5, 8].

(1) 무엇(What)에 관한 것으로, 대상에 대한 수행 기능이 뚜렷해야 한다.
(2) 어떻게(How)에 관한 것으로, 수행 방법이 명확하여야 한다.
(3) 모듈간의 인터페이스 내용이 뚜렷해야 한다.
(4) 모듈 분해 과정별 크기에 관한 것이 명확해야 한다.

이러한 모듈의 기본 특성을 바탕으로 하나의 시스템은 하향식 방법에 따라 단위 모듈을 얻기 위해 단계적인 모듈 분해가 이루어져야 하며, 분해된 단위 모듈은 다음과 같은 성질을 가져야 한다[1, 3, 8, 16].

(1) 모듈의 구조는 완전히 이해될 수 있도록 단순해야 한다.
(2) 다른 모듈과 관계없이 모듈의 구현을 변화시킬 수 있어야 한다.
(3) 모듈에 대한 변화가 다른 모듈에 영향을 미치지 않아야 한다.
(4) 모듈의 인터페이스를 변화시키지 않고도 모듈을 변화시킬 수 있어야 한다.
(5) 주요한 소프트웨어 변화가 개별적인 모듈에 대해 독립적인 변화의 집합으로 이루어질 수 있어야 한다.
(6) 모듈의 내부 설계를 살펴보지 않고도 관련있는 모듈을 쉽게 식별할 수 있어야 한다.

3.2 모듈 분해 기준 및 분해 방법

위와 같이 단위 모듈이 정의되기 위해서는 모듈 분해에 대한 정형화된 기준이 우선적으로 마련되어야 할 것이다.

따라서 본 논문에서는 분석, 설계, 구현 단계를 세분화하고, 분석에 이은 설계를 자연스럽게 연결함으로써 비교적 소프트웨어 개발 전 단계를 지원하고 있다고 볼 수 있는 Rumbaugh의 OMT 방법 중에서 분석 단계에서 정의된 객체 모델, 동적 모델, 기능 모델을 바탕으로 객체 지향 설계 단계를 객체 설계, 객체 모델 모듈 설계, 프로그램 설계 과정으로 세분화하고 각 과정별 다음과 같은 단계적인 절차에 따라 시스템을 분해하여 단위 모듈을 생성한다.

(1) 객체 설계 과정은 객체 모델 설계, 동적 모델 설계, 기능 모델 설계로 구분하여 객체들의 상세 사항을 표현한다.
(2) 객체 모델 모듈 설계 과정은 객체 설계 과정의 세 모델을 가장 상위 클래스(top-class)인 추상화 클래스를 기준으로 객체

모듈 설계 과정에서 하향식 방법으로 분해하여, 객체 모델 모듈, 동적 모델 모듈, 기능 모델 모듈로 세분화시킨다. 여기에서 추상화 클래스란,객체를 표현하는 방법으로 인간의 기억, 추론 등 모든 사고 행위의 기본적인 메카니즘을 모델링한 것으로 Aggregation-partition과 generalization-specialization 두가지 방법에 의해서 객체를 표현한다[1, 3, 7, 8, 9, 10, 11, 16]. Aggregation-Partition 방법은 하나의 사물을 표현할 때 그것이 가지는 부분들의 설명을 모아 총체적으로 사물을 표현하는 방법으로 종합적인 객체와 부분적인 객체 사이에 "Has-part-of"의 관계가 성립하고, Generalization-specialization은 하나의 사물을 표현할 때 그 사물을 포함하는 보다 넓은 의미의 사물들이 가지고 있는 개념들을 이용하여 하나의 사물을 표현하는 방법으로 좁은 의미의 객체와 넓은 의미의 객체 사이에 "Is-a"관계가 성립한다.

(3) 추상화 클래스를 구성(composite of or Is-part-of)하고 있는 단위 클래스까지 분해하여 이들의 관계를 정의하고, 이렇게 정의된 클래스들을 본 논문에서는 객체 모듈이라 정의한다. 여기에서 단위 클래스는 외부, 인터페이스, 추상화 클래스를 제외한 내부 객체 또는 클래스(internal object or class)를 말하며, 이러한 단위 클래스를 바탕으로 동적 모델 모듈 설계, 기능 모델 모듈 설계가 이루어 질 수 있게 한다.

(4) 프로그램 설계 과정은 객체 모델 모듈 설계 단계에서 정의된 모듈(단위 클래스)들의 이벤트(event) 또는 처리(process)를 기준으로, 실제 개발할 프로그램 단위를 정의하는 과정이다. 여기서 이벤트란 내부 이벤트(internal event)로서 프로그램 실행에 직접적으로 관계되는 생성(create), 정정(update), 삭제(delete), 조회(inquiry), 출력(output) 등을 의미한다. 따라서 모듈 단위가 클래스를 기준으로한 클래스 내부의 메소드(method)라고 정의한다면 프로그램 단위는 클래스 내부의 행위(behavior)를 표현한 것이라 할 수 있다[8].

(5) 구현 단계에서는 (4)에서 추출한 이벤트 또는 프로세스 단위를 기준으로 객체 모델 모듈, 동적 모델 모듈, 기능 모델 모듈을 결합하여 구현한다. 그러나 실제 구현 단계에서는, 객체 지향 설계서와 다른 설계 정보가 필요하게 될 수 있다. 이와 같은 이유는 각종 사용자 인터페이스 항목들을 업무와 독립되게 설계, 구현하는 경우가 종종 있기 때문이다.

이와 같이 본 논문에서 제시하는 모듈 분해 기준에 따라 설계 단계 동안의 모듈 분해 과정을 (그림 1)에 가시적으로 표현하였다. (그림 1)에서 하나의 시스템이 n개의 추상화 클래스로 분해되고, 하나의 추상화 클래스는 다시 n개의 추상화 클래스로 분해되어지고 있으며, 최종 단위 모듈은 클래스(프로그램 설계 차원은 이벤트)에서 결정되어짐을 알 수 있다.

그러나 이와 같은 모듈 분해 과정 동안 객체 모델에서 파생된 추상화 클래들(A1, …, An)의 총 개수 $\alpha 1$과 동적 모델에서 파생된 추상화 클래스들(B1, …, Bn)의 총 개수 $\beta 1$, 그리고 기능 모델에서 파생된 추상화 클래스들(C1, …, Cn)의 총 개수 $\gamma 1$과는 같은 값을 가져야 한다. 또한, 하나의 추상화 클래스가 또 다른 추상화 클래스로 분해될 수 있다면, 추상화 클래스에서 파생된 세 모델에 대한 또 다른 추상화 클래스들의 총 개수 $\alpha 2$, $\beta 2$, $\gamma 2$의 값 역시 같아야 하고, 이와 같

은 일련의 모듈 분해를 반복하여 최종적으로 얻어진 단위 클래스들의 총 개수를 $\alpha 3$, $\beta 3$, $\gamma 3$라 한다면 이들의 값 또한 같아야 한다.

그러므로 이들을 다시 정리하면 다음과 같은 관계가 성립할 수 있다.

$$\alpha 1 = \beta 1 = \gamma 1$$
$$\alpha 2 = \beta 2 = \gamma 2$$
$$\alpha 3 = \beta 3 = \gamma 3$$
$$\vdots$$
$$\alpha n = \beta n = \gamma n$$

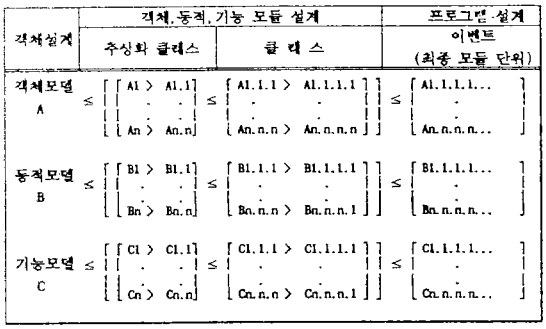
이와 같은 관계를 이용하여, 만약 세가지 값이

구 분	객체설계 단계	객체 모듈 설계 단계			프로그램 설계 단계		구현 단계
		추상화 클래스 단위 분할		클래스 단위 분할	이벤트단위 분할(프로그램단위)		코 딩
시스템	객체모델 A	추상화클래스A1 · · 추상화클래스An α1	추상화클래스A1.1 · · 추상화클래스A1.n α2	클래스 A1.1.1 클래스 A1.1.2 ..... 클래스 A1.1.n 클래스 A1.n.1 클래스 A1.n.2 ..... 클래스 A1.n.n α3	이벤트 A1.1.1.1 이벤트 A1.1.1.2 ..... 이벤트 A1.1.1.n 이벤트 A1.n.1.1 이벤트 A1.n.1.2 ..... 이벤트 A1.n.n.n	이벤트A1.n.1.1.1 이벤트A1.n.1.1.2 이벤트A1.n.1.1.3	
	동적모델 B	추상화클래스B1 · · 추상화클래스Bn β1	추상화클래스B1.1 · · 추상화클래스B1.n β2	클래스 B1.1.1 클래스 B1.1.2 ..... 클래스 B1.1.n 클래스 B1.n.1 클래스 B1.n.2 ..... 클래스 B1.n.n β3	이벤트 B1.1.1.1 이벤트 B1.1.1.2 ..... 이벤트 B1.1.1.n 이벤트 B1.n.1.1 이벤트 B1.n.1.2 ..... 이벤트 B1.n.n.n	이벤트B1.n.1.1.1 이벤트B1.n.1.1.2 이벤트B1.n.1.1.3	
	기능모델 C	추상화클래스C1 · · 추상화클래스Cn γ1	추상화클래스C1.1 · · 추상화클래스C1.n γ2	클래스 C1.1.1 클래스 C1.1.2 ..... 클래스 C1.1.n 클래스 C1.n.1 클래스 C1.n.2 ..... 클래스 C1.n.n γ3	이벤트 C1.1.1.1 이벤트 C1.1.1.2 ..... 이벤트 C1.1.1.n 이벤트 C1.n.1.1 이벤트 C1.n.1.2 ..... 이벤트 C1.n.n.n	이벤트C1.n.1.1.1 이벤트C1.n.1.1.2 이벤트C1.n.1.1.3	

(그림 1) 객체 지향 설계 과정별 모듈 분해
(Fig. 1) Decomposition of module for object-oriented design process

일치되지 않는 과정까지 모듈을 분해했다면 앞단계에서 모듈을 재정의하여야 한다. 즉, 하나의 객체 또는 클래스에 대한 데이타(data), 제어(control), 기능(function) 측면을 각각 설계/결합할 수 있어 시스템의 다면성을 가시화할 수 있어야 한다는 객체 지향 소프트웨어 공학적 차원을 반영한 것이다. 또한, 하나의 시스템을 모듈 단위로 분해하는 동안 단계적으로 모듈의 크기는 반드시 작아지고, 모듈의 개수는 전단계보다 많거나 같게됨을 (그림 2)와 같이 나타낼 수 있다.

즉, (그림 2)에서 추상화 클래스(A1)로부터 파생된 또 다른 추상화 클래스(A1.1)는 객체 모델로부터 파생된 추상화 클래스(A1) 보다 모듈의 크기가 반드시 작고, 자신으로 부터 파생된 클래스(A1.1.1) 보다 모듈의 크기가 반드시 크게 된다. 또한, 추상화 클래스 분해 과정에서 모듈의 총 개수는 클래스 단위에서 모듈의 총 개수보다 적거나 같게되고, 객체, 동적, 기능 모듈 설



(그림 2) 모듈 분해 과정별 모듈 크기 및 모듈 개수 관계
(Fig. 2) Relationship of module size and account for module decomposition process

계 단계의 모듈의 총 개수는 프로그램 설계 단계의 모듈의 총 개수보다 적거나 같게됨을 의미한다.

이와 같이 본 논문에서 제시한 모듈의 분해 기준은 다음과 같은 특징을 갖는다.

(1) 구조화 방법 등 기존의 소프트웨어 개발 방법들의 프로그램 단위 정의에 따른 기능 중심적인 모듈이 아닌, 객체를 중심으로한 모듈을 생성한다.

(2) 기존 객체 지향 소프트웨어 방법론들의 미흡하고 모호한 분해 기준를 정형화함으로서 개발자에게 명확한 지침을 제공할 수 있는 효과를 얻을 수 있다.

(3) 시스템을 이해하기 쉽도록 단계적으로 분해하여, 기존의 재사용할 모듈을 염두해 두고 분해하며, 경우에 따라서는 재사용 가능한 하위 모듈에서 결합이 될 수 있도록 하고 있다. 즉, 상위에서는 하향식 방법으로 시스템을 분해할 수 있도록 하고, 하위에서는 모듈들을 결합하여 시스템의 부분 구조를 만들 수 있도록 한다. 그렇게 함으로서 전체적인 객체 모델 구조를 유지하면서 기존의 모듈들을 재사용할 수 있게 한다.

(4) 이와 같은 방법론을 취하면 원시모듈 뿐만 아니라 객체 모델상의 중.상위의 수준에서 나타나는 추상화 클래스들도 재사용할 수 있다.

(5) 기존의 모듈을 염두해 두기 때문에 필요한 경우에 설계 자체를 기존의 설계 체계로 변경하여 설계의 표준화를 추구할 수 있는 장점이 있다.

이상과 같이 본 논문의 모듈 분해 이론은 객체 지향 소프트웨어 공학의 개념을 충족하고 있으며, 모듈의 특성을 충분히 반영하고 있어, 객체 지향적인 정형화된 모듈 분해 기준이 될 수 있다.

3.3 설계 검증

본 논문에서 제시한 모듈 분해 이론을 적용한 객체 지향 설계 후 다음 〈표 1〉과 같은 점검 항목에 따라 설계 검증이 이루어질 수 있도록 설계 검증 절차를 제시한다.

여기에서 제시하는 절차는 Pressman의 설계 평가 기준인[14]

1) 클래스간의 계층 구조 표현성
2) 클래스간의 기능 독립성(응집도)
3) 반복적인 수정작업 가능성
4) 클래스 단위의 상호 연결 강도(결합도)
5) 모듈화
6) 재사용성

등을 바탕으로 한다.

〈표 1〉 모듈 설계 검증 절차
〈Table 1〉 Procedure of module design veridation

점 검 항 목
• 각 모듈은 클래스 단위로 분해되었는가? • 모듈은 하향식으로 분해되었는가? • 모듈은 분해 이론에 적합한가? • 각 모듈은 독립적인가? • 모듈간의 접속이 명확한가? • 각 모듈의 응집도는 높은가? • 모듈사이의 결합도는 낮은가? • 각 모듈은 허용된 복잡도 범위안에서 설계되었는가? • 모든 모듈을 하나 이상의 요구사항으로부터 추적할 수 있는가? • 각 모듈은 재사용이 가능한가? • 각 모듈은 원시코드 일치성이 높은가?

4. 적용 사례

적용 사례로 E-Mail 시스템의 전체적인 객체 모델을 가시화하고, 문서관리 업무중 문서 발송

관리 모듈의 생성 과정을 통하여 본 논문의 모듈 분해 기준의 타당성과 실용성을 보이고, 설계 단계의 객체 모델 모듈 다이어그램과 구현 단계의 객체 지향 프로그래밍 언어 구문과의 원시코드 일치성을 보임으로서 효율적인 시스템 개발이 될 수 있음을 입증하고자 한다.

객체 지향 설계 단계를 시스템 설계, 객체 설계, 객체 모듈 설계, 프로그램 설계 과정으로 나누고, 본 논문에서는 시스템 설계 과정은 생략하고, 객체 설계 단계 중에서 객체 모델 설계와 객체 모듈 설계 과정부터 적용한다.

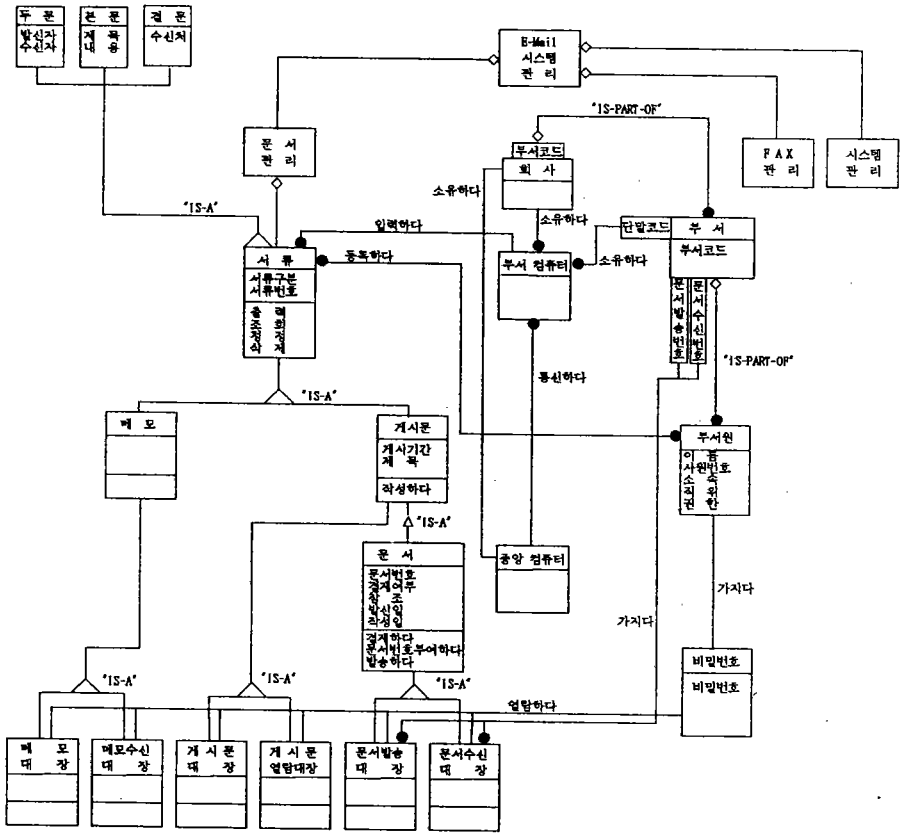
4.1 객체 모델 설계

객체 모델 설계 단계에서는 Rumbaugh의 OMT 방법 중에서 객체 모델을 바탕으로 설계 모델을 작성하는 단계이다. 여기에서의 작업 활동을 살펴보면, 객체의 분류, 변경된 클래스의 추가 및 삭제, 추상화 클래스 설정, 클래스간의 인터페이스 정의와 클래스 속성에 대한 후보 식별자 정의 등을 포함한다. (그림 3)은 OMT 방법의 객체 모델에서 최종 결과인 상속성을 이용한 다이어그램이며, (그림 3)의 다이어그램상에 앞에서 서술한 개선된 객체 모델 설계의 작업 활동들을 범례와 함께 나타낸 것이 (그림 4)이다.

4.2 객체 모델 모듈 설계

객체 모델 모듈 설계에서는 상위 단계인 객체 설계 단계에서 추출한 추상화 클래스를 기준으로 모듈을 분해하며, 분해된 모듈은 다시 클래스별로 나뉘어진다.

(그림 5)의 객체 모델에서 E-Mail 시스템은 문서 관리, FAX, 시스템 관리로 상위 모듈이 결



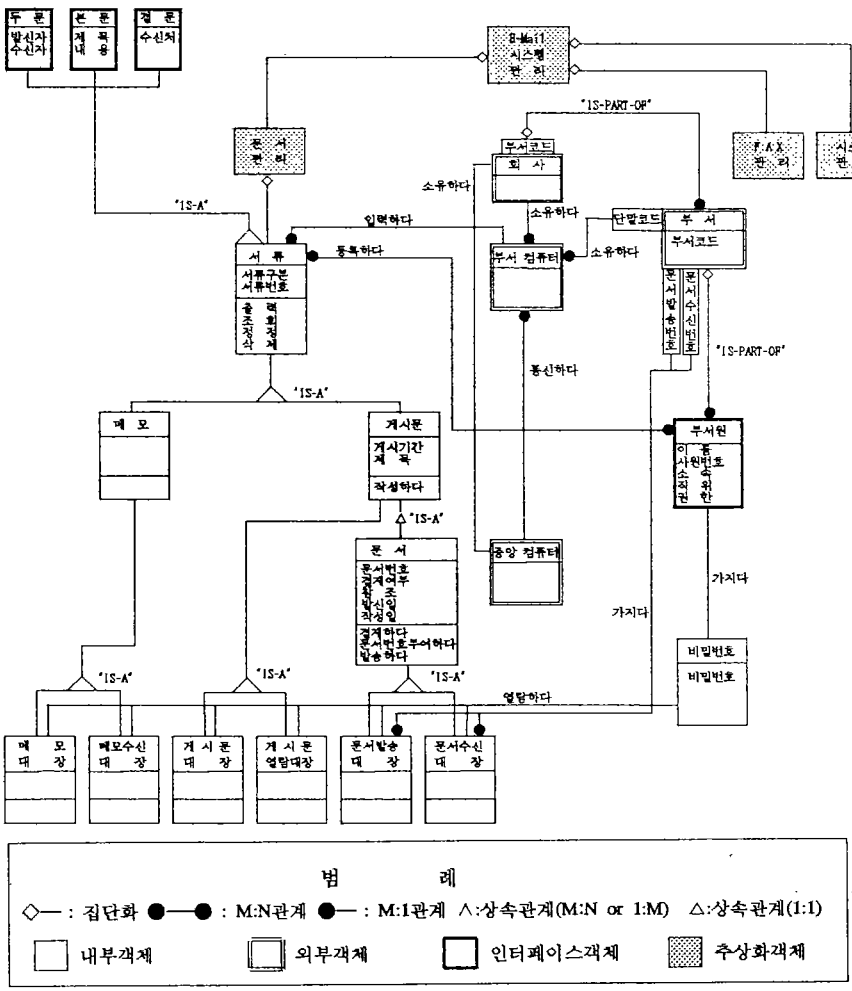
(그림 3) E-Mail 시스템 객체 상속 다이어그램
(Fig. 3) Diagram of Object Inheritance for E-Mail System

정되며, 문서 관리는 다시 서류, 메모, 게시문, 문서라는 클래스로 분해되어 서브 모듈이 형성되어있는 것을 알 수 있다. 그리고 서류는 메모, 게시문, 문서의 추상화 클래스가 되며, 게시문은 문서의 상위 추상화 클래스이다.

추상화 클래스에서 출력, 정정, 조회, 삭제는 실제 하위 클래스에서 구체적으로 구현되며, 하위 클래스별로 해당 클래스에 필요한 여러기능들이 추가될 수 있다. 즉, 게시문 클래스에서의 출력, 정정, 조회, 삭제와 문서 클래스에서의 출력, 정정, 조회, 삭제는 같은 부분도 있지만 하위 클래스로 갈수록 구체화되면서 해당 클래스에서 필요한 기능들이 추가된다.

게시문 클래스의 출력에서는 게시 기간, 제목, 내용들이 그 대상이 되지만 문서 클래스에서는 제목, 문서 번호, 결재, 참조, 발신일 등 클래스의 속성들이 대상이 되므로 제목, 내용은 같지만 나머지는 다른 기능이 필요하다.

객체 모델 모듈을 명세화하는 작업 산출물은 객체 모델 모듈 다이어그램(OMMD)과 객체들의 실체를 상속 관계를 적용하여 차트로 정의한 객체 속성 정의서(OAD) 등이 있다. 본 논문에서는 OMMD만을 실 시스템의 예로 제시하였다.



(그림 4) E-Mail 시스템 객체 모델 다이어그램
(Fig. 4) Diagram of Object Model for E-Mail System

4.3 동적 모델 모듈 설계

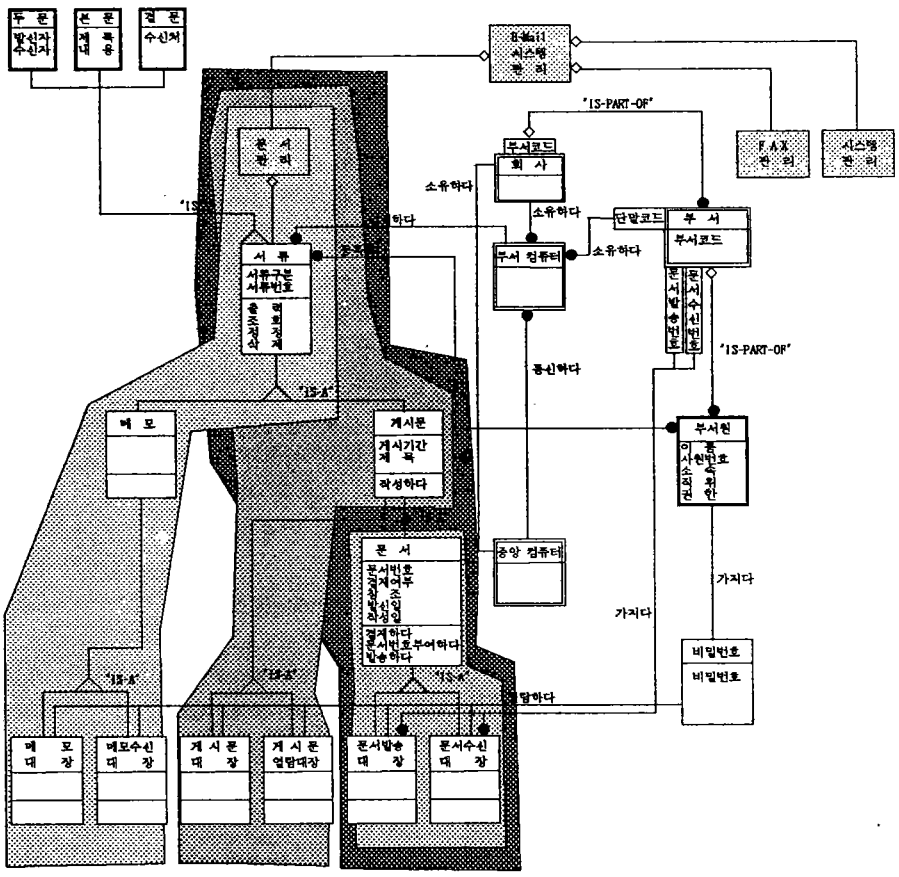
동적 모델 모듈 설계에서는 객체 모델 모듈 설계 단계에서 추출한 추상화 클래스 및 클래스를 바탕으로 시스템의 내부에서 발생하여 상태 변화를 일으키는 내부 이벤트(internal event)를 중심으로 이들 행위의 추적(trace)과 활동(action) 등을 기술한다. 물론 분석 단계의 동적 모델링을 참조하여 설계 단계의 동적 모델을 작성한다. 다만, 분해 기준이 객체 모델 모듈의 추상화 클래스 및 클래스가 된다는 점이다.

(그림 6, 7)은 서류, 게시문 추상화 클래스로부터 순차적으로 상속을 받는 문서 관리 추상화 클래스에 대해 문서 발송 업무에서 발송 행위들의 사건 추적(event trace)과 활동(action)을 나타낸 것이다. 여기에서 활동 기술서는 모듈 설계 단계의 최종 산출물인 프로그램 단위가 되며, 객체, 기능 모델 모듈과 결합하여 구현된다.

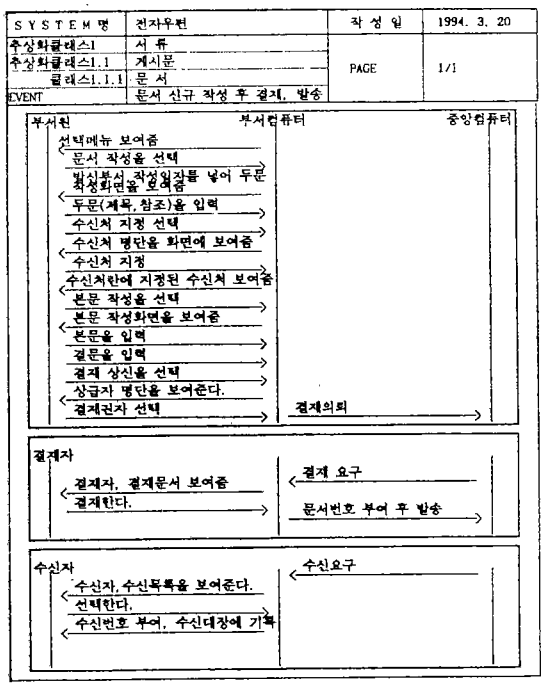
4.4 기능 모델 모듈 설계

기능 모델 모듈 설계에서도 동적 모델 모듈 설계에서와 같이 객체 모델 모듈 설계 단계에서 추출한 추상화 클래스 및 클래스를 바탕으로 시스템의 프로세스(process)에 따라 중심으로 객체들의 흐름을 나타낸다. 여기에서 프로세스(process)란, 동적 모델의 이벤트(event)와 같은 개념으로 간주하여도 된다[16].

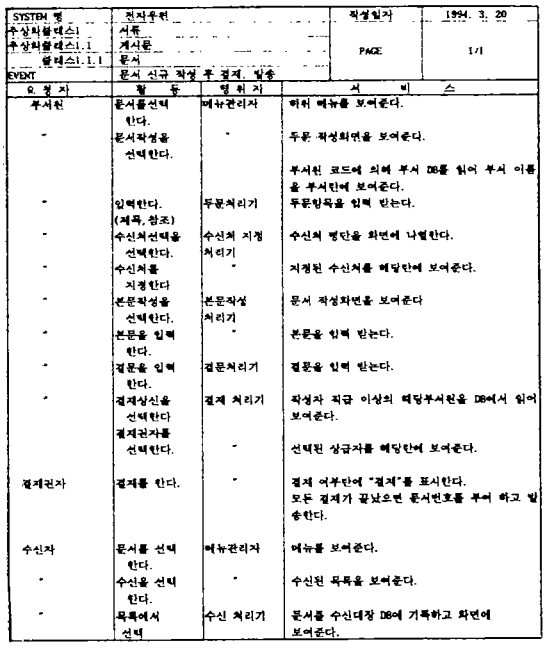
그리고 기능 모듈 설계 역시 분석 단계의 기능 모델을 참조하여 설계되어지며, 다만 분해 기준



(그림 5) E-Mail시스템 객체 모델 모듈 다이어그램
(Fig. 5) Diagram of Object Model Module for E-Mail System



(그림 6) 문서 발송 사건 추적도
(Fig. 6) Sheet of send on document event trace



(그림 7) 문서 발송 활동 기술서
(Fig. 7) Sheet of dynamic description for send on document

이 객체 모델 모듈의 추상화 클래스 및 클래스가 되고, 기능 모델의 객체 흐름도에 객체들의 흐름을 실제로 기술한다는 점이 다르다. 객체 상세 흐름도는 객체 모듈 설계 단계의 최종 산출물인 프로그램 단위가 되며, 객체, 동적 모델 모듈과 결합하여 구현된다. 본 논문에서는 객체 상세 흐름도 명세화는 생략하였다.

4.5 객체 지향 프로그램 설계

객체 지향 프로그램 설계 단계는 객체 모듈 설계의 최종 산출물인 객체, 동적, 기능 모델 모듈을 결합한 프로그램 처리 설명서를 참조하여 구현되어진다. 즉, 추상화 클래스 및 클래스를 바탕으로 이벤트 또는, 프로세스(event or process)를 중심으로 최하위 단계의 프로그램 단위 모듈을 말한다.

그리고 OOD에 의한 설계는 실제 개발 단계(알고리즘, 코딩)에서 다른 설계 정보가 추가되는 경우가 종종 발생한다. 이것은 설계 모델이 잘못된 것이 아니라 설계자가 객체 설계 모델의 복잡성을 줄이기 위하여 사용자 인터페이스 설계를 업무와 독립되게 구현할 수 있도록 하는 경우가 있기 때문이다.

객체 지향 프로그램 설계 모델을 명세화하는 작업 산출물은 프로그램 계층 구조도, 프로그램 처리 설명서, 모듈 알고리즘 및 사용자 인터페이스 설계 항목의 화면 레이아웃 및 레포트 레이아웃 등이 있다.

본 논문에서는 모듈 알고리즘만을 작성하여 객체 모듈 분해 방법의 적합성을 증명하고자 한다.

4.6 비교 분석

본 논문에서는 Rumbaugh의 OMT방법의 객체 모델, 동적 모델, 기능 모델을 바탕으로 개선된 객체 설계 방법을 제시함과 동시에 정형화된 모듈 분해 이론을 실 시스템에 적용하여 구현하였다. 그러나 객체 지향 기술을 적용한 정형화된 방법이라 할지라도 이것을 평가 할 수 있는 공식화된 평가 기준은 없다. 따라서 여기에서는 지금까지 사용해온 경험(heuristic)을 토대로 소프트

웨어의 품질을 평가한 [3.3 설계 검증]에서 Pressman의 평가기준을 참고하여 모듈 분해 방법을 중심으로 기존의 객체 지향 방법들과 본 논문에서 제시하는 방법에 관하여 특성 및 구현 원리를 살펴 보고, 그리고 이들 방법들의 장·단점을 비교하고자 한다.

먼저 Coad/Yourdon의 방법을 살펴보면, 객체 식별, 구조 식별, 서브젝트 정의, 속성과 연관 관

```
// 서류 추상화 클래스 : 두문 상속, 본문 상속, 결문 상속

class Paper : public Hearer,
              public Contents,
              public End_Contents {
      char Paper_Number[13];          // 서류번호
      char Paper_Class;               // 서류구분
public:
      Paper();
      ~Paper();
      virtual void Edit() = 0;
      virtual void Print() = 0;
      virtual void Query() = 0;
      virtual void Delete() = 0;
      virtual void Store() = 0;
      virtual void Update() = 0;
};
```

```
// 게시문 추상화 클래스 : 서류 클래스에서 상속

class Bulletin : public Paper {
      char From_Date[6];    // 게시 시작일
      char To_date[6];      // 게시 종료일
public:
      Bulletin(void);
      ~Bulletin(void);
      virtual int Check_Date(void);
      virtual void Set_From_To(char*);
      virtual void Notice();      // 게시하다.
```

```
// 문서 클래스 : 게시문에서 상속

class Document : public Bulletin {
      char* Reference;              // 참조
      char* Writer;                 // 기안자
      char Telphone[16];            // 전화번호
      char Write_Date[];            // 기안일자
      char Writet_Title[6];         // 기안자 직위
      char Send_Date[6];
      Confirm* Confirm_ptr;         // 결재자, 결재일, 결재자 직급, 결재여부, 수기
public:
      Document();
      ~Document();
     void Set_reference(char*);
      void Set_Writer(char*);
      void Set_TelePhone(char*);
      void Set_WriteDate(char*);
      void Set_Write_title(char*);
      void Set_SendDate(char*);
      char* Get_Reference();
      char* Get_Writer();
      char* Get_TelePhone();
      char* Get_WriteDate();
      char* Get_WriteTitle();
      char* Get_SendDate();
      int Doc_num_Check();          // 문서번호 자동입력
      int ConfirmCheck();           // 결재여부 검사
      void Write_Date_Input();      // default is current date 작성일 입력
      void Confirmation();          // 결재
      void Send_Documents();        // 문서발송
```

```
// EVENT : 발송

void TGM1000Dialogy::GM1030()
{
  getData(hDialogData.Reference);     // 화면에 입력된 데이타를 읽는다.
  Doc.Set_Reference(hDialogData.Writer);      // 참조
  Doc.Set_Writer(hDialogData.Writer);         // 기안자
  Doc.Set_TelePhone(hDialogData.Phone);       // 전화번호
  Doc.Set_WriteTitle(hDialogData.WriteDate);  // 기안일
  Doc.Set_SendDate(hDialogData.SendDate);     // 발송일

  if(Check_수신_발신_제목();)          // 수신, 발신, 제목은 문서의 필수임
    if(Doc_Num_Check()) {
      SendMessage("문서번호:",Doc.Paper_Number);
      Doc.Send_Documents();
      }
      else SendMessage("문서번호가 잘 못 되었습니다.");
    else messageBox("수신, 발신, 제목은 반드시 있어야 합니다.");
}
```

(그림 8) 모듈 알고리즘
(Fig. 8) Module Algorithms

계 정의의 5단계의 과정을 통하여 서브젝트(subject)라는 개념을 적용하여 시스템 분석자가 한 번에 취급하기 적당한 분량으로 객체를 분해한다. 그리고 서브젝트 사이에 메시지를 표현하여 하나의 시스템 모듈을 형성한다. 따라서 이 방법은 모듈 분해 기준과 분해된 단위 모듈을 이용한 구현 지침에 대한 세부 사항이 부족함을 알 수 있다.

Booch의 방법은 클래스와 객체의 식별, 이들에 대한 의미와 관계 식별, 구현의 4단계로 구성되어 있으며, 시스템의 형성 구조를 모형화하는 자료 흐름도(DFD)를 사용해서 객체를 분해하고, 객체들간의 인터페이스를 찾아 이것들을 Ada 프로그램으로 변환시키는 방법이다. 그러나 이 방법은 모듈 분해 기준이 기존의 프로세스 중심 방법인 구조적 방법의 자료 흐름도라는 것은 객체 지향 방법의 기본 원리에 어긋나며, 그리고 Ada 프로그램 언어라는 언어 제한적인 면이 있다.

Shlaer/Mellor의 방법은 정보 모델, 상태 모델, 프로세스 모델의 3단계로 구성되며, 하나의 시스템을 몇개의 영역(domain)으로 분할하여 서브 시스템을 구성한후 이들의 객체, 상태, 처리 모델을 생성한다. 여기에서 분할된 영역은 응용 시스템 영역(application system domain), 서비스 영역(service domain), 구조 영역(architecture domain), 구현 영역(implementation domain)으로 분류된다. 이 방법은 기존의 방법중에서 시스템의 모듈 분해 과정에 대한 많은 언급과 그리고 가장 우수하다고 할 수 있으나 객체 지향 개념에 근거한 방법이라기 보다는 기존의 정보 공학 방법과 유사성이 있으며, 설계 단계에서 구현에 대한 세부 사항이 부족하다.

Rumbaugh의 방법은 클래스의 외부 명세를 먼저 정의한후 일반화, 집단화, 관계성 원리에 입각하여 객체 모델, 동적 모델, 기능 모델을 설계한다. 이 방법에서는 모듈 설계에 대한 구체적인 언급은 없으며 다만 수직 분할이라는 개념으로 시스템을 몇개의 독립된 시스템들로 나누는 것을 제시하였다. 따라서 이 방법은 소프트웨어 개발 시스템의 전 생명주기 지원과 DB구조화에 용이하며, 시스템의 다면성을 가시화 하기 위한

충분한 표기법을 가지고 있다는 장점은 있으나 모듈 분해 기준이 부족함을 알 수 있다.

이상과 같이 모듈 분해에 관한 기존의 객체 지향 방법들을 살펴 보았듯이 기종 방법들은 설계 모델을 구현 모델에 직접 적용하기 어렵거나, 적용할 수 없으며, 개발 완료 후 재사용할 수 있는 단위 모듈 설계는 기대하기 어렵다. 그러나 본 논문에서 제시하는 모듈 분해 방법은 시스템의 다면성을 가시화 할 수 있는 Rumbaugh의 OMT 분석 모델을 바탕으로 개선된 설계 방법 즉, 객체 설계, 모듈 설계, 프로그램 설계 단계로 분류하여 상향식 방법으로 추출된 가장 상위의 추상화 클래스(abstraction class)에서부터 하향식 방법으로 프로그램의 실제 구현 단계인 가장 하위의 이벤트(event) 단위까지 객체 모델, 동적 모델, 기능 모델을 분해하므로서 설계 단계에서 구현 단계로의 효율적인 전환이 가능하다. 또한 하나의 객체 및 클래스에 대한 데이타(data), 제어(control), 기능(function) 측면을 각각 설계하여 프로그램 최종 단위 모듈을 결합할 수 있으므로 시스템의 다면성을 나타낼 수 있다. 따라서 시스템 개발 완료후에도 유지 보수 용이성과 단위 모듈을 재사용할 수 있는 효율성을 제공한다.

본 논문에서 제시한 방법은 지금까지 다른 모듈 분해 방법이 발표된 바가 없으므로 얼마나 효율적인지 상대적인 비교는 불가능하나, [3.2 모듈 분해 기준 및 분해 방법]의 (그림 1)과 (그림 2)에서 증명한 바와 같이 이 방법이 모듈 분해 및 결합의 원리에 의하여 수치적으로 증명할 수 있는 정형화된 이론에 타당하다는 것을 알 수 있다.

5. 결 론

기존의 객체 지향 설계 방법은 모듈 설계 방법과 분해 기준의 미흡으로 객체 지향 소프트웨어 구현을 전체 시스템 단위로 구현함으로써 하나의 소프트웨어 컴포넌트를 이해하기 위해서는 전체적인 객체 설계 모델 및 응용 프로그램을 분석해야 하는 많은 문제점이 있다. 따라서 본 논문은 객체 지향 설계를 위한 정형화된 모듈 분해 기준을 제시하여 기존 방법등에 대한 문제점을 개선하고, 실 시스템 적용사례를 들어 제시한 이론의 증명과 실용성을 보이고, 시스템의 설계 모델과 구현 모델간의 일치성을 확인하였다. 또한, 본 논문에서 제시한 객체 지향 설계를 위한 모듈 분해 기준은 대규모 프로젝트 수행시 효과적으로 활용할 수 있으며, 소프트웨어 공학에서 추구하는 소프트웨어 생산성 향상을 위한 코드 재사용에 많은 영향을 줄 것으로 기대된다.

앞으로의 연구 과제로는 분해된 모듈을 효과적으로 관리하고, 활용할 수 있도록 체계적인 버전 관리의 연구가 필요하다.

참 고 문 헌

[1] Brian Henderson-Sellers, "A Book of Object-Oriented Knowledge", Prentice-Hall, pp. 32-37, 72-76, 1991.

[2] C. Paul Allen, "Effective Structured Techniques", Prentice-Hall, Inc. pp. 81-88, 1991.

[3] Derek Coleman, Partrick Arnold, Stepanie Bodoff Chris Dollim, Helena Gilchrist, Fiona Hayes Paul Jeremaes, "Object-Oriented Development The Fusion Method", Prentice-Hall, Inc, pp. 96-100, 1994.

[4] E. Yourdon, "Structure Design Workshop", Third Edition, Yourdon Press, pp. 5.6-5.8, B.1-C.28, 1982.

[5] E. Yourdon, "Structured Analysis and System Specfication workshop", Edition, Yourdon press, pp. 3.5.1-3.7.5, 1984.

[6] E. Yourdon, "Software Engineering an Executive Briefing", G.I.S, pp. 53-84, 1992.

[7] G. Booch, "Object-Oriented Design with Applications", the Benjamin/Cumming Publishing Company,Inc. pp. 155-184, 187-194, 1991.

[8] Gunther Blaschek, "Object-Oriented Programming with prototypes", Springer-Verlag Berlin Heidelberg, pp. 67-70, 262-263, 1994.

[9] I. Jacoson, M. Christerson, P, Jonsson, G. Overgaard, "Object-Oriented Software Engineering- A case Driven Approach", ACM, Inc. pp. 465-493, 1992.

[10] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design", Prentice-Hall,Inc. pp. 84-89, 148-183, 227-229, 260-274, 1991.

[11] Kevin lano & Howard Haughton, "Object-Oriented Specification CASE studies", Prentice-Hall, Inc(UK) pp. 2-19, 32-34, 72-79, 1994.

[12] P. Coad, E. Yourdon, "Object-Oriented Design", Yourdon Press Computing Series II, Prentice-Hall, 1991.

[13] P. Coad,Jill Nicola, "Object-Oriented Programming", Prentice-Hall, pp. 227-230, 1993.

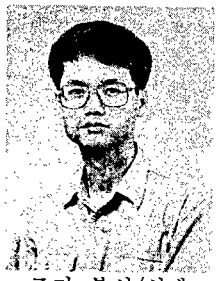
[14] R. S. Pressman, "Software Engineering: A practitioner's approach", McGRAW-HILL, 1987.

[15] Stephen L. Montgomery, "Object-Oriented Information Engineering Analysis, Design and Implementation", Academic Press, Inc. pp. 119-141, 184-185, 201-202, 1994.

[16] S. Shlaer, S. J. Meller, "Object-Life Cycles Modeling the World in States", Prentice-Hall, Inc, pp. 133-160, 1992.

[17] Wilkie, "Object-Oriented Software Engineering", Addison Wesley, pp. 106-132, 1993.

[18] 허계범, "효율적인 응용 소프트웨어 설계를 위한 객체 중심 데이타 모델링 방법에 관한 연구", 광운대학교 전산대학원 석사논문, pp. 12-20, 1993. 6.



허 계 범

1989년 경기대학교 경상대학 응용통계학과 경제학사
1993년 광운대학교 전산대학원 전자계산학과 이학석사
1988~현재 (주)경인양행 전산부 과장, 신흥전문대학 전산정보처리학과 강사
관심분야 : 소프트웨어 공학(구조적 분석/설계, 정보공학, 객체 지향 분석/설계, CASE, 프로그래밍 언어).

최 영 근

1980년 서울대학교 사범대학 수학 교육과 이학사
1982년 서울대학교 계산통계학과 이학석사
1989년 서울대학교 계산통계학과 이학박사
1983~현재 광운대학교 이과대학 전산과 교수
관심분야 : 프로그래밍 언어, 병렬 컴퓨터, 병렬 프로그래밍, 객체 지향 프로그래밍 언어 및 설계, 분산처리.