래피드 프로토타입핑 기법을 사용한 객체 지향 클래스 계층 구조 설계 방법

허 계 범† · 최 영 근†

요 약

객체 지향 설계 모델에서 클래스 계층 구조는 소프트웨어 재사용성과 복잡한 시스템 설계시에 효율적이다.

본 논문에서는 래피드 프로토타입핑 기법을 사용한 객체 지향 클래스 계층 구조 설계 방법을 제시한다. 이 방법은 객체 모델링 단계에서 식별된 클래스들을 새로운 분류 방법을 사용하여 관계성 식별과 유사성을 측정한다. 여기에는 클래스의 속성과 메소드의 측정이 요구된다. 하나의 설계 모듈인 클래스 계층 구조는 대화식으로 반복적인 작업을 통하여 생성되며, 각 모듈은 참조 관계, 상속 관계, 복합 관계로 구성된다. 이들 정보는 구현 및 프로그램 유지보수시에 도움을 주기 위하여 테이블에 저장하여 클래스 관계성을 그래프로 표현하고 노드 클래스를 아이콘화 하였다. 따라서 새로운 클래스 추가 및 삭제 작업이 용이하여 클래스 계층 구조 재구성과 설계 정보 재사용시에 효율적이다.

본 논문에서 제시하는 방법은 프로토타입 시스템으로 뿐만 아니라 실제 시스템에 최종 전환할 수 있기 때문에 시스템 분석, 설계, 구현의 능률을 높일 수 있다.

The Object-Oriented Class Hierarchy Structure Design Method using the Rapid Prototyping Techniques

Kwae Bum Heo† · Young Eun Choi†

ABSTRACT

The class hierarchy structure in an object-oriented design model is effective to the software reusability and the design of complex system.

This paper suggests the object-oriented class hierarchy structure design method using the rapid prototyping techniques. In this method, relationship recognition and similarity are estimated by the new class classification in object modeling level. Then the estimation of attribute and method in class is needed. Each design module such as class hierarchy structure which is generated with interactive and repeated work consists of reference relationship, inheritance relationship and composite relationship. These information are stored in the table to maintenance the program and implementation, the class relationship is represented with graph and the node class is iconized. This method is effective in restructuring of class hierarchy are reusing of design information, because of addition of new class and deletion with ease.

The efficiency of system analysis, design and implementation is enhanced by converting into prototype system and real system.

† 정 회 원:광운대학교 전자계산학과
논문접수:1997년 1월 29일, 심사완료:1997년 9월 5일

1. 서 론

래피드 프로토타입핑(rapid prototyping) 기법은 적은 비용으로 소프트웨어 개발 생명 주기의 초기에 소프트웨어 프로토타입을 생성하기 위한 방법을 말한다. 이 방법은 효율성에 있어 프로토타입 시스템에서의 사용자 경험이 요구의 불일치성, 모호성, 불완전성 등을 쉽게 지적해 주기 때문에 기존의 요구 명세 기법보다 목적 시스템에 대한 사실적인 정보를 제공한다[2, 5, 7, 10].

그러나 프로토타입핑 개발 방법은 원가 측면에서 여러 가지 도구를 추가로 필요로 하므로 도구 사용에 따른 비용 부담이 존재한다. 또한 큰 규모의 시스템에 대해서는 좀더 명백한 지침과 절차가 요구되므로 자료와 사용자가 많은 환경일 경우 내부 기술의 조정이 어렵고 비효율적이다[8].

클래스 계층 구조 설계 과정은 요구사항의 변경으로 인한 새로운 클래스의 추가 및 클래스들의 다양한 관계로 인한 반복적이고 점증적인 개발 과정을 필요로 한다[3, 4, 6]. 그러나 지금까지의 연구 대상은 재공학(reengineering) 또는 재구조화(restructuring) 즉, 소프트웨어 재사용을 위하여 이미 구현된 대상 언어들로부터 유사성을 측정하여 시스템을 구축하는 방법들이다. 따라서 올바른 설계의 결과가 우수한 소프트웨어를 생산하듯 분석 단계에서 설계 단계로 대화적(interactive)인 프로타입핑 기법을 사용하여 재사용할 수 있는 단위로 만들어진다면 부품의 이해 및 수정에 대한 용이성과 유연성을 제공하여 소프트웨어 구현시에 많은 잇점을 제공한다.

따라서 본 논문에서는 클래스 계층의 설계에 따른 비용을 최소화 하고 설계 문서의 재사용을 효과적으로 수행할 수 있는 방법을 제시한다. 이 방법은 먼저 클래스를 새롭게 분류하여 클래스들간의 관계성 식별과 유사성을 측정한다. 여기에는 클래스의 속성과 메소드의 측정이 요구되는데 객체 모델링 단계에서 정의된 정보 저장소(information repository) 정보를 사용한다. 하나의 설계 모듈인 클래스 계층 구조는 대화식으로 반복적인 작업을 통하여 생성된다. 그리고 이들 정보는 테이블에 저장하여 클래스 관계성을 그래프로 표현하고 노드 클래스를 아이콘화 하였다. 따라서 새로운 클래스 추가 및 삭제 작업이 용이하여 클래스 계층 구조 재구성과 설계 정보 재사용시에 효율적이다. 본 논문에서 제시하는 방법은 프로토타입 시스템으로 뿐만 아니라 실제 시스템에 최종 전환할 수 있기 때문에 시스템 분석, 설계, 구현의 능률을 높일 수 있다.

본 논문에서 제시하는 방법은 객체 지향 분산 환경의 대형 소프트웨어 개발에서 유용하게 사용되어 질 수 있으며 연구 내용은 다음과 같다.

첫째, 분석 단계로부터 클래스를 분류하여 모듈 생성과정을 자연스럽게 유도하고, 둘째, 객체 및 클래스 관계성 정의를 확장하여 정의하며, 셋째, 클래스간의 관계성을 분석하여 유사성을 측정한후 클래스 속성과 메소드를 추출하여 상속 및 복합 계층 구조를 형성하는 과정을 보인다.

본 논문의 구성은 다음과 같다. 제2장에서 관련 연구들과 이들의 문제점에 관하여 살펴보고, 제3장에서는 제안하는 분석 및 설계 방법을 제시하고, 제4장에서는 결론 및 앞으로의 연구 방향에 관하여 서술한다.

2. 관련 연구

본 논문과 관련된 연구를 살펴보면, 래피드 프로토타입핑 방법, 클래스 계층 구조 설계 방법, 클래스 유사성 측정 및 계산 방법등이 있다. 그러나 클래스 유사성 계산 방법은 제3장 3.3.2절에서 살펴 보기로 한다.

2.1 래피드 프로토타입핑 방법

래피드 프로토타이핑 방법은 소프트웨어 생명주기 동안 점진적으로 소프트웨어의 품질을 증진시키는데에 매우 중요한 역할을 한다. 이러한 래피드 프로토타입핑의 개념들이 소프트웨어 생명주기에서 어떻게 적용하는지에 따라 다음과 같은 모델들이 있다[2, 5, 7, 10].

(1) Temporary Model

이 모델은 구현 단계 즉, 코딩 단계에 적용되어 유용한 프로그래밍 언어에서 개발되고 소프트웨어 가용성이 점검된다. 작업 모델은 목적 시스템으로 폐기되거나 수정된다. 목적 시스템은 작업 시스템으로부터의 경험에 의하여 구성되어지는데 최상의 경우는 구현 언어와 프로토타입을 구성하기 위한 언어가 동일한 것이다.

(2) Evolutionary Model

이 모델은 소프트웨어 분석 단계의 요구 명세에 적용된다. 즉, 시스템은 비형식 명세로부터 구현되고 프로토타입은 요구와 설계의 완전성, 일치성, 모호성과 같은 타당성의 기준이나 사용자의 재고찰을 통하여 검토되고 발전된다. 그리고 사용자의 요구와 타당성 기준을 만족했을 때 실제 시스템이 프로토타입으로부터 구성된다.

(3) Revolutionary Model

이 모델은 소프트웨어 모든 생명주기에 적용된다. 프로토타입 자체는 형식 명세가 되고 형식 명세는 작업 시스템으로 자동적인 변형이 되는데 프로토타입 시스템이 아니라 실제 시스템으로 변형된다. 따라서 프로토타입 개발은 가용성을 기술하고 시스템의 기대되는 가능성에 대해 고객과 개발자 사이의 이해를 확증하므로 실제 시스템 개발과 관계이 있다. 이러한 모델을 소프트웨어 시스템 특성의 정확한 모델링에 부합되도록 하기 위한 시스템의 프로토타입을 구축하는데는 세가지 형태가 있다. 이것은 개발자가 프로토타입을 바라는 시스템의 특수한 면들에 따라 사용자 인터페이스 프로토타입, 기능 프로토타입, 성능 프로토타입으로 분류될 수 있다.

그리고 래피드 프로토타입핑의 구현을 위해 많이 사용하는 도구로는 4세대 언어, 보고서 생성기(report generator), 응용 생성기(application generator), 데이타 사전 시스템, 재사용 코드의 라이브러리등이 있다.

2.2 클래스 계층 구조 설계 방법

클래스 계층은 상속 관계 및 복합 관계로 설계되어진다. 먼저 클래스 상속 계층 구조 설계 방법을 살펴보기로 한다[3, 4, 6, 12].

첫째, 하향식 방법으로 가장 일반적인 상위 클래스로부터 세분화(specialization)를 통하여 특정한 하위 클래스를 설계한다. 이 방법은 하위 클래스가 상위 클래스보다 특정화된 행위를 나타내는 경우와, 그리고 상위 클래스가 어떤 인터페이스만을 표현하는 추상 클래스가되어 하위 클래스가 상위 클래스의 인터페이스에서 명시된 행위를 실제 구현하는 경우이다.

둘째, 상향식 방법으로 여러개의 별도의 클래스를 설계한후 클래스들 사이의 공통성을 추출하여 하나 또는 그이상의 클래스를 설계한다. 즉, 이미 정의된 클래스들 중에 공통적인 행위들인 서비스의 종류가 비슷한 것들을 모아 이 클래스들의 부모 클래스를 정의하여 공통 행위 및 상태 정보들을 위로 올려 공유함으로써 코드 재활용 및 공유의 장점을 가진다.

다음은 복합 계층 구성방법을 살펴보기로 한다[4, 12]. 복합 관계는 두 클래스간의 관계가 부분('is part-of) 관계의 형태를 표현하며, 상속 관계가 클래스와 관련이 있는 반면 복합 관계는 객체와 연관이 된다. 따라서 복합 관계는 관련 있는 객체들이 모여서 구성된 집합화의 예로서 복합 객체(composite object)라 한다. 하나의 객체를 구성하는 구성원 역시 객체가 된다는 성질을 말한다. 여기에서 복합 객체를 구성하는 객체를 구성 객체(sub-object)라 하며 이는 재귀적(recursive)인 의미를 갖는다. 여기에서 구성 객체가 한 객체에만 속할 경우에는 배타적(exclusive) 성격이 되며, 동시에 여러 객체에 속할 수 있을 경우에는 공유적(shared) 성격이 된다. 따라서 복합 객체는 구성 객체와의 관계를 기술함으로써 정의되며, 이들 관계는 복합 객체 계층 구성도로 표현된다.

2.3 유사성 측정 방법

클래스 계층 구조 설계시에는 문서 및 소프트웨어의 재사용 측면을 고려하여 유사성을 측정 한다. 대표적인 유사성 측정 방법은 inner-product measure, cosine measure, pseudo-cosine measure, Dice measure 방법등이 있다[9]. 이러한 방법은 두 문서(i, j)간에 가중치들을 결정한다. 이 가중치들은 정규화하는 식을 이용하여 두문서의 유사성을 결정한다. 문서의 유사성을 결정하는 Dice 측정방법에서는 다음식으로 결정한다.

$$s_{ij} = \frac{2 \times C}{A + B}$$

여기에서 C는 i 문서와 j 문서에 공통적으로 존재하는 의미있는 단어의 수이고, A는 i문서에 존재하는 의미있는 단어들의 수이고, B는 j문서에 존재하는 의미있는 단어들의 수이다.

그러나 이와 같은 방법들은 객체 지향 기술의 핵심 개념이라 할 수 있는 상속성을 표현하지 못하기 때문에 클래스 계층 구조를 효율적으로 변형하지 못하는

단점이 있다.

2.4 기존 방법의 문제점

지금까지 관련 연구를 살펴보았다. 객체 지향 설계 과정은 요구사항의 변경으로 인한 새로운 클래스의 추가 및 클래스들의 다양한 관계로 인한 반복적이고 점증적인 개발 과정을 필요로 한다. 그러나 지금까지의 연구 대상은 소프트웨어의 재사용을 위하여 이미 구현된 대상 언어들로부터 유사성을 측정하여 시스템을 재구축하는 방법들이다. 그리고 클래스 계층 구조 설계에 대한 지침은 많이 존재한다. 그러나 분석 단계에서부터 일관성 있는 개발 도구(tool)를 대화적(interactive)인 프로타입핑 기법을 사용한 객체 지향 설계 방법은 거의 없는 실정이다. 이러한 이유는 유용한 도구를 개발하기 위한 추가적인 비용과 큰 규모 시스템을 개발하기 위한 좀더 명백한 지침과 절차가 결여되어 있기 때문이다.

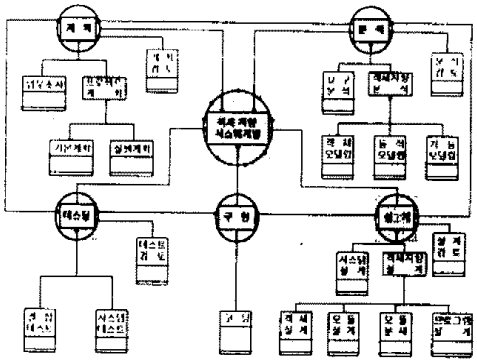
따라서 본 논문에서 제시하는 설계 방법은 요구 분석 단계에서 식별된 클래스 정보을 사용하여 프로토타입 시스템으로 뿐만 아니라 실제 시스템에 최종 전환할 수 있도록 한다. 그러므로 소프트웨어 개발 비용 절약과 분석, 설계, 구현의 능률을 높일 수 있도록 할 것이다.

3. 클래스 계층 구조 설계 방법

여기에서는 본 논문에서 제시하고자 하는 클래스 계층 구조 설계 방법의 적용 방법과 절차를 제시함과 동시에 세부적인 분석 및 설계 방법에 관하여 서술하고자 한다.

3.1 적용 방법

본 논문에서 제시하는 방법은 Rumbaugh의 OMT [6]를 바탕으로 설계 절차를 세분화하여 실제 업무에 적용할 수 있도록 하는데 그 목적이 있다. (그림 1)은 래피드 프로토타입핑 방법을 사용한 객체 지향 시스템 개발 과정을 나타내고 있다[14, 15]. OMT 방법은 객체, 동적, 기능 모델로 구성되어 있는데 본 논문의 연구 범위는 분석, 설계, 객체 모듈 설계단계에서 객체 모델만으로 한다.



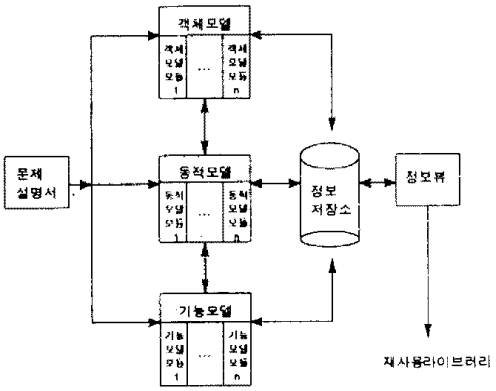
(그림 1) 객체 지향 시스템 개발 방법
(Fig. 1) Object-Oriented System Development Method

3.2 적용 절차 및 분석 방법

본 논문에서 제시하는 프로토타입핑 과정은 적합성 결정→요구 사항의 파악→프로토타입 개발→프로토타입 사용 및 평가→프로토 타입의 수정→최종 시스템으로 변환등이며, 본 논문에서 제안하는 객체 지향 클래스 계층 설계를 위한 분석 및 설계 절차는 다음과 같다.

① 업무 요구 분석:문제 설명서 작성
② 분석 단계에서 클래스 식별:클래스 분류(외부 클래스, 내부 클래스, 인터페이스 클래스)하고 클래스 속성 추가(멤버 함수 포함)하며 클래스 관계성을 식별(단순 참조 관계)한다.
③ 단계 2에서 식별된 클래스 정보를 정보 저장소(information repository)에 저장
④ 단계 3의 정보를 관계성을 적용하여 래피드 프로토타입핑 방법으로 시뮬레이션 처리하며
⑤ 상속 및 복합 관계를 적용하여 클래스 계층 구조 유도
⑥ 클래스간의 유사성 측정
⑦ 클래스 복잡도 측정
⑧ 클래스간의 관계성 재정의
⑨ 조율(tuning) 및 클래스 계층 구조 재정의(대화식으로 반복적인 작업을 통하여)
⑩ 클래스 계층 구조도 작성
⑪ 모듈별 클래스 관계성를 정보 저장소에 저장(클래스 관계성을 그래프로 표현하고 노드 클래스를 아이콘화)

(그림 2)는 정보 저장소에 저장된 객체 지향 분석 정보를 나타내고 있다. 본 논문에서는 설계 관점에서 클래스 계층 구조를 설계하기 위한 객체 모델정보만을 사용한다. 즉, 식별된 클래스, 데이타 사전, 클래스 관계성 정보등이다. 본 논문에서 정보 저장소 테이블은 ACCESS DB를 사용하였다.

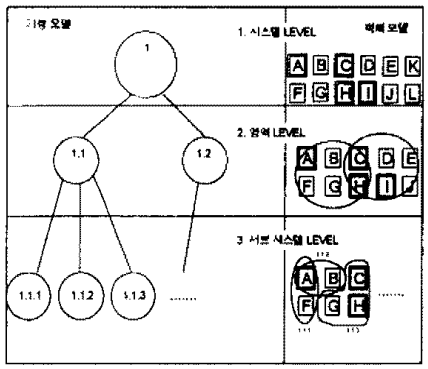


(그림 2) 정보 저장소
(Fig. 2) Information Repository

다음은 요구 분석 단계에서 얻어진 문제 설명서를 토대로 소프트웨어 개발에 실제 참여할 클래스들을 분류하고자 한다. 본 논문에서 제안하는 클래스는 다음과 같이 분류된다.

내부 클래스는 시스템 내부에 존재하는 객체로서 행위자, 역할자, 대행자 등의 기능이 직접 가능한 데이타 저장소에 해당하는 클래스이다. 외부 클래스는 시스템 범위 밖에 존재하는 객체로서, 내부 클래스와 직접적인 관계가 없는 클래스이다. 인터페이스 클래스는 내부 클래스를 만족하며 모듈을 형성하는 클래스로서의 역할을 한다. 즉, 다른 시스템 또는 다른 모듈에 관계된 클래스로 정의할 수 있다.

이와 같은 클래스의 분류는 클래스의 관계성 설정과 대형 소프트웨어 개발시에 매우 중요하게 사용되어진다. 즉, 외부 클래스는 소프트웨어 개발시에 직접적인 필요가 없으므로 설계 단계에서 제외시키며, 인터페이스 클래스는 시스템→영역→서브 시스템 수준(level)으로 업무의 단위를 분해하였을 때 공통적으로 필요한 클래스이므로 이들을 구분하여 표기 하여야 한다. 여기에서 인터페이스 클래스는 분산 환경이라면 지역별(sites) 클래스 운용 비용 분석을 하여 객체 할당 정책에 참여하게 된다. 또한, 내부 클래스와 인터페이스 클래스를 일반화하여 상위의 추상화 클래스를 만들 수 있다. 그러나 이것은 현재의 단계에서는 추출할 수 없고 설계 단계에서 클래스의 상속 및 복합 관계를 적용하여 생성할 수 있다.



(그림 3) 객체 모델과 기능 모델에서 클래스 분류 방법
(Fig. 3) Class Classification Methods of Object Model and Functional Model

본 논문에서는 □, □, □ 그림으로 각각 외부 클래스, 내부 클래스, 인터페이스 클래스를 표기하였다. (그림 3)에서는 객체 모델과 기능 모델의 각 업무 영역 수준에서 클래스의 분류된 상태를 보여주고 있다. 앞에서도 설명하였듯이 클래스의 분류 목적은 단위 모듈간에 관계성을 명확히 하기 위한 것이다. 그러나 분산 환경이라면 지역별 업무단위를 설정하여 할당해야 하는 문제를 추가적으로 고려하여야 한다. 즉, 할당 방법에서 시스템 수준이라면 문제가 없지만 영역 수준일 때 클래스 C와 H, 서브 시스템 수준일 때 클래스 A, C, H, I에 대해서 클래스 객체 사용 운용 비용 분석을 하여 해당 지역에 위치시켜야 한다.

3.3 설계 방법

여기에서는 클래스 계층 구조 설계를 위한 확장된 형식 정의 및 클래스 유사성 계산 방법을 알아본다. 그리고 이들 정의를 바탕으로 래피드 프로토타입핑 방법으로 시뮬레이션하여 클래스 객체들의 관계성 생성과, 상속 및 복합 관계를 적용한 클래스 계층 구

조의 형성 과정을 보인다.

3.3.1 형식 정의

이와 관계된 클래스 객체들의 관계성을 위한 정형화된 수식 정의에 관한 연구는 상대적으로 미흡하다. 그 이유는 객체 지향 기법의 발전이 수학적 기반보다는 경험적 원리를 기반으로 하고 있기 때문이다. 본 논문에서는 C++ 언어의 개념을 사용하여 보다 확장 정의하였다[11].

[정의 1] 클래스의 관계성은 상속 관계("kind-of", "is-a"), 복합 관계 ("composite of" or "has-a", or "is-part-of"), 참조(협동) 관계("reference-of" or "use-of")로 나뉘어진다. 이 세가지는 모두 연관(association) 관계의 부분 집합이다. 따라서 다음의 조건들을 만족하여야 한다.

①(연관 관계 ⊃ 상속 관계)
②(연관 관계 ⊃ 복합 관계)
③(연관 관계 ⊃ 참조 관계)

[정의 2] 실세계에 존재하는 객체들은 다음의 조건들을 만족하여야 한다.

①연결 연산자(concatenation)를 "•"로 표기(부품 관계) 할 때, 만약 O_i과 O_j가 객체이면 $O_i \bullet O_j$도 객체이다.
②$O_i \bullet O_i = O_i$이다.
③연결 연산자는 교환 법칙이 성립한다.
④연결 연산자는 결합 법칙이 성립한다.

[정의 3] 클래스 상속 계층 구조 관계는 다음의 조건들을 만족하여야 한다.

①클래스 C_i와 C_j가 상속 계층 구조라면 상위 클래스 C_i의 도메인과 하위 클래스 C_j의 도메인은 상호 독립적이어야 한다. 즉, $C_i(d) \cap C_j(d) = 0$의 조건을 만족한다.
②역으로 클래스 C_i와 C_j가 임의의 클래스 C_k의 상속 계층 구조를 가지려면 임의의 클래스 C_i의 도메인과 클래스 C_j의 도메인은 상호 독립이 아니다. 즉, $C_i(d) \cup C_j(d) \neq 0$의 조건을 만족한다.
③클래스 C_i와 C_j가 상속 계층 구조라면 상위 클래스 C_i와 하위 클래스 C_j는, $C_i \supset C_j$이다.
④③의 조건을 만족하고 새로운 클래스 C_k가 $C_k \subset C_j$를 만족하면 클래스 C_i는 최상위 클래스이고, 클래스 C_j는 중간 노드(node) 클래스이며, 클래스 C_k는 하위 클래스이다. 즉, $C_i \supset C_j \supset C_k$의 조건을 만족한다.
⑤클래스 $\{(C_k \subset C_i) \wedge (C_k \subset C_j)\}$를 만족하면 클래스 C_k는 클래스 C_i와 C_j로부터 다중 상속(multiple inheritance) 관계이다.

[정의 4] 클래스간의 상속 계층 관계에는 일반화(generalization)와 세분화(specialization)관계로 나뉘어진다.

①일반화는 상위 클래스 C_i의 도메인(domain)이 하위 클래스 C_j의 부분 집합이라면 C_i는 C_j의 일반화라 한다. 즉, $(C_i \supset C_j)$의 조건을 만족한다.
②세분화는 일반화의 역으로 $(C_j \supset C_i)$가 성립하면 하위 클래스 C_j는 클래스 C_i의 세분화라고 한다. 즉, $(C_j \supset C_i)$의 조건을 만족한다.

[정의 5] 클래스의 복합 관계는 다음의 조건들을 만족하여야 한다.

①객체 O_i와 O_j에 대하여 O_i가 O_j의 부분이라 할 때 필요 충분 조건은 $O_i \bullet O_j = O_j$이다. 그리고 객체의 합성(composition)은 그 객체의 부분 집합이다. 즉, $CO(O_i) = \{O_j \mid O_j \bullet O_i = O_i\}$의 조건을 만족한다.
②$CO(O_i)$의 원소가 오르지 하나이고 그 원소가 O_i일 때 이를 단순 객체(simple object)라 한다.
③$CO(O_i)$의 원소가 여러 개 일 때 이를 복합 객체(composite object)라 한다.
④어떠한 특정 객체와 그 부속품들간의 관계에는 주 종속의 관계를 이루므로 객체 O_i가 O_j의 부품이고 O_j가 임의의 객체 O_k의 부품이라 할 때 객체 O_i는 객체 O_k의 부품이다.
⑤임의의 두 개의 클래스 $C_i(dm)$와 $C_j(dm)$에 속하는 어떠한 객체 O_i와 O_j 사이에 ①, ③, ④의 조건을 만족할 때 클래스 $C_i(dm)$와 $C_j(dm)$는 복합관계를 이룬다.

여기에서 수식 표기 정의는, "•": 연결 연산자(concatenation), O_i: 객체 i, C_i: 클래스 i, $C_i(d)$: 클래스 C_i의 도메인(domain = 데이타 멤버(속성) + 멤버 함수(메소드)), $CO(O_i)$: O_i의 합성(composition), $C_i(dm)$: C_i의 데이타 멤버를 나타낸다.

[정의 3]의 ①, ②조건에서 멤버 함수의 중복에 대한 다형성 문제는 동일한 이름을 갖는 멤버 함수인지 타입을 비교하여 측정한다. 그리고 여기에서는 참조 관계인 협동 관계는 정의하지는 않았지만 클래스 사용자와 클래스 공급자와의 관계를 의미한다. 참조 관계는 클래스의 구현시에 멤버 함수에 대한 호출로 표현되며, 복합 관계와 시스템 구현 결과에서 동일한 형식을 취하기도 한다. 그렇지만 시스템 분석 단계에서 이들을 구별하여 모델링 하여야 한다. 즉, 두 클래스의 관계를 참조 관계로 해석할 것인가 또는 복합 관계로 해석할 것인가는 구현 관점보다 시스템의 본질적 관점을 이용하여 구별하여야 한다. 예를들어 클래스 $C_j(dm)$가 클래스 $C_i(dm)$의 부분일 때 이 관계는 복합 관계로 모델링 되어야 한다. 이때 복합 관계가 갖는 의미는 클래스 $C_j(dm)$의 객체 O_j가 클래스 $C_i(dm)$의 객체 O_i의 전체 생명주기를 통하여 O_i의 데이타 멤버가 존재하여야 함을 의미한다. 그러나 참조 관계의 경우에는 O_j가 O_i의 생명 주기 동안 항상 존재할 필요가 없다. 즉, 객체 O_i가 객체 O_j의 서비스를 필요로 할 경우에만 객체 O_j가 생성되면 된다.

3.3.2 클래스 유사성 계산 방법

일반적으로 클래스들의 유사성(similarity)이란 클래스 사이에서 동일한 데이타 멤버 및 멤버 함수를 측정한 값을 말한다. 이러한 측정 값은 상속 계층 구조 설정과 구현시 코드의 재사용 측면을 판단하게 된다.

본 논문에서는 기존의 연구 결과를 기준으로 한다 [1, 13].

[정의 6] 클래스들간의 유사성(SIM)에서는 다음의 조건들을 만족하여야 한다.

$SIM(C_{ij})$ = 임의의 클래스들간의 유사성을 측정하는 척도로 즉, 임의의 클래스 C_i와 C_j사이의 함수 $SIM(C_{ij}) = f(C_i, C_j)$로 정의하며, 다음의 조건을 만족하여야 한다.

① $0 \le SIM(C_{ij}) \le 1$,

② $SIM(C_{ij}) = SIM(C_{ji})$

③ $C_i = C_j \Rightarrow SIM(C_{ij}) = 1$

[정의 7] 클래스의 데이타 멤버 유사성

$$SIM_{dm}(A, B) = \frac{(dm(A \cap B))^2}{dm(A) \times dm(B)}$$

[정의 8] 클래스의 멤버 함수 유사성

$$SIM_{dmf}(A, B) = \frac{(dmf(A \cap B))^2}{dmf(A) \times dmf(B)}$$

[정의 9] 클래스들의 유사성

$$SIM(A, B) = P \times SIM_{dm}(A, B) + (1 - P) \times SIM_{dmf}(A, B)$$

[정의 7]에서 $dm(A)$는 클래스 A를 구성하는 데이타 멤버의 개수이고, $dm(B)$는 클래스 B를 구성하는 데이타 멤버의 개수를, 그리고 $dm(A \cap B)$는 두 클래스 내에서 공통적인 기능을 갖는 데이타 멤버의 개수를 나타낸다. [정의 8]에서는 데이타 멤버 함수(dmf)를 나타낸다. [정의 9]의 P는 클래스 A, B의 데이타 멤버들의 총 개수와 멤버 함수들의 총 개수를 합한 값에 대한 데이타 멤버들의 총 개수의 비율이다. 그러나 설계 관점에서 이러한 요구 사항들을 모두 만족하도록 유사성을 측정한다는 것은 매우 어렵다. 따라서 본 논문에서는 클래스들의 유사성은 같은 이름을 갖는 데이타 멤버, 멤버 함수인지를 비교하여 측정한다. 그리고 아래의 기준을 적용한다.

① 두 함수의 이름과 함수가 반환하는 값의 타입(type)이 같아야 한다.

② 함수의 모든 인자들의 타입이 같아야 한다.

③ 함수가 참조하는 전역 변수의 타입이 같아야 한다.

④ 지역 변수들의 타입이 같아야 한다.

이와 같이 클래스의 클래스 재사용과 상속구조 설계를 위한 유사성 계산방법을 살펴 보았다. 다음은 복합 구조 설계를 위한 다음의 식을 정의한다.

[정의 10] 클래스의 복합 관계 유사성

$SIMco(A, B) \Rightarrow A(dm_b) = B$

$SIMco(B, A) \Rightarrow B(dm_a) = A$

[정의 10]에서 $A(dm_b)$는 클래스 A를 구성하는 데이타 멤버 b이고, $B(dm_a)$는 클래스 B를 구성하는 데이타 멤버 a를 나타낸다. 즉, 클래스 A의 데이타 멤버 b와 클래스 B, 그리고 클래스 B의 데이타 멤버 a와 클래스 A가 같을 때 성립한다.

3.3.3 클래스 계층 구조 설계 방법

본 논문에서는 앞에서 제시한 3.3.1 형식정의, 3.3.2 클래스 유사성 계산 방법을 적용하여 분석 단계에서부터 상향식 방법으로 클래스 계층 구조를 이루는 방법을 제시한다. 여기에서 임의의 클래스 Y로부터 클래스 X의 상속 관계는 class Y inherits X, 클래스 X로부터 클래스 Y로의 복합 관계는 class X의 임의의 데이타 멤버 n composites Y로 정의할 수 있다.

그러나 구문적 접근 방법을 사용하여 고품질의 설계 결과를 낸다는 것은 어려운 일이며, 특히 문제 정의(definition of the problem)로부터 유용한 추상(abstraction)을 생성하는 것은 매우 어려운 일이다. 즉, 유용한 추상은 문제의 정의로부터 독립적이고 직관적인 요소를 적절하게 뽑아내는 것으로부터 나오기 때문에 여러번 계속적으로 재 설계를 할 수 있는 끈기를 가진 프로그래머에 의하여 만들어진다. 따라서 추상화의 이점과 필요성은 객체 지향 시스템의 개발에 많은 영향을 주고 있지만 아직까지 정형적(formal)인 방법은 존재하고 있지 않다.

그리고 본 논문에서 상속 관계는 상위 클래스로부터 하위 클래스로의 상속은 부분 상속이 아닌 전체 상속을 원칙으로 하였다. 그 이유는 부분 상속은 효율적인 방법이 아니기 때문이다.

〈객체 지향 클래스 계층 구조 설계 알고리즘〉

1. 문제 설명서에서 업무 영역을 서브 영역으로 분할한다.
2. 업무 서브 영역에서 클래스를 식별한다.
3. 식별된 클래스를 분류 한다. (내부 클래스, 외부 클래스, 인터페이스 클래스)

if [(클래스(C_i))] = [(데이타 저장소에 해당하는 클래스(Ci))] then

  −내부 클래스(internal class)

  if [(클래스(C_i))] = [(타 시스템, 타 모듈에 관계된 클래스(C_i))] then

    −인터페이스 클래스(interface class)

  else

    −외부 클래스(enternal class)

  end if

4. 내부 클래스와 인터페이스 클래스의 성질을 정의한다.
   −데이타 멤버명
   −데이타 멤버 타입
   −데이타 멤버 크기(size)
   −멤버 함수명
   −멤버 함수 타입
   −멤버 함수 발생 건수 및 주기
5. 클래스 간의 단순 관계성(reference-of)을 정의한다.
6. 클래스들의 유사성을 측정한다.

▶ 클래스 데이타 멤버 유사성

$$-\mathrm{SIM}_{dm}(A, B) = \frac{(dm(A \cap B))^2}{dm(A) \times dm(B)}$$

▶ 클래스 멤버 함수 유사성

$$-\mathrm{SIM}_{dmf}(A, B) = \frac{(dmf(A \cap B))^2}{dmf(A) \times dmf(B)}$$

▶ 클래스들의 유사성

$-\mathrm{SIM}(A, B) = P \times \mathrm{SIM}_{dm}(A, B) + (1-P) \times \mathrm{SIM}_{dmf}(A, B)$

▶ 클래스 복합(composite) 관계 유사성

$-\mathrm{SIMco}(A, B) => \mathrm{dm}_b(A) = B$

$-\mathrm{SIMco}(B, A) => \mathrm{dm}_a(B) = A$

7. 클래스 계층 구조도를 display 한다.
8. 클래스들의 복잡도를 측정한다.

▶ 클래스 수준에서 복잡도 측정 방법

−데이타 멤버 개수

−멤버 함수 개수

−멤버 함수의 평균 서비스 수행 횟수

▶ 모듈 수준(클래스 계층 구조)에서 복잡도 측정 방법

$$\text{c-struc} = \frac{1}{n} * \sum_{i=1}^{n} d(n-i, n-0),\ n = \text{class 개수},$$

$d(n-i, n-0)$ = 상속 및 복합 구조로 연결된 root node에서 다른 i-th nodes간의 거리.

−클래스 개수

−클래스 계층구조 깊이

−클래스 계층구조 너비

−서브 클래스 개수

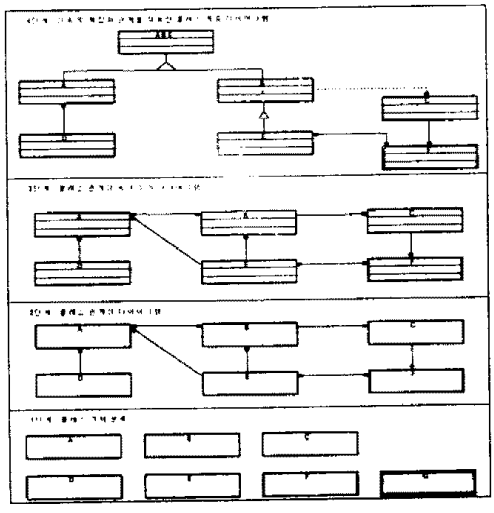
−데이타 멤버 평균개수

−멤버함수 평균개수

9. 추가된 클래스를 정의한다(추상화 클래스)
10. 클래스 간의 관계성(reference-of, is-a, is-part-of)을 재정의 및 등록한다.
11. 클래스 계층 구조 정보(모듈 정보) 등록 및 display 한다.

이와 같은 알고리즘의 세부적인 항목들을 살펴보면 1단계 문제 설명서에서 업무 영역을 서브 영역으로 분할에서 분석 단계에서 하향식 방법으로 시스템을 분해하여 모듈 단위로 분해되어지는 과정을 알 수 있다. 이러한 방법은 대형 소프트웨어 개발 및 분산 컴퓨팅 시스템에서 유용하게 사용되어진다. 3, 4단계에서는 2단계에서 식별한 클래스중에서 내부 클래스와 인터페이스 클래스의 성질를 정의하고 있다. 여기에서 클래스 데이타 멤버의 크기, 멤버함수 발생 건수, 주기등은 분산 환경에서 객체 할당 정책에 유용한 정보이다. 5단계에서는 이와 같은 정의된 클래스 성질을 바탕으로 클래스간의 단순 참조 관계를 정의하며, 6단계에서는 클래스간의 유사성을 측정한후 상속관계 및 복합 관계를 정의한다. 그러나 여기에서 완료되는 것이 아니라 8단계의 클래스 복잡도를 고려하여 추가된 클래스 정의 및 클래스간의 관계성을 재정의한다. 이러한 과정은 반복적인 작업을 필요로 한다.

(그림 4)에서는 클래스 계층 구조를 형성하는 설계 과정이다. 1단계 클래스 분류에서 내부 클래스, 외부 클래스, 인터페이스 클래스로 분류하였으며, 여기에서 외부 클래스 G는 클래스 관계성에서 제외된다. 2단계와 3단계에서는 클래스들의 관계성을 나타내고 있지만 상속 및 복합 계층구조는 아직까지는 알 수 없다.



(그림 4) 클래스 계층 구조 설계 과정
(Fig. 4) Processing of Class Hierarchy Structure Design

본 논문에서는 3단계의 정보를 래피드 프로토타입핑 방법으로 시뮬레이션하여 관계성을 설정하여 클래스 계층 구조를 설계한다. 클래스 ABE는 클래스 A, B, E의 공통된 데이타 멤버 및 멤버 함수를 가지고 있으며, 클래스 E는 클래스 ABE와 B로부터 상속을 받는다. 클래스 C는 클래스 B의 데이타 멤버 c와 복합 관계를 나타내고 있다. 그리고 클래스 A와 D, E와 F, C와 F는 단순 참조 관계를 나타낸다. 여기에서 새롭게 추가된 클래스와 재구성인 경우를 제외하고는 2단계 및 3단계에서 표시되지 않은 관계는 4단계에서 클래스간의 어떠한 관계성도 나타내지 못한다.

이 과정에서 분석 클래스들을 설계 클래스로 변형할 때 프로토타입을 위한 조율(tuning)항목으로는 〈표 1〉과 같다[3, 4, 6].

〈표 1〉 클래스 계층 구조 설계를 위한 조율 항목
〈Table 1〉 Tuning Item for Class Hierarichy Structure Design

항목	내용
분할	· 대부분의 분석 클래스들은 설계 클래스와 일대일 대응되지만 어떤 분석 클래스는 설계 단계에서 다중 클래스들로 분해 될 수 있다.
상속	· 부분 상속은 효율적인 방법이 아니므로 위임(delegation)을 통한 집합화(aggregation)를 고려해야 한다. · 클래스들간에 상속의 정의에서 "is-a" 관계가 아닌 두 클래스를 억지로 부모 자식 클래스로 정의하게 됨으로써 야기되는 혼돈을 피하여야 한다. · 추상적이고 응집력(coherent)있는 클래스 계층을 설계하여 어느 특정한 구현 사항에 변화가 없어야 한다.
할당	· 관계에 참여하지 못한 클래스들의 데이타 멤버와 함수들을 연관된 클래스에 할당시킨다.
표식	· 클래스의 멤버 함수는 입력 파라메타와 출력 값을 포함한 완벽한 표식(signature)을 갖는다.
인터페이스	· 만일 하나의 클래스가 여러개의 클래스와 교류하고 있을 때, 각 클래스들에 대한 인터페이스는 동일하여야 한다.

이상과 같이 래피드 프로토타입핑 기법을 사용한 객체 지향 클래스 계층 구조 설계 방법을 살펴보았다. 이 방법은 식별된 클래스로부터 클래스 유사성을 측정하여 상향식으로 클래스 계층 구조를 형성하는 방법이다. 본 논문에 제시한 방법은 분산 환경의 대형 소프트웨어 개발에서 유용하게 사용할 수 있다. 그 이유는 분석 단계에서부터 시스템→영역→서브시스템으로 분해하여 모듈 생성 과정을 유도 할 수 있으며, 또한 객체 및 클래스의 관계성 정의를 바탕으로 모듈, 영역, 시스템에서 공통으로 필요로하는 클래스를 인터페이스 클래스로 설정하여 그 구분을 명확하게 표현 할 수 있기 때문이다. 그리고 설계 관점에서 여러 수준으로 복잡도를 측정하여 설계의 최적화를 할 수 있다.

4. 결 론

본 논문에서는 Rumbaugh의 OMT방법을 확장하여 래피드 프로토타입핑 기법을 사용한 객체 지향 클래스 계층 구조 설계 방법을 제시하였다. 이 방법은 분석 단계로부터 대형 소프트웨어 개발의 모듈 생성과정을 자연스럽게 유도하고, 클래스간의 관계성을 분석한후 클래스들의 유사성을 측정하여 상속 및 복합 계층 구조를 형성한다.

본 논문에서 제시하는 방법의 장점은 다음과 같다.

첫째, 이미 정의된 클래스 정보를 바탕으로 프로토타입 시스템으로 뿐만 아니라 실제 시스템에 최종 전환할 수 있기 때문에 원가 측면에서 효율적이다.

둘째, 모듈 분해 과정을 자연스럽게 유도하므로 대형 시스템 개발과 객체 지향 분산 컴퓨팅 환경의 시스템 개발에 유용하게 사용 할 수 있다.

셋째, 전문적인 기술을 필요로하는 시스템 분석가가 아니라도 객체 지향 기술의 핵심요소라 할 수 있는 상속 및 복합 계층 구조를 유도 할 수 있다.

그러나 이 방법은 상속 및 복합 계층 구조의 형성 과정에서 보다 정형화된 제시 모델이 되지 못하는 단점이 있다. 따라서, 객체 지향 기술의 전문지식을 가진 설계자의 추가적인 지식과 반복적인 시뮬레이션 과정이 필요하다.

앞으로의 연구 과제로는, 상속 및 복합 계층 구조를 자동으로 설계할 수 있는 보다 정형화된 CASE 시스템 개발에 대한 연구가 필요하다.

참 고 문 헌

[1] Amos Tversky, "Features of Similarity", Psychological Review, Vol. 84, No. 4, pp. 327-352, Jul, 1977.

[2] B. W. Bohem, T. E. Gray, and T. Seewaldt, "Prototyping Versus Specifying:A Multiproject Experiment", IEEE Trans on Soft. Eng., Vol. SE-10, No. 3, May, pp. 290-303, 1984.

[3] Gustav Pomberger & Gunther Blaschek, "Object-Orientation and Prototyping in Software Engineering", Prentice-Hall, Inc, pp. 32-34, 49-59, 1996.

[4] Gunther Blaschek, "Object-Oriented Programming with prototypes", Sprinferverlag Berlin Heidelberg, pp. 92-117, 1994.

[5] H. Gomaa, "The Impact of Rapid Prototyping on Specifying User Requirement", ACM SIGSOFT SEN Notes, Vol. 8, No. 2, Apr, pp. 17-28, 1983.

[6] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design", Prentice-Hall, Inc. pp. 84-49, 148-183, 227-229, 260-274, 1991.

[7] L. Scharer, "The Prototyping Alternative", ITT Programming, Vol. 1, No. 1, pp. 34-43, 1983.

[8] M. Alavi, "An Assessment of the Prototyping Approach to Information Systems Development", Communication of the ACM, Jun, pp. 556-563, 1984.

[9] R. Prieto-Diaz, P. Freeman, "Classifying Software for Reusability", IEEE Software, pp. 6-16, Jan, 1987.

[10] T. Taylor, and T. A. Standish, "Initial Thoughts on Rapid Prototyping Techniques", ACM SIGSOFT SEN Notes, Vol. 7, No. 5, Dec, pp. 160-166, 1982.

[11] Y. Wand, R. Weber, "An Ontological Model of an Information System", IEEE Trans. on SE., Vol. 16, No. 11, pp 1282-1292, Nov. 1990.

[12] W. Kim, "Object-Oriented Concept, Databases, and Applications", ACM Press, Inc, pp. 6-9, 262-266, 1989.

[13] 김갑수, 신영길, "소프트웨어 재사용을 위한 C++ 클래스 계층 구조 변형 방법", 한국정보과학회 논문지, 제22권 제1호, pp. 88-93, 1995.

[14] 최영근, 허계범, "객체 지향 소프트웨어 공학", 도서출판 한국 실리콘, pp. 237-282, 1995.

[15] 허계범, 최영근, "객체 지향 설계를 위한 모듈 분해 방법", 한국정보처리학회 논문지, Vol. 2, No 3, pp. 299-313, 1995.

허 계 범

1989년 경기대학교 응용통계학과(학사)

1993년 광운대학교 전산대학원 전자계산학과(이학석사)

1988년~1995년 (주) 경인양행 전산부 과장

1995년~현재 광운대학교 전자계산학과 박사과정

관심분야: 객체 지향 소프트웨어 공학, 객체 지향 분산 컴퓨팅

최 영 근

1980년 서울대학교 수학교육과(학사)

1982년 서울대학교 계산통계학과(이학석사)

1989년 서울대학교 계산통계학과(이학박사)

1996년~1997년 미시간 주립 대학교 객원교수

1983년~현재 광운대학교 전자계산학과 교수

1997년~현재 광운대학교 전자계산소 소장

관심분야: 프로그래밍 언어, 병렬 컴퓨터, 객체 지향 분산 컴퓨팅