論文 90-27-8-10

객체지향 소프트웨어설계법에 관한 연구

(A Study on Object-Oriented Software Design Methodology)

宋 榮 宰*

(Young Jae Song)

要 約

자연언어 문장에 나타나는 어구를 소프트웨어 요소로 대응하는 객체지향 소프트웨어 설계법에 관한 연구가 진행되고 있으나 아직 설계지침이 명확히 정립되어 있지 않다.

본 논문에서는, 객체의 클래스를 표시하는 명사 뿐만 아니라 문장을 세분화함으로서 좋은 품질의 모듈구조를 얻을 수 있었으며, 객체와 그들 사이의 상호의존도에 주목하는 객체지향 소프트웨어 설계법을 제시하였다.

리프트의 제어문제를 예를들어 평가하였다.

Abstract

Though the studies on object-oriented software design which software component corresponds to phrases appearing in sentences of natural language have been going on, the design methodology has not been clearly set up yet.

In this paper, the module structure of good quality was found not only by nouns that indicate class of objects but also by subdividing sentences. I also suggested object-oriented methodology observing the interdependency between objects and their relationships, and assessed the control problem of lift as an example.

I. 서 론

1980년대에 들어와서 컴퓨터의 고속화, 대용량화 때문에 소프트웨어의 규모는 점점 커지고, 이용분야는 아울러 확대되고 있어서 소프트웨어 개발의 수요는 점점 증대되고 있다. 그러나 소프트웨어 개발자의 수는 수요에 따르지 못하고 있으며, 경험이 부족한 사람들이 대규모 소프트웨어를 개발한다는 상황이 되고 있다. 거기에다 기능이나 성능의 추가요구가 확대됨에 따라 보수작업은 증가하고, 소프트웨어의 위기라고 까지 말하고 있다. 또한 FORTRAN, COBOL 등의 절차적인 언어를 중심으로 하는 소프트웨어의 생산활동에 위기가 닥쳐와서 소프트웨어공학, 소프트웨어방법론이 대두되게 되었다. 이와같은 소프트웨어 위기에 대한 어프로치로서 생각한 것이 구조적 프로그래밍[1], 계층, 단계적 세부화[2], 추상화,[3] 모듈[4] 등의 중요한 개념이 계속하여 확립되었다.

이것에 부수적으로 언어의 개량도 이루어져서 구조(structure), 클래스(class), 추상데이타형(abstract data type)등의 기능을 갖춘 언어가 다수 출현하였다.

한편, 실무분야에서도 복합설계(composite design)[5] 데이타플로우에 의한 방법,[6] 데이타구조에 의한 방법(jackson 방법등)[7]이 고안되어서 설계합리화에 큰 역할을 하였다. 또한, 데이타에 대한 추상화 방법으

*正會員, 慶熙大學校 電子計算工學科
(Dept. of Computer Eng., Kyunghee Univ.)
接受日字 : 1990年 6月 7日

로서 추상데이타형[8]이 확립되게 되었다.

추상데이타형이란 추상화 하고저하는 데이타형의 내부표현이나 데이타에 대한 액세스방법, 조작을 실현하고 있는 절차적인 세부사항등, 그 데이타에 관한 자세한 정보를 모두 모듈내에 감추어서 캡슐화(encapsulation)하여 사용하는 것을 가리킨다. 따라서 그 모듈의 이용자는 공개된 조각들을 사양에 따라서 사용하는 것이 되며, 그 내부표현은 생각하지 않아도 된다. 즉 추상화시킨 형태로 그 데이타형을 택할 수가 있다. 이와같은 상황하에서 1980년대에 들어서면서 라이프싸이클(생명주기)에 대한 논의가 활발하게 되어 라이프싸이클 유해설[9]까지 이르게 되었다.

여기에 대한 추상데이타형의 확장으로서 객체지향(object-oriented)[10]을 생각하게 되었다. 이것은 소프트웨어의 유지, 보수에 필요로하는 비용를 줄이고, 수정, 개조가 쉬운 소프트웨어를 설계하기 위한 설계방법으로서 객체지향 설계법(object-oriented design)[11]이 제창되었다. 1980년대가 구조적 프로그래밍이 중심과제였던 것과 마찬가지로 1990년대 이후는 객체지향 프로그래밍이 중심과제가 되리라 생각된다.

한편, 객체지향 사상에 의한 언어나 시스템들도 몇년 사이에 적극적으로 연구되어 상품화되고 있다. 그것은 Smalltalk-80[12], Objective-C,[13] C++[14] Trellis/Owl,[15] CLOS[16] 등이다.

그리고, 객체지향이라는 개념이 프로그래밍언어 뿐만 아니라 멀티메디아 데이타베이스, 전문가 시스템, 자연언어 이해, 컴퓨터그래픽스 등 넓은 분야에서 연구 개발되고 있다.

객체지향에서는 세계가 독립된 「실체」의 집합이며, 「실체」와 「실체」가 서로 통신을 하고 있다고 생각할 수 있다. 예를들면, 인간의 세계에서 생각해보면 많은 인간이 서로 회화를 하면서 움직이고 있다는 관점이다. 그 인간에 해당하는 「실체」를 객체(object)라고 부른다. 그리고, 인간이 내는 소리를 메시지(message)라 부른다. 객체에 대한 명령은 메시지 형태만이 요구할 수가 있다. 따라서 조작(operations)의 호출은 메시지·패싱이라는 형태로 통일되게 된다.

이 객체와 메시지라는 개념은 인공지능 프로그래밍의 Actor모델[17,18] 이론에 의하여 생긴 것으로, 독립된 정보를 실행하는 단위인 객체들이 서로 메시지를 주고 받음으로서 그 객체 외부에 있는 다른 객체가 가지는 처리기능을 이용하는 것이다.

객체를 단위로 하는 소프트웨어의 부품을 생각할 때의 장점은, 객체라고 하는 개념이 특정한 응용분야에 의존하지 않는 자연적인 현실세계의 모델이 되며, 이미 존재하는 객체의 필요한 성질만을 계승(inheritance)[19]하여 새로운 객체가 쉽게 생성할 수 있다는 점이다.

그러나, 객체지향언어를 사용하여 응용개발이 많이 이루어 지고 있으나, 「무엇을, 어떻게 하여 객체로 하는가」, 「좋은 객체는 어떤 것인가」, 「그것을 표시할 때의 방법은」등과 같은 문제도 있다. 즉, 이것은 객체지향언어를 사용하여 개발하는 경우의 소프트웨어 설계법이 명확하지 않기 때문이다. 객체지향의 장점을 최대한 살리기 위해서는 그 장점을 가지는 개념, 기능을 어떻게 적용하면 좋은지의 지침을 부여할 필요가 있다.

따라서, 본 논문에서는 이문제를 해결하기 위하여 추상화의 하나의 방법인 객체지향 개념에 의한 새로운 소프트웨어 설계법을 제시하고저 한다.

본 논문에서 기술하는 방법은 컴퓨터에 의한 완전 자동화를 목표로 하는 것이 아니라, 구조적설계법, Jackson법, Booch의 객체지향 설계법과 같은 종래의 소프트웨어 설계법과 마찬가지로 인간이 계통적으로 소프트웨어를 설계하기 위한 하나의 방법을 제시하기 위함이다.

Ⅱ. 객체지향 설계법

1. 개발환경

프로그래밍언어와 소프트웨어 설계법과는 상호 관련이 있다. 종래의 COBOL, FORTRAN등의 절차형 언어에는 소프트웨어 도구로서 플로우챠트, 서브루우틴을 단위로 하는 기능 분할법이 주로 사용되어 왔다.

한편, 소프트웨어 개발에 있어서 소프트웨어의 신뢰성향상및 생산성향상은 중요한 과제이다. 이와같은 과제를 해결하기 위해 최근에는 개발프로세스 및 프로덕트의 형식적인 기술에 대하여 많이 논의가 되고 있으며, 그리고 객체지향 설계법(object-oriented design), 실시간 구조화 시스템 분석법, 잭슨지향 설계법등 여러 개발방법이 제안되고 있다. 그 중에서도 특히 개발의 체계성 및 지원환경의 보급이 확대됨에 따라서 객체지향 설계법이 주목되고 있다.

여기에서, 객체지향 프로그래밍[20,21]과 객체지향 설계법[11,22]에서도 여러가지 개념이 적용된다. 예를들면, 객체, 클래스, 메시지, 계승등.

한편 최근에는 개발해야 될 소프트웨어의 대규모화에 따라서 개발 도중에 설계 변경 요구에 대처할 것, 프로우트 타이핑에 의하여 개발할 것 등이 강하게 요청되고 있다. 이런 요청에 대응하기 위한서는

소프트웨어의 세부사항의 결정을 가능한 늦추어서 시스템 개발을 진행할 필요가 있다. 그러나 종래의 객체지향 설계법은 이러한 점에 있어서 아직 미흡하다.

객체지향 설계법에서는 일반적으로 자연언어문 가운데에 나타나는 명사는 데이타나 객체와 같은 실체를, 동사는 실체에 부가되는 조작을 표시하며, 이와 같은 대응은 형식사양에 이용한다. 그러나 이것만으로는 문장 가운데에 출현하는 많은 어구로 부터 실체와 조작이 되는 어구를 선택하여 소프트웨어 컴포넌트로서 어떻게 정리 할까는 불충분하다.

객체지향 개념에 의한 소프트웨어를 개발하기 위해서는 다음 세가지의 필요성을 중시하게 된다.

첫째는, 현재의 소프트웨어 개발 방법은 top-down 분해에 의한 라이프싸이클에 따른 것이 주류를 이루고 있다. 그러나, 사용자의 불만, 개발이나 보수에 대한 막대한 비용을 보더라도 현재의 소프트웨어 개발법이 반드시 좋다고만은 할 수 없다. 따라서, 현재의 문제점을 극복하기 위한 어프로치를 검토할 필요가 있다.

둘째는, 1980년대 후반에 들어와서 객체지향적이라고 하는 추상화방법이 관심을 불러 일으키고 있다. 그러나, 항간에는 객체지향 언어의 개발에 주목하고 있으며, 실제로 사용하는 것은 아직 부족한 실정이다. 따라서, 객체지향의 사상을 잘 사용하여 실용화할 수 있는 소프트웨어 개발법을 생각할 필요가 있다.

셋째는, 소프트웨어 개발에서도 비용삭감, 신뢰성 향상을 위해 여러가지 노력을 해오고 있으나 프로그래밍언어의 개발, 프로그래밍 방법론 검토, 소프트웨어 방법론의 검토, 개발·지원도구의 정비이다.

일반적으로 객체지향 설계 방법론은 설계순서, 설계의 표현법, 설계의 양부 판단기준과 같은 지침이 필요하다.

한편, 기존의 절차형에 비해 객체지향의 특징적인 개념, 기능 및 장점은 다음과 같은 것이 있다고 생각된다. 그것은, 문제를 해결하기 위한 기능을 컴퓨터에 대상(mapping)하기가 쉽다. 데이타 추상화나 기능을 국소화함에 의하여 기능 변경시에 따른 모듈에의 영향을 감소할 수 있으며, 재이용 하기가 쉽다. 비슷한 성질의 객체를 생성하기가 쉽다. 크래스를 계층적으로 정리함에 의해 관리와 재이용이 쉽게 이루어 진다. 그리고, 자주적으로 활성화하는 객체를 모델로 생각하기 때문에 동작을 자연스럽게 표현할 수 있다는 점이다.

따라서, 객체지향 설계법의 이상적인 방법은 이들 개념이나 기능을 살린 설계법이 중요하게 된다.

2. 개발순서

객체지향에 의한 소프트웨어 개발에서는 현실세계의 실체를 객체에 대응시킨다. 그러므로 보통 설계자가 가지는 현실세계의 이미지에 맞는 내부구조를 가진 소프트웨어가 작성되게 된다. 각 객체는 절차와 데이타를 내부가 가지는 모듈로서 실현되며, 데이타의 추상화와 정보은폐를 쉽게 실현할 수 있게 된다. 또한, 크래스의 기구를 사용하여 소프트웨어를 재사용할 수 있다.

문헌[11,23-25]에 기술하고 있는 객체지향 설계법의 개발순서는 다음과 같다.

1) 문제해결을 위한 비형식적인 전략의 정의
2) 객체, 조작, 속성의 추출
3) 각 객체의 인터페이스 결정
4) 각 객체의 세부적인 내부설계
5) 전체의 통합및 실현

그러나, 대규모적인 소프트웨어를 성공시키기 위하여는 다음 사항들도 만족되어야 한다.

즉, 처음에는 결정 가능하면서 필요한 것만을 결정한다. 단, 자세한 부분에 대해서는 나중에 결정하는 것이 좋다. 이 방침에 따라 작업을 진행하면 개발 도중에 있을 설계 변경에 대해서도 충분히 대처할 수 있다.

또한, 톱-다운 설계에서는 어느 단계에서 위로 부터의 요구가 아래로 부터의 요구와 조정이 필요하게 된다. 소프트웨어 설계법은 이를 위한 지침도 부여해야 한다. 앞에서 기술한 기존의 객체지향 설계에서는 객체의 내부설계를 하기 전에 셋째 단계에서 인터페이스 결정을 하기 때문에 넷째단계에서 대폭적으로 후퇴하게 될 염려가 있다.

그 다음은, 소프트웨어 개발에 사용되는 모델은 소프트웨어 라이프싸이클 전반에 대하여 유효함이 바람직하다.

이 논문에서는 이들을 만족하는 객체지향 개념에 의한 소프트웨어 개발법을 제안한다.

Ⅲ. 제안하는 객체지향 설계법

본 논문에서 제안하는 객체지향 소프트웨어 설계법을 다음에 기술한다. 개발해야 될 소프트웨어의 사양은 자연언어로 기술된 문제에 적용된다.

[단계 1] 객체와 그 속성의 추출… 여기에서 객체추출이란 객체를 결정하는 공정을 말하는 것으로서, 대상이 되는 문제 영역에서의 객체를 선정하여 그 모든 역할을 이해하는 것이 필요하다. 그것은 주어진 문장에서 명사와 명사구에 대응하여 하나의 객체명을 도입한다. 이 단계는 다음 단계와 함께 종래의

모듈분할 설계법에 있어서의 서브시스템, 기능계층에의 분할, 즉, 모듈에의 분할에 해당된다.

[단계 2] 객체 사이의 조작의 추출… 이 단계에서는 분할된 각각의 객체 또는 객체의 크래스의 조작을 결정한다. 즉, 객체가 가지는 메소드의 이름을 결정한다. 하나의 객체에 주목하여 그 자신에 의한 조작과 다른 객체에 의한 조작을 명확히 한다. 이 조작의 추출을 대상이 되는 객체 전체에 대하여 수행한다. 조작의 추출기준은 동사가 후보가 된다.

또한, 소프트웨어 설계서에서 하나의 문장 가운데 여러개의 용어를 포함하고 있는 경우, 예를들면 「A를 B에 C로 한다」라는 문이 있는 경우에 조작 「C로한다」가 객체A에 대한 조작인지, B에 대한 조작인지를 결정할 필요가 있다. 이와같은 경우에는 A, B가운데에서 추상도가 높은 쪽의 객체에 대한 조작으로 한다. 여기에서 추상도를 파악할 때 소프트웨어 전체를 서비스 기능단위로 분할하여 여러사람이 병행하여 요구분석을 하는 경우에는 똑같은 정보가 서로 다른 시각으로 볼 수 있기 때문에 객체관련도에 의한 전체적인 상을 파악하여 부자연스러운 관련이 없도록 조정한다.

[단계 3] 객체의 통합… 유사한 개념이 별도의 객체로서 추출되는 일이 있다. 이런 경우에는 개념적으로 하나의 용어로 표시할 수 있는 것은 본질적으로 똑같은 것이라 생각되므로 하나의 객체로 정리한다.

[단계 4] 단일 문장으로 변환…다음에 따라서 주어진 문장을 단문으로 바꾼다. 식별하기 위하여 각 단문에는 일련번호를 붙인다.

1. 수동태문은 능동태문으로 바꾼다.

2. 문 a는 b와 c로 이루어진다(a, b, c는 객체명)와 같이 구성되어 있는 경우에는 객체에 기호 e를 붙인다.

3. 조건부의 동작을 표시하는 단문은 다음 형식으로 기술한다.

(조건 : 조건이 성립할 때의 동작)

4. 단문i에서 동작주어 역할을 하는 객체에 실선을 긋고, 동작대상에 점선을 긋는다.

[단계 5] 크래스계층을 결정… 각 객체가 가지는 변수를 광역적, 국소적으로 크래스 사이에 계층을 둘것인지 아닌지를 판단한다. 이것은 객체 사이의 관계를 확인하는 일이다.

[단계 6] 객체그래프 구성… Booch[12]는 Booch 도표라고 하는 것으로 객체사이의 의존도를 표시 하였으나, 본 논문에서는 G=(V, E)로 구성되는 객체그래프를 사용하여 상호 관련도를 표시한다. 여기에서, 마디의 집합V는 객체명, 가지의 집합 E는 다음에 따라 정한다.

● 단문 i가 객체 j를 포함할 때에 객체 j가 주어진 경우에는 다른 객체로 향하여 화살표를 붙인다.

● 단문 i가 객체 j를 포함하는 것 끼리는 화살표가 없는 가지를 붙인다. 여기에서 가지가 있다는 것은 객체 사이에 관련이 있음을 나타낸다.

[단계 7] 각 모듈의 실현… 각 모듈의 내부객체, 외부객체, 매개객체를 기술한다. 그다음에 객체그래프를 사용하여 외부객체가 주어가 되어 있는 단문을 골라서 그것에 대응하는 모듈을 실현한다.

[단계 8] 인터페이스 확립… 위의 단계에서 모듈 분할을 하였으므로 개개 객체의 조작의 외부사양을 작성한다. 이것은 전 단계에서의 객체에 의한 모델을 구체화하는 것이다.

[단계 9] 객체의 실장…이 단계에서는 각 객체에 대응되는 표현을 선택하여 전 단계에서의 인터페이스를 실장한다. 이 과정에서는 분해와 합성이 함께 이루어 진다.

Ⅳ. 적용 예

Ⅲ장에서 제안한 객체지향 설계법에 따라서 리프트 제어문제[27]를 예를 들어 적용하기로 한다. 주어진 비형식적인 사양은 다음과 같다.

*n층을 가지는 빌딩에 리프트시스템이 설치되어 있다. 그 리프트와 제어기구는 제조자에 의하여 제공된다. 이들 내부기구는 주어진다고 가정한다. 각 층 사이에 리프트를 이동시키는 논리는 다음과 같은 제한사항에 따른다. 리프트는 각 층을 위하여 하나씩 버튼들의 집합을 가진다. 이들은 관련된 층을 방문하기 위하여 버튼들을 눌렀을 때 불이 들어온다. 관련된 층에 리프트가 도착하면 그 버튼은 해제된다. 각 층에는 2개의 버튼을 가지는데(맨 아래층과 맨윗층을 제외하고) 그것은 상승버튼과 하강버튼이다. 이들 버튼들은 눌려졌을 경우에 불이 켜지며, 리프트가 그 층에 도착하거나, 원하는 방향으로 이동하거나 요구가 없을 경우는 불빛이 해제된다. 서비스를 결정하기 위한 알고리듬은 양쪽의 요구를 위해 대기시간이 최소이어야 한다. 리프트가 서비스요구를 받지 않으면, 그 문이 닫힌 목적지에서 그대로 기다린다. 각 층으로 부터의 모든 서비스요구는 각 층이 똑같은 우선순위로 서비스 되어야 한다. 각 층을 위한 모든 요구는 진행방향으로 순서적으로 서비스된다. 리프트는 비상버튼을 가지며, 눌려졌을 경우는 바깥의 관리인에게 경보음이 전달되며, 그 리프트는 서비스금지 상태가 된다. *

주어진 자연언어 사양에서 객체지향 소프트웨어를 설계하기 위해서는 우선, 무엇을 객체로 할까를 결정하고, 그들 객체사이의 관계를 해석하여야 한다.

1단계(객체와 속성의 추출) : 대상이 되는 문제영역에 존재하는 실체를 열거한다. 이것은 현실세계의 자연언어를 표현하였을 경우의 명사의 리스트라고 생각해도 된다. 그러나 모든 명사가 객체에 대응되는 것은 아니다. 이 리스트를 검토하여 그 가운데에서 중복된 것, 문제영역(현실세계)을 기술하는 데에 부적절한 것은 삭제한다. 추출된 객체명을 다음에 기술한다.

제어기구, 리프트, 층, 상승버튼, 하강버튼, 비상버튼, 관리인, 문, 리프트버튼, 손님

모두 10개의 객체가 추출되었다. 이 과정에서는 모두 19개의 명사가 추출되었으나 9개의 명사는 파기하였다. 그러나 이 문장 가운데에서 전혀 출현하지 않는 단어인 「손님」은 상식적으로 판단하여 도입하였다. 버튼객체도 그 기능에 따라 구별하여 리프트버튼, 상승버튼, 하강버튼, 비상버튼으로 하였다.

또한, 경우에 따라서는 객체와 조작의 모든 경우에 속성을 관련시키는 것이 필요하다. 속성은 객체나 조작의 특성을 이해하는데 도움이 된다. 객체와 속성과의 관련도의 예를 그림 1에 표시한다.

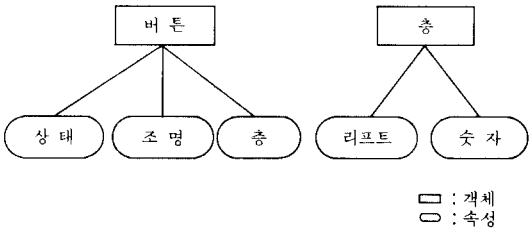

그림 1. 객체와 속성과의 관련도
Fig. 1. Relationship diagram of object and their attributes.

4단계(단일 문장으로 변환) :

1. n층을 가지는 빌딩에 리프트가 설치되어 있다.
2. 리프트와 제어기구는 제조자가 제공한다.
3. 각 층 사이에 리프트를 이동시키는 제어기구는 다음과 같은 제한사항에 따른다.
4. 각 층을 위하여 하나의 버튼을 가진다.
5. 손님이 관련된 층을 방문하기 위하여 버튼을 누른다.
6. 〔손님〕이 버튼을 누르면 불이 들어온다.
7. 리프트가 그 층에 도착하면 보튼의 불빛이 해제된다.
8. 각 층에는 상승버튼과 하강버튼을 가진다.
9. 〔제어기구〕리프트를 기다리는 시간은 최소이어야 한다.
10. 서비스 요구가 없으면 리프트는 최종목적지에에서 문을 닫고 기다린다.
11. 〔제어기구〕 리프트는 진행방향으로 순서적으로 처리된다.
12. 〔손님〕이 비상버튼을 눌렀을 경우에는 관리인에 경보음이 전달된다.

여기에서 밑줄의 실선은 동작주, 점선은 동작대상이다.

5단계(크래스계층의 선택) : 예제에서의 프로우트타입의 크래스계층의 예를 그림 2에 표시한다.

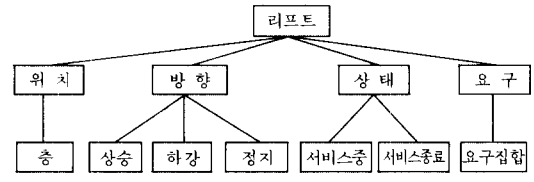

그림 2. 리프트의 크래스계층
Fig. 2. Class hierarchy of ltft system.

2, 3단계(조작의 추출및 객체의 통합) : 앞단계에서 얻어진 객체에 대하여 객체의 조작을 열거한다. 이 조작은 다른 객체로 부터의 메시지에 의해 의뢰함으로서 기동되고, 그 조작하는 어떤 순서관계가 있다. 주어진 사양문에서의 조작은 다음과 같다.

해제, 이동, 누름, 열림, 닫힘, 출발, 도착, 알림, 요구

조작을 추출하는 과정에서, 누름과 서비스요구, 이동과 방문은 같은 의미이므로 각각 후자를 삭제하였다.

6단계(객체그래프 구성) : 주어진 예제의 객체그래프를 그림 3에 표시한다.

7단계(모듈실현) : 소프트웨어설계의 품질에 대한 또 다른 중요한 측면은 모듈화, 즉 완전한 프로그램을 만들기 위해 결합되는 프로그램 구성요소(모듈)에 대한 명세이다. 객체를 그 자체가 다른 구성요소(비공개자료)에 접속할 수 있는 프로그램의 구성요소로서 정의한다. 리프트버튼의 모듈구조 예를 아래에 표시한다.

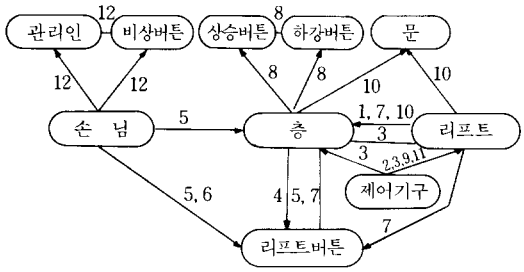

그림 3. 객체 그래프
Fig. 3. Object graphs.

〈리프트버튼 모듈〉
〈외부객체〉
손님, 층
〈내부객체〉
상승버튼, 하강버튼, 비상버튼
〈매개객체〉
제어기구, 관리인
〈단문기술〉
4. 각 층을 위하여 하나의 버튼을 가진다.
5. 손님이 관련된 층을 방문하기 위하여 버튼을 누른다.
6. 손님이 버튼을 누르면 불이 들어온다.
7. 리프트가 그 층에 도착하면 보턴의 불빛이 해제된다.

8. 9단계(인터페이스확립및 객체의 실장) : 그림3의 객체그래프를 참조하여 객체기술의 인터페이스부에 메시지와 그 메시지로 기동되는 동작을 기술한다. 리프트보턴의 상세한 사양 예를 예제 1에 기술한다.

〈예제1〉
Object 리프트버튼 Is
〈외부객체〉
손님, 층
〈내부객체〉
상승버튼, 하강버튼, 비상버튼
〈매개객체〉
제어기구, 관리인
〈Interface〉
Input Is
Mes누름(X : Boolean) ==>출발;
Mes해제(X : Boolean) ==>도착;
End Input
Output Is
Obj리프트<== Mes이동
End Output
〈actstruct〉 -- 동작구조부
*([누름;|해제;])
〈var〉
Obj리프트 : Object;
〈action〉 -- 동작부
4. 각 층을 위하여 하나의 버튼을 가진다.
5. 손님이 관련된 층을 방문하기 위하여 버튼을 누른다.
6. 손님이 버튼을 누르면 불이 들어온다.
7. 리프트가 그 층에 도착하면 버튼의 불빛이 해제된다.
〈error〉
누름 : ==>|관리인에게 연락|
End 리프트버튼

여기에서 Object정의부는 외부객체, 내부객체, 매개객체, 인터페이스부, 동작구조부, 변수선언부, 동작부, 에러처리부로 구성되어 있다.

이 Object정의부를 기술함에 의해 실제로 동작하는 실체의 형태가 결정되게 된다. 이 정의된 객체에 대하여 메시지로서 New를 보냄으로서 그 실체가 생성되게 된다.

인터페이스부는 객체에 대한 입출력을 기술한다.

입력부에는 도착하는 메시지와 그 메시지에 의해 기동되는 동작의 이름을 기술한다. 만일 메시지에 매개변수가 있는 경우에는 그 가인수명과 데이타형을 기술한다. 예를들면,

MesA(a:Integer)==>Act1;과 같이 기술한다.

출력부에는 출력으로서 승인되는 메시지와 그 송신처를 기술한다. 만일 메시지에 매개변수가 있는 경우에는 그 매개변수가 취하는 데이타형을 기술한다.

예를들면,

X<==MesB(Integer, Boolean);

과 같이 기술한다.

동작구조부에는 동작순서의 구조를 기술한다. 이 순서제약은 순차, 반복, 선택의 조합이며, 다음 방법을 사용하여 표시한다.

A;B;C; 순차
*(A;) 반복
[A;|B;|C;]선택

변수선언부에는 기본 데이타형이나 사용자가 정의

한 데이타형을 사용하여 객체의 상태를 보존하기 위한 변수선언을 기술한다. 여기에서 선언한 변수는 객체의 실체가 생성될 때 할당되어 실체가 소멸할 때까지 존재한다. 예를들면,

ObjA:Object;X, Y:Integer;

와 같이 기술한다.

동작부는 외부객체가 동작주가 되어있는 단문을 골라서 열거한다.

에러처리부는 동작구조부의 지정에 모순되거나 실행되지 않을 경우의 처리내용과 규칙의 형태를 기술한다. 예를들면, 조작A를 실행할려고 해도 실행되지 않을 경우의 처리는

A:==>|문장열거|

와 같이 기술한다.

V. 결 론

자연언어로 기술된 사양으로 부터 실행 가능한 사양 또는 프로그램을 얻기 위하여 객체지향 소프트웨어설계에 관한 연구가 진행되고 있으나 아직 설계지침이 명확하게 확립되어 있지 않으며, 여러 면에서 개량의 여지가 있다. 이를 위해 객체지향의 장점을 최대한 살리기 위한 설계지침이 필요하다.

따라서, 본 논문에서는 자연언어 문장에 출현하는 어구를 소프트웨어 요소로 대응하도록 체계적으로 정리함으로서 수정의 영향을 최대한 줄일 수 있으며, 재사용을 가능하게 하는 객체지향 소프트웨어설계법을 제시하였다.

본 논문에서 제안한 객체지향 소프트웨어 설계법에서는,

소프트웨어를 객체모델로서 취급하는 데에 있어서 기존의 방법과 달리 객체와 크래스를 표시하는 명사뿐만 아니라 문장을 세분화하여 모듈구조를 설계하는 방법을 택하므로서 보다 좋은 품질의 형식적사양을 구축할 수 있으며, 최상위레벨의 모듈구조를 얻을 수 있다.

또한, 객체와 그들 사이의 상호의존도에 주목하는 추상도가 높은 객체지향 설계법을 제시하였으며, 객체사이의 관련을 표시하는데에 Booch[11]는 Booch도표로 표시하였으나, 본 논문에서는 G=(V, E)로 구성리는 객체그래프를 사용하여 상호관련을 표시하였다.

기존의 설계법에서는 객체의 내부설계전에 인터페이스를 결정하기 때문에 내부설계 단계에서 대폭적으로 후퇴할 염려가 있으나, 본 연구에서는 인터페이스 결정을 뒤로 하므로 사양변화에 대한 영향을 줄일 수 있다.

그리고, 다채로운 자연언어의 표현에 대처하여 세부적인 구조를 표현할 수 있다.

본 논문에서 제안된 설계법에 따라 개발을 하면 객체지향개념을 효과적으로 활용한 소프트웨어개발이 가능하게 된다.

여기에서는 계층구조및 병행객체의 실현은 본 방법의 다음 단계에 위치하는 작업이므로 고려치 않았다.

앞으로는 제안된 개발법을 지원하기 위한 도구를 작성할 필요가 있다. 현재 IDL, Lex, Yace을 사용하여 C++언어로 변환되는 처리계를 작성중에 있음.

參 考 文 獻

[1] O. J. Dahl, Structured Programming, Academic Press (1972).

[2] N. Wirth, Program Development by Stepwise Refinement, *Comm. ACM*, vol. 14, no. 4, pp. 221-227 (April 1971).

[3] M. Shaw, Abstraction Techniques in Modern Programming Languages, *IEEE Software*, vol. 1, no. 4, pp. 10-26 (Oct. 1984).

[4] D. L Parnas, On the Criteria to be used in Decomposing Systems into Modules, *Comm. ACM*, vol. 15, no. 12, pp. 1053-1058, (Dec. 1972).

[5] G. J. Myers, , Reliable Software through Composite Design, Mason/Charter (1975).

[6] B. Alabiso, Transformation of Data Flow Analysis Models to Object-Oriented Design, *Proc. OOPSLA'88*, pp. 335-353 (Sept. 1988).

?7] M. A. Jackson, System Development, Prentice-Hall (1983).

[8] B. H. Liskov and S.N. Zilles, Programming with Abstract Data Types, *ACM SIGPLAN Notices*, vol. 9, no. 4, pp. 50-59 (1974).

[9] P. D. McCracken and M.A. Jackson, Life Cycle Concept Considered Harmful, *ACM Soft. Engr. Notes*, vol. 7, no. 2, pp. 29-32 (April 1982).

[10] K. A. Jamsa, Object Oriented vs. Structured Design, *ACM Soft. Eng. Notes*, vol. 9, no.1, pp. 43-49 (Jan. 1984).

[11] G. Booch, Object-Oriented Development, *IEEE Trans. Soft. Eng.* , vol. SE 12, no. 2, pp. 211-221 (Feb. 1986.).

[12] A. Goldberg, Smalltalk-80, The Language and its Implementation, Addison-Wesley (1983).

[13] B. J. Cox, Object Oriented Programming, An Evolutionary Approach, Addison-Wesley (1986).
[14] B. Stroustrup, The C++ Programming Language, Addison-Wesley (1986).
[15] C. Schaffert et. al., An Introduction to Trellis/Owl *Proc. OOPSLA' 86*, pp. 9-16 (1986).
[16] L. G. Pemichiel, The Common Lisp Object System: An Overview, *LNCS Object-Oriented Programming*, pp. 151-170 (1988).
[17] C. Hewitt, Viewing Control Structures at Patterns of Passing Message, *Journal of Artificial Intelligence*, vol. 8, pp. 323-364 (1977).
[18] C. Hewitt, An Universal Modular Actor Formalism for Artificial Intelligence, *Proc. of Int. Conf. on AI*, pp. 235-245 (1973).
[19] W. Cook, A Denotational Semantics of Inheritance and its Correctness, *Proc. ACM OOPSLA'89*, pp. 433-443 (1989).
[20] E. White, Object-Oriented Programming, *BYTE*, vol. 11, no. 8, pp. 137-233 (Aug. 1986).
[21] R. Simonian, True Object-Oriented Programming in Ada, *Journal of Object-Oriented Programming*, vol. 1, no. 4, pp. 14-21 (Nov/Dec. 1988).
[22] E. Seidewitz, Towards a General Object-Oriented Software Development Methodology, *SIGAda Ada Letters*. pp. 54-67 (July/Aug. 1987).
[23] M. Temte, Object Oriented Design and Ballistics Software, *ACM Ada Letters*, vol. 4, no. 3, pp. 25-36 (Nov./Dec. 1984).
[24] R. F. Sincovec, Modular Software Construction and Object-Oriented Design Using Ada, *Journal of Pascal/Ada & Modula-2*, pp. 29-34, (Marhc/April, 1984).
[25] R. J. Abbott, Program Design by Informal English Descriptions, *Comm. ACM*, vol. 26, no. 11, pp. 882-894 (1983).
[26] B. Myers, Reusability: The Case for Object-Oriented Design, *IEEE Software*, pp. 50-64 (March 1987).
[27] Problem Set, *Proc. of 4th International Workshop on Software Specifiction and Design*, pp. ix-x (1987)
[28] T. W. Page et. al., An Object-Oriented Modelling Environment, *CSD-890012, UCIA*, pp. 17 (1989).
[29] R. Agrawal, ODE: The Language and the Data Model, *Proc. ACM SIGMOD' 89*, pp. 36-45 (1989).
[30] K. Shannon and T. Maroney, IDL User's Manual, Softlab Document no. 8, Computer Science Dept., Univ. of North Carolina at Chapel Hell (May, 1986).
[31] M. E. Lesk, Lex-A Lexical Analyzer Generator, UNIX Programmer's Mannual, Bell Lab (July, 1978).
[32] S. C. Johnson, YACC-Yet Another Compiler-Compiler UNIX Programmer's Mannual, Bell Lab. (July, 1978).

著 者 紹 介

宋 榮 宰(正會員)

1947年 4月 20日生. 1969年 인하대학교 전기공학과(전자전공)졸업. 1976年 日本慶應義塾大學院(電算學)졸업. 1979年 명지대학교 대학원 졸업(공학박사). 1971年～1973年 日本 Toyo seiko(주) 연구원. 1980年～1983年 공업진흥청 공업표준심의위원. 1981年～1982年 경희대학교 교무차장. 1976年～1984年 경희대학교 전자공학과 조교수, 부교수. 1984年～현재 경희대학교 전자공학과 교수. 1984年～1988年 경희대학교 전자계산소장. 1982年～1983年 University of Maryland 연구교수. 1986年～1987年 대한전자공학회 전자계산연구회 전문위원장. 1987年～1989年 전국 대학 전산소장협의회 부회장. 1989年～1990年 일본 Keio University 객원 교수. 주관심분야는 Object-Oriented Programming & System. Software Engineering, Database System Design.