절차중심 시스템으로부터 객체추출 방법 및 도구개발에 관한 연구

김 정 종[†] · 손 창 민[††]

요 약

객체지향 패러다임을 적용한 시스템으로 재개발한다면 재사용을 통한 소프트웨어 생산성 향상을 이룰 수 있고, 유지보수 비용을 절감할 수 있게 된다.

절차중심의 시스템을 객체지향 패러다임을 적용한 형태로 변환시킬 때 특히 코드에서 자동 혹은 반자동으로 객체를 추출하고자 하는 기법들이 많이 연구되고 있다. 그러나 이들 방법에서는 개념적인 객체의 추출이 용이하지 않다. 따라서 추출되는 객체의 개념적 무결성이 떨어지는 문제가 발생한다.

본 논문에서는 절차중심으로 개발된 프로그램을 객체지향 시스템으로 바꾸려고 할 때 발생되는 문제점중의 하나인 객체추출 방법 및 툴에 대해 논한다. 원시 코드 및 설계 복구 정보를 입력받은 후 먼저 실세계의 응용영역에 대한 객체 모델링을 이용하여 객체를 추출한다. 그리고 코드의 기능, 코드의 전역변수 및 인터페이스를 분석하여 객체를 추출하므로써 객체의 개념적 무결성을 높이며, 클래스 계층구조의 구축을 용이하게 한다.

A Study on the Method and Tool Development for Extracting Objects from Procedure-Oriented System

Jung-Jong Kim[†] · Chang-Min Son[††]

ABSTRACT

If there is redeveloping into the system applying the object-oriented paradigm, productivity improvement of software through reuse would be accomplished and maintenance cost be reduced. When a procedure-oriented system is transformed to a type applying the object-oriented paradigm, various techniques are studied to extract objects from source code automatically or semi-automatically. However, it is not easy to extract conceptual objects when those techniques are applied. This problem entails another problem which drops the conceptual integrity of the extracted objects. In this paper, we suggest an object extraction method and tool development to resolve the problem occurring at the time when the programs developed through procedure-oriented is transformed to an object-oriented system. The suggested method allow to extract the desired objects using object modeling for various application domains of the real world given source code and design recovery information. During the extraction process, functionality and global variables of the source code as well as its interface are rigorously analyzed. This process can enhance the conceptual integrity of the objects and make easy to construct class hierarchies.

† 종신회원 : 경남대학교 컴퓨터공학과
†† 정 회 원 : 경북실업전문대학 전자계산과
논문접수 : 1997년 8월 13일, 심사완료 : 1998년 1월 30일

1. 서 론

하드웨어의 발전으로 응용분야가 넓어지면서 소프트웨어에 대한 수요가 급증했으나 소프트웨어의 생산성이 급격히 떨어지면서 소프트웨어 위기에 직면하게 되었다.

이를 극복하기 위해 소프트웨어의 재사용을 생각하게 되었으나 이는 다른 공학에서의 재사용과는 달리 소프트웨어가 가지고 있는 특성으로 인해 재사용이 힘들고, 개발자의 응용영역(application domain)에 대한 전문지식의 부족으로 인한 재사용 자원의 확보 및 활용이 용이하지 않게 되었다. 이로 인해 재사용이 활성화되지 못하고 부분적인 응용분야에 한정되어 사용되어졌다. 그러나 객체지향 패러다임이 많은 사람의 관심을 끌게 되고 보편화되면서 소프트웨어의 생산성 향상을 위한 하나의 수단으로 주목받고 있다.

또한 소프트웨어는 제작된 후 유지보수에 많은 손길을 필요로 하게 된다. 개발된 시스템들은 사용자들의 새로운 요구, 새로운 기술의 출현, 새로운 환경에 맞도록 수정 보완되어야 하며 이는 막대한 비용을 필요로 한다. 시간이 흐를수록 수정의 양도 많아지고 수정하는데도 많은 어려움이 따르게 된다. 소프트웨어가 개발된 후 시간이 지남에 따라 소프트웨어에 가해지는 변화에 따른 유지보수 비용은 전체 개발 노력의 70% 이상을 차지하며 앞으로 점점 증가할 것이다[1].

그리고 개발 당시의 소프트웨어에 적용되었던 응용영역의 제반 기술 및 소프트웨어 기술보다 뛰어난 새로운 기술이 개발됨에 따라 기존의 시스템을 새로이 만들도록 요구되어진다. 또한 소프트웨어의 적용범위가 넓어짐에 따라 다양한 소프트웨어가 생산되고 있으나 기존의 소프트웨어 대부분은 빈약한 설계구조와 충분한 문서화가 되어 있지 않고 과거에 운영한 시스템을 기반으로 설계되었기 때문에 새로운 시스템으로의 이식성에 문제가 발생할 수 있다[2].

따라서 구조적 방법을 적용하여 작성된 절차적 시스템을 최근의 객체지향 패러다임을 적용한 형태로 변환하고자 하는 경향이 있다. 이것은 현재 널리 사용되고 있는 구조적 방법은 사용자 요구사항 변경에 따른 융통성 있는 대응이 어렵고, 확장성이 적으며, 분석에서 설계로의 전환시 정보가 손실되는 등의 문제점을 지니고 있기 때문이다[3]. 그러나 객체지향 방법은 문제를 이해하기 쉽고, 개발자와 사용자간의 의사소통이 용이하고, 요구사항 변동에 대한 융통성 있는 대응이 가능하다는 장점을 가지고 있다. 또한 객체지향 방법의 빼놓을 수 없는 장점중의 하나는 유지보수의 용이성과 비용절감에 있다.

그리고 소프트웨어 역공학은 기존의 소프트웨어를 분석하여 정보나 문서를 추출하므로서 프로그램의 이해를 높일 뿐만 아니라 프로그램의 재사용 비율을 높이게 된다[4]. 이에 따라 소프트웨어의 유지보수 및 개발에 따른 비용을 줄일 수 있게 되었다.

이러한 흐름에 맞추어 시스템을 재공학할 때 기존 시스템을 유지보수 및 재사용에서 기존의 방식들 보다 유리한 객체지향 패러다임을 적용한 시스템으로 재개발한다면 이는 재사용을 통한 소프트웨어 생산성 향상을 이룰 수 있고, 소프트웨어 유지보수 비용을 절감할 수 있으며, 시스템에 대한 새로운 요구를 수용할 수 있게 되는 등 많은 장점을 지니게 된다.

본 논문에서는 기존의 절차중심으로 개발된 기존 프로그램을 객체지향 시스템으로 바꾸려고 할 때 발생되는 문제점중의 하나인 객체의 추출에 대한 방법론에 촛점을 맞추어 기존의 프로그램에서 보다 양질의 객체를 추출하기 위한 방법론을 제시하고자 한다.

2장에서는 이와 관련한 연구들에 대해 알아보고, 3장에서는 객체추출 방법론을 제시하며, 4장에서는 제시한 방법론을 실제 적용한 사례를 보이며, 5장에서는 결론을 기술한다.

2. 관련연구

COREM[5][6]에서는 설계복구, 응용 모델링, 객체매핑, 소스 코드의 변경과 같은 4단계로 기존 시스템을 객체지향 구조로 변환하고 있다. 이 방법은 다른 하위수준의 설계문서(구조도, 자료흐름도), E-R 다이아그램을 절차 프로그램의 소스 코드로부터 얻은 후 객체지향 응용모델을 작성하고 기존 절차적인 프로그램의 요구분석을 기본으로 또다른 객체모델을 생성하고 이 전단계에서 작성된 객체모델과 대응처리를 수행한다. 대응이 되는 경우 목표 객체모델은 원하는 객체지향구조를 나타내며 이것을 기반으로 필요한 소스 코드의 변경이 이루어진다. 소스 코드 변경단계는 소스 코드의 구문적 수준에서 프로그램 변

환과정을 완결시킨다.

[7]에서 제시한 방법은 E-R 다이아그램, 자료사전, 미니 명세서로부터 개체관계 테이블을 만들고 자료흐름도, 자료사전, 미니 명세서로부터 자료흐름 테이블을 만들어 객체모델 생성에 이용한다. 이는 구조적 분석모델에서 OMT의 객체모델로 사상시키는데 필요하며, 객체모델에서 필요한 모든 요소를 쉽게 빠짐없이 식별할 수 있게 한다. 다른 사상방법은 객체 식별 후 속성 식별 그리고 연산 식별의 순이나 이 방법은 동시에 3가지의 식별이 가능하다.

[8]에서 제시한 방법은 정적객체모델 추출알고리즘을 이용하여 자료사전과 자료흐름도로부터 정적객체모델을 추출하고, 자료흐름도와 미니 명세서로부터 동적객체모델을 추출한 후 이들을 융합하여 객체모델을 생성한다. 여기서 정적객체는 오퍼레이션을 갖지않고 데이타 역할만하는 객체이고, 동적객체는 오퍼레이션과 데이타구조를 모두 갖는 객체이다.

Monyhan[9]에서 제시된 연구는 PASCAL 프로그램을 Ada로 변환하기 위한 언어변환기를 구현하였으며, 이 방법에서는 필요한 설계복구를 수행하지 않고 라인 중심의 소스 코드 번역기로서 기존 코드의 각 줄에 대해 다른 언어의 소스 코드를 생성한다.

Byrne[10]는 Fortran에서 Ada로 역공학하기 위한 방법을 제시하고 있다. 역공학을 위해 기존 코드로부터 설계복구를 수행하고 난 후에 변환을 처리한다.

CIAS[11]는 C 프로그램으로부터 객체 및 객체간의 관계에 관한 정보를 추출하여 관계 데이타 베이스에 저장한다. 저장된 정보는 사용자의 요청에 따라 정보 viewer에 의해 테이블 형식으로 보여진다. 이 시스템은 리스트 표현구조를 사용하므로서 초보자의 프로그램 인식이 어렵고 제어흐름이나 자료흐름에 관한 정보가 미흡하다.

OODesigner[12]는 OMT의 객체모델 표기법을 지원하여 C++코드를 생성하는 순공학도구와 C++코드로부터 OMT객체모델을 생성하는 역공학 도구로 구성되어 있다. 이 시스템은 순공학 도구와 역공학 도구를 함께 구성하므로서 재구조화 작업과 오류수정이 용이하다. 그러나 객체지향 기법만을 지원하므로서 절차지향 기법으로 구성된 시스템의 경우는 적용하기가 힘들다.

3. 객체추출 시스템 설계 및 구현

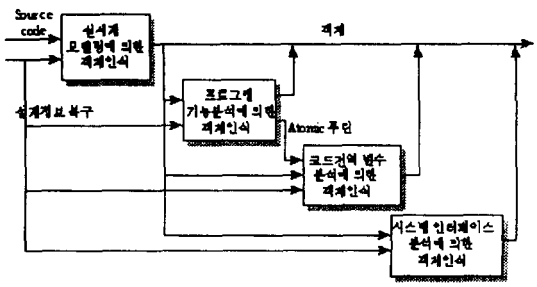
3.1 객체추출 방법론

기존의 객체지향 언어를 이용한 객체지향 시스템으로의 재공학 방법론들은 주로 코드를 이용하려 했다[13][14]. 특히 자동 혹은 반자동으로 코드에서 객체를 추출하고자 하는 기법들이 최근 많이 연구되고 있었다[13]. 그러나 이들 방법에서는 개념적인 객체의 추출이 용이하지 않았다[15]. 그래서 본 논문에서는 추출되는 객체의 개념적 무결성(conceptual integrity)를 높이기 위해 다음과 같이 객체를 추출하고자 한다.

1) 먼저 실세계의 응용영역에 대한 객체 모델링(object modeling)을 이용하여 객체를 추출 한다.
2) 그 다음, 코드의 기능, 코드의 전역변수, 인터페이스를 분석하여 객체를 추출한다.

이 방법의 특징은 먼저 코드에만 의존하지 않고 실세계의 응용영역에 대한 객체모델링을 한다는 것이다. 둘째, 코드를 통해 객체를 추출하는 방법의 다양화이다. 셋째, 실세계의 응용영역에 대한 객체모델링만으로 객체를 추출할 때 속성 및 연산을 찾기 위해 많은 수고가 필요했으나 이들을 코드에서 찾게하므로서 속성 및 연산의 추출을 용이하게 하였다.

본 논문에서는 기존 시스템에서 객체를 추출하기 위해 (그림 1)과 같은 프로세스에 따라 시행한다.



(그림 1) 객체 추출 프로세스
(Fig. 1) Object Extraction Process

3.1.1 실세계의 응용영역에 대한 객체모델링을 통한 객체추출

본 절에서는 실세계의 응용영역을 객체모델링하여 개념적인 객체를 추출하는 방법을 기술한다. 일반적

으로 실세계의 응용영역에서 객체를 추출하기 위한 방법은 자연 언어 기술(Natural Language Description)을 이용한 문법적인 접근방법 및 구조적 분석(Structured Analysis)의 결과물, 즉 DFD와 E-R 다이아그램 등을 이용하는 방법 등 크게 두가지로 분류된다[16]. 이들 방법에는 Booch[16], Rumbaugh[17], Coad & Yourdon[18], Shlaer & Mellor[19]등 여러가지가 있다. 실세계의 응용영역에서 객체를 추출하기 위하여 본 논문에서는 Coad & Yourdon의 객체모델링을 사용하며 프로세스는 다음과 같다.

1)객체 및 클래스 식별

먼저 객체와 클래스를 식별한다. 객체를 구별하기 위해 각 객체를 객체명, 속성, 객체 자신이 갖고 있는 기능을 명세하기 위한 연산의 세 부분으로 구성한다. 그리고 공통된 성질을 갖고 있는 객체들을 표현하기 위한 본형(template)으로 클래스를 정의한다. 이때 객체와 클래스를 식별하는데 사용되는 항목은 문제 영역, 텍스트, 그림 등 시스템의 구조, 다른 시스템과 연결되는 외부 터미네이터(external terminator), 장치, 사건(event), 행해지는 역할, 처리 절차, 위치, 조직의 단위 등이다.

2)구조 식별

시스템내의 객체와 클래스를 추출, 식별한 다음, 문제 영역에서 발생하는 복잡성을 다루기 위하여 객체 및 클래스 사이의 관계를 관계도로 표현한다.

이들 구조에 나타나는 관계는 크게 분류화(classification) 개념인 일반-특수 구조와 집단화(assembly) 개념인 전체-부분 구조로 나누어진다. 따라서 일반-특수 구조는 객체 및 클래스를 일반화와 특수화시키는 것으로 이 구조상의 클래스간에는 "is-a", 또는 "is-kind-of" 관계성이 성립하고, 전체-부분 구조는 "has-a", 또는 "is-part-of" 관계성이 성립한다.

3)주제 정의

복잡한 대형 시스템의 구조를 이해하기 위해서 시스템을 그룹별로 분할하여 하나의 추상 계층을 만든다. 주제(subject)는 한번에 취급하기 적당한 분량의 객체들로 분할하는 단위로써 객체들이 주제로 분할되고, 주제들 사이에 메시지를 표현하여 하나의 완전한 시스템을 형성한다. 이렇게 하기 위해서는 전체-부분 구조에서 가장 높은 레벨의 객체를 조사하여 이것을 후보 주제 영역으로 정의한다.

소규모 시스템에서는 주제 정의가 반드시 필요한 것이 아니므로 생략할 수도 있다.

4)속성 및 관계 정의

모델링될 객체에 대한 자세한 사항을 추가 함으로써 클래스와 객체 의미를 명확하게 하며, 클래스와 객체에는 은닉될 정보와 서비스에 의해 다루어질 정보를 기술한다. 식별된 속성 및 관계는 객체 구조 속에서 어느 곳에 위치해야 될지 결정하는 일과 객체들의 연관성을 표현하는 것에 의해 결정된다.

5)서비스 정의

클래스의 연산(operation)을 정의하는 것으로 서비스는 메시지 수령에 따라 행해져야 하는데, 서비스는 전체 모델이 완벽할 수 있도록 기술되어야 한다.

서비스 정의는 문제 영역에서 시스템이 수행해야 할 역할을 명세화하는 작업으로서, 역할들을 작업 순서에 따라 객체의 동적 행위를 묘사한 객체 생명주기를 정의한 후, 시스템의 상태와 외부 사건에 대한 응답으로 구성된 상태-사건, 응답(state-event, response) 형태로 서비스를 정의한다.

3.1.2 프로그램의 기능을 이용한 객체추출

본 절에서는 프로그램의 기능을 분석하여 객체를 추출하는 방법을 기술한다.

소스 코드가 기능 위주로 되어 있으므로 이들을 잘 분석하면 객체를 찾을 수 있다. 즉 시스템내에 존재하는 객체의 연산이 되도록 코드를 기능적 응집도(Functional Cohesion)를 갖는 부분(단위 루틴, Atomic routine)으로 분할할 수 있다[14]. 또한 일반적으로 전역변수는 객체의 속성이 될 가능성이 많이 있으므로[13] 객체의 속성은 단위 루틴의 행위의 대상이 되는 단위 루틴내의 변수가 될 가능성이 많다. 그러나 이들 단위 루틴이 어느 객체의 연산이 되는가를 판단하는 것이 문제가 된다.

이 문제를 해결하기 위해 단위 루틴을 일정한 템플릿에 맞추어 단위 루틴의 행위를 기입하고, 이 템플릿의 내용을 이용해서 단위 루틴을 클러스터링(Clus-

tering)하여 개체 추상화(Entity Abstraction)를 지니는 객체를 추출한다. 즉 코드의 기능을 분석하여 객체를 추출하는 방법은 첫째 단위 루틴을 인식하고, 둘째 템플릿에 기술하며, 셋째 이를 이용하여 객체를 추출하는 프로세스로 행해진다.

이들 각 부분에 대한 자세한 내용은 다음과 같다.

(1) 단위 루틴의 추출

속성이 될 수 있는 많은 데이타들은 코드상에서 변수(전역변수와 함수 매개변수)의 형태로 나타난다. 그러므로 이들 변수를 분석해 보면 속성을 찾을 수 있다. 또한 이들에 행해지는 행위를 중심으로 기능적 응집도를 이루도록 코드에서 단위 루틴을 인식하고 이를 연산으로 정의할 수 있다[14].

소스 코드에서 단위 루틴을 인식할 때 다음의 가이드라인을 참고한다.

1) 단위 루틴은 기능적 응집도를 지녀야 한다.

단위 루틴은 객체의 연산이 되도록 정의한다. 그러나 한 단위 루틴(연산)이 여러 개의 기능을 수행하면 이 연산은 좋은 연산이 되지 못한다. 그러므로 하나의 단위 루틴은 하나의 기능을 수행하기 위하여 꼭 필요한 내용만 포함될 수 있도록 코드를 분할하여 단위 루틴이 기능적 응집도가 되도록 하여야 한다.

2) 전역변수 혹은 함수 매개변수(Function Call Parameter)를 중심으로 단위 루틴을 인식하는 것이 좋다.

전역변수와 서브루틴의 매개변수 중 일부는 객체의 속성이 될 가능성이 많이 있다[13]. 그러므로 전역변수와 함수 매개변수에 대해 어떤 조작이 일어나는 코드상의 특정부분을 중심으로 하나의 단위 루틴을 만들면 이 단위 루틴의 기능은 객체에 대한 연산으로 정의될 수 있다.

3) 핵심적인 변수를 중심으로 단위 루틴을 만들어야 한다.

각 서브루틴 혹은 블록내에는 여러 개의 변수들이 존재한다. 그러나 이들 변수는 실제 기능의 중심이 되는 변수와 그 변수에 대한 기능을 수행하기 위한 보조적인 데이타를 가지고 있는 변수들로 구성되어진다. 그러므로 단위 루틴을 만들 때 실제 기능의 중심이 되는 변수에 대한 동작이 일어나는 코드상의 범위를 단위 루틴으로 만든다.

예를 들면, 아래의 코드에는 info->data라는 자료구조의 항목과 record_size라는 변수가 존재한다. 이 중 info->data는 이 루틴의 동작의 중심이 되고 record_size는 info->data를 화일에서 읽기 위한 하나의 보조 데이타 역할을 한다.

이때 상위수준의 연산을 추출하기 위해 info->data를 중심으로 단위 루틴을 인식하는 것이 좋다.

```
if (1 != fread(info->data, record_size, 1, fp)) {
  if (ferror(fp)) {
    printf("error reading data file\n");
    fclose(fp);
    return 0;
  }
}
```

4) 단위 루틴은 가능하면 상위수준 연산(High Level Operation)이 되도록 한다.

전역변수 및 함수 매개변수를 중심으로 단위 루틴을 만들 때 각 변수에 대한 하나의 문장에 국한되어 단위 루틴을 만들어서는 안된다. 경우에 따라 한 문장을 단위 루틴으로 만들 필요가 있을 수도 있으나 너무 변수에 대한 단위 동작에 국한되어 단위 루틴을 만들면 그 단위 루틴의 내용이 단위 기능적 내용이 되어 저수준의 연산이 생성된다. 단위 루틴을 만들 때 한 변수에 대해 일어나는 동작을 포괄할 수 있도록 코드상에서 단위 루틴의 범위를 정하고 이를 통해 상위수준 연산을 추출하도록 한다.

예를 들면, 아래의 코드의 "가" 부분을 하나의 단위 루틴으로 인식하면 이는 change_current_record() 라는 연산으로 인식이 가능하다. 그러나 코드의 전체를 보면 이것은 현재의 레코드를 디스크 화일에 기록하는 루틴임을 알 수 있고 전체에서 save_data_record()라는 연산을 찾을 수 있다.

```
while(info) {
  fwrite(info->data, record_size, 1, fp);
  info = info->next;-----(가)
```

```
  if (ferror(fp)) {
    printf("cannot open destination file\n") ;
    fclose(fp) ;
    return 0 ;
  }
}
```

5) 코드에서 단위 루틴을 만들 때 구현시에 필요한 내용은 제외한다.

객체를 디자인할 때 코드상의 내용 중 일부는 객체의 기능을 설명하기 보다는 실제 시스템 상에서 구현하기 위해 필요한 내용들이 많이 있다. 이들은 실제 객체의 연산으로 정의되어서는 곤란하다.

예를 들면, 아래의 코드는 새로운 데이타를 화일로부터 로딩할 때 기존의 데이터 등을 위해 할당되어 있던 메모리를 해제하는 루틴으로 이는 코딩시 프로그래머에 의해 필요로 하는 내용이다. 그러므로 이들은 연산으로 정의할 필요가 없다.

```
while (first) {
    info = first -> next;
    free(info -> data);
    free(info);
    first = info;
}
```

6) 코드상에서 여러 서브루틴에 팬인(Fan-In)되면서 기능적 응집도를 지니는 서브루틴을 단위 루틴으로 만든다.

여러 서브루틴에 팬인되면서 하나의 기능을 수행하는 서브루틴은 연산이 될 가능성이 많이 있다.

예를 들면, 아래의 루틴은 CRT상의 지정된 위치로 커서를 옮기는 기능을 하는 루틴이며 이는 여러 루틴으로 팬인된다. 그리고 기능적 응집도를 지니고 있다. 이 루틴을 하나의 단위 루틴으로 두고 이를 연산으로 정의한다.

```
void gotoxy(int x, int y) {
    union REGS i;
    i.h.dh = y;
    i.h.dl = y;
    i.h.ah = 2;
    i.h.bh = 0;
    int86(16, &i, &i);
}
```

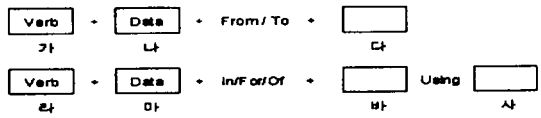
(2) 기능 기술

전 절에서는 객체의 추출을 위해 객체의 연산이 될 수 있는 단위 루틴을 인식하는 각종 가이드라인에 대해 알아 보았다. 본 절에서는 전 절에서 만들어진 각 단위 루틴을 객체를 추출하기에 용이한 형태의 다음 템플릿에 맞추어 그 기능을 기술할 때 필요한 가이드라인을 제시한다.

1) 단위 루틴의 기능은 다음과 같은 템플릿에 맞도록 정리한다.

첫번째 템플릿은 단위 루틴의 기능이 객체에 정보를 전송하거나 정보를 받아 들일 때 사용할 수 있다. 이 템플릿은 메소드 연산으로 사용되는 단위 루틴을 기술할 때 사용할 수 있다. 즉 다른 객체의 어떤 데이타를 읽어오거나 어떤 데이타를 변경하기 위하여 사용되는 메시지 전송(message passing)에 사용한다.

두번째 템플릿은 그 이외의 단위 루틴의 기능에 대해 단위 루틴의 중심이 되는 변수에 대한 동작을 기술하는데 사용된다. 두번째 템플릿의 동사(verb) 즉 "라"에는 "change", "search", "compute", "insert", "delete" 등이 사용될 수 있다. "사"에는 알고리즘, 디바이스, 모듈, 다른 객체 등이 들어갈 수 있다.



(그림 2) 템플릿
(Fig. 2) Template

2) 템플릿에 기술할 때 단위 루틴의 기능을 대표하는 동사를 기술하여야 한다.

단위 루틴의 기능을 대표할 수 있는 내용을 잘 선택해야 한다. 또한 비슷한 용어를 겸하여 쓰는 것보다 하나로 통일하여 쓰는 것이 좋다.

3) "다"와 "바"에는 데이타를 포괄할 수 있는 외부 개체나 자료구조 등을 기술할 수 있다.

템플릿의 "다"와 "바"에 들어갈 수 있는 내용을 기술할 때 실세계의 응용영역을 객체모델링하여 나온 객체를 참고로 하라. 객체모델링을 통해 얻어진 객체들은 현재의 응용영역에서 존재하는 객체의 경계를 설정하고 있다. 단위 루틴의 기능 및 단위 루틴의 중심이 되는 변수가 어느 객체에 사상될 수 있는 가를 파악하여 그 객체의 이름을 템플릿의 "다"와 "바"에 기술한다.

4) 동일한 동작을 수행하는 루틴이 코드상의 여러 서브루틴에 반복되어 있을 수 있으므로 이들을 추출할 때 동일한 이름으로 기술되도록 해야한다.

용도에 따라 동일한 동작을 수행하는 루틴들의 논리도 약간의 차이가 있을 수 있다. 그러므로 완전히 동일한 루틴은 반복되게 기술할 필요는 없으나 약간의 논리가 차이가 나는 것들은 동일한 이름으로 기술을 한다.

예를 들면,

salary = salary +100 : 세금이 환급되어 나올 경우
salary = salary * 0.8 : 휴직인 직원의 봉급인 경우

이들 두개의 문장은 각각 코드상의 여러 곳에 따로 존재할 수 있으며 이들의 기능은 모두 update_salary ()로 기술할 수 있다.

(3) 객체의 추출

전 절에서는 객체의 연산이 될 수 있는 단위 루틴을 객체를 인식하기 용이한 적절한 템플릿에 맞추어 기술하는 가이드라인을 알아 보았다. 본 절에서는 전 절에서 만들어진 템플릿을 이용하여 객체를 추출할 때 참고가 되는 가이드라인을 알아 보자.

1) 각 템플릿의 "다"와 "바" 부분이 같은 템플릿을 하나로 모으고, 외부개체 혹은 실세계의 응용영역에 대해 객체모델링을 통해 나온 객체를 참조하여 이들을 객체로 정의한다.

2) 각 그룹에 모여진 각 템플릿들의 "나"와 "마" 부분을 객체의 속성으로 둔다.

3) 각 그룹에 모여진 템플릿의 동사(verb)를 객체의 연산으로 한다. 템플릿의 "가"와 "라" 부분 및 "나"와 "마"를 중심으로 템플릿의 기능에 맞도록 연산이름을 만든다.

3.1.3 전역변수를 이용한 객체추출

본 절에서는 소스 코드상에서 사용되는 전역변수를 이용하여 객체를 인식하는 방법을 기술한다.

일반적으로 전역변수는 객체의 속성 및 객체가 될 가능성이 많이 있다[13]. 그러므로 이들 전역변수가 하나의 객체로 정의될 수도 있고 전역변수들 사이의 관계를 위주로 하여 클러스터링(clustering)하여 하나로 클러스터링된 전역변수들을 속성으로 하는 하나의 개념적 객체를 정의할 수도 있다. 이때 전역변수를 이용하여 추출된 객체는 시스템내의 표준 자료구조 및 사용자 정의 자료구조 등이 될 경우가 많다.

전역변수를 이용한 객체의 추출을 위한 프로세스는 먼저 전역변수를 클러스터링하는 단계와 클러스터로부터 객체와 속성을 추출하는 단계로 구성된다.

(1) 전역변수 클러스터링

객체추출을 위하여 다음의 세가지 방법으로 전역변수를 클러스터링한다.

1) C에서 Struct 형태의 자료구조를 이루는 전역변수는 하나로 클러스터링한다.

소프트웨어 시스템에서 관련된 개체를 나타내는 자료구조는 함께 사용 혹은 수정되기 때문에 그룹화되기 쉽고 이들 자료구조는 객체의 인스탄스 변수에 대응된다[13]. 그리고 이들 자료구조들의 모임(aggregated data structure)은 객체가 될 수 있다[13].

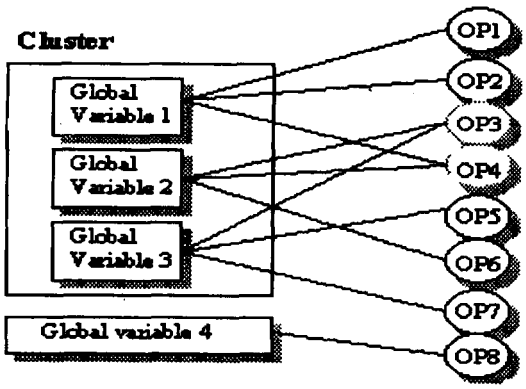
2) 공통의 데이타를 지니고 있는 전역변수들을 하나로 클러스터링한다.

3) 코드의 기능분석에서 나온 단위 루틴을 이용하여 전역변수들을 클러스터링한다[14].

이 클러스터링을 행할 때 다음의 세가지 단계로 한다.

단계 1 : 각 전역변수에 대해 그 전역변수를 사용하는 각 단위 루틴을 모은다.

단계 2 : 각 전역변수들에 공유되는 단위 루틴이 존재



(그림 3) 단위 루틴을 공유하는 전역변수들의 클러스터링 예
(Fig. 3) Clustering example of global variables sharing atomic routine

하는가 조사한다((그림 3)의 "op 4"와 같이 여러 전역변수에 공유되는 연산을 찾는다.).

단계 3: 단위 루틴을 공유하는 전역변수들을 하나로 클러스터링할 수 있는 가를 판단하기 위해 이들을 집단화(aggregation)하여 "consist of"의 관계를 지니거나, 표준 자료구조를 나타내거나, 다른 방법에서 추출된 객체에 사상되는 경우 이들 전역변수들을 하나로 클러스터링한다. 또한 역공학 과정에서 얻어진 전역변수의 의미 및 그들의 용도를 이용하면 보다 용이하게 클러스터링 여부를 결정할 수 있다. 예를 들면, (그림 3)에서 처럼 "op 4"를 공유하는 global 1, global 2를 집단화하여 이들이 하나로 클러스터링될 수 있는 가를 판단한다.

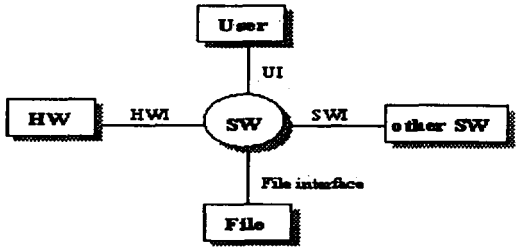
(2) 객체 및 속성 추출

전 절에서 3가지 방법으로 클러스터링한 각 클러스터를 각각 하나의 객체로 추출한다. 이때 클러스터링된 전역변수들은 클러스터링을 통해 정의된 개념적 객체의 속성이 된다. 또한 단위 루틴들은 전역변수들 중 각 단위 루틴의 동작의 핵심이 되는 전역변수가 속하는 개념적 객체의 연산이 된다. 또한 이들 전역변수를 사용하는 루틴에서의 전역변수에 대한 이들의 변경/이용하는 연산을 추출한다.

3.1.4 인터페이스를 이용한 객체추출

소프트웨어는 외부개체와 인터페이스를 하며 이들 외부개체들은 객체로서, 이들과의 인터페이스는 객체지향 방식으로 구현할 수 있다. 객체지향 시스템을 만들기 위해서는 소프트웨어와 인터페이스를 하는 외부개체를 가상기계(virtual machine)로 내부적으로 구현할 필요가 있으며 이들 외부개체를 객체로 정의할 수 있다. 그러므로 기존 시스템에 구현된 인터페이스를 분석하여 객체를 추출한다.

일반적으로 시스템의 인터페이스는 사용자 인터페이스, 하드웨어 인터페이스, 소프트웨어 인터페이스, 파일과의 인터페이스 등 4가지 형태가 있으며, 이는 다음 (그림 4)와 같이 나타난다.



(그림 4) 인터페이스 구성도
(Fig. 4) Interface Configuration Diagram

이때 사용자 인터페이스는 CRT, 마우스, 키보드 등을 통하여 이루어진다. 그리고 이들을 제외한 외부장치들을 하드웨어라 하며 이들을 통해 하드웨어 인터페이스가 일어난다. 또한 현재 시스템 이외의 외부 소프트웨어와 소프트웨어 인터페이스가 일어난다. 그리고 파일과의 파일 인터페이스가 일어난다.

인터페이스로부터 객체를 인식하는 프로세스는 다음과 같다.

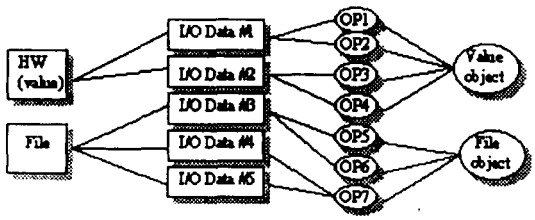
단계 1: 시스템과 인터페이스하는 외부개체를 찾는다.

시스템과 인터페이스하는 외부개체에는 사용자와 인터페이스하는 하드웨어(CRT, 키보드, 마우스, 프린터 등), 하드웨어(motor, valve, pump 등), 외부 소프트웨어와 같은 것들이 있으며, 이들은 역공학을 수행하는 과정에서 얻을 수 있다. (그림 5)의 예를 볼 때

단계 1에서는 시스템에 존재하는 외부개체로서 valve 및 파일이 존재한다.

단계 2: 각 외부개체별로 I/O를 수행하는 루틴 및 I/O 데이타를 찾아 모은다.

I/O를 수행하는 루틴들은 네트워크 관련 라이브러리 기능, 통신방법, I/O관련 시스템과 소프트웨어 라이브러리와 같은 것들이 있다. 또한 이들 I/O를 수행하는 루틴의 매개변수중에 이들 I/O데이타가 들어 있을 가능성이 많다. (그림 5)의 단계 2에서는 단계 1에서 얻은 각 외부개체 별로 일어나는 I/O루틴 및 I/O데이타를 찾아 기술한 형태이다. 즉 (그림 5)에서 하드웨어(valve)에 I/O되는 루틴 및 I/O데이타를 찾은 결과 I/O데이타 1(I/O루틴 1), I/O데이타 2(I/O 루틴 2)가 구해진다.



(그림 5) 인터페이스로부터 객체 인식 프로세스 예
(Fig. 5) Example of object extraction process from interface

단계 3: I/O를 수행하는 루틴 및 I/O데이타를 이용하여 속성과 연산을 추출한다.

I/O를 수행하는 루틴에서 I/O되는 데이타를 속성으로 둔다. 그러나 연산을 추출하기 위해 다음의 가이드라인을 참고로 한다.

A) I/O데이타에 설정되는 값을 분석하여 의미를 추출하고 이들 의미를 대상으로 각 연산을 만든다. (그림 5)의 경우 I/O데이타 1에 설정되는 값에 따라 연산 1 및 연산 2가 추출된다. 이러한 경우의 예를 들면, valve에 대해 On/Off를 조절하는 데이타를 인터페이스할 경우 valve_status = 0은 valve를 On, valve_status = 1은 valve 를 Off하라는 의미를 지니고 있다. 이들을 의미별로 valve_on( ), valve_off( )라는 각각의 연산으로 정의할 수 있다.

B) 몇 개의 I/O데이타를 조합하여 하나의 의미를 만들 수 있고 이것이 하나의 연산이 된다. (그림 5)에서 I/O데이타 4와 I/O데이타 5를 합하여 하나의 연산이 되는 경우도 있다.

예를 들면, 아래와 같이 메뉴를 만들기 위해 여러 개의 printf 문을 사용할 경우, 이로부터 4개의 I/O루틴을 얻을 수 있으며, 이들을 조합하여 보면 CRT에 메뉴를 출력하기 위한 루틴들임을 알 수 있다. 그러므로 조합하여 하나의 연산(e.g., Display_Menu( ))을 추출할 수 있다.

```
printf("1.Clear");
printf("2.Delete");
printf("3.Insert");
printf("4.Modify");
```

단계 4: 객체를 추출한다.

객체를 추출하기 위해 다음의 가이드라인을 참고로 한다.

A) 하드웨어, 외부 소프트웨어 및 파일의 경우 이들을 하나의 객체로 한다. 즉, 실세계의 개체를 소프트웨어 경계내에 구현하기 위한 가상기계 기능을 하는 객체로 된다. 예를 들면 (그림 5)의 외부개체인 valve의 경우 그와 관련된 모든 연산을 모두 갖는 객체가 된다.

B) 실세계의 응용분야에 대한 객체모델링에서 추출된 객체, 전역변수에서 추출된 객체, 프로그램의 기능을 분석하여 추출한 객체를 참고하여 I/O루틴을 그룹화하는데 이용한다.

단계 5: 위에서 찾아진 연산의 처리

A) 속성이 기존의 객체로 존재할 경우, 앞에서 찾아진 연산은 그 객체의 연산으로 첨가한다. 속성은 기존 객체의 속성으로 추가한다. (단, 그 속성이 다른 객체의 속성으로 등록되어 있지 않아야 한다.)

B) 속성이 기존의 추출된 객체의 속성으로 존재할 경우, 앞에서 찾아진 연산은 속성이 소속된 객체의 연산으로 첨가한다.

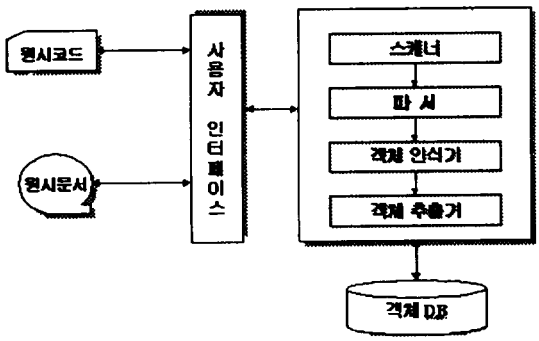
C) 속성이 기존의 객체 및 속성으로 정의되어 있지 않은 경우, 이들 속성들을 모아 그들 사이의 관계에

의해 집단화를 한 후 추상객체를 정의하고 집단화된 각 데이타들은 속성으로 두고 추상화를 할 수 없는 경우, 이때의 속성을 하나의 객체로 둔다.

4. 툴의 구현

4.1 툴의 구조

기존 절차 중심의 프로그램으로부터 객체를 추출하는 툴의 구조는 (그림 6)과 같다.



(그림 6) 객체 추출 시스템의 구조
(Fig. 6) Configuration of Object Extraction System

1) 스캐너(scanner)/파서(parser)

변수는 객체의 속성이 될 수 있다. 특히 전역변수는 객체가 될 가능성이 높다. 스캐너는 원시 코드를 읽어 들여 그 속에 포함된 변수(전역변수, 함수 매개변수 등)를 찾아낸다. 원시 코드(C 언어로 작성)를 입력하여 기존 BNF에서 정의된 문법대로 스캐닝하여 토큰을 분리하고, 파서는 이들 토큰에 기준하여 변수, 함수 등을 중심으로 파싱하는 단계이다.

2) 객체 인식기

객체 인식기는 객체의 속성이 될 가능성이 높은 변수를 분석하여 속성을 찾는 서브 시스템이다. 그리고 이들에 행해지는 행위를 중심으로 기능적 응집도를 갖도록 단위 루틴을 인식하여 연산으로 정의할 수 있다. 따라서 이렇게 하기 위해 인식된 단위 루틴을 템플릿에 맞도록 기술하는데 이때 단위 루틴의 기능을 대표할 수 있는 동사를 사용하며, 데이타를 포괄할 수 있는 외부 개체나 자료구조 등을 기술한다.

또한 각 외부 개체별로 I/O를 수행하는 루틴 및 I/O 데이타를 찾아 이들을 이용하여 속성 및 연산을 찾을 수 있다.

3) 객체 추출기

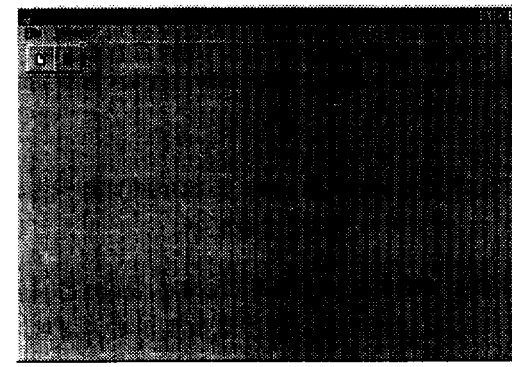
인식된 객체에서 전역변수들은 공통의 데이타를 지니고 있는 것들 끼리 혹은 단위 루틴을 이용하여 하나로 클러스터링(Clustering)하며, 각 클러스터를 하나의 객체로 추출한다. 또한 외부 개체(하드웨어, 외부 소프트웨어, 파일 등)는 하나의 객체로 정의한다. 추출된 객체는 C^{++} 클래스 형태로 출력되거나 DB에 저장된다.

4) 객체 DB

추출된 객체, 인식된 객체, 그리고 원시 코드 등이 저장되며 객체명, 속성, 연산의 형태로 만들어 저장한다.

4.2 툴의 실행 예

본 논문에서 구현된 툴은 Delphi를 사용하였으며, 데이터베이스는 Delphi DB를 사용하였고, 구현된 툴의 초기화면은 (그림 7)과 같다.

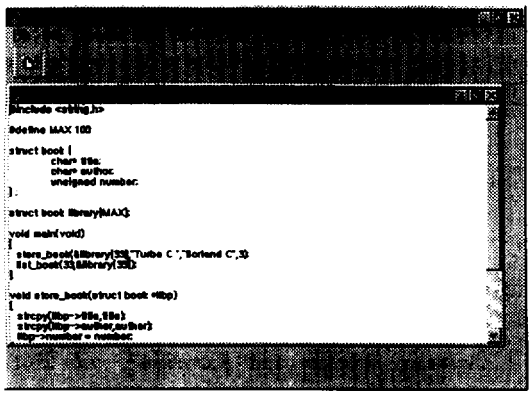


(그림 7) 초기화면
(Fig. 7) Initial Overview

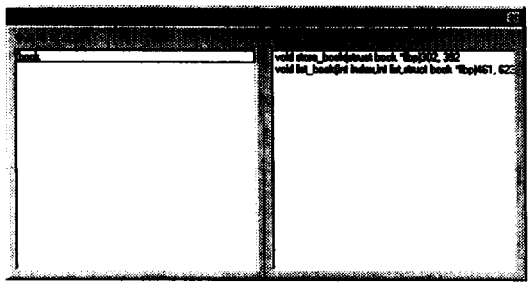
(그림 8)과 같이 C언어로 작성된 기존 원시 코드로부터 본 논문에서 제시한 방법대로 툴을 실행한 결과 (그림 9)와 같은 구조체와 함수 리스트를 추출하였으며, (그림 10)과 같은 클래스를 생성할 수 있었다.

(그림 11)은 C언어로 작성된 complex처리를 위한

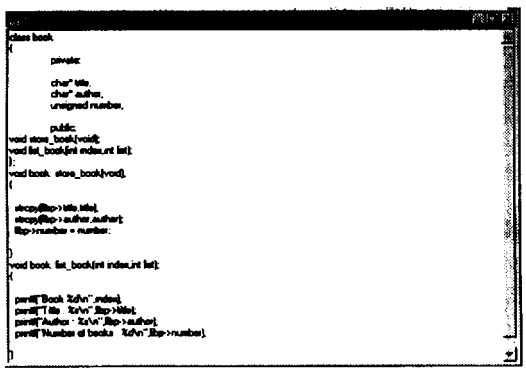
구조체의 예이다. 이를 구현된 도구에 실행한 결과 (그림 12)와 같은 구조체 리스트와 함수 리스트를 추출하였으며, (그림13)과 같은 클래스를 생성하였다.



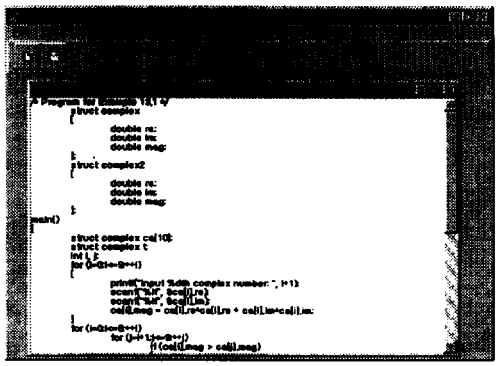
(그림 8) 예1의 원시 코드
(Fig. 8) Source Code of Example 1



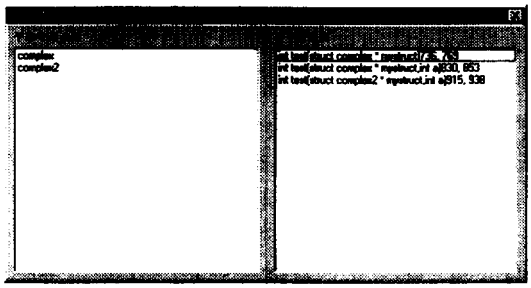
(그림 9) 예1의 추출된 구조체와 함수 리스트
(Fig. 9) Extracted Structure and Function List of Example 1



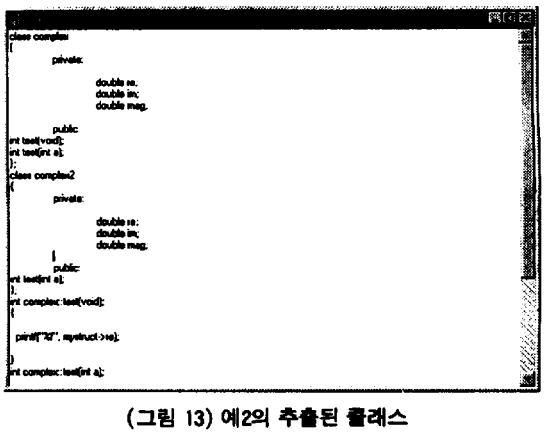
(그림 10) 예1의 추출된 클래스
(Fig. 10) Extracted Class of Example 1



(그림 11) 예2의 원시코드
(Fig. 11) Source Code of Example 2



(그림 12) 예2의 추출된 구조체와 함수 리스트
(Fig. 12) Extracted Structure and Funtion List of Example 2



(그림 13) 예2의 추출된 클래스
(Fig. 13) Extracted Class of Example 2

5. 결 론

지금까지 기존 시스템을 객체지향 시스템으로 재

공학하기 위해 많은 시도가 있었다. 기존의 문서를 사용하는 경우는 기존의 문서가 유지보수되는 과정에서의 변경 내용을 적극적으로 수용하지 못함으로 인해 큰 실효성을 얻지 못하게 되는 경우가 많이 있고, 코드에만 의존하게 되는 경우 객체지향의 장점을 잘 살릴 수 있는 객체의 추출에 문제가 있었다. 그래서 본 논문에서는 코드에만 의존하지 않고 기존 시스템의 분석을 통한 실세계의 응용영역에 대한 객체 모델링을 통해 시스템에 존재하는 객체의 종류를 파악하여 코드에서 객체를 추출할 때 얻기 힘든 개념적 무결성 문제를 해결할 수 있다.

또한 기존의 방법들이 코드에서 객체를 인식할 때 코드에 존재하는 데이터 및 그에 대한 기능을 중심으로 객체를 인식하고자 하였으나, 본 논문에서는 이들 관점이외에도 코드에서 일어나는 외부 개체 및 타 시스템과의 인터페이스를 이용하여 객체를 인식하고자 하는 방법 등 객체 추출의 새로운 관점을 도입하여 코드상에 존재하는 객체의 추출을 다양화하여 보다 많은 종류의 객체를 추출할 수 있는 방법을 제시하였다.

그러나 개념적인 객체의 추출을 위한 방법은 제시되고 있으나 이를 위한 가이드라인이 다소 부족하며 클래스 계층구조 및 객체간의 인터페이스를 위한 가이드라인이 미흡하다. 또한 인식되어진 객체들로서 기존 프로그램의 기능을 만족시킬 수 있는가에 대한 검증이 필요하며 객체추출을 자동화할 도구의 개발도 연구과제라 할 수 있다.

참 고 문 헌

[1] 강성구, "기능 데이타, 인터페이스 분석에 의한 객체 추출법", 포항 공대 정보산업 대학원, 1994.

[2] Steven Woods, Qiang Yang, "The Program Understanding Problem: Analysis and Heuristic Approach", Proceeding of ICSE, pp6-15, 1996.

[3] James Martin, James J. Odell, Object-oriented Analysis and Design, prentice-hall, 1992.

[4] E. Yourdon, "RE-3", American Programer, vol 12, no. 4, pp3-10, 1989.

[5] Harald C. Gall, Rene R. Klosch, Roland T. Mittermeir, "Architectural Transformation of Legacy Systems", ICSE-17 workshop on program transformation for software evolution technical report CS95-418, 1995.

[6] Evelin Kofler et al, "Balancing in Reverse Engineering and in Object-oriented Systems Engineering to Improve Reusability and Maintainability", COMPSAC '94, 1994.

[7] 장종표, 강대욱, 김병기, "구조적 분석모델로부터 OMT의 객체모델로의 사상", 정보처리학회 학술발표 논문집 3권 2호, pp561-565, 1996.

[8] 이정진, 김치수, "DFD로부터 E-R모델과 객체모델 유도를 위한 툴의 설계", 정보처리학회 학술발표 논문집 2권 2호, pp310-314, 1995.

[9] Vincent D. Moynhan, "The Design and Implementation of a High-level Language Converter", Software Practice and Experience, vol 21(4), pp391-400, 1991.

[10] Eric J. Byrne, "Software Reverse Engineering: A Case Study", Software Practice and Experience, vol 21(12), pp1349-1364, 1992.

[11] Yih-Farn Chen, Michael Y. Nishimoto, C. V. Ramamoorthy, "The C Information Abstraction", IEEE transaction on software engineering, vol 16, no. 3, pp325-334, 1990.

[12] 김태균, "C++ 언어에 대한 역공학도구의 설계및 구현", 정보과학회지 1권 2호, pp135-145, 1995.

[13] C. L. Ong, W. T. Tsai, "Class and Object Extraction from Imperative Code", JOOP, March-April, 1993.

[14] S. Lee, J. Lee, D. Chi, K. Kim, "A reengineering Technique for Transformation of Function Orieted Program to Object-based Program", Proceeding infoscience '93.

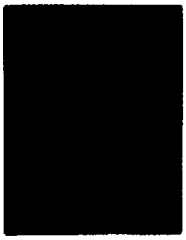
[15] H. P. Haughton, K. Lano, "Object Revisit", Proceedings conference on software maintenance 1991, IEEE comput. Soc. press.

[16] Grady Booch, Object-oriented Design with Application, Benjamin/Cummings pub., 1991.

[17] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenson, Object-oriented Modeling and Design, prentice-hall, 1991.

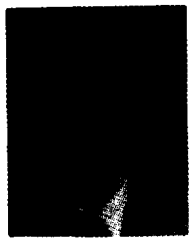
[18] P coad, E. Yourdon, Object-orinted Analysis,

prentice-hall, 1990.
[19] S. Shlaer, S. Mellor, Object Lifecycles Modeling the World in States, Yourdon press, 1992.



김 정 종

1977년 중앙대학교 전자계산학과 졸업(이학사)
1981년 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1988년 중앙대학교 대학원 전자계산학과 졸업(이학박사)
1978년 삼미금속(주)
1979년~1980년 금성통신(주) 연구소
1981년~현재 경남대학교 컴퓨터공학과 교수



손 창 민

1978년 경북대학교 컴퓨터공학과 졸업(공학사)
1983년 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1993년 경남대학교 대학원 컴퓨터공학과 박사과정
1982년~1984년 삼일경영경제연구원
1985년~1986년 금성하니웰(주)
1987년~현재 경북실업전문대학 전자계산과 부교수