절차적 추상화의 분류와 다형성

김 성 기†

요 약

소프트웨어의 개발, 유지보수 및 확장을 용이하기 하기 위하여 프로그래밍 언어에 여러 추상화 기법이 도입되었다. 그러나 추상화와 연관된 여러 개념과 용어가 통일성이 있게 정의되지 못하므로 인하여 관점에 따라 다르게 설명되기도 한다. 특히 추상화를 가장 강력한 기반으로 하여 태동된 객체 지향 패러다임에서 이러한 혼란은 아직도 계속되고 있는 실정이다. 본 논문에서는 프로그래밍 언어에서 가장 중요한 추상화의 하나인 절차적 추상화를 분석하고 이에 대한 유형을 살펴본다. 이를 통하여 절차적 추상화가 지금까지 일반적으로 인식되어온 '처리과정을 기능으로 바꾸는 추상화'라는 기본적인 추상화의 개념 외에도 '추상화에 대한 추상화'의 개념이 내포된 고수준의 추상화도 포함되어 있음을 밝힌다. 또한 절차적 추상화와 관련된 다형성을 추상화의 관점에서 설명한다. 이러한 분석과 고찰을 통하여 절차적 추상화에 대한 수준 높은 이해가 이루어질 수 있을 것이다.

Taxonomy of Procedural Abstraction and Polymorphism

Sung Ki Kim†

ABSTRACT

Many abstraction techniques are introduced in programming languages in order to facilitate the development, maintenance and extension of softwares. But several concepts and terms related to abstraction have not been uniformly defined and have been explained differently. Expecially in object-oriented paradigm strongly based on abstraction common consensus was not derived. In this paper, we analyse procedural abstraction, propose the taxonomy of procedural abstraction and show that procedural abstraction contains the concept of abstraction on abstraction in addition to the transformation of procedures into abstraction. We also explain polymorphism in the view of abstraction. These analysis and consideration will make it possible to understand procedural abstraction more intensively.

키워드 : 절차적 추상화(Procedural Abstraction), 부프로그램(Subprogram), 다형성(Polymorphism)

1. 서 론

인간을 둘러싼 세계는 아주 복잡하며 끊임없이 변화한다. 복잡하고 계속 변화하는 구체적인 제반 사물과 현상을 있는 그 자체로 인식하고 기억하는 것은 불가능하며 그러할 필요도 없다. 인간은 외부세계를 모델링하여 단순화되고 추상화된 인식체계를 형성하며 이를 기반으로 추상적 삶을 영위한다. 실제의 사물과 현상에서 핵심적이며 중요한 것을 추출하고 기억하며 활용하는 능력은 인간을 인간답게 하는 중요한 요소라고 할 수 있다.

추상화(abstraction)는 기존의 것에 대하여 지금까지 존재하지 않았던 기호, 명칭, 개념, 용어 등 추상적인 것을 새로이 창조하는 것이다. 인간이 시도한 최초의 추상화는 무엇이며 최초의 추상화에 대한 기록은 어디에서 볼 수 있는가도 흥미로운 물음이 될 것이다. 가장 기본적인 추상화가 사물에 대한 명칭을 붙이는 것이라 할 수 있다. 최초로 이름을 부여하는 것이 구체적으로 명시된 문헌 중 대표적인 것이 성경이다. 성경의 창세기 2장 19절에는 "여호와 하나님이 흙으로 각종 들짐승과 공중의 각종 새를 지으시고 아담이 어떻게 이름을 짓나 보시려고 그것들을 그에게로 이끌어 이르시니 아담이 각 생물을 일컫는 바가 곧 그 이름이라."라고 되어있다[3]. 아담이 한 동물에게 부여한 그 이름은 모양, 상태, 행동 등 그 동물과 관련된 모든 것을 대표하는 추상화이다. 이러한 이름 추상화는 다양한 특성을 갖는 여러 객체들이 있을 때 상호 구별이 가능한 간단한 추상물을 생성한 후 이름을 통하여 한 객체를 정확하면서 간단하게 언급할 수 있게 한다. 인간 최초의 지적 활동으로 이름 추상화가 성경에 명시된 것은 인간에게 있어서 추상화가 매우 중요한 능력임을 강조하려는 의도를 엿볼 수 있다.

아담이 부여한 이름이 객체의 이름인지 분류명인지에 대해 논란의 여지가 있다. 이름 '홍길동'은 홍길동이라는 특정 사람을 대표하기 위한 객체 이름이며, '쥐', '소' 등은 한 종류의 모든 동물들을 통합적으로 일컷기 위한 추상물로서,

※ 이 논문은 2002년도 한신대학교 학술연구비 지원에 의하여 연구되었음.
† 정 회 원 : 한신대학교 컴퓨터학과 교수
논문접수 : 2002년 9월 13일, 심사완료 : 2003년 3월 10일

동일한 특성을 갖는 동물 전체를 나타내는 분류명이다. 객체 이름, 분류명 등 새로운 이름의 생성으로부터 시작되는 여러 형태의 추상화는 문명의 근간을 형성하게 한 언어의 기반이 되며, 인간을 창조적이며 고차원적 존재로 만드는 한 요인 중의 하나일 것이다.

추상적인 대상의 특성을 다루는 수학에서 추상화는 특히 중요하다. 0, 1, 2 등의 숫자, +, - 등의 연산자, 점, 선, 삼각형 등의 정의, 피타고라스 정리, 괴델의 불완전성의 정리 등의 여러 정리들은 모두 추상화된 결과를 표현하고 이를 조작하는 방법을 제공한다. 컴퓨터 분야에서도 추상화는 중요한 역할을 하고 있다. 컴퓨터 분야에서의 가장 기본적인 추상화는 명령(instruction)과 코드 체계(code system)이다. 명령은 일련의 마이크로명령들을 추상화시킨 것으로 프로세서의 기본적 처리 단위를 나타낸다. 코드 체계는 다양하게 표현되어지는 것들을 하나의 통일된 형태로 표현하는 방법을 제공하며, 코드 체계의 코드 하나하나는 추상적 산물이다.

명령과 코드로부터 시작된 프로그래밍 언어에서는 변수, 타입 등의 보다 추상적인 개념들을 수용하였다. 변수 추상화와 타입 추상화가 어셈블러나 컴파일러에 의해 제공되기 시작한 이래 프로그래머들은 하드웨어의 상세함을 고려하지 않은 고차원적인 프로그래밍을 할 수 있게 되었다.

고급 프로그래밍 언어의 개발 및 데이터 처리 기술의 발전과 더불어 더욱 다양한 추상화의 개념과 기법이 도입되었다. 추상화와 관련된 개념 또는 용어로는 절차적 추상화(procedural abstraction)[17], 복합 데이터(complex data), 데이터 추상화(data abstraction)와 추상 데이터 타입(abstract data type)[8, 17, 18], 복합 객체(complex object), 객체 추상화(object abstraction)[10], 상속(inheritance)[21], 분류(classification)[23], 일반화(generalization)와 집단화(aggregation)[19], 캡슐화(encapsulation)[10, 18], 다형성(polymorphism)[1, 6, 10], 릴레이션 및 데이터베이스[22] 등 아주 다양하다.

지금까지 컴퓨터 분야에서 다양한 형태와 종류의 추상화가 개발되면서 그 자체 하나하나는 잘 정의되고 널리 활용되어 왔지만 아직 추상화라는 전체적인 관점에서 통일성 있게 정의되거나 설명되지 못하고 있다. 그 결과 추상화라는 개념은 다소 애매모호한 특성을 갖는 용어로 인식되어지고 있다. 또한 추상화와 연관된 여러 개념과 용어도 통일성이 있게 정의되지 못하며 관점에 따라 다르게 설명되는 혼란스러움을 경험하고 있다. 특히 추상화를 가장 강력한 기반으로 하여 태동된 객체 지향 패러다임에서 이러한 혼란은 아직도 계속되고 있는 실정이다[21]. 이것은 컴퓨터 분야에서 제공되어지는 추상화에 대하여 정확한 분석이나 체계적 분류가 이루어지지 않았기 때문이다.

본 논문에서는 모듈(module), 또는 부프로그램(subprogram)에 나타난 절차적 추상화의 양상을 분석한다. 실행 문장들의 단순한 대체에서부터 매개변수화, 통합 등의 고차원적 추상화를 통하여 보다 강력한 기능을 수행할 수 있도록 발전한 과정 속에 내재된 여러 형태의 추상화를 분석하고 분류하며, 또한 부프로그램 이름의 중복에 의해 야기되는 다형성에 대해서도 논의한다. 이러한 절차적 추상화에 대한 체계적 고찰은 우리가 아는 범위에서는 아직 시도된 바가 없으며, 이를 통하여 절차적 추상화에 대한 체계적 이해와 정형적 정의가 가능할 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 먼저 추상화 및 절차적 추상화의 정의를 살펴본다. 3장에서는 절차적 추상화를 분류하고 각 유형별로 분석한다. 4장에서는 절차적 추상화와 연관된 다형성에 대하여 논의하고 5장에서 결론 및 앞으로의 연구방향을 제시한다.

2. 추상화 및 절차적 추상화의 정의

추상화, 추상 또는 추상화 과정 그 자체는 철학, 수학, 과학 등 여러 학문분야에서 기본적인 학문의 방법으로 사용되어온 개념으로 엄밀히 정의되기 어려운 개념이다. 사이버 두산세계백과대사전[9]에서 추상이라는 용어를 살펴보면 다음과 같다.

> 대상으로서의 소여 전체로부터 특정성질이나 공통징표를 분리하고, 골라내는 정신작용. 예를 들면, 인간의 얼굴만을 표상으로서 분리하는 것은 대상과 동차원상의 절단이며 본래의 추상이라고는 할 수 없다. 빨간 넥타이로부터 '빨강' 혹은 '형' 만을 추출하는 것, 또 빨간 우체통, 잘 익은 토마토 등에서 공통적인 '빨강'을 골라내고, 적·청·황으로부터 '색'을 빼내는 것은 추상이다. 추상은 불필요한 계기를 버리는 사상을 표리일체로 동반한다. 추상에는 보편성·일반성의 정도가 있고, 고도의 추상은 언어작용과 밀접히 관계하여 보통명사나 명제의 형성, 유형화, 이론구성의 전제가 되어, 일상적·학문적 활동에 불가결한 작용이다.

이러한 추상화의 일반적 정의에서는 추상화는 전체에서 무엇인가를 분리하고 골라내는 과정을 거쳐 새로운 어떤 것을 형성하는 것임을 말하고 있다. 추상화는 복잡하고 다양하고 구체적인 사물과 현상에서 일반적이거나, 핵심적이거나, 필수적이거나, 통일성이 있거나, 단순하거나, 대표성이 있는 어떠한 진수를 분리하고 추출하여 이를 기호, 아이콘, 명칭, 개념 등으로 표현하는 것이다.

추상화에서 비롯되는 심각한 문제는 추상화의 결과로 생겨난 신, 사랑, 점, 선, 면 등의 추상적 용어가 정확히 무엇을 추상하였는가, 즉 추상의 실제적 대상이 무엇이며, 분리되고 골라진 것이 무엇인가가 명확히 정의되지 않는다는 것이다. 그로 인하여 하나의 용어에 대하여 주관적 해석이 가능하며, 해석을 달리 하는 사람들 간에는 의견 차이와 오

해가 발생하기도 한다. 엄밀한 정의로부터 출발하여 증명을 통하여 정리를 제시하는 수학에서조차 무정의 용어(점, 선 면 등 구체적인 정의를 내리지 않고 그 성질을 공리로 규정하는 수학적 개념)와 무증명 명제(자명한 것이라고 생각되어 증명을 요하지 않는 명확한 명제)인 공리(axiom)를 포함하는 것은 모든 추상적 개념을 엄밀하게 정의할 수 없다는 것을 단적으로 보여준다.

프로그래밍 언어에서 추상화를 위한 수단들은 기계 또는 하위 단계의 상세함으로부터 벗어나 보다 고수준의 프로그래밍을 가능하게 한다. 프로그래밍 언어에서의 추상화의 몇 가지 정의를 살펴보면, 주어진 문제나 시스템 중에서 중요하고 관계있는 부분만을 분리하여 간결하고 이해하기 쉽게 만드는 작업[9], 어떤 범주의 과정이나 객체가 그것의 특성의 일부들인 핵심적 특성들에 의해 표현되어지고 나머지의 다른 특성들은 제거되거나 숨겨지는 것[16], 여러 복잡한 것들을 모아 하나의 단위로 취급할 수 있는 단일 개체(entity)로 변환하는 과정 등이 있다. 이러한 정의들은 추상화의 일반적 정의에서 크게 벗어나지 않는다.

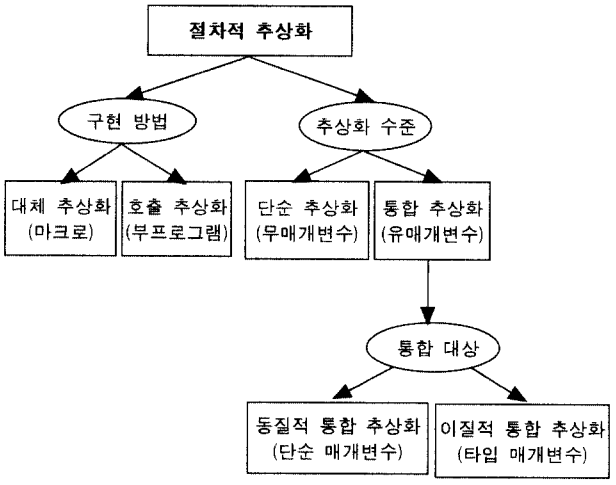
프로그래밍 언어에서의 추상화는 추상 결과의 명료성과 상호 대체성이라는 다른 분야에서 볼 수 없는 특징을 가진다. 추상 결과의 명료성은 추상화된 결과가 엄밀한 표기를 통하여 정형적으로 표현됨을 말하며, 이는 수학적 추상화의 맥을 그대로 이어받은 결과이며 정형 언어인 프로그래밍 언어가 갖추어야 할 기본적인 요건이다. 더욱 중요한 특징이 추상화 대상과 추상화 결과간의 상호 대체성이다. 상호 대체성은 추상화의 대상과 추상화의 결과가 명확히 정의됨으로 인하여 나타나는 현상으로, 추상화의 결과가 원래의 추상화 대상으로 환원될 수 있다는 것이다. 추상적 요소를 포함하는 프로그램은 궁극적으로 기계가 수행 가능한 이진수 코드로 변환되어야 컴퓨터 하드웨어 상에서 수행될 수 있으므로 상호 대체성은 반드시 보장되어야 한다. 변수와 타입의 추상화에서 시작하여 객체 및 클래스 추상화에 이르기까지 모든 추상화에서 상호 대체성은 제공되어진다.

프로그래밍 언어에 나타난 여러 추상화 중 대표적 추상화는 절차적 추상화(procedural abstraction)이다. 절차적 추상화는 초기 프로그래밍 언어에서의 가장 중요한 추상화였고 지금도 역시 중심 개념으로 자리잡고 있다. 절차적 추상화란 처리에 필요한 일련의 과정들을 추상화하여 이를 간단하게 호출될 수 있는 형태로 표현하는 것을 말한다[16]. 절차적 추상화가 이루어지면 '처리되는 과정'은 '처리되는 기능'으로 바꾸어진다. 그리하여 일련의 문장들이 내부적인 처리절차인 구현(implementation)과 외부적인 기능을 나타내는 인터페이스(interface)로 서로 분리되어 하나의 부프로그램을 형성한다. 부프로그램은 선언된 후 인터페이스를 통하여 호출될 수 있다. 부프로그램을 선언할 때 인터페이스에는 이름과 매개변수가 명시되며, 구현에는 주어진 매개변수들을 통하여 수행될 처리과정이 구체적으로 명시된다. 그러나 인터페이스에는 처리과정을 나타내거나 암시할 필요가 전혀 없다. 부프로그램이 호출되면 명시된 실제적인 처리과정이 수행되어진다.

보다 정형적으로, 절차적 추상화는 구현의 집합 Im과 인터페이스의 집합 In간의 관계 Ap로 정의된다. 즉, $Ap \subseteq Im \times In$이다. 절차적 추상화는 여러 이점을 가진다. 첫째, 복잡한 처리과정을 간단한 메소드 호출로 대체함으로 프로그램이 간단해진다. 둘째, 프로그램이 기능적인 모듈들로 분해됨으로 작성과 디버깅이 용이해진다. 셋째, 정보 은폐(information hiding)가 가능하다. 넷째, 인터페이스에 영향을 주지 않고서 구현의 변경이 가능하므로 프로그램의 변경이 용이하다. 다섯째, 한번 작성된 메소드는 다른 프로그램에서도 사용할 수 있으므로 재사용성이 높아진다.

3. 절차적 추상화의 분류

본 장에서는 절차적 추상화에 대한 체계적인 분류를 실시한다. 절차적 추상화의 기본적 분류기준은 구현방법과 추상화 수준이다. 구현방법은 절차적 추상화가 실제 구현되는 방법에 따른 분류기준으로, 이에 의거하여 절차적 추상화는 마크로에서의 추상화에 해당되는 대체 추상화와 부프로그램에서의 추상화에 해당되는 호출 추상화로 분류된다. 또한 절차적 추상화는 추상화의 수준에 따라 단순 추상화와 통합 추상화로 분류된다. 단순 추상화는 수행될 여러 문장들을 간단한 이름으로 추상화한 것으로, 매개변수가 없는 마크로나 부프로그램의 추상화에 해당된다. 통합 추상화는 유사한 여러 추상화들로부터 새로운 추상화를 형성하는 것으로, 통합 추상화의 결과 매개변수가 도입된다. 또한 통합 추상화는 통합 대상의 성질에 따라 동질적 통합 추상화와 이질적 통합 추상화로 분류된다. (그림 1)은 절차적 추상화의 분류를 나타낸다.



(그림 1) 절차적 추상화의 분류도

3.1 구현방법에 의한 분류

초기의 컴퓨터에서는 프로그래머들은 기억장소의 주소와 데이터 그리고 기계어 명령들을 모두 이진수 패턴으로 직접 변환하여 컴퓨터에 입력하였다. 이러한 변환 과정과 주소, 데이터 및 명령의 입력과정에서 오류가 발생하기가 쉬웠다. 주소, 데이터 그리고 명령을 기호로 나타내는 어셈블리어와 어셈블리 프로그램의 기호를 이진수로 바꾸어주는 어셈블러가 개발되어 어셈블리어로 프로그래밍을 하게됨에 따라 프로그래머의 생산성이 그 전에 비하여 크게 향상되었고, 변환 오류의 감소, 빠른 번역, 용이한 프로그램의 변경 등이 가능하게 되었다.

큰 어셈블리 프로그램에서는 똑같은 명령들이 여러번 반복적으로 나타나게 된다. 반복되는 명령들을 일일이 명시하지 않고 간단한 하나의 명령으로 정의할 수 있게 하는 마크로(macro)가 어셈블러의 한 기능으로 제공되었다. 단순한 문자열 대체에 불과하지만 마크로는 프로그램의 간단한 작성, 이해성의 증대, 프로그램의 크기 감소, 여러 프로그램에서 마크로의 공유 등 여러 장점을 가진다. 마크로는 매개변수를 가질 수 있도록 확장되었으며, C 등의 고급 프로그래밍 언어에서도 마크로는 제공되어진다. 다음은 C 언어에서의 마크로 정의의 마크로 호출의 예이다.

```
#define add10toSize (size = size+10)     // 마크로 정의
#define max(a, b) ((a)>(b) ? (a) : (b))  // 마크로 정의

add10toSize ;          // 마크로 확장 : (size = size+10) ;
m = max(5, size) ;  // 마크로 확장 :
                       m = ((5) > (size) ? (5) : (size)) ;
```

마크로는 최초의 절차적 추상화를 내포하는 프로그램의 한 요소이라고 할 수 있다. 마크로에서 나타난 추상화는 여러 실행문장들을 간단한 마크로 이름으로 대체하는 대체 추상화이다. 대체 절차적 추상화에서도 절차적 추상화의 가장 핵심인 구현과 인터페이스의 분리가 이루어진다.

대체 추상화인 마크로가 프로그램 내에서 출현하면 마크로는 정의된 코드로 대체된다. 마크로 정의는 소스 프로그램에서 반복되는 코드 부분을 간단한 마크로 이름으로 나타날 수 있게 하지만, 목적 프로그램에서 실제 수행되는 코드는 반복되어 나타내게 되며, 이로 인하여 목적 프로그램의 크기가 커지게 된다.

대체 추상화와 달리 호출 추상화는 정의된 추상물이 프로그램에서 나타나면(이를 호출이라고 함) 수행될 코드가 대체되지 않고 프로그램의 제어가 수행될 코드로 이동한 후 수행된다. 호출 추상화는 부프로그램을 통하여 구현된다. 부프로그램은 외형적으로 마크로와 유사하다. 즉, 부프로그램도 실제 수행될 여러 문장들을 부프로그램 이름으로 추상화되며, 프로그램에서 부프로그램의 이름이 표시되어 부프로그램이 호출되면 추상화된 문장들이 수행된다. 이 과정에서 호출된 곳과 다른 곳으로 제어가 이동하여 추상화된 문장들이 수행된 후 다시 호출된 곳으로 제어가 돌아온다. 이와 같은 부프로그램의 호출, 제어의 이동, 추상화된 문장의 수행, 제어의 복귀는 목적 프로그램에서 반복적으로 수행될 부분들이 한번만 나타나게 한다. 또한 하나의 부프로그램은 외부적 기능 단위를 형성할 뿐 아니라 내부적 실행의 단위가 된다. 1부터 4까지의 합과 1부터 20까지의 합을 구하는 부프로그램을 C 함수로 작성한 것은 (그림 2)와 같다.

```
int getSumFrom1To4 ( ) {
    return (10) ;        /* 미리 알고 있는 합을 반환 */
}

int getSumFrom1To20( ) {
   int sum = 0 ;
   int i ;
   for (i = 1 ; i <= 20 ; i++)
      sum = sum + i ;
   return(sum) ;        /* 합을 구하여 반환 */
}
```

(그림 2) 합을 반환하는 C 함수들

3.2 추상화 수준에 의한 분류

추상화 수준에 따라 추상화는 단순 추상화와 통합 추상화로 분류된다. 단순 추상화는 수행될 여러 문장들을 간단한 이름으로 추상화하고 이름을 통하여 수행될 문장들을 참조할 수 있게 한다. 매개변수가 없는 부프로그램 또는 마크로는 단순 추상화가 적용된 것이며, (그림 2)의 매개변수가 없는 함수인 getSumFrom1To4()와 getSumFrom1To20()은 단순 추상화의 예이다.

매개변수가 있는 부프로그램과 매개변수가 없는 부프로그램을 추상화에 관점에서 엄밀히 분석하면 이들 사이에는 커다란 차이를 발견하게 된다. 매개변수가 없는 부프로그램에서의 추상화는 여러 문장들을 단순한 부프로그램 이름으로 사상(mapping)한 것이다. 이에 반하여 매개변수가 있는 부프로그램에서의 추상화는 수행될 문장들과 부프로그램 이름간의 단순한 사상이 넘어 이미 추상화된 여러 부프로그램을 하나의 부프로그램으로 통합시키는 통합 추상화가 내포되어 있다.

```
int getSumFrom1To(int n) {
   int sum = 0 ;
   int i ;
   for (i = 1 ; i <= n ; i++)
      sum = sum + i ;
   return(sum) ;
}
```

(그림 3) 1부터 n까지의 합을 구하는 C 함수

예를 들어, 1부터 임의의 정수 n까지의 합을 구하는 부프

로그램을 C 함수로 작성하면 (그림 3)과 같다. 하나의 매개변수를 가지는 함수 getSumFrom1To(int n)은 1부터 특정 정수까지의 합을 구하는 매개변수가 없는 함수들(getSumFrom1To1(), getSumFrom1To2(), …, getSumFrom1To20(), getSumFrom1To21(), … 등)을 하나의 함수로 통합한 것이다.

마찬가지로 임의의 정수에서 임의의 정수까지의 합을 구하는, 매개변수가 2개인 함수 getSum(int from, int to)는 특정 정수에서 특정 정수까지의 합을 구하는, 매개변수가 없는 함수들(getSumFrom1To1(), getSumFrom1To2(), …, getSumFrom2To2(), getSumFrom2To3(), … 등)을 하나의 부프로그램으로 통합적으로 추상화한 것이다.

통합 추상화에서는 구현에서의 통합과 인터페이스에서의 통합 등 2가지 통합이 이루어지게 된다. 구현에서의 통합은 특정 대상에 대한 구현부분을 임의의 대상에 대해 수행될 수 있도록 일반적인 기능을 갖는 구현부분으로 변화시키는 것이다. 인터페이스에서의 통합은 특정 대상에 대한 인터페이스를 임의의 대상에 대한 인터페이스로 변화시키는 과정으로, 이때에 새로운 매개변수가 도입된다. 매개변수의 개수가 많을수록 부프로그램은 보다 일반적이며 포괄적인 기능을 수행할 수 있다. 함수 getSum(int from, int to)는 함수 getSumFom1to10()이나 getSumFrom1to(int n) 보다 매개변수가 많으며, 보다 일반적인 기능을 수행한다.

매개변수의 도입을 통하여 유사한 여러 부프로그램들을 하나의 부프로그램으로 통합하려는 시도는 기능이 유사하면서 매개변수의 타입이 다른 여러 부프로그램들을 하나의 부프로그램으로 통합하려고 하였다. 그 결과 포괄 함수(generic function)[6]가 도입되었으며, 프로그래밍 언어는 파라메트릭 다형성(parametric polymorphism)[6, 12]을 제공하게 되었다. 포괄 함수는 여러 타입의 매개변수에 대해 작동할 수 있는 함수로, 매개변수의 타입과 상관없이 특정한 종류의 작업의 수행한다. C++[20], Eiffel[14], Ada 등에서는 포괄 함수를 직접 선언할 수 있다.

여러 타입의 데이터들에 대해 데이터를 정렬하기 위해서는 각 타입의 데이터에 대한 정렬을 수행하는 부프로그램이 다음과 같이 타입마다 C 함수로 작성되어야 한다.

```
void sortInt(int data[ ], int n) { … }
                    /* int 배열에 대한 정렬 함수 */
void sortFloat(float data[ ], int n) { … }
                    /* float 배열에 대한 정렬 함수 */
void sortStudent(struct student[], int n) { … }
                    /* student 구조체 배열에 대한 정렬 함수 */
… /* 다른 타입의 배열에 대한 정렬 함수 */
```

이들 정렬 함수들은 매개변수와 함수 이름이 다르지만 배열에 저장된 여러 원소들을 정렬한다는 공통된 기능을 갖는다. 이들 함수들도 하나의 함수로 통합될 수 있다. 다음은 여러 정렬 함수를 통합한 타입 매개변수를 갖는 C++의 포괄 함수이다.

```
template <class T>
  class Sort {   /* Sort 클래스는 포괄 클래스로 선언됨 */
      public :
      …
         /* 타입 매개변수 T를 갖는 포괄 함수 sort() */
         void sort(const T data[ ], int n) {
            …   /* 타입 T의 배열 data에 대한 정렬의 구현 */
      }
  }
```

C++에서는 클래스 템플리트(class template)를 통하여 포괄 함수가 포함될 포괄 클래스를 선언한다. 선언된 포괄 클래스는 나중에 특정 클래스로 인스턴스화되며, 이 순간 포괄 함수의 타입 매개변수도 특정 타입으로 바인딩된다.

매개변수의 타입이 서로 다른 여러 부프로그램을 하나의 부프로그램으로 통합하는 것은 여러 부프로그램의 단순한 통합과 다른 점을 가진다. 첫째, 여러 부프로그램들의 구현에서 각 타입의 데이터에 대한 처리가 다르게 이루어지므로 통합된 함수의 구현에서는 여러 타입의 데이터에 대한 통합적 처리가 가능하여야 하며, 이를 위해서는 실행시간에 타입 검사가 필요할 수도 있다. 둘째, 여러 부프로그램의 서로 다른 타입의 매개변수들이 하나의 매개변수로 통합되어야 한다. 이를 위해서는 타입 변수(type variable)가 도입되어야 하며, 타입 변수를 구체적인 타입으로 변환하는 타입 변수의 타입 인스턴스화가 이루어져야 한다. 포괄 함수의 타입 변수를 인스턴스화하는 방법에 대한 연구는 [5]에 기술되어 있다.

본 논문에서는 동일한 타입의 여러 데이터를 매개변수로 통합하는 절차적 추상화를 동질적 절차적 추상화라고 하며, 서로 다른 타입의 여러 데이터를 타입 매개변수로 통합하는 절차적 추상화를 이질적 절차적 추상화라고 한다. 이러한 통합 추상화의 효용성은 단순 추상화에 비하여 매우 크다. 왜냐하면 많은 개수의 유사한 부프로그램들이 하나의 부프로그램으로 통합되기 때문이다. 이러한 통합된 부프로그램은 라이브러리를 형성한다. 이로 인하여 한 프로그램에서 필요로 하는 부프로그램의 개수가 현저히 감소되며, 프로그램 개발, 개발된 프로그램의 관리 및 수정 등이 용이하게 된다. 프로그래밍 역사에서 절차적 추상화의 결과로 대두된 모듈라 프로그래밍이 구조적 프로그래밍, 하향식 설계 등과 더불어 1970년대 이후 소프트웨어 개발의 핵심적인 방법론이 대두된 것은 부프로그램 속에 보이지 않는 통합 추상화라는 고수준의 추상화가 내포되어 있기 때문이다.

통합 추상화는 추상화에 대한 추상화이며 이는 고차원적인 추상화이다. 프로그래밍 작업에는 이러한 고차원적인 통합 추상화가 포함되게 된다. 초보자가 처음 Pascal, C 언어 등 고급 프로그래밍 언어를 배울 때 매개변수를 갖는 부프

로그램의 학습이 가장 난해한 부분으로 대두되는 것은 부프로그램 속에 내포되어 있는 통합성 때문일 것이다. 부프로그램을 명확히 이해하기 위해서는 부프로그램이란 단순한 추상화뿐 아니라 추상화에 대한 추상화도 포함한다라는 사실을 올바르게 이해하여야 할 것이다.

4. 절차적 추상화와 다형성

4.1 다형성의 기본 개념

추상화는 주관적인 인식과정이다. 그러므로 여러 사람에 의해 추상화가 이루어질 경우 같은 대상에 대하여 서로 다른 추상 결과물이 생성될 수 있으며, 또한 다른 대상에 대해 같은 추상 결과물이 생성될 수 있다. 자연어에서 하나의 대상이 서로 다른 단어로 표시되면 동의어가 되며, 서로 다른 여러 대상이 같은 단어로 표시되면 중의어가 된다. 이러한 동의어와 중의어는 단어 추상화에서 서로 다른 대상이 서로 다른 추상 결과물을 생성하지 않기 때문에 발생한다. 동의어와 중의어를 허용하는 자연어에서 추상화 대상과 추상 결과물간의 사상 또는 대응(correspondence)은 일대일(one-to-one)이 아니며 다대다(many-to-many)이다.

추상화의 대상과 결과물간의 사상을 일대일로 제한할 경우 추상 결과물은 유일성(uniqueness)를 가지며 식별성(identity)이 보장된다. 그러나 이로 인하여 추상 결과물의 개수가 증대되며 유일성을 만족시키기 위해 많은 노력이 필요하다. 특히 여러 사람에 의해 추상화가 이루어질 경우 유일성을 만족시키는 것은 매우 어렵다. 새로운 추상 결과물을 생성할 때마다 유일성 보장을 위해 기존의 모든 추상화를 확인하고 새로운 추상 결과물을 모든 참여자에게 통보하여 알려주어야 하는 번거로운 과정을 거쳐야 하기 때문이다. 그러므로 다수가 참여하여 여러 대상에 대하여 추상화 작업을 할 경우에는 추상화 대상과 추상 결과물의 일대일 사상보다 다대다 사상이 일반적이며, 그 결과 다형성을 갖는 추상 결과물이 생성된다. 절차적 추상화에서도 서로 다른 기능의 여러 구현이 하나의 인터페이스로 추상화될 수 있으므로 구현과 인터페이스간의 다대다 사상이 가능하며, 이 경우 다른 기능의 처리과정이 동일한 외부적 호출표현을 가질 수 있다. 기능이 다르면서 동일한 인터페이스를 갖는 이러한 부프로그램들을 다형성을 갖는 부프로그램이라고 말한다[6].

다형성은 하나가 여러 가지로 해석될 수 있는 특성을 말한다. 여러 의미를 나타내는 중의어가 문장에서 사용된 경우 중의어의 여러 의미 중 특정한 하나를 결정할 수 있는 단서와 방법 제공되어야 한다. 그렇지 않으면 그 단어가 사용된 문장은 애매성을 갖게된다. 예를 들어 '배가 하나 있다.'라는 문장이 주어지면 그 문장에서의 배가 과일의 한 종류인 배인지, 물위에 떠 있는 배인지, 아니면 또다른 의미의 배인지 파악할 수 없게 된다. 마찬가지로 다형성을 갖는 부프로그램들이 프로그램에서 사용되면 여러 부프로그램 중 어느 것을 수행하여야 할지를 결정하는 단서와 방법이 제공되어야 한다. 그렇지 않으면 호출된 부프로그램은 수행되어질 수 없게 된다.

4.2 절차적 추상화에서의 다형성

절차적 추상화는 실행 문장의 대체인 마크로에서 시작되어 타입 매개변수를 갖는 부프로그램으로 발전하였다. 이러한 과정에서 프로그래밍 언어들은 점차적으로 고수준의 절차적 추상화를 제공하여 왔으며, 다형성은 C++, Java 등 최근의 객체 지향 언어에서 기본적인 특징으로 제공되고 있다. 본 절에서는 절차적 추상화와 관련된 다형성에 대해 살펴본다.

마크로를 도입한 최초의 어셈블리 언어에서는 마크로 정의시 같은 이름의 마크로를 정의할 수 없도록 하였다. Fortran, Pascal, COBOL 등 초기의 고급 프로그래밍 언어는 극히 제한적인 다형성만을 제공하였다. 즉, 정수에 대한 산술 연산자와 실수에 대한 산술 연산자를 구분하지 않고 같은 연산자로 표현할 수 있게 하는 연산자 오버로딩(operator overloading)[10] 형태의 다형성만을 제공하였다. C 언어도 기본적으로 부프로그램에서의 다형성을 제공하지 않는다. 그러므로 하나의 C 프로그램에서 사용된 모든 함수의 함수 이름은 달라야 한다. 즉 C 함수 이름은 프로그램의 범위 내에서 유일하여야 한다. 이로 인하여 C 프로그램 개발자들은 프로그래머들이 사용하지 않는 이름의 라이브러리 함수명을 부여하기 위해 특수한 형태(함수명의 첫 부분을 '__'으로 시작하는 등)를 이용하기도 한다.

Java[1]는 여러 수준의 메소드 다형성을 제공하는 객체 지향 언어이다. Java에서 지원되는 메소드 다형성은 여러 클래스들에서 같은 이름의 메소드를 허용하는 다형성, 한 클래스 내에서 같은 메소드 이름을 사용할 수 있도록 허용하는 메소드 오버로딩(method overloading), 상위 클래스에 정의된 메소드와 같은 메소드 이름과 매개변수를 허용하는 메소드 오버라이딩(method overriding) 등이 있다[10]. Java에서는 여러 클래스에서 동일한 시그니처를 갖는 메소드를 허용하는 다형성에 대해 별도로 언급을 않는다. 이것은 여러 함수나 메소드에서 동일한 이름의 지역변수를 선언할 수 있는 것, 같은 이름의 클래스를 다른 패키지에서 선언하도록 하는 것 등이 다형성으로 언급되지 않는 것과 같은 이유이다. 이러한 비일관적인 서술은 프로그래밍 언어에서 다형성이 전체적 관점에서 체계적으로 분류되지 않았음을 의미한다.

다형성이 있는 메소드가 호출되면 가능한 여러 메소드 중에서 실제 수행될 하나의 메소드가 결정되어야 한다. 오버로딩 메소드가 호출되면 실매개변수의 개수 및 타입을

검사하여 수행될 메소드를 결정하여야 하는데, 이는 컴파일 타임에 타입 검사를 통하여 가능하므로 컴파일 타임 바인딩이 이루어진다. 그러나 오버라이딩 메소드가 호출되면 컴파일 타임에 수행될 메소드를 결정할 수 없다. 왜냐하면 수신자 객체의 타입에 해당되는 실제적 클래스를 확인하여야 하는데, 이 확인작업은 컴파일 시간에 이루어질 수 없기 때문이다. 그러므로 수행될 오버라이딩 메소드는 실행시간에 결정되어야 하며, 이로 인하여 메소드 오버라이딩을 지원하는 C++, Java 등은 실행시간 바인딩을 포함하는 프로그래밍 언어가 되었다. 메소드 오버라이딩은 실행시간에 수행될 메소드를 결정하는 실행시간 바인딩을 필요로 하므로 컴파일 타임 바인딩보다 성능을 저하시킨다. 그러나 프로그램의 대형화와 신속한 버전 업그레이드의 중요성이 강조되는 최근의 프로그램 개발 상황에서 메소드 오버라이딩은 클래스의 확장성과 재사용성을 증대시키므로 필수적인 다형성이다. 다형성에 관련된 보다 자세한 사항은 [5, 6, 12]에서 찾아볼 수 있다.

5. 결 론

현재 절차적 추상화는 타입 매개변수를 갖는 부프로그램을 가능하게 하며 객체 지향 패러다임의 상속과 결합하여 메소드 오버라이딩을 허용하는 고수준의 다형성을 제공하고 있다. 이로 인하여 상황에 따라 동적 바인딩을 수행하는 실행시간 시스템을 통하여 프로그램은 자동적으로 행동을 결정하고 수행하는 지능적인 수준에 이르고 있다.

본 논문에서는 부프로그램에 적용되어지는 절차적 추상화를 체계적으로 분석하고 이에 대한 분류를 실시하였다. 절차적 추상화는 구현 방법에 따라 대체 추상화와 호출 추상화, 추상화의 수준에 따라 단순 추상화와 통합 추상화로 나누어지며, 통합 추상화는 다시 동질적 통합 추상화와 이질적 통합 추상화로 나누어진다. 한편 추상화의 결과로 동일한 이름을 갖는 추상화가 필연적이며 이로 인하여 하나의 이름이 여러 다른 절차적 추상화를 표상하게 되므로 절차적 추상화에서는 다형성이 중요한 이슈가 됨을 살펴보았다.

앞으로의 연구로, 프로그래밍 언어에서 제공되는 일반적인 추상화에 대하여 고찰하여 프로그래밍 언어에서의 추상화를 새롭게 정의하고 일관성 있는 분류체계를 형성하여야 할 것이다. 또한 현재 연산자 오버로딩, 메소드 오버로딩, 메소드 오버라이딩 등 부프로그램 중심으로 다형성이 분류되어 있다. 이를 확장하여 프로그래밍 언어에서의 일반적인 다형성의 정의를 확립하고 분류체계도 형성하여야 할 것이다.

참 고 문 헌

[1] M. Abadi, L. Cardelli and P. L. Curien, "Formal Parametric Polymorphism," Proceedings of the 20th ACM Symposium on Principles of Programming Languages, 1993.

[2] K. Arnold and J. Gosling, The Java Programming language, Addison-Wesley, 1996.

[3] The Bible Societies, The Holy Bible Authorized Version, The Bible Societies, 1972.

[4] A. Bogida, J. Mylopoulos and H. Wong, "Generalization/Specialization as a Basis for Software Specification," On Conceptual Modelling : Perspectives from Artificial Intelligence, Databases, and Programming Languages, M. L. Brodie, J. Mylopoulos and J. W. Schmidt, Eds, Springer-Verlag, 1984.

[5] P. Canning, W. Cook, W. Hill, W. Olthoff and J. C. Mitchell, "F-bounded Polymorphism for Object-Oriented Programming," Proceedings of the 4th Functional Programming Languages and Computer Architecture, 1989.

[6] L. Cardelli and P. Wegner, "On Understanding Types, Data Abstraction, and Polymorphism," ACM Computing Survey, Vol.17, No.4, Dec., 1985.

[7] O. J. Dahl and K. Nygaard, "Simula - An Algol-based Simulation Language," Comm. ACM, No.9. Vol.9, Sep., 1966.

[8] S. Danforth and C. Tomlinson, "Type Theories and Object-Oriented Programming," ACM Computing Surveys, Vol.20, No.1, Mar., 1988.

[9] http://www.encyber.com/.

[10] T. Korson and J. D. McGregor, "Understanding Object-Oriented : a Unifying Paradigm," Comm. of ACM, Vol.33, No.9, Sep., 1990.

[11] B. W. Kenighan, D. M. Ritchie, The C Programming Language, Prentice-Hall, 1988.

[12] Q. Ma, "Parametricity as Subtyping," Proceedings of the 19th ACM Symposium on Principles of Programming Languages, 1992.

[13] M. N. Mattos, "Abstraction Concepts : the Basis for Knowledge Modeling," Proc. of Conf. on Entity-Relationship Approach, 1988.

[14] B. Mayer, Eiffel : The Language, Prentice-Hall, 1992.

[15] O. Nierstrasz, "A Survey of Object-Oriented Concepts," in Object-Oriented Concepts, Databases, and Application, ACM Press, 1989.

[16] R. W. Sebesta, Concepts of Programming Languages, The Benjamin/Cummings Publishing Company, 1992.

[17] R. Sethi, "Programming Languages," Concepts and Constructs, Addison-Wesley, 1989.

[18] A. Snyder, "Encapsulation and Inheritance in Object-Oriented Programming language," Proceedings of the First ACM Conference on Object-oriented programming Systems, Languages, and Applications, 1986.

[19] J. M. Smith and D. C. P. Smith, "Database Abstraction : Aggregation and Generalization," ACM TODS, Vol.2. No.2, June, 1977.
[20] B. Stroustrup, The C++ programming Language, Addison Wesley, 1993.
[21] A. Taivalaari, "On the Notion of Inheritance," ACM Computing Surveys, Vol.28, No.3, Sep., 1996.
[22] J. D. Ullman and J. Widom, A First Course in Database Systems, prentice-Hall, 1997.
[23] P. Wegner, "Classification in Object-oriented Systems," ACM SIGPLAN Notices, Vol.21, No.10, Oct., 1986.

김 성 기

e-mail : skkim@hanshin.ac.kr
1983년 서울대학교 컴퓨터공학과(공학사)
1985년 서울대학교 대학원 컴퓨터학과 (공학석사)
1992년 서울대학교 대학원 컴퓨터학과 (공학박사)
1985년~1986년 삼성전자 연구원
1993년~현재 한신대학교 컴퓨터학과 부교수
관심분야 : 데이타베이스 시스템, 객체 지향 시스템