추상화의 분류

김 성 기†

요 약

프로그래밍 언어에서의 추상화는 변수, 함수, 복합 데이터 등에서부터 추상 데이터 타입, 클래스, 다형성 등에 이르기까지 광범위하게 적용되는 중요한 개념이다. 그러나 추상화 개념의 명확성과 통일성 부족으로 인하여 추상화는 다소 애매모호한 특성을 갖는 용어로 인식되어지고 있으며, 관점에 따라 다르게 설명되는 혼란스러움을 경험하고 있다. 본 논문에서는 여러 프로그래밍 언어에 나타난 추상화에 대한 체계적인 분석을 실시하여 추상화의 분류를 제안한다. 추상화는 추상물의 생성 방법에 따라 사상 추상화, 묶음 추상화, 통합 추상화, 확장 추상화 등 4가지 범주로 분류된다. 또한 함수, 추상 데이터 타입, 객체, 캡슐화, 클래스 등 추상화를 기반으로 하여 형성된 여러 개념들이 추상화의 관점에서 고찰된다. 이러한 체계적인 추상화의 분석과 분류를 통하여 지금까지 개별적이며 서로 다른 형태의 추상화로 취급된 여러 추상화 양상이 하나의 통일된 개념으로 설명될 수 있을 것이며, 추상화의 의미, 필요성, 중요성에 대하여 보다 깊은 이해가 가능할 것이다.

Taxonomy of Abstraction

Sung-Ki Kim†

ABSTRACT

Abstraction is an important concept applied widely to variables, functions, complex data, abstract data types, classes and polymorphism in programming languages. However, the concept of abstraction has been considered as ambiguous and explained differently because it is not defined clearly and uniformly. In this paper, we analyse many aspects of abstraction in programming languages, and propose the taxonomy of abstraction. We classify abstraction according to the mechanism of formation into 4 categories such as mapping abstraction, bundling abstraction, integrating abstraction and extending abstraction. We also consider many concepts related closely to abstraction such as functions, abstract data types, objects, encapsulation and classes in the view of abstraction. These analysis and consideration will make it possible to explain uniformly various aspects of abstraction which have been treated individually and differently, and to understand the meanings, necessity and importance of abstraction more intensively.

키워드 : 추상화(Abstraction), 절차적 추상화(Procedural Abstraction), 추상 데이터 타입(Abstract Data Type), 캡슐화(Encapsulation), 클래스(Class), 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 요소이라고 할 수 있다.

추상화(abstraction)는 기존의 것에 대하여 지금까지 존재하지 않았던 기호, 명칭, 개념, 용어 등 추상적인 것을 새로이 창조하는 것이다. 인간은 여러 사물에 대해 이름을 부여한 명사, 사물의 상태와 행동을 나타내는 형용사와 동사 등의 단어를 통하여 추상화된 외부 세계를 인식하고 이를 표현할 수 있고 전달할 수 있다. 언어의 근간을 이루는 여러 형태의 추상화는 인간을 창조적이며 고차원적인 존재가 되게 만든 가장 중요한 요인이다.

추상적인 대상의 특성을 다루는 수학에서 추상화는 특히 중요하다. 0, 1, 2 등의 숫자, +, - 등의 연산자, 점, 선, 삼각형 등에 대한 정의, 피타고라스 정리, 괴델의 불완전성의 정리 등의 여러 정리들은 모두가 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 체

※ 이 논문은 2003년도 한신대학교 특별연구비 지원에 의하여 연구되었음.
† 정 회 원 : 한신대학교 컴퓨터학과 교수
논문접수 : 2003년 9월 22일, 심사완료 : 2003년 12월 26일

계의 코드 하나하나는 추상적 산물이다. 명령과 코드로부터 시작된 프로그래밍 언어에서는 변수, 타입 등의 보다 추상적인 개념들을 수용하였다. 타입 추상화와 변수 추상화가 어셈블러나 컴파일러에 의해 제공되기 시작한 이래 프로그래머들은 하드웨어의 상세함을 고려하지 않은, 의미를 많이 함축하는 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 추상화는 전체적인 관점에서 통일성 있게 정의되거나 설명되지 못하고 있다. 예를 들어, 처리에 필요한 일련의 과정들을 추상화하여 이를 간단하게 호출될 수 있는 형태로 표현하는 절차적 추상화, 연관된 데이터와 연산들을 하나의 단위인 추상 데이터 객체로 추상화하는 데이터 추상화, 상속, 캡슐화, 다형성의 개념들이 혼재된 클래스 등 추상화를 기반으로 형성된 여러 개념들은 추상화의 큰 틀에서 명확하게 구분되지 않는다. 그 결과 추상화라는 개념은 다소 애매모호한 특성을 갖는 용어로 인식되어지고 있다. 또한 추상화와 연관된 여러 개념과 용어도 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 사용되는 추상화에 대하여 정확한 분석이나 체계적 분류가 이루어지지 않았기 때문이다.

절차적 추상화에 대한 체계적인 분류는 [24]에서 시도되었다. [24]에서는 절차적 추상화를 구현방법에 따라 대체 추상화와 호출 추상화로, 추상화 수준에 따라 단순 추상화와 통합 추상화로 분류하였으며, 통합 추상화는 다시 통합 대상에 따라 동질적 통합 추상화와 이질적 통합 추상화로 분류하였으며, 절차적 추상화와 관련된 다형성에 대해서도 분석하였다. 또한 [25]에서는 프로그래밍 언어에 나타난 여러 추상화의 양상을 발전적인 관점에서 살펴보았다.

본 논문은 [24]와 [25]의 후속작업으로서, FORTRAN, C, C++, Java 등 여러 프로그래밍 언어[2, 11, 17, 20]에 나타난 추상화에 대한 체계적인 분석을 실시하여 추상화의 분류체계를 새롭게 제안한다. 제안된 분류체계에서는 추상화를 사상 추상화, 묶음 추상화, 통합 추상화, 확장 추상화라는 4개의 범주로 분류하고 각 범주에 대한 세부 분류를 제시한다. 이러한 체계적인 추상화의 분석과 분류를 통하여 지금까지 절차적 추상화, 복합 데이터, 데이터 추상화, 다형성 등 개별적이며 서로 다른 형태의 추상화로 취급된 다양한 형태의 추상화를 하나의 통일된 개념으로 통합할 수 있을 것이다. 또한 이러한 분석과 분류를 통하여 추상화의 의미, 필요성, 중요성에 대하여 체계적인 이해도 가능할 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 먼저 추상화의 정의를 살펴보고, 3장에서는 프로그래밍 언어에서의 추상화 양상을 정리한다. 4장에서는 추상화에 대한 분류기준을 설정하여 분류체계를 제시하고, 5장에서는 추상 데이터 타입, 캡슐화, 객체, 클래스 등의 개념들에 대해 제시된 추상화 분류를 적용해 고찰한다. 6장에서는 결론 및 앞으로의 연구 방향을 제시한다.

2. 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 "대상으로서의 소여 전체로부터 특정성질이나 공통징표를 분리하고, 골라내는 정신작용. 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."라고 정의되어 있다.

이러한 추상화의 일반적 정의에서는 추상화는 주어진 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.

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] 등이 있다.

Java[2]에는 변수, 연산자, 타입, 메소드, 배열 등의 기본적인 추상적 요소뿐 아니라 상태와 행위를 캡슐화한 객체, 이를 통합한 클래스 및 클래스를 확장하는 서브 클래스, 클래스의 집합인 패키지 등 고차원적인 추상적 요소들을 제공한다. 또한 이와 더불어 범위 규칙(scope rule), 절차적 추상화, 데이터 추상화, 추상 데이터 타입, 객체 추상화, 상속, 다형성 등의 추상화와 연관된 개념을 내포하고 있다. C++[20]에서는 파라메트릭 다형성을 제공하는 포괄 클래스도 지원되고 있다. 이러한 고도의 추상화를 지원하는 프로그래밍 언어가 개발되고 새로운 소프트웨어 개발 방법론이

제시됨에 따라 컴퓨터는 단순한 계산이나 데이터의 처리라는 관점을 뛰어넘어 현실세계를 모델링하며 시뮬레이션할 수 있는 추상적인 도구로 발전하였다.

추상화에서 비롯되는 심각한 문제는 추상화의 결과가 정확히 무엇을 추상하였는가, 즉 추상의 실제적 대상이 무엇이며, 분리되고 골라진 것이 무엇인가가 명확히 정의되지 않는다는 것이다. 그로 인하여 신, 사랑, 우주 등 추상적 용어에 대하여 주관적 해석이 가능하며, 해석을 달리 하는 사람들 간에는 의견 차이와 오해가 발생하기도 한다. 이에 반하여 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 말한다. 더욱 중요한 특징이 추상화 대상과 추상화 결과간의 상호 대체성이다. 상호 대체성은 추상화 대상과 추상화 결과가 명확히 정의되므로 추상화 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. 추상 결과의 명료성과 상호 대체 가능성을 갖는 추상화는 정형적으로 다음과 같이 정의된다.

[정의 1] 추상화는 추상화의 대상 O와 추상화의 결과물 A의 순서쌍 (O, A)을 정의하는 것이며, 추상화의 대상 O를 추상화 내포(intension), 추상화의 결과물 A를 추상화 표상(denotation), (O, A)를 추상물(abstract)이라 한다.

변수 추상화의 경우 추상화 내포는 기억장소의 주소, 저장된 값, 기억장소의 크기, 값의 타입 등이며, 추상화 표상은 변수 이름이다. 절차적 추상화의 경우 추상화 내포는 부프로그램의 처리과정을 나타내는 구현(implementation)이며, 추상화 표상은 부프로그램의 기능을 나타내는 인터페이스(interface)이다.

3. 추상화 개념의 확장과 변화

기억장소에 데이터와 명령을 직접 저장하였던 초기의 컴퓨터에서 프로그래머들은 주소와 데이터 그리고 기계어 명령들을 모두 이진수 패턴으로 직접 작성하였고 이를 직접 입력하였다. 주소, 데이터 그리고 명령을 기호로 나타내는 어셈블리어가 등장하고, 어셈블리 프로그램의 기호를 이진수로 바꾸어주는 어셈블러(assembler)가 개발되어 어셈블리어로 프로그래밍을 하게 됨에 따라 프로그래머의 생산성이 그 전에 비하여 크게 향상되었고, 변환 오류의 감소, 빠른 번역, 용이한 프로그램의 변경 등이 가능하게 되었다.

어셈블리어에서 사용된 추상화로는, 이진수 주소를 변수 이름으로 추상화하고, 이진수의 명령을 연산자 이름으로 추상화하고, 이진수 데이터를 10진수 또는 다른 형태의 보다 친숙한 표현으로 추상화하는 등 이진수를 이해하기 쉬운 기호로 대체하는 추상화가 주종을 이룬다. 그러나 기호와 이진수간의 일대일 사상은 프로그래머에게 여전히 하드웨어 종속적인 많은 코딩 지식을 요구하였다.

특정 기능을 수행하는 여러 문장들을 하나의 단위인 부프로그램의 인터페이스로 추상화하고 이를 간단히 호출하게 하는 절차적 추상화는 프로그래밍 언어에 도입된 추상화 중 가장 중요한 위치를 차지한다. 하나의 프로그램을 여러 기능적 단위로 분해할 수 있게 되었으며, 그 결과 구조적 프로그래밍, 하향식 설계, 코드 재사용 등 향상된 프로그래밍 기법을 이용한 다양한 소프트웨어가 개발되었다.

또한 여러 연관된 데이터들을 하나의 단위로 표현하고 전체적으로 취급할 수 있도록 하는 배열, 레코드, 사용자 정의 타입 등 복합 데이터도 ALGOL 60과 Pascal에서 지원되기 시작했다. 복합 데이터에 적용된 추상화는 여러 요소들을 하나의 전체로 구성할 수 있게 하고, 또다시 전체를 통하여 각 요소를 접근할 수 있게 하는 추상화로서, 이는 기호 대체 추상화 및 절차적 추상화와는 다른 양상의 추상화이다. 한편 부프로그램과 복합 데이터에서는 여러 요소들을 하나의 단위로 표현하여 전체를 하나로 취급할 수 있도록 하는 양상의 추상화가 이루어졌다. 이러한 양상의 추상화는 복잡한 대상을 간단한 형태로 표현하게 하는 추상화의 본격적인 양상이라 할 수 있다. 이보다 진보된 양상의 추상화는 여러 값과 연산을 통합시키는 타입에서 나타난다.

타입은 사용과 행동에 따라 값을 분류하기 위해 처음에 도입되었으며, 컴퓨팅 과정에서 동적으로 생기는 값들을 특징짓는 수단으로 사용되었다[6]. 추상화의 관점에서 타입이란 공통된 특징을 갖는 값들의 집합을 추상화한 것이라 할 수 있다. 그리하여 타입이 정의되면 공통된 특징은 값의 제약조건이 되며 타입의 한 값은 제약조건을 만족해야 한다. 하나의 타입이 존재하기 위해서는 공통된 특성을 갖는 값들의 집합이 먼저 상정되어야 하고 그 값들에 대한 추상화의 결과 새로운 타입이 탄생된다.

프로그램의 설계와 구축에서 추상화의 중요성이 폭 넓게 인식되었을 때 추상 데이터 타입(abstract data type : ADT)의 개념이 도입되어 데이터 추상화가 확립되었다. 추상 데이터 타입은 데이터 객체의 표현에 관한 구조정보와 그 객체를 조작하는 구현정보를 추상화할 수 있도록 기존의 타입 개념을 확장하였다[8]. 데이터 추상화는 추상 데이터 타입을 이용하여 연관된 데이터와 연산들을 하나의 단위인 추상 데이터 객체로 만들게 하며, 여러 데이터와 여러 연산을 하나의 단위로 다룰 수 있게 한다. 데이터 추상화가 제공되면서 데이터 객체를 구성하는 요소인 데이터와 연산에 대한 선택적인 접근이 가능하게 된 것이 큰 특징이며, 이러한 구성요소들의 선택적 은폐 및 공개 기능이 캡슐화이다[10]. 데이터 추상화에서 이질적인 요소인 데이터와 연산을 하나의 단위로 추상화하였다는 것과 요소들에 대해 선택적인 접근을 가능하게 하였다는 것은 배열, 레코드, 부프로그램, 타입 등에 적용된 추상화에서는 고려되지 않았던 새로운 양상이며, 이는 데이터 추상화가 새로운 수준의 진보된 개념으로 확장됨을 말한다.

절차적 추상화, 타입에서의 추상화, 복합 데이터에서의

추상화, 데이터 추상화 등 여러 추상화에 나타난 공통적인 양상은 여러 요소들에 대한 새로운 추상적 단위를 형성한다는 것이다. 새로운 추상적 단위의 필요성은 전체를 구성하는 요소들의 개수와 비례하여 증가한다. 큰 프로그램이 개발됨에 따라 문장의 개수가 많아지자 여러 문장들은 호출의 단위인 부프로그램으로 모듈화되어 작성되었다. 사용되는 데이터의 개수가 많아짐에 따라 연관된 여러 데이터는 하나의 복합 타입과 복합 데이터로 다룰 수 있게 되었다. 그런데 또다시 부프로그램의 개수와 복합 데이터의 개수가 많아짐에 따라 관련성이 있는 타입과 부프로그램들을 하나의 단위로 추상화할 필요성이 대두되었고, 데이터 추상화를 통하여 여러 종류의 데이터들과 부프로그램들을 하나의 추상 데이터 객체로 묶을 수 있게 되었다.

매개변수의 타입이 다르지만 동일한 기능을 수행하는 여러 부프로그램들을 하나의 부프로그램으로 통합하려는 노력의 결과 포괄 함수(generic function)[6]가 도입되었으며, 프로그래밍 언어는 파라메트릭 다형성(parametric polymorphism)을 제공하게 되었다. 포괄 함수는 여러 타입의 매개변수에 대해 작동할 수 있는 함수로, 매개변수의 타입과 상관없이 특정한 작업을 수행한다. C++, Eiffel[14], Ada 등에서는 포괄 함수를 직접 선언할 수 있으며, C++에서는 클래스 템플리트(class template)를 통하여 포괄 클래스(generic class)를 선언할 수 있다.

추상화의 발전단계적인 측면에서 고찰하면 다음 단계의 진보된 추상화는 여러 추상 데이터 타입들을 하나의 단위로 만드는 것이다. 한 프로그램에서 사용되는 추상 데이터 타입의 개수가 많아지게 됨에 따라 추상 데이터 타입들에 대한 구조화 필요성이 제기되었고, 객체 지향 패러다임에서 그 모습을 드러내었다. 객체 지향 패러다임에서의 가장 핵심적인 추상화 기법은 추상 데이터 타입인 클래스들의 공통점을 통합하여 상위 클래스로 추상화하는 것이다.

그런데 상위 클래스로의 추상화는 프로그램의 설계단계에서 이루어지는, 보이지 않는 과정인 반면 실제적인 프로그램의 구현단계에서는 이미 정의된 상위 클래스를 확장한 하위 클래스가 상위 클래스의 상태와 행위를 상속받는 추상화 형태로 표출되었다. 겉으로 표출된 추상화 기법이 내재된 추상화 기법과 다른 양상을 띠게 되므로 객체 지향 프로그래밍에서는 하위 클래스, 상속, 캡슐화, 다형성 등 결과적인 측면이 강조되었다. 한편, 모든 것은 객체라는 포괄적 선언, 클래스가 추상 데이터 타입, 객체 생산소, 객체 저장소 등의 다양한 역할 수행, 상위 클래스에서 하위 클래스로의 다중 상속 및 이름의 중복[21] 등은 객체 지향 패러다임의 정형적인 정의를 사실상 어렵게 하였다. 프로그래밍 언어에 나타난 추상화의 양상은 [25]에 보다 자세히 기술되어 있다.

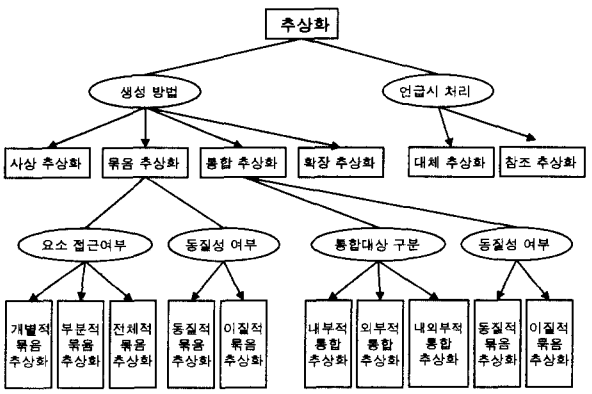
4. 추상화의 분류

본 장에서는 프로그래밍 언어에 나타난 추상화 양상에 기반하여 추상화에 대한 전체적인 분류체계를 제시한다. 추상화는 추상물의 생성 방법에 따라 크게 사상 추상화(mapping abstraction), 묶음 추상화(bundling abstraction), 통합 추상화(integrating abstraction), 그리고 확장 추상화(extending abstraction)로 나누어진다. 한편, 추상화는 추상화 표상의 언급에 대한 처리 방법에 따라 대체 추상화(substitute abstraction)와 참조 추상화(reference abstraction)로 나누어진다.

사상 추상화는 하나의 추상화 대상에 대해 추상화 표상을 새로이 부여하는 것으로, 가장 기본적인 추상화이다. 묶음 추상화는 추상화 대상이 되는 여러 요소들을 하나로 묶은 후 이를 대표하는 추상화 표상을 생성하는 것으로, 묶여진 요소들에 대한 접근 여부에 따라 개별적 묶음 추상화(individually bundling abstraction), 부분적 묶음 추상화(partially bundling abstraction), 전체적 묶음 추상화(totally bundling abstraction)로 나누어진다. 또한 묶여지는 요소들의 동질성 여부에 따라 동질적 묶음 추상화(homogeneously bundling abstraction)와 이질적 묶음 추상화(heterogeneously bundling abstraction)로 나누어진다.

통합 추상화는 여러 추상물들을 통합하여 새로운 추상물 내포와 표상을 생성하는 것으로, 통합 대상이 내부 요소인 내포인가 외부 요소인 표상인가에 따라 내부적 통합 추상화(internally integrating abstraction), 외부적 통합 추상화(externally integrating abstraction), 내외부적 통합 추상화(internally and externally integrating abstraction)로 나누어지고, 통합 대상이 동질적인가 이질적인가 따라 동질적 통합 추상화(homogeneously integrating abstraction)와 이질적 통합 추상화(heterogeneously integrating abstraction)로 나누어진다. 확장 추상화는 기존의 추상물로부터 새로운 요소가 첨가된 추상물들을 생성하는 것이다.

대체 추상화는 추상화 표상이 언급될 때마다 추상화 표상이 추상화 내포로 대체되어지는 것이며, 참조 추상화는 추상화 표상이 언급되면 추상화 내포가 간접적으로 참조되는 것이다. (그림 1)은 추상화의 분류도를 나타낸다.



(그림 1) 추상화의 분류도

4.1 사상 추상화

사상 추상화는 추상화의 대상 O를 추상화 표상 A로 사상하여 추상물 (O, A)를 생성한다. 그 결과 추상화의 대상

O는 추상화 내포로 전환되고, 추상화 표상 A는 O를 나타내게 된다. 추상화 내포를 보다 익숙하거나, 의미를 잘 나타내거나, 보다 고수준인 표상으로 대응시키는 사상 추상화는 가장 기초적인 추상화로서, 어셈블리 언어에서 많이 볼 수 있는 이진수 명령어 코드를 ADD, SUB 등의 기호로 표시하는 것, 이진수 정수를 10진수 정수로 표시하는 것 등이 대표적인 예이다. 또한 ASCII, EBCDIC, Unicode 등 문자 코드 체계의 코드 하나하나는 사상 추상화의 한 예이다.

4.2 묶음 추상화

묶음 추상화는 추상화의 대상인 여러 요소들을 하나의 단위로 묶고 추상화하여 새로운 추상화 표상을 생성한다. 즉, 요소들 $<E_1, E_2, \cdots, E_n>$과 요소에 대한 요소 식별자들 $<I_1, I_2, \cdots, I_n>$을 추상화 표상 A로 추상화하여 추상물 $([I_1 : E_1, I_2 : E_2, \cdots, I_n : E_n], A)$를 생성한다[1]. 묶음 추상화의 표상은 첫째, 전체를 대표하면서 간단히 전체를 언급하고, 둘째, 요소들 하나하나를 겉으로 드러내지 않으며, 셋째, 여러 요소의 변화와 관계없이 전체의 식별성을 유지하는 기능을 가진다.

묶음 추상화는 전체적 관점에서 추상화 내포의 모든 요소를 접근할 수 있는 개별적 묶음 추상화, 일부 요소는 접근할 수 있고 나머지 요소는 접근할 수 없는 부분적 묶음 추상화, 어느 요소도 접근할 수 없는 전체적 묶음 추상화로 나누어진다. 한편 묶음 추상화는 묶여지는 요소들이 동일한 특성을 갖는 동질적 묶음 추상화와 요소들이 서로 다른 특성을 갖는 이질적 묶음 추상화로 나누어진다.

4.2.1 개별적, 부분적, 전체적 묶음 추상화

추상화된 요소 모두를 개별적으로 접근할 수 있는 개별적 묶음 추상화가 적용된 대표적인 것은 배열과 레코드이다. 같은 타입의 여러 원소들을 묶어 배열이름으로 추상화시킨 배열에는 묶음 추상화가 적용된다. 그런데 배열이름과 인덱스를 이용하여 배열의 각 원소를 모두 접근할 수 있으므로 배열에서의 추상화는 개별적 묶음 추상화이다. 여러 멤버들을 묶어 레코드 이름으로 추상화시킨 레코드도 묶음 추상화가 적용되며, 레코드 이름과 멤버이름을 통하여 모든 멤버를 접근할 수 있으므로 역시 레코드에서의 추상화도 개별적 묶음 추상화이다. 그 외에도 저장장소의 주소, 길이, 저장될 값의 타입, 저장되는 값 등의 여러 요소를 하나로 묶고 추상화시킨 변수에서의 추상화도 개별적 묶음 추상화이다[2].

요소에 대한 일부분만 접근 가능하게 하는 부분적 묶음 추상화가 적용된 것은 객체 지향 패러다임에서의 객체[10]이다. 객체는 여러 상태와 여러 행위를 하나로 묶어 객체 식별자로 추상화한 것인데, 객체의 특정 상태나 행위는 개방되어 접근할 수 있는 반면 어떤 상태나 행위는 은폐되어 접근이 금지되므로 부분적 묶음 추상화가 적용된다[3]. 부분적 묶음 추상화는 결과적으로 객체의 내부적 상태와 연산을 선택적으로 은폐시키거나 공개할 수 있도록 하는 캡슐화 능력을 제공한다.

추상화된 요소들이 전체의 관점에서 모두 접근되지 못하는 전체적 묶음 추상화가 적용된 것은 함수, 메소드 등이다. 함수는 함수의 인터페이스와 구현으로 구성되는 프로그램의 요소로서, 추상화의 관점에서 함수는 특정 기능을 수행하기 위한 여러 문장을 묶어 인터페이스로 추상화시킨 것이다. 추상화된 함수의 구현에서 개별적인 요소인 문장의 참조는 불가능하므로 함수에서의 추상화는 전체적 묶음 추상화이다.

개별적 묶음 추상화 또는 부분적 묶음 추상화에서 추상화되는 요소에 대한 요소 식별자가 명시적으로 제공되든지 시스템에 의해 자동적으로 제공되어야 한다. 레코드와 객체의 경우 이를 선언할 때 요소 식별자가 명시적으로 주어져야 하며[4], 배열의 경우 요소 식별자는 자동적으로 제공된다.

4.2.2 동질적, 이질적 묶음 추상화

동질적 묶음 추상화가 적용된 것은 배열, 집합 등이며, 이질적 묶음 추상화가 적용된 것은 레코드, 객체, 함수이다. 배열은 동일한 타입의 원소들을 묶어 배열 이름으로 표상하게 하므로 동질적 묶음 추상화가 적용된다. 한편, 레코드는 서로 상이한 타입의 멤버들을 묶어 레코드 이름으로 추상화하며, 함수의 구현도 여러 종류의 문장들로 구성되므로 레코드와 함수에는 이질적 묶음 추상화가 적용된다.

4.2.3 묶음 추상화의 반복적 적용

묶음 추상화는 여러 요소들을 결합하여 새로운 수준의 단위를 형성한다. 묶음 추상화의 추상물은 또 다시 다른 묶음 추상화의 요소가 될 수 있으며, 다차원 배열, 레코드 배열, 객체를 멤버변수로 갖는 객체 등이 이러한 경우이다. 묶음 추상화가 반복되어 적용될 경우 복잡한 구조의 대상을 표상하는 추상물이 생성되며, 결과적으로 복잡한 세계나 상황을 조직적이며 체계적으로 표현할 수 있게 된다.

4.3 통합 추상화

통합 추상화는 사상 추상화나 묶음 추상화와 달리 공통점이 있는 여러 추상물들을 통합하여 새로운 통합 추상물을 생성하는 고차원적 추상화로서, 여러 추상물들 $<(O_1, A_1), (O_2, A_2) \cdots, (O_n, A_n)>$으로부터 통합된 하나의 추상물 (O, A)를 생성한다. 통합 추상화가 이루어지면 보다 고수준의 추상물이 생성되며, 통합 대상들이 갖는 공통된 특성은 통합 대상들로부터 제거되어 통합된 추상물에 의해 관리되어질 수 있다.

통합 추상화를 위해서는 통합 대상들에 대한 비교분석이 필수적이며, 표상들의 표현양식과 내포들의 구조 및 표현양

1) 요소의 식별자는 없을 수도 있다.

2) 프로그램에서 변수는 값을 참조하고 저장할 수 있도록 제한되지만 주소, 타입, 길이 등 변수와 관련된 다른 요소들은 궁극적으로 컴파일러나 실행 시스템에 의해 접근되어진다.

3) 변수, 배열, 레코드를 선언할 때 표상을 나타내는 명칭이 주어진다. 그러나 Java에서처럼 클래스를 통하여 생성되는 객체는 객체의 표상으로 명칭이 주어지지 않는 대신 객체 식별자가 내부적으로 생성되며, 객체 식별자를 통하여 전체적인 참조 및 각 요소의 접근이 이루어진다.

4) 대부분의 프로그래밍 언어에서 레코드나 객체를 직접 생성하지 않고 타입이나 클래스를 통해 생성하기 때문에 요소 식별자의 제공이 레코드나 객체 그 자체와는 분리되어 있는 것처럼 보인다.

식뿐 아니라 이들의 의미도 고려되어야 한다. 예를 들어, 10!을 구하는 함수의 표상인 인터페이스를 getFact10()라 하고, 15!을 구하는 함수의 인터페이스를 factorial15()라 한다고 하자. 이 두 함수의 표상(즉, 함수의 인터페이스)은 외관상으로 공통점이 없는 것으로 보인다. 그러나 두 함수는 의미적으로 '특정 정수의 팩토리알을 구한다'라는 공통점을 가진다. 이러한 공통된 의미를 잘 나타내도록 이 두 함수의 표상은 get_factorial_10()과 get_factorial_15()로 통일성 있게 변화되어야 한다[5]. 의미적으로 통일된 형식으로 변화된 표상들 또는 내포들에 대해 통합 추상화가 이루어질 수 있다.

통합 추상화는 추상물들의 표상에 대해 통합이 이루어지는 외부적 통합 추상화, 추상물들의 내포에 대해 통합이 이루어지는 내부적 통합 추상화, 표상과 내포 모두에 대해 통합이 이루어지는 내외부적 통합 추상화로 나누어진다. 한편, 통합 추상화는 구조 또는 타입이 동일한 대상에 대해 통합이 이루어지는 동질적 통합 추상화와 구조 또는 타입이 상이한 대상에 대해 통합이 이루어지는 이질적 통합 추상화로 나누어진다.

4.3.1 내부적, 외부적, 내외부적 통합 추상화

외부적 통합 추상화는 통합 대상인 추상물들의 표상들에 대해 통합이 이루어진다. 외부적 통합 추상화에서는 통합할 추상물들의 내부 표현인 내포는 고려되지 않고 표상들의 의미만을 고려하여 추상화가 이루어지며, 의미적으로 보다 포괄적인 새로운 추상물이 생성된다. 외부적 통합 추상화의 예로서는 Java에서 똑같은 객체를 복사하기 위해서 사용되는 Cloneable 인터페이스[6] 또는 스레드를 위한 Runnable 인터페이스이다. 이들 인터페이스는 관련이 없는 여러 클래스들에 통합하여 Cloneable 또는 Runnable처럼 사용될 수 있게 한다.

한편, 내부적 통합 추상화는 추상물들의 내포에 대한 통합이 이루어지며, 이는 내포의 구조 및 표현양식이 고려되어야 한다. 내부적 통합 추상화가 적용된 것은 타입 및 클래스, 상위 클래스 등이다. 클래스는 그 클래스의 모든 객체들이 공통적으로 가지는 내부적 특성을 기술한 것으로, 추상화의 관점에서 클래스란 객체에 관한 공통된 내부 정보를 클래스 이름으로 추상화한 것이다. 클래스에 의해 개별 객체의 특성이 명시되면 객체마다 객체의 특성을 유지할 필요가 없으며, 객체를 생성할 때마다 객체의 특성을 일일이 명시하는 불편을 제거하고, 한 클래스의 객체 전체를 용이하게 관리할 수 있다.

상위 클래스는 여러 하위 클래스들의 공통적인 특성을 기술하는 클래스이다[7]. 상위 클래스를 정의하기 위해 하위 클래스들의 내포들이 통합되므로 내부적 통합 추상화가 이루어진다. 하위 클래스들을 통합하여 새로운 상위 클래스를 형성하는 것은 일반화(generalization)[19]로 알려져 있으며, 하위 클래스들의 내부적 통합 추상화는 소프트웨어 분석 단계 또는 실제 세계를 개념화시키는 모델링 단계에서 이루어지며, 프로그램 그 자체에서는 명시적으로 드러나지 않는다.

내외부적 통합 추상화는 외부적 통합 추상화와 내부적 통합 추상화가 동시에 이루어지는 추상화이다. 가장 간단한 형태의 내외부적 통합 추상화가 적용된 것이 매개변수가 있는 함수이다. 예를 들어, 특정 정수에 대해 팩토리알을 구하는 함수들의 표상인 인터페이스를 get_factorial_0(), get_factorial_1(), get_factorial_2() 등이라고 할 때, 이들 표상들은 하나의 표상 get_factorial(int n)으로 통합 추상화될 수 있다. 즉, 0, 1, 2 등 특정 정수값은 이를 대표하는 int n으로 통합된다. 또한 이들 함수들의 내포인 구현들도 하나의 구현으로 통합된다. 또한 임의 개수의 정수를 정렬하는 함수 sort(int data[], int n)은 sort_1(int data[]), sort_2(int data[]), sort_3(int data[]) 등 특정 개수의 정수들을 정렬하는 여러 함수들을 통합시킨 함수로 내외부적 통합 추상화가 적용된 예이다.

내외부적 통합 추상화의 또 다른 경우는 포괄 함수와 포괄 클래스에서 볼 수 있다. 포괄 함수에서의 추상화는 여러 함수의 인터페이스들을 타입변수로 두어 하나의 인터페이스로 통합하는 외부적 통합과정과, 여러 구현들을 하나의 구현으로 통합하는 내부적 통합과정이 포함된다. 예를 들어, sort_int(int data[], int n), sort_float(float data[], int n), sort_double(double data[], int n) 등 각 타입의 값들을 정렬하는 함수들은 내외부적 통합 추상화를 통하여 하나의 포괄 함수 sort(type t data[], int n)으로 통합될 수 있다. C++, Eiffel, Ada 등에서 지원되는 포괄 클래스도 내외부적 통합 추상화가 적용된 예이다.

4.3.2 동질적, 이질적 통합 추상화

동질적 통합 추상화는 동일한 타입이나 구조를 갖는 추상물들을 통합하는 것이다. 동일한 구조 및 특성을 갖는 값(객체)들의 공통된 구조와 특성을 추출하고 타입명(클래스 이름)을 부여한 타입(클래스)는 동질적 통합 추상화가 적용된 대표적인 경우이다. 또한 단순매개변수를 갖는 부프로그램도 동질적 통합 추상화가 적용되는 경우이다.

상이한 것들을 하나로 통합하려는 노력의 절정이 이질적 통합 추상화이다. 이질적 통합 추상화는 서로 다른 타입이나 구조의 여러 대상들을 통합하여 새로운 추상물을 형성하는 것으로, 이는 다형성 개념으로 발전되어 서로 상이한 것들을 하나의 표상으로 나타낼 수 있게 되었다. 이질적 통합 추상화가 적용된 경우로는 포괄 함수, 포괄 클래스, 상위 클래스 등이 있다. 포괄 함수와 포괄 클래스는 서로 다른 타입과 구조를 갖는 통합 대상들에 대해 내부적 통합 및 외부적 통합이 동시에 이루어지며, 특히 서로 다른 타입의 데이타를 통합하기 위해 타입 매개변수가 도입된다. 이에 반하여, 상위 클래스에서의 추상화는 통합 대상이 되는

---

5) 추상화란 정신적인 작업이므로 구체적인 표현양식은 추상화 과정에서는 중요하지 않다. 그러나 추상물이 프로그램에서 정의되고 참조될 때 표현양식은 매우 중요하다.

6) 이 인터페이스는 함수에서의 인터페이스와 다른, Java에서의 특별한 클래스인 인터페이스 클래스를 말한다.

7) 객체 지향 프로그램에서는 먼저 정의된 상위 클래스를 확장하여 하위 클래스를 정의한다. 그러나 프로그램 설계단계에서는 여러 하위 클래스들을 통합한 상위 클래스가 설계되기도 한다.

다른 구조의 클래스들에 대해 내부적으로 공통성을 추출하는 통합만 이루어지므로 별도의 타입 매개변수는 도입되지 않으며, 통합할 클래스들을 모두 수용할 수 있는 상위 클래스가 생성된다.

4.4 확장 추상화

통합 추상화의 역인 확장 추상화는 하나의 대상으로부터 하나 또는 그 이상의 보다 구체적인 새로운 추상물을 생성하는 것으로, 확장 대상이 되는 추상물 (O, A)로부터 <(O_1, A_1), (O_2, A_2), … (O_n, A_n)>을 생성한다. 여기에서 A_1, A_2, … A_n은 확장 추상물의 표상이며, O_1, O_2, … O_n은 확장 추상물의 내포이다. 확장 추상화를 통하여 생성된 추상물들의 내포 O_1, O_2, … O_n은 O의 내포를 그대로 가지면서 새로운 요소가 첨가되어 확장된 내포를 갖는다.

확장 추상화가 적용된 대표적인 경우가 Java, C++ 등 객체 지향 언어에서 상위 클래스를 확장하여 하위 클래스를 선언하는 것이다. 한 클래스 S가 클래스 C를 확장하면 클래스 S는 C의 하위 클래스로 확장되며, S는 C의 모든 특성을 자동적으로 가지게 되는데, 이를 상속이라 한다. 한편 S는 C에 포함되지 않은 S만의 고유한 특성들을 부가적으로 가질 수 있게 된다.

4.5 대체 추상화와 참조 추상화

대체 추상화와 참조 추상화는 추상화 표상이 언급될 경우 이에 대한 처리방법에 따른 분류이다. 대체 추상화는 추상화 표상이 추상화 내포로 대체되어지는 것으로, 어셈블리어에서의 변수이름과 연산자, C 언어에서의 마크로 등에 적용되는 추상화이다. 변수이름, 연산자, 마크로 이름 등 추상화 표상이 언급되면 이진수 주소, 이진수 코드, 마크로 정의 등 해당 추상화 내포로 대체된다.

참조 추상화는 추상화 표상이 언급되면 추상화 내포가 간접적으로 참조되는 것으로, 복합 데이터, 함수, 타입, 객체, 클래스 등에 적용되는 추상화이다. 예를 들어, 함수의 표상이 프로그램에서 나타나면(즉, 함수가 호출되면), 함수의 구현으로 대체되지 않고 함수의 구현이 저장된 곳으로 제어가 이동된 후 구현이 수행된다. 추상화 내포가 참조되기 위해서는 추상화 내포가 저장되는 주소가 내부적으로 관리되어야하며, 추상화 표상과 주소간의 변환도 이루어져야 한다. 대체 추상화와 참조 추상화는 [24]에 보다 자세히 설명되어 있다[8].

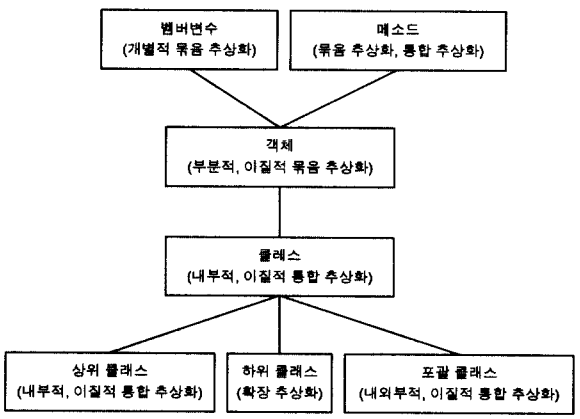
5. 추상화 분류의 적용

4장에서 제시한 추상화 전반에 대한 명확한 정의 및 분류는 데이터 추상화, 추상 데이터 타입, 캡슐화, 클래스 등 관점에 따라 다르게 설명되는 용어를 보다 엄밀하고 통일성 있게 설명할 수 있게 한다. 예를 들어, 데이터 추상화는 상태를 표현하기 위한 데이터와 처리를 위한 연산을 하나로 묶고 이를 추상화한 것으로, 분류된 여러 추상화의 관점에서 '데이터 추상화 = 연관된 데이터와 연산에 대한 묶음 추상화'로 정의될 수 있다. 데이터 추상화가 이루어진 추상물은 데이터 객체라 불리며, 데이터 객체는 데이터 표현의 방법과 데이터 처리를 위한 연산의 구현을 외부에 노출할 필요도 없으며 외부에 영향을 주지 않으면서 내부 표현이나 구현을 변경할 수 있게 된다.

한편, 정의되어진 엄격한 외부 인터페이스만을 통하여 추상화된 객체의 특정 요소들을 접근할 수 있게 하는 캡슐화는 묶음 추상화에서 내부적 요소들에 대한 부분적인 접근을 가능하게 하는 것으로, '캡슐화 = 연관된 데이터와 연산에 대한 부분적 묶음 추상화'라고 할 수 있다. 한편, 추상 데이터 타입은 동일한 구조적 특성을 갖는 데이터 객체에 대한 내부적 통합 추상화에 해당되므로 '추상 데이터 타입 = 캡슐화된 객체들에 대한 내부적 통합 추상화 및 동질적 통합 추상화'이라 할 수 있다.

상하위 클래스 또는 포괄 클래스의 추상화는 여러 단계의 추상화가 적용된 복합적 개념이다. 하나의 클래스가 정의되기 위해서는 기본적으로 복합 데이터로 구성되는 멤버변수에 대해 개별적 묶음 추상화가 적용되며, 메소드에 대해 묶음 추상화 및 통합 추상화가 적용된다. 멤버변수와 메소드로 구성되는 객체에 대해 부분적 묶음 추상화 및 이질적 묶음 추상화가 적용되고, 같은 특성의 객체들을 통합시킨 클래스에 대해 내부적 통합 추상화 및 동질적 통합 추상화가 적용된다. 그리고 상위 클래스를 형성하기 위해서는 내포가 다른 여러 클래스를 통합시키는 내부적 통합 추상화 및 이질적 통합 추상화가 적용되고, 하위 클래스를 형성하기 위해 확장 추상화가 적용되며, 포괄 클래스를 형성하기 위해서는 내외부적 통합 추상화 및 이질적 통합 추상화가 동시에 적용되어야 한다.

객체 지향 프로그램 언어에서의 멤버변수와 메소드에서 시작하여 상하위 클래스 및 포괄 클래스에 이러기까지의 단계별 추상화 적용과정은 (그림 2)에 표시되어 있다. 여러 추상화 과정을 통하여 생성되는 하나의 상하위 클래스 또는 포괄 클래스는 4단계 이상의 여러 추상화가 적용된 아주 고수준의 추상물임을 알 수 있다.



(그림 2) 추상화의 단계별 적용

8) [24]에서 부프로그램에 대한 참조 추상화를 호출 추상화라고 정의하였다.

6. 결 론

본 논문에서는 추상화 개념을 일관성 있고 통일성 있게 정의하기 위해 여러 프로그래밍 언어에 나타난 추상화 양상에 대하여 살펴보았고, 추상화의 분류체계를 새롭게 제안하였다. 추상화는 사상 추상화, 묶음 추상화, 통합 추상화, 확장 추상화로 분류되었으며, 함수, 복합 데이터, 추상 데이터 타입, 객체, 캡슐화, 클래스 등 추상화를 기반으로 하여 형성된 여러 개념들이 추상화의 관점에서 고찰되었다.

체계적인 추상화의 분석과 분류를 통하여 지금까지 절차적 추상화, 복합 데이터, 데이터 추상화, 다형성 등 개별적이며 서로 다른 형태의 추상화로 취급된 여러 추상화 양상이 하나의 통일된 개념으로 통합될 수 있으며 추상화의 의미, 필요성, 중요성에 대하여 보다 체계적인 설명도 가능하였다. 또한 상하위 클래스 또는 포괄 클래스에서는 서로 다른 수준의 여러 추상화가 반복적으로 적용됨을 확인할 수 있었다.

앞으로의 연구로, 프로그래밍 언어에서 가장 중요한 부분이며 통합 추상화의 결과로 나타나는 타입 개념과, 추상화의 또 다른 양상으로 중요시 되는 다형성 개념에 대해 추상화의 관점에서 심도 있는 고찰을 통하여 개념을 명확히 정립하고 이에 대한 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, Vol.9, No.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.

[24] 김성기, "절차적 추상화의 분류와 다형성", 한국정보처리학회논문지A, Vol.10A, No.1, Mar., 2003.

[25] 김성기, "프로그래밍 언어에서의 추상화", 한신대학교 정보과학연구소논문지, Vol.6, No.1, Dec., 2002.

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 교수

관심분야 : 데이타베이스 시스템, 객체 지향 시스템