2005 Proceedings of the Autumn Conference, Korea Statistical Society

R의 객체지향성에 대하여

이윤동[1)]

요 약

통계 소프트웨어 R은 여러 가지 특징을 가진 도구이다. S라는 전산언어를 기반으로 하고 이에 수학함수와 통계함수, 그리고 그래픽함수들이 결합되어 편리한 계산 작업 환경을 제공하고 있다. R이 기반으로 하고 있는 S언어에는 문법적, 의미론적 특징이 잘 어울려 있다. S언어의 주요 특징 중 하나는 객체지향성이다. 본 연구에서는 R의 특징인 객체지향성과 그 의미에 대하여 살펴보게 된다.

주요용어 : 객체지향, 클래스, 메쏘드, lexical scoping

1. 객체지향성의 이해

통계 소프트웨어 R은 여러 가지 특징을 가진 도구이다. S라는 전산언어를 기반으로 하고 이에 수학함수와 통계함수, 그리고 그래픽함수들이 결합되어, 편리한 계산 작업 환경을 제공하고 있다. R이 기반으로 하고 있는 S언어는 문법적(syntax) 면에서는 S-plus와 매우 유사한 모습을 보이면서도 그 의미론적(semantic) 면에서는 Algol60 계열의 언어인 Scheme이나 혹은 Pascal의 작용 방식과 동일한 (nested) lexical scoping 규칙을 채택하고 있다 (참조, Gentlman과 Ihaka, 2000; Ihaka와 Gentleman, 1996). 이와 함께 S언어의 중요한 특징 중 하나는 객체지향적 (object oriented) 성질을 가지고 있다는 점이다.

객체지향성은 1970년대 Xerox 연구소의 A. Kay에 의하여 만들어진 Smalltalk 이라는 언어에서부터 시작된 개념이다. 객체지향성은

"Everything is an object, and every object has a class"

라는 말로 대표된다. 언어의 모든 구성요소가 다 객체(object) 이고, 모든 객체는 클래스를 갖는다는 말이다. R이 기반을 둔 S언어에도 마찬가지 개념이 통용된다. 실제 이 한 마디에 객체지향성의 모든 개념이 들어 있다고 볼 수 있다. 그러나 간결한 한 마디 말에 들어 있는 모든 의미를 금방 파악하기는 쉽지 않다.

객체지향 패러다임(paradigm)의 이해를 위해서는 초기 포트란으로부터 시작하여 근래의 구조적 프로그래밍 패러다임에 이르기까지의 전산언어 발달 과정에 대한 이해와, 특정 언어로 작성된 프로그램 코드가 어떻게 컴파일 되고, 링크되어 수행되는지에 대한 이해가 필요하다. C언어로 대표되는 구조적 프로그래밍 패러다임은 작업 수행과정을 몇 개의 작은 부함수(subroutine) 혹은 함수로 구분하여 이들의 연개로서 작업을 수행하게 하고자 하는 것이다. 결국 구조적 프로그래밍 패러다임은 그 구성 부분인 함수 개념이 갖는 특성을 함께 갖는다. 함수는 입력과 출력에 의하여 규정된다. 어떤 입력을 받을 수 있는지 그리고 어떤 결과를 주게 되는지가, 수학적 혹은 전산학적 입장에서 함수의 특성을 규정하는 일차적인 방법이다. 이러한 함수적 방법은 크게 두 가지 단점을 갖는다. 하나는 함수의 정의를 위해서 입력과 출력에 대한

1) 조교수, 건국대학교, 응용통계학과, 143-701 서울시 광진구 화양동, poisson@dreamwiz.com

자료형(type) 혹은 자료 구조의 명시가 필요하다는 점이다. 다른 하나는 함수의 동적바인딩(dynamic binding)이 불가능하다는 점이다.

함수가 받아들이게 되고 만들어 낼 자료형의 명시는 프로그램 작성자에게는 매우 귀찮고 자주 에러를 만들어 내게 만드는 제약이다. K.E. Iverson 에 의하여 1960년대에 제안된 APL은 대표적인 무형(typeless) 언어로서 자료형의 구분에서 오는 불편함을 제거하고자 하는 시도를 하였다. 그러나 자료형의 개념을 사용하지 않는 무형언어들은 메모리의 효율적인 사용이 어렵고 계산 속도가 느린 이유로 현실적 대안으로 받아들여지기 어려웠다.

무형언어가 주는 편리함을 추구하면서도 무형언어의 비효율성을 극복하기 위한 대안으로 고려될 수 있는 방법이, 자료형의 구분을 없애는 것이 아니라, 자료형의 인식과 그 변화를 프로그래머가 관리하지 않고 프로그램이 컴파일 되고 수행되는 과정에서 컴퓨터가 담당하게 하는 방법이다. 그를 위해서는 함수의 전달인자가 될 수 있는 모든 것을 묶을 수 있는 하나의 틀을 만들고, 그 틀에 속하는 구성원들의 특성을 명시하여 그 특성에 따라 함수가 적절히 작용하도록 하게 해야 한다. 이 때 이렇게 정의되는 틀이 객체이다. 즉 함수의 전달 인자가 될 수 있는 대상인, 변수, 데이터, 심지어는 다른 함수까지, 언어 체계 내에서 다루게 되는 모든 구성요소를 객체라는 동일한 틀로 묶게 되는 것이다. 그 것을 나타내는 말이 "everything is an object"이다. 또 객체의 특성에 따라 함수가 다르게 작용하도록 하기 위해서는 개체의 특성을 설명하는 정보가 필요하다. 이를 위해 객체가 가지고 있는 특성을 설명하는 정보가 클래스이다. 즉 클래스는 주어진 객체가 변수라면 어떤 자료형을 갖는지, 데이터라면 어떤 자료구조를 갖는지, 혹은 그것이 다른 함수인지를 설명하는 정보가 들어있게 된다. 클래스의 구성 요소를 슬롯(slot) 이라 하고, 특정 클래스에 속하는 객체를 대상으로 하여 작성된 함수를 메쏘드(method) 라 한다.

데이터베이스 관점에서 객체를 테이블을 구성하는 개개의 레코드의 값들이라고 한다면 클래스는 그 테이블을 이루는 레코드의 구조라고 할 수 있다. 테이블을 구성하는 레코드 수를 미리 명시하여 테이블을 정의하는 것보다, 레코드 구조만을 선언 하는 방법으로 테이블을 정의하고 프로그램 실행 중에 상황에 맞게 테이블의 레코드 수를 조절하는 방법이 보다 효율적인 메모리 자원 관리가 가능하고 상황 변화에 유연한 프로그래밍이 가능하다.

또 하나, 통합적 틀로서의 객체와 그 특성 정보인 클래스 개념을 통하여 얻을 수 있는 장점은, 구조적 프로그래밍 패러다임의 주 구성 부분이던 함수를 클래스 정보를 입력으로 받아 수행되는 메쏘드로 개념으로 바꿈으로써, 함수에 대한 주소 할당 시점을 컴파일 시점이나 링크 시점이 아니라 해당 프로그램이 실행되는 시점으로 늦출 수 있는 동적바인딩을 적용할 수 있는 기반을 마련하게 되었다는 점이다. 객체의 특성, 즉 클래스 정보에 따라 함수의 작용 방식이 달라야 하고 객체의 설정은 프로그램 실행 중에 가능하기 때문에 당연히 해당 객체에 작용될 메쏘드의 선택도 프로그램 실행 시점으로 늦추는 것이 가능해야 한다. 이를 구현하기 위하여 실행될 함수에 메모리상 의 주소를 실행 중에 할당하는 방법이 동적바인딩이다.

대표적인 객체지향 언어인 C++로 작성된 Windows의 현실적 성공에도 불구하고, 객체지향성이 갖는 것으로 알려진 소프트웨어 공학적인 측면에서의 기타 다양한 장점들에 대하여는 그 실효성에 대하여 찬반양론이 있는 것이 사실이다. 뿐만 아니라 R은 자체가 하나의 응용프로그램이기도 하고, 다른 프로그램 개발이 가능한 전산언어이기도한 두 가지 입장을 가지고 있다. 이 때문에 R에서의 객체지향성이 갖는 의미와 효과는 다르게 평가될 수도 있다. 또한 실제로 R이나 Unix 같은 훌륭한 소프트웨어가 객체지향 언어가 아닌 C로 작성되었다는 점은 객체지향성이 갖는 실효성에 대하여 의문을 제기하기에 충분한 면이 있다. 그러나 기본적으로 어느 정도 규모 있는 소프트웨어 작성을 위해서라면 객체지향성에 대한 이해는 필수적이라고 보인다.

2. R에서 객체지향 방법의 변화

보통 S언어에는 네 가지 버전이 있다고 말한다. 이를 S언어 발달 과정에 발표되었던 네 권의 책으로 구분하기도 하는데, 책표지의 색깔에 따라 Brown, Blue , White, Green 으로 구별하기도 하고, S1, S2, S3, S4 와 같이 구별하기도 한다. 1996년 R이 처음 세상에 모습을 보이면서 채택한 것은 Chambers와 Hastie (1993)을 말하는 White book 버전인 S3 였다. 2003년 R 버전 1.7.0이 발표되면서, Chambers(1998)을 말하는 Green book, 즉 S4의 주요 특징이 R에 구현되었다. S3와 S4의 주요 차이는 객체지향성의 구현 방식에 있다. 이에 대하여는 Bates(2003)과 Lumley(2004)에 자세한 사항이 언급이 되어 있다.

S3에서의 객체지향성의 구현 방법은 특별함이 없이 특정 문자열의 일치에 의하여 판단하는 방법이었다. 예를 들어, 선형모형을 적합 하는 함수 lm(y~x) 의 결과물을 lmout 이라고 하자. lmout의 클래스는 "lm"으로 지정이 된다. 이때 print(lmout) 과 같이 print() 라는 일반 함수(generic function)을 사용하여 lmout을 출력시키고자 한다고 하자. print 함수는 UseMethod ("print") 에 따라, lmout이라는 객체의 클래스에 맞는 메쏘드 print.lm()을 찾아 사용하게 된다. 이 과정에서 R이 해당 메쏘드를 선택하는 방법은 함수 이름인 print와 클래스 이름인 lm 그리고 이를 구별하는 "." 문자열을 접합(paste) 하여 만든 print.lm 이라는 이름의 함수를 찾게 되는 것이다. 그러므로 사용자가 자신이 만든 특정 함수의 이름을 print.lm 이라고 해 놓았다면 print(lmout)은 lmout의 결과를 출력하는 대신 예상치 못한 결과를 주게 되는 문제점이 있다.

또한 S3에서 구현된 방법은 상속성 (inheritance)의 관점에서도 다음과 같은 문제점을 갖는다. 클래스들 사이의 상속의 개념은 "동물"이라는 클래스와 그 하위 클래스인 "조류"라는 클래스를 고려할 때 "동물" 클래스에 작용하던 연산이 자동적으로 "조류" 클래스에도 작용하도록 하고자 하는 것이다. 구체적인 예로 분산분석 함수 aov() 를 사용하여 얻어지는 결과물 aovout은 "aov" 클래스에 속하고, 통계학적 의미에서 분산분석은 당연히 선형모형으로 해석되므로 클래스 "lm"에도 속해야 한다. 이런 관계를 설정하기 위해서는 "lm" 이라는 클래스와 "aov" 클래스들 사이에 관계를 설정해야 한다. 그러나 S3 는 이를 aovout 의 클래스 정보 즉 class(aovout)에 "aov"와 "lm"을 c("aov","lm") 와 같은 형태로 동시에 명시해 놓는 방법을 사용한다. 즉 상속의 문제가 클래스들 사이에서 정의된 것이 아니라, 편법적으로 객체 특성으로 정의되어 있는 것이다. 이런 방식에서는 특별한 경우 상속성이 어떤 클래스에 속하는 전체 객체에 대하여 일관되게 작동되지 않을 수 있다는 문제점을 갖고 있다.

이러한 문제점의 개선을 위하여 S4에서는 클래스의 공식적 (formal) 정의와 활용 방식을 채택하고 있다. x와 y라는 슬롯을 가진 클래스 "track" 을 정의하고 클래스 "track" 에 작용하는 print 메쏘드를 정의하기 위하여

```
setClass("track", representation(x="vector",y="vector"))
setMethod("print","track", function(t,...) print(t$x) )
```

와 같은 방식을 이용한다. 또한 클래스들 사이의 상속성을 나타내기 위한 함수 setIs() 가 정의되어 있어 위의 예와 같이 "aov" 클래스가 "lm" 클래스의 특성을 상속하게 하고자 하는 경우는 setIs("aov","lm")와 같이 클래스들 사이의 관계를 직접 정의할 수 있게 하였다.

참고문헌

J. Chambers와 T. Hastie (1993), *Statistical models in S*, Chapman and Hall.

J. Chambers (1998), *Programming with data*, Springer.

D. Bates(2003), Converting packages to S4, *Rnews*, **3-1**, pp 6-8.

T. Lumley(2004) Programmers' niche, *Rnews*, **4-1**, pp33-36.

R. Gentlman과 R. Ihaka (2000), Lexical scope and statistical computing, JCGS, **9**, 491-508.

R. Ihaka와 R. Gentleman (1996), R: a language for data analysis and ...,, JCGS, **3**, 299-314.