Scope 기능을 갖는 객체 지향 모델에서 파라미터화된 모듈 구현 연구

권 기 항† · 신 현 삼††

요 약

기존의 객체 지향 모델들은 공유와 재사용 측면에서 효율적이지만, 아쉽게도 객체의 사용 범위를 정의할 수 있는 방법이 결여되어 있다. 본 논문에서는 객체의 사용 범위를 정의할 수 있도록 하는 객체 지향 모델을 살펴본다. 우리는 이 모델이 프로그램의 관리를 최적화 할뿐만 아니라, 파라미터화된 모듈개념을 쉽게 지원할 수 있음을 예제를 통해 보인다.

Implementing Parameterized Modules in an Object-oriented Model with an Notion of Scope

Kee-Hang Kwon† · Hyun-Sam Shin††

ABSTRACT

While object-oriented models are effective in achieving sharing and code reusability, they unfortunately lack a mechanism for giving scope to objects. We revisit an object-oriented model in which each object can be given a scope. We illustrate the usefulness of this model by showing that it supports the notion of parameterized modules without difficulty.

1. 서 론

현재 다양한 객체 지향 모델들이 제시되어 있다[1-5]. 하지만 이와 같은 객체 지향 모델들은 뛰어난 재 사용성을 가지고 있음에도 불구하고 객체가 사용되는 범위를 지정할 수 있는 구문이 결여되어 있다는 점에서 비효율적이다. 최근에 제시된 모델[6]은 사용자가 각 객체의 사용범위를 지정할 수 있게 한다. 따라서 프로그램은 항상 필요한 객체들에 제한하여 관리하게 되므로 프로그램의 크기가 줄어든다.

기존의 객체 지향 모델들과 [6]에서 제시하는 객체 지향 모델을 비교하기 위하여 다음과 같은 구문을 살펴보자 :

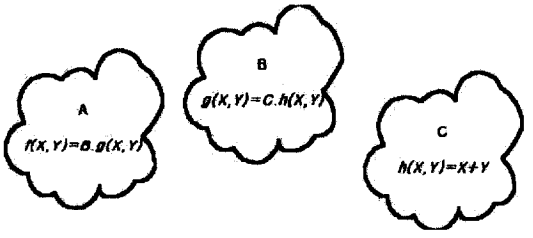
obj.f

기존의 객체 지향 모델에서 이 구문은 "*obj* 라는 객체에 메시지 *f* 를 전송한다"는 의미로 해석된다. 이 해석은 메시지 *f* 를 수행하기 전에 *obj* 가 프로그램 내에 이미 존재한다는 것을 전제로 한다. [6]에서 제시된 객체 지향 모델에서 위 구문의 해석은 다음과 같이 바뀐다 : "***obj*** 를 현재의 프로그램(즉, 객체의 리스트)에 동적으로 추가하고 메시지 ***f*** 를 전송한다."

새로운 해석을 이해하기 위하여, (그림 1)과 같은 객체들의 집합이 있다고 가정하자. 이 경우 A.f(a, b)의 값을 구하기 위해서는 기존의 객체지향 모델에서는 프로그램 내에 객체 A, B, C가 필요하고 다음과 같이 (1)~(4)의 순서를 거쳐 값을 구할 수 있다.(여기서 P|-E는 프로그램 P를 사용하여 expression E의 값을 구한다는

※ 이 논문은 1997년도 동아대학교 학술연구조성비(일반과제)에 의하여 연구되었음.
† 정 회 원 : 동아대학교 컴퓨터공학과 교수
†† 준 회 원 : 동아대학교 대학원 컴퓨터공학과
논문접수 : 2000년 3월 29일, 심사완료 : 2000년 6월 29일

의미이다.)



(그림 1) 객체들의 집합

(1) A, B, C ⊢ A.f(a, b)
(2) A, B, C ⊢ B.g(a, b)
(3) A, B, C ⊢ C.h(a, b)
(4) A, B, C ⊢ a+b

여기서 주목할 점은 (1)~(4)에서 프로그램 내에 객체 A, B, C가 계속 존재한다는 것이다. 하지만 객체 A, B, C가 계속 존재할 필요는 없다. (1), (2), (3) 각각의 경우에 대해서 프로그램에 B와 C, A와 C, A와 B가 불필요하게 포함되어 있으며, (4)의 경우 a+b 값을 구한 후, 프로그램에 A, B, C가 존재할 필요가 없다.

새로운 해석에서는 다음과 같이 프로그램 nil로부터 A.f(a, b)의 값을 구할 수 있으며, 필요한 객체는 계산 과정에서 프로그램에 점진적으로 도입된다.

(1) nil ⊢ A.f(a, b)
(2) A ⊢ f(a, b)
(3) B, A ⊢ g(a, b)
(4) C, B, A ⊢ h(a, b)
(5) C, B, A ⊢ a+b

이와 같이 scope 기능을 갖는 객체 지향 모델은 항상 필요한 객체만을 프로그램에 동적으로 추가한다. 이 방법의 장점은 프로그램이 실행 중 필요한 객체만을 관리하면 되므로 프로그램의 크기를 줄일 수 있다. 본 논문에서는 이와 같은 객체 지향 모델을 정의하고 이 모델 내에서 parameterized module이 어떻게 구현되고 있는지 설명한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 scope 기능을 갖는 객체 지향 모델에 대해서 정의한다. 3장에서는 2장에서 설명한 모델을 사용하여 실제 parameterized module 예를 보인다. 마지막으로 4장에서는 결론과 앞으로 연구해야 할 과제를 제시한다.

2. 객체 지향 모델 정의

이 장에서는 scope 기능을 갖는 객체 지향 모델을 정의한다. 이 정의는 [6]의 정의를 근거로 하고 있다.

다음은 본 논문에서 제시하는 객체 지향 모델을 정의하기 위한 몇 가지 기본적인 용어이다.

정의 2.1 ***nil*** 은 널(null) 리스트를 의미하고, :: : 리스트 생성자를 나타낸다. 또 $a_1 :: a_2 : :... :: a_n :: nil$ 은 a_1 부터 a_n 까지 n개의 원소를 갖는 리스트를 나타낸다.

정의 2.2 본 논문에서 사용할 구문은 변수 ***Obj***와 ***Expression***을 사용해서 다음과 같이 정의된다.

$$Obj = (f(X_1, X_2, \ldots, X_n) = Expression) :: nil$$
$$\mid (f(X_1, X_2, \ldots, X_n) = Expression) :: Obj$$

$$Expression := Obj.Expression$$
$$\mid f(Expression, \ldots, Expression)$$
$$\mid Constant$$
$$\mid Variable$$
$$\mid if\ Expression\ then\ Expression\ else\ Expression$$

정의 2.2에서 볼 수 있는 것처럼 본 논문에서 제시한 객체 지향 모델에서 하나의 객체는 메소드들의 리스트로 구성되어 있으며, 객체에 전송하는 메시지인 *Expression*은 메소드, 상수, 제어문 등의 여러 구문을 사용할 수 있다. 주목할 점은 *Expression*에서 객체가 반복하여 나올 수 있다는 점이다. 즉 *a, b, c*가 객체일 때 *a.b.c.f(d)*와 같은 표현이 허용된다.

정의 2.3 프로그램은 ***obj***의 리스트로서 구성되어 있다.

$$Program := nil$$
$$\mid obj :: Program$$

지금까지 scope 기능을 갖는 객체 지향 모델의 문법을 정의하였다. 이번에는 이 객체 지향 모델에 사용되는 언어의 의미에 대해 정의한다. 이를 위해서 프로그램 P와 *Expression* E가 주어졌을 경우, 값을 구하는 함수 $P \mid\!- E$를 다음과 같이 정의한다.

정의 2.4 프로그램 ***P***와 ***Expression E***가 주어질 때 ***P***로부터 ***E***의 값을 구하는 함수 $P \vdash E$는 다음과

같이 정의된다.

가) 만일 E가 $obj.E_1$이면

$$P \vdash E = obj :: P \vdash E_1$$

나) 만일 E가 $f(E_1, E_2, , E_n)$이면

여기서 P는 $obj_1 :: obj_2 :: :: obj_n :: nil$이라 하고 obj_1부터 obj_{k-1}에 f가 정의되어 있지 않고 obj_k에 f가 처음으로 다음과 같이 정의되어 있다고 가정한다.

$$f(X_1, X_2, ..., X_n) = E_k$$

그러면

$$P \vdash E = P \vdash [R_1/X_1, R_2/X_2, ,R_n/X_n]E_k$$

여기서 $R_m = P \vdash E_m$, $1 <= m <= n$, $[R_1/X_1, R_2/X_2, ,R_n/X_n]E_k$는 E_k에서 X_1 대신 R_1을 대입, X_2 대신 R_2을 대입, ..., X_n 대신 R_n을 대입한 결과를 의미한다.

다) 만일 E가 $Constant\ C$이면

$$P \vdash E = C$$

라) 만일 E가 $if\ E_1\ then\ E_2\ else\ E_3$이면

$$P \vdash E = P \vdash E_2 \ if\ P \vdash E_1 > 0$$
$$P \vdash E = P \vdash E_3 \ if\ P \vdash E_1 = 0$$

정의 2.4의 가)는 P로부터 $obj.E_1$의 값을 구하기 위해서는, P의 맨앞에 객체 obj를 추가하여 생성된 객체의 리스트 $obj::P$에서 E_1의 값을 구한다는 것을 의미한다.

정의 2.4의 나)는 E가 임의의 메소드 f일 경우 객체의 리스트 P의 가장 왼쪽 객체(가장 마지막으로 객체의 리스트에 추가된 객체)부터 오른쪽 객체의 순으로 검색하여 f가 처음으로 정의되어 있는 obj_k를 찾고, obj_k의 메소드 f를 사용하여 $Expression$의 값을 구한다. 여기서 메소드 f의 값을 구하기 위해 필요한 인자 $E_1, E_2, , E_n$는 $R_k = eval(P, E_k)$, $1 <= k <= n$와 같은 계산 방법을 사용하여 값을 구하고, 이 값을 메소드 f에 $R_1/X_1, R_2/X_2, ,R_n/X_n$와 같이 인자로 대입하여 $f(E_1, E_2, , E_n)$의 값을 구할 수 있다.

정의 2.4의 다)는 E가 상수일 경우 계산 과정이 필요하지 않으므로 상수 자체가 계산 결과임을 보여 주고 있다.

정의 2.4의 라)는 조건문의 경우 E_1의 값을 구하여 0보다 큰 경우 E_2의 값을 구하고, 0일 경우는 E_3의 구문을 계산한다는 것을 보여준다.

이와 같이 [6]에서 제시한 객체 지향 모델은 필요한 객체를 동적으로 추가하여 사용할 수 있다. 이런 객체의 동적 추가 기능은 객체 지향 모델의 여러 유용한 기능들 — 동적상속, 파라미터화된 모듈등 — 을 아주 쉽게 구현하게 할 수 있게 해 준다.

지금까지 본 논문에서 제안한 객체 지향 모델을 정의하였다. 다음 장에서는 본 논문에서 제시한 객체 지향 모델의 사용 예를 보인다.

3. 객체 지향 모델 사용의 예

이 장에서는 2장에서 정의한 객체 지향 모델을 사용한 실제 예를 보인다.

표준 ML 모듈 시스템[7]은 파라미터화된 모듈 동작을 처리하는 functor들의 개념을 제공한다. 한편 C++는 template의 개념을 사용하여 파라미터화된 모듈을 지원한다. 하지만 scope기능을 갖는 객체 지향 모델에서는 이런 새로운 개념의 도움 없이 파라미터화된 모듈을 쉽게 지원할 수 있다. 다음은, 임의의 데이터 타입의 원소들을 정열하는 파라미터화된 quicksort 를 작성한 예이다.

```
class qsort.
qsort(nil) = nil.
qsort(X::L) = list ⇒
   append(qsort(lessthan(X, L)), append(X::nil, qsort(nonlessthan(X, L)))).
class integer.
lessthan(X, nil) = nil.
lessthan(X, Y::L) = Y::lessthan(X, L) if X > Y.
lessthan(X, Y::L) = lessthan(X, L)    if X ≤ Y.
nonlessthan(X, nil) = nil
nonlessthan(X, Y::L) = Y::nonlessthan(X, L) if X ≤ Y.
nonlessthan(X, Y::L) = nonlessthan(X, L)    if X > Y.
```

예제 3.1 파라미터화 된 *quicksort*

클래스 *qsort*는 파라미터화된 *quicksort*를 구현하고, 클래스 *integer*은 정수들의 순서를 나타내는 *lessthan*과 *nonlessthan* 함수들을 정의한다. 위의 예에서, 일반적 quicksort의 개념이 functor들과 같은 개념의 부가적인 construct들을 애써 만들지 않고도 쉽게 구현할 수 있다. 예를 들어, *qsort.integer.qsort(1::3::2::4::nil))*

의 표현은 정수 리스트를 정렬하는데 사용될 수 있다. 이와 비슷한 방식으로, *qsort.character.qsort(a::c::b::r::nil)*의 표현은 문자 리스트를 정렬하는데 사용된다. 여기서 클래스 character는 character 데이터형에 대한 *lessthan, nonlessthan*의 정의를 갖고 있어야 한다. 즉 클래스 *qsort*내의 *lessthan*과 *nonlessthan*은 정의되지 않고 open된 상태로 존재하다가 다른 모듈과 동적으로 결합시 bind된다는 것을 주목 할 필요가 있다. 파라미터화된 stack 데이터형 역시 비슷한 방법으로 구현될 수 있다.

이와 같은 본 논문에서 제시한 객체지향 모델은 객체를 동적으로 연결하여 사용범위를 지정하고 또 파라미터화된 모듈개념을 쉽게 지원할 수 있다.

4. 결 론

본 논문에서 제시한 객체 지향 모델은 프로그램에 사용되는 객체의 집합을 동적으로 변경할 수 있도록 하였다. 그러므로 프로그램에 항상 필요한 객체의 집합만이 유지할 수 있으므로 프로그램의 크기를 줄일 수 있다는 장점이 있다. 이 장점 외에도 파라미터화된 모듈을 별도의 확장 없이 지원해 줄 수 있다는 점이 본 논문의 핵심내용이다. 우리는 이를 파라미터화된 quicksort의 예를 통해 보였다. 제시된 모델은 동적 상속 역시 별도의 확장 없이 지원해 줄 수 있는데, 이는 [6]에 설명되어 있다.

결론적으로, 현재 대부분의 객체지향모델에 scope기능이 지원이 안되고 있는데, 언급된 여러 장점들을 고려 할 때 scope기능은 객체 지향 모델의 필수요소라고 판단된다.

Actor 모델[3]에도 객체의 scope기능이 어느 정도 부여되어 있으나 — 즉 actor가 dormant 될 수 있음 — 개념이 다소 복잡하고 제시된 모델의 scope 기능만큼 자연스럽지 못한 것으로 판단된다.

참 고 문 헌

[1] Lynn Andrea Stein, Henry Liberman, David Ungar, *"A Shared View of Sharing : The Treaty of Orlando"* In Kim W., and Lochovosky, F., editors, Object-Oriented Cocepts, Applications, and Databases, Addison-Wesley, 1988.

[2] D. Ungar and R. B. Smith "Self : The Power of Simplicity," *Proceddings of the Second ACM Conference on Object-Oriented Programming Systems, Languages and Applications, SIGPLAN Notices*, Vol.22, No.10, pp.227-242, 1987.

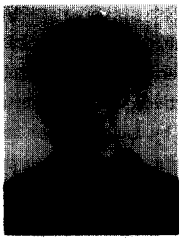
[3] G. A. Agha, "ACTORS : *A Model of Concurrent Computation in Distributed Systems,*" The MIT Press, Cambridge, MA, 1987.

[4] A. Mercaado Jr., "Hybrid : Implementing Classes with Protypes, Masters's Thesis," *Technical Report CS-88-12, Brown University Department of Computer Science,* Province, RI, July, 1988.

[5] David Ungar, Craig Chambers, Bay-WeiChang, Urs-Holzle, "Organizing Program without Classes," *LISP AND SYMBOLIC COMPUTATION* , 1991.

[6] 권기항, 김지승, "객체지향 모델에서 사용범위 기능 도입에 관한 연구", 한국멀티미디어학회 논문지, 제2권 제2호, 1999.6.

[7] R. Harper, "Introduction to Standard ML," *School of Computer Science Canegie Mellon University*, 1986.



권 기 항

e-mail : kwon@se.donga.ac.kr

1983년 서울대학교 컴퓨터공학과 (학사)

1985년 미국 Georgia Tech Computer Science(석사)

1995년 미국 Duke University Computer Science(박사)

현재 동아대학교 컴퓨터공학과 조교수

관심분야 : 프로그래밍 언어, 소프트웨어 공학

신 현 삼

e-mail : sami32@ce.donga.ac.kr

1998년 경성대학교 정치외교학과 (학사)

현재 동아대학교 컴퓨터공학과 석사과정

관심분야 : 프로그래밍 언어, 소프트웨어공학