객체 모델의 프로토타이핑을 위한 클래스 스키마

○
강미영 조영석
동국대학교 전자계산학과

Class Schema for Prototyping of Object Models

○
Young Mi Kang Young Suck Cho
Dept. of Computer Science, Dongguk University
dasuni@wonhyo.dongguk.ac.kr youngs@wonhyo.dongguk.ac.kr

요 약

정형방법들은 명세와 설계 문제들에서 모호성, 불완전성, 그리고 불일치성을 보다 쉽게 발견하고 수정하기 위해 사용된다. Z 정형 명세 언어를 객체 지향 패러다임으로 기술하기 위하여, 클래스 기술에 대한 명확한 정의가 필요하다. 그리고 이를 증명할 수 있는 프로토타입이 필수적이다. Z 스키마의 숨김 연산자와 그의 확장인 논리 한정 연산자와 투사 연산자를 이용하여 클래스의 원리를 가지는 클래스 스키마를 유도하여 정의한다. 유도된 클래스 스키마가 상속성, 다형성, 캡슐화 등의 객체 지향 원리를 가지는가에 대한 증명하기 위한 프로토타입을 설계한다.

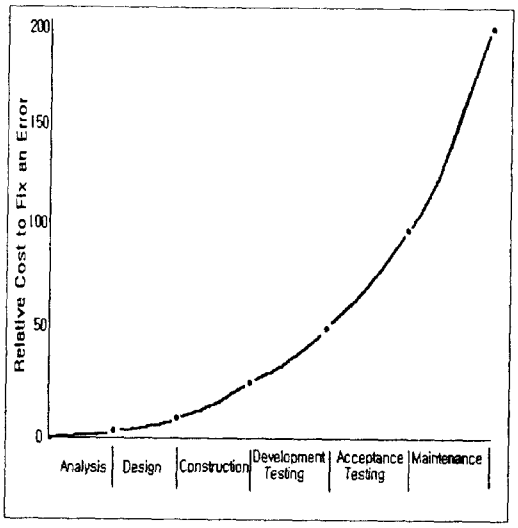
1. 서 론

소프트웨어 개발 초기 단계가 중요하게 인식된다. 분석과 설계 단계에서 발생하는 오류를 수정하는 것이 구현 및 유지 보수 단계에서의 수정보다 적은 노력과 비용으로 가능하기 때문이다[그림1]. 요구를 분석하는 정형적 명세 기법은 수학적 표기법을 적용하여 시스템을 기술하고, 개발하고, 그리고 검증을 가능하게 해 준다. 정형방법은 정형 명세 언어(formal specification language)를 사용해서 일관성, 완전성, 정확성이 체계적인 형식으로 평가할 수 있도록 시스템을 기술하는 수단을 제공해 준다[PRES 1995]. 정형 명세의 필요성은 아래와 같다[GRAV 1990].

- 정형 명세는 문제를 명확하게 이해하는 것을 돕는다.
- 고객(또는 다른 개발자)에게 개발 의도를 서로 대화한다.
- 아이디어를 제시(demonstrate)하기 위해 프로토타입을 제공한다.
- 디자인을 위한 기초로 사용한다.
- 소프트웨어의 정확성(correct)을 제공한다.
- 명세화을 수학적으로 확인한다.

명세화는 "시스템이 어떻게 그것을 하느냐?"에 대한 설명이 아니라 시스템이 무엇을 하는가?"에 대한 기술이다.

Z와 같은 정형적 명세언어는 표현력에 제한이 없으며, 범용적이고, 명세로부터 설계 사항들을 분리시킬 수 있는 능력을 갖고 있다. 이러한 특징들 때문에 Z가 비실행적인(non-executable) 명세 언어이지만 시스템을 명세하는데 많이 활용되고 있다[STEC 1987]. Z는 객체 지향적인 명세 언어가 아니다. 객체 지향 개념을 명세화에 도입함으로써 얻는



[그림 1] Error Correction Costs[KEND 1996]

장점은 객체 중심의 기술로 이해하기가 쉽고, 만들어진 객체들을 재사용하기가 용이하다.

객체 지향적 분석 방법은 시스템 행동과 구조의 추상적이고 이해 가능한 모델을 생성한다. 객체 지향적 패러다임의 장점을 이용하기 위해 Z의 객체 지향적 확장 노력을 하고 있다. 그러나 명확한 클래스의 정의가 없다[강 1998]. Venus[JIA 1998]는 UML의 모델링으로 클래스의 관계성을 설명하고 자세한 부분의 동작은 Z 개념으로 설명하여 코드의 자동화를 할 수 있는 툴이다.

이 논문에서는 Z 정형 명세 언어로 객체 지향 패러다임을 기술하기 위하여, Z 스키마의 숨김 연산자와 그의 확장인 논리 한정 연산자와 투사 연산자를 이용하여 클래스 스키마를 유도한다. 클래스 스키마가 상속성, 다형성, 캡슐화 등의 객체지향 원리를 가지는가에 대한 프로토타이핑을 위해 프로토타입을 설계한다.

2. Z 스키마의 구성

Z는 formal specification notation이며, 수학적 기반 위에 술어 논리(predicate logic)와 집합 이론(set theory)을 가지고 있다. 어플리케이션(applications)의 상태(state)인 면은 집합에 의해 표현되고, 그의 동작(operations)은 동적면을 나타낸다[FRAN 1997]. 이 장에서는 단지 Z notation 중에서 본 논문에서 사용되는 스키마 기본 구성을 소개한다.

Z로 시스템의 구조(construct)를 구성하는 기본이 스키마이다. 스키마는 선언부(declaration)와 기술부(predicate) 두 부분으로 나누어진다. 선언 부분은 w:Type 형태의 변수 선언을 구성한다. 변수 이름은 w, Type은 타입(type) 이름이다. 기술 부분은 선언된 변수사이에 관계성을 표현한다.

```
┌--- Schema ------------------
| Declaration
|-----------
| Predicate
└-------------------------------
```

또는

Schema ≙ [Declaration | Predicate]

Z 타입은 기본(basic)과 합성(composite) 타입이 있다. 기본 타입의 요소는 좀더 복잡한 요소를 위한 기본 빌딩 블럭(building blocks)으로 사용된다. 기본 type은

[BASIC_TYPE_NAME]

으로 기술한다. 기본 타입의 요소의 내부적 구조는 추상적이다. 집합 타입, Cartesian product 타입, 스키마 타입 등 3 종류 합성 타입이 있다. 합성 타입의 선언 예는

```
┌-------------------------------
| s : P S
| t : A × B
| u : Schema
└-------------------------------
```

s는 S의 집합 요소이다.
t는 첫 번째 요소가 A의 요소이고, 두 번째는 B의 요소로 구성된 요소들의 쌍이다.
u는 Schema인 스키마에서 선언된 변수들의 value들을 바인딩한다.

스키마타는 한 시스템의 정적인 면과 동적인 면을 모델링하기 위해 사용된다. 시스템의 정적인 면을 나타내는 스키마는 상태 스키마(state schema)이고 동적인 면을 나타내는 스키마는 동작 스키마(operation schema)로 적용된다.

2.1 상태 스키마

상태 스키마에서, 시스템의 상태의 요소들은 선언부에서 선언된다. 그리고 기술부에 상태의 제한을 나타낸다. 상태 스키마의 예로, 기본 타입을 가지는 Library 스키마를 Z명세 언어로 기술한다.

<상태 스키마의 예>
[Copy, Book, Reader] //기본 타입

```
Library
stock: Copy ⇸ Book
issued: Copy ⇸ Reader
shelved: F Copy
readers: F Reader
  // ⇸는 partial function arrow
  // F는 finite
---
  // 선언부와 기술부의 분리 표시
shelved ∪ dom issued = dom stock
  // dom은 domain
shelved ∩ dom issued = ∅
ran issued ⊆ readers
  // ran은 range
∀ r: readers •
    #(issued ▷ {r}) ≤ maxloands
  // ▷는 domain restriction operater
```

선언부는 다음과 같이 표시된다.

- stock는 현재 사용 중인 현재 사용중인 각각 copy로 구성된 책의 기록이다
- issued는 누구에게 대출해준 copy들의 기록이다.
- shelved는 빌려 갈 수 있는 선반 위의 copy들의 부분집합이다.
- readers는 가입된 readers의 집합이다.

스키마 Library의 기술부는 아래와 같이 설명된다.

- 서고의 모든 copy들은 대출중이거나 선반 위에 있다. 그러나 둘 다는 아니다.
- 단지 가입된 reader만 대출 할 수 있다.
- 마지막으로, 어떤 가입된 reader가 대출 할 수 있는 수는 허락된 최대수 보다 크지 않다.

2.2 동작 스키마

동작 스키마는 동작(operation) 실행의 시작과 끝에서 상태의 사이의 관계를 정의한다. 동작 스키마의 선언 부분은 상태의 전과 후, 입력과 출력, 그리고 다른 변수들이 관계성을 정의하기 위해 필요한 어떤 다른 변수를 표현하는 변수들을 선언한다. 동작 스키마의 기술부는 상태의 전과 후 상태들 사이에 관계성을 정의한다. 즉, △Library는 Library의 시작전의 상태와 Library'의 operation 실행 후의 상태를 표현한다. Ξnotation은 △의 확장으로 상태 변화가 없음을 나타낸다.

<동작 스키마의 정의>

```
△S
S
S'
```

```
ΞS
△S
---
NoChange
```

<동작 스키마의 예>

```
StockTransaction
△Library
---
readers' = readers; issued' = issued
```

readers의 변환전의 상태가 변환 후 상태인 readers'로 설정된다.

```
WhoHasCopy
ΞLibrary
c? : Copy; r! : Reader
---
c? ∈ dom issued; r! = issued c?
```

스키마 WhoHasCopy의 역할은 다음과 같다.

- ΞLibrary는 Library의 상태가 변함이 없음을 나타낸다.
- 변수 뒤에 ?의 의미는 입력변수를 나타낸다.
- 변수 뒤에 !의 의미는 출력변수를 나타낸다.
- 기술부에서 c?가 issued의 영역의 집합이고, 입력된 c?를 대출한 reader를 출력한다.

3. 클래스 스키마의 유도

Z 명세 언어는 객체 지향 언어가 아니므로, 객체 지향 패러다임의 장점을 이용하기 위해서, 먼저 클래스의 정의를 해야 한다. Z 스키마에서 정의된 숨김 연산자(hiding operator)와 여기서 좀더 발전된 논리 한정 연산자(quantification operator)와 투사 연산자(projection operator)를 이용하여 클래스 역할을 할 수 있는 스키마를 유도한다.

3.1 숨김 연산자 '\'의 이용

연산자 '\'의 오른쪽에 변수들을 연산자의 왼쪽에 제공되는 스키마에 제공된다.

[Decs|Pred] \ HiddenVars

[ReducedDecs| ∃ HiddenDecs • Pred]

예로써, 아래의 AssignNewCopy에서 선언부의 c?:Copy변수를 '\'연산자 오른쪽에 기술함으로써 EnterNewCopy에서 숨길 수 있다.

EnterNewCopy ≙ AssignNewCopy \ (c?)

EnterNewCopy와 EnterNewCopy의 스키마 기술하면 다음과 같다. 변수의 표현은 바뀌지만 의미는 변하지 않는다.

< 숨김연산자의 사용 예>

```
----- AssignNewCopy----------
StockTransaction
b?:Book; c?:Copy
----------
c? ∉ dom stock
stock' = stock ⊕ {c?↦b?}
shelved' = shelved ∪{c?}
-------------------------------
--- EnterNewCopy-----------------
StockTransaction
b?:Book
----------
∃c?: Copy • c?∉ dom stock
        ∧stock' = stock ⊕ {c?↦b?}
        ∧shelved' = shelved ∪ {c?}
-------------------------------
```

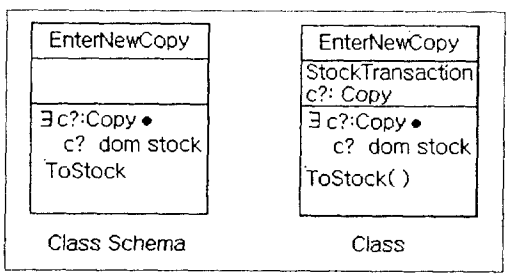
3.2 기술부에 스키마 포함

논리 한정 연산자(quantification operator)를 사용하여 스키마의 기술부에 논리 한정을 제공한다. 논리 한정된 변수는 스키마 선언부에서 제거된다. 다시 말하면, 한 스키마의 변수들이 서술부의 스키마들에 의해 기술되어 있다.

EnterNewCopy
≙∃c?:Copy | c?∉dom stock • ToStock

```
--- ToStock ------------------
StockTransaction
c?: Copy; b?:Book
-------------
stock' = stock ⊕ {c? ↦ b?}
shelved' = shelved ∪ {c?}
-------------------------------
```

서술부에 스키마를 포함 할 수 있으므로 클래스



[그림2] 클래스와 클래스 스키마

로 나타내면 [그림2] 과 같다. 여기서 스키마와 논리한정 안에 선언된 변수들 사이에 타입 파괴는 아니다. 이 스키마 논리한정 연산은 논리 한정된 변수를 숨기는 효과가 있다. 그리고 이 변수의 값에 제한을 가할 수 있다. 스키마에서 클래스의 개념을 Z로 표현을 하기 위하여, 클래스 스키마를 제한한다.

3.3 숨김 연산자의 확장

제안된 클래스 스키마는 논리 한정 연산자(quantification operator)와 투사 연산자(projection operator)를 사용한다.

◈전체적 논리 한정 연산자(universal quantification operator)로 아래와 같이 기술된다[POTT 1996].

∀Decs | Pred • Schema

클래스 스키마에서 기술부에 스키마를 클래스의 멤버함수처럼 사용하기 위해 투사 연산자를 사용한다.

◈투사 연산자(projection operator), '↾', 는 아래와 같이 기술된다[POTT 1996].

$Schema_1$ ↾ $Schema_2$

이 두 스키마는 $Schema_2$에서 선언된 것을 제외한 $Schema_1$의 모든 변수들을 숨길 수 있는 type-compatible이다.

위의 두 연산자를 사용하여 클래스 스키마를 나타내면 아래와 같다.

∀Decs | Pred • $Schema_1$
| $Schema_2$
| $Schema_3$

위의 스키마를 클래스로 나타내면 선언부와 서술부로 나누고 서술부에 동일한 선언부를 가지는 동적 스키마를 기술할 수 있다.

< CenterPoint 클래스 스키마>

[X, Y] //basic type : X, Y의 범위는 스크린의 영역이다.

```
---- CenterPoint ----------------------
// x: S X        // 정의된 S X, Y
// y: S Y
-------------
∀x:S X; ∀y:S Y • Draw
                ↾Erase
                ↾Move
-------------------------------
```

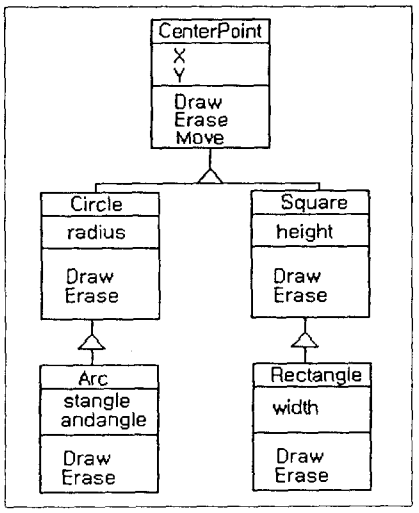
CenterPoint
≙ ∀x:N; ∀y:N • Draw ⨾ Erase ⨾ Move

4. 클래스 스키마와 객체 지향 원리

유도된 클래스 스키마는 대표적인 객체 지향 원리를 나타낸다. 객체 모델링한 [그림3]을 이용하여 상속성, 다형성, 캡슐화를 클래스 스키마로 기술한다.

4.1 상속(inheritance)

객체 지향 패러다임에서는 상위 클래스의 모든 속성을 하위 클래스에서 물려받는 것으로, 하위 클래스는 상위 클래스의 속성과 연산을 다시 정의하지



[그림 3] 객체 모델링

않고서도 자신의 속성으로 가질 수 있다. 스키마로 상속성을 표현하면 더욱 간결하고 명확하다.

<상속의 예>
[Radius] // basic type: Radius는 반지름으로 0 보다 큰 자연수를 나타낸다.

```
Circle ----------
CenterPoint
      //CenterPoint 스키마를 그대로
        상속받는다.
----------
∀radius :Radius • Draw
                ⨾ Erase
----------
```

상속의 표현은 선언부에 상속받는 클래스 스키마 이름을 기술한다. Circle는 CenterPoint에서 정의된 한점을 그대로 상속받는다. CenterPoint의 기술부에 기술된 동적 스키마 Draw와 Erase는 radius와 상속받은 x, y로 동작한다.

4.2 다형성(Polymorphism)

다형성은 한 함수의 이름이나 연산자가 여러 목적으로 사용이 가능한 것이다. 클래스가 하나의 메시지에 대해 각 클래스가 가지고 있는 방법으로 응답한다. 클래스 스키마의 기술부에 동작 스키마의 기술이 가능하므로, 다른 스키마에서 호출하는 하나의 메시지에 대해 각 클래스 스키마가 가지고 있는 고유한 방법으로 응답 할 수 있다.

Circle의 Move는 CenterPoint의 Move를 상속 받아서 Circle::Move를 호출하여 사용한다. Circle의 같은 이름의 동적 스키마를 기술했을 때, Draw를 호출할 때, CenterPoint::Draw로 점을 나타내는 것이 아니라 Circle::Draw를 수행하여 원을 그린다.

<Arc의 예>
[Angle] //basic type : 0 ~ 360의 각

```
Arc ----------
Circle    //Circle의 선언부와 기술부를 포함하는
            스키마를 상속받는다.
//stangle : S Angle
//andangle : S Angle
----------
∀stangle : S Angle;
∀andangle: S Angle • Draw
                    ⨾ Erase
----------
```

4.3 캡슐화(Encapsulation)

캡슐화(encapsulation)는 데이터와 데이터를 조작하는 연산을 하나로 묶는 것을 말한다. 일상 생활에서도 정보와 이 정보를 다루는 작업이 개념적으로 서로 연관되어 있음을 많이 본다. 이러한 연관된 정보와 작업, 또는 기능들을 하나로 묶는 것은 객체 지향 패러다임의 특징을 가장 잘 나타낸다. 클래스 스키마에는 스키마의 선언부에 선언된 변수들과 스키마 기술부에 서술된 동작 스키마를 나열할 수 있고, 이를 이용하여 추상화된 정보를 나타낼 수 있다.

4.4 클래스 스키마의 인스턴스

스키마에 대한 클래스의 인스턴스는 클래스를 표현한는 스키마안에서 선언된 변수들에 한정된 값들

을 나타내준다.

```
--- Book ----------------------------
제목, 저자, 출판사, ISBN: String
출판년도, 페이지수: N
-------------
slength 제목 ≤ 80
slength 저자 ≤ 40
slength 출판사 ≤ 40
slength ISBN = 13
(checksum ISBN) mod 11 = 0
0 < 페이지수; 1800 ≤출판년도 ≤ 2050
-------------------------------------
```

```
-------------------------------
book1: Book
----------
book1.제목 = Z in Practice
∧ book1.저자 = Rosalind Barden
∧ book1.출판사 = Prentuce Hall
∧ book1.ISBN = 0-13-124934-7
∧ book1.출판년도 = 1994
∧ book1.페이지수 = 407
-------------------------------
```

5. 프로토타입의 구조

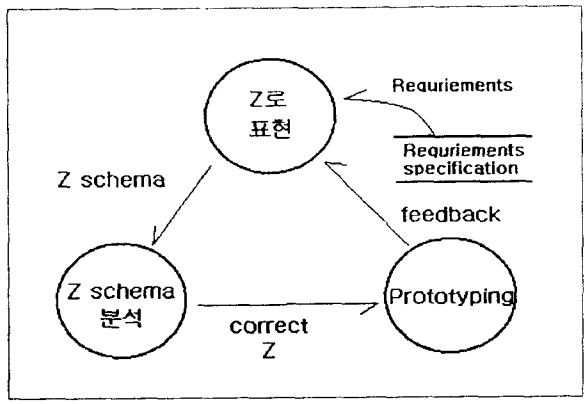
프로토타입을 구현하기 위해 우선 요구 문장에서 명사를 찾아 클래스 명으로 지정하고 동사를 찾아서 동작 스키마로 만든다. 형용사는 속성의 이름이 된다. 여기서 불필요하거나 반복되는 클래스 명이나 연산들을 삭제한다. 그 다음 Z specification으로 기술한다.

클래스 스키마를 이용하고 객체 지향 원리에 반영한다. Z schema 분석기에서 scope체크와 type 체크를 한다. 올바른 Z 명세 요구를 프로토타이핑한다. 여기서 수정이 필요한 Z 명세는 피드백되어 다시 Z specification를 수정한다[그림4].

6. 결론 및 향후 연구 방향

객체 지향 패러다임을 명확하고 간결하게 표현하기 위해 Z 명세언어를 사용하였다. 그러나 Z는 객체 지향 명세 언어가 아니다. 그러므로 클래스를 기술하기 위하여 Z 스키마의 숨김 연산자와 더 발전한 개념인 논리 한정 연산자와 투사 연산자를 사용하였다. 이를 클래스 스키마로 정의하였다. 클래스 스키마가 상속성, 다형성, 캡슐화 등의 객체지향 원리를 가지는가에 대한 프로토타이핑을 위해 프로토타입을 설계했다.

앞으로 연구방향은 클래스 스키마를 이용하여, Z의 의미적 문법에 최소의 변화를 가정하고 객체 지향 패러다임을 기술하는 연구가 필요하다. 그리고



[그림 4] 테스트 프로토타입의 구조

설계된 프로토타입의 구현과 구현된 C++프로그램으로 출력하는 증명이 필요하다.

참고 문헌

[BOOC 1994] Grady Booch, *Object-Oriented Analysis and Design with Applications,* Benjamin/Cummins, 1994

[FRAN 1997] France, R.B., Bruel, J.M., and Larrondo-Petrie, M.M., "An Integrated Object-Oriented and Formal Modeling Environment," The Journal of Object Oriented Programming, Vol.10, No.7, Nov/Dec. 1997

[KEND 1996] Kendall, P.A., *Introduction to System Analysis and Design 3th,* IRWIN, 1996

[POTT 1996] Potter, Ben, Sinclair, Jane, and Till, David, *An Introduction to Formal Specification and Z 2ed,* Prentice Hall, 1996

[PRES 1995] Pressman, Roger S., *Software Engineering-A Practitioner's Approach 3th,* McGraw-Hill, 1995

[SPIV 1998] Spivey, J. M., The Z Notation: A Reference Manual,1988 http://spivey.oriel.ox.ac.uk/~mike/ZRM

[RUMB 1991] Rumbaugh, J. 외 4명, *Object-Oriented Modeling and Design,* Prentice hall, 1991

[STEC 1987] Stepeny, S. and Lord. S.P., "Formal Specification of an Access Control System," Software-Practice and Experience, Vol.17, No.9, 1987, pp575-593

[강 1998] 강미영, 조영석, "정형 명세어에서 클래스 스키마의 유도," '98 추계 정보 처리 학회, 1998, pp502-505