2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

# 클래스 다이아그램의 정형 명세 변환

김진수 정제홍 김병수
건양대학교 정보전자통신공학부
jinskim@kytis.konyang.ac.kr

# A Formal Specification Translation of the Class Diagram

Jin-Soo Kim Je-Hong Jeong Byung-Soo Kim
Dept. of Information, Electronics & Communication Engineering,
Konyang University

요 약

일반적으로 크고 복잡한 소프트웨어 시스템은 커다란 다이아그램의 집합으로 구성되지만 이들 각각의 다이아그램들이 일관성이 있고 완전한가를 알기는 매우 어렵다. 이러한 문제를 해결하기 위하여 [1]에서는 최근 객체지향 개발에서 많이 사용되고 있는 UML 다이아그램들에 대한 일관성과 완전성을 검사하기 위하여 각 다이아그램들을 ER 모델로 표현하고, 각 다이아그램에 대한 공통된 표현으로 통합된 ER 다이아그램을 제공하였다. 이렇게 통합된 다이아그램은 일련의 집합과 함수들을 사용하여 정형적으로 명세되며, 이러한 정형 명세를 기반으로하여 일관성과 완전성 검사를 수행하기 위한 규칙을 제공하였다. 그러나 [1]에서 정형적으로 명세된 집합과 함수를 실제로 사용하기 위해서는 우리가 알고 있는 정형 명세 언어로 변환할 필요성이 있다. 따라서 본 논문에서는 이러한 집합을 추상자료형으로 명세하여 정의하고, 추상자료형으로 표현된 명세를 우리가 잘 알고 있는 Z 명세로 변환하여 검증 도구를 설계하는 기초로 사용하고자 한다. Z 명세는 쉽게 실행가능한 형태로 변환되어 검증 시스템을 구축할 수 있게 된다.

## 1. 서 론

최근에 객체지향 방법이 소프트웨어를 개발하는 탁월한 파라다임이 되고 있고 객체지향 방법이 기존의 개발 방법들에 비해 개발되는 소프트웨어의 품질을 향상시키고 생산성을 증가시킨다는 것은 이미 많이 알려져 있다. 일반적으로 크고 복잡한 소프트웨어 시스템은 커다란 다이아그램의 집합으로 구성되지만 이들 각각의 다이아그램들이 일관성이 있고 완전한가를 알기는 매우 어렵다. 이러한 문제를 해결하기 위하여 [1]에서는 최근 객체지향 개발에서 많이 사용되고 있는 UML 다이아그램들에 대한 일관성과 완전성을 검사하기 위하여 각 다이아그램들을 ER 모델로 표현하고, 각 다이아그램에 대한 공통된 표현으로 통합된 ER 다이아그램을 제공하였다. 이렇게 통합된 다이아그램은 일련의 집합과 함수들을 사용하여 정형적으로 명세되며, 이러한 정형 명세를 기반으로하여 일관성과 완전성 검사를 수행하기 위한 규칙을 제공하였다. 그러나 [1]에서 정형적으로 명세된 집합과 함수를 실제로 사용하기 위해서는 우리가 알고 있는 정형 명세 언어로 변환할 필요성이 있다. 따라서 본 논문에서는 이러한 집합을 추상자료형으로 명세하여 정의하고, 추상자료형으로 표현된 명세를 우리가 잘 알고 있는 Z 명세로 변환하여 검증 도구를 설계하는 기초로 사용하고자 한다. Z 명세는 쉽게 실행가능한 형태로 변환되어 검증 시스템을 구축할 수 있게 된다.

2장에서는 집합을 추상자료형으로 표현하는 과정을 보이고, 3장에서는 표현된 추상자료형을 Z 명세로 변환하는 과정을 제시하고, 4장에서 결론을 맺는다.

## 2. 집합의 추상자료형

추상자료형은 객체의 집합과 이들 객체에 적용되는 오퍼레이션의 집합의 조합으로 정의될 수 있으며 추상자료형의 정형 명세는 NAME, SETS, SYNTAX, SEMANTICS의 네개의 엔트리를 가지며

NAME 엔트리에서는 추상자료형의 종류가 선언된다[2]. [1]에서 집합 E의 각 원소는 하나의 집합으로 정의되며 추상자료형의 정형 명세에 대한 일반적인 형태를 이용하여 집합 E의 각 원소들을 추상자료형 집합으로 명세할 수 있다. [1]에서 집합 E의 첫 원소는 클래스들의 집합인 집합 C이며 Class 추상자료형으로 정형적으로 명세될 수 있다. Class 추상자료형은 클래스들의 집합이고 다음과 같이 NAME 엔트리에서 선언될 수 있다.

NAME
    set ( class )

Class 추상자료형은 SETS 엔트리에서 다음과 같은 세 개의 집합을 선언한다.

SETS
| | |
|---|---|
| S | 집합의 집합 |
| C | 클래스들의 집합 |
| B | Boolean 값인 참, 거짓의 집합 |

집합 S는 집합의 집합이고 집합 S의 각 집합 s는 클래스 다이아그램에서 그려진 클래스들의 추상화이다. 클래스 다이아그램에서 하나의 클래스를 그릴 때마다 클래스 c를 집합 s에 추가한다는 것을 의미한다. 또한 클래스 다이아그램에서 하나의 클래스를 삭제할 때마다 집합 s에서 클래스 c를 제거한다는 것을 의미한다. 하나의 클래스에 이름을 부여할 때마다 클래스 c의 이름이 집합 s의 원소인가 아닌가를 물어보아야 한다는 것을 의미한다. 이렇게 생각해보면 Class 추상자료형은 집합 S, C, B의 원소에 대해 작용할 수 있는 add, isinset, remove의 세가지 기본 오퍼레이션을 가지고 있어야 한다. 이러한 세 오퍼레이션의 구문은 SYNTAX 엔트리에서 다음과 같이 정의된다.

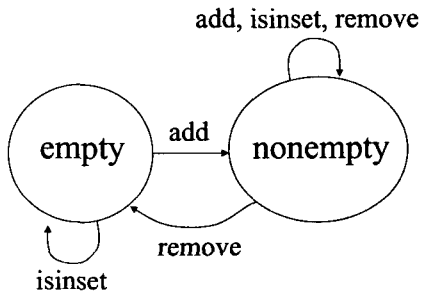
SYNTAX
| | |
|---|---|
| add : | C × S → S |
| isinset : | C × S → B |
| remove : | C × S → S |

하나의 집합은 보통 empty와 nonempty의 두가지 상태를 갖는데 집합의 원소가 하나도 없다면 그것은 empty 상태에 있다고 할 수 있고 적어도 하나의 원소라도 있다면 nonempty 상태가 된다.

집합 $s_1$이 어떤 상태이든 간에 오퍼레이션 add는 이 집합에 클래스 c를 추가할 것이고 이 오퍼레이션의 선행조건은 클래스 c가 집합 $s_1$의 원소가 아니어야 한다는 것이다. 그렇게 되면 이 오퍼레이션은 집합 $s_1$에 클래스 c를 추가하고 집합 $s_2$가 된다. 이 오퍼레이션의 후행조건은 클래스 c가 집합 $s_2$의 원소라는 것이다. 이 오퍼레이션을 수행하기 전에 집합 $s_1$이 empty 상태라면 이 오퍼레이션이 완료된 다음에 집합 $s_2$는 nonempty 상태가 되며, 이 오퍼레이션 수행전에 집합 $s_1$이 nonempty 상태였다면 이 오퍼레이션이 완료된 다음에도 집합 $s_2$는 nonempty 상태로 남아있게 된다. add 오퍼레이션의 의미는 SEMANTICS 엔트리에서 다음과 같이 명세된다.

SEMANTICS
(empty → nonempty) ∨ (nonempty → nonempty)
    {ASSUME  $c \notin s_1$}
        add( c : C, $s_1$ : S ) : $s_2$ : S
    {ASSERT  $s_2 = s_1 \cup \{c\}$}

유사하게 isinset 오퍼레이션과 remove 오퍼레이션이 정의될 수 있다. 이러한 상태와 오퍼레이션간의 관련성은 다음 <그림 1>의 오퍼레이션에 대한 상태전이 다이아그램과 같이 나타낼 수 있다.




<그림 1> 오퍼레이션에 대한 상태전이 다이아그램

[1]에서 집합 E의 다른 모든 원소들도 Class 추상자료형과 같이 추상자료형 집합으로 명세될 수 있고 이들 추상자료형의 정형적인 명세는 몇가지의 차이점을 제외하고는 Class 추상자료형과 거의 유사하게 표현될 수 있다.

### 3. 추상자료형의 Z 명세 변환

Z 언어에서 기본적인 추상자료 모델은 집합이고 하나의 집합은 타입이라고 하는 동일한 종류의 값들의 모임이라고 할 수 있다. [1]에서의 각 집합들은 이제 Z 명세에서 하나의 타입이 된다. Class 추상자료형을 예로 들어 설명하면 다음과 같다.

2장에서 Class 추상자료형 정형명세의 SYNTAX에서 add, isinset, remove의 구문을 정의하였고 Z 명세에서는 각 오퍼레이션에 대하여 하나의 응답을 생성할 것이며 이러한 오퍼레이션 응답들의 집합은 다음과 같이 타입 R로 정의된다.

```
R  ::=  added  |  alreadyadded
      | found  |  notfound
      | removed  |  undefined
```

Z 명세에서 하나의 타입은 값들의 집합이라고 할 수 있으며 하나의 타입이 가질 수 있는 값들의 수가 N이라면 값들의 집합은 $2^N$이 된다. 이러한 $2^N$ 집합의 모임을 타입의 멱집합(power set)이라고 하고 타입 이름앞에 P를 붙인다.

Class 추상자료형 정형명세의 SETS 엔트리에서 집합 C는 클래스들의 집합으로 선언된다. Z 명세에서 집합 C를 나타내기 위하여 classes를 사용한다면 집합 classes는 Z 명세에서 Class 타입의 멱집합의 부분집합이 된다. 유사하게 [1]에서 집합 plains, 집합 abstracts, 집합 parameterized도 Class 타입의 멱집합의 부분집합이 된다. 이들 집합은 TypeClass 스키마의 선언부에서 다음과 같이 선언된다.

```
---TypeClass-------------------------------
classes, plains, abstracts, parameterized : P Class
-------------------------------------------
plains && abstracts = { } ;
plains && parameterized = { } ;
abstracts && parameterized = { } ;
plains || abstracts || parameterized = classes
-------------------------------------------
```

Z 언어에서 하나의 관계는 순서화된 값들의 그룹의 집합이라고 할 수 있다. 각 그룹은 둘이상의 값을 가지고 있다. 여기에서 우리는 두 개의 값만을 갖는 경우에 대해서만 고려하기로 한다. 그룹의 첫 번째 값은 하나의 집합으로부터의 값이고 이 관계의 정의역이라고 하고 그룹의 두 번째 값은 또다른 집합으로부터의 값이며 이 관계의 치역이라고 한다. 정의역의 각 값들이 치역의 하나의 값에만 관계된다면 이관계는 함수라고 할 수 있다. 따라서 함수는 제약을 가진 관계가 된다. Z 언어에서 함수는 정의역 집합과 치역 집합의 뒤에 나오는 이름에 의해 선언된다.

일반적으로 하나의 함수는 두 집합과 관계된다. 따라서 Z 명세에서의 함수의 선언은 두 개의 타입이 표현된 하나의 스키마에 포함되어야 한다. Z 명세에서 [1]에서의 RfC 함수를 표현하기 위해서 RelClass 스키마에 TypeClass 스키마와 TypeRelationship 스키마를 합해서 표현한다. RelClass 스키마에서 RfC 함수와 RtC 함수를 다음과 같이 표현한다.

```
---RelClass-------------------------------
classes, plains, abstracts, parameterized :
    P Class ;
relationships, instantiations, inheritance :
    P Relationship ;
aggregations, associations  : P Relationship ;
rfc : Relationship --> Class ;
rtc : Relationship --> Class
-------------------------------------------
relationships = dom rfc ;
ran  rfc = classes ;
relationships = dom rtc ;
ran  rtc = classes
-------------------------------------------
```

각 추상자료형 집합과 함수는 Z 명세에서 하나의 상태 데이터라고 하는 집합으로 표현된다. 어떠한 오퍼레이션이 이 집합들의 값에 적용되기 전에 각 집합은 공집합이라고 가정한다. 모든 집합들을 공집합으로 만들기 위해서는 상태 데이터를 초기화하여야 한다. 초기화 오퍼레이션은 모든 집합에 수행된다고 간주한다. Z 명세에서는 집합의 시작상태는 집합의 이름으로 나타내고 집합의 종료상태는 집합의 이름뒤에 ' 를 붙여서 나타낸다. 따라서 Class 타입 집합의 시작상태와 시작상태들간의 관련성은 TypeClass 스키마에서 선언부와 predicate에 의해서 표현된다. 이 집합의 종료상태와 종료상태들간의 관련성은 다음과 같이 TypeClass' 스키마에서 다음과 같이 표현된다.

```
---TypeClass'-------------------------------
classes', plains', abstracts', parameterized' :
      P Class
---------------------------------------
plains' && abstracts' = { } ;
plains' && parameterized' = { } ;
abstracts' && parameterized' = { } ;
plains' || abstracts' || parameterized' = classes'
---------------------------------------
```

유사하게 다른 타입의 집합과 함수들도 이러한 형태로 표현할 수 있다. 먼저 Class 추상자료형의 add 오퍼레이션에 대한 Z 변환을 설명하면 다음과 같다.

add 오퍼레이션의 입력은 하나의 c 클래스와 클래스들의 집합인 $s_1$이다. c 클래스는 cname?으로 표현된다. ?으로 끝나는 변수 이름이 오퍼레이션의 입력으로 사용되는 것이 Z의 규약이다. 집합 $s_1$은 add 오퍼레이션에 대한 시작상태를 나타내는 TypeClass 선언에 의해 표현되는 classes를 의미한다. add 오퍼레이션의 출력은 그 안에 c 클래스를 포함하고 있는 클래스들의 새로운 집합인 $s_2$가 된다. 새로운 집합 $s_2$는 add 오퍼레이션에 대한 종료상태를 나타내는 TypeClass' 선언에 의해 표현되는 classes'를 의미한다. add 오퍼레이션에 대한 TypeClass 시작상태 선언과 TypeClass' 종료상태 선언은 Delta TypeClass 선언으로 표현된다. add 오퍼레이션의 응답은 r!로 표현되는데 변수 이름뒤에 !이 붙으면 오퍼레이션의 응답으로 사용된다는 것을 의미한다.

add 오퍼레이션의 선행조건은 cname?이 classes 집합의 원소가 아니라는 것이고, add 오퍼레이션의 후행조건은 cname? 클래스가 classes' 집합의 원소라는 것이다. cname? 클래스가 classes 집합에 추가된 다음 add 오퍼레이션의 응답 값이 추가되는데 이것은 predicate에 의해 표현된다. add 오퍼레이션의 정형명세는 다음과 같이 ClassAdd에 대한 Z 스키마로 표현될 수 있다.

```
---ClassAdd---------------------------------
Delta TypeClass ;
cname? : Class ;
r! : Response
---------------------------------------
cname? notin classes ;
classes' = classes || { cname? } ;
r! = added
---------------------------------------
```

Class 추상자료형의 isinset 오퍼레이션과 remove 오퍼레이션도 유사한 과정으로 Z 명세로 변환될 수 있다.

## 4. 결 론

[1]에서 정형적으로 명세된 집합과 함수를 실제로 사용하기 위해서는 우리가 알고 있는 정형 명세 언어로 변환할 필요성이 있다. 본 논문에서는 이러한 집합을 추상자료형으로 명세하여 정의하였고, 추상자료형으로 표현된 명세를 우리가 잘 알고 있는 Z 명세로 변환하였다.

이렇게 변환된 Z 명세는 [1]에서 UML 다이아그램간의 일관성과 완전성을 검사할 수 있는 검증 시스템에 대한 정형 명세라고 할 수 있으며, 쉽게 실행가능한 형태로 변환되어 검증 시스템을 구축할 수 있는 토대를 마련할 수 있게 된다.

향후연구과제로는 UML의 모든 다이아그램 요소와 일관성과 완전성 검사를 위한 모든 규칙들을 Z 명세로 변환하여 자동화된 검증 시스템을 구축하는 것이다.

## 참고문헌


[1] 김진수 외, "UML 다이아그램간의 일관성과 완전성을 위한 검증 규칙 생성에 관한 연구", 한국멀티미디어학회 논문지, 제3권, 제3호, 290-298, 2000

[2] Thomas, P., Robinson, H. and Emms, J., *Abstract Data Types : their Specification, Representation and Use*, Clarendon Press, Oxford, 1988

[3] 김진수 외, "프레임을 이용한 다이아그램 검증 모델의 설계", 한국정보처리학회 추계학술발표논문집(하), 제7권, 제2호, 1381-1384, 2000

[4] Terry Quatrani, *Visual Modeling with Rational Rose and UML*, Addison-Wesley, 1998

[5] Robert H. Bourdeau and Betty H. Cheng, "A Formal Semantic for Object Model Diagrams", IEEE Trans. on Software Engineering, Vol. 21, No. 10, pp. 799-821, 1995

[6] Paul Hramon, Mark Watson, *Understanding UML : The Developer's Guide,* Morgan Kaufmann Publishers, Inc., 1997