DOI: 10.3745/KIPSTD.2008.15-D.5.629

OCL에 바탕을 둔 메트릭 기술 언어를 이용한 메트릭의 표현 방법 개선

김 태 연[†] · 김 윤 규[††] · 채 흥 석[†††]

요 약

보다 정확한 표현을 위하여 Object Constraint Language(OCL)을 이용하여 메트릭을 기술하는 연구들이 있다. 그러나 OCL은 Unified Modeling Language(UML)에서 구조적 제약사항을 기술하기 위한 언어이기 때문에 메트릭을 표현하기에는 부적절하다. 본 논문에서는 이러한 문제점을 해결하기 위하여 메트릭을 표현하는 상위 수준의 언어인 Metric Description Language(MDL)을 제안한다. MDL은 복잡한 메트릭의 분할 기술, 집계함수, 대상간의 자동 탐색 기능을 지원하여 OCL의 복잡성 문제점을 해결하고 있다. 또한 본 논문은 MDL을 기술하고 UML 모델을 대상으로 메트릭의 자동 계산을 지원하는 도구를 개발하였다. 그리고 사례연구로 기존에 제시된 다양한 유형의 메트릭을 MDL로 기술하였으며 OCL로 표현된 메트릭보다 단순함을 확인하였다.

키워드 : 소프트웨어 메트릭, OCL, UML 다이어그램

Method for Improving Description of Software Metrics Using Metric Description Language Based on OCL

Tae Yeon Kim[†] · Yun Kyu Kim[††] · Heung Seok Chae[†††]

ABSTRACT

Because most metricsin the literatures are described by a natural language, they can be interpreted in an ambigous manner. To cope with this problem, there are some researches to express based on Object Constraint Language(OCL). Because OCL has been proposed to describe structural constraintsfor Unified Modeling Language(UML) diagrams, it is difficult and awkward. In this paper, we propose *Metric Description Language*(*MDL*) which is a high level language to describe metrics. *MDL* supports a modular description of complex metrics, aggregation function, and automatic navigation between entities. Moreover, we develop MetriUs for describing metrics using *MDL* and supporting an automated computation for UML diagrams. In a case study, we have described a variety of existing metrics using *MDL* and found that MDL contributes to producing simpler expression of metrics than OCL.

Keywords : Software Metrics, Ocl, Uml Diagram

1. 서 론

기존의 연구들에서 소프트웨어의 크기, 복잡도, 응집도, 결합도 등을 측정하는 메트릭이 제안되었다. 예를 들어크기를 측정하는 Lines Of Code(LOC)[1], 복잡도를 측정하는 Cyclomatic Complexity(CC)[2], 응집도를 측정하는 Lack of Cohesion Of Methods(LCOM)[3], 결합도를 측정하는 Response For a Class(RFC)[3]등이 대표적이다. 또한 개발 단계 초기에 디자인 메트릭을 이용하여 유지보수성, 신뢰성 등을 예측하기 위한 실험적 연구들이 수행되었다. 특히 객체지향 소프트웨어 개발 단계의 초기에 디자인 메트릭을 사용하면 모델러와 디자이너에게 품질에 영향을 미치는 결정을 도와줄 수 있으며 개발 단계 후반에 불필요한 수정 없이 고품질의 소프트웨어를 개발 할 수 있다[4]. 이런 이유로 다수의 디자인 메트릭이 UML 다이어그램의 복잡도와 크기를 측정할 수 있도록 개발되었다.

그러나 기존 메트릭의 일부는 자연어를 이용하여 그 정의를 하고 있어 여러가지 의미로 해석되는 문제를 가지고 있다. 예를 들어Henry의 Number Of local Method(NOM) 메트릭은 클래스의 메소드 수를 의미한다[5]. 그러나 Henderson에 의하면 Henry의 NOM메트릭은 정확히 클래스의 매소드 수가 아니라 공용 메소드의 수로 정의했다[6].

※ "본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음" (IITA-2008-(C1090-0801-0032))
† 준 회 원 : 부산대학교 컴퓨터공학과 석사
†† 준 회 원 : 부산대학교 컴퓨터공학과 석사
††† 정 회 원 : 부산대학교 컴퓨터공학과 조교수
논문접수 : 2008년 4월 28일
수 정 일 : 1차 2008년 6월 9일, 2차 2008년 6월 25일
심사완료 : 2008년 6월 27일

기존 일부 연구자들이 수학과 집합의 정의를 이용하여 응집도와 결합도에 관한 메트릭 프레임웍을 정의하고 수식으로 표현하였다[7, 8]. 그러나 측정 메트릭의 특정 유형을 위한 프레임웍이며 다른 유형의 메트릭에 관한 지원이 없어 일반적으로 사용될 수 없다. 게다가 비표준 언어를 사용함으로써 프레임웍을 구현하기 위해 필요한 도구의 지원이 없어 이것을 직접 구현해야 한다.

이러한 문제점을 해결하기 위하여 메트릭 기술 언어로서 Object Constraint Language(OCL)[9]을 이용하는 연구가 있어왔다. OCL은 Unified Modeling Language(UML)에서 구조적 제약 사항을 기술하기 위한표준언어이며UML 다이어그램의정보들을 접근하기 위한 풍부한 연산자들을 지원한다. OCL을 이용하여 메트릭을 기술하면 표준언어의 사용으로 언어의 접근성이 높으며, OCL 컴파일러, 평가기, 상용도구, OCL 자바코드 변환기 등의 지원도구가 있다. 게다가 OCL의 특징인 수학적 기초를 바탕으로 메트릭의 정의를 명확하게 기술할 수 있다.

그러나 OCL을 메트릭 기술 언어로 사용하는 경우에UML 메타모델의 복잡함으로 인하여 메트릭 기술이 어려워진다[10]. 그 이유는OCL이 메타모델의 엔터티들을 탐색하기 위해서는 이들 사이의 경로를 단계적으로 탐색해야 하기때문이다. 또한 OCL은 단순한 집계연산만을 지원하고 있어서 일부 메트릭의 기술시 다른 연산들을 복합적으로 사용해야만 한다. OCL표현식의 복잡함은 메트릭을 기술시 오류 발생 가능성이 높으며 메트릭 정의의 가독성이 떨어진다. 따라서 메트릭 기술자는 메트릭의 기술에 많은 노력이 필요하다.

본 논문에서는 메트릭을 기술하기 위하여 OCL에 바탕을 둔 상위 수준의 언어인 *Metric Description Language*(MDL)을 제안한다. MDL은 OCL의 복잡성 문제를 해결하기 위하여 첫 번째로UML 메타모델 대상간의 경로를 자동으로 탐색하여 탐색 표현식을 생성한다. 즉, MDL은 복잡한 탐색표현식을 직접 기술하지 않아도 된다. 둘째로 MDL은 확장된 집계함수를 지원한다. 이를 통하여 MDL은 메트릭 기술시 복합연산들이 불필요하다. 셋째로 MDL은 복잡한 메트릭을 분할하여 기술할 수 있도록 서브메트릭을 지원한다. 즉, 서브 메트릭은 복잡한 문제를 해결하는 방법중 하나인 분할하여 정복하는 기법을 메트릭 기술 시에도 적용할 수 있도록 해준다. 이를 통하여 메트릭 기술자는 메트릭 기술을 보다 용이하게 할 수 있다.

MDL이 OCL을 기반으로 하는 이유는 OCL의 수학적 기초를 바탕으로 명확하게 메트릭의 정의를 기술할 수 있으며 표준언어로서 OCL을 지원하는 도구들를 이용하여 효율적으로 메트릭을 계산할 수 있기 때문이다. 집합론, 술어논리, 수학적 의미론을 기초로 만들어진 OCL의 특징인 명확성과 정확성은 MDL이 OCL로 변환되면 그 특징을 그대로 가진다. 또한 OCL의 컴파일러 및 지원 라이브러리를 이용하면 MDL 컴파일러와 라이브러리를 구축하는데 드는 비용을 줄일 수 있다. 사용자는OCL평가 도구를 사용하여 MDL에서 OCL로 변환한 메트릭 표현식으로부터 메트릭을 측정할 수 있다.

본 논문에서는MDL 를 지원하고 UML 산출물을 대상으로 기술된 메트릭을 측정할 수 있는MetriUs도구를 개발하였다. MetriUs는MDL을 OCL로 변환하는 기능과 사용자가 디자인한 UML 모델과 변환된 메트릭 기술을 입력으로 그 결과를 계산하는 기능으로 구성되어 있다. 그리고 사례연구로 기존에 제시된 다양한 유형의 메트릭을 MDL로 기술하였다. 그 결과로 MDL은 크기, 복잡도, 결합도, 은닉성, 상속과 같은 다양한 측정유형의 메트릭을 표현할 수 있었다. 또한 MDL로 표현된 메트릭은 OCL로 표현된 메트릭보다 단순함을 확인하였다.

본 논문은 다음과 같이 구성된다. 먼저 2장에서는 OCL을 이용하여 메트릭을 기술하는 연구와 문제점을 소개한다. 다음으로 3장에서는 MDL 언어의 문법과 사용사례를 설명하고 4장에서는 MDL에서 OCL로 변환하는 알고리즘을 설명한다. 5장에서는 지원도구를 설명하고 적용사례를 기술한다. 6장에서는 메트릭 기술언어와 관련된 연구를 소개한다. 마지막으로 7장에서 결론 및 향후연구 방향을 밝힌다.

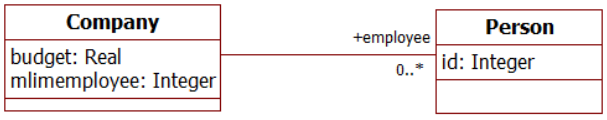
2. OCL을 이용한 메트릭의 표현

2.1 OCL

Object Constraint Language(OCL)은 UML 모델의 문맥에서 구조적 제약사항을 체계적으로 기술하기 위한 표준언어이다[9, 11]. OCL은 수학적 배경을 기초로 하고 있으므로 UML 다이어그램에서 OCL로 기술된 제약사항은 자연어로 기술된 것과 비교시 그 의미를 정확하고 명확하게 전달한다. 그리고 OCL은 명세 언어이므로 OCL 표현식은 시스템 및 모델의 상태를 변화시키지 않으며 프로그램 로직 및 제어 흐름을 기술할 수 없는 특징을 가진다. OCL은 UML 모델의 불변식, 클래스 메소드의 pre/post 조건과 가드 조건등을 기술하기 위하여 사용된다. 더욱이 OCL은 모델 검증[12, 13]과 테스트 케이스 생성[14]과 같은 쟁점들을 해결하기 위하여 사용된다.

(그림 1)은 UML 클래스 다이어그램의예이다[15]. 클래스 다이어그램은 시스템의 structural property를 시각적으로 표현한다. 이 그림에는 Company와 Person 클래스가 있다. Company 클래스는 budget과 mlimemployee 프로퍼티를 가지며, Person 클래스는 id 프로퍼티를 가진다. 두 클래스는 연관관계가 있으며 연관관계의 role name은 employee이다. 이 연관관계의 multiplicity는 0에서 *이다. 그러나 (그림 1)은 모델의 구조적 제약사항을 표현하진 못한다.

이것을 표현하기 위한 표준언어인 OCL를 이용하면 제약사항을 기술할 수 있다. (그림 2)는 “모든 Company 클래스의 객체는 budget 속성이 0보다 커야한다.”인 제약사항을 OCL



(그림 1) 클래스 다이어그램의 예제

```
context Company
inv NoBudgetNegative :
self.mlimemployee > self.employee→size()
```

(그림 2) OCL 표현식의 예제

NOM = number of local methods.
The number of local methods defined in a class may indicate the operation property of a class. The more methods a class has, the more complex the class's interface.

(그림 3) NOM 메트릭의 정의

표현식을 이용하여 기술한 예이다[15]. context에는 제약사항을 적용할 대상을 기술한다. inv는 불변식을 뜻한다. 불변식의 이름은 inv 다음에 위치하며 OCL 표현식의 body는 콜론(:) 이후이다. Body의 self 키워드는 적용 대상인 Company를 뜻한다. dot(.) 표기법은 객체간의 연관관계 및 객체의 속성과 오퍼레이션을탐색할 수 있다. Company 클래스에서 Person 클래스로의 탐색은 연관관계의 role name인 employee을이용한다. size() 함수는 collection형의 원소의 개수를 반환한다. 불변식은 객체의 라이프타임 동안 참이되어야 하는 제약사항이다[11]. 그러므로 (그림 2)의 OCL 표현식은 Company 클래스의 mlimemployee 속성 값이 employee의 수보다 큰 경우에만 참이 된다.

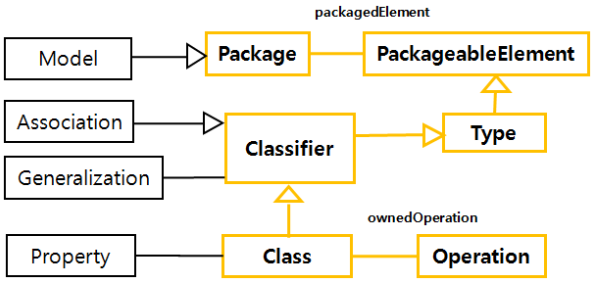
2.2 메트릭 표현을 위한 OCL의 활용

(그림 3)은 NOM 메트릭의 정의이다. 하지만 NOM의 정의는 자연어로 기술되어 있어 클래스 메소드에 관한명확한 정의가 없어 혼란이 발생한다. Henderson은 자신의 저서에서 NOM의 정확한 의미를 논문의 저자에게 확인하였으며 그 의미는 클래스의 인터페이스 수를 의미한다고 기술하였다.

이를 해결하기 위하여 메트릭 기술 언어로 OCL이 사용되었다. OCL은 구조적 제약사항을 기술할 뿐 아니라 UML 메타모델을 대상으로 하는 메트릭을 기술하는 언어로도 사용할 수 있다.

(그림 4)는 메트릭 기술에 사용되는 UML 메타모델의 일부이다[16]. 이 그림에서는 메타모델의 속성 및 오퍼레이션 정보는 생략하고 모델간의 관계만을 표시하였다. Package와 PackageableElement 엔터티는 연관관계를 가진다. PackageableElement에서 Class 엔터티까지는 3개의 일반화 관계가 있다. Class 엔터티는 Operation과 Property 엔터티와 연관관계를 가진다.

(그림 5)는 OCL을 이용하여 NM 메트릭을 기술한 예이다[4]. Number Of Methods(NM) 메트릭은 클래스 다이어그램에서 모든 클래스의 메소드 중 상속된 메소드를 제외한 메소드의 총 수를 뜻한다[4]. Line 1은 Context가 Package 엔터티임을 의미한다. Line 2는 OCL 표현식이 불변식임을 나타내며 이름이 NM임을 뜻한다. (그림 4)에서 Package는 PackageableElement 엔터티와 연관관계를 가진다. 그러므로 Line



(그림 4) UML 메타 모델

```
1:  context Package
2:  inv NM :
3:  self.packagedElement
4:  →select( p:PackageableElement | p.oclIsTypeOf (Class)
    =true )
5:  .oclAsType(Class).ownedOperation→size()
```

(그림 5) OCL을 이용하여 Number Of Methods(NM) 메트릭을 기술한 예

3에서는 OCL의 탐색 표현인 dot 표기법을 이용하여 연관관계의 role name인 packagedElement로 탐색할 수 있다. PackageableElement 엔터티를 탐색하면 이 클래스의 객체가 collection 형으로 반환된다.

Line 4에서 사용된 select() 함수는 해당 집합에서 조건을 만족하는 객체만을 반환한다. 그리고 oclIsTypeOf(Type) 함수는 파라미터로 넘겨받은 형과 해당 집합이 같은 형이면 참값(true)을 반환한다. 따라서 Line 4에서는 PackageableElement 엔터티 중 형이 Class인 객체를 반환한다. Line 5에서는 Class 엔터티에서 Operation 엔터티로 탐색하여 Operation 객체의 집합을 반환한 후 size() 함수를 이용하여 그 개수를 반환한다.

메트릭 기술 언어로서 OCL은 다음과 같은 장점이 있다.

- 명확성, 정확성, 이해성 : OCL은 자연어에 비하여 명확하고 정확하게 메트릭을 정의할 수 있다. OCL의 우수한 특징은 집합론, 술어논리, 수학적 의미론[17]을 기초로 하고 있는 점이다. 따라서 자연어를 이용하여 기술된 메트릭에 비하여 OCL을 이용하여 기술된 메트릭은 모호하지 않다. 그리고 모델링 언어는 수학의 견고성과 정확성이 필요하지만 자연어처럼 사용이 쉬워야 하므로 OCL은 수학적 기호를 사용하는 대신에 동일한 의미를 평문으로 표현한다. 따라서 메트릭 디자이너는 OCL로 기술된 메트릭이 수학적 표기법과 동일한 수준으로 명확하고 정확하게 메트릭을 이해할 수 있으며 보다 쉽게 메트릭을 읽고 기술할 수 있다.

- 표준언어 : OCL은 객체지향 분석과 디자인을 위한 UML, OMG 표준언어이다[11]. UML 사용의 증가로 OCL 사용자는 점점증가하고 있다. 또한 OCL을 UML모델에 적용할 수 있도록 지원하는 도구 역시 증가추세이다.

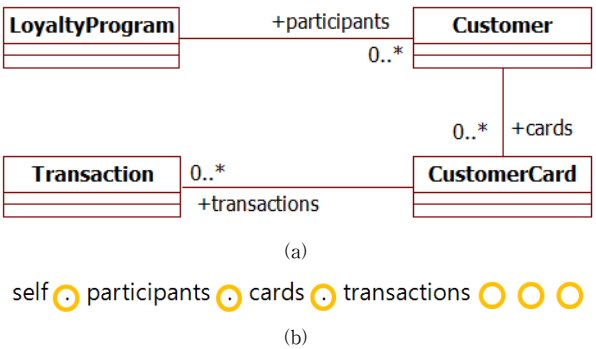
Together 2007[18], Eclipse MDT[19]는 OCL을 이용하여 UML 모델에 구조적 제약사항을 기술할 수 있다. 또한 Together 2007은 메트릭 기술 언어로서 OCL을 지원하는 대표적인 도구이다. Octopus[20]는 OCL을 JAVA 코드로 변환해 준다. 이 밖에도 OCL complier[21] 및 OCL Evaluator[22]가 있다.

2.3 문제점

2.2에서 소개된 장점에도 불구하고 메트릭 기술 언어로서 OCL은 아래와 같은 이유로 메트릭을 위한표현식이 매우 복잡해진다. 복잡한 OCL 표현식 때문에 메트릭 기술자는 OCL을 이용하여 메트릭을 기술할 때 많은 노력이 필요하다. 일반적으로 언어의 표현식이 복잡하면 오류가 발생할 확률이 높으며 작성된 표현식의 가독성 또한 낮아진다.

- **탐색 표현식의 복잡함** : OCL의 질의표현식은 대부분 복잡한 탐색 표현식이 된다. 그 이유는 한 엔터티에서 다른 엔터티로 접근하기 위해서는 단계적으로 접근해야 하기 때문이다. 따라서 UML 메타모델이 각각의 엔터티와 복잡한 관계를 형성하고 있으므로 원거리 엔터티를 접근하기 위해서는 길고복잡한 탐색 표현식이 불가피하다[15, 23]. (그림 6)에 (a)는 [11]에서 소개된 클래스 다이어그램의 일부이며 (b)는 클래스간의 탐색을 표현하는 OCL 표현식이다. LoyaltyProgram 클래스에서 Transaction 클래스를 탐색하기 위해서 OCL의 dot 표기법을 반복해서 사용하고 있다.

- **집합 함수의 단순함** : OCL은 모든 collection 형에 size(), sum() 연산을 지원한다. 비록 OCL이 집합 함수를 지원하지만 이들 함수로 표현하기가 적절하지 않은 메트릭이 있다. (그림 7)은 클래스에 정의된 메소드의 visibility가 public인 경우 가중치를 3의 값으로 합산하는 Weighted-Method 메트릭을 OCL 표현식으로 기술한 예이다. 공용 메소드에 가중치를 부여하기 위해서 부득이 OCL의 iterate() 연산을 사용하였다. collection 집합의 원소들은 각각 3의 값으로 누적하여 합산된다.



(그림 6) OCL의 복잡한 탐색 표현식

```
context Class
inv WeightedMethod :
self.ownedOperation→ select(o:Operation|o.visibility
=VisibilityKind::public)
→iterate( iter ; result:Integer=0 | result+3)
```

(그림 7) WeightedMethod 메트릭의 OCL 표현식

- **낮은 가독성** : OCL로 표현된 메트릭은 탐색표현식의 복잡함과 집합 연산의 단순함으로 인하여 가독성에 문제가 있다. 탐색으로 인하여 길고 복잡한 표현식을 사용하면 탐색경로의 클래스에 관하여 모두 알고 있어야 하며 OCL 표현식의 작성, 가독 및 이해가 매우 어렵게 되므로 [11]의 저자는 이와 같은 표현식을 피하라고 권고한다. 더욱이 OCL에서 제공하는 size() 함수는 단순히 collection 집합의 원소의 개수를 반환한다. 따라서 collection 집합의 각 원소별로 가중치를 부여하여 카운팅을 하는 경우 불필요한 OCL 표현식을 추가해야만 한다.

3. 메트릭 기술 언어 : MDL

본 절에서는 2절에서 설명한 메트릭 언어로서 OCL의 문제점을 해결하기 위하여 제안된 메트릭 기술 언어인 MDL을 소개한다. 먼저 MDL의 문법과 각 구문의 의미를 설명하고 다음으로 예제를 통하여 MDL로 기술된 메트릭을 살펴보고 사용사례를 설명한다. 마지막으로 OCL과 MDL로 기술된 동일한 메트릭을 비교하여 MDL이 OCL의 메트릭 기술 언어로서의 문제점을 어떻게 해결하였는지 보인다.

3.1 MDL

MDL은 UML 산출물을 대상으로 메트릭을 기술하는 언어이다. MDL은 OCL에 바탕을 두고 있으며 MDL로 기술된 메트릭은 OCL로 변환되어 계산된다. OCL이 수학적 배경을 기초로 하는 언어이므로 MDL로 기술된 메트릭은 명확하고 정확하다. MDL은 하나의 메트릭을 몇 개의 하위 메트릭으로 분할하여 기술할 수 있으므로 표현이 복잡한 메트릭을 단순하게 기술할 수 있다. 그리고 MDL은 UML 메타모델의 엔터티들을 대상으로 메트릭 계산에 필요한 사칙연산, 합산, 카운팅 연산을 지원한다.

(그림 8)은 MDL의 문법일부를 Backus-Naur Form(BNF)를 이용하여 정의한 것이다. 이 그림에서 보는 것처럼 MDL은 크게 <METRIC definition>과 <SUBMETRIC definition> 규칙으로 이루어져 있다.

- <METRIC definition>은 기술하고자 하는 메트릭에 대하여 이름, 적용대상 및 평가식을 정의하는 표현식이다. 이 규칙은 MDL4UML 이용하여 메트릭을 기술할 때 필수적으로 작성해야하는 부분이며 메트릭 정의표현식에서 한 번만 사용할 수 있다.

```
<MDL> ::=          <METRIC definition> <SUBMETRIC definitions>
<METRIC definition> ::=    "METRIC" <metric name> "FOR" <entity name>
"[" <condition> "]"
["VALUE" <value expression>] ["TYPE" <type> ]
    <entity name> ::=<meta class name in UML metamodel>
    <condition> ::=<ocl expression>
    <value expression> ::= <single value> | <expression>
    <single value> ::= <digit>* | <aggregation function>
    <expression> ::= <expression> <operator> <expression> |
                     <operand>
     <aggregation function> ::= SUM "(" <metric name> ")" | CNT "(" <metric name> ")"
     <operand> ::= <digit>* | <metric name>|<aggregation function>
     <operator> ::=  + | - | * | /
     <type> ::= Integer | Real
<SUBMETRIC definitions> ::= <SUBMETRIC definition> |
          <SUBMETRIC definitions> <SUBMETRIC definition>
<SUBMETRIC definition> ::=  "BEGIN"
<SUBMETRIC body> <SUBMETRIC definitions>
"END" |
<SUBMETRIC body>
<SUBMETRIC body> ::=    "SUBMETRIC" <metric name> "FOR" <entity name>
"[" <condition> "]"
["VALUE" <value expression>] ["TYPE" <type>]
```

(그림 8) MDL의 문법

- ◆ <metric name>은 메트릭의 이름이다. 메트릭 이름은 알파벳으로 작성한다.
- ◆ <entity name>은 UML 표준문서에서 정의하고 있는 메타 클래스의 이름이다. 엔터티 이름은 사용자가 임의로 정의할 수 없다.
- ◆ <condition>은 엔터티에 적용할 조건이다. 조건은 OCL 표현식을 사용하여 작성한다.
- ◆ <value expression>은 메트릭을 계산하는 계산식이다. 이 구문에서는 피연산자와 연산자를 사용한다. 피연산자는 숫자, 하위 메트릭 이름, 집계함수(CNT, SUM)이며 연산자는 사칙연산(+, -, *, /)이다. 생략하는 경우에 기본값으로 1을 사용한다.
- ◆ <type>은 메트릭의 계산 결과 값의 타입이다. Integer와 Real 형을 지원하며 생략하는 경우에 기본값은 Integer이다.

- <SUBMETRIC definition>은 하위 메트릭들을 기술한다. 이 구문은 <METRIC definition> 뒤에 기술해야 하며 여러 번 반복해서 사용할 수 있다.
- <SUBMETRIC body>는 하위 메트릭의 이름, 적용대상 및 평가식을 정의한다. <METRIC definition>부분과 매우 유사하다. 각 구문의 설명은 <METRIC definition>을 참조한다.

3.2 MDL 사용 예

MDL에서 메트릭을 기술하는 유형에는 단순유형과 다중유형이 있다. 단순 유형은 각각 METRIC과 SUBMETRIC 하나로 구성되어 있다. 단순 유형은 메트릭을 MDL로 기술하기 위한 최소의 구성이다. METRIC 혹은 SUBMETRIC 하나만으로는 메트릭을 기술할 수 없다. 다중 유형은 METRIC 하나와 SUBMETRIC 두개 이상으로 구성되어 있다. 다중 유형은 표현이 복잡한 메트릭을 분할하여 기술하는 경우에 유용하다.

(그림 9)는 NM 메트릭을 MDL로 기술한 예이다. 예제에서 METRIC 이름은 NM이며 측정대상은 Package이다. SUBMETRIC의 이름은 NOO이며 측정대상은 Class의Operation이다. 단순 유형이므로 더 이상 간략히 기술할 수 없다. 메트릭 값은 NOO SUBMETRIC의 원소의 개수이다.

(그림 10)은 ClassSize 메트릭을 MDL로 기술한 예이다. ClassSize 메트릭은 클래스의 메소드 수와 속성의 수를합하여 계산된다. OC, AC SUBMETRIC은 각각메소드의 수와 속성의 수를 카운팅하기 위하여 분할하여 기술 하였다. METRIC의 값은 OC SUBMETRIC의 원소의 개수와 AC SUBMETRIC의 원소의 개수를 합산한 값이다.

메트릭 기술 언어로서 MDL는 OCL과 비교하여 다음과 같은 특징이 있다.

- 메트릭을 분할하여 기술 : 복잡한 메트릭을 기술하는 경우에 SUBMETRIC을 사용하여 메트릭을 하위 메트릭들로 분할하여 처리할 수 있다. 예를 들어, (그림 11)는 클래스 다이어그램에서 클래스의 추상화 비율을 의미하는 AR 메트릭을 OCL과 MDL를 이용하여 각각기술한 예이다. (그림 11 (a))의 OCL 표현식에서는 Interface의

```
METRIC NM FOR Package Value CNT(NOO)
    SUBMETRIC NOO FOR Class.Operation
```

(그림 9) MDL을 이용하여 NM 메트릭을 기술한 예

```
METRIC ClassSize FOR Class VALUE CNT(OC)+CNT(AC)
SUBMETRIC OC FOR Class.Operation
SUBMETRIC AC FOR Class.Property
```

(그림 10) MDL을 이용하여 ClassSize 메트릭을 기술한 예

```
1: context Package inv AR :
2: self.packagedElement →select(P1|P1.oclIsTypeOf(Class)).
oclAsType(Class)→select(isAbstract=true)→size()  +
3: self.packagedElement →select(P3|P3.oclIsTypeOf(Interface)).
oclAsType(Interface) →size()                    /
4: ( self.packagedElement →select(P4|P4.oclIsTypeOf(Class)).
oclAsType(Class) → size()   +
5:     self.packagedElement → select(P5|P5.oclIsTypeOf
(Interface)).oclAsType(Interface)→size()  )
```

(a) AR 메트릭의 OCL 표현식

```
1: METRIC AR FOR Package TYPE Integer VALUE
CNT(NOAC) + CNT(NOI) / ( CNT(NOC) + CNT(NOI) )
2:   SUBMETRIC NOC FOR Class TYPE Integer VALUE 1
3:   SUBMETRIC NOAC FOR Class[isAbstract=true] TYPE
Integer VALUE 1
4:   SUBMETRIC NOI FOR Interface TYPE Integer VALUE 1
```

(b) AR 메트릭의 MDL 표현식

(그림 11) OCL과 MDL 표현으로 기술된 AR 메트릭의 예

개수를 카운팅하는 동일한 표현식이 중복되어 사용되었다(3,5 행). 그러나 (그림 11 (b))의 MDL 표현식에서는 이 부분을 NOI SUBMETRIC으로 한번만 표현하였다(4 행).

- 탐색 표현식의 미사용 : MDL에서는 OCL과 비교하여 UML 메타모델의 엔터티 사이를 탐색하는 표현식이 없다. 그 이유는 MDL에서는 두 엔터티 사이의 탐색 경로를 자동으로 찾기 때문이다. 예를 들어, (그림 10)은 클래스 개수를 의미하는 Number of Class(NC) 메트릭의 예이다. 그림 12 (a)의 OCL 표현식에서는 "." 표현식을 이용하여 두 엔터티 사이를 탐색한다(2 행). 또한 select() 함수와 oclIsTypeOf() 함수를 이용하여 일반화 관계도 탐색한다(3 행). 그러나 (그림 12(b))의 MDL 표현식에서는Package와 Class 사이에 탐색 표현식이 존재하지 않는다.

- 집계함수의 지원 : MDL에서는 집합의 원소의 개수를 구하는 CNT() 함수와 원소들을 특정 값으로 누적하여 합산하는 SUM() 함수를 지원한다. (그림 13)은 MDL

```
1: context Package inv NC :
2: self.packagedElement
3: →select(P1|P1.oclIsTypeOf(Class)).oclAsType(Class)→iterate
(iter1|Cla1:Integer;Cla1+=1)
```

(a) NC 메트릭의 OCL 표현식

```
1: METRIC NC FOR Package VALUE CNT(NOC)
2:   SUBMETRIC NOC FOR Class
```

(b) NC 메트릭의 MDL 표현식

(그림 12) OCL과 MDL 표현으로 기술된 NC 메트릭의 예

```
1   METRIC WeightedMethod FOR Class VALUE SUM(PUB)
2       SUBMETRIC PUB FOR Operation [visibility =
VisibilityKind::public] VALUE 3
```

(그림 13) MDL을 이용하여 WeightedMethod 메트릭을 기술한 예

을 이용하여 WeightedMethod 메트릭을 기술한 예이다. MDL이 집계함수를 지원하므로 (그림 7)의 OCL 표현식에서 집계를 위해 사용한 iterate() 함수가 불필요하다.

4. 변환 알고리즘

본 절에서는 MDL을 OCL로 변환하는 알고리즘을 설명한다. 4.1절에서는 알고리즘에 필요한 기본 개념을 정의하고 4.2절에서는 알고리즘을 코드와 예제를 통하여 상세히 설명한다.

4.1 기본 개념

정의 1. MC는 UML 메타모델을 구성하는 클래스들의 집합이다. 예를 들어, 그림4에서 MC={Package, PackageableElement, Classifier, Type, Property, Class, Operation}이다. name(mc_i) = mc_i.name이고 abstract(mc_i) = mc_i.isAbstract 이다.

정의 2. R은 MC 상호간의 관계들의 집합이다. $R = R_a \cup R_c \cup R_g$로 표현한다. R_a 는 연관관계 R_c 는 포함관계 R_g 는 일반화 관계를 의미한다. 예를 들어, (그림 4)에서 R_c = {<Package, Packageable Element>, <Classifier, Generalization>, <Class, Operation>, <Class, Property>}이며 R_g = {<PackageableElement, Type>, <Type, Classifier>, <Classifier, Class>} 이다.

정의 3. MDL 트리는 MDL 표현식의 트리 형태의 표현이다. MDL 표현식 m에 대한 MDL 트리는 MDLTree(m)로 정의한다. (그림 8)에서 정의된 MDL 문법의 BNF 표현에서 <*METRIC definition*>규칙과 <*SUBMETRIC definitions*> 규칙이 MDL 트리의 노드를 표현한다. MDL 트리는 <*METRIC definition*> 노드를 루트노드로 하고 <*SUBMETRIC definitions*> 노드를 자식노드로 한다.

변환 알고리즘을 설명하기 위하여 MDL 트리의 노드 n에 대하여 다음을 정의한다.

- name(n) : n의 <metric name>규칙에 해당되는 값이다.
- entity_name(n) : n의 <entity name>규칙에 해당되는 값이다.
- value_expression_tokens(n) : <*value expression*>규칙을 구문 분석하여 얻어진 피연산자와 연산자들의 시퀀스이다. Token의 각 원소 t는 상수형, 함수형, 메트릭 이름, 연산자로 구분된다. 상수형은 OCL에서 지원하는 Integer, Real형이 있으며 함수형에는 CNT(), SUM() 이 있다. 연산자의 종류에는 OCL이 정수 및 실수형 타입에 대하여 지원하는 사칙연산이 있다.
- ocl_expression(n) : n의 <*value expression*>규칙이 OCL 표현식으로 변환된 값이다.

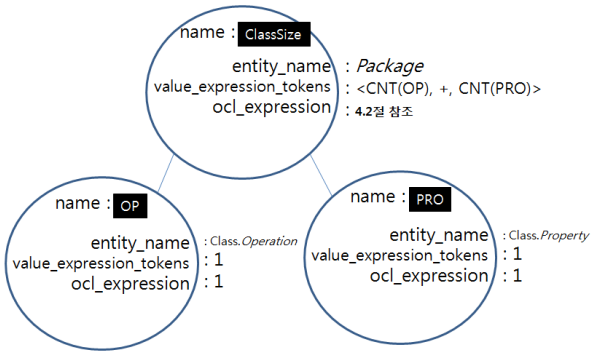
(그림 14)는 ClassSize 메트릭을 MDL로 기술한 예이다. ClassSize 메트릭은 패키지 내에존재하는 메소드와 프로퍼티의 총 개수이다. ClassSize는 하나의 METRIC과 2개의 SUBMETRIC으로 구성되어 있다.

(그림 15)는 ClassSize에 해당하는 MDL 트리를 표현한 것이다. MDL 트리에는 3개의 노드가 있다. <*METRIC definition*> 규칙은 MDL 트리에서 루트노드가 된다. 루트노드의 이름은 ClassSize이며 엔터티 이름은 Package이다. value_expression_tokens은 <CNT(OP), +, CNT(PRO)>이 된다. ocl_expression은 변환 알고리즘을 통하여 반환된 OCL 표현식이 된다.

<*SUBMETRIC definitions*>규칙은 MDL 트리의 노드가 된다. (그림 15)에서 SUBMETRIC OP와 PRO는 하위에 SUBMETRIC을 가지지 않으므로 말단노드로 표현되었다. 각 노드의 이름은 OP, PRO이며 엔터티 이름은 Class.Operation과 Class.Property이다. (그림 14)의 SUBMETRIC OP, PRO의 <*value expression*>규칙은 생략되어 있으므로 기본값을 가진다. 따라서 value_expression_tokens은 1이 된다.

METRIC ClassSize ***For*** Package ***Value*** CNT(OP) + CNT(PRO)
SUBMETRIC OP ***For*** Class.Operation
SUBMETRIC PRO ***For*** Class.Property

(그림 14) MDL을 이용하여 ClassSize 메트릭을 기술한 예



(그림 15) MDL로 기술된 ClassSize 메트릭을 MDL 트리로 표현한 예제

4.2 MDL을 OCL로 변환하는 알고리즘

본 절에서는 MDL 표현식을 OCL 표현식으로 변환하는 과정을 알고리즘 코드와 ClassSize 메트릭 예제를 이용하여 설명한다.

(그림 16)은 이전 절에서 정의한 기본개념을 이용하여 MDL 표현식을 OCL 표현식으로 변환하는 알고리즘을 보여준다. 알고리즘은 MDL 트리를 입력으로 받아서 OCL 표현식을 반환한다. 이 동작은 MDL2OCL 함수가 담당한다. 함수는 먼저 트리를 후위순회로 탐색하면서 각 노드의 <*value expression*> 규칙을 OCL 표현식으로 변환한다. 다음으로 각 노드에서 변환된 OCL 표현식을 하나의 OCL 표현식으로 병합한다. MDL2OCL 함수가 종료되면 MDL 트리의 루트노드에는 최종 변환된 OCL 표현식이 저장된다.

```
1:  OCL Expression MDL2OCL(node ) begin
2:      for each child in children(node)
3:              MDL2OCL(child)
4:      for each token  in value_tokens_sequence(node)
5:              switch(token 유형)
6:              case '+', '-', '*', '/' : //사칙연산
7:                      ocl_expression(node) += token   break;
8:              case : 상수 :
9:                      ocl_expression(node) += token; break;
10:             case : name(c in children(node)) == token : //메트릭 이름
11:                     ocl_expression(node) += ocl_expression(c);      break;
12:             case : 'CNT(', 'SUM(' : //집계 함수
13:                     ocl_expression(node) += generate_Navigation_OCL_Expression (node, c);
14:                     ocl_expression(node) +=
15:                             generate_Aggregation_Function_OCL_Expression (token); break;
16:             end
17: end
```

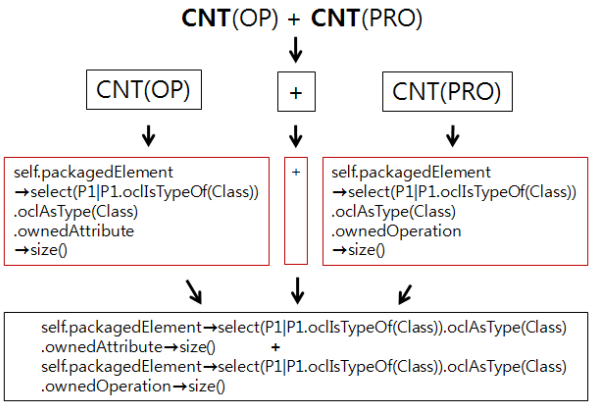
(그림 16) MDL을OCL로 변환하는 알고리즘 중 MDL2OCL 함수부분

- MDL2OCL 함수는 MDL 트리를 입력으로 받아서 OCL 표현식을 반환하는 기능을 담당한다. 먼저 재귀호출을 이용하여 트리를 후위순회 한다(2-3행). 토큰의 유형이 사칙연산, 상수인 경우에는 OCL표현식으로 직접 표현한다(6-9행). 서브메트릭 이름인 경우에는 해당 서브메트릭의 ocl_expression(n)을 사용한다(10-11행). 함수형인 경우에는 먼저 현재노드와 이 노드의 자식 노드 중 하나 사이에 OCL 탐색 표현식을 자동으로 생성하고 집계함수의 유형에 따라서 OCL 표현식을 생성한다. 이 기능들은 각각 generate_Navigation_OCL_Expression 함수와 generate_Aggregation_Function_OCL_Expression함수가 담당한다(12-16 행).

(그림 17)은 ClassSize 메트릭의 MDL 트리에서 루트노드의 토큰 유형에 따라서 OCL 표현식로 변환되는 예를 보여준다. 연산자 "+"는 MDL에서 표현된 것처럼 그대로 OCL 표현식에서도 "+"로 변환된다. 함수형에 해당하는 CNT(OP), CNT(PRO)는 부모 노드의 entity인 Pacakge와 자식노드의 entity인 Operation, Property의 탐색할 수 있는 OCL 표현식과 집계함수 표현식으로 변환된다.

4.2.1 OCL 탐색 표현식의 자동 생성

(그림 18)은 토큰의 유형이 함수형일 경우에 두 개의노드 - 현재노드와 이 노드의 자식 노드 중 하나를 입력으로 받아서 이들 사이의 OCL 탐색 표현식을 자동으로 생성시켜주는 알고리즘이다.



(그림 17) 토큰의 유형에 따른 OCL 표현식 변환 예제

generate_Navigation_OCL_Expression 함수는 인자로 주어진 시작노드로부터 목적노드까지 탐색을 위한 OCL 표현식을 자동으로 생성한다. 이 함수는 먼저 UML 메타모델에서 부모노드와 자식노드의 최단 경로를 찾는다(2-3행). findPath_in_UMLModel 함수는 탐색된 경로의 유형을 다음과 같이 분류한다.

1) **유일한 경로가 존재하는 경우 :** 이 경우는 시작노드와 목적노드 사이의 최단 경로가 유일함을 의미한다. 최단 경로를 찾는 이유는 OCL 탐색 표현식을 최대한 단순화 하여 생성시키기 위해서이다. 예를 들어 Package와 Class 사이의 탐색 경로가 유일하다.

```
01:  OCL Expression generate_Navigation_OCL_Expression ( node, child ) begin
02:      let PATH is a sequence of path, which is represented by Stack, from node to child
03:      PATH = findPath_in_UMLModel( node , child );

04:      if PATH is not empty then // 1) 유일한 경로가 존재하는 경우
05:              from_vertex, to_vertex ∈ MC, r ∈ R, from_vertex = PATH.pop();
06:              while( PATH is not empty )
07:                      if to_vertex is not null then from_vertext = to_vertex end if
08:                      to_vertex = PATH.pop(), r = findMC( to_vertex )
09:                      switch ( r )
10:                      case : r is association ∨ r is composition //연관, 포함관계
11:                              OCLString += "." + r.rolename; break;
12:                      case : r is generalization  //일반화 관계
13:                              OCLString += generate_ generalization_OCL(PATH); break;
14:                      end
15:              end while
16:      else    // 2) 경로가 존재하지 않는 경우
17:              OCLString+= to_vertex.name + ".allInstances()→select(" + variable + "|" +
                         "self.name→includes(" + variable + ".name ) ).oclAsType(" +
                         to_vertex.name + ")"
18:      end if
19:      return OCLString
20:  end
```

(그림 18) OCL의 탐색 표현식을 자동으로 생성하는 알고리즘

MDL 표현식
METRIC NOC **FOR** *Package* ... **SUBMETRIC** NC **FOR** *Class* ...
OCL 표현식
self.packagedElement → **select**(P1|P1.oclIsTypeOf(*Class*)).oclAsType(*Class*)

2) 경로가 존재하지 않는 경우 : 이 경우는 UML 메타모델에서 시작노드에서 목적노드까지 경로가 없는 경우로 정의한다. 연결되어 있지않다는 의미이다. 따라서 OCL 탐색식으로 탐색을 할 수가 없다. 예를 들면 Class와 Actor가UML 메타모델에서탐색 경로가 존재하지 않는다. 경로가 없을 경우에는 OCL의 AllInstances()를 이용하여 두 노드간의 OCL 탐색 표현식을 생성한다.

MDL 표현식
METRIC NOC **FOR** *Class* ... **SUBMETRIC** NC **FOR** *Actor* ...
OCL 표현식
Actor.**allInstances()** → **select**(A1|**self.**name → includes(A1.name)).oclAsType(*Actor*)

3) 복수개의 경로가 존재하는 경우 : 이 경우는 두 경로 사이에 동일한 길이의 최단 탐색경로가 존재한다는 의미이다. 복수개의 경로가 존재하는 경우에는 탐색 경로가 모호하여 OCL 표현식을 자동으로 생성시킬 수 없다 따라서 MDL에서는 이 경우에 오류를 발생시킨다. 예를 들어, Package 엔터티와 Operation 엔터티의 탐색 경로를 찾는 경우에 Artifact 엔티티의 Operation과 Class의 Operation의 최단 탐색 경로가 같다. 이와 같은 경우에 메트릭 기술자는 Operation의 상위 엔터티를 명확하게 기술해야 한다. MDL에서는 "." 연산자를 사용하여 Class.Operation과 같이 엔터티의 범위를 명시적으로 기술할 수 있다.

MDL 표현식
METRIC NOC **FOR** *Package* ... **SUBMETRIC** NC **FOR** *Operation* ...
OCL 표현식
self.packagedElement → **select**(P1|P1.oclIsTypeOf(*Class*)). oclAsType(*Class*).ownedOperation

경로가 적절한 경우에는 경로사이의 관계유형에 따라서 생성되는 OCL 표현식이 달라진다. 연관, 집합관계인 경우에는 OCL의 "." 표현식을 생성한다(10-11행). 일반화 관계인 경우에는 generate_ generalization_OCL 함수를 사용하여 일반화 탐색 표현식을 생성한다(12-13행). 경로가 없는 경우에는 OCL의 AllInstances() 함수를 사용하여 OCL 탐색 표현식을 생성한다(17행).

4.2.2 집계함수 표현식 생성

(그림 19)는 집계함수의 유형에 따라서 OCL 표현식을 생성하는 알고리즘이다.

generate_Aggregation_Function_OCL_Expression 함수는 집계함수의 유형에 따라서OCL 표현식 을 생성한다(1행). 먼저 집계하고자 하는 개체집합의 전체를 대상으로 집계를 하기위하여 OCL의 iterator 함수를 생성한다(3행). 다음으로 집계함수 유형이 CNT()인 경우는 1을 SUM() 함수인 경우는 자식노드의 value_expression상수 값을 더하는 OCL 표현식을 생성한다(4 행).

5. 사례 연구

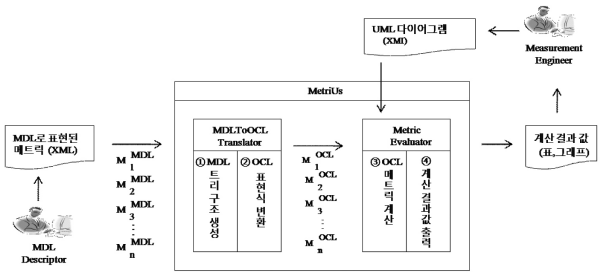
본 절에서는 MDL로 메트릭을 표현하고 UML 산출물을 대상으로 계산하는 자동화 도구 MetriUs를 소개한다. MetriUs는 MDL로 표현된 메트릭을 OCL 표현식으로 변환할 수 있고 UML 산출물을 대상으로 메트릭 값을 계산하여 측정결과를 제공한다. 5.1절에서는 MetriUs의 아키텍처를 살펴보고 5.2절에서는 적용 사례 연구를 통해 MDL과 OCL 메트릭 표현식의 크기를 비교 한다.

5.1 지원 도구

(그림 20)은 MetriUs의 아키텍처이다. MetriUs는 입력으로 MDL로 표현된 메트릭과 측정 대상인 UML 다이어그램이 필요하다. MDL로 표현된 메트릭은 XML 형식으로 작성 되었고 UML 산출물은 XMI 파일 형식으로 변환 되었다. MetriUs의 모듈은 크게 MDLToOCLTranslator와 MetricEvaluator로

```
01: String generate_Aggregation_Function_OCL_Expression (token)
02: begin
03: return "->iterate( " + ITERATOR + "; " + 변수명 + ":" + TYPE
    + "=0 | "
04:         + 변수명 + ( token is 'CNT(' ? " + 1 " : VALUE  ) + ")";
05: end
```

(그림 19) 집계함수의 유형에 따라서 OCL 표현식을 생성하는 알고리즘



(그림 20) MetriUs의 아키텍처

나눌 수 있다. MetriUs의 기능은 MDL 트리 구조 생성, OCL 표현식 변환, OCL 메트릭 계산, 그리고 계산 결과값 출력이 있다. 다음은 MetriUs에 의해 MDL로 표현된 메트릭을 OCL로 표현된 메트릭으로 변환한 후에 계산하는 과정이다.

① MDL 트리 구조 생성: MDL로 표현된 메트릭을 입력 받아서 JDOM으로 구문 분석하고 결과를 트리형태로 생성한다.
② OCL 표현식 변환: ① 과정에 의해생성된 MDL 트리 구조는 OCL 표현식으로 변환된다. OCL을 변환 하는 과정에서 UML 2.0을 지원하는 메타 모델을 이용한다. 또한 변환된 OCL 표현식은 OCL 메트릭 저장소에 저장된다.
③ OCL 메트릭 계산: ② 과정에 의해 저장된 OCL 표현식과 사용자가 등록한 UML 산출물을 입력 받아 UML 산출물의 엔터티 단위로 메트릭을 계산한다. OCL로 기술된 메트릭을 대상 엔터티에 계산하기 위해서 Eclipse OCL2 라이브러리를 이용한다. 또한 계산된 메트릭 결과값은 메트릭 계산 결과 저장소에 저장된다.
④ 계산 결과값 출력: 메트릭 계산 결과 값은 가공되어 측정 엔지니어에게 표 혹은 그래프 형태로 제공된다.

5.2 적용 사례

본 절에서는 객체지향 디자인 메트릭을 OCL과 MDL로 기술하고 두 언어의 표현식을 비교하여

MDL의 표현식이 OCL 표현식보다 간결함을 분석한다. MDL과 OCL 표현식의 비교는 표현식의 크기를 기준으로 하고 기존의 연구에서 소개되는 UML 기반 소프트웨어 디자인 메트릭을 대상으로 한다.

현재 기존의 여러 연구를 통해 소프트웨어 디자인 메트릭이 소개되고 있다. 적용 사례에 사용된 메트릭은 이미 제안된 메트릭에 대해서 조사한 기존의 연구를 중심으로 선정하였다[24-28]. 메트릭의 선정대상은 UML 기반 객체지향 디자인 메트릭으로 하며 메트릭의 유형에 따라 <표 1>과 같다.

MES = NNC + WNCO

(그림 21) 메트릭 표현식의 크기

해당 메트릭은 적용 가능한 다이어그램과 측정 유형을 기준으로 분류 하였다. 분류된 메트릭은 클래스 다이어그램에서 크기와 복잡도를 계산하는 메트릭이 대표적이다. 그리고 제시된 메트릭은 각각 OCL과 MDL로 표현 되었다.

기존의 연구에서는 메트릭 표현식은 크기가 작으면 가독성이 좋고 수정이 용이함을 설명한다[29]. 그리고 OCL 표현식의 크기를 측정하기 위해서 UML 메타 클래스간의 연관관계와 집합 함수를 이용한 메트릭을 제안하고 있다[15, 29]. 그러므로 OCL표현식의 크기 측정을 위해 제안된 메트릭을 사용하여 OCL 표현식을 계산하고 MDL 표현식과 비교 할 수 있다. (그림 21)은 메트릭 표현식의 크기를 계산하기 위해 NNC(Number of Navigated Classes) 메트릭과 WNCO (Weighted Number of Collection Operations) 메트릭이 정의되어 있다[29]. 메트릭 표현식의 크기(MES)는 NNC와 WNCO 메트릭 계산 결과값의 합으로 정의한다. MES의 값이 크다는 것은 해당 메트릭 표현식이 복잡하고 이해하기 어렵다는 것을 의미한다.

- NNC는 메트릭 표현식에서 정의된UML 메타 클래스의 수를 구하는 메트릭이다. OCL 표현식에서는 UML 메타클래스의 이름과 메타 클래스의 role name이 해당 된다. 그리고 MDL 표현식에서는 MDL문법에서 정의한 SUBMETRIC의 “<entitiy name>”이 UML 메타 클래스에 해당 된다.
- WNCO는 메트릭 표현식에서 사용된 집합 연산자가 선언된 레벨 정도에 따라서 가중치를 주고 총합하는 메트릭이다. 집합 연산자는 OCL에 표준으로 정의되어 있고 OCL 표현식에서는 select(), size(), sum(), 그리고 iterate() 등이 집합 연산자에 해당된다[9, 11]. 그리고 MDL 표현식에서는 집계함수인 SUM(), CNT()가

〈표 1〉 사례 연구로 활용된 객체지향 디자인 메트릭 유형

순번	메트릭 이름	다이어그램	유형	참조	순번	메트릭 이름	다이어그램	유형	참조
1	WMC	클래스	복잡도	[24-27]	11	Nassoc	클래스	커플링	[26, 27]
2	DAM	클래스	캡슐화	[25]	12	Ngen	클래스	상속	[26, 27]
3	PM	클래스	크기	[24, 26]	13	Ndep	클래스	커플링	[27]
4	NM	클래스	크기	[24-27]	14	AR	클래스	캡슐화	[28]
5	NV	클래스	크기	[24, 26, 27]	15	NCU	유스케이스	복잡도	[26]
6	NPV	클래스	크기	[24-26]	16	Na	유스케이스	복잡도	[26]
7	M1	클래스	복잡도	[24, 26, 27]	17	NS	상태도	크기	[26]
8	MHF	클래스	캡슐화	[24-26]	18	NT	상태도	복잡도	[26]
9	AHF	클래스	캡슐화	[24, 26, 28]	19	NEntryA	상태도	크기	[26]
10	Nagg	클래스	커플링	[26, 27]	20	NExitA	상태도	크기	[26]

```
1:  context Class inv NPRIOP:
2:  self.ownedOperation->
3:    select( visibility = VisibilityKind::public)->
4:      size() +
5:  self.ownedAttribute->
6:    select( visibility = VisibilityKind::public)->
7:      size()
```

(그림 22) NPRIOP 메트릭의 OCL 표현식

OCL의 집합 연산자에 해당 된다. 레벨은 메트릭 표현식에서 정의된 특정 메타 클래스가 다른 메타 클래스로 접근하기 위해 단계적으로 탐색하는 경우 가중치를 1씩 더한다.

(그림 22)는 MES로 OCL 메트릭 표현식을 계산하기 위해 사용할 예제이다. NPRIOP 는 한 클래스에 있는 private 멤버함수와 멤버변수의 총 개수를 구하는 메트릭이다.

- NNC 메트릭을 계산하기 위해서는 Class로 부터 탐색된 owenedOperation, ownedAttribute의정의된 수를 셈하면 된다. 각각 UML 메타 클래스의 Operation, Attribute를 의미하므로 결과값이 2이다(2, 5 행).
- WNCO 메트릭을 계산하면 select()가 1레벨에서 두 번 선언 되었다(3,6 행). 따라서 결과값은 2이다. 그리고 size()가 2레벨에서 두 번 선언 되었다(4, 7행). 따라서 가중치는 2 이며 결과값은 4이다. WNCO 메트릭의 결과값은 모든 집합 함수 결과값의 합이므로 6이다.
- MES 계산 결과값은 NNC와 WNCO의 합이므로 2와 6를 더한 8이다.

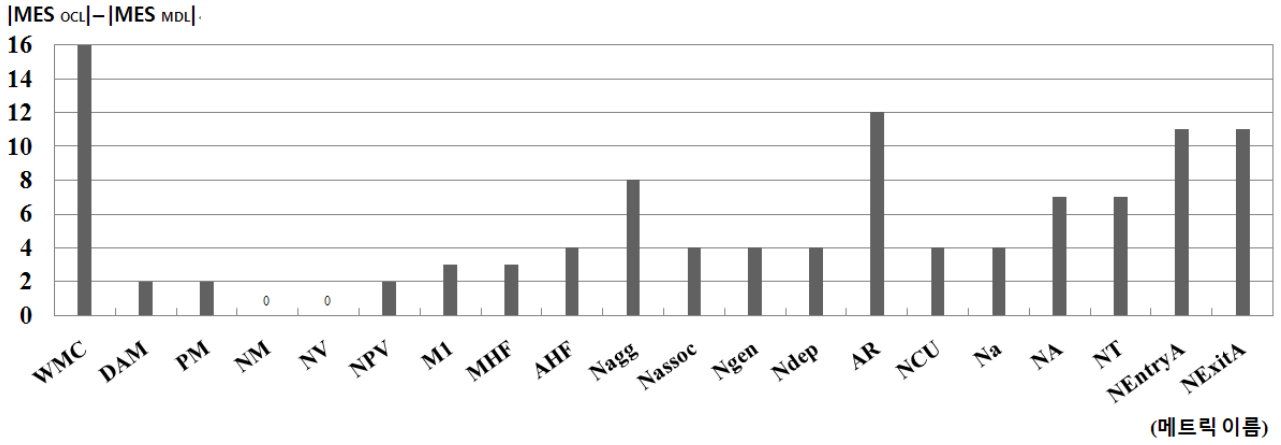
(그림 23)은 OCL과 MDL 메트릭 표현식을 MES로 계산한 결과값의 차이다. X축은 메트릭의 이름이고 Y축은 OCL 표현식의 크기에서 MDL 표현식의 크기를 뺄셈한 값이다. 그러므로 Y축의 값이높은 것은 MDL 표현식이 OCL 표현식보다 간결하다는 의미이다. 모든 OCL 표현식에서 MES의 계산 값이 MDL 표현식보다 크다.

MDL로 표현된 메트릭이 OCL 표현식 보다 간결하다는 것은 NM(Number of Methods)과 NV(Number of Variable) 메트릭을 제외한 모든 메트릭에서 확인 하였다. 특히 WMC 메트릭과 AR 메트릭의 경우 다른 메트릭보다 큰 차이를 보인다. 그 이유는 OCL 표현식이 특정 엔터티에서 다른 엔터티로 접근하기 위해서 복잡한 탐색 표현식과 상대적으로 많은 집계 함수를 사용한 것으로 분석된다. 이는 MDL로 메트릭을 표현하면 자동으로 엔터티 간의 탐색 표현식을 생성하고 MDL에 정의되어 있는 집계 함수를 사용하여 누적 값을 계산하기 때문에 OCL 표현식보다 집합 함수의 사용이 적게 되었음을 의미한다.

예를 들어 (그림 11)의 AR 메트릭인 경우 OCL과 MDL 표현식에서 MES를 계산하면 결과값은 각각 19와 7이 나왔다. MES는 WNCO와 NNC메트릭의 계산 결과값의 합이며 아래의 같이 계산 되었다.

- WNCO 메트릭인 경우에는 OCL과 MDL의 계산 결과값은 15와 4이다. 이는 OCL 표현식에서 select()와 size() 함수의 정의가 9번 되어 있고 집합 함수간에 중첩되어 표현되었기 때문에 계산 결과 값이 높은 것으로 분석된다. 반면에 MDL 표현식의 경우 CNT() 집계 함수가 4번 사용되었고 중첩이 되지 않기 때문에 표현식이 OCL보다 간결하다고 분석 된다.
- NNC 메트릭인 경우에는 OCL과 MDL의 계산 결과값은 4와 3이다. OCL 메트릭 표현식에서는 Package, PackageableElement, Class 그리고 Interface 가 UML 메타 클래스로 사용되었다. 하지만 MDL 메트릭 표현식에서는 Package, Class, Interface 만이 UML 메타 클래스로 사용되었기 때문에 OCL보다 MDL 표현식이 간결한 것으로 분석된다.

반면에 NM과 NV메트릭 표현식의 크기는 OCL과 MDL에서 차이가 없었다. 이는 NM과 NV메트릭 정의 자체가 간단해서 연관된 UML 메타 클래스의 수와 사용된 집합 함수의 개수가 MDL과 OCL 표현식에서 각각 1이기 때문으로 분석된다.



(그림 23) OCL과 MDL 메트릭 표현식의 크기 비교: |MES OCL| - |MES MDL|

6. 관련 연구

본 절에서는 기존의 메트릭 기술 언어에 대한 연구를 소개한다. 다음으로 메트릭을 사용하여 UML 산출물을 계산하는 측정 도구에 대해서 소개한다.

6.1 메트릭 기술 언어

메트릭 기술 언어에 대한 연구는 크게 두 가지로 분류된다. 기존의 언어를 활용하여 메트릭을 기술하는 연구와 새롭게 메트릭 기술언어를 정의하는 연구가 있다. 우선 기존의 언어 중 SQL, XQuery, 그리고 OCL을 바탕으로 메트릭을 기술하는 연구를 소개한다.

SQL을 활용한 연구에서는 C++과 Java와 같은 객체지향 프로그래밍 언어를 분석하여 객체지향 프로그래밍 언어의 메타모델을 데이터베이스 스키마로 표현 했다[30]. 표현된 메타모델으로 메트릭을 계산하기 위하여 복잡도에 관한메트릭 11개를 표현했고 이때 SQL을 적용했다. SQL 형식으로 표현된 메트릭은 메트릭 측정 도구(EMBER)를 사용하여 계산 할 수 있다. 하지만 SQL로 메트릭을 활용할 경우 데이터베이스에 소스 코드기반 산출물의 메타 모델에 대한 정보가 있어야 하며 SQL이 다양한 버전을 가진다는 한계가 있다.

XQuery를 활용한 연구에서는 XMI 형식인 UML 산출물에 대해서 메트릭을 적용하는 언어로 XQuery를 사용했다[31]. XQuery는 XML로 표현된 쿼리 언어이고 W3C의 표준이다. 하지만 XQuery로 표현된 메트릭 내용은 이해하기가 어렵다. 또한 UML 다이어그램 중 클래스 다이어그램에 대해서 메트릭 측정을 하며 행위 모델에 대한 언급이 부족하다.

마지막으로 OCL을 활용한 연구가 있다. OCL은 UML 모델에 관해 구조적 제약사항을 기술하기 위한 언어로서 UML의 표준으로 되어 있다[9]. 하지만 메트릭 기술 언어로서 OCL을 활용하는 연구도 많이 이루어 지고있으며 그 중에서 MaQuillan과 Baroni는OCL을 사용하여 UML 모델에 다양한 메트릭을 기술하였다[32-35]. 하지만 메트릭 기술 엔지니어가 OCL을 사용하여 메트릭을 표현하려면 UML 메타모델 엔터티에 대한 수준 높은 이해가 필요하며 메트릭을 읽고 이해하기 복잡하다는 단점을 가지고 있다. 또한 [29]의 연구에서는 OCL 표현식 에서 엔터티 간의 연관관계 많거나 집계함수가 사용될수록 메트릭 표현식은 이해하거나 수정하기가 어렵다고 한다.

위의 세가지 연구는 메타 모델 엔터티의 탐색과 표현이 복잡하여 메트릭 표현식을 생성하는데 어려운 점이 있다. 하지만 MDL은 연관된 엔터티의 탐색 표현식을 자동으로 생성하기 때문에 메트릭 기술 엔지니어는 최소한의 UML 메타모델의 정보만 필요하다. 그리고 하위 메트릭과 적은집계 함수 사용으로 인해 메트릭 표현식은 다른 언어보다 간결하며 가독성과 수정이 용이하다.

다음으로 새로운 메트릭 기술 언어를 제안한 연구가 있다. Alikacem가 제시한 새로운 메트릭 기술 언어는 Primitives, Operators, Operations, 그리고 Iterator를 사용하여 UML 대상 메트릭을 표현한다[36]. Primitives는 함수인 classes( ), methods( ), parents( )를 기본적으로 정의하고 있으며 forAll( )과 같은 Iterator를 사용하여 객체지향 메트릭을 표현하고 있다. 하지만 현재 소스 코드 기반의 메트릭 적용 프레임웍을 설계했으며 디자인 단계의 UML 다이어그램에 대해서 메트릭을 적용하는 것에 대한고려가 없다. 또한 자신들의 언어를 제시하고 자신들의 언어를 사용하는 도구를 구현했지만 도구가 공개되어 있지 않다.

또 다른 메트릭 기술 언어에는 SDMetrics에서 사용하는 메트릭 기술 언어가 있다[37]. SDMetrics가 제안한 언어는 UML 산출물을 측정 대상으로 하며XML 형식으로 작성되어 있다. 해당 언어가 메트릭을 표현하기 위해서는 XML의 각 태그가 필요하며 해당 태그는 디자인 규칙에 의해정의되어 있다. SDMetrics가 제시한 언어는 UML 메타 모델의 재구성하여 기존의 객체지향 디자인 메트릭이 UML 산출물에 적용되도록 지원한다. 하지만 UML 메타 모델을 재구성 했기때문에 모든 메트릭의 유형에 대해서 정의가 가능한지 검증이 필요하다.

마지막으로 메트릭을 수학적인 형식 언어로 표현한 연구가 있다. Object-Oriented Design Model(ODEM) 은 메트릭을 수학적인 기반으로 정의하기 위한 모델이다[10]. 해당 모델은 UML 메타모델을 기반으로 하는 객체지향 설계 모델이다. ODEM은 객체지향 디자인 메트릭을 표현하기 위해 21개의 정형 인식자(Identifier)를 이용한다. 비록 형식언어를 사용하여 메트릭을 표현하면 측정 대상에 대해서 명확히 규정 할 수 있지만 메트릭 기술 엔지니어에게 수준높은 수학적 기술 능력을 요구한다. 또한 해당 언어를 지원하는 도구나 라이브러리는 공개 되어있지 않다.

6.2 UML 산출물을 대상으로 하는 측정 도구

현재 UML 모델을 편집하는 도구들은 다양하게 있다. <표 2>는 UML 편집 도구 중에서 메트릭 적용이 가능한 도구들을 보여준다.

- 적용 대상: UML 산출물의 메트릭 적용에 대한 확장성을 비교 하는 요소로서 2개 이상의 다른 UML 다이어그램에 메트릭을 계산 할 수 있으면 UML 사용으로 작성하였고 하나의 다이어그램에 대해서만 메트릭 계산이 가능하면 해당 다이어그램 명을 명시하였다.
- MI 파일 입력 유무: 해당 UML 측정 도구는 XMI 파일 형식으로 변환된 UML 산출물을 메트릭으로 측정할 수 있는지 나타낸다. XMI 파일은 UML 모델을 기술하기 위한표준 노테이션 중에 하나이다[16]. 따라서 XMI 파일을 입력 받아서 메트릭을 측정 할 수 있는지는 측정 도구의 중요한 평가요소라고 볼 수 있다.
- 사용자 정의 메트릭: UML 측정 도구에서 메트릭의 확장성을 비교 하는 요소로서 사용자가 정의한 새로운 메트릭이 추가 가능한지 나타낸다.
- CL 지원 여부: UML 측정 도구에서 새로운 메트릭을

〈표 2〉 UML 대상 측정 도구 비교

UML 측정 도구	적용 대상	XMI파일 입력 유무	사용자 정의 메트릭	OCL 지원 여부	새로운 언어 사용 유무
Fast&Serius(Rose)/UMP(Rose)	UML 사용	X	X	-	X
Objecteering	UML 사용	X	X	-	X
Together Architect	UML 사용	O	O	O	X
MOVA	클래스 다이어그램	X	O	O	X
SDMetrics	UML 사용	O	O	X	O
MetriUs	UML 사용	O	O	O	O

추가하도록 지원한다면 메트릭 기술언어로 OCL을 사용하는지 나타낸다. 메트릭 기술 언어 중에서 OCL은 현재 많은 연구가 진행되고 있으며 다양한 메트릭을 제공한다. 비록 OCL로 메트릭을 기술할 때 UML 메타 모델에 대한 수준 높은 이해가 필요하지만 기존의 OCL 기술 엔지니어를 위해 OCL 로 메트릭을 정의 가능하게 기능은 제공되어야 한다.

- **새로운 언어 사용 유무**: UML 측정 도구에서 새로운 메트릭 기술 언어를 정의했는지 유무이다.

UML 측정 도구인 Fast&Serius[38]와 UMP(UML Metrics Producer)[39]는 Rose의 모델을 지원하는 메트릭 적용 모델 중 하나이다. 하지만 위의 두 가지 메트릭 적용모델은 Rose로 생성된 다이어그램에 대해서만 적용이 가능하고 새로운 메트릭을 정의 할 수 없다. UML 측정 도구인 Objecteering[40]도 마찬가지로 Objecteering으로 생성한 UML 모델에 대해서 메트릭을 적용하며 새로운 메트릭을 추가하는 기능은 없다.

Borland사의 Together Architect[18]는 UML 산출물을 XMI 파일 형식으로 입력 받거나 출력이 가능하며 UML 산출물을 메트릭으로 측정하기 위해 OCL을 사용하고 새로운 메트릭의 추가도 가능하다. 하지만 Together Architect는 메트릭 기술 언어로서 OCL만을 사용하기 때문에 메트릭을 새로 추가할 사용자에게 UML 메타 모델의 높은이해를 요구한다.

MOVA[41]는 UML 모델 중 클래스 다이어그램과 객체 다이어그램을 편집하며 OCL로 표현된 메트릭을 사용하여 측정하는 도구이다. 하지만 측정대상은 편집한 클래스 다이어그램만 가능하며 XMI 형식의 입출력을 지원하지 않는다.

SDMetrics[37]는OCL을 사용하지 않고 새로운 메트릭 기술 언어를 사용하여 UML 모델을 측정하는 도구이다. OCL을 사용하여 메트릭을 기술하는 엔지니어는 새로운 메트릭 기술 언어를 배워야지만 도구를 사용 할 수 있다. 또한 SDMetrics는 표준 UML 메타 모델에 대한 관계를 재구성 했다.

7. 결론 및 향후 연구

본 논문에서는 OCL을 이용하여 메트릭을 기술시의 복잡도를 극복하기 위하여 메트릭 기술 언어인 MDL을 제안하고 이것을 지원하는 도구를 개발하였다. MDL은 기존 메트릭 기술 언어와 비교하여 몇 가지 장점이 있다.

첫째로, UML 메타모델이 복잡하기 때문에 발생하는 메트릭 기술의 어려움을 극복하기 위하여 메타 클래스간의 관계를 자동으로 찾을 수 있도록 하였다. 또한, OCL 언어에서 제공하지 않는 집계함수를 제공하여 메트릭 표현을 단순화 하였다. 둘째로, 표준 UML 메타모델을 지원하므로 UML의 모든 산출물에 대하여 메트릭을 기술 할 수 있도록 하여 언어의 표현력을 높였다.

앞으로 MDL의 표현 능력을 개선하기 위한연구가 필요하다. 특히 객체지향 메트릭 중에서 상속과 관련된 메트릭을 기술하기 위해서는 순환구조의 표현이 필요하다. 대부분의 프로그램 언어에서는 함수를 이용하여 순환구조를 표현하고 있으므로 이와 유사한 표현법을 지원해야 한다. 그리고 웹을 기반으로 지원도구를 개발함으로써 도구사용의 접근성이 매우 높으나 프로그램의 사용환경을 생각해 보았을 경우에 기업의 내부 설계 산출물이 외부로 공개되어야 하는 문제점이 있다. 이를 위하여 지원도구는 패키지 형태의 소프트웨어로 지원될 필요성이 있다.

참 고 문 헌

[1] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software engineering metrics and models*: Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA, 1986.

[2] T. McCabe, "A software complexity measure," *IEEE Trans. Software Engineering,* Vol.2, No.12, pp.308-320, 1976.

[3] S. R. Chidamber, C. F. Kemerer, and C. Mit, "A metrics suite for object oriented design," *Software Engineering, IEEE Transactions on,* Vol.20, No.6, pp.476-493, 1994.

[4] G. Marcela, M. Esperanza, V. Aaron *et al.*, "Building measure-based prediction models for UML class diagram maintainability," *Empirical Software Engineering,* Vol.12, No.5, pp.517-549, 2007.

[5] W. Li, and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software,* Vol.23, No.2, pp.111-122, 1993.

[6] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*: Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1995.

[7] L. C. Briand, J. W. Daly, and J. Wüst, "A unified framework for cohesion measurement in object-oriented systems," *Empirical*

Software Engineering, Vol.3, No.1, pp.65-117, 1998.

[8] L. C. Briand, J. W. Daly, and J. K. Wüst, "A unified framework for coupling measurement in object-oriented systems," *IEEE Transactions on Software Engineering,* Vol.25, No.1, pp.91-121, 1999.

[9] O. M. G. UML, "2.0 OCL specification," *OMG Adopted Specification (ptc/03-10-14)*, 2003.

[10] R. Reißing, "Towards a model for object-oriented design measurement," *Proceedings of the 5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (Budapest, June 2001)*, pp.71-84, 2001.

[11] J. Warmer, and A. Kleppe, *The object constraint language: precise modeling with UML*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1998.

[12] L. Ol'khovich, and D. V. Koznov, "OCL-based automated validation method for UML specifications," *Programming and Computer Software,* Vol.29, No.6, pp.323-327, 2003.

[13] P. Ziemann, and M. Gogolla, "Validating OCL specifications with the use tool an example based on the BART case study," *Electronic Notes in Theoretical Computer Science,* Vol.80, pp.157-169, 2003.

[14] M. Benattou, J. M. Bruel, and N. Hameurlain, "Generating test data from OCL specification," *ECOOP2002 Workshop on Integration and Transformation of UML Models (WITUML02)*, 2002.

[15] L. Reynoso, M. Genero, and M. Piattini, "Towards a metric suite for OCL expressions expressed within UML/OCL models," *Journal of Comptuer Science and Technology,* Vol.4, No.1, pp.38-44, 2004.

[16] O. M. G. UML, "2.0 superstructure specification," *OMG ed*, 2003.

[17] M. Richters, *A precise approach to validating UML models and OCL constraints*: Logos, 2002.

[18] Borland. "Together 2007," http://www.borland.com/together/index.html.

[19] Eclipse. "Eclipse MDT 0.9.1," http://www.eclipse.org/modeling/mdt/.

[20] J. Warmer, and A. Kleppe, "Octopus: OCL tool for precise UML specifications," 2006.

[21] A. Toval, V. Requena, and J. L. Fernández, "Emerging OCL tools," *Software and Systems Modeling,* Vol.2, No.4, pp.248-261, 2003.

[22] D. Chiorean. "OCL environment 2.0.4," http://lci.cs.ubbcluj.ro/ocle/index.htm.

[23] Y. Ledru, S. Dupuy-Chessa, and H. Fadil, "Towards computer-aided design of OCL constraints," *CAiSE'04 Workshops Proceedings,* Vol.1, pp.329-338.

[24] R. Harrison, S. Counsell, and R. Nithi, "An overview of object-oriented design metrics," *Software Technology and Engineering Practice, 1997. Proceedings., Eighth IEEE International Workshop on [incorporating Computer Aided Software Engineering]*, pp.230-235, 1997.

[25] M. Xenos, D. Stavrinoudis, K. Zikouli *et al.*, "Object-oriented metrics-a survey," *Proceedings of the FESMA 2000, Federation of European Software Measurement Associations*, 2000.

[26] M. Genero, M. Piattini-Velthuis, J. A. Cruz-Lemus *et al.*, "Metrics for UML models," *UPGRADE,* Vol.V, 2004.

[27] M. G. Bocco, D. L. Moody, and M. Piattini, "Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation," *J. Softw. Maint. Evol.: Res. Pract,*Vol.17, pp.225-246, 2005.

[28] R. C. Martin, *Agile software development: principles, patterns, and practices*: Prentice Hall PTR Upper Saddle River, NJ, USA, 2003.

[29] L. Reynoso, M. Genero, M. Piattini *et al.*, "Assessing the impact of coupling on the understandability and modifiability of OCL expressions within UML/OCL combined models," *Software Metrics, 2005. 11th IEEE International Symposium*, pp.14-14, 2005.

[30] F. G. Wilkie, and T. J. Harmer, "Tool support for measuring complexity in heterogeneous object-oriented software," *Software Maintenance, 2002. Proceedings. International Conference on*, pp.152-161, 2002.

[31] M. El-Wakil, A. El-Bastawisi, M. Riad *et al.*, "A novel approach to formalize object-oriented design metrics," in 9th International Conference on Empirical Assessment in Software Engineering 2005.

[32] A. L. Baroni, S. Braz, and F. B. Abreu, "Using OCL to formalize object-oriented design metrics definitions," *ECOOP'02 Workshop on Quantitative Approaches in OO Software Engineering, Lecture Notes in Computer Science*, 2002.

[33] A. L. Baroni, and F. B. Abreu, "Formalizing object-oriented design metrics upon the UML meta-model," *Brazilian Symposium on Software Engineering, Gramado-RS, Brazil*, 2002.

[34] A. L. Baroni, and F. B. e Abreu, "An OCL-based formalization of the moose metric suite," *7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, 2003.

[35] A. L. Baroni, and F. B. e Abreu, "A formal library for aiding metrics extraction," *International Workshop on Object-Oriented Re-Engineering at ECOOP*, 2003.

[36] E. H. Alikacem, Sahroui, H., "Generic metric extraction framework," *In proceedings of the 16th International workshop on software measurement and Metrik Kongress* pp.383-390, 2006.

[37] SDMetrics. "SDMetrics v2.02," http://www.sdmetrics.com/index.html.

[38] M. Carbone, and G. Santucci, "Fast & serious: a UML based metric for effort estimation," *Proceedings of the 6 thECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'02).*

[39] H. Kim, and C. Boldyreff, "Developing software metrics

applicable to UML models," *6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, 2002.
[40] Objecteering. "Objecteering/metrics," http://www.objecteering.com.
[41] M. Clavel, and V. Torres, "MOVA: a tool for modeling, measuring and validating UML class diagrams," *Accepted in the Academic Posters and Demonstrations Session of MODELS 2007*, 2007.

부 록

순번	메트릭이름	언어	설 명	다이어그램	유형	참조
1	WMC	자연어	The weighted Methods Per Class.	클래스	복잡도	[24-27]
		OCL	context Class inv WMC : self.ownedOperation → select(isAbstract=false) → iterate(Ope1:cc:Integer=0| cc +Activity.allInstances() → select(oclIsTypeOf(Activity) and Ope1.name=name) → iterate(result:Integer=0 | result+edge → size()-node → select (oclIsKindOf(Action)) → size()+2))			
		MDL	METRIC WMC FOR Class VALUE MW SUBMETRIC MW FOR Operation[isAbstract=false] VALUE SUM(CC) SUBMETRIC CC FOR Activity[OUTER.name=name] TYPE String VALUE OCL(edge → size()-node → select(oclIsKindOf(Action)) → size()+2			
2	DAM	자연어	The ratio of the number of private attributes to the total number of attributes declared in the class	클래스	캡슐화	[25]
		OCL	context Class inv DAM : self.ownedAttribute → select(visibility=VisibilityKind::protected) → size()/self.ownedAttribute → size()			
		MDL	METRIC DAM FOR Class VALUE CNT(prip)/CNT(totalp) SUBMETRIC prip FOR Property[visibility=VisibilityKind::protected] SUBMETRIC totalp FOR Property			
3	PM	자연어	The number of public methods in a class.	클래스	크기	[24, 26]
		OCL	context Class inv NOPOC : self.ownedOperation → select(visibility=VisibilityKind::public) → size()			
		MDL	METRIC NOPOC FOR Class VALUE CNT(OP) SUBMETRIC OP FOR Operation[visibility=VisibilityKind::public]			
4	NM	자연어	The total number of methods in a class. This metric is not including inherited methods.	클래스	크기	[24-27]
		OCL	context Class inv NM : self.ownedOperation → size()			
		MDL	METRIC NM FOR Class VALUE CNT(OP) SUBMETRIC OP FOR Operation			
5	NV	자연어	The number of variables per class	클래스	크기	[24, 26, 27]
		OCL	context Class inv NV : self.ownedAttribute → size()			
		MDL	METRIC NV FOR Class VALUE CNT(PRO) SUBMETRIC PRO FOR Property			
6	NPV	자연어	The number of public variables per class	클래스	크기	[24-26]
		OCL	context Class inv NPV: self.ownedAttribute → select(visibility = VisibilityKind::public) → size()			
		MDL	METRIC NV FOR Class VALUE CNT(PRO) SUBMETRIC PRO FOR Property[visibility=VisibilityKind::public]			
7	M1	자연어	The number of classes in a class diagram	클래스	복잡도	[24, 26, 27]
		OCL	context Package inv M1 : self.packagedElement → select(P1|P1.oclIsTypeOf(Class)).oclAsType(Class) → size()			
		MDL	METRIC M1 FOR Package VALUE CNT(cl) SUBMETRIC cl FOR Class			

부 록(계속)

순번	메트릭이름	언어	설 명	다이어그램	유형	참조
8	MHF	자연어	Method Hiding Factor (MHF). This metric is the ratio of hidden (private or protected) methods to total methods. As such, MHF is proposed as a measure of encapsulation.	클래스	캡슐화	[24-26]
		OCL	context Class inv MHF : self.ownedOperation → select(visibility = VisibilityKind::protected) → size()+self.ownedOperation → select(visibility = VisibilityKind::private) → size()/self.ownedOperation → size()			
		MDL	METRIC MHF FOR Class VALUE CNT(PRO)+CNT(PRI)/CNT(TOT) SUBMETRIC PRO FOR Operation[visibility=VisibilityKind::protected] SUBMETRIC PRI FOR Operation[visibility=VisibilityKind::private] SUBMETRIC TOT FOR Operation			
9	AHF	자연어	Attribute Hiding Factor (AHF). This metric is the ratio of hidden (private or protected) attributes to total attributes. AHF is also proposed as a measure of encapsulation.	클래스	캡슐화	[24, 26, 28]
		OCL	context Class inv AHF : self.ownedAttribute → select(visibility=VisibilityKind::protected) → iterate(iter3; result4:Integer=0 | result4 + 1) + self.ownedAttribute → select(visibility=VisibilityKind::private) → iterate(iter5;result6: Integer=0 | result6 + 1)/self.ownedAttribute → iterate(iter7; result8:Integer=0 | result8 + 1)			
		MDL	METRIC AHF FOR Class VALUE CNT(PRO)+CNT(PRI)/CNT(TOT) SUBMETRIC PRO FOR Property[visibility=VisibilityKind::protected] SUBMETRIC PRI FOR Property[visibility=VisibilityKind::private] SUBMETRIC TOT FOR Property			
10	Nagg	자연어	The total number of aggregation relationships within a class diagram	클래스	커플링	[26, 27]
		OCL	context Package inv Nagg : self.packagedElement → select(P1|P1.oclIsTypeOf(Association)).oclAsType(Association) → select(memberEnd.aggregation<>Sequence{uml::AggregationKind::none,uml::AggregationKind::none}) → size()			
		MDL	METRIC Nagg FOR Package VALUE CNT(Agg) SUBMETRIC Agg FOR Association[memberEnd.aggregation<>Sequence{uml::AggregationKind::none,uml::AggregationKind::none}]			
11	Nassoc	자연어	The total number of associations.	클래스	커플링	[26, 27]
		OCL	context Package inv Nassoc : self.packagedElement → select(P1|P1.oclIsTypeOf(Association)).oclAsType(Association) → size()			
		MDL	METRIC Nassoc FOR Package VALUE CNT(Na) SUBMETRIC Na FOR Association			
12	Ngen	자연어	The total number of generalization relationships within a class diagram.	클래스	상속	[26, 27]
		OCL	context Package inv Ngen : self.packagedElement → select(P1|P1.oclIsTypeOf(Gerneralization)).oclAsType(Gerneralization) → size()			
		MDL	METRIC Ngen FOR Package VALUE CNT(Ng) SUBMETRIC Ng FOR Gerneralization			
13	Ndep	자연어	The total number of dependency relationships.	클래스	커플링	[27]
		OCL	context Package inv Ndep : self.packagedElement → select(P1|P1.oclIsTypeOf(Dependency)).oclAsType(Dependency) → size()			
		MDL	METRIC Ndep FOR Package VALUE CNT(Nd) SUBMETRIC Nd FOR Dependency			

부 록(계속)

순번	메트릭이름	언어	설 명	다이어그램	유형	참조
14	AR	자연어	Abstractness Ratio. The ratio of the number of abstract classes (and interfaces) in the analyzed package to the total number of classes in the analyzed package.	클래스	캡슐화	[28]
		OCL	context Model inv AR : self.packagedElement → select(P1|P1.oclIsTypeOf(Class)).oclAsType(Class) → select(isAbstract=true) → size()+self.packagedElement → select(P3|P3 .oclIsTypeOf(Interface)).oclAsType(Interface) → size())/(self.packagedElement → select(P4|P4.oclIsTypeOf(Class)).oclAsType(Class) → size()+self.packagedElement → select(P5|P5.oclIsTypeOf(Interface)).oclAsType(Interface) → size()			
		MDL	METRIC AR FOR Model VALUE (NOACI) / (NOCI) SUBMETRIC NOACI FOR Package VALUE CNT(NOAC)+CNT(NOI) SUBMETRIC NOAC FOR Class[isAbstract=true] SUBMETRIC NOI FOR Interface SUBMETRIC NOCI FOR Package VALUE CNT(NOC)+CNT(NOI2) SUBMETRIC NOC FOR Class SUBMETRIC NOI2 FOR Interface			
15	NCU	자연어	The number of UseCases in a Model.	유스케이스	복잡도	[26]
		OCL	context Model inv Nuse : self.packagedElement → select(P1|P1.oclIsTypeOf(UseCase)).oclAsType(UseCase) → size()			
		MDL	METRIC NCU FOR Model VALUE CNT(USE) SUBMETRIC USE FOR UseCase			
16	Na	자연어	The number of Actors in a Model.	유스케이스	복잡도	[26]
		OCL	context Model inv: self.pacakgedElement → select(p1 | p1.isOclTypeOf(Actor) = true) → size()			
		MDL	METRIC Na FOR Model VALUE CNT(USE) SUBMETRIC USE FOR Actor			
17	NA	자연어	The total number of activities in the statechart diagram	상태도	크기	[26]
		OCL	context Model inv NA : self.packagedElement → select(P1|P1.oclIsTypeOf(StateMachine)).oclAsType(StateMachine).region. subvertex → size()			
		MDL	METRIC NAFOR Model VALUE CNT(state) SUBMETRIC state FOR State			
18	NT	자연어	The total number of transitions. Considering common transitions	상태도	복잡도	[26]
		OCL	context Model inv NT : self.packagedElement → select(P1|P1.oclIsTypeOf(StateMachine)).oclAsType(StateMachine).region. transition → size()			
		MDL	METRIC NTFOR Model VALUE Tstate SUBMETRIC Tstate FOR StateVALUE CNT(tran) SUBMETRIC tran FOR Trasition			
19	NEntryA	자연어	The total number of entry actions	상태도	크기	[26]
		OCL	context Model inv NEntryA : self.packagedElement → select(P1|P1.oclIsTypeOf(StateMachine)).oclAsType(StateMachine).region. subvertex → select(EntryPoint=true) → size()			
		MDL	METRIC NEntryA FOR Model VALUE CNT(state) SUBMETRIC state FOR State[EntryPoint=true]			
20	NExitA	자연어	The total number of exit actions	상태도	크기	[26]
		OCL	context Model inv NExitA : self.packagedElement → select(P1|P1.oclIsTypeOf(StateMachine)).oclAsType(StateMachine).region .subvertex → select(V2|V2.oclIsTypeOf(FinalState)).oclAsType(FinalState) → size()			
		MDL	METRIC NExitA FOR Model VALUE CNT(Fstate) SUBMETRIC Fstate FOR FinalState			

김 태 연

e-mail : tykim@pusan.ac.kr
1998년~2001년 ㈜세안IT 병원사업부 사원
2003년~2005년 ㈜에이치원 병원사업부 대리
2007년 부산대학교 정보컴퓨터공학(학사)
2007년~현 재 부산대학교 컴퓨터공학과(석사)
관심분야 : 데이터 모델 설계, 대용량 데이터 솔루션, 객체지향 설계, 소프트웨어 테스팅, 소프트웨어 메트릭

김 윤 규

e-mail : kim_yunkyu@pusan.ac.kr
2008년 부산대학교 정보컴퓨터공학(학사)
2008년~현 재 부산대학교 컴퓨터공학과 (석사)
관심분야 : 소프트웨어 아키텍처, 객체지향 방법론, 소프트웨어 테스팅, 소프트웨어 메트릭

채 흥 석

e-mail : hschae@pusan.ac.kr
1994년 서울대 원자핵공학(학사)
1996년 한국과학기술원 전산학(석사)
2000년 한국과학기술원 전산학(박사)
2000년~2003년 ㈜동양시스템즈 기술연구소 선임연구원
2003년~2004년 한국과학기술원 전산학과 초빙교수
2004년~현 재 부산대학교 컴퓨터공학과 조교수
관심분야 : 객체지향 방법론, 소프트웨어 테스팅, 소프트웨어 메트릭, 소프트웨어 유지보수, 미들웨어 설계, 프로덕트라인 공학