모델 슬라이싱을 이용한 UML 메타 모델의 모듈화*

배정호○ 이광민 박진욱 채흥석
부산대학교 객체지향시스템연구실
{jhbae83○, leekm, jwpark, hschae}@pusan.ac.kr

Jung Ho Bae○ Lee KwangMin Jin-Wook Park Heung Seok Chae
Pusan National University

요 약

UML 모델을 사용하는 다양한 곳에서 UML 메타모델을 준수하면 많은 이점을 가진다. 하지만 UML 메타 모델 자체 규모가 점점 거대해지고 있으며 UML 메타 모델을 사용하는 다양한 곳에서도 UML 메타 모델의 일부분만을 필요로 하는 경우도 있다. 즉, UML 메타모델에서 사용하고자 하는 특정 요소들만을 추출하여 사용할 필요가 있다. UML 메타모델의 특정 부분만을 추출하여 사용함으로써 UML 모델링 도구나 분석도구를 좀더 쉽고 가볍게 개발할 수 있다. 본 논문에서는 UML 메타모델에서 특정 다이어그램 요소만을 추출하기 위한 슬라이싱 알고리즘을 제시하고 실제로 적용한 결과를 보여준다.

1. 서 론

UML 메타모델은 UML 다이어그램의 표기법과 내용을 나타내는 UML의 상위 개념 모델로 제약이 포함된 UML의 클래스 다이어그램 형식으로 표현된다[1]. 이 UML 메타모델을 준수하면 UML을 사용하는 모든 곳에서 내용에 대한 호환이 가능하다. 즉, 서로 다른 UML 모델링 도구라도 UML 메타모델을 따르면 쉽게 다이어그램의 내용을 서로 호환할 수 있다[2].

하지만 UML 메타 모델은 점점 규모가 커지고 있으며 UML 메타모델을 사용하는 조직이나 도구에서는 UML 메타모델의 일부만을 필요로 하는 경우가 있다. 예를 들어, 필요에 따라 UML 메타모델 사용자가 클래스 다이어그램과 시퀀스 다이어그램만을 필요로 할 수도 있으며, 상태기계 다이어그램이나 액티비티 다이어그램만을 원할 수도 있다. 하지만 현재의 UML 메타모델 명세는 모든 다이어그램 요소를 표현하기 위해 260여 개의 모델요소(Model Elements)들을 포함하고 있다. UML 다이어그램 중에서 특정 다이어그램의 메타모델만이 필요하고, 기존의 UML 메타 모델을 준수하는 내용과도 호환이 가능하게 하기 위해서는 기존의 UML 메타모델에서 사용자가 필요로 하는 부분만을 잘라낼 필요가 있다[3].

기존에 프로그램 소스에 기반해서 변수의 추적을 위한 슬라이싱 기술이 연구 되었고[4,5], 프로그램 슬라이싱 기술에 기반하여 모델을 슬라이싱 하는 알고리즘이 제시되었다[3]. 하지만 기존의 알고리즘은 주요 요소의 상위 클래스와 다른 슬라이스에 포함된 클래스 간의 연관 관계(association), 합성 관계(composition), 포함 관계(aggregation)을 고려하지 않는다는 문제가 있다. 상속을 받는 클래스들은 상위 클래스가 가지는 연관 관계를 모두 가지기 때문에 주요 클래스간의 연관 관계를 추가 하는 것과 마찬가지로 상위 클래스와 다른 클래스간의 연관 관계도 추가되어야 한다.

그러므로 UML 메타모델에서 특정 다이어그램 요소들만을 추출해 내기 위한 새로운 알고리즘이 필요하다. 본 논문에서는 UML의 특정 다이어그램 요소들만을 슬라이싱하는 알고리즘을 제시하고, 알고리즘을 사용하여 적용한 사례를 설명한다.

2절에서는 연구 배경을 설명하며 3절에서는 UML 메타모델 슬라이싱 알고리즘을 제시한다. 4절에서는 제시한 알고리즘을 적용한 사례를 살펴보고 5절에서 결론과 향후 연구 방향에 대해 설명한다.

2. 연구 배경

슬라이싱은 규모가 큰 대상을 원하는 크기의 적은 규모로 줄이는 방법 중 하나이다. 기본적으로 원하는 부분만을 남겨두고, 원하는 작업에 사용하지 않는 나머지 부분을 삭제하는 방법이 많이 사용된다. 그러나 필요 없는 부분을 삭제하는 것이 아니라 원하는 부분만 하이라이팅 하는 방법이 사용 되기도 한다[6,7].

슬라이싱은 규모가 큰 대상을 적은 규모로 나누어서 분석할 수 있으므로 분산 작업이 가능하게 하고, 특정 부분에 집중하여 분석할 수 있게 한다[5]. 그리고 관심이 있는 부분에만 초점을 맞추어 그 자취를 추적함으로써 분석의 능력을 향상 시키고, 프로그램의 유지 보수를 용이하게 하며 프로그램을 이해하는데 도움을 준다[8].

현재 소프트웨어 공학 분야에서 연구된 대부분의 슬라이싱 기술은 프로그램 슬라이싱[5,9] 관련 기술이다. 모델 슬라이싱 기술은 프로그램 슬라이싱 기술을 기반으로 한다. 프로그램 슬라이싱은 대상이 프로그램이고 입력이 변수 또는 특정 함수 등인 반면 모델 슬라이싱은 모델을 대상으로 하고 입력은 모델의 클래스가 된다. 이 논문에서는 UML 메타 모델을 대상으로 하여 프로그램 슬라이싱 기술을 확장한 모델 슬라이싱 기법을 소개한다.

2.1 프로그램 슬라이싱

프로그램 슬라이싱은 규모가 큰 프로그램을 특정 목적을 위해서 필요한 부분을 잘라내는 것을 말한다[10]. 프로그램에서 현재 관심있는 부분을 제외한 나머지 부분을 모두 제거하여 프로그램 디버깅, 테스트, 통합, 보안, 이해, 유지보수를 용이하게 하기 위한 목적으로 사용된다. 예를 들어 개발자가 프로그램 테스트 중 특정 변수의 값이 원하던

* 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다.

값이 아니면 디버깅을 한다. 이때 슬라이싱 기법을 사용하여 해당 변수와 관련이 있는 부분만 남겨 해당 변수의 자취를 관찰 함으로써 개발자는 에러가 발생한 부분을 보다 쉽게 파악할 수 있다. [4,5,6,8,9,10].

지역 변수뿐만 아니라 함수나 전역 변수, 객체 등을 기준으로 슬라이싱 하면 프로그램을 모듈화 하는 데에 도움을 줄 수 있다. 특정 객체를 추적함으로써 모듈 간의 응집도를 높이고 결합도를 줄이면서 스파게티 프로그램에 대한 역공학(Reverse Engineering)과정을 적용하고 모듈을 파악하는데 많은 도움을 준다.

2.2 모델 슬라이싱

모델 슬라이싱은 방대한 크기의 모델을 특정 목적을 위해서 필요한 부분을 잘라내는 것을 말한다[11]. 모델 슬라이싱은 전체 모델에서 관심 있는 요소를 제외한 나머지 부분을 삭제함으로써 모델을 이해하는데 도움을 준다. 모델 슬라이싱과 프로그램 슬라이싱은 상당히 유사한 부분이 많으므로 프로그램 슬라이싱 기술을 모델 슬라이싱에도 일부분 적용 시킬 수 있다. 모델 슬라이싱은 입력으로 주어진 전체 모델에서 슬라이스 하고자 하는 특정 요소를 선택하여 선택한 요소와 관련이 없는 부분은 모두 삭제한다.

또한 모델을 이용하여 시스템을 개발하려고 할 때 전체 모델을 사용하지 않고 슬라이싱 된 모델을 사용할 수 있다. 그러면 전체 모델에서 시스템 개발자가 원하는 부분과 그와 연관된 메타 클래스들만으로 구성된 모델을 사용할 수 있다. 즉, 시스템을 제작할 때 모델 전체를 이해하고 분석해야 하는 오버헤드를 줄일 수 있게 된다.

모델, 특히 UML 메타 모델 슬라이싱[3]은 UML 메타 모델에서 관심이 있는 메타 클래스를 기준으로 하여 슬라이싱을 한다. UML 메타 모델 슬라이싱을 통해 메타 모델의 복잡도를 낮출 수 있다. 즉, 해당 메타 클래스의 역할과 관련 메타 클래스에 대한 정보를 좀 더 쉽게 파악할 수 있다.

2.2 UML 메타 모델

그림 1은 MDD(Model Driven Development) 기술의 기초가 되는 전통적인 4-레이어 구조를 보여준다. 그림 1의 OMG(Object Management Group) 구조는 UML과 MOF(Meta-Object Facility)[12]와 같은 MDD 기술의 토대이다[13].

OMG 모델링 구조는 모델 레벨의 계층(hierarchy)으로 구성된다. 각 레벨은 상위 레벨의 인스턴스로 특성화 된다. 예를 들어 M1레벨의 인스턴스가 M0레벨이 된다. 최 하위 레벨인 M0는 사용자 데이터를 가진다. 사용자 데이터는 소프트웨어가 실제로 사용하기 위해 사용자가 입/출력하고 가공되는 실제 데이터 객체이다. 다음 레벨인 M1 레벨은 M0

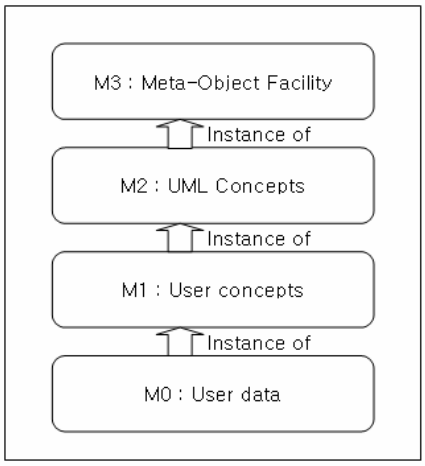

그림 1. 전통적인 Object Management Group 모델링 구조

(사용자 데이터)의 모델을 가진다. 사용자 모델이 이 레벨에서 존재한다. M2 레벨은 M1 레벨에 있는 모델 정보의 모델을 가진다. 이것은 모델의 모델이기 때문에 메타 모델이라 부른다. 마지막으로 M3 레벨은 M2에 있는 정보의 모델을 가지고 있다. 그러므로 메타-메타모델이라 불린다. 역사 적인 이유로 Meta-Object Facility라고도 부른다.

메타 모델은 M2레벨에 속한다. 메타 모델은 의미 있는 특정 모델 집합과 구조를 보여준다. 메타모델은 모델의 특정 집합이 어떤 의미를 가지는지를 설명한다. UML 메타 모델은 UML을 사용하는 모델들을 해석하는 방법을 설명한다. UML 메타 모델은 모델을 설명하기 위해 사용하는 언어이며, 모델은 메타모델의 인스턴스이다.

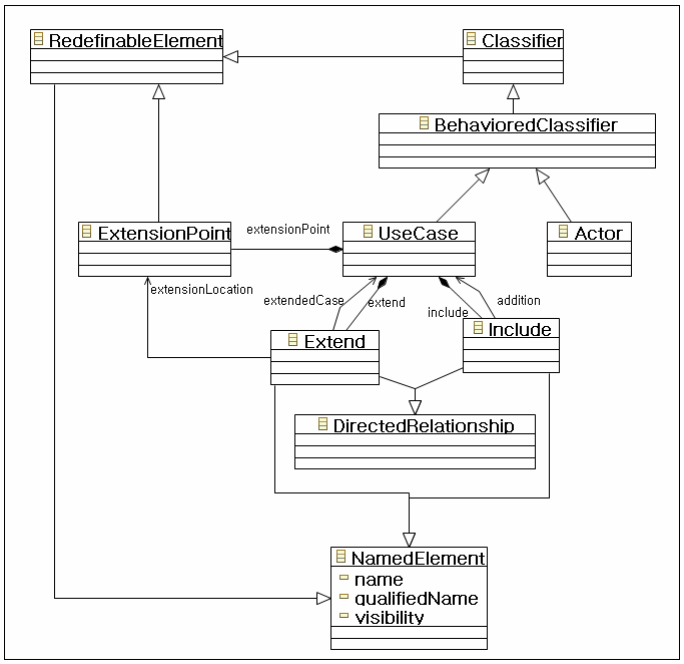

그림 2. Use Case Meta Model

UML 메타 모델은 메타 클래스와 그들간의 연관 관계로 구성된다. 메타 클래스는 속성(attribute), 연관 관계(association), 합성 관계(composition), 집합 관계(aggregation) 그리고 실현(realization)으로 구성된다. 그림 2는 UML2 명세서(Specification)에 나타나 있는 Use Case 메타 모델의 일부이다[1]. UseCase, Actor, Extend 등이 메타 클래스이고, NamedElement의 name,

qualifiedName, visibility가 속성이다. 그리고 UseCase와 ExtensionPoint는 extensionPoint 합성관계이며, Include와 UseCase는 addition 연관 관계이다. 또한 UseCase는 BehavioredClassifier의 실현이다.

3. UML 메타 모델 슬라이싱

그림 3은 UML 모델 슬라이싱 알고리즘의 입/출력을 나타낸다. UML 메타 모델 슬라이서는 UML 메타 모델과 모델의 주요 요소를 입력으로 받아서 UML 메타 모델 슬라이스를 출력한다. 입력 중 하나인 모델의 주요 요소는 UML 메타 모델의 모듈화를 위해 필요한 주요 클래스이다. 이 논문에서는 UML 메타 모델을 사용하므로 특정 모델의 메타 클래스들을 입력으로 사용한다. UseCase를 예로 들면 UseCase, Actor, Association, Extend, Include, ExtensionPoint가 주요 요소의 입력으로 들어 갈 수 있다. 출력은 입력으로 추가된 주요 요소들과 그들의 상위 메타 클래스를 포함한 슬라이스 된 메타 모델이다.

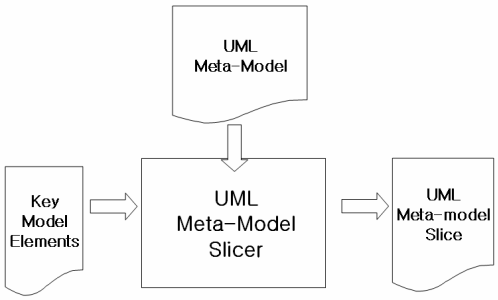

그림 3. UML 모델 슬라이서의 입력과 출력

그림 4는 UML 메타모델 슬라이싱 알고리즘을 보여준다. 알고리즘에 나타나는 add함수는 메타 모델 M_a과 클래스 e_a를 입력으로 하고 메타 모델 M_a'를 출력한다. 먼저 입력으로 받은 e_a와 M_a의 합집합을 M_a'에 추가한다. 그리고 M_a'의 각 메타 클래스 중에서 e_a와 연관 관계(합성 관계와 집합 관계 포함, 이후 연관 관계는 집합 관계와 합성관계를 모두 포함한다)를 M_a'에 추가한다. 이때는 M_a'의 임의의 원소 e_a'에서 e_a로 가는 연관/합성 관계와 e_a에서 e_a'로 가는 연관/합성 관계를 모두 추가한다/ M'에 M과 e_a를 추가하고 M_a'과 e_a와의 관계를 추가하므로 M⊆M'의 관계가 성립한다.

UML 메타 모델 슬라이싱은 다음과 같은 2단계로 구성된다. 이해를 돕기 위해 UseCase 다이어그램의 메타 모델을 예로 들어 설명을 한다.

- Step 1

Step 1에서는 입력으로 주어진 주요 요소들과 이들과 직접적인 연관 관계를 가지는 요소를 고려한다. 주요 요소 집합 E에 속하는 각 원소 e에서 대해서 먼저, 해당 원소들을 슬라이스 된 모델 집합 M'에 추가한다. 그리고 e가 가지는 관계 중 실현(realization)을 제외한 연관 관계를 집합 R로 둔다. 집합 R의 원소 r에 대해서 r의 목표 노드를 M'에 추가한다. 그리고 add함수를 통해 M'에 e를 추가하면서 M'의 원소와 e의 연관 관계를 M'에 추가한다.

예를 들어, 초기 주요 요소로 UseCase 메타 클래스를 입력으로 하면 UseCase 메타 클래스의 연관 관계는 include : Include[*], extend : Extend[*], extensionPoint : ExtensionPoint[*], subject : Classifier[*]이다. 이때 ":" 앞은 연관 관계 이름을, 뒤는 연관이 있는 메타 클래스 이름을, []는 다중성(multiplicity)을 나타낸다. UseCase와 연관 관계가 있는 Include, Extend, ExtensionPoint, Classifier를 M'에 새로 추가한다. 그리고 add 함수를 통해 메타 클래스를 집합 M' 추가하고 각각의 연관 관계를 추가한다. 이러한 과정을 초기 주요 요소들 각각에 대하여 반복한다. 이 예에서는 초기 주요 요소가 UseCase 메타 클래스 하나이므로 UseCase 메타 클래스에 대한 연관 관계만 추가하게 된다. UseCase 메타 클래스를 초기 주요 요소로 하여 Step 1을 종료하면 Classifier, Extend, Include, ExtensionPoint 메타 클래스가 슬라이스 된 모델 집합 M'에 추가된다.

- Step 2

Step 2는 첫 번째 단계에서 메타 모델 슬라이스 M'에 추가된 요소들의 상위 요소들을 추가한다. M'에 속하는 각 원소 e에 대해서 e의 상위 메타 클래스의 집합을 S라 두면 S의 원소 각각의 원소 s를 M'에 추가한다.

```
1.  function modelSliser
2.  input : M is a meta-model
3.  input : E is a set of key class elements
4.  output : M' is a model slice
5.
6.  //step 1
7.  for each e ∈ E begin
8.      add(M', e) // add e to M'
9.      let R be a set of associations including
                compositions and aggregations with e
10.     for each r ∈ R begin
11.         let R be the target class of r
12.         add(M', t) // add t to M'
13.     end for
14. end for
15.
16. //step 2
17. for each e ∈ M such that type(e) = class begin
18.     let S be a set of super classes of e
19.     for each s ∈ S begin
20.         add(M', s) // add s to M'
21.     end for
22. end for
23.
24. return M'
25. end function
26.
27. function add
28. input : Ma is a meta-model
29. input : ea is a meta-class
30. output : Ma' is a meta-model
31. begin
32. Ma' = Ma ∪ {ea}
33.
34. for each ea' ∈ Ma such that type( ea') = CLASS begin
35.     let R be a set associations including
                compositions and aggregations between ea and ea'
36.     Ma' = Ma' ∪ R
37. end for
38.
39. return Ma'
40. end function
```

그림 4. UML 모델 슬라이싱 알고리즘

Classifier를 예를 들어 설명하면 Classifier가 추가 되기 전에 Extend의 상위 클래스 중 하나인 NamedElement가

추가 되어 있다고 가정하면 Classifier의 상위 클래스는 Namespace, RedefinableElement, Type, TemplateableElement이다. 이중에 하나인 Namespace를 추가하게 되면 Namespace가 가지는 연관관계는 모두 6개 이지만 이중에서 member : NamedElement[*]와 ownedMember : NamedElement[*]만 M'에 추가된다. 나머지 4개의 연관관계는 목표 메타 클래스가 M'에 존재 하지 않는다. 그리고 Namespace가 추가 됨으로 해서 NamedElement의 namespace : Namespace [0..1]가 M'에 추가된다.

두 번째 단계가 계속 되면 M'의 크기는 점점 증가하게 된다. 그리고 새로 추가된 모든 요소들이 두 번째 단계를 거친다. M'에 속해 있는 모든 요소들의 상위요소를 추가하면 두 번째 단계는 종료한다.

그림 5는 UseCase 메타 클래스를 초기 주요 요소로 사용하여 Step 1, 2를 모두 적용한 결과를 보여준다. Association, Realization, Actor는 Use Case의 요소이지만 UseCase의 연관 관계도 아니고 Use Case의 상위 클래스도 아니다. 그리고 1단계에서 추가된 요소들의 상위요소도 아니므로 UseCase 메타 클래스만을 입력으로 한 예의 결과에는 속하지 않는다.

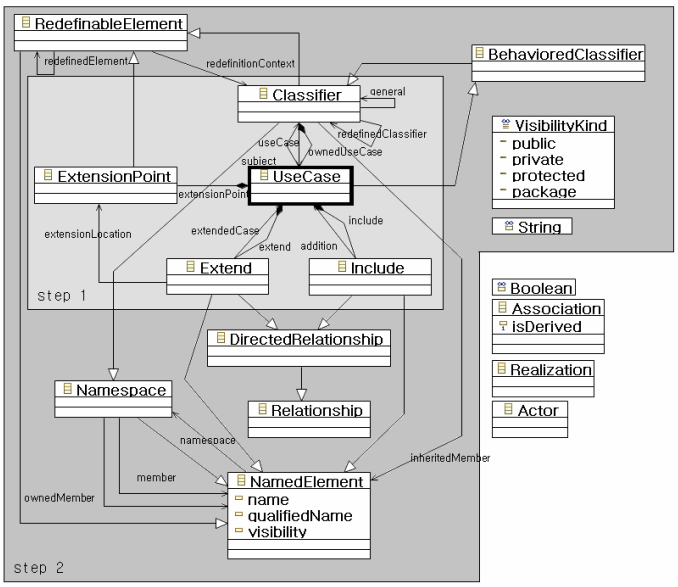

그림 5. Sliced Model with UseCase Meta Class

4. 적용 사례

이 절에서는 3절에서 제시한 모델 슬라이싱 알고리즘을 UML 메타모델에 직접 적용한 결과를 보여준다.

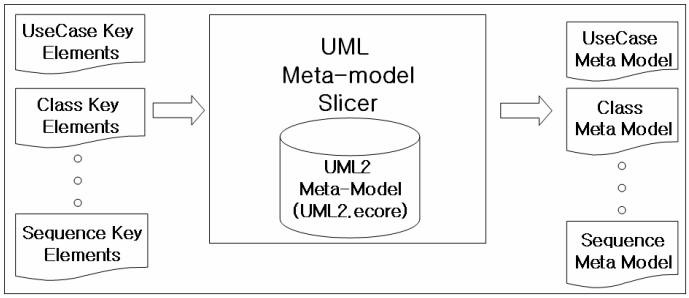

그림 6. UML 메타모델 모듈화를 위한 슬라이싱 도구

모델 슬라이싱 도구에서는 그림 6과 같은 과정을 거쳐 슬라이스된 메타 모델을 추출한다. 모델 슬라이싱 알고리즘을 이용하여 UML 메타 모델을 슬라이싱 하기 위해서는 우선 UML 메타 모델을 알맞은 형태로 저장해야 할 필요가 있다. 본 논문에서는 UML2 프로젝트에서 XMI로 기술한 UML2.x 메타 모델 정보를 사용하였다.

UML2 프로젝트는 모델링 도구 개발을 지원하는 메타모델 구현 및 범용 XMI(XML Metadata Interchange) 스키마, 테스트 케이스, 검증 규칙(validation rules)를 제공하는 프로젝트이다[14]. UML2 프로젝트는 UML2.x 메타 모델 정보를 XMI로 기술된 UML2.ecore 파일로 제공하고 있다.

모델 슬라이싱 도구는 3절에서 제시한 모델 슬라이싱 알고리즘에 초기 주요 요소 E를 입력으로 하여, UML2 메타 모델 집합 M에서 슬라이스된 모델 집합 M'를 추출한다. 그리고 초기 주요 요소 E에서, 슬라이스된 모델 집합 M'를 추출하는 데 필수적인 초기 주요 요소 E'를 찾아내었다. 필수적인 초기 주요 요소 E'는 초기 주요 요소 E의 부분집합을 새로운 입력으로 하여 추출되는 슬라이스된 모델 집합이 기존의 슬라이스된 모델 집합 M'와 같은 부분집합 중 가장 작은 부분집합으로 선택하였다.

UML2.ecore 파일에는 UML2.0 명세에서 사용하고 있는 263개의 요소가 정의되어 있다. 여기서는 UML2.0 명세를 참조하여 UML2.0 전체 메타 클래스 정보에서 각 다이어그램에서 필요로 하는 초기 주요 요소 E를 선택하여 슬라이스된 모델 집합 M'를 구한다.

표 1은 UML2.0에서 제시된 전체 다이어그램 중 몇 개의 다이어그램에 대해, 그 다이어그램에서 사용되는 초기 주요 요소의 수 |E|를 보여준다. 두 번째 열에서는 슬라이스된 모델 집합 M'를 추출하기 위한 필수적인 초기 주요 요소의 수 |E'| 를 보여준다. 세 번째 열에서는 초기 주요 요소 E를 입력으로 하여 3절의 모델 슬라이싱 알고리즘을 적용하였을 때 추출되는 슬라이스된 모델 집합 M'의 모델 요소의 수 |M'|를 보여준다. 그리고 마지막 열에는 슬라이스된 모델 집합 요소의 수 |M'|와 UML2 프로젝트의 전체 모델 요소의 수 |M|와의 비 (|M'|/|M|)를 계산하였다.

표 1. UML2.0 다이어그램의 초기 주요 요소와 슬라이스된 모델 집합 요소의 수

다이어그램 이름	초기 주요 요소의 수(|E|)	필수적인 초기 주요 요소의 수(|E'|)	슬라이스된 모델 집합 요소의 수(|M'|)	전체 UML2 메타 모델 요소와의 비 (|M'|/|M|)
Class Diagram	55 개	28 개	68 개	25.9%
Component Diagram	9 개	4 개	22 개	8.4%
Use Case Diagram	13 개	8 개	23 개	8.7%
Deployment Diagram	20 개	9 개	44 개	16.7%
Sequence Diagram	42 개	26 개	65 개	24.7%
Activity Diagram	52 개	30 개	70 개	26.6%
State-machine Diagram	25 개	13 개	51 개	19.4%

실험 결과, 전체 UML2 모델 요소 중 슬라이스된 모델 집합 요소(|M'|/|M|)는 최대 27% 정도였으며, 평균 19%정도의 요소를 사용하여 각 다이어그램의 모든 필수 요소를 추출해낼 수 있었다. 즉, 다이어그램 요소의 복잡도가 전체 다이어그램

요소를 모두 사용할 때와 비교하여 1/5 정도로 감소하였다.

초기 주요 요소의 수 |E|에 비해 필수적인 초기 주요 요소의 수 |E'|는 평균 55% 정도였다. 즉, 초기 주요 요소 중 55% 정도의 필수적인 초기 주요 요소만을 선택하더라도 필요한 모든 요소를 포함하는 슬라이스된 모델 집합을 추출해 낼 수 있다는 것을 알 수 있다.

표 2~8은 표 1 에서 제시된 각 다이어그램의 초기 주요 요소 E 및 필수적인 초기 주요 요소 E'를 보여준다.

표 2. 클래스 다이어그램의 초기 주요 요소와 필수적인 초기 주요 요소

E	Element, Comment, Relationship, Package, GeneralizationSet, DirectedRelationship, NamedElement, Type, VisibilityKind, PackageableElement, Class, Namespace, ElementImport, PackageImport, MultiplicityElement, Expression, Abstraction, ValueSpecification, TypedElement, Operation, OpaqueExpression, LiteralSpecification, InstanceValue, Usage, InstanceSpecification, LiteralBoolean, LiteralInteger, LiteralString, LiteralUnlimitedNatural, LiteralNull, Constraint, Slot, StructuralFeature, Classifier, DataType, RedefinableElement, Generalization, Property, Feature, ParameterDirectionKind, Parameter, BehavioralFeature, Association, PrimitiveType, Enumeration, Interface, EnumerationLiteral, PackageMerge, Dependency, AggregationKind, Realization, Substitution, BehavioralClassifier, InterfaceRealization, AssociationClass,
E'	Comment, ElementImport, PackageImport, Package, Expression, OpaqueExpression, InstanceValue, LiteralBoolean, LiteralInteger, LiteralString, LiteralUnlimitedNatural, LiteralNull, Constraint, Slot, Generalization, Property, Parameter, Operation, PrimitiveType, Enumeration, EnumerationLiteral, PackageMerge, Usage, Substitution, Interface, InterfaceRealization, AssociationClass, GeneralizationSet

* Composite Structure 부분 제외

표 3. 컴포넌트 다이어그램의 초기 주요 요소와 필수적인 초기 주요 요소

E	Class, Component, Interface, Classifier, Connector, Behavior, ConnectorKind, PackageableElement, Relaization
E'	Component, Interface, Connector, Behavior

* Composite Structure 부분 제외

표 4. 디플로이먼트 다이어그램의 초기 주요 요소와 필수적인 초기 주요 요소

E	Classifier, Abstraction, Artifact, Manifestation, PackageableElement, Operation, Property, Class, Node, Device, ExecutionEnvironment, Association, CommunicationPath, NamedElement, DeploymentTarget, InstanceSpecification, Dependency, Deployment, DeployedArtifact, DeploymentSpecification
E'	Manifestation, Operation, Property, Device, ExecutionEnvironment, CommunicationPath, InstanceSpecification, Deployment, DeploymentSpecification

표 5. 유스케이스 다이어그램의 초기 주요 요소와 필수적인 초기 주요 요소

E	RedefinableElement, BehavioredClassifier, Classifier, ExtensionPoint, UseCase, Actor, Extend, Include, Constraint, DirectedRelationship, NamedElement, Association, Generalization
E'	ExtensionPoint, UseCase, Actor, Extend, Include, Constraint, Association, Generalization

표 6. 스테이트머신 다이어그램의 초기 주요 요소와 필수적인 초기 주요 요소

E	Behavior, TransitionKind, PseudostateKind, StateMachine, Namespace, NamedElement, Region, Vertex, Transition, Pseudostate, State, ConnectionPointReference, FinalState, Trigger, Constraint, RedefinableElement, Classifier, TimeEvent, DirectedRelationship, Port, Interface, ProtocolStateMachine, ProtocolConformance, Operation, ProtocolTransition
E'	Region, Pseudostate, ConnectionPointReference, FinalState, Trigger, Constraint, TimeEvent, Port, Interface, ProtocolState-Machine, ProtocolConformance, Operation, ProtocolTransition

표 7. 시퀀스 다이어그램의 초기 주요 요소와 필수적인 초기 주요 요소

E	Action, ActionExecutionSpecification, Behavior, BehaviorExecutionSpecification, CombinedFragment, ConnectableElement, Connector, ConsiderIgnoreFragment, Constraint, Continuation, CreationEvent, DestructionEvent, Event, ExecutionEvent, ExecutionOccurrenceSpecification, ExecutionSpecification, Gate, GeneralOrdering, Interaction, InteractionConstraint, InteractionFragment, InteractionOperand, InteractionOperatorKind, InteractionUse, Lifeline, Message, MessageEnd, MessageEvent, MessageKind, ValueSpecification, MessageOccurrenceSpecification, MessageSort, NamedElement, Namespace, OccurrenceSpecification, OpaqueExpression, Operation, PartDecomposition, SendOperationEvent, SendSignalEvent, Signal, StateInvariant,
E'	ExecutionEvent, CreationEvent, SendOperationEvent, SendSignalEvent, Operation, Signal, DestructionEvent, Interaction, StateInvariant, Lifeline, ConnectableElement, OpaqueExpression, Message, Connector, MessageOccurrence-Specification, GeneralOrdering, ExecutionOccurrence-Specification, ActionExecutionSpecification, BehaviorExecution-Specification, Action, InteractionOperand, Continuation, ConsiderIgnoreFragment, InteractionConstraint, Gate, PartDecomposition

표 8. 액티비티 다이어그램의 초기 주요 요소와 필수적인 초기 주요 요소

E	Action, Activity, ActivityEdge, ActivityFinalNode, ActivityGroup, ActivityNode, ActivityParameterNode, ActivityPartition, Behavior, BehavioralFeature, CentralBufferNode, Classifier, Clause, ConditionalNode, Constraint, ControlFlow, ControlNode, DataStoreNode, DecisionNode, Element, ExceptionHandler, ExecutableNode, ExpansionKind, ExpansionNode, ExpansionRegion, FinalNode, FlowFinalNode, ForkNode, InitialNode, InputPin, InterruptibleActivityRegion, JoinNode, LoopNode, MergeNode, MultiplicityElement, NamedElement, Namespace, ObjectFlow, ObjectNode, ObjectNodeOrderingKind, OutputPin, Parameter, ParameterEffectKind, ParameterSet, Pin, RedefinableElement, SequenceNode, State, Structured-ActivityNode, TypedElement, ValueSpecification, Variable
E'	Activity, ActivityParameterNode, Parameter, ActivityFinalNode, InitialNode, ControlFlow, ObjectFlow, ForkNode, JoinNode, MergeNode, DecisionNode, FlowFinalNode, ActivityPartition, ValueSpecification, Constraint, State, DataStoreNode, BehavioralFeature, ParameterSet, InterruptibleActivityRegion, Variable, ConditionalNode, SequenceNode, Clause, LoopNode, OutputPin, InputPin, ExceptionHandler, ExpansionRegion, ExpansionNode

5. 결론

본 논문에서는 UML의 특정 다이어그램 요소만을 UML 모델에서 슬라이싱하는 알고리즘을 제시하고 실제로 UML 메타모델에 알고리즘을 적용해 보았다. 슬라이싱 알고리즘을

적용하여 얻어낸 결과는 각 다이어그램 요소만을 필요로 하는 UML 모델 도구 개발이나 분석도구 개발에 유용하게 사용 될 수 있다. 즉, UML 메타모델을 슬라이싱 함으로써 매우 경량화된 UML 메타모델만으로도 UML의 특정 다이어그램 메타모델을 활용할 수 있게 되었다. 실제 결과에서 유스케이스 다이어그램이나 컴포넌트 다이어그램의 경우는 매우 작은 메타모델만으로도 해당 다이어그램의 요소를 모두 표현하는 것을 볼 수 있다.

향후에는 본 논문에서 제시한 알고리즘으로 슬라이싱한 메타모델을 사용하여 UML 편집 도구를 개발하고 도구 개발 과정을 통해서 슬라이싱 알고리즘을 개선해 나갈 계획이다. 본 논문에서 제시한 결과를 사용하여 실제 도구를 개발하는 과정에서 발견되는 부족한 부분을 모두 해결할 수 있도록 하겠다.

참고문헌

[1] OMG(Object Management Group), Unified Modeling Language Specification, 2005

[2] Colombo, P., Pradella, M., and Rossi, M. A UML 2-compatible language and tool for formal modeling real-time system architectures. In Proceedings of the 2006 ACM Symposium on Applied Computing, 2006

[3] Kagdi, H., Maletic, J. I., and Sutton, A. Context-Free Slicing of UML Class Models. In Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005

[4] Kai Pan, Sunghun Kim, E. James Whitehead, Jr., Bug Classification Using Program Slicing Metrics. Proceedings of the 6th IEEE International Workshop on Source Code Analysis and Manipulation, September 27-29, 2006.

[5] Sloane, A. M. and Holdsworth, J. Beyond traditional program Slicing. In *Proceedings of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysi, 1996*

[6] Korel, B. Rilling, J. Program Slicing in understanding of large programs, Proceedings., 6th International Workshop on Program Comprehension, Jun 1998

[7]Louise A. Dennis Enhancing Theorem Prover Interfaces with Program Slice Information, User Interfaces for Theorem Provers (UITP 2006), 2006

[8]Korel, B. and Rilling, J. Dynamic Program Slicing in Understanding of Program Execution. In *Proceedings of the 5th International Workshop on Program Comprehension, 1997*

[9]De Lucia, A. Program Slicing: Methods and applications. In 1st IEEE International Workshop on Source Code Analysis and Manipulation, pp. 142-149, 2001

[10]Weiser, M. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering, 1981*

[11]Matthew D., and John H. Slicing software for model construction. Proceedings of the 1999 ACM Workshop on Partial Evaluation and Program Manipulation, January 1999

[12]OMG(Object Management Group), Meta Object Facility (MOF) Specification, 2003

[13]Colin A., Thomas K. Model-Driven Development: A Metamodeling Foundation, *IEEE Software*, vol. 20, no. 5, pp. 36-41, 2003

[14]Model Development Tools(MDT) project - UML2 project, http://www.eclipse.org/modeling/mdt/?project=uml2