JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 24, No. 2: 244~251, Feb. 2020

UML 다이어그램 도구를 위한 다이어그램 정보의 구축과 설계

김윤호*

A Design of Constructing Diagram Repository for UML Diagram Tools

Yun-Ho Kim*

*Professor, Department of Computer Engineering, Andong National University, Andong, 36729 Korea

요 약

본 논문에서는 UML 다이어그램의 정보를 구문적으로 분석하여 클래스들의 정보를 구조적으로 구축하는 정보 저장소인 '메타 클래스 레포지토리 (MCR)'의 구현 방법을 제시한다. 구문적으로 분석된 클래스의 정보를 구조적으로 구축하기 위하여 수퍼 클래스인 '메타 클래스 (meta-class)'를 정의하고, 이들 메타 클래스들의 컬렉션으로 구성된 레포지토리를 구축하여 정보를 관리한다. 또한, MCR이 보유하고 있는 메타 클래스 정보에 기반하여 클래스에 상응하는 코드를 생성하기 위하여 '코드 생성 엔진 (CGE)'을 설계하여 제시한다. 코드 생성 엔진의 로직을 구성함에 있어서는 클래스에 대한 정보와 프로그래밍 언어의 구문 규칙이 합법적으로 조합되어 코드가 생성되어야 한다. 따라서, 이러한 클래스 다이어그램으로부터 코드를 생성하는 데에 MCR과 CGE가 통합적으로 협력하여 수행될 수 있도록 구현하는 방법을 제시한다. CGR의 동작 메커니즘에 대한 알고리즘을 유한 상태 머신 형태로 제시함으로써 CGE의 로직을 형식화하여 표현함과 동시에 구현 상의 용이함을 취할 수 있게 하였다.

ABSTRACT

This paper presents a design of the Meta-Class Repository (MCR) which maintain syntactically analyzed and structured meta-class information from UML diagrams, and then proposes 'meta-class,' also known as super-class, to construct structured information analyzed syntactically. The MCR is a collection of these meta-classes which contains the information extracted from diagrams. This paper also presents a design of the Code Generation Engine (CGE) which roles generating codes corresponding classes from UML diagrams based on the MCR maintaining a collection of meta-classes which is syntactically-analyzed and constructed in previous process. The logics of CGE are designed to generate codes collaborated with MCR and CGE with integration. The logics of CGE mechanism is presented with the form of finite state machine to present the algorithms of code generation formally and have the advantages of simplicity and easiness in development.

키워드 : UML 다이어그램, 다이어그램 정보 구조화, 다이어그램 형식화, 자동 코드 생성, 객체 지향 소프트웨어 시스템

Keywords : UML diagrams, Diagram information structuring, Diagram formalization, Automatic code generation, Object-oriented software system

Received 12 November 2019, **Revised** 17 December 2019, **Accepted** 18 December 2019

*** Corresponding Author** Yun-Ho Kim(E-mail:javian99@gmail.com, Tel:+82-54-820-5898)
Professor, Department of Computer Engineering, Andong National University, Andong, 36729 Korea

Open Access http://doi.org/10.6109/jkiice.2020.24.2.244 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

현재 소프트웨어는 대규모의 고도로 집체적인 성향을 띠고 있으며, 개발 시간의 단축 또한 중요한 개발 요소로 인식되고 있다. 이러한 소프트웨어 개발 시에 제기되는 문제를 해결하기 위하여 제시된 객체 지향 개발에서 모델링은 필수적인 요소이며, 이를 지원하기 위한 모델링 도구의 사용은 이미 기본적인 개발 환경이 되어 있다. 소프트웨어 모델링을 위한 표준으로서 제시된 UML [1]을 지원하는 여러 다이어그래밍 도구들이 제작되어 배포되고 있는데, 이들은 각기 도구의 특성에 고유한 기능 상의 장점과 단점을 내재하고 있다. 따라서, 여러 개발 환경에 특화된 개발 도구의 제작 필요성에 부응하여 지속적으로 다이어그램 도구의 기반 기술에 대한 여러 연구 [2-9]가 지속되고 있다.

UML 다이어그램 도구들의 개발에서 주요 이슈가 되는 것은 클래스 다이어그램과 그에 대한 정보의 구축을 하는 방법이며, 이는 클래스 다이어그램이 다른 다이어그램의 구현의 기반이 되기 때문이다. 또한 코드 생성과 관련하여 다이어그램이 내포하고 있는 정보를 통합적으로 구축하는 것 또한 주요한 구비요소로서 널리 수용되고 있다. 기존의 연구들에서는 클래스 다이어그램에 대한 정보를 구축하는 방법의 기술에 있어서 비형식적 기술에 의존하거나 [2-6], 2-레벨 문법 등을 이용하여 형식화하는 방법[7-9]을 제시하였다. 이들 연구에서는 형식화하는 점에서는 장점이 있으나 실제 시스템의 구현상에 있어서는 구현 언어와의 간격이 커서 개발에 직접적인 적용이 용이하지 않은 단점을 가지고 있다.

본 논문에서는 UML 다이어그램 정보를 입력받아 클래스에 대한 정보를 구축하고 클래스 코드 생성을 위한 엔진을 제시하고자 한다. 클래스 구문적 정보를 체계적으로 저장하기 위하여 메타 클래스를 설계하고 이에 대한 메타 클래스 정보 저장소를 구축하는 방법을 제시한다. 또한, 클래스 정보 저장소에 기반하여 코드를 생성하기 위하여 클래스 코드 생성을 위한 엔진의 구성을 제시한다. 클래스 정보로부터 클래스 코드 생성의 처리 과정에 대한 전체적인 절차를 그림 1에서 보인다. 클래스 다이어그램으로부터 자바 코드를 생성하는 과정에서 메타 클래스 정보 저장소와 코드 생성 엔진이 개입하여 클래스 코드를 생성하게 된다. Ⅱ장과 Ⅲ장에서 이에 대한 내용을 상술한다.

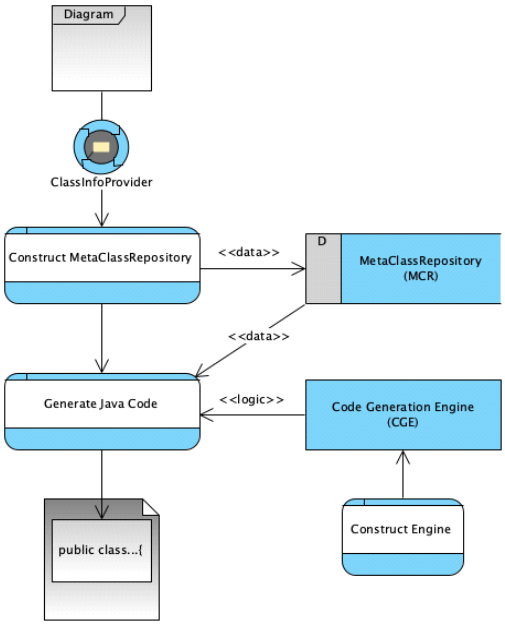

Fig. 1 Overview of processes and datastores for constructing class information repository and generating codes from diagrams

Ⅱ. UML 다이어그램의 메타 정보

이 장에서는 그림 1에서 보인 바와 같이 클래스 다이어그램으로부터의 정보를 구문적으로 분석하여 클래스 정보를 저장하는 저장소 (**MetaClassRepository, MCR**)를 구축하는 방법을 제시한다. 클래스 정보 저장소는 코드 생성에 적용될 수 있도록 코드 생성 로직 (**Code Generation Engine, CGE**)과 통합적으로 적용될 수 있도록 구축된다.

2.1. 클래스의 메타 정보와 메타 클래스

UML에서 클래스는 클래스의 가시성 (visibility), 클래스의 이름, 클래스의 속성, 클래스의 오퍼레이션으로 정의된다. 클래스의 속성은 다시 속성의 가시성, 속성의 이름, 속성의 타입, 그리고 속성의 기본값 (default)으로 정의된다. 또한 클래스의 오퍼레이션은 오퍼레이션의 가시성, 오퍼레이션의의 이름, 오퍼레이션의 파라미터 리스트, 오퍼레이션의 반환 값인 리턴 (return)으로 정의된다.

[class] --> visibility name [attributes] [operations]
[attribute] --> visibility name: type = default
[operation] --> visibility name (parameter-list) : return type

오퍼레이션의 파라미터 리스트는 다시 파라미터의 이름, 파라미터의 타입, 파라미터의 기본 값으로 정의된다. 클래스의 구성 요소에 대한 값들을 저장하고 처리하기 위해서는 이러한 구문적 내용들에 대한 정보를 구조적으로 관리하여야 한다. 본 논문에서는 클래스의 이러한 구문적 정보를 구조적으로 가지고 있는 클래스 즉, '메타 클래스 (meta-class)'인 **MetaClass**를 설계하고 이를 기반으로 논의를 전개한다. 메타 클래스는 클래스의 인스턴스가 객체가 아닌 클래스를 인스턴스로 하는 클래스 즉, 클래스들의 클래스인 파워 클래스 [10]를 의미한다. 또한, 클래스의 속성, 오퍼레이션, 파라미터에 대한 구문적 정보를 가지는 메타 속성 클래스, 메타 오퍼레이션 클래스, 메타 파라미터 클래스로 **MetaAttribute**, **MetaOperation**, **MetaParameter** 클래스를 설정한다.

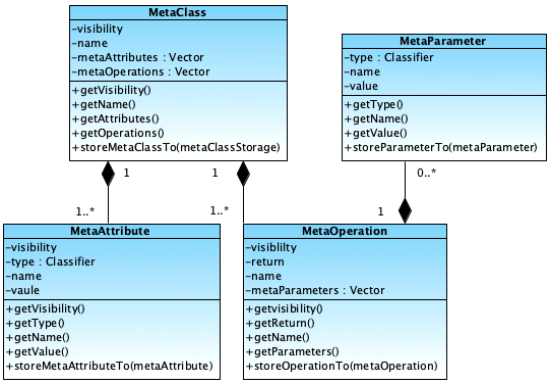

Fig. 2 Class diagram of Meta-Class

그림 2는 클래스의 구문적 정보를 가지는 메타 클래스 **MetaClass**의 구성을 보여준다. **MetaClass**는 속성으로서 visibility (클래스의 가시성), name (클래스의 이름), metaAttributes (클래스의 속성들), metaOperations (클래스의 오퍼레이션들)으로 구성된다. 클래스의 속성들은 한 클래스에 대해 1..*의 다수성 (multiplicity)을 가지므로 속성들의 메타 클래스인 **MetaAttribute** 클래스는 **MetaClass** 클래스와 1..*의 집합 관계 (aggregation)를 가진다. 또한 클래스의 오퍼레이션들 역시 한 클래스에 대해 1..*의 다수성을 가지므로 오퍼레이션들의 메타 클래스인 **MetaOperation** 클래스는 **MetaClass**와 1..*의 집합 관계를 가진다.

한편, **MetaAttribute** 클래스는 멤버 속성으로서 visibility (속성의 가시성), name (속성의 이름), type (속성의 클래스 타입), value (속성의 값)으로 구성된다. **MetaOperation** 클래스는 속성으로서 visibility (오퍼레이션의 가시성), name (오퍼레이션의 이름), return (오퍼레이션의 리턴), 그리고 metaParameters (오퍼레이션의 파라미터 리스트)로 구성된다. 오퍼레이션의 파라미터는 0..*의 다수성을 가지므로 파라미터는 한 오퍼레이션에 대해 0..*의 집합 관계를 가진다. 즉, 파라미터의 메타 클래스인 **MetaParameter** 클래스는 **MetaOperation** 클래스에 대해 0..*의 집합관계를 가진다. **MetaParameter** 클래스는 속성으로서 name (파라미터의 이름), type (파라미터의 클래스 타입), value (파라미터의 값)으로 구성된다.

2.2. 메타 클래스 레포지토리의 구축

클래스들에 대한 정보를 저장하기 위해서는 상응하는 메타 클래스들을 생성하고 이들을 저장하는 저장소가 필요하다. 이 절에서는 메타 클래스들을 생성하고 메타 클래스들을 저장하기 위한 저장소를 구축하는 방법을 기술한다. 본 논문에서는 클래스에 대한 정보를 제공받아 메타 클래스들의 컬렉션을 저장하는 클래스 메타 클래스 레포지토리 클래스 **MetaClassRepository (MCR)**를 설정한다. 메타 클래스 레포지토리 클래스 **MCR**은 그림 3에서 보인 바와 같이 메타클래스들의 컬렉션을 저장하는 metaClassStorage를 내포하고 있는 클래스이다.

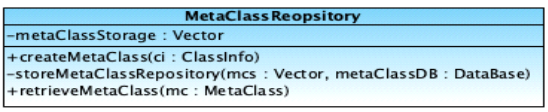

Fig. 3 Class diagram of MCR class

MCR 클래스의 createMetaClass() 오퍼레이션은 클래스 정보 (ci)를 파라미터로 전달받아 상응하는 메타 클래스를 생성하는 역할을 한다. retriveMetaClass() 오퍼레이션은 저장되어 있는 메타 클래스를 회수하는 기능을 한다. storeMetaClassRepository() 오퍼레이션은 생성된 메타 클래스 저장소를 metaClassDB에 저장하는 기능을 한다. 저장소 metaClassStorage는 로컬 시스템에 저장할 수도 있고, 데이터베이스 인터페이스를 경유하

여 외부 데이터베이스 시스템에 저장할 수도 있다.

MCR 클래스는 클래스들에 대한 정보를 입력으로 받아 메타 클래스를 저장하는 역할을 하는 클래스이다. **ClassInfoProvider (CIP)** 클래스 정보를 제공하는 역할을 하는 클래스이다. **CIP**를 통해 클래스 정보를 받아 상응하는 메타 클래스을 구축하는 과정을 그림 4에서 보인다. **CIP** 클래스는 클래스에 대한 정보를 ci와 함께 **MCR** 클래스에게 createMetaClass() 메시지를 보냄으로써 클래스 정보 구축이 시작된다. '**Loop** 결합 프래그먼트 (combined fragment)' 표시는 반복을 의미하며, 여기서는 메타 클래스를 구축할 클래스 정보가 있는 동안은 박스로 표시된 구역의 시퀀스들을 반복함을 의미한다.

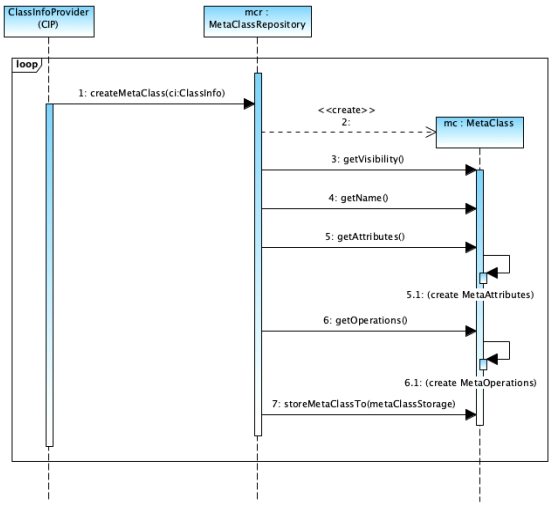

Fig. 4 Procedure of constructing MetaClass information

그림 4에서 클래스 **MCR** 객체 (mcr)은 메타 클래스를 생성하라는 메시지 (1번 시퀀스)를 받으면, 먼저 메타 클래스 객체 (mc)를 생성하고 (2번 시퀀스), 클래스의 각 구성 요소들에 대한 정보를 메타 클래스 객체의 속성에 저장한다 즉, 클래스의 가시성과 클래스의 이름, 클래스의 속성들, 클래스의 연산들에 대한 정보를 구축하고 (3번 ~ 6번 시퀀스), 완성된 메타 클래스 객체를 메타 클래스 저장소에 저장한다(7번 시퀀스).

getVisibility(), getName() 오퍼레이션은 클래스의 가시성 정보와 클래스의 이름을 가져와서 메타 클래스의 visibility, name 속성에 저장하는 임무를 수행한다. getAttributes() 오퍼레이션은 클래스의 속성들에 대한 정보를 가져와서 **MetaClass**의 metaAttributes에 저장한다 (그림4의 5.1번 시퀀스). getOperations() 오퍼레이션은 클래스의 오퍼레이션들에 대한 정보를 가져와서 **MetaClass**의 metaOperations에 저장하는 역할을 한다 (그림4의 6.1번 시퀀스). 메타 속성과 메타 오퍼레이션은 하나 이상의 다수성의 관계를 가지고 있으므로, 이에 대한 정보를 생성하는 역할은 **MetaAttribute** 클래스와 **MetaOperation** 클래스에 정보 구축의 임무를 위임하는 방식으로 처리한다.

그림 4에서 시퀀스 5.1 (create MetaAttributes)로 표시한 메타 속성 클래스 **MetaAttribute**를 구축하는 세부 절차를 그림 5에서 보인다.

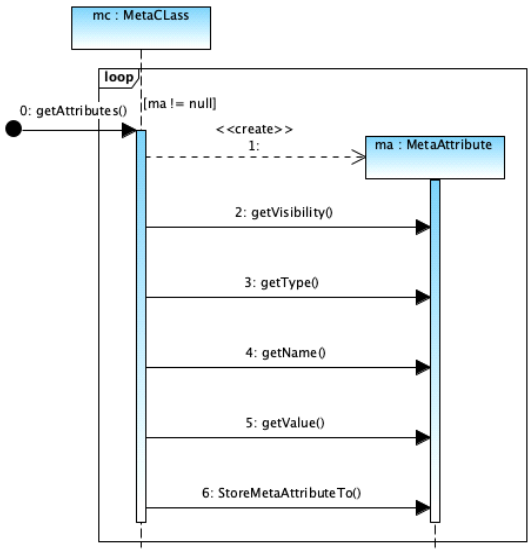

Fig. 5 Procedure of constructing MetaAttribute class information

메타 클래스는 메타 속성 정보를 구성하기 위하여 먼저 메타 속성 객체 (ma)를 생성하고 (1번 시퀀스), 클래스의 속성 구조 정보에 따라 속성의 가시성 (2번 시퀀스), 속성의 타입 (3번 시퀀스), 속성의 이름 (4번 시퀀스) 그리고 속성의 기본 값 (5번 시퀀스)에 대한 정보를 구성하여 **MetaClass**의 metaAttributes에 저장한다 (6번 시퀀스). 속성은 하나 이상의 다수성을 가지므로 처리할 속성이 있는 동안 이 과정을 반복한다. [loop] 프래그먼트는 속성 정보가 있는 동안 반복됨을 표시한다.

그림 4에서 시퀀스 6.1 (create MetaOperations)로 표

시한 메타 오퍼레이션 클래스 **MetaOperation**을 구축하는 세부 절차는 그림 6에 보인 바와 같다. 메타 클래스는 메타 오퍼레이션 정보를 구성하기 위하여 첫 번째로 메타 오퍼레이션 객체 (mo)을 생성한다. 다음으로 오퍼레이션의 가시성, 오퍼레이션의 리턴, 오퍼레이션의 이름, 오퍼레이션의 파라미터 정보를 가져와서 메타 오퍼레이션의 visibility, return, name, metaParameter의 내용을 구성한다 (1번~5번 시퀀스). 마지막으로 구성된 메타 오퍼레이션 객체 (mo)를 메타 클래스의 metaOperations에 저장한다.

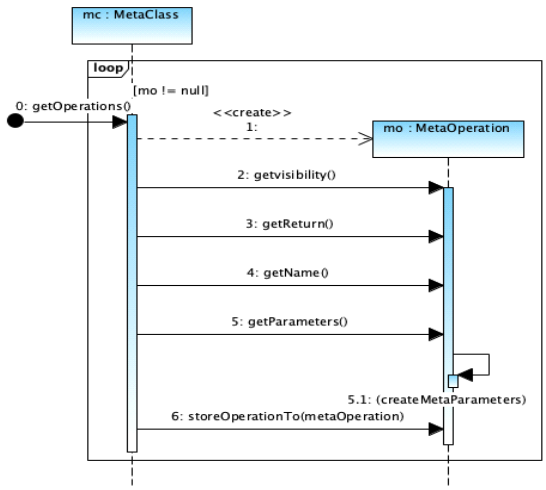

Fig. 6 Procedure of construncting MetaOperation class information

클래스의 오퍼레이션은 0..*의 다수성 관계를 파라미터와 가지므로, 클래스의 파라미터 정보가 있는 동안 파라미터 정보를 반복적으로 구성해야 한다. 따라서, 그림 5에서 시퀀스 5.1 (create MetaParameters)로 표시한 메타 파라미터 클래스를 구축하는 세부 철차를 그림7에서 보여 준다. 메타 오퍼레이션 클래스에서 메타 파라미터 클래스를 구축하는 절차는 다음과 같다. 첫 번째로 **MetaParameter** 클래스의 객체 (mp)를 생성하고, 구문 정보에 따라서 파라미터의 타입, 파라미터의 이름, 파라미터의 기본 값을 가져와서 메타 파라미터 클래스의 type, name, value의 내용을 구성한다 (1번 ~ 4번 시퀀스). 마지막으로 구성된 메타 파라미터의 객체 (mp)를 **MetaOperation** 클래스의 metaParameters에 저장한다 (5번 시퀀스). 오퍼레이션의 파라미터는 한 오퍼레이션에 대해서 0..*의 다수성 관계를 가지므로 파라미터 정보가 있는 동안 반복적으로 수행된다. 지금까지 II장에서 기술한 절차에 의해 클래스 정보를 제공받아서 메타 클래스 레포지토리를 구축하는 과정을 수행하게 된다.

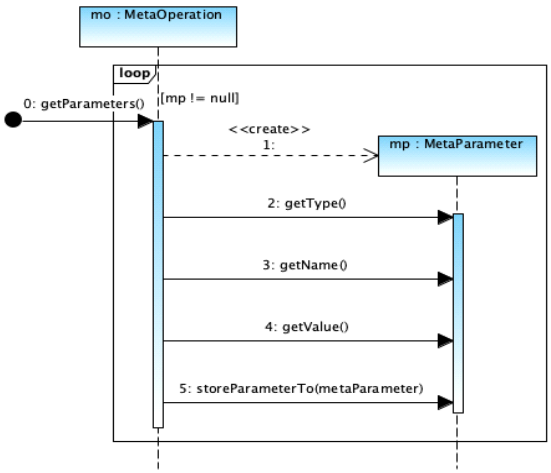

Fig. 7 Procedure of constructing MetaParameter class information

Ⅲ. 메타 클래스로부터의 코드 생성

이 장에서는 그림 1에서 보인 구축된 메타 클래스 저장소인 **MCR** 클래스로부터의 메타 클래스 정보로부터 코드를 생성하는 코드 생성 엔진 (**Code Generation Engine, CGE**)을 설계하는 방법을 기술한다. 본 논문에서는 코드 생성 언어로 자바 프로그래밍 언어를 채택하였다. 또한, 클래스의 구문 정보를 가진 메타 클래스의 정보를 기반하여 대응되는 코드 생성을 하기 위한 알고리즘을 상태 다이어그램 형태의 유한 상태 머신 (Finite State Machine)으로 표현한다.

3.1. 클래스 코드 생성 엔진

UML 클래스 다이어그램에 대한 자바 언어에서 클래스 코딩의 기본 형태를 2-레벨 문법 형태로 나타내면 다음과 같다.

STARTc => **public class** c_name **{**
    ATTRIBUTESc
    OPERATIONSc
}

여기서, 밑줄로 그어진 부분은 키워드로서 터미널(terminal)을 나타내며, c_name은 클래스 이름 값을 의미한다. *ATTRIBUTESc*는 넌터미널(nonterminal)로서 클래스의 속성들을 나타내며 *OPERATIONSc*는 넌터미널로서 클래스의 메소드들을 나타낸다. 이와 같은 형태의 표현식은 생성 표현식을 형식화하는 데는 장점이 있지만 이를 구현하는 문제는 별도로 다루어야 하는 단점이 있다.

본 논문에서는 이러한 문제를 해결하기 위하여 유한 상태 머신 형태로 코드 생성 로직을 제시한다. 이는 유한 상태 머신으로 표현된 알고리즘은 코드 생성 엔진의 알고리즘을 형식화하여 표현할 수 있는 동시에 구현 시에 상태 다이어그램의 코드 생성에 관한 연구 [11,12]를 이용하여 구현 상의 용이함을 얻을 수 있기 때문이다. 상태 다이어그램 형태의 오토마타로 생성 알고리즘을 표현하면 생성 알고리즘을 형식화하는 동시에 개발한 알고리즘의 구현 시에 직접적으로 참조할 수 있는 설계가 될 수 있다. 따라서, 본 논문에서는 제시하는 코드 생성 알고리즘을 유한 상태 머신 형태로 제시한다.

그림 8은 자바 언어로 클래스의 기본 구조를 코딩하는 알고리즘을 유한 상태 머신으로 표현한 것이다. 그림 8에서 **mc.visibility** 상태는 클래스의 가시성을, **mc.name** 상태는 클래스의 이름을 처리하는 상태를 의미한다. 각 상태에서 이벤트 **entry**는 그 상태에 진입할 때에 행하는 액션을 의미하며, 이벤트 **exit**는 그 상태에서 진출할 때에 행하는 액션을 의미한다. **keyword** 상태는 키워드를 출력하는 상태를 의미하며 " "의 내에 있는 문자열을 출력하는 상태이다. <<**mc.MetaAttribute**>> 상태는 클래스의 속성들에 대한 코드를 생성하는 수퍼 상태를 의미하며 수퍼 상태는 세부적인 서브 상태 다이어그램을 가진다. <<**mc.MetaOperation**>> 상태는 클래스의 오퍼레이션 코드를 생성하는 수퍼 상태이며 세부적인 서브 상태 다이어그램을 가진다.

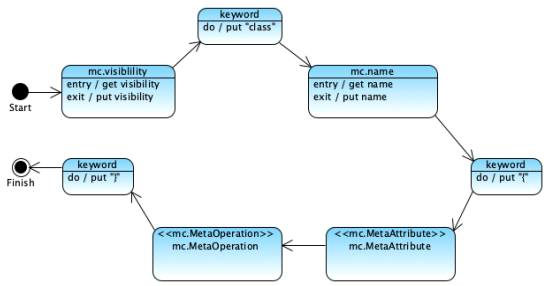

Fig. 8 Finite state machine for "class" code generation

그림 8에서 보인 클래스의 기본 형태에 대한 코드를 생성하는 절차는 먼저 mc.visibility 상태로 들어가게 되며, 클래스의 visibility 값을 가져와서 그 값을 출력하는 상태이다. 다음의 keyword 상태로 전이해서 "class"를 출력하고 mc.name 상태로 진입한다. mc.name 상태에서는 클래스의 name 값을 가져와서 출력하고 keyword 상태로 전이해서 "{"를 출력한다. 다음으로 <<meta.Attribute>> 상태로 전이해서 클래스의 속성들을 출력하고, <meta.Operation>> 상태로 전이해서 클래스의 오퍼레이션들을 출력한다. 마지막으로 keyword 상태로 전이해서 "}"를 출력하고 종료하게 된다.

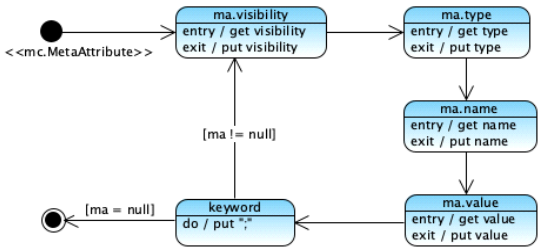

Fig. 9 Finite state machine for "attributes" code generation

그림 8의 클래스의 속성들을 출력하는 수퍼 상태 <<**mc.MetaAttribute**>>의 서브 상태 다이어그램을 그림 9에 보인다. 속성들을 출력하기 위해서 **ma.visibility** 상태에서 가시성 값을 가져와서 출력하고 **ma.type** 상태로 전이한다. 이 상태에서 **ma.type** 상태에서 속성의 이름 값을 가져와서 출력하고, 다음 상태인 **ma.value** 상태에서 속성의 기본 값을 가져와서 출력한다. **keyword** 상태에서 ";"을 출력한 후에, 더 이상의 속성이 있는 지를 검사하여 있으면 **ma.visibility** 상태로 전이해서 다음의 속성들을 출력하고, 남은 속성이 없으면 속성들을 출력하는 <<**ma.MetaAttribute**>> 상태를 종료하게 된다.

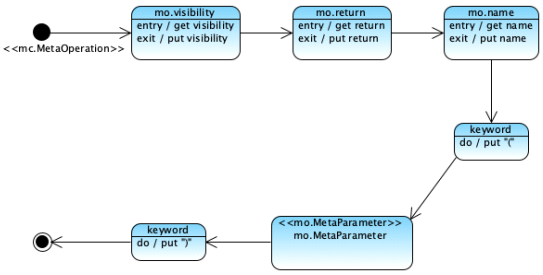

Fig. 10 Finite state machine for "operations" code generation

그림 8의 클래스의 오퍼레이션들을 출력하는 수퍼 상태인 <<**mc.MetaOperation**>>의 서브 상태 다이어그램을 그림 10에 보인다. 이 수퍼 상태에 진입하면 **mo.visibility** 상태에서 오퍼레이션의 가시성을 가져와 출력하고 **mo.return** 상태로 전이된다. **mo.return** 상태에서 리턴 값을 가져와서 출력하고 다음의 **mo.name** 상태에서 오퍼레이션 이름을 출력한다. 다음에는 파라미터 시작인 "(" 를 출력하고 파라미터 리스트를 출력하는 수퍼 상태 <<**mo.MetaParameter**>> 상태로 전이된다. 다음으로 **keyword** 상태에서 ")"를 출력하고 종료하게 된다.

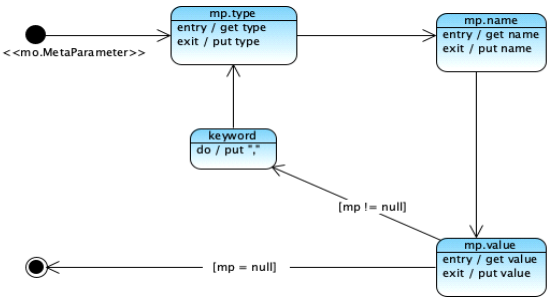

Fig. 11 Finite state machine for "parameter-list" code generation

파라미터 리스트는 하나 이상의 파라미터를 가지므로 이를 처리하기 위한 <<**mo.MetaParameter**>> 상태의 서브 상태 다이어그램을 그림 11에 보인다. <<**mo.MetaParameter**>> 상태로 진입하면 파라미터의 타입을 mo.type 상태에서 출력하게 되며 mo.name 상태에서 파라미터의 이름을 출력한다. 더 이상의 파라미터가 없으면 종료하고, 남은 파라미터가 있으면 **mo.type** 상태로 전이하여 반복하게 된다.

3.2. 클래스 코드 생성의 예

이 절에서는 UML 다이어그램으로부터 클래스 정보를 입력받아 코드를 생성하는 예를 보인다. 그림 12는 예제 클래스로서 Store 클래스와 이 클래스의 정보로부터 구축된 메타 클래스를 보인다. Store 클래스는 속성으로서 product와 amount들이 있으며, 오퍼레이션으로서 Delivery 타입의 d를 파라미터로 가지는 deliver() 메소드가 있다. Store 클래스의 메타 클래스 정보는 mc1이라는 이름의 **MetaClass** 객체로 구성되며, 속성 product, mount의 메타 속성 정보는 ma1과 ma2의 **MetaAttribute** 객체로 구성된다. deliver() 메소드는 **MetaOperation** 객체 mo1에, 파라미터는 MetaParamter 객체 mp1에 각각 구성된다. 이러한 정보에 기반하여 코드 생성 엔진 **CGE**에 의해 생성된 결과의 예를 그림 13에 보인다.

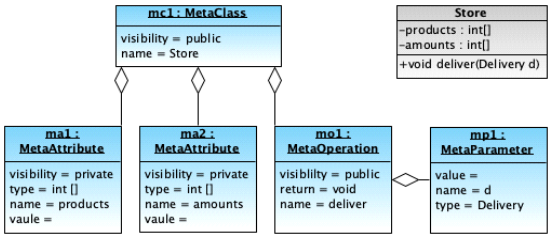

Fig. 12 A diagram of MetaClass's objects and class Store

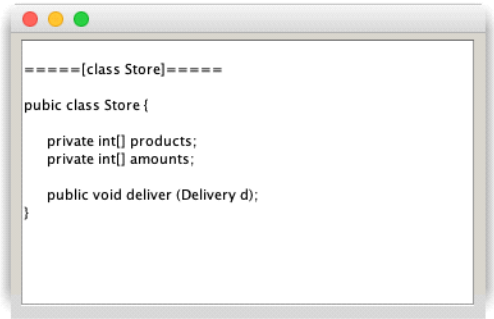

Fig. 13 A result of generating code for class Store

Ⅳ. 결 론

대규모의 고도로 집체적인 소프트웨어를 신속히 개발하기 위해서는 객체 지향적인 설계와 구현 및 설계 도구의 사용은 소프트웨어 개발 환경에서 필수적인 사항이다. 이를 위하여 일반적으로 사용되고 있는 UML 다이어그래밍 도구의 개발은 고도의 기술 수준을 갖춘 개발 환경의 구축과 객체 지향 프로그래밍에서 적용되는 원천 기술에 해당한다. 본 논문에서는 이러한 도구의 개발을 위하여 다이어그래밍 도구 구축의 핵심적인 역할을 하는 클래스 정보의 구축을 위하여 UML 클래스 구문 정보를 구조적으로 가지는 메타 클래스 정보 저장소를 구축하는 방법을 제시하였다. 또한 이 정보에 기반하여 코드 생성을 위한 메커니즘을 가진 코드 생성 엔진을 유한 상태 머신으로 설계하여 구현하는 방법을 제시하

였다. 한편, 오퍼레이션의 몸체 부분은 그것의 시퀀스 다이어그램으로부터의 정보에 기반하여 코드를 생성할 수 있다. 현재 구축된 시퀀스 다이어그램의 메시지 정보[13]에 기반하여 오퍼레이션의 몸체 코드를 생성하는 연구를 진행 중에 있다.

ACKNOWLEDGEMENT

This work was supported by a grant from 2016 Research Funds of Andong National University.

References

[1] Object Management Group (OMG), *Unified Modeling Language Specification*, Version 2.2, 2009.

[2] H. Zhang, "An approach for extracting UML diagram form object-oriented program based on J2X," *Advances in Engineering Reseach (IFMCA)*, vol. 113, pp. 266-276, 2016.

[3] A. Soumiya, and B. Mohamed, "Converting UML class diagrams into temporal object relational database," *International Journal of Electrical and Computer Engineering*, vol. 7, no. 5, pp. 2823-2832, 2017.

[4] M. Mukhtar, and B. Galadanci, "Automatic code generation from UML diagrams: The-State-of-the-art," *Science World Journal*, vol. 13, no. 4, pp. 47-60, 2018.

[5] P. Pawde, and V. Chole, "Generation of Java code structure from UML class diagrams," *International Journal of Innovative Science and Modern Engineering*, vol. 2, no. 7, pp. 7-10, 2014.

[6] J. Kim, and Y. Kim, "Constructing and Processing of the Metadata information for UML Class Auntorization Tool," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 15, no. 8, pp. 71-80, 2011.

[7] P. Kluisritrakul, and Y. Limpiyakorn, "Generation of Java code from UML sequence and class diagrams," *Information Science and Applications (ICISA)*, pp. 1117-1125, 2016.

[8] N. Sangal, E. Farrell, K. Lieberherr, and D. Lorenz, "Interaction schema: Compiling interactioins to code," *Proceedings of Technology of Object-Oriented Language and Systems*, pp. 268-277, Aug. 1999.

[9] I. Niaz, and J. Tanaka, "An Object-oriented approach to generate Java code from UML statecharts," *International Journal of Computer and Information Science*, vol. 6, no. 2, 2005.

[10] Y. Kim, "Resolving the runtime class reference problem of the type object design pattern by type object class," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 11, no. 3, pp. 500-506, 2007.

[11] M. Qu, L. Meng, X. Wu, and N. Cui, "Software Modeling and Automatic Code Generation Based on Reactive State Diagram," *International Conference on Computer Information Systems and Industrial Applications*, pp. 899-901, 2015.

[12] E. Sunitha, and P. Samuel, "Automatic code generation from UML state chart diagrams," *IEEE Access*, vol. 7, pp. 8591-8608, 2019.

[13] Y. Kim, "Information structuring of diagram repository for UML diagrams," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 23, no. 12, pp. 1588-1595, 2019.

김윤호(Yun-Ho Kim)

1983 경북대학교 전자공학과 학사
1993 경북대학교 컴퓨터공학과 석사
1997 경북대학교 컴퓨터공학과 박사
1997 – 현재 안동대학교 컴퓨터공학과 교수
※관심분야 : 객체지향시스템, 모바일 소프트웨어, 인공지능, 병렬처리