JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 24, No. 7: 857~865, Jul. 2020

메시지 패턴에 기반한 UML 시퀀스 다이어그램의 자동 코드 생성 방법

김윤호*

A Method of Automatic Code Generation for UML Sequence Diagrams Based on Message Patterns

Yun-Ho Kim*

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

요 약

본 논문에서는 시퀀스 다이어그램의 메시지 패턴에 기반하여 자바 코드를 생성하는 방법을 제시하고자 한다. 시퀀스 다이어그램에서 메시지는 몇 가지 유형이 전형적으로 나타낸다. 따라서 이들 유형을 분류하여 패턴으로 정의하고, 이들 각 패턴의 메시지에 대한 정보를 분석하여 코드 변환을 하기 위한 메타 정보를 구축한다. 시퀀스 다이어그램 메시지의 구조적 정보를 분석한 '메타 메시지' 정보를 MetaMessage 데이터스토어에 구축하고, 이들 메타 메시지 정보로부터 구성되는 '메타 메소드' 정보를 MetaMethod 데이터스토어에 구축한다. 이 과정에서 메타 클래스 정보와 메타 오브젝트 정보가 함께 구축되고 사용된다. 각 패턴의 메타 정보에 기반하여 타겟 프로그래밍 언어의 문법에 따라 메시지에 상응하는 코드를 생성하게 된다. 또한, 기존 연구에서는 통합적으로 잘 다루지 않았던 분기와 반복에 대한 복합 패턴을 제시함으로써 기본 패턴과 통합된 코드 생성을 가능하게 한다.

ABSTRACT

This paper presents a method for code generation of UML sequence diagrams based on message patterns. In the sequence diagrams, it is shown that messages are some types of forms typically. This paper classifies according to type as three patterns, and construct meta-information for code generation analysing structural infomation for each patterns. The meta-message of structural information (MetaMessage) is stored in the MetaMessage datastore and the meta-method information from the MetaMessage is stored in the MetaMethod datastore. And then, the structural information of MetaClass and MetaObject is constructed in each datastore too. For each pattern, this paper presents a method for code generation based on the meta information of message patterns and the syntax of target progamming language. Also, branching and looping that has been seldom handled integratedly in the previous works are handled as same as the basic patterns by classifying the branching pattern and the looping pattern for code generation integratedly.

키워드 : UML, 시퀀스 다이어그램, 자동 코드 생성, 객체 지향 설계, 객체 지향 소프트웨어 시스템

Keywords : UML, Sequence diagram, Automatic code generation, Object-oriented design, Object-oriented S/W system

Received 22 April 2020, **Revised** 23 April 2020, **Accepted** 1 May 2020

*** 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.2020.24.7.857 print ISSN: 2234-4772 online ISSN: 2288-4165

ⓒThis is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(http://creativecommons.org/li-censes/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright Ⓒ The Korea Institute of Information and Communication Engineering.

Ⅰ. 서 론

객체 지향 소프트웨어의 개발에서 UML [1]을 지원하는 개발 도구의 사용은 이제 보편적인 것이 되었다. 이러한 개발 도구들은 객체 지향적 설계뿐만이 아니라 설계된 다이어그램의 코드 변환 기능 또한 개발 도구에서의 기본 기능으로 이미 자리를 잡았다. 한편, 코드 생성 기술은 상용의 중량급 개발 도구의 주요 기능으로서의 유용성뿐만이 아니라 UML로 설계된 설계도에 상응하는 코드에 대한 실험 및 교육을 위한 경량급 도구의 개발에도 유용하게 사용되고 있는 요소 기술이다.

UML 시퀀스 다이어그램에 대한 지대한 관심은 클래스 다이어그램과 함께 자동 코드 생성을 위한 핵심 정보를 포함하고 있기 때문이다 [2-4]. 따라서, 시퀀스 다이어그램으로부터 코드 생성을 위한 많은 연구들이 진행되어 왔다. 이들 연구들의 접근 방법은 비형식적이거나 스키마타를 이용하거나 2-레벨 문법을 이용하는 방법들이 제시되었으나 [5-7], 이러한 방법은 코드 생성 규칙들이 적용하기에 복잡하거나 특정 툴의 형식을 전제하고 있어서 일반적인 코드 생성 알고리즘으로 사용하기에는 불편한 점이 많다 [8,9]. 따라서, UML과 타겟 언어의 문법의 대응을 효율적으로 하면서 구현이 용이한 유한 상태 머신 표현 코드 생성 알고리즘으로 표현하는 것이 코드 생성 규칙의 적용에 용이하다 [10-12].

본 논문에서는 시퀀스 다이어그램의 메시지 패턴에 기반하여 코드를 생성하는 방법을 제시하고자 한다. 먼저, 시퀀스 다이어그램의 전형적인 메시지 유형을 포착하여 이를 메시지 패턴들로 정의한다. 각 메시지 패턴에 대한 메타 정보를 구축하고, 이 메타 정보를 이용하여 각 메시지 패턴에 상응하는 코드 생성 알고리즘을 제안한다.

메시지의 메타 정보를 구축하고 이에 기반하여 코드를 생성하는 과정의 개관을 그림1에서 보인다. 시퀀스 다이어그램의 메시지의 구조적 정보를 분석한 '메타 메시지' 정보를 MetaMessage 데이터스토어에 구축하고, 이들 메타 메시지 정보로부터 구성되는 '메타 메소드' 정보를 MetaMethod 데이터스토어에 구축한다. 이 과정에서 메타 클래스 정보와 메타 오브젝트 정보가 함께 구축된다. 구축된 메타 메소드 정보로부터 시퀀스 다이어그램에 상응하는 자바 코드를 생성하게 된다. II장부터 IV장에서 이에 대한 상세한 내용을 기술한다.

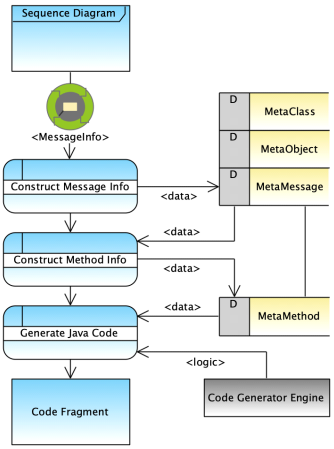

Fig. 1 Overview of constructing message information and generating codes from sequence diagram

Ⅱ. 메시지 메타 정보

2.1. 메시지와 메소드 정보 구축

시퀀스 다이어그램은 일련의 메시지들의 시퀀스들로 이루어져 있으며, 각 메시지는 소스 객체로부터 타겟 객체의 메소드에 대한 호출로 대응된다. 또한, 이들 일련의 메시지들은 소스 객체가 속한 클래스의 한 메소드의 몸체를 이루게 된다. 그림2에서 한 메소드와 그 메소드를 구성하는 일련의 메시지들 사이의 관계를 보인다. 메시지 1.1과 1.2는 타겟 객체 ot의 클래스 Target에 정의되어 있는 메소드 opr1()과 opr2()의 호출과 대응되며, 이들 메소드들은 소스 객체 os의 클래스 Source에 정의되어 있는 메소드 proc()의 몸체 코드들을 이루게 된다. 그림2는 좌측의 시퀀스 다이어그램에 대응되는 코드를 그림 우측에서 보여 준다. 이와 같이 시퀀스 다이어그램을 해석하여 코드를 생성하려면 먼저, 이러한 메시지 정보와 메소드 정보를 추출하고, 그 정보를 구조적인 형태로 저장하는 것이 필요하다.

따라서 본 논문에서는 메시지에 대한 구조적 정보를 가진 MetaMessage 클래스와 클래스 내에서 정의되는 메소드에 대한 정보를 가진 MetaMethod 클래스의 설계를 제시한다. MetaMessage 클래스는 MetaObject클래스와 MetaClass 클래스에 기반해서 설계되며, MetaMethod 클래스는 MetaMessage 클래스로 구성된다.

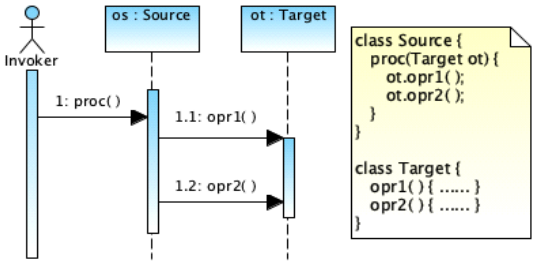

Fig. 2 A typical example of seqence diagram messages and corresponding code fragment

먼저 그림3에서 보는 바와 같이 메시지와 연관된 객체에 대한 메타 정보를 가진 MetaObject 클래스를 정의한다. MetaObject 클래스는 소스 객체 또는 타겟 객체의 객체명과 그 객체가 속한 클래스에 대한 정보를 가지고 있다. 또한, MetaObject 클래스는 객체에 대한 상세 정보를 저장하기 위하여 MetaClass 클래스를 사용한다 [13]. MetaClass 클래스는 어떠한 클래스의 상위 구조적 정보를 가지고 있는 메타 클래스로서, 클래스의 '속성', '오퍼레이션', 그리고 오퍼레이션과 연관된 '파라미터'에 대한 메타 정보를 가지고 있는 MetaAttribute, MetaOperation, MetaParameter 클래스들로 구성된다.

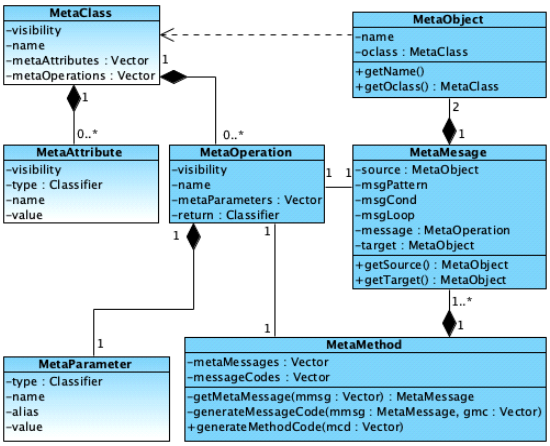

Fig. 3 The relatioship between class MetaMessage and class MetaClass

그림 3에서 메시지에 대한 상위 구조적 정보를 가지고 있는 메타 메시지 클래스 MetaMessage는 소스 객체와 타겟 객체에 대한 정보와 메시지의 오퍼레이션 정보, 그리고 메시지의 패턴에 대한 정보 등으로 구성된다. 메시지의 소스 객체와 타겟 객체에 대한 정보는 각각 MetaObject 클래스를 사용하여 구축된다. 메시지의 오퍼레이션에 대한 정보는 MetaOperation 클래스를 이용하여 정보를 저장한다. MetaMessage 클래스의 getMessage()는 한 메시지에 대하여 소스 객체, 타겟 객체, 메시지 패턴 등의 정보를 구축하는 기능을 한다.

또한 그림3에서 메소드에 대한 구조적 정보를 가지고 있는 메타 메소드 클래스인 MetaMethod는 MetaMessage 클래스로 구성되는데, 자신을 구성하는 일련의 메타 메시지들의 정보를 metaMessages 배열에 저장하고, generateMessageCode() 메소드로써 각각의 메시지들에 대한 코드를 생성하며, 이들 코드들을 메소드 몸체를 구성하는 코드들의 집합체인 messageCodes 배열에 저장하는 기능을 한다. 또한 MetaMethod 클래스의 generateMethodCode() 메소드는 자신을 구성하고 있는 메소드 몸체 코드를 생성하는 기능을 한다.

2.2. 메시지 패턴과 패턴정보 구축

[메시지 패턴 #1: 일반형]

메시지 패턴 #1은 가장 많이 나타나는 형태로서 소스 객체에서 타겟 객체로 메시지가 전달되는 패턴이다. 이 패턴은 소스 객체의 클래스에 정의되어 있는 메소드 내에서 타겟 객체의 클래스에 정의되어 있는 메소드를 호출하는 형태이다.

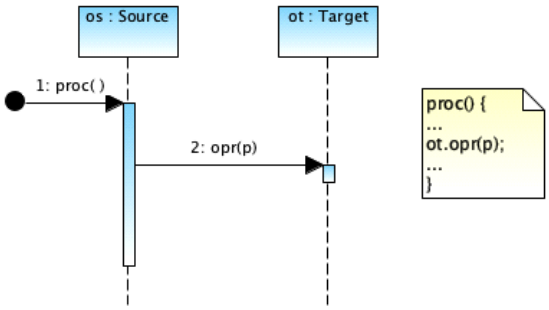

Fig. 4 Message pattern #1: General message pattern

그림4에서 보는 바와 같이 클래스 Source에 정의되어 있는 proc() 메소드의 내에서 클래스 Target에 정의되어 있는 opr(p) 메소드를 호출하는 형태이며, 메시지 패턴 #1의 경우에 상응하는 코드는 그림4의 우측에 보인 바와 같다. 메시지 패턴 #1에 대하여 코드 생성을 위해 구축되는 메타 정보를 보이면 그림5와 같다.

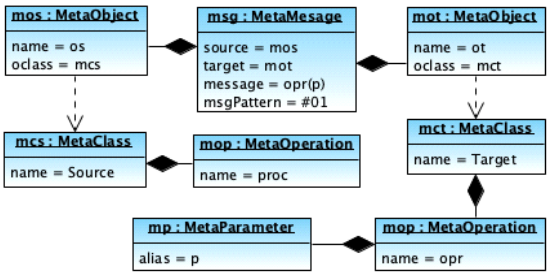

Fig. 5 Information structure of message pattern #1

그림5에서 보는 바와 같이 패턴 #1 형태의 메시지 msg:MetaMessage는 소스 객체 mos:MetaObject와 타겟 객체 mot:MetaObject에 대한 정보를 source와 target 속성에 가지고 있다. 또한 message 속성에 메시지의 오퍼레이션 정보 "opr(p)"를 가지고 있으며, msgPattern속성에 패턴 정보 "#01"을 가지고 있다. 소스 객체 mos:MetaObject는 객체명 "os"과 소속 클래스의 메타클래스명 "mcs"를 속성값으로 가지고 있다. 메타클래스 mcs:MetaClass는 소스 객체의 클래스명 "Source"를 가지고 있다. mcs:MetaClass의 구성 요소인 메타오퍼레이션 mop:MetaOperation는 오퍼레이션명 "proc"을 속성값으로 가지고 있다. mot:MetaObject는 속성값으로 타겟 객체명 "ot"와 소속 클래스의 메타클래스명 "mct"를 속성값으로 가지고 있다. mct;MetaClass는 타겟 객체의 클래스명 "Target"을 가지고 있으며, 구성 요소인 메타오퍼레이션 opt:MetaOperation은 속성값으로 오퍼레이션명 "opr"을 가지고 있으며, 메타 오퍼레이션의 구성 요소인 메타 파라미터 mpt:MetaParameter는 오퍼레이션의 인자 (argument) 명인 "p"를 alias 속성값으로 가지고 있다.

[메시지 패턴 #2: 생성형]

메시지 패턴 #2는 실행 중에 객체를 생성할 경우에 사용되는 메시지 패턴이다. 이 패턴은 소스 객체의 클래스에 정의되어 있는 메소드 내에서 타겟 객체의 클래스에 대한 생성자를 호출하는 메시지를 보내는 형태이다. 그림6에서 보는 바와 같이 클래스 Source에 정의되어 있는 proc() 메소드 내에서 클래스 Target의 생성자를 호출하는 형태이며, 메시지 패턴 #2의 경우에 상응하는 코드는 그림6의 우측에 보인 바와 같다.

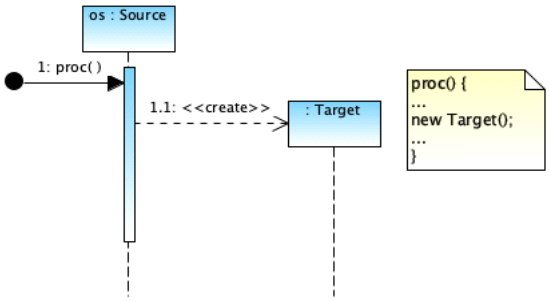

Fig. 6 Message pattern #2: Create message pattern

메시지 패턴 #2에 대하여 코드 생성을 위해 구축되는 메타 정보를 보이면 그림 7과 같다.그림 7에서 보는 바와 같이 패턴 #2 형태의 메시지 msg:MetaMessage는 소스 객체 mos:MetaObject와 타겟 객체 mot:MetaObject에 대한 정보를 가지고 있다. 또한 메시지의 오퍼레이션 정보인 "Target()"와 패턴 정보 "#02"를 가지고 있다. 소스 객체 mos:MetaObject는 객체명 "os"와 소속 클래스의 메타클래스명 "mcs"를 속성값으로 가지고 있다.

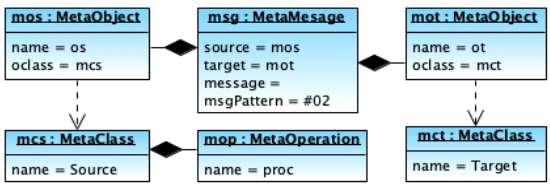

Fig. 7 Information structure of message pattern #2

그림7에서 메타 클래스 mcs:MetaClass는 소스 객체의 클래스명 "Source"를 가지고 있으며, 구성요소인 메타오퍼레이션 ops:MetaOperation은 오퍼레이션명인 "proc"을 가지고 있다. 타겟 객체 mot:MetaObject는 타겟 객체명 "ot"와 소속 클래스의 메타 클래스명 "mct"를 가지고 있으며, mct;MetaClass는 타겟 객체의 클래스명 "Target"을 가지고 있다. 생성자에 인자가 있을 경우에는 패턴#1에서 인자정보를 구축하는 것과 같은 방식으로 MetaOperation과 MetaParameter 클래스를 이용해서 인자 정보를 구축하여서 코드 생성을 한다. new 연산자의 처리는 코드 생성 시에 패턴 #2를 식별하여 "new"를 코드로 삽입하도록 설계하는 방법을 선택하였다.

[메시지 패턴 #3: 셀프형]

메시지 패턴 #3은 가장 많이 나타나는 형태로서 소스 객체에서 소스 객체로 메시지가 전달되는 패턴이다. 이

패턴은 소스 객체의 클래스에 정의되어 있는 메소드 내에서 소스 객체의 클래스에 정의되어 있는 다른 메소드를 호출하는 형태이다. 즉 소스 객체와 타겟 객체가 동일한 형태를 가진 패턴이다.

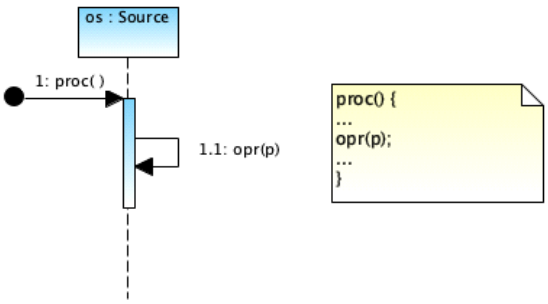

Fig. 8 Message pattern #3: Self message pattern

그림8에서 보는 바와 같이 클래스 Source에 정의되어 있는 proc() 메소드의 내에서 클래스 Source에 정의되어 있는 다른 메소드 opr(p)을 호출하는 형태이며, 메시지 패턴 #3의 경우에 상응하는 코드는 그림8의 우측에 보인 바와 같다. 메시지 패턴 #3에 대하여 코드 생성을 위해 구축되는 메타 정보를 보이면 그림9와 같다.

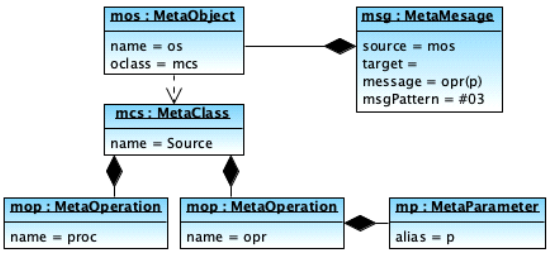

Fig. 9 Information structure of message pattern #3

그림9에서 보는 바와 같이 패턴 #3 형태의 메시지 msg:MetaMessage는 소스 객체와 타겟 객체에 대한 정보 source와 target 속성에 같은 mos:MetaObject에 대한 정보를 가지며, 메시지의 오퍼레이션 정보인 "opr(p)"와 패턴 정보 "#03"을 가지고 있다. 소스 객체 mos: MetaObject는 객체명 "os"와 소속 클래스의 메타 클래스명 "mcs"를 속성값으로 가지고 있다. 메타클래스 mcs: MetaClass는 클래스명 "Source"를 속성값으로 가지고 있으며, 이 클래스를 구성하고 있는 메타오퍼레이션 ops:MetaOperation은 오퍼레이션명 "proc"을 속성값으로 가지고 있다. 타겟 객체에 대한 mot:MetaObject클래스는 생성할 객체의 클래스명 "Target"을 속성값으로 가지고 있으며, 이 클래스를 구성하고 있는 메타오퍼레이션 opt:MetaOperation은 오퍼레이션명 "opr"을 속성값으로 가지고 있다. 이 오퍼레이션을 구성하고 있는 메타파라미터 mpt:MetaParameter는 alias 값으로 오퍼레이션의 인자 (argument) 명인 "p"를 가지고 있다.

Ⅲ. 메시지 패턴과 코드 생성

이 장에서는 시퀀스 다이어그램의 메시지 정보로부터 자동으로 코드를 생성하는 방법을 기술한다. 메시지로부터의 코드의 생성은 대상 프로그래밍 언어의 문법에 의존하여 생성되므로, 본 논문에서는 여러 연구에서 타겟 언어로 사용하고 있는 자바 언어를 대상으로 알고리즘을 제시한다. 또한 코드를 자동 생성하는 절차는 유한 상태 머신으로 알고리즘을 표현한다.

3.1. 메시지의 코드 생성

시퀀스 다이어그램에 표현되어 있는 정보는 자바 언어의 클래스 문법에 기초하여 코드가 생성된다. 클래스는 클래스의 이름과 속성들, 메소드들로 구성되며, 이러한 클래스에 대한 코딩의 기본 형태를 2-레벨 문법의 형태로 나타내면 다음과 같다.

public class c_name **{**
ATTRIBUTESc
METHODSc
}

여기서 밑줄로 강조된 부분을 키워드로서 터미널 (terminal)을 나타내며, c_name은 클래스 이름을 의미한다. ATTRIBUTESc은 넌터미널 (nonterminal)로서 클래스의 속성들을 나타내며, METHODSc는 넌터미널로서 클래스의 메소드들을 나타낸다. 이것을 유한 상태 머신 (Finite State Macine)으로 나타내면 다음의 그림10과 같다. 그림10에서 mc.visiblity와 mc.name, 상태는 각각 클래스의 가시성과 클래스의 이름을 가져와서 출력하는 상태이며, keyword 상태는 ""안에 있는 키워드를 출력하는 상태를 의미한다. <>로 표시한 상태는 넌터미널에 대응되며 하위 상태 다이어그램을 가짐을 의미한다. <mc.Attributes>는 클래스의 속성들을 생성하는 상태이며, <mc.Attributes> 상태에 대한 코드 생성은 [13]의 <<mc.Attributes>> 코드 생성 알고리즘을 사용한다.

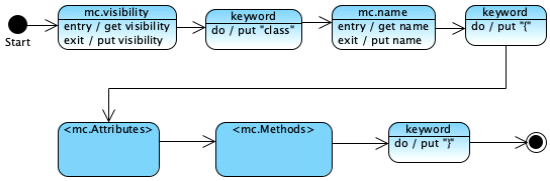

Fig. 10 FSM for “class” code generation

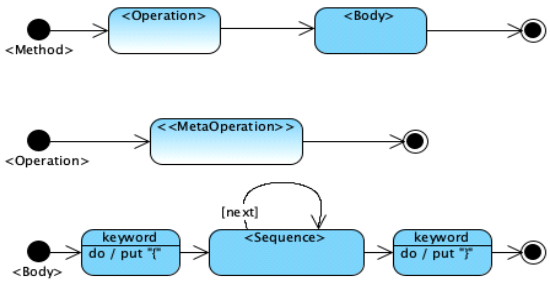

Fig. 11 FSM for “Method“ code generation

클래스의 메소드는 메소드 헤드 (오퍼레이션) 부분과 메소드 몸체로 구성된다. <mc.Methods> 상태는 메소드를 생성하는 상태이며, 절차는 그림11과 같다. 그림11에서 메소드는 <Operation>상태와 <Body> 상태로 전이되며, <Operation> 상태에 대한 코드 생성은 [13]의 <<MetaOperation>> 코드 생성 알고리즘을 사용한다. 메소드 몸체는 일련의 문장들의 시퀀스로 구성되므로 <Sequence> 상태에서 코드를 생성하게 되며, 자바 언어의 문법에 따라 "{ "와 "} "사이에 코드가 생성된다.

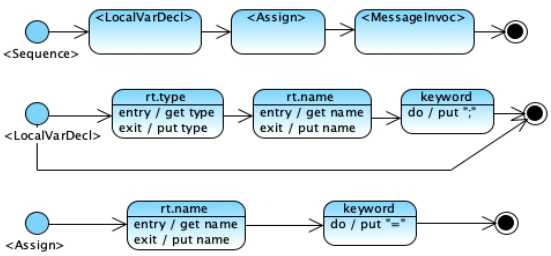

Fig. 12 FSM for “Sequence” code generation

시퀀스 다이어그램의 메시지의 시퀀스는 자바 언어의 문장들과 대응되므로 이들 시퀀스에 대한 코드 생성 절차를 그림12에서 보인다. 그림12에서 메소드의 한 시퀀스는 자바 언어의 한 문장과 대응되며, 변수를 선언하는 <LocalVarDecl> 상태 (예: 'Bank bk;')와 변수 지정하는 <Assign> 상태(예: 'bk =')와 메시지를 호출하는 <MessageInvoc> 상태 (예: 'card.proc()')로 나누어진다. 메시지를 호출하는 <MessageInvoc> 상태는 2장에서 기술한 메시지의 패턴에 따라 코드를 생성하는 방법이 달라진다.

3.2. 패턴에 따른 코드 생성

시퀀스 다이어그램에서의 메시지는 기본적으로 3가지 패턴으로 나누어지며, 이들 패턴에 근거하여 코드를 생성하는 절차를 기술하고자 한다. 이러한 코드 생성은 <MessageInvoc> 상태에서 이루어지며, 패턴에 따라 각각 다음의 해당하는 절차에 의해 코드가 생성된다. 메시지 패턴 정보는 그림3의 MetaMessage 클래스의 msgPattern 속성에 저장되어 있으므로, 이 정보에 기반하여 메시지의 각 패턴에 따른 <MessageInvoc> 상태가 호출되어 코드가 생성된다. 각 패턴 별로 <MessageInvoc> 상태의 코드 생성 절차를 기술하면 다음과 같다.

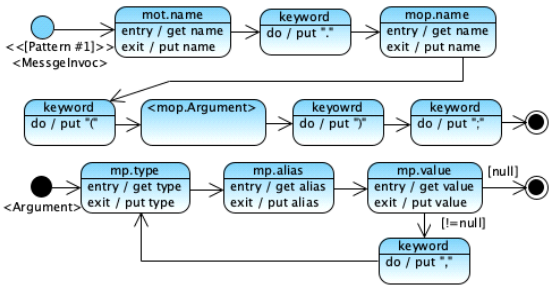

Fig. 13 FSM for pattern #1 code generation

[메시지 패턴 #1]

메시지 패턴 #1의 코드 생성 절차는 그림13에서 보인 바와 같다. 이 패턴은 가장 일반적인 메시지 패턴으로서, 지정연산자 '=' 다음에 메시지의 타겟 객체명과 타겟 객체의 메소드명이 표시되는 형태를 가지고 있다. 그림13에서 이 패턴은 먼저 <mot.name> 상태와 <keyword>/"." 상태, <mop.name> 상태에서, 타겟 객체의 이름과 "." 연산자, 타겟 객체의 메소드명 (오퍼레이션명)의 순서로 출력한다. 그리고 다음에 메소드의 인자 (argument)부분을 <mop.Argument> 상태에서 출력하고 <keyword>/";" 상태에서 ";"를 출력함으로써 종료된다. <mop.Argument>는 <Agrument> 상태로 처리되며, 인자의 테이터 타입과 인자명 그리고 인자값이 선언되어 있는 경우에는 값을 출력하고 종료된다. 인자가 없는 경우에는 바로 종료되며, 인자가 여러 개 있는 경우에는 인자가 있는 동안 반복해서 출력한다.

[메시지 패턴 #2]

메시지 패턴 #2의 코드생성 절차는 그림 14에서 보인 바와 같다. 이 패턴은 타겟 객체를 생성하는 메시지를 타겟 객체에 주는 형태이며, 'new' 연산자의 코드 생성을 수반한다. 그 다음에 인스턴스를 생성할 클래스의 생성자의 코드를 생성하게 되는 패턴이다. 따라서 그림14에서 보는 바와 같이 먼저, <keyword>/"new" 상태에서 "new" 연산자를 출력하고 <mot.name> 상태에서 클래스명을 가져와서 출력한다. 다음으로 생성자의 인자 부분은 그림13의 <Argument> 상태를 사용하여, 인자 부분의 유무에 따라 인자에 대한 코드를 "(" 와 ")" 사이에 출력하고 종료한다.

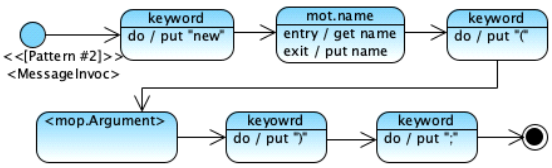

Fig. 14 FSM for pattern #2 code generation

[메시지 패턴 #3]

메시지 패턴 #3의 코드 생성 절차는 그림15에 보인 바와 같다. 이 패턴은 소스 객체에서 자신의 클래스 내에 있는 메소드를 호출하는 셀프 호출 형태를 보이는 메시지 패턴이다. 그림15에서 <mop.name> 상태에서 메소드명을 가져와서 출력하며, 인자 부분은 "("를 출력한 후에 그림13의 <Argument> 상태를 사용하여 처리하고 종료한다.

이와 같은 세 가지의 패턴은 메시지의 기본 패턴이며, 메시지의 시퀀스는 이들 패턴들이 조합되어 나타난다. 또한 이러한 패턴들을 기본으로 하여, 메시지들 중에서 분기 (branch/if)와 부분 반복 (loop)이 적용되기도 한다. 이러한 분기와 반복에 대한 문제를 다음 장에서 기술한다.

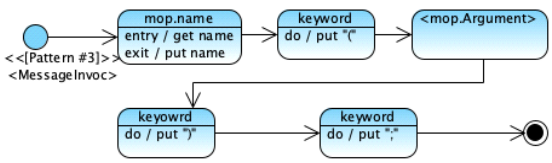

Fig. 15 FSM for pattern #3 code generation

Ⅳ. 분기와 루프의 처리

본 논문에서는 분기와 반복의 처리를 기존의 연구에서와는 달리 기본 패턴에 대한 처리와 통합되게 코드를 생성할 수 있는 시스템을 구성한다. 통합된 코드생성을 함으로써 메시시의 패턴에 기반한 코드 생성 알고리즘을 기본 패턴의 알고리즘을 일관되게 적용할 수 있게 한다. 이를 위하여 분기와 반복에 대해서도 패턴으로 분류하고, 각 패턴에 대해서 메타 정보를 구축한다. 구축된 메타 정보에 기반해서 코드 생성을 하는 알고리즘을 설계한다.

시퀀스 다이어그램에서는 메시지들 사이에 조건에 의한 메시지의 실행과 일부 메시지들의 반복 수행을 나타내기도 한다. 시퀀스 다이어그램으로부터의 이러한 분기 또는 반복에 대한 정보는 그림3의 MetaMessage 클래스의 msgCond 속성에 저장하고, 이들 근거로 하여 분기와 반복에 대한 코드를 생성한다. 이 장에서는 이러한 분기와 반복에 대한 코드 처리에 대한 방법을 기술한다.

4.1. 분기의 코드 생성

메시지에서의 분기는 메시지가 가지고 있는 조건의 만족 여부에 따라 의해 메시지가 선택적으로 수행됨을 의미한다. 그림16에서 보는 바와 같이 분기는 가드(guard)인 [cond]의 조건에 따라 1: ot.opr1( ) 메시지가 수행되든지, 2: ot.opr2( )가 수행되는 형태를 보여준다. 분기는 전형적으로 if문을 사용하여 다음과 같은 코드를 생성하게 된다.

```
if (cond) {
   ot.opr1( );
} else {
   ot.opr2( );
}
```

분기 메시지인 경우에는 MetaMessgae 클래스의 msgCond 속성에 [cond]에 해당하는 값을 가지고 있고, null값을 가지면 분기가 아님을 의미한다. 그림 16의 분기에 대한 처리는 그림17에 보인 절차에 따라 코드를 생성하게 된다.

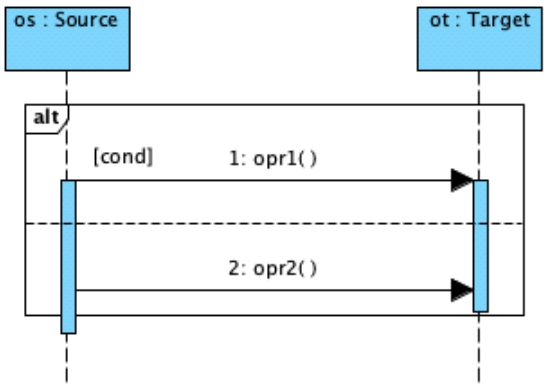

Fig. 16 Braching in sequence diagram

그림17에서 먼저 분기를 나타내는 <keyword>/"if" 상태에서 "if"를 출력하고 <cond> 상태에서 [cond] 부분을 "("와 ")" 사이에 출력한다. 그리고 "{"와 "}" 사이에 <Sequence> 상태에 의해 [cond]를 만족할 때 수행할 메시지를 출력한다. 그리고 [cond]를 만족하지 않을 때 수행할 메시지를 위하여 <keyword>/"else" 상태에서 "else"를 출력한다. 그 후에 "{"와 "}" 사이에 <Sequence> 상태에 의해 [cond]를 만족하지 않을 때 수행할 코드들을 출력하고 종료하게 된다.

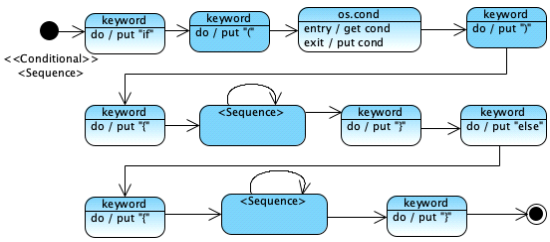

Fig. 17 FSM for Conditional sequence

4.2. 반복의 코드 생성

시퀀스 다이어그램에서 메시지에서의 반복은 그림18에서 보는 바와 같이 가드인 [cond] 조건을 만족하는 동안 일련의 메시지를 반복 수행하는 것을 의미한다. 코드 생성에서는 반복문의 표현과 반복문의 블락 부분의 메시지를 코드로 생성하게 된다. 반복은 통상 for문이나 while문으로 표현되는 데, 여기서는 while문으로 코드를 생성하는 예를 보이면 다음과 같다.

```
while (cond) {
    ot.opr1( );
    ot.opr2( );
}
```

반복은 MetaMessage 클래스의 msgLoop 속성에 [cond]값을 가질 때 식별되며, null 값을 가지는 경우에는 반복이 아님을 의미한다. 그림18의 반복에 대한 처리는 그림19에 보인 절차에 따라 코드를 생성하게 된다. <keyword>/loop에서 “while”을 출력하고 <cond> 상태에서 [cond] 부분을 "("와 ")" 사이에 출력한다. "{"와 "}" 사이에 <Sequence> 상태에서 반복될 코드들을 출력하고 종료한다.

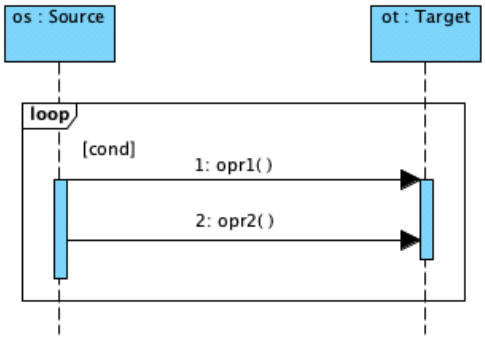

Fig. 18 Looping in sequence diagram

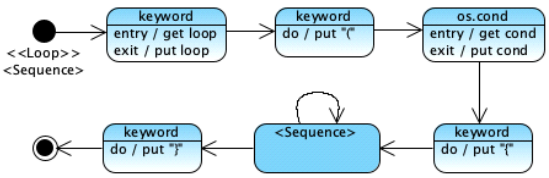

Fig. 19 FSM for Loop sequence

Ⅴ. 결 론

본 논문에서는 시퀀스 다이어그램으로부터의 메시지 정보를 해석하여 메시지 패턴에 기반하여 코드를 생성하는 방법을 제시하였다. 먼저, 코드를 생성하기 위하여 메시지의 정보와 이에 대응되는 메소드의 정보를 구축하는 방법을 제시하였다. 시퀀스 다이어그램으로부터 메시지에 대한 객체들의 클래스와 메소드의 구조적 메타 정보를 해석하여 메타클래스 정보와 메타메시지 정보를 구축하였다. 이들 메타 클래스 정보와 메타 메소드 정보를 이용하여 해당 메시지의 상응하는 메소드의 코

드를 생성하였다.

체계적 코드 생성을 위하여 시퀀스 다이어그램의 메시지들의 원소적 유형을 패턴으로 분류하고, 이들 각각의 패턴에 상응하는 구조적 메타 정보를 구축하였다. 시퀀스 다이어그램의 메시지들의 패턴 종류를 식별하고, 해당 패턴의 구축된 메타정보를 사용하여 코드를 생성하는 방법을 제시하였다. 또한, 시퀀스 다이어그램에서의 분기와 반복에 대한 처리를 기존의 연구와는 달리 기본 패턴을 처리하는 방법과 통합하여 처리하는 방법을 제시함으로써 전체적으로 통합된 코드 생성 알고리즘을 제시하였다. 향후 UML에서 지원하는 시퀀스 다이어그램의 다양한 옵션들을 추가로 지원하기 위하여 기능을 확장하는 연구가 필요하다. 본 논문에서 제시하는 시퀀스 다이어그램에 대한 정보 구축과 코드 생성 알고리즘은 상업용이나 교육용 UML 다이어그램 도구의 개발에 용이하게 적용하여 활용할 수 있을 것으로 기대한다.

ACKNOWLEDGEMENT

This work was supported by a Research Grant of Andong National University.

References

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

[2] P. Pawde, 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. [3] A. Soumiya, 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] Y. Kim, "Information structuring of diagram repository for UML diagrams," *Journal of the Korea Institute of Information and Communication Engineering*, Vol. 23, No. 12, 2019.

[5] 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.

[6] G. Engels, R. Hucking, S. Sauer, and A. Wagner, "UML collaboration diagrams and their transforamtion to Java," *Second International Conference on the UML*, pp. 437-488, 1999.

[7] G. Engels, R. Heekel, J. Kuster, "Reul-based Specification of behavioral consistency based on the UML meta-model," *Fourth International Conference on the UML*,, pp. 272-286, 2001.

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

[9] 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.

[10] 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.

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

[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, "A design of constructing diagram repository for UML diagram tools," *Journal of the Korea Institute of Information and Communication Engineering*, Vol. 24, No. 2, 2020.

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