JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 23, No. 12: 1588~1595, Dec. 2019

UML 다이어그램을 위한 다이어그램 레포지토리의 정보구조화

김윤호*

Information Structuring of Diagram Repository for UML Diagrams

Yun-Ho Kim*

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

요 약

본 논문에서는 UML 다이어그램에 대응되는 코드 생성을 위하여 요구되는 정보를 저장하기 위한 레포지토리를 구축하는 방법을 제시한다. 객체 지향 프로그래밍에서는 객체 간의 상호 작용이 핵심이므로, UML 다이어그램들 중에서 클래스 다이어그램과 시퀀스 다이어그램에 대해서 논의를 전개한다. 클래스 다어어그램을 기반으로 실행 시간에 객체가 상호 동작하는 절차를 보이는 시퀀스 다이어그램으로부터 상응하는 코드를 생성하게 되며, 이를 위해서는 코드생성에 필요한 정보를 추출하여 정보 저장소인 레포지토리를 구축하여야 한다. 따라서 본 논문에서는 시퀀스 다이어그램의 메시지 유형을 다섯 개로 분류하고 이들로부터 각각의 레포지토리를 구성하기 위하여 필요한 항목과 그 값에 대한 정보를 추출하여 구조적으로 정보를 저장하는 방법을 제시한다. 시퀀스 다이어그램은 이들 메시지들로 구성되므로, 각 메시지에 대한 구조화된 레포지토리를 순차적으로 수집하여 최종적인 레포지토리를 구성한다.

ABSTRACT

This paper presents the technique on structuring information of the diagram repository for UML diagrams. Because object interactions are the body of object-oriented programming, this paper handles especially the sequence diagrams and class diagrams among UML diagrams. Based on class diagrams, sequence diagrams represent the procedure of object interactions in run-time and then the corresponding codes are generated from the contents of those sequence diagrams. To do this work, this paper presents a method to construct the information repository for generating code from the contents of sequence diagrams. This paper classifies the five message types of sequence diagrams and then extracts the needed information including items and values on the corresponding message types for constructing message repositories. Because sequence diagram is composed of messages included, the final repository is constructed by collecting each of structured repositories on messages sequentially.

키워드 : UML 다이어그램, 정보구조화, 다이어그램 변환, 자동 코드 생성, 객체 지향 시스템

Keywords : UML diagrams, information structuring, diagram transformation, automatic code generation, object-oriented system

Received 21 October 2019, **Revised** 7 November 2019, **Accepted** 11 November 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.2019.23.12.1588 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

엔터프라이즈 애플리케이션 개발에서부터 스마트 폰을 위한 모바일 앱 개발에 이르기까지 현재의 소프트웨어 시스템의 규모와 복잡성 그리고 급속 개발의 요구는 지속적으로 화두가 되어 왔다. 이를 해결하기 위한 객체 지향 프로그래밍으로의 패러다임 이동은 이제는 소프트웨어 개발 커뮤너티의 일반적 추세를 확보하고 있다. 객체 지향 소프트웨어 개발에서는 개발 프로세서의 수행과 개발자 간의 의사소통을 위하여 UML [1]을 사용하여 분석 및 설계를 수행하고 문서화하고 있다.

UML은 객체 지향 시스템을 정적인 관점과 동적인 관점에서 표현을 할 수 있는 시각적 모델링 언어이다. 정적인 관점을 제시하는 클래스 다이어그램은 소프트웨어 시스템을 구성하는 클래스들에 대한 정의와 이들 간의 상호관계성을 표현한다. 또한, 동적인 관점을 제시하는 상호 작용 다이어그램은 실행 시에 객체들 간의 협력 관계 또는 수행하는 책임 (역할)의 이행 절차를 나타내며, 시퀀스 다이어그램이 대표적이다.

한편, UML을 이용한 분석 및 설계를 위한 UML CASE 도구들이 소규모에서부터 대규모의 소프트웨어 개발을 위해 필수적으로 사용되어 왔다 [2,3]. 이러한 도구들은 UML을 이용한 모델링 뿐만 아니라 코드 자동 생성의 기능까지 제공하고 있으며, 디자인 모드에서 가시적으로 다이어그램을 작성하고 이에 대응되는 코드로부터 자동 코드 생성이 양방향으로 동작하는 기능도 필수적인 기능으로 받아들여지고 있다 [4-6]. 이러한 도구를 개발하기 위해서는 다이어그램의 작성과 코드 생성을 위한 막대한 정보를 저장하는 레포지토리가 필요함은 객체 지향 설계 초창기부터 제기되어 왔으며 , 이들 정보들을 어떻게 구조화하여 구축하는 것이 이슈가 된다 [7-9]. 따라서, 본 논문에서는 UML 다이어그램과 대응되는 코드 생성을 위한 레포지토리에 저장할 필요 정보를 추출하여 구조적으로 레포지토리가 구축될 수 있도록 하는 방법을 제시하고자 한다.

Ⅱ. UML 다이어그램의 정보구조화

객체 지향 프로그래밍에서는 객체 상호간의 작용을 프로그래밍 하는 것이 핵심을 이루며, UML 다이어그램에 대하여 그에 대응되는 코드를 생성하기 위해서는 다이어그램이 내포하고 있는 의미에 대한 다양한 정보를 구조적으로 저장하는 '레포지토리'를 어떻게 구축하는가가 중요하다. 따라서, 본 논문에서는 UML의 시퀀스 다이어그램과 클래스 다이어그램에 대한 레포지토리 구축을 위한 이들 다이어그램 정보의 구조화를 제시한다.

2.1. UML 다이어그램 정보와 대응 코드

본 논문에서는 상호 작용 다이어그램이 포함하고 있는 정보를 추출하고 구조적으로 정보를 저장하는 방법을 제시하기 위하여, 시퀀스 다이어그램의 전형적인 메시지 형태를 제시한 사례 [10,11]를 확장하여 논의를 전개하고자 한다. 그림 1에서는 사용자가 도서관에서 책을 대출하는 문제에 대한 클래스 다이어그램을 보여 준다. 이 그림에서 Library 클래스는 User 클래스와 연관관계가 있으며, 대출을 요청한 도서가 재고로 가지고 있으면 사용자와 대출 요청 도서를 요청 대기 목록에서 제거하는 removeBook() 메서드를 정의하고 있다. 또한 대출을 요청하는 사용자에 대한 정보(users)와 대출 요청 도서에 대한 정보(books)를 가지고 있다. User 클래스는 도서 대출을 원하는 사용자를 정의하며, Book 클래스와 연관 관계가 있다. 사용자 usr 객체에 특정 유저 uid가 대출할 특정 도서에 대한 도서 식별 번호 bkid를 통해 보유 여부를 검사하는 checkIn(uid,bkid) 메시지가 정의되어 있다. Book 클래스는 보유 도서를 정의하는 클래스이며, 특정 유저가 대출하려는 도서가 있는지를 확인하고 보유 도서를 반환하는 getBook() 메서드가 정의되어 있다.

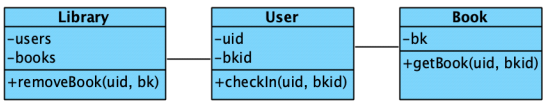

Fig. 1 Class Diagram of An Library Example

그림2에서는 그림1에 보인 클래스의 객체들에 대하여 도서 대출을 위한 절차를 시퀀스 다이어그램으로 표현한 것이다 . 여기서는 세 개의 객체 (Library 클래스의 lib 객체, User 클래스의 usr 객체, Book 클래스의 bk 객체)가 서로 메시지를 주고받으며 대출 절차를 이루고 있다. 즉, uid의 사용자가 bkid의 책을 대출하고자 하는 경우, lib 객체에서 usr 객체에 checkId(uid,bkid) 메시지를 usr 객체에 보내면 bk 객체에서 해당하는 책이 있는가를

검사한다. 그 결과 요청한 책이 있으면 대출을 요청한 사용자에게 할당하고, lib 객체에서 그 책을 재고 목록에서 제거함으로써 대출 과정을 종료하게 된다.

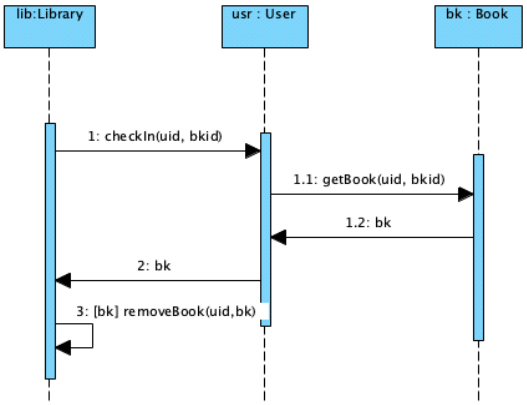

Fig. 2 Sequence Diagram of An Library Example

그림1에서 보인 시퀀스 다이어그램이 의미하고 있는 정보와 대응되는 코드의 부분을 보이면 다음과 같다.

```
class Library {
  User usr;
  Book bk;
  public foo(uid,bkid) {
    bk = usr.checkIn(uid,bkid);
    if (bk) { removeBook(uid,bk);
  }
}
class User {
  public Book checkIn(uid,bkid) {
    return getBook(uid,bkid);
  }
}
class Book {
  public Book getBook(uid,bkid) {
    return bk;
  }
}
```

UML 시퀀스 다이어그램으로부터 코드를 생성하려면, 다이어그램이 내포하고 있는 의미에 대한 정보를 추출하고 이를 레퍼지토리에 저장하여 코드 생성시에 그 근거로 삼아야 한다. 상호 작용 다이어그램은 클래스 다이어그램의 정보를 기반으로 하여 도식된다 [12]. 본 논문에서는 구조적인 정보 구축을 위하여 시퀀스 다이어그램의 정보를 특정한 유형으로 분류하여 정보를 구조화시키는 방법을 제시한다.

2.2. 상호작용 다이어그램의 세부 유형

Sangal 등 [11]은 메시지의 유형을 일반적인 메시지 형태와 조건을 포함하는 메시지의 두 가지에 대하여 논의를 전개하였다. 시퀀스 다이어그램으로부터 자동 코드 생성을 위하여 필요한 정보의 추출을 위해서는 이러한 분류만으로써는 충분하지 못하다. 따라서, 본 논문에서는 시퀀스 다이어그램의 메시지 형태를 5가지 유형으로 세분화하여, 시퀀스 다이어그램이 내포하고 있는 정보를 보다 상세하게 추출하는 방법을 다음과 같이 제시한다.

첫 번째로, [유형 1]은 두 객체 사이에 단순한 메시지를 주고받는 경우이다. 그림 3에서 보는 바와 같이 클래스 A의 a 객체와 클래스 B의 b 객체 사이에 m() 메시지가 보내지는 형태이다. 그림2에서는 메시지 1, 메시지 1.1이 유형 1에 해당된다.

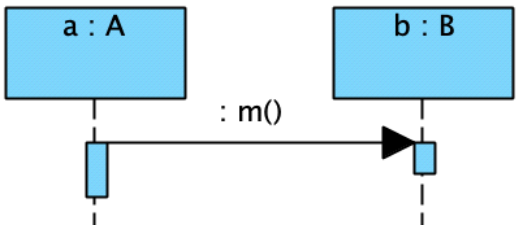

Fig. 3 Type 1 of Sequence Diagram

유형 1은 시퀀스 다이어그램의 정보를 해석하여 다음과 같은 형태의 코드로 생성되는 유형이다.

```
class A {
  ...
  public foo() {
    b.m();
  }
...
}
```

여기서 b 객체의 m()메서드가 호출된다는 정보는 도식 정보이며, A 클래스 내부의 임의의 foo() 메서드 내에서 b.m(); 으로 코딩이 되는 것은 생성 정보에 해당한다. 이 두 정보에 의거해서 위와 같이 코드 생성이 이루어질 수 있다. 따라서 이 유형의 코드 생성을 위해서는 소스와 타겟에 대한 객체명, 클래스명과 메서드명, 해당 메서드의 파라미터 리스트에 대한 정보가 다이어그램 레퍼지토리에 구조화되어 구성된다 (표 1).

Table. 1 Diagram Repository Structure for Type 1

Source Object Name	Source Class Name	Target Object Name
Target Class Name	Method Name	Method Parameter List

두 번째로, [유형 2]는 한 객체가 자기 자신과 메시지를 주고 받는 경우이다. 그림 4에서 보는 바와 같이 클래스 A의 객체 a가 자신의 클래스 내부에 있는 메서드를 호출하는 형태이다. 그림 2에서는 메시지 3이 유형 2에 해당된다.

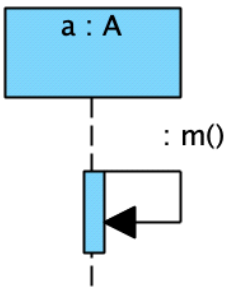

Fig. 4 Type 2 of Sequence Diagram

유형 2에 대해서는 시퀀스 다이어그램의 정보를 해석하여 다음과 같은 코드 형태로 생성되는 유형이다.

```
class A {
  ......
  public foo() {
    m();
  }
  ......
}
```

위의 코드에서 보는 바와 같이 이 메시지는 셀프 메시지라는 도식 정보와 클래스 A의 내부의 임의의 foo() 메서드 내에서 객체 지정없이 호출되는 메서드 m(); 형태로 코딩이 된다는 생성 정보에 따라서 코드 생성이 이루어질 수 있다. 따라서 이러한 유형의 코드 생성을 위해서는 객체명, 클래스명, 메서드명, 메서드의 파라미터 리스트에 대한 정보가 다이어그램 레퍼지토리에 구조화되어 저장되어야 한다 (표2).

Table. 2 Diagram Repository Structure for Type 2

Object Name	Class Name	Method Name	Method Parameter List

세 번째로, [유형 3]은 조건에 따라 수행되거나 수행되지 않는 경우이다. 그림 5에서 보는 바와 같이 UML에서 guard로 표현되는 조건이 만족되면 수행되고 만족되지 않으면 실행하지 않는 메시지이다. 즉, [cond]가 guard 즉 조건에 해당되며 boolean 값으로서 참이면 메시지가 수행되고 false이면 수행되지 않는다. 그림 2의 메시지 3에서는 [bk]를 가지는 removeBook()메시지이가 유형 3에 해당된다.

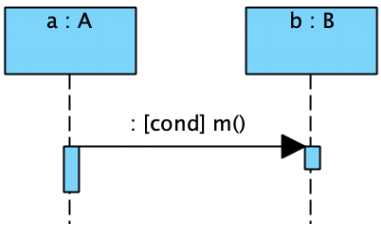

Fig. 5 Type 3 of Sequence Diagram

유형 3은 시퀀스 다이어그램의 정보를 해석하여 다음과 같이 코드 생성이 이루어지는 형태이다.

```
class A {
  ...
  public foo() {
    if (cond) { b.m1(); }
  }
...
}
```

유형 3은 이 유형에 해당하는 시퀀스 다이어그램의 도식 정보와 Guard인 [cond]의 값이 true일 때 메시지 m()이 수행되므로 if 문으로 코딩이 이루어지는 형태라

는 생성 정보를 이용하여 위와 같은 코드가 생성된다. 따라서 이 유형에서는 소스와 타겟에 해당하는 객체명, 클래스명과 Guard 값에 대한 정보, 메서드와 해당 메서드의 파라미터 리스트에 대한 정보가 다이어그램 레퍼지토리에 구조화되어 구성된다 (표3).

Table. 3 Diagram Repository Structure for Type 3

Source Object Name	Source Class Name	Target Object Name	
Target Class Name	Guard	Method Name	Method Parameter List

다음으로, [유형 4]는 하나 또는 그 이상의 메시지가 반복적으로 수행되는 경우이다. 그림 6에서 보는 바와 같이 클래스 A의 a 객체와 클래스 B의 b 객체사이에 m1(), m2() 메시지가 호출되며, 이는 순서적으로 수행된다. 또한, 이 두 메서드는 특정 조건이 맞는 동안에 반복 수행됨을 나타낸다. 메서드의 수는 하나이상 가능하며 이 그림에서는 두 개의 메서드가 loop 내에 포함됨을 예시한다.

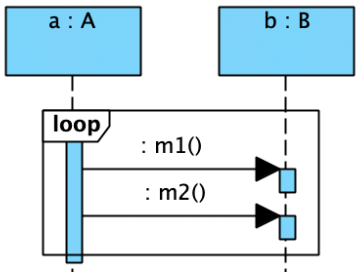

Fig. 6 Type 4 of Sequence Diagram

유형 4는 시퀀스 다이어그램의 정보를 해석하여 다음과 같은 코드로 생성되는 유형이다.

```
class A {
  ......
  public foo() {
    while (cond) {
      b.m1();
      b.m2();
    }
    ......
  }
```

위의 코드에서 loop에는 반복의 조건이 표시되고 특정 조건식이 참인 동안에 반복된다. 반복은 조건이 주어지는 형식에 따라 while 문이나 for 문으로 코드가 생성된다. 이러한 도식 정보와 생성정보에 의거하여 다음과 같은 형태로 코드가 생성되는 유형이다. 따라서, 이 유형의 코드 생성을 위해 필요한 정보는 소스 및 타겟 객체명과 클래스명, 반복 블록 여부, 반복 블록 내의 메서드와 파라미터 리스트에 대한 정보, 반복 조건에 대한 정보들이 다이어그램 레퍼지토리에 구성된다 (표4).

Table. 4 Diagram Repository Structure for Type 4

Source Object Name	Source Class Name	Target Object Name
Target Class Name	Method Name[]	Method Parameter List[]
Loop	Condition	Block

마지막으로 [유형 5]는 그림 7에서 보는 바와 같이 메서드 리턴을 나타낸다. 리턴 값 obj는 클래스 B의 임의의 메서드 내에서 반환되는 값이며, foo 메서드를 호출한 객체 b에서 그 값을 이용하는 형태이다. 그림 2에서는 메시지 1.2와 메시지 2가 유형 5에 해당된다.

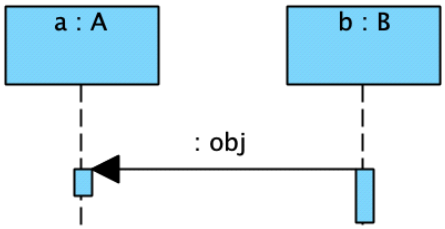

Fig. 7 Type 5 of Sequence Diagram

아래와 같이 코드로 생성되는 유형이며, 표 5에서 보이는 바와 같은 정보가 레포지토리에 저장된다.

```
class B {
  type_bk foo() {
    return bk;
  }
}
```

Table. 5 Diagram Repository Structure for Type 5

Source Object Name	Source Class Name	Target Object Name
Target Class Name	Method Name	Return

Ⅲ. 정보 레포지토리의 구성

이 장에서는 시퀀스 다이어그램을 해석하여 코드 생성을 위해 필요한 정보를 구축하기 위하여, 시퀀스 다이어그램 내의 메시지 유형에 따라 요구되는 정보를 레포지토리로 저장하고 수집된 개별적인 레포지토리를 종합하여 최종적인 레포지토리를 구조화하는 방법을 제시하고자 한다.

3.1. 유형별 레포지토리 정보의 구성

레포지토리에 저장되는 정보는 레포지토리 항목과 그 값으로 구성된다. 이를 위하여 Java의 HashMap 자료구조를 사용하여 정보를 구조화하여 구축한다. 즉, HashMap *H*는 다음과 같이 항목 item이 key 값과 value 값의 쌍으로 이루어지는 집합으로 구성된다.

$$H = \{\text{item} \mid (\text{key}, \text{value})\}$$

레포지토리의 항목이 key값으로 항목의 해당 값이 value 값으로 지정된다.

해시 맵 자료구조를 이용하여 [유형 1]에서 제시된 레포지토리 정보 (표1)은 다음과 같이 해시 맵 *Repo1*으로 구성된다.

Repo1 = {(Source Object Name, a), (Source Class Name, A), (Target Object Name, b), (Target Class Name, B), (Method Name, m), (Method Parameter List, P)}, where P = {parameter | (Data Type, varible)}.

파라미터 리스트는 널 (null)이거나 하나 이상의 파라미터로 구성된다. 파라미터는 변수와 그 변수의 데이터 타입으로 구성된다. 따라서, 파라미터 리스트 *P* 역시 해시 맵으로 구현하며, *P*는 데이터 타입과 변수명의 쌍을 요소로 하는 집합으로 이루어진다.

다음으로, [유형 2]에서 제시된 레포지토리 정보 (표2)는 다음과 같이 해시 맵 *Repo2*로 구성된다.

Repo2 = {(Source Object Name, a), (Source Class Name, A), (Target Object Name, a), (Target Class Name, A), (Method Name, m), (Method Parameter List, P)}, where P = {parameter | (Data Type, varible)}.

이 유형에서는 재귀적 메시지이므로 소스와 타겟 객체 또는 클래스가 동일하다. 따라서, 해시 맵 구성에서는 소스와 타겟이 같은 항목과 값을 가지게 된다.

[유형 3] 역시 표3에 제시된 레포지토리 정보에 의거하여 해시 맵 *Repo3*으로 구성된다.

Repo3 = {(Guard, cond), (Source Object Name, a), (Source Class Name, A), (Target Object Name, b), (Target Class Name, B), (Method Name, m), (Method Parameter List, *P*)}, where *P* = {parameter | (Data Type, varible)}, where Guard = **true**.

이 유형에서는 Guard인 [cond]의 값이 **true**일 경우에 나머지 정보가 유효하고, **false**인 경우에는 유효하지 않다. 여기서 유효성은 코드 생성의 여부와 관련되어 정의되는 것으로서 유효하지 않으면 코드를 생성하지 않음을 의미한다.

[유형 4]에서는 해당 레포지토리 정보 (표4)에 의거하여 다음과 같이 해시 맵 *Repo4*로 구성된다.

Repo4 = { (Loop, “Boolean”), (Condition, “expression”), (Source Object Name, a), (Source Class Name, A), (Target Object Name, b), (Target Class Name, B), (Method Name, m), (Method Parameter List, *P*)}, where *P* = {parameter | (Data Type, varible)}, where Loop = true, repeat on Condition.

[유형 5]에서는 표5에 제시된 해당 레포지토리 정보에 의거하여 다음과 같이 해시 맵 *Repo5*로 구성된다.

Repo5 = {(Source Object Name, a), (Source Class Name, A), (Target Object Name, b), (Target Class Name, B), (Method Name, foo), (Return, *R*)}, where *R* = {(Data Type, “Object”) or *e*}.

여기서, 리턴 값 R은 반환되는 객체 “Object”와 그 객체의 타입으로 표현된다. 또한 *e*는 리턴 값이 없는 void에 해당한다.

3.2. 시퀀스 다이어그램 유형의 종합적 적용

이 절에서는 지금까지 기술한 시퀀스 다이어그램의 유형과 해당 레포지토리 구축을 2.1절에 기술한 예에 대하여 적용한 결과를 기술한다. 하나의 시퀀스 다이어그

램에 대한 레포지토리 Ω_s는 각 메시지에 대한 레포지토리 R_i의 컬렉션으로 이루어진다.

$$\Omega_s = \Sigma_{i=0}^{n} R_i$$

다이어그램 레포지토리를 구성하는 절차는 그림 8에 보인 바와 같다. 시퀀스 다이어그램의 메시지들에 대한 정보를 입력받아 메시지 타입을 결정한다. 부분적인 레포지토리는 타입 항목과 그 항목에 대한 값들을 수집하여 구성되며, 각 메시지들에 대한 부분 레포지토리들을 종합하여 최종적인 레포지토리를 구성하게 된다.

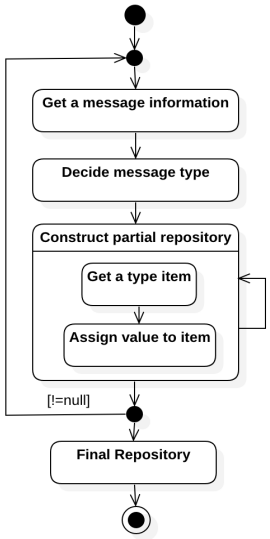

Fig. 8 Procedure for Constructing Diagram Repository

그림2에서의 시퀀스 다이어그램의 메시지는 5개로 이루어져 있으며, 메시지 1과 메시지 1.1은 유형 1에 해당한다. 메시지 1.2와 메시지 2는 유형 5에 해당하며, 메시지 3은 유형 3에 해당한다. 그림 8에 보인 바와 같이 최종적인 레포지토리는 각 유형에 따른 레포지토리를 적용한 결과의 레포지토리들로 이루어진다. 즉, 최종적 레포지토리 Ω_s는 시퀀스 다이어그램 내의 각 메시지 S_i에 대하여 구축되는 레포지토리 R_i로 구성된다. 그림 2의 시퀀스 다이어그램은 5개의 메시지를 가지고 있으며, 각 메시지에 대하여 구성되는 레포지토리를 종합한 최종적인 레포지토리 Ω_s는 다음과 같다 (그림 9참조).

Ω_s = { R_1, R_2, R_3, R_4, R_5 }, where

$R1$ = {(Source Object Name, lib), (Source Class Name, Library), (Target Object Name, usr), (Target Class Name, User), (Method Name, checkId), (Method Parameter List, ((UserId, uid), (BookID, bkid)))},

R_2 = {(Source Object Name, lib), (Source Class Name, Library), (Target Object Name, usr), (Target Class Name, User), (Method Name, getBook), (Method Parameter List, ((UserId, uid), (BookID, bkid)))},

R_3 = {(Source Object Name, bk), (Source Class Name, Book), (Target Object Name, usr), (Target Class Name, User), (Method Name, getBook), (Return, (Book,bk))},

R_4= {(Source Object Name, usr), (Source Class Name, User), (Target Object Name, lib), (Target Class Name, Library), (Method Name, checkId), (Return, (Book,bk))},

R_5= {(Source Object Name, lib), (Source Class Name, Library), (Target Object Name, lib), (Target Class Name, Library), (Method Parameter List, ((UserId, uid), (BookID, bkid)))}.

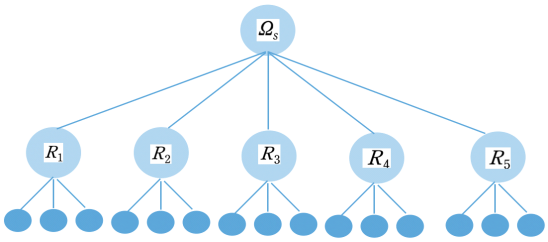

Fig. 9 Information Structure of Diagram Repository

Ⅳ. 결 론

본 논문에서는 클래스 다이어그램에 기반하여 시퀀스 다이어그램이 내포하고 있는 정보를 추출하여 코드 생성을 위한 정보 저장소인 레포지토리를 구축하기 위하여 정보 구조를 체계적으로 형식화하는 방법을 제시하였다. 이를 위하여 시퀀스 다이어그램의 메시지 유형을 다섯 가지로 분류하고, 메시지의 각 유형에 따라 코드 생성에 필요한 정보를 추출하고 수집함으로써 레포지토리의 정보를 구조화하였다. 시퀀스 다이어그램의 각 메시지로부터 구축된 개별 레포지토리를 수집하여 최종적인 레포지토리의 구조화된 정보를 구축하게 된다. 다음 단계로서 이러한 레포지토리로부터 코드를 생성하기 위하여는 다이어그램의 작성시에 구성되는 정

보와 UML 파서 그리고 정보 레포지토리 사이의 종합적이고 체계적인 통합이 필요하다. 향후 본 논문에서 제시된 레포지토리에 기반하여 코드 자동 생성을 구현할 수 있도록, 정보 레포지토리와 코드 생성을 연결하는 메타정보를 구축하기 위한 메커니즘의 구현 문제를 해결하고자 한다.

ACKNOWLEDGEMENT

This work was supported by a grant from 2015 Research Funds of Andong National University.

References

[1] J. Rumbough, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manual*, Addson-Wesley, 1999.

[2] D. Orleans and K. Lieberherr, "DJ: Dynamic Adaptive Programming in Java," *International conference on Meatalevel Architectures and Reflection*, pp. 73-80, 2001.

[3] T. Quatrani, *Visual Modeling with Rational Rose 2002 and the UML*, 3rd. ed., Addison-Wesley, 2002.

[4] F. Ciccozzi, "Execution of UML models: a system review of research and practice," *Software & Systems Modeling*, Vol. 18, No. 3, pp. 2313-2360, 2019.

[5] S. Gotti, S. Mbarki, "UML executable: a comparative study of UML compilers and interpreters." *International Conference on Information Technology for Organizations Development (IT4OD)*, pp. 1-5, 2016.

[6] A. Soumiya, B. Mohamed, "Converting UML Class Diagrams into Temporal Object Relational Dababase," *International Journal of Electrical and Computer Engineering*, Vol. 7, No. 5, pp. 2823-2832, 2017.

[7] 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.

[8] 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.

[9] 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.

[10] 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, 1999.

[11] A. Abdurazik and J. Offutt, "Using UML Collaboration Diagrams for Static Checking and Test Generation," *The Third International Conference on the Unified Modeling Language*, pp. 383-395, 2000.

[12] J. Kim, 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.

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