韓國 컴퓨터情報學會 論文誌
第13卷 第5號, 2008. 9.

2008-13-5-1-8

컴포넌트 모델구축을 위한 클래스 코드 자동생성 방법

임 근*, 이 기 영**

Class Code Generation method for Component model construction

Keun Lim*, Ki-Young Lee**

요 약

본 논문에서는 컴포넌트의 단위인 클래스의 일관된 생성과정 및 정형화된 형식을 기반으로 클래스 코드생성을 위한 프로토타입을 구현하였다. 특히 컴포넌트의 기반이 되는 클래스간 연결관계와 집합관계 등을 객체지향언어로는 표현하기 어려운 문제들을 해결하기 위하여 관련성 규칙을 제안하였으며, 이를 통해서 일관성 있는 코드생성의 정형화가 가능하도록 하였다. 또한 코드 생성기를 이용해서 생성된 소스코드는 컴포넌트 모델 구축에 적용되며, 이후에 어플리케이션 개발시 재사용 대상인 비즈니스 컴포넌트의 조립과 분배의 기반이 되도록 한다.

Abstract

In this thesis, we implemented the prototype system for the class code generator based on consistent code generation process and standard type, the class to be component unit. Particularly, we proposed relationship rule to solve the difficult problem by the object-oriented language to association and aggregation between classes based on component, through this method we can make to consistent code generation standard. Also it is adopted to component model construction which is generated code using code generation, and it can be basic assembly and deployment of business components to reusable target in developing application system.

▸ Keyword : 코드생성(Code generation), 컴포넌트 모델(Component Model), 관련성 규칙(Relationship rule), 컴포넌트 분배(Component deployment)

• 제1저자 : 임 근 교신저자 : 이기영
• 접수일 : 2008. 7. 25, 심사일 : 2008. 9. 11, 심사완료일 : 2008. 9. 25.
* 을지대학교 의료산업학부 교수

Ⅰ. 서 론

컴포넌트기반 개발 방법은 닷넷 프레임웍과 같은 플랫폼이나 개발 툴에서 제공하는 라이브러리와 사용자 인터페이스 등을 프로그램의 재사용 대상으로 유용하게 사용되고 있다. 그러나 기존의 컴포넌트 기반 기술로는 서로 다른 개발자에 의해 생산된 컴포넌트들을 소스코드 수정 없이 조립하여 새로운 어플리케이션을 작성할 수 없다. 또한 독자적으로 개발된 컴포넌트의 소스코드를 구할 수 있는 방법도 막연하고, 타인이 작성한 소스코드에 대한 분석이나 일관성 여부를 확신하기는 매우 어렵다. 본 논문에서는 컴포넌트의 단위가 되는 클래스의 소스코드가 일관된 생성과정과 정형화된 표현형식 및 인터페이스가 필요하다고 간주하여, 클래스코드 자동 생성기 구현을 위한 코드 생성 규칙, 연관성 표현방법, 라이브러리 구성 등의 기준을 제시하였다. 본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 살펴보고, 3장에서 클래스코드 생성규칙 설명하며, 4장에 구현 사항, 5장에서 결론및 향후 연구과제를 제시한다.

Ⅱ. 관련연구

2.1 컴포넌트 모델 정의

컴포넌트는 바라보는 시각 또는 관점에 따라 여러가지의 다양한 형태로 표현된다. 또한 컴포넌트에 대한 개념들을 혼돈하는 가장 큰 이유중의 하나는 한쪽 측면에서만 컴포넌트를 정의함으로써 비롯된다고 볼 수 있다[1]. 넓은 의미로 컴포넌트를 정의하는 경우에는 소프트웨어 개발 전 과정에서 생산된 산출물 가운데 재사용할 수 있는 모든 단위의 산출물을 컴포넌트라고 정의할 수 있다. 컴포넌트들은 추상화 수준에서 구별한다. 추상화 정도가 클수록 앞 단계에서 개발된 산출물이고, 후반부에 산출된 결과는 추상화 정도가 작다고 할 수 있다. 따라서 앞 단계에서의 산출물이 개발과정에 많은 영향을 줄 수 있는 요인이므로 정형화와 일관성에 기준하여 설계되고 구현되어야 한다[2]. 다음은 컴포넌트를 여러시각에서 정의한 내용이다.

Fayad : 객체지향 프레임웍은 상속개념을 통해 정의된 여러 클래스들을 구체화하여 어플리케이션을 구축할 수 있는 클래스의 집합이다. 상속 개념은 여러 장점을 가지고 있지만 상속의 속성상 의존성을 내포하는 문제점이 있다[3].

Latchem : 컴포넌트간 최소한의 결합도가 필요하며, 컴포넌트간 역할을 강조한 모델로서 컴포넌트의 기능을 분류하고 정의한다 [4].

Atkinson : 접근 방법에 따라 문서화된 논리적인 컴포넌트를 의미하며, 컴포넌트간 조합을 위한 방법을 내포하고 있다. 이 모델에서 컴포넌트 의미는 타입과 실시간 요소가 아닌 명세 수준으로 제한하고 있다[5]. 본 논문의 의도 또한 클래스 수준의 소스코드 자동 생성을 통해서 정형화와 일관성을 부여할 수 있는 기반을 제공하고자 한다.

2.2 컴포넌트 구성을 위한 관련성 정의

본 논문에서는 개발언어가 지원하는 상속성 이외에 연결관계와 집합관계의 특수성과 다양성을 고려하여 코드생성 규칙과 구현에 적용하였다.

1) 연결관계

연결은 인스탄스간 물리적 또는 개념적 연결을 의미하여 다중성을 정의할 수 있도록 하였다. 특히 역할 이름을 부여하여 다수의 연결관계가 존재할 때 구분자의 기능을 가능하게 하며, 이를 통해 모델 분석과 설계에 명확성을 부여할 수 있다[6].

2) 집합 관계

집합관계는 연결관계의 특수한 형태로 두 클래스가 전체와 부분의 관계이며, 관련성 정의로 비대칭성(Asymmetry), 이행성(Transitivity), 전달성(Propagate)의 특징을 갖는다.

집합관계는 부분과 전체에 대하여 다음과 같이 3가지 기본 형태로 구분될 수 있다.

- 구성(Configuration) : 부분이 전체에 대해 기능적, 구조적 관계를 갖는지의 여부
- 준동형(Homomorphic) : 부분이 전체에 대해 동종 분류로 구분되는지의 여부
- 무변화(Invariant) : 부분이 전체에서 분리될 수 있는지의 여부

〈표 1〉은 3가지 기준에 준하여 부품과 객체 또는 물질과 객체 등의 관계를 표현한 것으로 예를 들어 부품은 객체로 조합이 되더라도 부품자체의 정체성을 잃지 않으며, 객체는 부품의 기준에 따라 분리될 수 있다는 논리에 따라 구성요소는 가능하고 준동형, 무변화 기준은 만족하지 않는다는 표현을 〈표 1〉과 같이 나타낼 수 있다. 이하 각 경우의 사례를 표현한 내용이다.

표 1.집합관계 분류
Table 1. Aggregation Classification

	구성	준동형	무변화
부품-객체 조합	O	X	X
물질-객체 조합	O	X	O
부분-객체 조합	O	O	X
위치-영역 조합	O	O	O
집단-부분 조합	X	X	X
집단-구성원 조합	X	X	O

2.3 링크속성과 링크 클래스

링크속성이란 클래스에 속성이 존재하는 것과 같이 연결관계의 링크에 속성이 존재하는 것을 의미한다. 링크속성은 연산과 클래스의 이름을 가질 수 있다. 이러한 경우를 링크 클래스라 한다[7].

III. 클래스 코드 생성을 위한 규칙

3.1 코드생성 규칙

클래스 정의 부분에서 선언되어야 하는 내용은 데이터 맴버와 맴버함수의 선언과 이들의 가시성 정도를 표현해야 한다. 객체 모델 결과로 코드를 생성하기 위해 객체모델의 작성 단계에서 속성의 타입, 연산의 경우 반환타입, 인수이름, 인수 타입, 그리고 클래스 속성과 연산에 대한 가시성을 결정해 주어야 한다. 이를 위한 본 논문에서 제시하는 코드생성 지침은 다음과 같다.

첫째, 일반적인 개발에서는 연결관계를 양방향성으로 코드에 반영하지만 연결관계의 코드 생성시에 역할 이름을 연결관계의 참조에 대한 방향성 기준으로 제시하고 역할이름이 존재하는 경우 방향성을 갖게 됨으로 이에 따른 참조를 갖게 되도록 코드생성에 반영하였다.

둘째, 집합관계를 코드에 반영할 경우 일반적으로 참조 포인터와 객체 선언방법이 존재한다. 여기에 개발자에게 두 가지 방법에 대한 일관된 기준을 제시하기 위해 무변화 특성을 적용하였다.

셋째, 링크클래스의 코드생성에서는 링크클래스와 연결관계를 가진 클래스와의 변형을 통해 코드에 반영하였다. 또한 연결관계의 다중성까지 반영하여 참조자에 대한 참조를 단수 혹은 복수로 입력받아 코드에 반영할 수 있도록 하였다.

3.2 연결관계 코드생성 규칙

본 논문에서는 방향성을 구분하는 기준으로 역할이름을 이용하는 것을 제안하였다. 연결관계를 구성하는 요소로는 둘 또는 그 이상의 클래스들간 관계 이름이 정의되어야 하며, 관련성에 따라 다양한 관계를 가질 수 있다[8]. 본 절에서는 단방향성, 양방향성, 일대일, 일대다, 다대다 등 다양한 경우를 표현하고 있으나 특히 모든 경우를 내포하고 있는 양방향성에서 다대다의 연결관계에 대한 코드 생성을 설명한다.

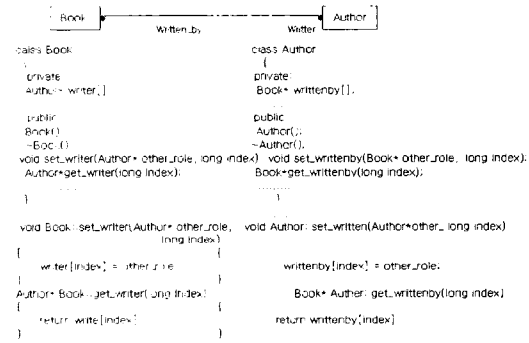


그림 1. 양방향 다대다 연결관계의 코드생성
Fig 1. Either side m:n link Code Generation

〈그림 1〉에서 Book 객체와 Author객체의 양방향, 다대다 참조 포인터 배열을 가지고 있음을 알 수 있다. 개발자가 코드 사용시 주의해야 할 사항은 객체가 소멸하는 경우 연결관계에서 상대편이 자신을 가리키는 참조 포인터를 해지해야 한다.

3.3 집합관계 코드생성 규칙

연결관계와 더불어 집합관계는 객체지향언어에서 직접적으로 처리하지 못하는 부분이다. 그러나

설계 과정에서 상속관계와 더불어 강력한 추상화를 제공해 준다. 집합관계의 분류 기준이 되는 구성(Configuration)은 부분이 전체에 대해 구조적, 기능적 부분이 됨을 의미한다.

1) 집합관계의 준동형 기준 코드 생성

이것은 동질의 종류로 이루어진 경우를 의미한다. 부분이 동질의 종류인 정의에 의해 준동형은 집합관계에서 단 하나의 부품을 다수의 다중성을 가진 관계로 나타낼 수 있다.

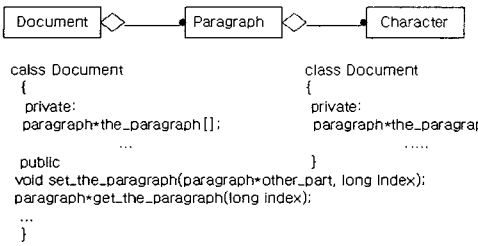


그림 2. 집합관계의 준동형 코드 생성
Fig 2. Homomorphic of Aggregation Code Generation

〈그림 2〉에서 character와 paragraph 그리고 document 의 관계를 나타내고 있다. 집합관계가 준동형을 가지는 것과 여러 부분 객체를 가지고 그 중 일부분의 객체가 다수개의 다중성을 가지는 것은 차이가 있다. 준동형의 정의에서 부분이 동질의 종류가 되어야 하기 때문에 유일하게 하나의 부분 객체가 존재해야 한다. 참조 포인터의 배열이나 부분 객체의 배열 선언으로 코드가 생성된다.

2) 집합관계의 무변화 기준 코드 생성

〈그림 3〉은 전체와 부분이 제거될 수 있는지 여부를 의미한다. 만약 집합관계가 무변화를 갖는다면 객체 선언을 이용한 집합관계 코드를 생성하고, 무변화를 갖지 않는다면 참조 포인터를 이용한 집합관계 코드를 생성한다. 참조포인터는 전체 객체도 존재하고 부분의 객체도 존재할 경우 연결관계와 유사한 관계를 가지게 한다. window 아래 각 버튼은 각각 무변화 기준으로 분류되어 기준 코드를 생성하는 의미를 내포하고 있다.

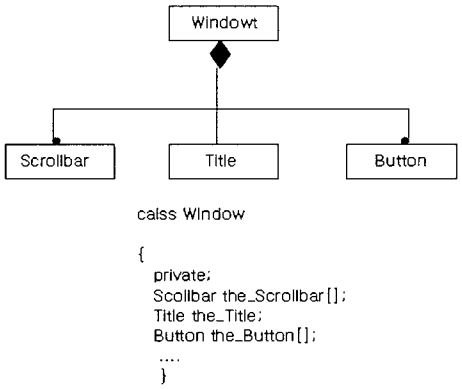


그림 3. 불변성을 갖는 집합관계
Fig 3. Aggregation for Invariant

3.5 링크 클래스의 코드생성

링크 속성의 경우는 링크 클래스 연산은 없고 속성만 존재한다. 링크 클래스는 클래스간 연결관계가 속성과 연산을 가지는 것을 의미하므로 링크 클래스는 링크 속성의 확장으로 생각할 수 있다.

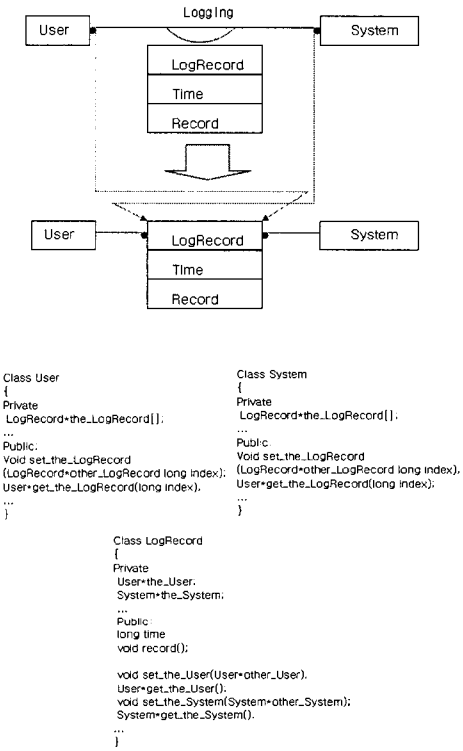


그림 4. 링크클래스 코드생성
Fig 4. Link Class Code Generation

〈그림 4〉는 모델링 결과로서, 이를 User 클래스와 LogRecord 클래스가 일대다의 연결관계를 갖고, System 클래스와 LogRecord 클래스가 일대다의 연결관계를 갖고 있는 경우로 변환하여 코드를 생성한다. 링크 클래스는 연결관계가 클래스와 유사하게 속성과 연산을 가지는 경우이다. 이러한 링크 클래스를 코드 생성할 때는 연결관계의 확장으로 변환한다.

IV. 코드생성기 프로토타입 구현

소프트웨어를 개발할 경우 설계의 결과물인 모델링 결과를 가지고 자동으로 기본 코드를 생성함으로서 설계단계의 결과물인 모델링 구성요소에서 구현으로의 전이를 가능하게 한다.

4.1 코드 생성기 구성 요소 및 관련성

〈그림 5〉 코드생성기의 전체적인 구조는 편집기, 판독기 및 코드 변환규칙을 저장소에 보관하게 되며 이를 기반으로 코드생성기를 통해 코드새성이 이루어진다.

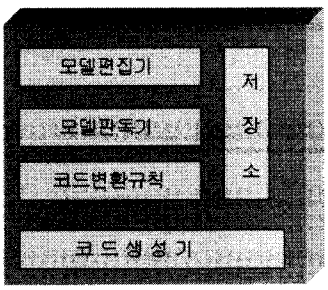


그림 5. 코드 생성기 구조
Fig 5. Architecture of code generator

각 구성요소의 역할및 구성요소간 관련성은 〈그림 6〉과 같이 저장소에 보관된 클래스와 관련성의 내용 및 모델의 편집과정을 통해 원하는 조건의 코드가 코드생성기를 통해 생성된다. 생성된 코드는 코트 변환규칙에 적합한지의 여부를 클래스 변환규칙, 상속관계, 연결관계, 집합관계의 검증을 통해 생성되는 과정을 보이고 있다.

1) 모델 편집기

분석가, 설계자가 모델 편집기를 이용하여 객체 모델을 작성한다. 추후 모델에 대한 변경과 삭제가 모델편집기를 통해서 이루어진다.

2) 모델 판독기

모델편집기에서 작성된 객체 모델 요소를 저장소에 저장하고 검색하는 기능을 수행한다.

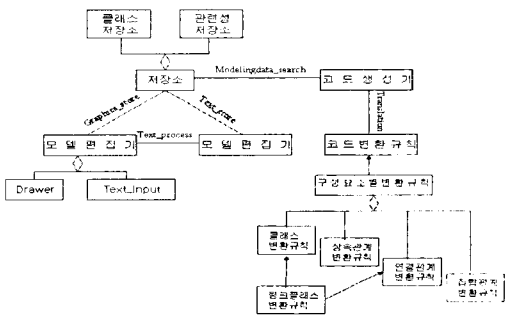
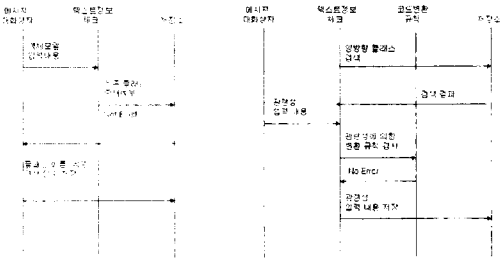


그림 6. 코드생성기 구성요소간 관계
Fig 6. Relation of Code Generation Constitution

객체모델이 클래스인 경우에는 이름, 식별자, 속성이름, 연산이름을 입력받아 처리하고, 객체모델이 관련성인 경우에는 종류, 이름, 다중성, 역할이름을 입력해야 한다.



a) 클래스의 예 b) 관련성의 예

그림 7 모델 판독기 인터액션 모델
Fig 7. Model interpretation Interaction model

〈그림 7〉은 모델 판독기의 기능을 인터액션 모델로 표현한 것으로 판독을 위한 과정을 시각적으로 표현하고 있다.

3) 코드 변환 규칙

모델링 결과를 기반으로 코드 생성을 하기 위해 변환이 필요한 연결관계, 집합관계를 내부적으로 변환하는 규칙을 포함하고 있다.

4) 저장소

편집기에서 객체 모델 구성요소의 그래픽 환경에 대한 정보를 저장하고 모델 판독기에서 객체 모델의 텍스트 입력에 대한 정보를 보관한다. 이후에 이를 기반으로 코드 생성기에서 검색하여 참조한다.

5)코드 변환규칙

코드 변환 규칙은 설계에서 구현으로 자연스러운 전이가 어려운 연결관계와 집합관계의 코드 변환규칙을 가지고 있다. 연결관계및 집합관계의 코드 변환논리는 〈그림 8〉의 Association, Aggregation 참조자 처리를 통해 이루어진다.

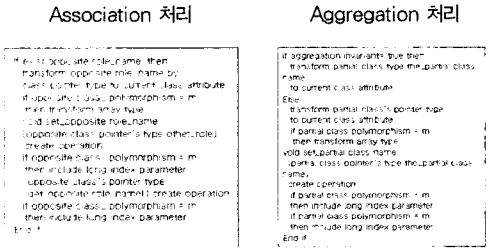


그림 8. Association, Aggregation 참조자 처리
Fig 8. Referance treatment of Association, Aggregation

6) 코드 생성기

저장소에 저장된 내용과 코드 변환 규칙을 이용하여 객체 모델의 기본 코드를 작성한다. C++ 코드를 생성할 경우에는 헤더 파일과 CPP 파일 두 가지 경우를 고려해야 한다. 클래스, 상속관계는 객체지향언어에서 직접적인 처리가 가능하고 링크 클래스는 연결관계와의 관계로 변형하여 코드를 생성한다. 헤더파일 생성논리는 〈그림 9〉와 같다.

4.2 적용사례

프로토타입 시스템을 이용하여 코드생성과정을 설명한다. Matrix 클래스는 행의 크기와 열의 크기, 행렬 요소의 배열을 속성으로 하며, 열 크기 값을 반환하는 get_width 연산, 행 크기를 반환하는 get_height 연산, 그리고 주어진 행렬 요소 값을 정하는 set_element 연산, 반환하는 get_element, display 연산 등이 존재한다. 〈그림 9〉는 프로코타입을 통해 생성된 Matrix 클래스의 헤더 파일과 클래스 연산의 코드를 보이고 있다.

헤더 파일출력

```
클래스 헤더 생성
/ 상속관계 수 > 0 then /* 상속관계가 존재하면*/
  상속관계 코드 생성
end if
클래스 명세, 참생자 수행환경 출력
/* private 속성처리*/
for i=0 to 속성수 - 1
  / private = y then 속성처리
  end if
next i
Aggregation 참조자 처리
Association 참조자 처리

/* private 연산처리*/
for i= 0 to 연산수 -1
  if private = y then
    operation process
  end if
next i
/*protected 속성과 연산출력*/

/*public 속성과 연산출력*/
Aggregation 참조자 처리
Association 참조자 처리
```

클래스 연산출력

```
Printf class Nmae & "  " & opt(index) Name
Set DB = OpenDatabase
Set para_record = DB.OpenRecordset
operation parameter
if 파라메터 수 > 0 then
  Printf "(" & LineNew
Else
  Printf parameniter
End if
if 멀티미디어 클래스 = y then
  멀티미디어 코드검색후 출력
  Exit Sub
End if
Printf "{" & LN
Printf Tabs & "//TODO  add code here " & LN
return type Printf
Printf "}" & LineNew
DB Close
```

그림 9. 헤더 파일 및 클래스함수 생성논리
Fig 9. Generation logic of Header File, Class

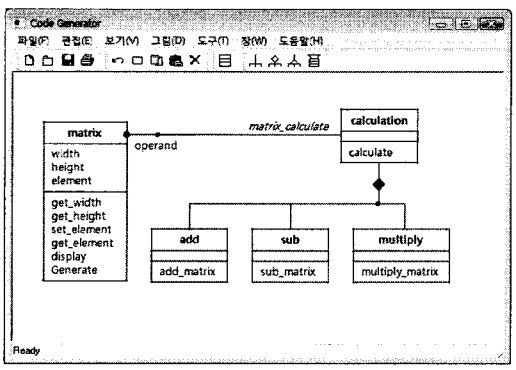


그림 10. 행렬계산 컴포넌트 설계
Fig 10. Component design of Matrix Caculation

〈그림 10〉은 행렬계산에 필요한 4칙연산 컴포넌트를 설계하는 과정으로 Caculation 클래스는 행렬의 계산을 수행하는 calculate 연산이 존재하고 부분 클래스들은 덧셈과 뺄셈, 곱셈에 해당하는 연산이 존재한다. 코드 생성시에는 출력 파일 지정 탭에서 편집영역에 작성된 모든 클래스에 대한 코드를 파일로 생성할 것인지, 또는 하나의 클래스에 대한 코드를 생성할 것인지를 선택해야 한다. 〈그림 11〉은 사용자가 원하는 형태의 코드를 생성할 수 있는 조건을 부여하는 입력탭 설정과정이다.

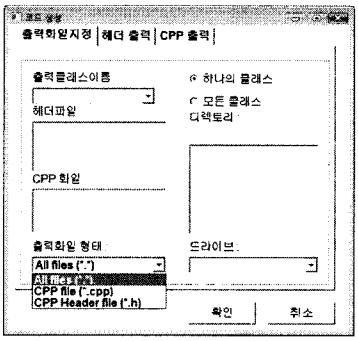


그림 11. 코드 생성을 위한 입력탭
Fig 11. Input Tap for Code Generation

원하는 경로를 지정하고 헤더 파일만 생성할 것인지 CPP, 또는 모든 파일을 생성할지를 선택해야 한다. 〈그림 12〉는 앞 단계에서 선택한 조건의 결과를 CPP 코드로 변환하여 보여준다. 생성된 코드를 이용하여 어플리케이션 개발시 원하는 조건의 코드를 생성하게되고, 이후 수정 요구가 발생할 경우 역추적을 통하여 수정이 가능하다.

```
#include <stdio.h>
#include "calculation"
calculation::calculation()
{
}
calculation::~calculation()
{
}
bool calculation::calculate(int optype)
{
  bool success;
  if (optype == ADD) {
success=the_add.add_matrix(operand[0], operand[1],operand[2]);
  if (success == false) return false;
  else return true;
  }
if (optype == SUB) {
success=the_add.add_matrix(operand[0],operand[1],operand[2]);
  if (success == false) return false;
  else return true;
  }
  if (optype == MULT) {
success=the_add.add_matrix(operand[0], operand[1],operand[2]);
  if (success == false) return false;
  else return true;
  }
  return true;
}
Void calculation::set_operand(matrix* other_rile, long index)
{
    operand[index] = other_role;
}
Matrix* calculation::get_operand(long index)
{
    return operand[index];
}
```

그림 12. Calculation 클래스 CPP 코드
Fig 12. Calculation Class CPP Code

4.3 평가 및 검증

컴포넌트들은 추상화 수준에서 구별한다. 추상화 정도가 클수록 앞 단계에서 개발된 산출물이고, 후반부에 산출된 결과는 추상화 정도가 작다고 할 수 있다. 따라서 앞 단계에서의 산출물이 개발과정에 많은 영향을 줄 수 있는 요인이므로 정형화와 일관성에 기준하여 설계되고 구현되어야 한다. 본 논문에서는 이러한 기준에 부합하기 위하여 정형화와 일관성 기준으로 방향성을 고려한 규칙과, 집합관계에서 무변화와 준동형 요소를 기준으로 비교평가 하였다. 기존의 두가지 방법에서는 방향성의 측면에서는 양방향지향성을 갖는 반면 본 논문에서는 단방향 요소도 지원함으로서 좀더 구체적인 구현이 가능하도록 하였으며, 집합관계에서 무변화의 기준을 만족하는지의 여부로 참조 포인터 구현 방법과 객체 선언 구현 방법을 지원할 수 있도록 하여 기존 방법에 비하여 유연성을 증가시킬수 있을 것으로 기대한다. 이러한 2가지 설계에서의 기준은 코드생성에 좀더 효율적인 코드를 생산하여 개발자에게 보다 합리적이고, 일관된 연결관계, 집합관계를 사용할 수 있도록 한다.

표 2. 비교 평가
Table 2. Comparision Evaluation

	본 논문의 생성기	Rational Rose	WithClass
방향성을 고려한 코드생성	양방향/단방향	양방향	양방향
집합관계 무변화 코드생성	참조포인터/객체선언	참조 포인터	없음
집합관계 준동형에 대한 설계시 제한	준동형의 집합관계에서 또다른 부분을 가질수 없음	없음	없음
동적 모델링 제공	없음	sequence collaboration diagram	state diagram

이러한 기준은 설계단계에서의 집합관계에 대한 의미를 좀더 명확하게 전달할 수 있다. 그리고 링크 클래스를 연결관계와 클래스로 변환한 후 연결관계의 다중성 의미까지 반영된 코드를 생성할 수 있다. 다만 동적 모델링을 제공할 수 있는 근거는 본 논문에서는 지원하고 있지 못한 점이 향후 보완되어야 할 사항이다.

V. 결론 및 향후 연구과제

컴포넌트 구현의 대상이 되는 객체모델 중 관련성에서 연결관계와 집합관계를 처리하는 문제를 보다 효과적으로 구현하기 위해서 연결관계의 경우 역할이름에 방향성의 의미를 갖도록 하여 설계시에 연결관계의 단방향, 양방향성 구분의 기준으로 제시하고, 집합관계의 경우 부분이 전체에서 제거될 수 있음을 의미하는 집합관계의 무변화 특성을 집합관계 코드 생성 기준으로 제시하여 무변화 특성에 적합한 코드를 생성할 수 있도록 함으로서, 구현단계에서 이를 반영하여 개발자가 효과적인 구현은 물론 일관된 소스코드 구성을 통해서 컴포넌트 모델 구축을 위한 전단계의 역할을 할 수 있도록 하였다. 이를 통해서 일관성을 갖는 코드생성의 정형화가 가능하도록 하였다.

향후 연구과제로는 앞서 비교평가에서도 제시한 동적모델링을 위한 역공학 기능이 포함된 모델-코드 관리가 가능한 환경에 대하여 연구가 필요하다.

참고문헌

[1] Crnkovic, I., and Larsson, M., Bulding Reliable Component-based Software Systems, Artech House, 2002

[2] Sally Shlaer and Stephen J. Mellor, Object-Oriented System Anaysis : Object Life Cycles Modeling the World in States, Prentice Hall, 1998

[3] Fayad, M. and Schmidt, D., "Object-Oriented Application Frameworks," Communications of the ACM, Vol.40, No.10, pp.32-38, Oct. 1997.

[4] Latchem, S,. "Component Infrastructures: Placing Software Components in Context," Component_Based Software Engineering, Putting the Pieces Together, Heineman G. and Council, W., ed., Addison-Wesley, 2001.

[5] Atkinson, C., et al., Component-Based Product Line Engineering with UML, Addison Wesley, 2002.

[6] Kleppe, Anneke G, et al,. MDA Explained, Addison Wesley, 2003.

[7] Frankel, D. S., Model Driven Architecture-Applying MDA to Enterprise computing, Wiley Publishing, Inc,. 2003.
[8] 임근, 형식명세로 변환된 객체모델의 검증방법과 시뮬레이션, 한국컴퓨터정보학회논문지, 제12권 6호, 2007, pp123-130.

저 자 소 개

임 근

1992년 3월 - 현재
을지대학교 의료산업학부 교수
1998년 중앙대학교 대학원 컴퓨터공학과(공학박사)
관심분야 : S/W공학, CBD 방법론, HCI 등

이 기 영

1991년 3월 - 현재
을지대학교 의료산업학부 교수
2005년 건국대학교 대학원 컴퓨터공학과(공학박사)
1984년~1991년 한국해양연구원 연구원
1996년~1998년 한국컴퓨터정보학회 이사 및 서울동부지회장
관심분야 : 공간 데이타베이스, GIS, LBS, USN, 텔레매틱스 등