DOI : 10.5392/JKCA.2011.11.5.075

추상구문트리를 이용한 어스팩트 마이닝 프로세스 설계

Aspect Mining Process Design Using Abstract Syntax Tree

이승형, 송영재
경희대학교 컴퓨터공학과

Seung-Hyung Lee(shlee7@khu.ac.kr), Young-Jae Song(yjsong@khu.ac.kr)

요약

어스팩트 지향 프로그래밍은 시스템에서 크로스커팅 개념을 추출하고 소프트웨어 모듈화를 통하여 기능의 분산과 코드의 혼란을 해결하기 위한 패러다임이다. 현존하는 어스팩트 개발 방법은 크로스커팅 대상 영역을 추출에 어려움이 있기 때문에, 어스팩트 마이닝을 적용하기가 쉽지 않다. 어스팩트 마이닝에서는 기존 프로그램의 리팩토링 요소를 크로스커팅 영역으로 변환하는 기술이 필수적이다.

본 논문에서는 리팩토링에 적합한 크로스커팅 영역 자동 추출을 위한 시스템에서 크로스커팅 개념을 추출하기 위한 어스팩트 마이닝 방법을 제안한다. 소스 모듈의 추상 구문구조 명세를 이용하여, 모듈의 구조적 중복 관계 요소를 추출한다. Apriori 알고리즘을 통하여 중복 구문트리를 생성하고, 크로스커팅 영역 대상인 중복된 소스 모듈을 자동 생성, 최적화 할 수 있다. Berkeley Yacc의 berbose.c 모듈을 제안하는 마이닝 프로세스에 적용해 본 결과, 원본 대비 9.47%의 길이와 부피의 감소하였고, CCFinder 대비 4.92%의 길이 감소, 5.11%의 부피 감소 효과를 확인하였다.

■ 중심어 : | 크로스커팅 개념 | 추상 구문 트리 | 어스팩트 마이닝 |

Abstract

Aspect-oriented programming is the paradigm which extracts crosscutting concern from a system and solves scattering of a function and confusion of a code through software modularization. Existing aspect developing method has a difficult to extract a target area, so it is not easy to apply aspect mining. In an aspect minning, it is necessary a technique that convert existing program refactoring elements to crosscutting area.

In the paper, it is suggested an aspect mining technique for extracting crosscutting concern in a system. Using abstract syntax structure specification, extract functional duplicated relation elements. Through Apriori algorithm, it is possible to create a duplicated syntax tree and automatic creation and optimization of a duplicated source module, target of crosscutting area. As a result of applying module of Berkeley Yacc(berbose.c) to mining process, it is confirmed that the length and volume of program has been decreased of 9.47% compared with original module, and it has been decreased of 4.92% in length and 5.11% in volume compared with CCFinder.

■ keyword : | Crosscutting Concern | Abstract Syntax Tree | Aspect Mining |

접수번호 : #110426-003
접수일자 : 2011년 04월 26일
심사완료일 : 2011년 05월 12일
교신저자 : 송영재, e-mail : yjsong@khu.ac.kr

I. 서 론

구조적/객체지향 프로그래밍은 여러 장점을 가지고 시스템 개발에 널리 사용되고 있는 방법이다. 하지만, 복잡한 시스템에서는 요구사항과 기능을 모두 충족시키면서 설계를 하는 것이 어려워진다. 장점을 살리지 못하는 어려움은 크게 두 가지로 나타난다. 기능의 분산(scattering)과 코드의 혼란(confusion)이다. 기존의 프로그래밍 등은 각 모듈에서 수행해야 하는 기본적이고 대표적인 관심사인 핵심 관심사를 모듈화 하는데 효율적이지만, 여러 모듈에 걸치는 속성을 지닌 크로스커팅 영역을 모듈화 하는데 어려움[1]이 있다.

크로스커팅 문제에 가장 유력한 해결 방안으로는 어스팩트 지향 프로그래밍이 손꼽히고 있다. AOP는 한 모듈이 다른 모듈의 구조나 행위을 수정하거나 확장할 수 있도록 해 주는 메커니즘을 제공하고 이를 통해 크로스커팅 문제를 해결한다.

크로스커팅 영역을 이용하여 시스템을 최적화하기 위하여, 어스팩트 마이닝에서 주요한 어스팩트로 고려될 수 있는 것은 중복 코드이며 중복 코드를 크로스커팅 영역으로 변환하는 마이닝 기술이 필요[2]하다.

크로스커팅 영역의 추출 방법으로 추상구문 분석을 통하여 생성되는 구문분석 엘리먼트를 이용한다. 분석된 엘리먼트에 인덱스를 부여하고, 인덱스 사이의 구문관계를 생성한다. 생성된 구문관계의 제어의존 관계를 분석하여 구문분석 트리를 생성한다. 구문분석 트리에 고유한 에지 인덱스를 부여하고, Apriori 알고리즘을 이용하여 빈발 에지 인덱스를 추출하여 중복구문 트리를 생성한다. 생성된 중복구문 트리를 크로스커팅 영역으로 변환하는 기술을 현존하는 시스템에 적용, 어스팩트 마이닝에 적용함으로 시스템 모듈을 최적화할 수 있다.

본 연구는 서론에 이어 제2장에서는 어스팩트 마이닝과 중복코드 추출기법에 대하여 설명하였다. 제3장에서는 추상구문트리를 이용한 어스팩트 마이닝 프로세스를 제안하였다. 제4장에서는 표준 파서기인 Berkeley Yacc의 시스템 모듈을 도입하여, 어스팩트 마이닝 프로세스의 상세한 과정을 설명하였다. 제5장에서는 원본모듈, ccFinder, 제안하는 어스팩트 마이닝 프로세스를 적용하여 크로스커팅 영역을 추출하고, Halstead's 방법을 이용하여 정량적인 평가를 하였다. 마지막으로 제6장에서는 결론에 대하여 기술하였다.

II. 관련연구

1. 크로스커팅 개념과 어스팩트 마이닝

크로스커팅의 필요성을 이해하는 가장 기초가 되는 개념은 '관심의 분리'이다. 관심의 분리는 컴퓨터 프로그래밍의 가장 기초가 되는 원리 중 하나이다. 핵심 관심은 기존의 객체지향 분석/설계(OOAD)를 통해 쉽게 모듈화와 추상화가 가능하지만[3], 크로스커팅 영역은 객체지향의 기본 원칙을 지키면서 분리해서 모듈화하는 것이 어렵다. 주 관심사에 의해 시스템을 모듈화하고 나면 나머지 부 관심사에 해당하는 코드는 여기저기에 분산, 중복되어 지저분한 코드가 발생[4]되기 때문이다.

어스팩트 마이닝은 크로스커팅 영역으로 설계되지 않은 기존 시스템을 분석, 크로스커팅 영역을 추출하고 추출된 크로스커팅 영역을 AOP 방법에 적용함으로서, [그림 1]과 같은 구조로 시스템을 최적화 하는 것이다.

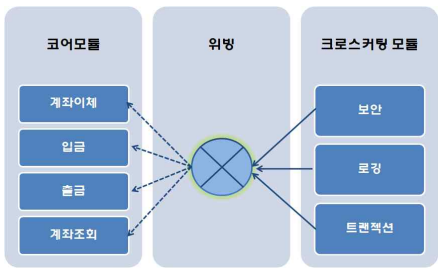

그림 1. 어스팩트 마이닝의 개념

2. 중복코드 추출 기법

코드 검출은 프로그램 스캔에 의하여 수동으로 찾아서 하나씩 중복 식별할 수 있다. 프로그램 사이즈에 의존하여, 수동 프로세스는 지루하고 노동 집약적일 수 있다. 자동 코드 검출 도구는 검출하기 위해 필요한 시간과 노력을 감소시킬 수 있다. 다양한 연구를 통하여

프로그램 표현의 다른 수준을 검사하는 코드 검출 툴의 개발을 하고 있다. 이러한 연구는 텍스트 기반, 토근 기반이 있다. 각각의 기반은 소스 코드를 정형화하기 위한 방법을 의미한다.

텍스트 기반 추출[5]은 텍스트의 공백 또는 특정 문자를 치환 삭제하여 정형화 하는 것으로 텍스트 고유의 형태를 그대로 유지하면서 불필요한 요소(공백 등)들을 제거하여 비교한다.

토큰 기반 추출방법은 CCFinder[6]에서 사용하는 방법이다. 소스코드에 대한 어휘 분석이 적용되고, 그 다음 검출을 위한 근거로 토큰을 사용할 수 있다. 토큰 기반[7]은 더 많은 영역을 치환/정형화 한다. 제어문과 연산자를 제외한 파라미터를 토큰으로 치환하며 「for (; i != s . end ( ) ; ++ i)」 의 경우 제어문과 비교연산자를 제외한 파라미터를 토큰으로 대체하여 「for (; $p != $p . $p ( ) ; ++ $p)」과 같은 패턴을 작성한다. 리팩토링 시 대상의 명확성을 고려할 때 텍스트 기반이 토큰에 비해 상대적으로 불필요한 중복요소를 줄일 수 있다.

Ⅲ. 추상구문트리를 이용한 어스팩트 마이닝 프로세스

어스팩트 마이닝의 목적은 현존하는 시스템에서 크로스커팅 영역을 자동 추출하고, 추출된 크로스커팅 영역을 AOP 방법에 적용함으로서 시스템 구성을 최적화하는 것이 목적이다. 크로스커팅 영역을 추출하기 위하여 텍스트 정형화 방법을 이용한다면, 동일한 구조이나 식별자가 다른 크로스커팅 영역으로 추출할 수 없다. 토큰 기반 정형화 방법을 사용한다면, 「register int i, j;」, 「register int i register int j;」와 같이 코드의 의미는 동일하나 구조가 다를 경우, 크로스커팅 영역으로 추출할 수 없다. 이러한 문제를 해결하기 위하여 추상구문트리 기반의 어스팩트 마이닝 프로세스를 [그림 2]와 같이 정의 제안한다.

그림 2. 추상구문트리를 이용한 어스팩트 마이닝 프로세스

첫 번째는 현존하는 시스템의 각 모듈을 파싱하여 추상구문분석을 하는 단계이다. 파싱을 통하여 구문분석 엘리먼트를 추출하고 명세서를 작성한다. 두 번째는 각 모듈의 추상구문트리를 생성하는 단계이다. 작성된 명세서를 이용하여 엘리먼트 사이의 제어 의존관계를 분석하고, 추상구문트리를 생성한다. 세 번째는 중복구문트리를 생성하는 단계이다. 각 모듈의 추상구문트리에서 반복되는 에지를 검색, 중복구문 트리를 생성한다. 네 번째는 크로스커팅 영역을 추출하는 단계이다. 생성된 중복구문트리를 이용하여, 크로스커팅 영역을 추출한다. 다섯 번째는 어스팩트 마이닝을 수행하는 단계이다. 추출된 크로스커팅 영역을 이용하여 AOP언어에 적용함으로서 시스템 모듈 구성을 최적화한다. 각 단계의 상세한 내용은 다음 장에서 기술하였다.

Ⅳ. 어스팩트 마이닝 프로세스의 상세

추상구문 분석을 위하여 파서 생성기 Berkeley Yacc [8]의 시스템 모듈을 사용하였다. 어스팩트 마이닝 프로세스 적용을 위하여 발췌한 모듈을 사용하였다.

1. 추상구문 분석

구문분석 파싱을 통하여, 추상구문 분석 엘리먼트를 추출한다. 추상구문트리 생성을 위하여 필요한 엘리먼트 분류와 명세요소는 [그림 3]과 같다.

분 류	세부항목	설 명
FileAST:		AST의 최상위
Func_Def	Decl	이름 선언
	TypeDecl	기본 타입
	IdentifierType	타입에 대한 식별자
param_decls	Decl	선언된 이름
	Storage	저장소 명세 리스트
	IdentifierType	타입에 대한 식별자
body	If, switch	제어문
	For, while	반복문
	FuncCall, FuncDecl	함수 관련문
	ArrayDecl, ArrayRef	배열 관련문
	Assignment	연산자
	Return	반환 파라미터 관련

그림 3. 구문분석 엘리먼트의 분류와 명세요소

구문분석 파싱을 통하여 생성된 추상구문 명세는 [그림 4]와 같다.

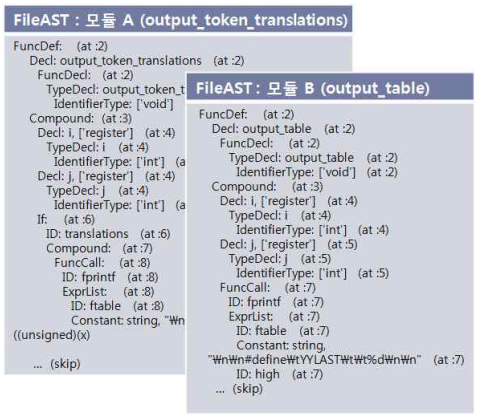

그림 4. 구문분석을 통하여 생성된 추상 구문분석 명세

2. 추상구문트리 생성

추상 구문분석 명세에서 엘리먼트 사이의 제어의존관계를 분석하여, 중복된 엘리먼트를 추출하였다. 추출된 엘리먼트를 이용하여 GDL(Graph Description Language) 명세를 생성하고, 각 모듈별 구분분석 트리를 생성하였다.

[그림 5]는 각 엘리먼트의 제어의존관계 생성을 위하여, 부여된 고유한 엘리먼트 인덱스이다.

인덱스	엘 리 먼 트
AA	FuncDef:
AB	Decl: output_token_translations
AC	FuncDecl:
AD	TypeDecl: output_token_translations
AE	IdentifierType: ['void']
AF	Compound:
AG	Decl: i, ['register']
AH	TypeDecl: i
AI	IdentifierType: ['int']
...	... (skip)

그림 5. 고유한 엘리먼트 인덱스

[그림 6]은 구문분석과정을 통하여 추출된 추상구문 명세에 엘리먼트 인덱스를 매핑한 것이다. [그림 5]의 고유한 인덱스를 중심으로, 영역깊이(Scope_Depth)와 소스라인의 번호를 매핑 하였다.

모듈 A (output_token_translations)

영영 깊이	인덱스	소스라인
1	AA	2
2	AB	2
3	AC	
4	AD	
5	AE	
2	AF	
3	AG	
4	AH	
...	... (skip)	

모듈 B (output_table)

영영 깊이	인덱스	소스라인
1	AA	2
2	BQ	2
3	AC	2
4	BR	2
5	AE	2
2	AF	3
3	AG	4
4	AH	4
...	... (skip)	...

그림 6. 추상구문 명세에 엘리먼트 인덱스 매핑

[그림 7]은 제어의존관계 추출을 위하여 제안한 의사코드이다. 우선, 인덱스 매핑 테이블에서 각 매핑 라인 분한다. 그 다음 매핑 라인에서 Scope, Code, LineNumber를 분리를 분리한다. 각 노드(Node)를Code와 LineNumber로 정의한다. 「Scope = 1」은 상위 엘리먼트가 없기 때문에, 1이상의 Scope에서만 수행하게 된다. Node_Vector[2]이면, Node_Vector[1]을 Start_Node로 저장하고, Node_Vector[2]를 End_Node로 저장하는 과정을 반복수행하면서, 제어의존관계를 추출하였다.

```
//Control Dependency Relation Detect Pseudocode
Load Index_Mapping_Table
Define Node_Vector for each Scope
 for each Mapping_Line in Index_Mapping_Table
Begin
  (Scope,Code,LineNumber) = split each element in Mapping_Line
   Define Node = Code + LineNumber
   Node_Vector[Scope] = Node
   if (Scope > 1)
   Begin
      Start_Node = Node_Vector[Scope - 1]
      End_Node = Node_Vector[Scope]
      Print Edge
   End
End
```

그림 7. 제어의존관계 추출을 위한 의사코드

[그림 7]의 의사코드를 통하여, 추출된 의존관계는 [그림 8]과 같다.

모듈 A (output_token_translations)

의존관계	의존관계
AA2,AB2	AH4,AI
AB2,AC2	AF3,AJ4
AC2,AD2	
AD2,AE4	
AA2,AF3	
AF3,AG4	
AG4,AH4	

모듈 B (output_table)

의존관계	의존관계
AA2,BQ2	AF3,AJ5
BQ2,AC2	AJ5,AK5
AC2,BR2	AK5,AI
BR2,AE2	AF3,AN7
AA2,AF3	AN7,AO7
AF3,AG4	AN7,AP7
AG4,AH4	... (skip)
AH4,AI	

그림 8. 영역 깊이를 고려한 의존관계

각 구문 사이의 제어 의존성을 해당 라인에 대한 방향성 순서쌍으로 표현하고, 순서쌍을 이용하여 각 모듈의 구문분석 트리를 일부를 [그림 9]와 같이 생성하였다.

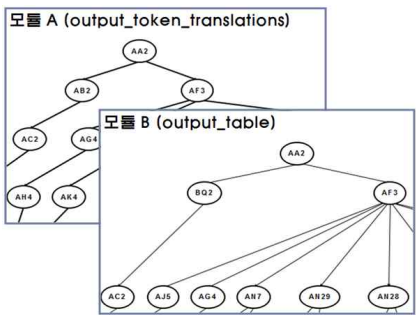

그림 9. 생성된 각 모듈의 추상구조트리의 일부

3. 중복구문트리 생성

생성된 각 모듈의 구문분석트리를 이용하여, 중복구문트리를 생성하였다. 중복구문트리를 생성하기 위하여, 추상구조트의 에지관계를 분석한다. 자주 반복되는 에지를 추출하게 되면, 반복되는 엘리먼트를 확인할 수 있기 때문이다.

에지관계 분석을 위하여, 각 구문트리에 3연속 에지를 추출하기 위한 인덱스를 [그림 10]과 같이 생성하였다.

인덱스	공 통 관 계
aa	AA,AB,AC,AD
ab	AB,AC,AD,AE
ac	AA,AF,AG,AH
ad	AA,AF,AJ,AK
ae	AA,AF,AL,AM
af	AA,AF,AL,AF
ag	AF,AG,AH,AI

인덱스	공 통 관 계
ah	AF,AJ,AK,AI
ai	AF,AL,AF,AN
aj	AF,AL,AF,AL
ak	AF,AL,AF,AZ
al	AF,AL,AF,BC
am	AL,AF,AN,AO
an	AL,AF,AN,AP
...	... (skip)

그림 10. 부여된 고유 에지 인덱스

중복구문 트리의 명확도를 향상하기 위하여, 4개의 구문 엘리먼트의 공통관계(연속 3에지)를 적용하였다. 연속 3에지를 검출하기 위한 의사코드를 [그림 11]과 같이 제안하였다.

```
//3Edge Detect Pseudocode

Load ORG_Edge_List
  Edge_List EL_A = ORG_Edge_List
  Edge_List EL_B = ORG_Edge_List
  Edge_List EL_C = ORG_Edge_List

for each Edge in EL_A
Begin
  (NodeA_EL_A , NodeB_EL_A) = split Node in Edge
    for each Edge in EL_B
    Begin
      (NodeA_EL_B , NodeB_EL_B) = split Node in Edge
      if (NodeA_EL_B == NodeB_EL_A)
        Begin
          for each Edge in EL_C
          Begin
            (NodeA_EL_C , NodeB_EL_C) = split Node in Edge
              if (NodeA_EL_C == NodeB_EL_B)
              Begin
                PRINT NodeA_EL_A,NodeB_EL_A,NodeB_EL_B,NodeB_EL_C
              End
          End
          Shift(Pop) Edge in EL_C //Prevent Loop
        End
    End
    Shift(Pop) Edge in EL_B //Prevent Loop
End
```

그림 11. 연속된 3에지 검출을 위한 의사코드

분리된 에지 NodeA_EL_B (AB2)와 NodeB_EL_A (AB2)를 비교하여 동일하다면, 분리된 EL_C와 NodeA_EL_C (AC2)와 NodeB_EL_B (AC2)를 비교를

수행하고 동일하다면, 연속된 세 에지로 판명되어 NodeA_EL_A, NodeB_EL_A, NodeB_EL_B, NodeB_EL_C (AA2, AB2, AC2)로 출력하는 과정을 반복하면서, 연속 세 개의 에지를 추출한다.

중복된 인덱스를 추출하기 위하여, Apriori 알고리즘 적용한다. Apriori 알고리즘은 연관 규칙 마이닝의 방법이다. 반복되는 패턴 분석을 위하여 사용되고, 본 논문에서는 에지 인덱스에 대한 발생 빈도를 기반으로 각 에지 사이의 연관관계를 밝히기 위한 방법으로 사용한다.

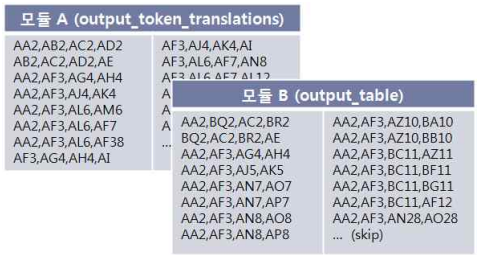

그림 12. 빈발되는 연속 3에지 검색을 위한, 인덱스의 후보 집합

[그림 12]는 각 모듈 구문트리에서 발생되는 연속 3 에지를 추출한 것이다. 추출된 후보 집합은 크로스커팅 영역 후보 코드를 추출하기 위하여 사용한다. [그림 13]은 후보 집합을 [그림 10]에서 부여한 고유 에지 인덱스를 이용하여, 각 모듈의 전체 에지 인덱스를 표현한 것이다.

모듈 ID	항목 (에지 인덱스)
모듈A과 모듈B	aa ab ac ad ae af ag ah ai aj ak al am an ao ap aq ar as at au av aw ax ay az ba bb bc bd be bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv bw bx by bz ca cb cc cd ce cf cg ch ci cj ck cl cm cn co cp cq

그림 13. 모듈A와 모듈B의 발견되는 에지 인덱스

중복된 빈발 인덱스를 추출하기 위하여, Apriori 알고리즘[9]을 이용한다. Apriori는 반복되는 패턴 분석을 위하여 사용되는 알고리즘으로 연관 규칙 마이닝 방법(Association Rule Mining)으로 사용된다. 제안하는 논문에서는 에지 인덱스에 대한 발생 빈도를 기반으로 각 에지 사이의 연관관계를 밝히기 위한 방법으로 사용한다. 각 모듈에서 빈발 에지를 추출하기 위하여, Apriori 알고리즘을 적용하여 [그림 14]와 같이 추출하였다.

그림 14. 빈발 에지의 추출

추출된 최종 빈발 에지의 항목 집합은 [그림 15]와 같다.

모듈 ID	항목 (에지 인덱스)
추출된 빈발항목	ac ad ai ak an aq bc bd be bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv bx by

그림 15. 추출된 빈발 에지 항목

중복구문 트리를 생성하기 위하여, 추출된 빈발 에지 항목([그림 15])을 이용한다. 빈발항목 ac에 맵핑된 구문 패턴 AA,AF,AG,AH의 관계를 포함하는 의존관계를 각 모듈 별로 AA2,AF3,AG4,AH4와 같이 검색하고, 이에 해당되는 소스 코드는 라인번호 2, 3, 4, 4를 근거로 추출한다. 추출된 빈발 엘리먼트는 [그림 16]과 같다.

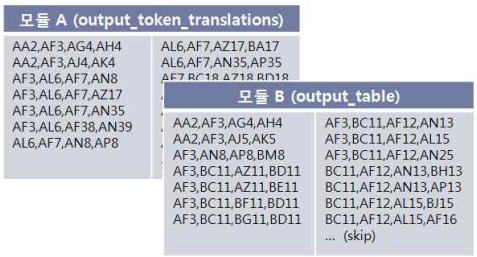

그림 16. 빈발 엘리먼트 추출

추출된 빈발 엘리먼트를 이용하여 생성된 각 모듈의 중복구문 트리의 일부는 [그림 17]과 같다.

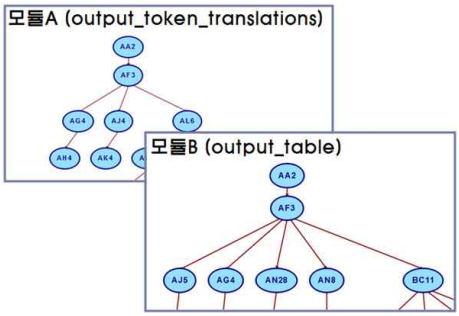

그림 17. 생성된 중복구문 트리의 일부

4. 크로스커팅 영역 추출

생성된 각 모듈의 중복구문 트리 이용하여, 최적화된 크로스커팅 영역을 추출한다. 추출코드를 최적화하기 위하여, 필터링을 이용한다. 연속된 엘리먼트 노드 수 1 이하를 제외한 나머지 엘리먼트를 이용하여 크로스커팅 영역을 추출하였다.

연속된 엘리먼트 노드 수 : 1이하 제한
모듈A (output_token_translations)의 코드

엘 리 먼 트	라인	추 출 코 드
AG,AH,AJ,AK	4	register int i, j;
AN,AP	8	fprintf(ftable,
AZ,BA	17	j = 10;
AZ,BC,BD,BE,BF,BG	18	for (i = 1; i <= max_user_token_number; i++)
AN,AP,AQ,BH	20	putc(',', ftable);
AL,BA,BB,BJ	22	if (j >= 10)
AN,AP	24	putc('₩n', ftable);
AZ,BA,BE	25	j = 1;
BA,BG	29	j++;
AN,AO,AP,AQ,BD,BL,BM	32	fprintf(ftable, "%6d", token_translations[i]);
AN,AP,AQ,BO	35	fprintf(ftable, "₩n};₩n");
AN,AP,AQ	39	fprintf(ftable, "₩n#define YYTRANSLATE(x) (x)₩n");

그림 18. 추출된 모듈A의 크로스커팅 영역

연속된 엘리먼트 노드 수 : 1이하 제한
모듈B (output_table)의 코드

엘 리 먼 트	라인	추 출 코 드
AZ,BC,BD,BE,BF,BG	11	for (i = 1; i <= high; i++)
AN,AP,AQ,BH	13	putc(',', ftable);
AL,BA,BB,BJ	15	if (j >= 10)
AN,AP	17	putc('₩n', ftable);
AZ,BA,BE	18	j = 1;
BA,BG	22	j++;
AN,AO,AP,AQ,BD,BL,BM	25	fprintf(ftable, "%6d", table[i]);
AN,AP,AQ,BO	28	fprintf(ftable, "₩n};₩n");

그림 19. 추출된 모듈B의 크로스커팅 영역

5. 어스팩트 마이닝 수행

포인트컷으로 설정과 조인포인트에서의 실행으로 어스팩트 마이닝을 적용한다. 메소드 호출 전후에 사용자 지정 행위를 수행하는 어드바이스인 around()를 사용하고, for문 실행 시 파라미터 호출을 우회하여 어드바이스의 정의 내용이 실행되도록 하며 이를 크로스커팅 기능 명세([그림 20])로 작성한 후 위빙을 실시한다.

```
aspect New_Yacc_Fnc {
infile ("output.c") && infunc(output_token_translations);
infile ("output.c") && infunc(output_table);
....
advice output_token_translations() : temp_int = max_user_token_number; temp_arr = token_translations;
advice output_table() :  temp_int =  high; temp_arr = table;
around(register int i, register int temp_int): call($ for (i = 1; i <= temp_int; i++)) && args(i,temp_int) {

    putc(',', ftable);

    if (j >= 10)
                    {
                     putc('\n', ftable);
                     j = 1;
                    }
    else
                    {
                     j++;
                    }

    fprintf(ftable, "%6d", temp_arr[i]);
  }
 }
}
```

그림 20. AspectC를 이용한 크로스커팅 명세

V. 평가

본 연구의 목표는 제안하는 마이닝 프로세스를 이용하여, 기존 중복코드 검출 기법의 문제들을 극복하여 어스팩트 마이닝에 크로스커팅 개념을 성공적으로 적용하는 것이다. 제안하는 프로세스가 크로스커팅 영역을 효율적으로 수행하는지 정량적인 평가를 위하여, 소프트웨어의 복잡도를 측정, 수치화 하는 Halstead's 소프트웨어 복잡도 평가기법을 도입하였다.

Halstead's 복잡도 측정 요소의 주요 척도는 프로그램의 길이, 프로그램의 부피, 프로그램의 레벨이며 각 요소는 다음과 같이 정의된다.

$$Length\ N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

$$Volume\ V = N \log_2 (n_1 + n_2)$$

$$Level\ L = \frac{2}{n_1} * \frac{n_2}{N_2}$$

여기에서 n_1은 프로그램에서 사용된 연산자(operator)의 종류로서 실행 가능한 문장, 콤마, 라이브러리 함수 등 있다. n_2는 피연산자(operand)의 종류로

서 변수, 상수 등이 있다. N_1은 연산자의 전체 출현횟수, N_2는 피연산자 전체 출현횟수로 $n = n_1 + n_2$, $N = N_1 + N_2$을 나타낸다. $Length\ N$과 $Volume\ V$는 작을수록, $Level\ L$은 높을수록 복잡도가 낮은 것을 의미한다.

제안된 프로세스와 CCFinder의 어스팩트 후보 추출 후, 리팩토링에 대한 복잡도를 각각 측정하여 [그림 21]과 같은 결과를 산출하였다.

모듈	원본 모듈(ORG)			CCFinder(CCF)			제안하는 프로세스(AST)		
	Length	Volume	Level	Length	Volume	Level	Length	Volume	Level
error.c	876	5687	0.021853	836	5400	0.022275	744	4756	0.023742
lalr.c	2383	17491	0.00596	2252	16509	0.006285	1890	13699	0.007433
lr0.c	2030	14771	0.007513	1888	13648	0.007733	1781	12840	0.008081
main.c	1309	9968	0.015106	1098	8261	0.017425	1098	8261	0.017425
output.c	4420	35234	0.005669	3927	31074	0.006117	3840	30269	0.006098
reader.c	7754	65689	0.004034	7398	62492	0.004131	7248	61074	0.004151
verbose.c	1336	9146	0.008359	1241	8280	0.008939	1180	7857	0.009698

그림 21. 측정된 Halstead's 복잡도

berbose.c의 모듈을 예를 들면, 프로그램 길이는 원본 모듈[6]에서는 1336, CCFinder에서는 1241, 제안하는 프로세스(ASS)에서는 1180으로 측정되었다. 이 결과는 원본 모듈 대비 9.47%, CCFinder 대비 4.92%로 향상된 결과를 보였다. 프로그램 부피는 원본모듈에서는 9146, CCFinder에서는 8280, 제안하는 프로세스(ASS)에서는 7857으로 측정되었다. 이 결과는 원본 모듈 대비 9.47%, CCFinder 대비 5.11%로 향상된 결과를 확인하였다.

VI. 결론

객체지향 프로그래밍에서는 캡슐화 된 객체 내의 특정 요소만을 크로스커팅 후 재조합 하는 것이 불가능하였다. 따라서 각 객체 내의 공통적인 리팩토링 요소인 중복코드 처리에 있어서는 많은 어려움을 가지고 있다. 이를 해결하기 위한 방안으로 어스팩트 개념을 도입, 활용할 수 있다. 하지만, 어스팩트 개념 자체가 객체지향 리팩토링 요소에 대한 구체적이고 객관화된 접근방법을 제공하고 있지 않다.

제안하는 논문에서는 객체지향 프로그램의 리팩토링 요소에 대해 어스팩트 적용 과정을 개선하였다. 객체지향 리팩토링에 주요한 관심 요소인 메소드 내의 파라미터 사이에 존재하는 중복 코드에 대한 처리는 AspectJ, AspectC 등의 어스팩트 지원 프레임워크에서 직접적인 명세가 불가능하고 어스팩트 지원 프레임워크의 핵심인 크로스커팅 영역에 대한 정의가 모호하여 사용목적에 따라 이를 특정 지을 수 있는 접근 방법이 필요하였다. 리팩토링 시 크로스커팅 영역의 모호성을 구체화할 수 있는 방안으로 추상 구문구조 분석을 이용한 5단계의 정형화된 프로세스 모델을 제시하였으며, 어스팩트 지원 프레임워크의 종류에 관계없이 일관된 결과를 얻을 수 있다. Berkeley Yacc의 berbose.c 모듈을 제안하는 마이닝 프로세스에 적용해 본 결과, 원본 대비 90.53%의 길이와 부피의 감소하였고, CCFinder 대비 95.08%의 길이 감소, 94.89%의 부피 감소효과를 확인하였다.

본 논문에서 제안하는 마이닝 프로세스를 어스팩트 개념으로 리팩토링에 접근하여 캡슐화 된 객체의 재조합 문제를 해결할 수 있었으며, 크로스커팅 리팩토링의 난제인 크로스커팅 영역 추출 또한 추상구문트리를 이용하여 최적화된 시스템을 구성할 수 있게 되었다.

참 고 문 헌

[1] D. P. Mohapatra and M. Sahu, "Dynamic Slicing of Aspect-Oriented Programs," in Proc. of Informatica, 2008.

[2] S. Apel, C. Kastner, and D. Batory., "Program refactoring using functional aspects," In Proceedings of the 7th International Conference on Generative Programming and Component Engineering. ACM Press, pp.161-170, 2008.

[3] M. P. Monteiro, "Object-to-aspect refracturing for feature extraction," in Proc. of AOSD, 2004.

[4] S. Hanenberg, C. Oberschulte, and R. Unland., "Refactoring of Aspect-Oriented Software," In Proceedings of the 4th International Conference on Object-Oriented and Internet-based

Technologies, Concepts, and Applications for a Networked World, pp.9-35, 2003.

[5] M. Rieger and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code," in Proc. of ICSM, 1999.

[6] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue, "CCFinder: A Multilinguistic Token Based Code Clone Detection System for Large Scale Source Code," IEEE transactions on software engineering, Vol.29, No.7, 2002(7).

[7] B. S. Baker, "A Program for Identifying Duplicated Code," in Proc. of WCRE, 1999.

[8] http://invisible-island.net/byacc/byacc.html

[9] M. Hahsler, B. Grun, K. Hornik, and C. Buchta, "Introduction to arules - A computational environment for mining association rules and frequent item sets," The Comprehensive R Archive Network, 2010(3).

저 자 소 개

이 승 형(Seung-Hyung Lee) 정회원

- 1999년 8월 : 용인대학교 전산통계학과(이학사)
- 2002년 2월 : 경희대학교 전자계산공학과(공학석사)
- 2004년 2월 : 경희대학교 전자계산공학과(박사과정수료)

<관심분야> : AOP, 크로스커팅, 시스템 최적화 마이닝, 리팩토링

송 영 재(Young-Jae Song) 정회원

- 1969년 2월 : 인하대학교 전자공학과(공학사)
- 1976년 2월 : 일본 Keio University 전산학과(공학석사)
- 1979년 2월 : 명지대학교 대학원(공학박사)
- 1982년 ~ 1983년 : 미국 Maryland University 전산학과 연구교수
- 1989년 ~ 1990년 : 일본 Keio University 전산학과 객원교수
- 1976년 ~ 현재 : 경희대학교 전자정보대학 교수

<관심분야> : 컴포넌트웨어, 웹서비스, AOP, 요구공학