대용량 소스코드 시각화기법 연구

서동수†

요 약

프로그램 소스코드는 텍스트를 기반으로 하는 정보이며 동시에 논리 구조를 포함하고 있는 복잡한 구문의 집합체이다. 특히 소스코드의 규모가 수만 라인에 이르는 경우 구조적, 논리적인 복잡함으로 인해 기존의 빅데이터 시각화 기법이 잘 적용되기 힘들다는 문제가 발생한다. 본 논문은 소스코드가 갖는 구조적인 특징을 시각화하는데 있어 필요한 절차를 제안한다. 이를 위해 본 논문은 파싱 과정을 거쳐 생성된 추상구문트리를 대상으로 프로그램의 구조특징을 표현하기 위한 자료형의 정의, 함수간 호출관계를 표현한다. 이들 정보를 바탕으로 제어 정보를 네트워크 형태로 시각화함으로써 모듈의 구조적인 특징을 개괄적으로 살펴볼 수 있는 방법을 제시한다. 본 연구의 결과는 대규모 소프트웨어의 구조적 특징을 이해하거나 변경을 관리하는 효과적인 수단으로 활용할 수 있다.

주제어 : 정보시각화, 소스코드 시각화, 소스코드 복잡도

Visualization Techniques for Massive Source Code

서동수†

ABSTRACT

Program source code is a set of complex syntactic information which are expressed in text forms, and contains complex logical structures. Structural and logical complexity inside source code become barriers in applying visualization techniques shown in traditional big-data approaches when the volume of source code become over ten-thousand lines of code. This paper suggests a procedure for making visualization of structural characteristics in source code. For this purpose, this paper defines internal data structures as well as inter-procedural relationships among functions. The paper also suggests a means of outlining the structural characteristics of source code by visualizing the source codes with network forms The result of the research work can be used as a means of controling and understanding the massive volume of source code.

Keywords :Information visualization, source code visualization, source code complexity

† 정 회 원: 성신여자대학교 교수
논문접수: 2015년 6월 3일, 심사완료: 2015년 6월 9일, 게재확정: 2015년 6월 16일
* 본 논문은 2013년 성신여자대학교 학술연구조성비 지원으로 수행되었음

1. 서 론

현재 인터넷을 통해 유통되는 데이터의 양은 기하급수적으로 늘어나고 있다. IDC는 2020년 1인당 평균 보유 데이터양이 130테라바이트까지 증가할 것으로 예상한 바가 있다[1]. 과거의 정보처리 시스템이 한정된 분량의 데이터를 빠른 시간에 정교하게 처리할 수 있도록 개발된 것이라면 지금의 처리방식은 대규모의 정형 혹은 비정형 데이터를 대상으로 유용한 정보를 추출하는데 무게가 실려 있다[2].

빅 데이터 활성화와 함께 주목받는 기술이 정보시각화다. 정보시각화는 그래픽 요소를 활용하여 사용자에게 더욱 유용한 정보를 추출해내고 표현되도록 형상화 하는 과정이라 정의 된다[3][8]. 정보시각화 기술은 이용하면 사용자는 많은 데이터를 동시에 차별화할 수 있고, 지각적 추론을 가능도록 하며, 감성적 표현을 통해 정보처리 능력을 확장할 수 있다. 정보시각화는 소비재 구매패턴 분석과 같은 소비자패턴 분석에서부터 독감의 유행패턴 분석, 범죄예방 분석에 이르기까지 다양한 분야에 응용되고 있다.

프로그램 소스코드는 실행파일로 컴파일되어 사용되기 이전에 사실은 매우 복잡한 구조를 갖는 텍스트 형태의 자료이다. 상용 소프트웨어의 소스코드가 수만~수십만 라인에 이른 것으로 볼 때 이들은 일종의 빅 데이터로 이해되고, 처리될 수 있다. 그러나 소스코드만의 특수성으로 인해 일반 텍스트 정보의 처리방법이 그대로 적용될 수 없는 몇 가지 문제가 발생한다.

첫째, 빅 데이터는 전통적으로 정형 혹은 비정형 형태의 텍스트 데이터를 대상으로 한다. 소스코드의 경우는 일반적으로 정형적인 범주의 텍스트 정보와는 비교될 수 없을 정도의 복잡한 구문구조를 가진다. 이러한 이유로 정보 해석을 위해서는 구문 분석, 혹은 파싱이라는 별도의 처리를 필요로 한다.

둘째, 프로그램 정보는 구문분석을 통해 얻을 수 도 있지만 다른 부류의 정보, 예를 들면, 제어흐름 정보나 자료흐름 정보와 같은 것은 의미 분석을 통해야만 획득할 수 있다. 이러한 정보는 기존의 통계적 분석, 기계학습을 통한 분석 등의 전통적인 빅 데이터 분석 기법으로는 접근할 수 없는 종류의 정보이다.

본 논문에서는 프로그램 소스코드를 대상으로 이를 시각화하는 과정에서 고려 사항은 무엇인지, 그리고 소스코드를 시각화하는 절차가 기존의 시각과 방법과 비교하여 어떤 차이점을 가져야 하는지에 관해 논한다.

2. 관련 연구

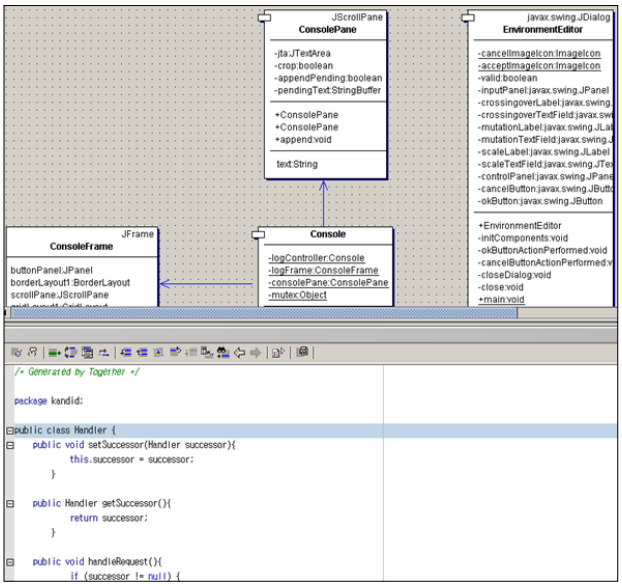
정보시각화 분야와 관련한 소스코드의 시각화 연구는 두 종류의 시도록 구분한다. 첫 째, 독립된 인지과정을 통해 파악된 소스코드 정보를 유의미한 형태로 시각화하는 방법을 연구하는 부류이다. 대표적으로 Servant(2013)는 소스코드를 수정할 때 발생하는 변경 정보를 중심으로 소스코드의 이력을 시각화하였다[4]. 이 방법은 <그림 1>과 같이 막대 차트 형식으로 소스코드의 개정부분을 시각화함으로써 개정이력들 간의 비교가 직관적으로 가능하게 한 사례라 할 수 있다.

<그림 1> Servant의 소스코드 이력의 시각화

Kastner(2008)는 소스코드 시각화를 소프트웨어 프로덕트 라인으로 확장시킨바 있다[5]. 그는 연구에서는 프로덕트 라인에서 소프트웨어를 개발할 때 코드 유닛에 산발적으로 분포되어 있는 특성(feature) 요소를 시각화함으로써 특성의 추적이 가능함을 보여주었다.

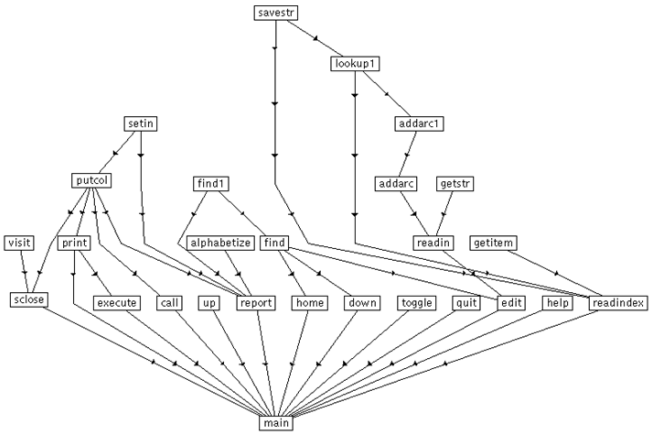
Briand(2006)는 자바 코드로 부터 역공학 과정을 통해 UML 시퀀스 다이어그램을 추출하여 시각화 한 바가 있다[6]. Together, RSA와 같은 많은 UML 도구들에서는 <그림 2>와 같은 형태의

역공학 기능을 제공함으로써 소스코드가 갖는 구조 특징, 특히 클래스의 구조 특징을 UML 형식으로 시각화하였다.



<그림 2> UML 클래스의 역공학 시각화

객체지향 언어인 자바와는 달리 C로 구성된 함수 위주의 프로그램에 있어 함수간의 호출 관계는 프로그램을 이해하는데 있어 중요한 정보이다. Milanova(2007)는 이와 관련하여 함수호출 그래프를 통해 시각화를 시도하였다[7]. 함수호출 그래프는 함수 간의 호출 순서를 구조적으로 보여주는 그래프로 <그림 3>과 같이 함수명을 가진 노드와 호출자와 피호출자 간의 의존 관계를 표시하는 방향 간선의 쌍으로 표현된다.



<그림 3> 함수 호출 그래프 표현

둘째, 소스코드의 시각화 연구 분야로 인지과정에서 발생하는 심미적인 요소를 강조하는 소스코드 시각화가 있다. 대표적으로 Chen(2015)의 시각화 연구를 들 수 있으며, 오퍼레이션 시스템을 구성하는 커널의 호출 분포를 시각화 하여 모듈의 의존 특성을 심미적인 그래프로 표현하였다[8].

<그림 4> 마이크로 커널의 심미적 표현

<그림 4>과 같은 표현은 스타 형식으로 소스코드의 모듈 정보를 표현하는 것으로 모듈의 자세한 정보는 과감히 생략하고 모듈의 길이, 위치를 잘 구조화된 공간에 적절히 배치시킴으로써 모듈의 분포 정보와 더불어 시각적인 효과까지 강조한다.

3. 소스코드 시각화 절차

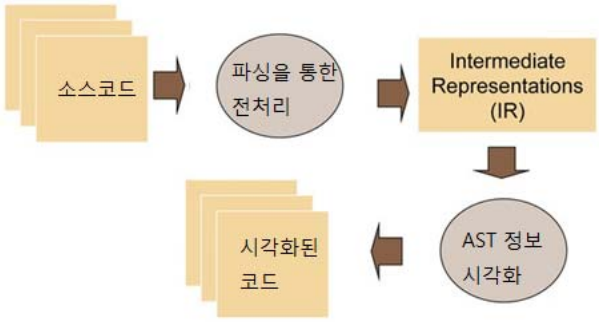
3.1 시각화 단계의 구성

소스코드는 텍스트를 기반으로 하는 정보이며 동시에 논리 구조를 포함하고 있는 복잡한 구조의 집합체이다. 특히 대규모의 소스코드는 파생되는 함수호출로 인해 그 복잡성이 더욱 심해진다.

소스코드를 대상으로 하는 시각화 기법이 Kim(2014)과 Silic(2010) 등에서 소개된 전통적인 텍스트 데이터의 시각화 방법과 차별되는 가장 큰 점은 정적분석을 선행 작업으로 포함한다는 점이다[10][11]. 정적분석(static analysis)이란 실제 코드의 실행을 통하지 않고 프로그램의 특성을 얻고자 하는 자동 혹은 수동의 기법을 말한다. 컴파일러는 중간코드의 최적화를 위해 정적분석 기법을 사용하지만 그 외에도 프로그램 구조의 이해, 프로그램 오류의 검출의 목적으로도 정적기

법이 사용되며, 최근에는 보안 취약점 탐지나 코드 품질 메트릭 추출 등에도 적용되고 있다[12].

소스코드의 시각화를 위한 전처리 과정은 다음의 두 단계로 구성이 된다.



<그림 5> 소스코드 시각화 단계

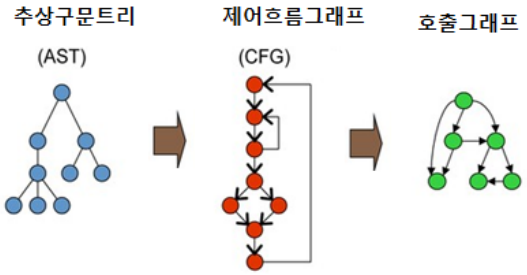
첫 번째 단계는 <그림 5>와 같이 소스코드에 포함되어 있는 유용한 프로그램 정보를 추출하는 전처리 단계로 파서를 이용하여 구문분석을 수행한다. 중간표현(Intermediate Representation)이라 불리는 내부구조가 구문 분석의 결과로 얻어지며, 중간표현은 프로그램 코드 내 식별자 정보를 담고 있는 심볼 테이블, 컴파일러의 전처리 과정을 거쳐 추출된 정보들을 트리 형태로 보관하는 추상구문트리(Abstract Syntax Tree, AST)를 포함한다.

두 번째 단계는 IR 요소 중 추상구문트리로 대표되는 구문정보를 읽어 제어흐름그래프로 변환하고 이 정보를 바탕으로 한 공간배치와 컬러링을 하는 시각화 단계로 구성된다.

이 과정은 <그림 6>에서와 같이 각 단계별로 산출되는 정보들의 변환 관계로 설명이 된다. 즉 전처리를 통해 생성된 추상구문트리 정보로부터 제어흐름 그래프와 호출 그래프가 유도되고, 최종적으로는 이들 제어흐름 그래프와 호출 그래프가 시각화 처리 과정의 주요 입력으로 사용됨을 보여준다.

3.2 파싱을 통한 전처리

파싱(parsing)이란 소스코드를 예약어, 식별자, 연산자 등과 같이 토큰으로 분리된 요소들을 대상으로 언어 문법에 맞는지를 확인하는 작업이다.



<그림 6> 전처리 과정의 구성

이 작업을 수행하는 파서는 프로그램의 구조 특징을 표현하기위해 추상구문트리와 심볼 테이블을 이용한다.

3.2.1 추상구문트리

추상구문트리는 내부 자료구조의 한 형태로 프로그램의 시작점에서부터 접근 가능한 명령어들을 트리 형태로 보관한다. 추상구문트리는 향후 프로그램의 여러 특성을 찾아내는데 기본으로 사용되는 자료구조이다. 이러한 특성으로는 초기화되지 않은 변수, 도달 할 수 없는 변수, 중복 선언된 변수의 존재 등이 있으며, 이를 파악하기 위한 순행 분석(forward analysis), 혹은 역행 분석(backward analysis)의 대상이 된다.

프로그램이 단순한 텍스트의 집합, 즉 명령어들의 집합이라 한다면 제어 흐름은 이들 명령어를 어떤 순서로 실행시킬 것인가를 결정짓는 중요한 요소이다. 실제 프로그램이 어떤 순서로 수행하게 될지를 판단하는 것은 심볼 테이블, 혹은 추상구문트리만 가지고서는 해결할 수 없다. 제어흐름그래프(Control Flow Graph, CFG)는 순차, 선택, 반복문으로 구성된 프로그램의 제어 구조를 표현하며, 추상구문 트리는 제어흐름 그래프를 생성하는데 직접 사용이 된다.

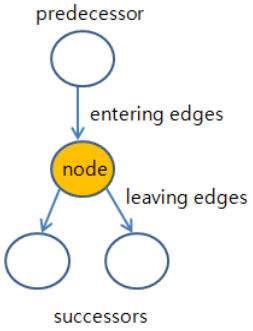
3.2.2 제어흐름 그래프의 구성

제어흐름은 함수내(intra-procedural) 제어흐름과 함수간(inter-procedural) 제어흐름으로 나뉘어 구분된다[13]. 함수간 제어흐름 그래프를 간단히 표현한 것이 함수 호출 그래프이다.

프로그램의 구조를 표현하기 위해서는 추상구문트리로부터 제어의 흐름을 표현하는 제어흐름

그래프를 를 유도하는 것이 필요하다. 이를 위해 고려해야 할 것은 프로그램의 제어구조를 노드(node)와 간선(edge)으로 이루어진 그래프 형태로 표현하는 것이다. 이를 위해 본 논문은 다음의 노드-간선을 클래스로 정의하여 사용한다.

- 노드: 노드는 명령어를 표현하는데 사용된다. 노드에 의해 표현되는 정보는 명령어 종류, 라인 번호, 명령어 내용이다. 하나의 명령어 노드는 선행노드(predecessor node)와 후행 노드(successor node)를 갖는다.



<그림 7> 노드 구성

<그림 7>에서 주목할 점은 노드 정의에 간선 정보가 같이 사용된다는 점이다. 이들 정보가 동시에 표현될 경우 선행노드의 정보를 통해 간선을 추출하고, 다시 간선 정보를 통해 후행 노드를 탐색하는 것이 가능하다는 점이다. 이렇게 함으로써 탐색을 효율을 높일 수 있는 장점이 있다. 본 구현 시스템에서는 노드 표현을 위한 클래스 구조로 다음과 같은 구상의 클래스를 사용한다.

```
public class CNode{
  private final int nodeNumber;
  private final int lineNumber;
  public CNode(int pLineNum, String FName);
  public int getLineNumber();
  public int getNodeNumber();
  public void addLeavingEdge(CEdge pEdge);
  public void removeLeavingEdge(CEdge pEdge);
  public int getNumLeavingEdges();
  public CEdge getLeavingEdge(int pIndex);
  public void addEnteringEdge(CEdge pEdge);
  public void removeEnteringEdge(CEdge pEdge);
  public int getNumEnteringEdges();
 이하 생략..
...}
```

선행노드와 후행노드는 클래스로 정의되지만 이들의 관계는 테이블로도 정의가 가능하다. 예로서 다음의 코드 조각과 관련된 선행, 후행 노드의 내용은 <표 1>과 같은 테이블로도 표현된다.

```
10  middle=(i+count)/2;
11  if(search<a[middle])
12      count = middle;
13  else
14      count = low;
15 ...
```

<표 1> 선행노드와 후행노드 관계

노드번호	선행노드	후행노드	라벨
10	9	11	middle=(i+count)/2;
11	10	12, 14	if(search<a[middle])
12	11	15	count = middle;
14	11	15	count = low;

- 간선: 명령어와 명령어 사이의 연결 관계 나타내는 것으로 제어의 흐름이 간선에 의해 표현된다. 간선은 <그림 7>과 같이 노드를 중심으로 진입간선(entering edge)과 진출간선(leaving edge)으로 구분된다. 다음은 간선 클래스의 구성이다.

```
public class CEdge {
  private final CNode predecessor;
  private final CNode successor;
  private final int lineNumber
  public CEdge(String pStamt, int pLNumber,
     CNode pPred, CNode pSucc);
  public CNode getPredecessor();
  public CNode getSuccessor();
  public CEdgeType getEdgeType();
  public Optional<?> getRawAST();
  public int getLineNumber();
  public String getRawStatement(); ..
 이하 생략.. }
```

3.3 제어흐름그래프의 시각화

3.3.1 제어흐름 그래프의 구성

제어흐름 그래프는 프로그램의 제어구조를 보여줄 수 있는 핵심 정보로 코드시각화의 대상이다. 제어 흐름도를 구성하기 위해서는 전처리 과정을 통해 구성한 노드, 간선 테이블을 참조하여

구성할 수 있다.

다음은 CFG를 구축하기 위한 알고리즘이다.

```
입력: 명령어 블럭 리스트 m
출력: 노드가 기본블럭인 CFG
for i = 1 to m
  x = last instruction of block[i]
  if instr x is a branch
      for each target (to block j) of instr x
     create an edge from block i to block j
  if instr x is not an unconditional branch
    create an edge from block i to block i+1
```

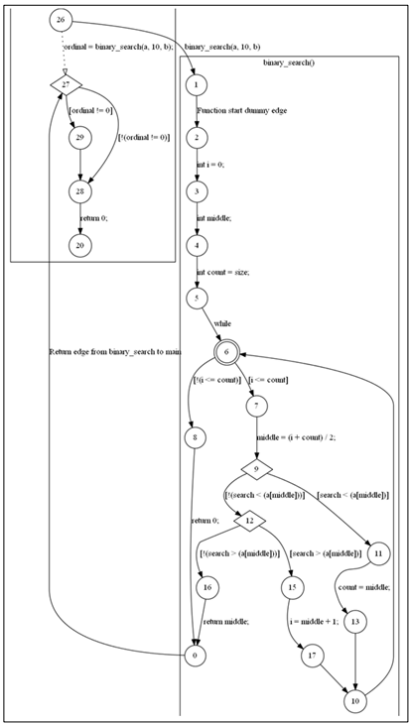
알고리즘에 의해 얻어진 제어흐름 그래프는 향후 수행되는 시각화 처리를 위해 (노드, 간선, 라벨)의 쌍으로 표현한다. 예로서, 프로그램 binary_search를 입력 받아 구성된 CFG는 <표 2>와 같은 (노드, 간선, 라벨) 형식으로 표현된다.

```
int binary_search(int a[], int size, int search) {
 int i=0, middle, count=size;
 while(i<=count) {
  middle=(i+count)/2;
  if(search<a[middle])
   count = middle;
  else if(search>a[middle])
   i=middle+1;
  else return middle;
 }
 return 0;
}
```

<표 2> binary_searc의 노드 간선 표현

```
1 -> 2 [label="Function start dummy edge"]
2 -> 3 [label="int i = 0;"]
3 -> 4 [label="int middle;"]
4 -> 5 [label="int count = size;"]
5 -> 6 [label="while"]
6 -> 7 [label="[i <= count]"]
6 -> 8 [label="[!(i <= count)]"]
8 -> 0 [label="return 0;"]
7 -> 9 [label="middle = (i + count) / 2;"]
9 -> 11 [label="[search < (a[middle])]"]
9 -> 12 [label="[!(search < (a[middle]))]"]
12 -> 15 [label="[search > (a[middle])]"]
이하 생략
```

이러한 노드 간선 리스트를 공개 소프트웨어인 Graphviz를 통해 시각화 하면 <그림 8>과 같은 결과를 얻을 수 있다.



<그림 8> 제어흐름그래프의 예

3.3.2 제어흐름그래프의 공간배열

제어흐름그래프는 명령어 간의 제어 흐름 순서를 시각화한다는 점에서 프로그램 구조를 이해하는 좋은 수단으로 활용될 수 있다. 그러나 시각화라는 측면에서는 다음 사항이 개선되어야 한다. 첫째, 그래프의 크기에 관한 문제이다. 제어흐름그래프가 단순히 함수내 제어흐름을 보여 준다면 시각화에는 별 문제가 없다. 그러나 수백라인이 넘어가며 함수 호출의 복잡도로 인해 제한된 공간 내에서 함수간의 흐름정보를 요약해서 보여주기 쉽지 않게 된다.

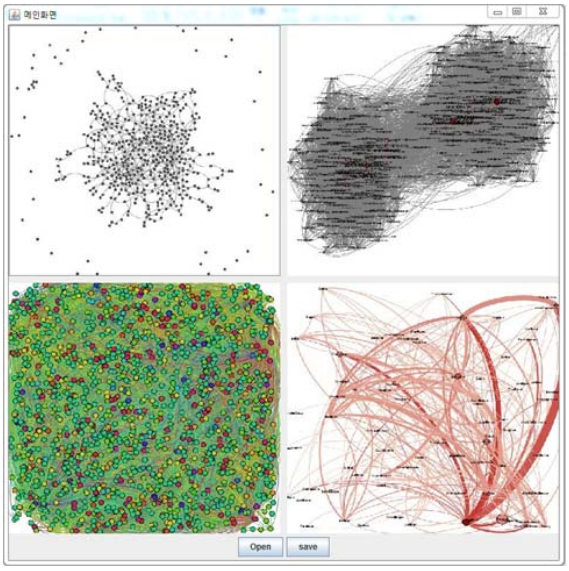
둘째, 명령어의 구분이 필요하다는 점이다. 명령어의 중요도는 제어흐름에서의 역할, 혹은 잠재적인 오류의 정도에 따라 구분되어야 좀 더 정확한 정보를 전해줄 수 있다.

셋째, 시각화과정에서 심미적인 요소가 전혀 고려되지 않고 있다. 사람들에게 친근한 조형원리가 반영된 시각화 결과물은 정보의 이해를 용이하게 하고 만족도를 높여줄 수 있다.

본 연구에서는 심미적인 시각적 배열을 위해 공개 소프트웨어인 Gephi[14]의 공간 배열 알고리즘을 채용하여 설계에 반영하였다.

4. 시각화 적용 사례분석

본 연구에서는 438 LOC 규모의 자바언어로 코딩된 소스코드를 대상으로 시각화를 하였다. 실험은 <그림 9>와 같은 시각화 도구 인터페이스를 설계하여 대상 파일을 선택하면 소스코드의 시각화된 결과를 볼 수 있게 구현하였다. 출력은 PDF 형식으로 제공함으로써 결과물을 다양한 해상도에서 손실 없이 볼 수 있도록 설계하였다.

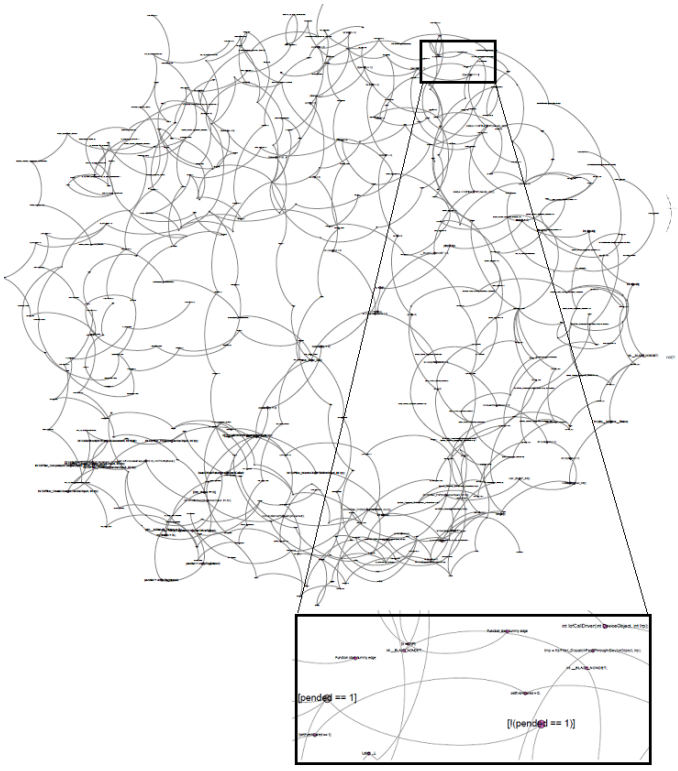


<그림 9> 시각화도구의 GUI

소스코드 시각화기의 작동 과정은 다음과 같다. 먼저 선택된 소스파일은 이클립스 파서의 전처리 기능을 통해 추상구문트리로, 이는 다시 제어흐름 그래프로 변환되어 DOT형식의 외부 파일로 저장된다. DOT형식의 출력물은 본 연구를 통해 구현된 시각화도구의 사용자 인터페이스 <그림 9>를 통해 XML 형태로 변환되어 공간 배치 알고리즘을 적용하여 다음과 같은 내용의 PDF파일로 출력한다.

본 연구를 통해 구현된 자바 소스코드 시각화 도구는 다음 정보를 심미적으로 시각화함으로써 사용자에게 다음과 같은 도움을 줄 수 있다.

- McCabe의 복잡도에 직접 영향을 주는 요소인 If문의 분포관계를 직관적으로 알 수 있다. 이들은 매듭(knot)형태로 표현되며, 자주 수행되는 If문의 라벨을 <그림 10>의 박스처럼 볼드 타입으로 강조함으로써 쉽게 인지할 수 있도록 한다.
- 복잡한 논리구조의 분포를 한 눈에 나타나도록 한다. 프로그램 내의 간선들이 조밀하게 분포된 부분은 에러가 발생할 가능성이 높은 부분이다. 아래 결과의 경우 원형을 중심으로 상단 과 하단 부분이 이에 속하는 경우이며, 이를 통해 프로그램의 변경과 유지 보수에 중점을 두어야 할 부분을 인식할 수 있다.



<그림 10> 샘플 자바코드의 시각화 결과

5. 결론 및 제언

소스코드는 빅 데이터로서 텍스트 정보의 집합체이며, 동시에 일반 텍스트 정보와 구분되는 복잡한 논리구조를 갖는다. 이러한 이유로 소스코드를 시각화하기 위해서는 기존의 빅 데이터 분석에서 사용하던 통계적 기법, 기계학습 기법과는 다른 접근을 해야 할 필요성이 대두된다.

본 연구는 프로그램의 구문과 의미정보를 정적 분석 기법을 활용하여 추출하고, 이를 정보시각화

에 사용하는 방법을 도출하였다. 본 연구가 기존의 소스코드 시각화와는 차별화될 수 있는 점은 복잡도가 높아지는 구간의 표현, 조건문 분기 정보의 표현을 통해 프로그램의 거시적은 특징의 표현, 그리고 이들 정보를 심미적인 측면에서 시각화하기 위한 공간배치를 했다는 점이다.

연구 결과의 활용 측면에서 본 연구가 갖는 시사점은 다음과 같다. 첫째, 대규모 프로그램의 복잡도 특징을 시각화함으로써 개발자들은 바람직한 프로그램 구조에 대한 통찰을 가능하게 한다. 둘째, 유지보수 담당자는 기존의 소스코드의 시각화를 통해 버전별 특성의 변화, 유지 보수의 주안점을 두어야 할 부분에 관해 직관적인 이해를 할 수 있다. 향후 본 연구의 결과는 이클립스와 같은 공개 소프트웨어의 플러그인으로 활용할 수 있도록 확장할 계획이다.

참 고 문 헌

[1] 이진형(2012). **데이터 빅뱅, 빅 데이터(BIG DATA)의 동향.** 방송통신전파저널 (47). 43-55

[2] 한국정보화진흥원(2013). **새로운 미래를 여는 빅데이터 시대.** 한국정보화진흥원 빅데이타 연구센터

[3] 신화숙, 임정묵, 박준석(2013). **정보 시각화 기술과 시각장애인을 위한 정보표현 기술.** 전자통신동향분석. 한국전자통신연구원. 28(1). 81-91

[4] Servant, F, Jones, J.(2013). *Chronos: Visualizing Slice of Source Code History.* IEEE Working Conference on Software Visualization (VISSOFT 2013). 1-4

[5] Kastner C. et al (2008) *Visualizing Software Product Line Variability in Source Code.* Proc. SPLC Workshop on Visualization in Software Product Line Engineering, 303 - 313

[6] Briand, C. et al(2006). *Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software.* IEEE Trans on Software Engineering 32(9). 642-663

[7] Milanova, A, Rountev A, Ryder B(2004), *Precise and Efficient Call Graph Construction for C programs with Function Pointers.* Automated Software Engineering 11(1), 7-26

[8] Chen, F et al,(2015), *Discovering and Visualizing Operations Processes with POD-Discovery and POD-Viz.* International Conference on Dependable Systems and Networks (DSN), Rio de Janeiro

[9] Ware, C(2013), *Information Visualization: Perception for Design.* Morgan Kaufmann Press

[10] Kim, H., Park, J, Seo D (2014), *Readability Visualization for Massive Text Data*, International Journal of Multimedia and Ubiquitous Engineering. 9(9). 241-248

[11] Silic A.(2010), *Basic Visualization of Text Streams: A Survey.* 14th International Conference, KES, Cardiff, UK,

[12] Chess, B., West, J.(2007), S*ecure Programming with Static Analysis*, Addison Wesley

[13] Khedker, U., Sanyal, A., Karkare, B.(2009) *Data Flow Analysis: Theory and Practice.* CRC Press

[14] Bastian, M., Heymann, S., Jacomy, M.(2009) Gephi: A*n Open Source Software for Exploring and Manipulation Networks.* ICWSM, (8) 361-362

[15] Ball, T, Eick, S.(1996). *Software Visualization in the Large*. Computer, 29(4) 33-43

서 동 수

1987 중앙대학교 컴퓨터공학과(이학사)
1990 Univ. of Manchester (전산학 석사)
1994 Univ. of Manchester (전산학 박사)
1998~현재 성신여자대학교 IT학부 교수
관심분야: 소프트웨어공학, 정보보호
E-Mail: dseo@sungshin.ac.kr