메타 정보와 Graphviz를 이용한 흐름도 자동 생성 도구 구현

천준석*, 이기화*, 우균*
*부산대학교 컴퓨터공학과
e-mail: {rance18, dlrlghkekk, woogyun}@pusan.ac.kr*

Implementation of the Flowchart Auto Generator Based on Metadata and Graphviz

Joonseok Chun*, Kihwa Lee*, Gyun Woo*
*Dept of Computer Engineering, Pusan National University

요 약

컴퓨터의 발달로 소프트웨어의 규모가 커지면서 유지 보수가 어려워지고 있다. 프로그램 유지보수를 위한 방법 중 하나로 흐름도를 작성하는 것을 들 수 있다. 흐름도를 작성하는 방법에는 개발자가 수작업으로 작성하는 방법이 있고, 자동 생성 도구를 사용하는 방법이 있다. 수작업은 시간이 오래 걸리며, 수정이 힘들지만 원하는 정보를 적확하게 표현할 수 있다. 한편, 자동 생성 도구로 만들어진 흐름도는 빠르고 정확하게 생성되지만 원래 코드 파악이 어렵다. 이 논문에서는 개발자가 원하는 정보가 포함된 흐름도를 빠르고 정확하게 생성하기 위하여 메타 정보와 Graphviz 라이브러리를 이용하여 흐름도를 생성한다. 본 연구 결과를 바탕으로 다른 프로그래밍 언어에 대해서도 흐름도를 생성할 수 있다면 소프트웨어 유지보수성을 높이고 나아가 소프트웨어 품질 향상에 기여할 수 있을 것으로 기대된다.

1. 서론

오늘날의 컴퓨터는 과거에 비하여 계산 용량과 문제의 복잡성이 급격히 증가하였다. 따라서 하드웨어의 성능을 충분히 발휘할 수 있는 소프트웨어 개발이 요구되었다. 하지만 소프트웨어가 복잡해짐으로 인해서 개발 예산 초과, 일정 지연, 소프트웨어 품질 저하, 유지 및 보수의 어려움 등의 문제를 해결해야 하는 상황에 이르게 되었는데, 이를 흔히 소프트웨어 위기(Software Crisis)라 한다[1].

소프트웨어 유지보수 문제는 자신이 작성한 코드보다 다른 사람이 작성한 코드일 때 더욱 심하게 나타난다. 이를 해결하기 위해 수작업 혹은 흐름도 자동 생성 도구를 이용하여 흐름도(Flowchart)를 작성하는 방법이 있다[2]. 수작업의 경우에는 개발자가 흐름도를 일일이 그려야 하기 때문에 시간이 오래 걸리며, 오류가 발생하기 쉽다. 또한, 많은 수정이 필요하여 전체적인 레이아웃이 크게 바뀔 때 수정이 힘든 상황이 발생하기 쉽다. 한편 소스 코드를 입력받아 자동으로 흐름도를 생성해주는 도구를 사용하면 흐름도를 쉽고 빠르게 생성할 수 있다. 하지만 개발자가 원하는 모양으로 출력되지 않으며, 직관적이지 못한 단점이 있다.

이 논문에서는 메타 태그와 Graphviz 라이브러리를 이용한 흐름도 생성 도구를 제안한다. 이 논문은 2장에서는 현재 출시된 흐름도 자동 생성 도구를 소개하고, 3장에서는 제안하는 시스템의 구조를 설계하였으며, 4장에서는 3장에서 제안한 흐름도 자동 생성 도구를 구현하고, 5장에서 결론을 맺는다.

2. 관련 연구

소스 코드를 이용하여 흐름도를 생성하는 그래프 시각화 도구 중의 하나로 Visustin v6 Flow chart generator[3]가 있다. 이 도구는 입력받은 소스 코드를 자동으로 흐름도로 변환해주는데, 반복 구문은 분기문 형식으로 바꾸어서 출력되며, 흐름도 노드(node)에 소스 코드가 그대로 출력된다는 단점이 있다. 또한, 메인 함수 이외의 함수는 출력되지 않는다.

Microsoft Visual Studio 2010에서는 작성한 코드를 통하여 손쉽게 시퀀스 다이어그램을 생성할 수 있다[4]. 그러나 흐름도를 생성해주지는 않으며, Ultimate 버전에서만 이 기능을 지원한다는 문제점이 있다.

Eclipse Control Flow Graph Generator[5]는 Eclipse IDE에서 플러그인 형식으로 사용할 수 있는 흐름도 생성기이다. 이 플러그인은 모든 노드가 사각형으로 되어 있으며, 노드의 종류는 색깔로 구분되며, 메소드 단위로 흐름도를 생성할 수 있다. 하지만 생성된 흐름도의 정보가 부족해서 흐름도만으로 코드가 어떻게 동작하는지 확인하기 어렵다.

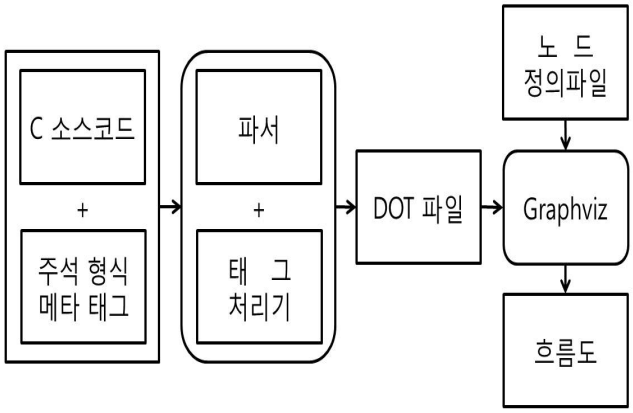
앞에서 살펴본 자동 흐름도 생성 도구는 생성된 흐름도만으로는 원래 소스 코드를 파악하기 어렵다는 단점이 있다. 따라서 이 논문에서는 흐름도를 생성할 때 개발자가 직접 메타 정보를 기재하여 좀 더 명확한 흐름도를 생성

"본 연구는 LG전자 산학연구과제의 일환으로 수행하였음. [C2012018737, 소스 코드를 이용한 Flowchart 자동화 도구]"

하고자 한다.

3. 시스템 설계

우선 시스템을 구성하기에 앞서 어떤 언어를 대상으로 구현할 것인지에 대하여 결정해야 한다. 현재 프로그래밍에 사용되는 언어는 많이 존재한다. 이 중에서 현재 점유율이 높은 언어에는 C, Java, Objective-C, C#이 있다[6]. 이 중에서 C 언어는 저급언어의 효율성과 고급언어의 생산성을 균형 있게 가지고 있는 언어이기 때문에 임베디드, OS, 수학 연산 등의 많은 분야에서 쓰인다. 따라서 이 논문에서는 C 언어를 대상으로 함수 단위의 흐름도 생성을 위한 시스템을 구성한다.



(그림 1) 시스템 구성도

그림 1과 같이 C 소스 코드에 주석 형식의 메타 태그를 추가한다. 이후 파서에서 코드 구문 분석을 통해 소스코드에 맞는 DOT 파일을 자동 생성한다. 이 DOT 파일과 노드 정의 파일, Graphviz 라이브러리를 사용하여 흐름도를 생성한다.

Graphviz는 DOT 언어로 작성된 텍스트를 분석하여 자동으로 그래프를 생성하는 도구이다[7]. Graphviz는 dot, neato, twopi, circo, fdp 등의 그래프 프로세서를 사용하여 이미지 형식의 그래프 출력 파일을 생성한다[8]. 위의 프로세서 중 dot과 neato가 주로 사용되는데, dot은 계층 구조(Hierarchical Layout)의 그래프를 생성 시 사용되며, neato는 대칭적 구조(Symmetric Layout)의 그래프를 생성 시 사용한다[9].

메타 태그는 부가적인 정보를 얻기 위해 삽입하는 정보로서 외줄 주석인 // 뒤에 역 따옴표(back quote) 두 개를 사용하여 그 사이에 한글 및 영어 메타 정보가 들어가는 형식으로 구성한다. 역 따옴표는 C 언어에서 사용되지 않는 특수문자[10]이기 때문에 메타 태그로 사용한다. 표 1은 메타 태그를 사용한 C 코드를 나타낸다. 변수일 경우에는 변수 옆에 메타 태그를 붙이는 것을 허용하였고, 그 이외에는 해당 구문 위에 메타 태그를 붙이도록 지정한다.

<표 2> 메타 태그가 포함된 C 코드

```
//`이진탐색트리 탐색`
treenode* searchBST(treenode* root, char x) {
    treeNode* p;
    p = root;
    //`현재 탐색노드의 하위 노드가 있는가?`
    while (p != NULL) {
        //`찾는 값이 현재 노드보다 작은가?`
        if (x < p->key)
            //`왼쪽 서브트리에서 탐색`
            p = p->left;
        //`찾는 값이 현재 노드보다 큰가?`
        else if (x > p->key)
            //`오른쪽 서브트리에서 탐색`
            p = p->right;
        //`찾는 값이 현재 노드인가?`
        else
            //`탐색 성공`
            return p;
    }
    //`찾는 노드가 없음`
    printf("\n찾는 노드가 없습니다!");
    //`탐색 실패`
    return p;
}
```

파서는 OpenC++[11]를 변경하여 구현하였다. OpenC++ 파서는 표 1과 같은 C 코드를 읽어온 후 메타 태그를 수집하여 추상구문트리(Abstract Syntax Tree)를 생성한다. 다음으로 생성된 추상구문트리를 탐색하여 분기 등의 요소에 해당하는 구문 중 메타 정보화 된 구문을 파악하여 표 2와 같은 DOT 파일을 생성한다.

<표 3> 구문 분석을 통해 생성된 DOT 코드

```
digraph test {
  node[fontname="HYSMyeongJo-Medium--KSC-EUC-H"]
  edge[fontname="HYSMyeongJo-Medium--KSC-EUC-H"]
  "이진탐색트리 탐색"[shape=invhouse]
  "현재 탐색 노드의 하위 노드가 있는가?"[shape=diamond]
  "찾는 노드가 없음"[shape=sdl_call]
  "탐색 실패"[shape=box]
  "찾는 값이 현재 노드보다 작은가?"[shape=diamond]
  "왼쪽 서브트리에서 탐색"[shape=box]
  "찾는 값이 현재 노드보다 큰가?"[shape=diamond]
  ...
  "이진탐색트리 탐색" -> "현재 탐색 노드의 하위 노드가 있는가?";
  "현재 탐색 노드의 하위 노드가 있는가?" -> "찾는 노드가 없음"[label="거짓"];
  "찾는 노드가 없음" -> "탐색 실패";
  "찾는 값이 현재 노드보다 작은가?" -> "왼쪽 서브트리에서 탐색"[label="참"];
  "찾는 값이 현재 노드보다 작은가?" -> "찾는 값이 현재 노드보다 큰가?"[label="거짓"];
  ...
  "찾는 값이 현재 노드인가?" -> "탐색 성공"[label="참"];
  {"탐색 성공", "탐색 실패"} -> "이진탐색트리 탐색 종료";
  {rank=same;"현재 탐색 노드의 하위 노드가 있는가?";"찾는 값이 현재 노드보다 작은가?"}
}
```

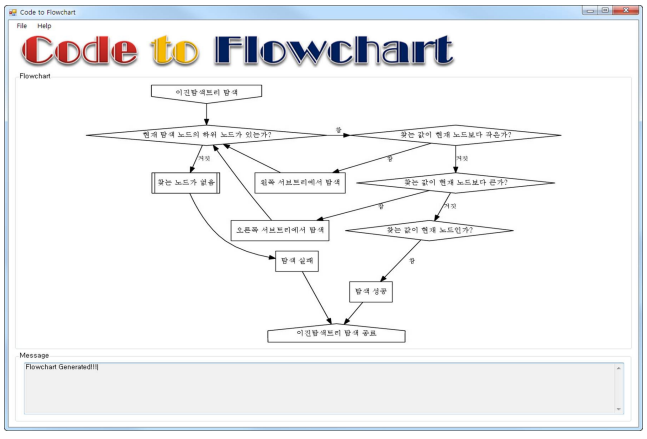
다음으로 표 3과 같이 분기문(Branch), 선택 수행문(Switch-case), 수행문(Statement), 반복문(Loop), 함수 호출(Function Call) 구문 등 흐름도를 구성하게 될 구문별 노드 모양을 정의한다. 흐름도는 함수 시작 노드에서 시작하여 함수 종료 노드에서 끝난다. 이후 함수 내에 정의된 메타 태그를 이용하여 함수에 대한 흐름도가 생성된다. 함수 호출을 클릭하면 해당 함수의 흐름도를 보여준다.

<표 4> 구문별 노드 모양

구 문	노드 표현	구 문	노드 표현
함 수 시 작	start	함 수 종 료	end
수행문	statement	함 수 호 출	f_call
분기문	if, if_stmt, else, else_stmt	선 택 수행문	switch, case1, case2, case3
		반복문	loop_cond, stmt

4. 시스템 구현

이 논문에서 제안한 흐름도 자동 생성 도구 구현을 위해 .NET Framework 3.5 기반 환경에서 시스템을 구현하였다. 개발 언어는 C#이며, 흐름도를 출력하기 위해 Graphviz 라이브러리를 사용하였다. 시스템은 크게 흐름도를 보여주는 부분과 각종 메시지를 출력해주는 부분으로 나뉜다. 메시지 출력 부분에는 흐름도 자동 생성 시 진행상황, 구문 분석 및 흐름도 생성 시의 오류 메시지를 출력해준다.



(그림 2) 흐름도 자동 생성 도구 실행 화면

소스 코드에 정의된 메타 태그를 파서가 분석하여 생성된 DOT 파일을 Graphviz 라이브러리를 통해 생성된 최종 흐름도는 그림 2와 같다. 최종 생성된 흐름도는 기존의 자동 흐름도 생성기보다 소스 코드 흐름을 파악하기 쉽다. 또한, 수작업보다 흐름도를 빠르게 생성할 수 있다.

5. 결 론

이 논문은 흐름도 작성에 드는 시간과 비용을 절약하기 위한 흐름도 작성 자동화 시스템을 구성하는 것에 목표를 두었다. 구현된 시스템은 흐름도를 정확하고 빠르게 생성하였다. 또한 흐름도에 필요한 정보를 개발자가 메타 정보로 추가하였기 때문에 기존의 자동 흐름도 생성 도구보다 알아보기 쉽다. 마지막으로 기존의 자동 흐름도 생성기가 구문별로 노드 차이가 분명하지 않았던 문제를 해결하기 위해 별도의 구문별 노드 모양을 정의하였다.

이 논문에서 제안하는 흐름도 자동 생성 도구를 이용하면 소프트웨어 유지 및 보수에 드는 시간과 노력을 줄일 수 있다. 개발자는 코드에 메타 정보를 기입하는 것만으로 흐름도를 쉽게 생성해낼 수 있다. 본 연구 결과를 바탕으로 다른 프로그래밍 언어에 대해서도 흐름도를 생성할 수 있다면 소프트웨어 품질 향상에 기여할 수 있을 것이다.

참고문헌

[1] B. Randell, “History of Software Engineering”, The 1968/69 MATO Software Engineering Reports, 1996.

[2] L.D. Landis, P.M. Hyland, A.L. Gilbert, A.J. Fine, “Documentation in a software maintenance environment”, IEEE, 1988.

[3] Aivosto, http://www.aivosto.com/, 2012년 8월 12일 방문.

[4] MSDN, http://msdn.microsoft.com/ko-kr/library/dd409377.aspx, 2012년 10월 2일 방문.

[5] Aldi Alimucaj, http://eclipsefcg.sourceforge.net/, 2012년 10월 3일 방문.

[6] TIOBE Programming Community Index for September 2012, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, 2012년 10월 2일 방문.

[7] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, GordonWoodhull “Graphviz - Open Source Graph Drawing Tools”, Graph Drawing p 483-484, 2002.

[8] Emden R. Gansner, “Drawing graphs with Graphviz”, Graphviz, 2011.

[9] Emden Gansner, Eleftherios Koutsofios, Stephen North “Drawing graphs with dot”, 2006.

[10] CER International bv, FieldCommander JavaScript Reference Guide, p47, 2008.

[11] OpenC++, http://opencxx.sourceforge.net/, 2012년 9월 10일 방문.