소프트웨어 정적 분석의 가시적 표현 모델

송승화 김윤관 장천현
건국대학교 컴퓨터·정보통신 공학과
sshtel@konkuk.ac.kr, apostlez@konkuk.ac.kr, chchang@konkuk.ac.kr

Graphical Presentation Model for Static Analysis of Software

Seung-Hwa Song Yun-Kwan Kim Chun-Hyon Chang
Dept. of Computer Science and Engineering , Konkuk University

요 약

오늘날 소프트웨어의 설계 및 개발과 관련된 연구들은 많은 발전을 이루고, UML과 같은 의사소통을 위한 표준 언어가 만들어졌으며 많은 사람들이 관련 이론을 수용 및 활용하고 있다. 또한, 개발 프로세스에서의 설계 및 구현과 더불어 소프트웨어의 유지 보수 단계는 매우 중요하며 이러한 소프트웨어의 유지 보수를 위한 소프트웨어 검증에 관련된 연구는 근래에 많은 주목을 받고 있다. 하지만 소프트웨어 검증의 기술 개발 수준은 설계 및 개발에 비하여 많이 미흡한 실정이다. 소프트웨어 검증은 주로 검증하기 위한 입력 데이터를 산출하여 프로그램 실행을 통해 결과를 확인하는 동적 분석에 대한 연구가 주를 이룬다. 이러한 동적 분석은 원하는 산출물의 확인을 주목적으로 하며, 결과를 표현하는 데에는 따로 정형화된 형식이 필요 없다. 하지만 소스코드를 분석하여 소프트웨어의 구조 관계와 흐름을 파악하는 정적 분석은 분석 자료를 표현하는 표현 모델이 중요하다. 현재는 정적 분석의 결과를 표현하기 위한 공통의 표현 모델이 없기 때문에 검증 과정에서의 의사소통에서 의견의 불일치의 가능성이 있고, 설계 단계에서 사용되는 표현 모델로는 정적 분석 정보의 모든 내용을 표현하는데 한계가 있다. 따라서 본 논문에서는 소프트웨어의 정적 분석 과정을 분석 4계층으로 구분하고, 각 계층마다 분석 결과를 나타내기 위한 표현 모델을 제시한다. 그리고 이 표현 모델을 활용한 소프트웨어 분석 도구의 개발을 위해, 소스 분석 데이터를 가시적으로 표현하기 위한 자료구조의 설계에 대한 내용을 다룬다.

1. 서론

오늘날 소프트웨어 개발은 과거에 비해 그 규모나 복잡성, 사용자의 요구사항, 제품을 시장에 내놓기까지의 개발 기간 감소 요구 등이 현저히 증가하였고, 이를 해결하기 위한 과정에서 높은 비용이 발생한다. 따라서 이러한 요구를 만족시키고 개발 프로세스의 원활한 진행을 위한 소프트웨어의 설계 및 개발과 관련된 많은 연구들과 기술들이 소개 되어 왔다.

개발자는 품질 요구사항을 만족하는 소프트웨어를 개발할 책임이 있다. 그런데 소프트웨어 개발 프로세스의 후반에 발견되어지는 결함은 초기에 발견되어 수정되어지는 것 보다 많은 비용이 들며, 이러한 이유에서 품질관리를 위한 소프트웨어 검증은 중요하다. 현재 설계 및 구현을 위한 RAD tool(Rapid Application Development tool)들이 널리 보급되어 사용되고 있다. 하지만 소프트웨어의 검증을 위한 관련 연구 및 기술 수준은 설계 및 구현에 비하여 많이 미흡한 실정으로써, 원시 코드를 대상으로 실제 수행 시 처리되어지는 수행경로나 변수들의 값의 검사 등의 중요 검사 요소를 확인할 수 있는 마땅한 도구가 거의 없다[1, 7].

소프트웨어 검증은 주로 동적 분석과 관련한 내용들이 많이 연구되어 왔다. 동적 분석은 구현한 프로그램을 바로 실행하면서 원하는 데이터가 산출되는지 여부를 바로 확인할 수 있고 다양한 테스팅 데이터를 시험할 수 있다. 따라서 소프트웨어에 존재할 수 있는 위험요소를 예방하기 좋고, 데이터 오류 검증에 최적화 되어 있다. 하지만 동적 분석으로는 구현된 소프트웨어의 구조가 어떻게 되어 있는지 파악하기 어렵기 때문에 중복되는 로직이나 비효율적 메모리 할당, 구현상의 복잡성으로 인한 성능 저하 등의 결함이 발생하더라도 원인을 파악하기 힘들다. 동적 분석으로는 이러한 현상을 발견해 낼 수 있을 뿐이다. 정적 분석을 통한 테스팅은 소스 코드의 전체적인 구조 및 구성 요소들 간의 연관관계, 그리고 호출 관계들을 분석함으로써 이러한 결함을 찾기 용이하며, 좀 더 가용성이 높고 부하가 적은 소프트웨어로 가공하기 위한 정보를 제공한다.

하지만 이러한 정적 분석의 결과 정보를 표현하는 데는 몇 가지 문제점이 존재한다. 어떤 목적에 따른 특수한 데이터는 개발자들이 서로 다른 의미로 받아들이거나 표기법을 달리 하여 오해의 소지가 있을 수 있다. 또한, 설계 단계에서의 표현 모델로는 분석 정보의 모든 내용을 표현하는데 한계가 있다.

본 논문에서는 이러한 분석 정보를 효율적으로 표현하기 위한 표현 모델을 제시하고, 이 모델에 기반을 둔 도구를 개발하기 위한 가시적 표현 모델 자료구조를 설계하였다. 표현 모델의 제시는 정적 분석 정보를 보는 관점에 따라 분석 정보의 특성에 기반을 둔 것이며, 정적 분석을 통해 도출되는 데이터를

특성별로 분류하여 분석 4계층으로 구분한다. 이를 기반으로 한 정적 분석 도구를 활용하면, 소프트웨어의 검증 단계에서 정적 분석 정보의 대한 일관성 있는 관리와 원활한 협력 작업 환경을 제공할 수 있을 것이다.

2. 관련 연구

2.1. 정적 분석

정적 분석은 프로그램을 컴파일하고 실행하기 전에 소스 코드 수준의 어휘, 구문 분석을 통한 문법 구조를 분석하여 소스코드의 정보를 얻어내는 것을 말한다. 정적 분석을 통해 프로그램의 모든 제어의 경로 상에서 초기치가 잡혀있지 않은 변수나 정의되었지만 사용하지 않는 변수, 또는 중복으로 정의된 변수, 에러를 유발시킬 코드, 타입 별로 문장의 순환횟수 및 실행되지 않는 문장, 사용한 identifier의 cross-reference map, identifier가 각 문장에 어떻게 사용되었는지의 해석, 모듈 간 인터페이스 등의 정보를 얻을 수 있다[8]. 기존의 최악 실행시간 분석을 위한 연구에서는 소스코드 분석기에서 소스코드를 분석하여 함수들 간의 호출 관계 정보, 공유되는 함수들의 정보와 데이터, 사용자가 명시한 시간 정보 등을 추출하고 이 정보들을 기반으로 실행 시간 분석을 위해 프로그램의 흐름 정보와 관계 정보를 정적으로 분석했다[11]. 정적 분석을 위한 또 다른 연구에서는 어휘 분석 및 구문분석을 하여 AST(Abstract Syntax Tree) 정보를 생성하여 실행시간을 계산하는데 활용된다[12]. 이렇게 정적 분석 기법을 통해 소프트웨어의 흐름 정보와 관계 정보를 분석해 낸다.

하지만 현재로서는 이러한 정적 분석의 결과 정보들을 표현하기 위한 마땅한 표현 모델의 기준이 없으며 몇 가지 문제점이 있다.

첫째, 개발자들의 정적 분석 결과에 대한 의사소통의 불일치는 소프트웨어 검증 단계의 지연 현상을 가져오기 때문이다. 현재 많은 종류의 정적 분석 방법론과 분석 도구들이 제시되고 있다. 하지만 표기법이 모두 다르기 때문에 혼란의 여지가 존재한다. 이러한 언어의 불일치 문제를 해결하기 위해서 공통 표현 모델의 개발이 필요하다.

둘째, 설계 과정에서 사용되는 표기법으로는 검증 과정에서 표기해야 하는 많은 정보들을 표기하는데 한계가 있기 때문이다. 표준 표기법인 UML(Unified Modeling Language)은 객체 지향적 사고방식에 기반을 둔 요구사항 분석 및 설계, 구현을 위한 소통 언어이다. 하지만 특정 프로그래밍 언어에 의존적이지 않기 때문에 소프트웨어의 구현된 내용을 자세히 표현하기 어려우며 구현 언어마다 다른 형태의 구조를 표현하는 데에는 한계가 있다. 예를 들어, C나 C++의 공용 함수 및 전역 변수는 객체 지향적 표현 방식인 UML로는 표현이 불가능하다. 따라서 이렇게 다양한 언어의 특징 별 차이점을 모두 표현 가능한 표현 모델이 필요하다.

3. 정적 분석을 위한 표현 모델

소프트웨어를 정적으로 분석하는데 있어 관점의 구분은 중요하다. 개발자나 테스터들이 소프트웨어를 분석할 때의 관점은 다양하다. 클래스를 늘어놓고 그들 간의 관계를 따져 보거나, 함수나 메소드의 호출관계를 따라가기도 한다. 그리고 이러한 관점마다 분석하는 대상은 상황에 따라 다르며, 그 목적 또한 분명히 다르다.

이렇게 정적 분석의 결과를 표현하기 위한 표현 모델을 정의하기 위해서 정적 분석을 통해 도출되는 분석 데이터의 종류 및 특징을 분류하여 표현 모델의 관점을 정리했다.

소프트웨어 정적 분석의 관점은 크게 구조를 파악하기 위한 관계와 특정한 시점을 파악하기 위한 흐름으로 나뉘며, 각각의 관점을 다음 그림 1과 같이 Layer로 구분하였다. Project Layer와 Relation Layer는 관계를, Flow Layer와 Syntax Layer는 흐름을 표현한다.

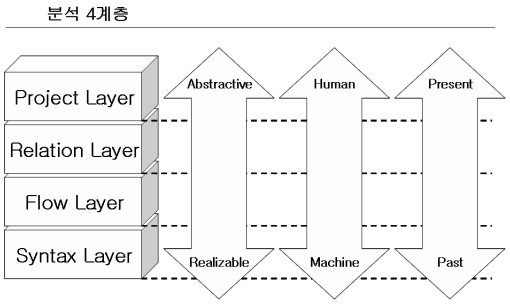


그림 1. Analysis 4 Layer

각 Layer에서 소프트웨어를 분석하는 관점은 다르며, 각 관점마다 다른 형태의 분석 결과를 볼 수 있다. 상위 레이어로 올라갈수록 소프트웨어를 표현하는 개념은 그 범위가 넓어지고 추상적이며, 사용자 입장에 가까워진다. 반대로 하위 레이어로 내려갈수록 들여다보는 관점의 범위는 좁고 구체적이며, 기계어와 가까워진다.

예를 들면 Relation Layer에서 다루는 클래스라는 심벌은 구조적 프로그래밍 언어로부터 확장된 객체 지향의 개념으로서 구조적 프로그래밍 언어의 서브루틴 단위들과 공유 변수를 하나로 묶은 단위로, 좀 더 넓은 개념의 모듈이다. 그에 비하여 함수는 그 표현 범위가 하나의 서브루틴 단위로서 상대적으로 범위가 좁고, 순차적인 구조를 갖는다. 이는 Flow Layer에서 다루는 심벌이다.

3.2. Project Layer

Project Layer는 프로젝트 단위의 요소들을 표현하는 가장 넓은 범위의 계층으로서, 소프트웨어를 가장 넓게 추상화 한 개념이다. 여기서 Project라는 단어를 사용한 것은, 이 논문에서 다루고자 하는 소프트웨어의 의미가 광범위한 범위가 아니기 때문이다. 즉, 사전에 계획된 분명한 목적을 갖고 설계된 하

나의 소프트웨어를 분석하는 입장이기 때문이다.

이 계층에서는 소프트웨어의 운영체제 의존성, 외부 컴포넌트, 라이브러리 의존성 등의 관계를 묘사하기 위한 계층이다. 이러한 요소들은 소스코드에 모두 명시되어 있으며, 정적 분석을 통해 쉽게 파악할 수 있다. 예를 들면, 특정 운영체제에 의존적인 라이브러리나 시스템 함수들을 사용하기 위해 소스 코드에 기술하는 API(Application Programming Interface)들은 그 기능에 대한 정의와 출처가 매우 명확하다. 리눅스의 fork() 같은 시스템 콜 의존적인 함수들은 분명히 잘 알려진 함수이고 이 함수를 사용한 소스는 어떤 환경에 의존하는지를 말해준다.

이 계층에서 보이는 정보들을 통해 소프트웨어의 동작 환경과 외부 환경과의 의존성, 라이브러리 의존성 등을 쉽게 파악할 수 있다.

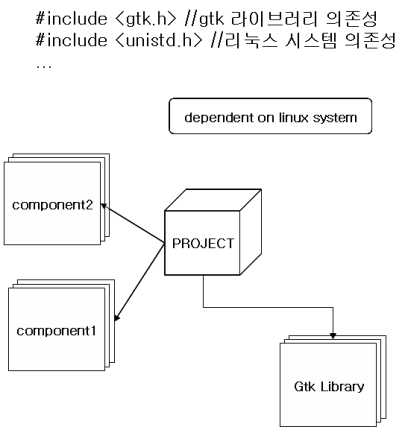


그림 2. Project Layer 분석 모델 예시

3.3. Relation Layer

소프트웨어는 점차 인간의 사고를 반영하기 쉬운 구조로 발전해 왔다. 객체 지향 언어가 그 대표적인 예이다. 언어의 발전과 함께 소프트웨어를 구성하는 각각의 모듈들은 점차 독립성과 캡슐화가 뚜렷해지고 현대의 소프트웨어는 각각의 독립적으로 동작하는 각 모듈들의 짜임새 있는 집합들로 구성되어 있다. 또한, 소프트웨어 규모가 커짐에 따라 모듈은 그 종류뿐만 아니라 개체수도 기하급수적으로 늘어난다. 참조, 의존성, 상속 등의 관계는 소프트웨어 구조의 중요한 요소이다.

Relation Layer는 소프트웨어를 이루고 있는 각각의 크고 작은 모듈들 사이의 관계를 표현 한다. 하나의 프로젝트를 구성하는 컴포넌트, 패키지, 클래스 등의 요소들과 이들 간의 참조 관계, 상속관계, 의존성 관계 등의 관계들을 포함한다.

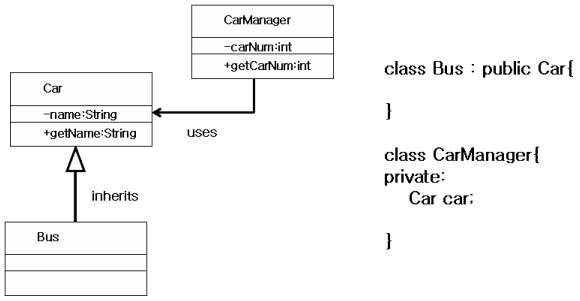


그림 3. Relation Layer 분석 모델 예시

3.4. Flow Layer

소프트웨어의 흐름을 보고자 하는 관점은 분석에 있어 중요하며, 또 가장 많이 사용된다. Flow Layer는 좀 더 구체적으로 소프트웨어의 흐름을 파악하는 계층이며 함수와 메소드 같은 최소 단위의 모듈들의 호출 관계를 표현한다. Relation Layer에서 주로 객체지향의 특징이 많이 반영된 분석 자료를 다룬다면, Flow Layer에서의 결과는 순차적 특징이 많이 보인다.

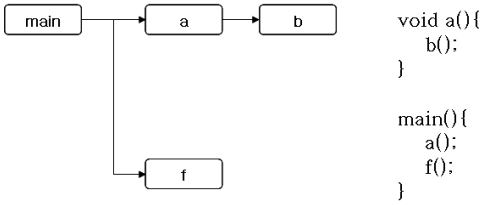


그림 4. Flow Layer 분석 모델 예시

3.5. Syntax Layer

Syntax Layer는 최소 단위의 프로시저의 세부적인 프로그램의 내용을 표현하는데 사용된다. 즉, 개발자가 서브루틴의 기능을 만들기 위해 짜는 프로그램의 문법적인 구조를 분석하여 보여주는 계층이다. 이 계층에서는 그림 5와 같이 소스코드 상에 기술된 문장들의 구문 구조를 보여주며, 이 정보들을 통해 비효율적 문장의 구조를 분석해 내거나 사용되지 않는 변수 정보 등을 보여준다. 또한, 복잡한 수식이나 문장의 순서를 표현해줌으로써 논리적 오류를 검증할 수 있다.

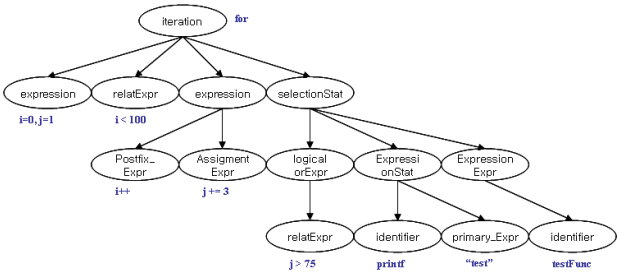


그림 5. Syntax Layer 분석 모델 예시

3.6. 계층 구분의 장점

이렇게 단계적으로 이전의 개념들을 발전시키면서 확장된 소프트웨어의 특성을 바탕으로 계층을 구분함으로써 소프트웨어의 정적 분석을 통해 얻는 데이터를 쉽게 정리 및 관리할 수 있다는 이점이 있다. Project Layer에서 보이는 구성 요소들은 구현된 하나의 소프트웨어 단위와 운영체제, 네트워크, 데이터베이스, 외부 컴포넌트, 라이브러리 등의 자원과의 연관 관계를 표현한다. 따라서 소프트웨어를 구동하기 위한 환경과 필요한 외부 요소들이 무엇인지 등의 여부를 아키텍처 레벨에서 파악이 가능하다. 여기서 하위 계층인 Relation Layer에서는 구현된 하나의 프로젝트에 대한 클래스 간의 관계 정보를 표현하며 전체적인 소프트웨어의 구성을 파악할 수 있다. 이 계층을 구성하는 하나의 요소는 다시 여러 하위 서브루틴들의 흐름들로 구성되어 있다. 이 흐름 정보들은 Flow Layer에서 호출 관계 정보를 보여주며, 다시 이 Flow Layer를 구성하는 서브루틴의 세부 내용은 Syntax Layer에서 그 구조를 보여준다. 이렇게 소프트웨어의 구성 요소들을 각 계층에 두고, 각 요소들의 포함 관계를 정의하면, 분석 과정에서 도출되는 수많은 정보들을 체계적으로 관리할 수 있다.

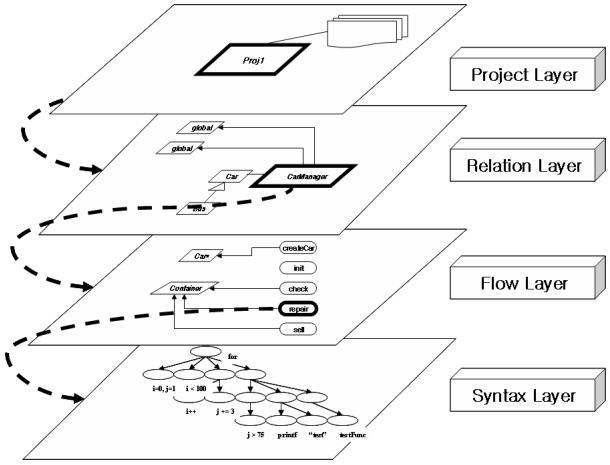


그림 6. 계층 구조

4. 분석 도구 설계

이 장에서는 앞에서 설명한 소프트웨어 분석 모델의 제시 방안을 바탕으로 효율적인 분석 도구를 만들기 위한 설계와 구현 방안에 대하여 언급한다.

4.1. 소프트웨어 분석 정보의 시각화 과정

소프트웨어 정적 분석은 프로그램의 구문 분석부터 시작 된다. 컴파일러 기술을 이용하여 소프트웨어의 심벌들과 관계들을 정리한 뒤, 가시적으로 표현하기 위한 모델 구조에 맞춰 자료구조를 정리한다. 클라이언트 입장에서 이 모델 자료 구조를 이용하여 어플리케이션을 개발할 수 있도록 환경을 제공한다.

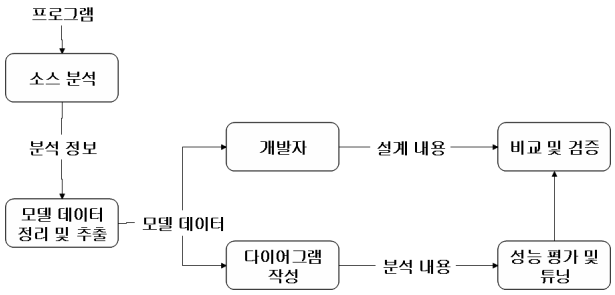


그림 7. 정적 분석 데이터의 시각화 과정

정적 분석은, 소스 코드만으로 분석 정보를 추출하고, 그 정보를 가시적으로 표현함으로써 사용자가 설계 내용과 비교를 하여 구현 여부를 확인하거나 소프트웨어 튜닝을 하기위해 사용된다.

4.2. 정적 분석 도구의 구조

분석 도구의 구조는 소스 분석 모듈, 표현 모델 자료 구조, 어플리케이션 모듈로 나뉜다. 소스 분석 모듈에서 소프트웨어의 심벌과 관계 정보들을 추출해 내게 된다. 여기서 심벌은 소프트웨어를 이루는 구성 요소들을 말한다. 즉 함수, 클래스, 전역 변수, 전역 함수 같은 소프트웨어를 구성하는 서브루틴과 정적 메모리 영역 등이다.

이 정보들을 분리한 다음에는 이를 가시적으로 표현하기 위한 자료구조로 가공해야 한다. 가시적 표현이란 분석 정보를 단지 문자 형식으로 정렬 하여 보여주기 위함이 아니라 다이어그램 형태로 나타내는 것을 말한다. 이렇게 분석 결과 정보를 그림으로 표현하기 위한 자료로 가공하여 화면상에 표현하는 어플리케이션의 개발을 위한 라이브러리를 제공한다.

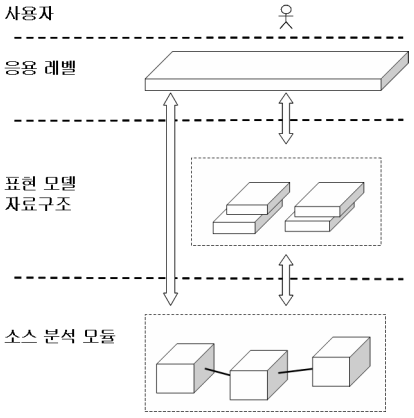
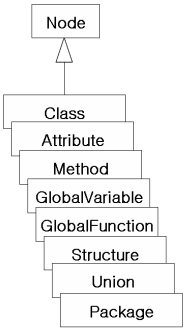


그림 8. 정적 분석 도구 구조

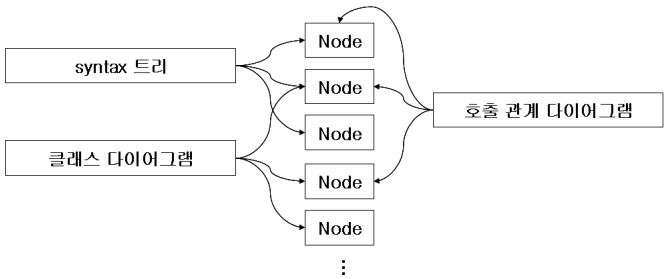
4.3. 표현 모델 자료구조의 설계

표현 모델 자료구조 에서는 소스 분석 정보를 가시적으로 표현하기 좋게 자료를 재정리한 것이며 실제로 이 데이터를 다이어그램으로 표현하기 위해서는 GUI 라이브러리를 활용하여 구현한다. 그런데 GUI 환경의 어플리케이션을 구현하게 되면, 그

환경을 제공하는 운영체제에 의존적일 수밖에 없다. 따라서 이러한 의존성을 없애기 위하여 이 자료구조에서는 다이어그램의 표현 방식과 위치 배열 같은 가시적 정보들은 응용 레벨로 책임을 넘기고, 소스의 구성 요소와 구성 요소 간의 관계 정보에 대한 데이터만을 다룬다. 본문에서는 소스의 분석 정보를 그래프로 나타내기 용이한 자료로 재가공하기 위한 모듈의 설계에 대하여 언급하겠다. 소스의 분석 결과는 모두 소프트웨어를 구성하는 심벌과 그것들 간의 관계를 나타내기 위한 정보로 저장된다. 분석 정보를 그래프로 표현하는 방법은 여러 가지가 존재할 수 있고, 참조하는 정보도 다양하다. 따라서 심벌과 관계 정보들을 일관성 있게 모아 두면 이 정보들을 참조하여 다양한 다이어그램을 구성할 수 있는 유연성을 갖게 된다. 또한, 다양한 언어를 표현하는 데에도 이러한 구조는 매우 유연하다. 각각의 언어들이 표현할 수 있는 심벌과 관계는 다른 부분이 존재 한다. C에서는 함수와 구조체, 전역 변수, 호출 관계 등을 다루지만 C++에서는 클래스와 멤버 변수, 메소드도 같이 다룬다. 또, 자바에서는 C++처럼 클래스와 상속, 호출 관계를 다루지만 전역변수나 함수는 다루지 않는다. 따라서 이러한 심벌들을 그림 9-(a)처럼 클래스로 구현 하고, 분석하고자 하는 언어의 특성에 맞게 골라서 사용한다.



(a). Node 클래스



(b). Node 객체와 참조 다이어그램간의 관계

그림 9. 표현 모델 객체 구조

그림 9-(b)는 여러 다이어그램에서 심벌정보를 참조하여, 메모리 낭비를 줄이고, 유연성을 제공함을 보여준다.

이 Node 객체들은 SymbolTree 클래스에서 생성과 파괴를 책임지도록 하였고, 다음 그림 11처럼 많은 언어들이 갖는 전형적인 심벌 트리구조를 갖도록 각각의 객체를 링크 시켰다.

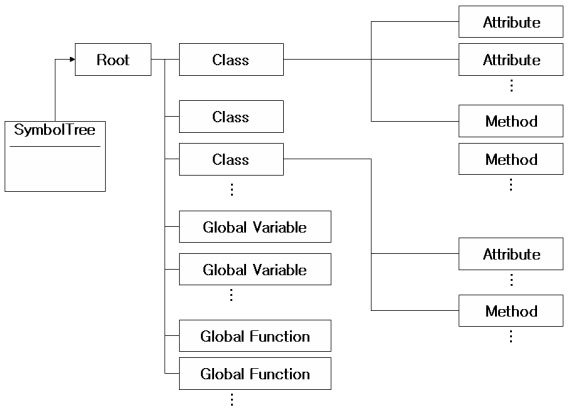


그림 11. 심벌 트리구조

각각의 Node간의 관계 정보는 Node타입의 포인터로 각 객체에서 리스트를 관리한다. 이렇게 함으로써 메모리 효율성을 높일 수 있으며, 그래프로 표현할 때는 대상 객체와 연관이 있는 다른 객체를 따라가면서 그리는 것이 편하기 때문에 포인터를 따라가면서 각각 연관이 있는 객체에 쉽게 접근할 수 있기 때문이다.

여러 심벌들은 타입뿐만 아니라 예약어를 붙임으로써 다양한 의미와 기능을 부여받는다. 이러한 예약어들은 언어마다 각각 다른 형태로 존재하며, 이러한 정보를 저장하기 위한 방법이 필요하다. 이러한 정보들을 저장하는 방법은 비트연산 플래그를 활용한다. 각 심벌의 유무만을 따지면 되기 때문에 그림 12처럼 하나의 변수에 여러 가지 예약어 정보를 저장한다.

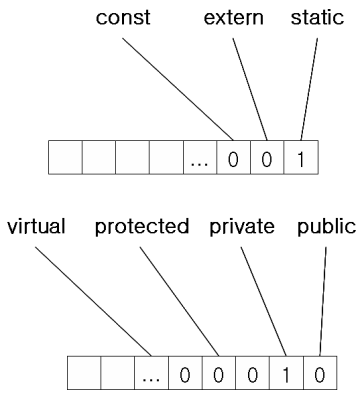


그림 12. 예약어 저장 비트

4.4. 응용 레벨

응용 레벨에서는 잘 가공된 데이터를 응용하여 다이어그램으로 표현하기 위한 기능을 구현한다. Windows 환경에서 MFC나 Borland C++ Builder를 통해 Win32 API를 이용하거나 리눅스 환경에서 Gtk 라이브러리를 써서 구현할 수 있을 것이다. GUI Applicattion의 개발은 이 논문의 범위를 벗어나며, Model Data Structure의 정보를 응용하여 개발하는 단계이므로 향후 연구로만 언급하도록 하겠다.

5. 결론 및 향후 연구

본 논문에서는 소프트웨어 정적 분석의 중요성과 정적 분석을 통해 얻은 분석 자료를 표현하기 위한 모델의 중요성에 대하여 언급하고, 이 표현 모델을 제시하였다. 소프트웨어의 정적 분석을 통해서 함수의 호출 정보, 구문 분석 정보와 같은 흐름 정보와 공유 변수, 공유 함수, 클래스 참조 등의 관계 정보를 분석해 내고, 이를 표현하기 위한 분석 4계층 모델을 제시하였다. 여기서 분석 4계층은 Project Layer, Relation Layer, Flow Layer, Syntax Layer의 4계층으로 이루어져 있으며, 현재의 프로그래밍 언어로 구현한 소스를 분석하기 위한 접근 방법을 제시 하였다. Project Layer에서는 개발하고자 하는 소프트웨어의 운영체제 의존성 및 라이브러리, 컴포넌트의 참조 관계를 보여주고, Relation Layer에서는 소프트웨어를 구성하는 구성 요소들 간의 참조, 상속, 의존 관계 등의 정보를 표현하였다. Flow Layer에서는 소프트웨어의 흐름을 보여주고, Syntax Layer에서는 소프트웨어 최소 단위 프로시저의 세부적인 프로그램 기술 내용을 표현한다. 이렇게 분석하는 시각에 따라 각 계층을 나눈 뒤, 분석 정보를 표현하기 위한 다이어그램의 활용 예시를 보이고 소프트웨어의 분석 정보를 이러한 모델에 기반을 둔 자료구조를 설계하였다. 이 표현 모델을 활용함으로써 정적 분석을 하는 개발자 간의 의사소통의 불일치를 해소하고, 분석 데이터의 일관성 있는 관리가 용이하도록 하였다. 또한, 설계 과정에서 사용되는 표현 모델로는 표현할 수 없는 구현상의 상세한 분석 내용을 표현하기 위한 방안을 제시하였다.

앞으로는 본 논문에서 소개한 설계 자료를 바탕으로 분석 도구를 개발하고 이 도구를 사용하는 개발자의 입장을 반영하여 분석 정보를 효율적으로 나타낼 수 있는 다이어그램을 확장하는 연구가 진행되어야 할 것이다. 또한, 분석 정보에 대한 의사소통의 불일치를 해소하고 소스 정보를 자세하게 표현할 수 있는 표현 모델의 구체적인 발전 방안에 대하여 연구를 진행할 계획이다.

6. 참고문헌

[1] Binder, "Testing Object-Oriented System", Addison Weskey, 1999.

[2] Jeffrey S. Foster, “Improving Software Quality with Static Analysis”, PASTE(Program Analysis for Software Tools and Engineering), 2007

[3] Gary McGraw, "Static Analysis for Security", IEEE COMPUTER SOCIETY, 2004

[4] 신원, “정적 실행시간 분석기의 기반 구조”.

[5] 권상훈, “객체지향 소프트웨어의 테스팅 전략에 관한 연구”, 한국정보과학회, 1995.

[6] 정창신, “소프트웨어 자동 테스팅 도구의 발전 로드맵 분석”, 한국 컴퓨터정보학회, 2004.

[7] 김상영, “Worm 코드를 이용한 정적분석 도구의 설계”, 한국정보과학회 봄 학술발표논문집 Vol. 29. No. 1, 2002.

[8] 황선명, “소프트웨어 품질보증을 위한 테스팅 기법과 검증”, 정보 과학회지 8권 4호, 1990.

[9] 최신 소프트웨어 공학 기법, 한/카네기멜론대학 기술교류협회 편

[10] Akira Hirasawa, “성공과 실패를 결정하는 1%의 객체 지향 원리”

[11] 이재석, "최악 실행시간 분석을 위한 소스코드 분석기 설계“, 건국대학교 석사학위 논문, 2006.8.

[12] 김윤관, “TMO모델 기반 응용프로그램의 정적 분석을 위한 PTETBA의 설계 및 구현”, 건국대학교 석사학위 논문, 2007.2.

[13] 권상훈, “객체지향 소프트웨어의 테스팅 전략에 관한 연구”, 한국 정보과학회 가을 학술 발표 논문집, Vol.22, No. 2, 1995.