자바 코드로부터 시퀀스 다이어그램 추출 기능의 설계 및 구현

윤성아[0] 김태균 윤석진*
부산외국어대학교 컴퓨터공학과, 한국전자통신연구원*
{sayoon[0], ktg}@saejong.pufs.ac.kr, sjyoon@etri.re.kr

Design and Implementation of Sequence Diagram Extraction from Java Code

Seong-Ah Yoon[0] Tae-Gyun Kim Seok-Jin Yoon*
Dept. of Computer Engineering, Pusan Univ. of Foreign Studies, ETRI*

요 약

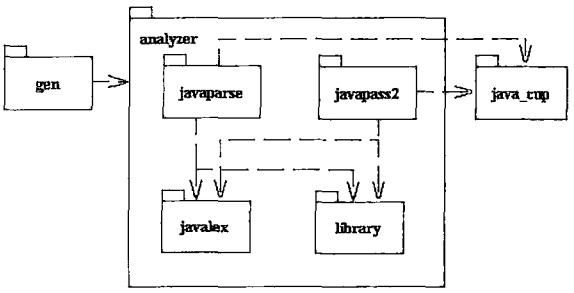
1980년대 이후 많은 연구 개발 성과가 있는 객체 지향 패러다임의 활성화에 이어서 수년 전부터 컴포넌트 기술의 보급이 확산되고 있다. 2000년부터 한국전자통신연구원의 컴포넌트 공학 연구팀 주관으로 개발되고 있는 COBALT 시스템은 EJB 기술을 기반으로 상업성있는 컴포넌트를 개발하기 위한 환경 구축을 목적으로 한다. COBALT 시스템은 UML을 이용한 영역 모델링 기능, 컴포넌트 생성 및 배치 기능, 기존의 자바 코드에 대한 역공학 기능 등을 갖추고 있다. 본 논문은 COBALT 시스템의 부 시스템으로 구현된 역공학 기능 중에서 시퀀스 다이어그램 추출 기능의 설계 및 구현 결과를 다룬다. UML 시퀀스 다이어그램은 객체들 간의 메시지 전달 상황을 모델링하기 위한 것으로 시스템의 실행 흐름을 표현한다. 본 논문에서 구현된 시퀀스 다이어그램 추출 기능을 통하여 컴포넌트 개발자는 기존에 작성된 자바 코드를 쉽게 이해할 수 있게되므로 자바 코드의 재사용성을 향상시킬 수 있으며 재사용된 자바 코드를 이용하여 EJB 컴포넌트를 개발할 수 있다.

1. 서 론

1990 년대 중반 이후로 급격히 발전되고 있는 소프트웨어 컴포넌트 기술은 원시 코드가 아닌 실행 단위의 소프트웨어 부품들을 연결하여 응용 소프트웨어를 조립할 수 있는 기술로서 향후 소프트웨어 개발을 위한 새로운 패러다임으로 자리잡고 있다. 현재 컴포넌트 관련 기술은 Microsoft 사의 DCOM, Sun 사의 EJB, OMG의 CORBA 기술에 의해 주도되고 있다. 국내에서도 이들 기술을 이용한 컴포넌트 개발이 활성화되고 있는 중이며 이러한 점에서 소프트웨어 컴포넌트 개발을 지원하는 개발 환경의 구축이 시급하다. 한국전자통신 연구원의 컴포넌트 공학 연구팀에서는 2000년 이후로 EJB를 이용한 소프트웨어 컴포넌트 기술 기반 연구를 진행 중이며 개발 환경으로 COBALT 시스템을 구현 중이다. COBALT 시스템은 자바로 구현되었으며 UML[1]을 이용한 영역 모델링 기능, 컴포넌트 조립 및 배치 기능, 자바 언어에 대한 코드 분석 기능 및 컴포넌트 추출 기능 등을 갖고 있다. 본 논문은 COBALT의 기능 중에서 자바 언어 분석 기능과 다이어그램 추출 기능을 개발하는 과정에서 얻어진 설계 및 구현 결과에 대하여 다룬다. 자바 언어 분석 기능은 JLEX/CUP[2]을 이용하여 구현되었으며 다이어그램의 추출 기능의 GUI는 자바 Swing Set을 이용하여 구현되었다. 본 논문의 2 장에서는 시스템의 설계 결과에 대하여 논하고 3 장에서는 구현 결과에 대하여 논하며 마지막으로 4 장에서는 결론과 함께 차후 연구 목표에 대하여 논한다.

2. 시스템 설계

본 장에서는 COBALT 시스템의 부 시스템으로 구현된 자바 코드 분석 기능 및 다이어그램 추출 기능의 개요 설계 및 클래스 설계에 대하여 다룬다. 본 논문의 기능을 구현하기 위한 개략 설계는 (그림 1)에서와 같은 패키지 구조를 갖도록 설계가 이루어 졌다.



(그림 1) 자바 코드 분석 및 다이어그램 추출을 위한 패키지 구조

(그림 1)의 각 패키지 기능은 다음과 같다.

- gen 패키지 : 이 패키지는 전체 도구의 통합 GUI를 구성한다. 본 논문에서 구현된 기능들은 gen 패키지에 속하는 통합 GUI의 메뉴 선택에 의해 관련 함수가 호출됨으로써 수행된다.
- analyzer 패키지 : 이 패키지는 본 연구의 결과로 작성된 프로그램들을 포함하는 메인 패키지이다.

• javalex 패키지 : 이 패키지는 자바 프로그램에 대하여 렉시칼 분석을 실행할 때 필요한 기능을 수행하는 클래스들을 포함하는 패키지이다.

• javaparse 패키지 : 이 패키지는 자바 프로그램에 대하여 구문 분석을 수행한 후 파스 트리를 만들어 주는 기능을 구현하기 위해 필요한 클래스들을 포함하는 패키지이다. 본 도구에서는 자바 코드 분석을 위하여 이중 패스 알고리즘을 사용하고 있는데 첫 번째 패스에서는 주로 클래스 다이어그램 생성을 위한 데이터를 수집하고 두 번째 패스에서는 시퀀스 다이어그램 생성을 위한 데이터를 수집한다. javaparse 패키지는 그 중에서 첫 번째 패스를 담당한다.

• javapass2 패키지 : 이 패키지는 이중 패스 알고리즘 중에서 두 번째 패스를 담당하는 것으로 객체 간의 호출 관계를 처리한다.

• java_cup 패키지 : 이 패키지는 클래스의 내용들이 고정되어있는 패키지로 자바 CUP을 사용하기 위해 런타임 시에 링크되어야하는 패키지이다.

• library 패키지 : 이 패키지는 파싱된 결과를 저장하는 자료구조와 시스템 구현을 위해 보편적으로 필요한 클래스들을 포함하는 패키지이다.

본 논문에서 구현된 도구는 이중 패스를 통하여 자바 코드를 파싱한다. 원시 코드 파싱 과정을 두 단계로 나눈 이유는 시퀀스 다이어그램의 생성을 위해서이다. 시퀀스 다이어그램의 생성을 위한 주된 작업은 자바 멤버 함수의 본체에 포함된 다른 객체에 대한 메시지 호출처리이다. 그런데 멤버 함수의 본체를 처리할 때 필요한 정보는 참조된 클래스들의 메소드 정의이므로 참조된 클래스에 대한 메소드 정보가 미리 존재해야만 한다. 이러한 점 때문에 두 단계의 파싱 과정이 이용되고 있으며 두 단계의 패스에서 분담된 역할은 다음과 같다.

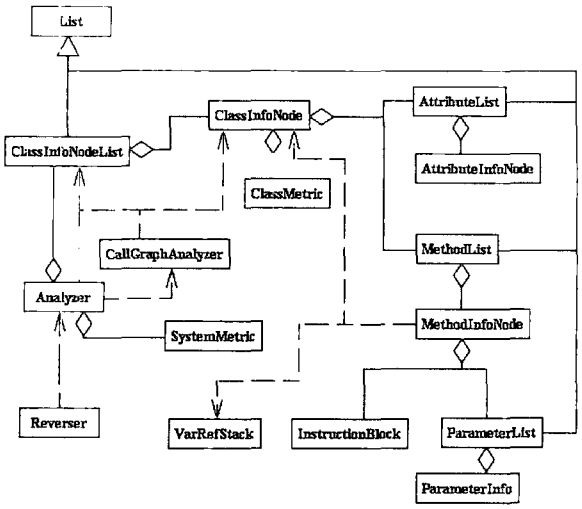
• 패스 1 : 이 단계에서는 모든 클래스의 데이터 멤버, 멤버 함수 정의를 처리한다. 아울러 추출된 파스 트리에 상속 관계에 대한 정보를 반영함으로써 패스 2에서 상속을 고려한 멤버 함수 탐색 작업이 가능하도록 한다.

• 패스 2 : 이 단계에서는 주로 멤버 함수의 본체를 파싱하면서 다른 객체에 대한 메시지 호출 정보를 추출한다. 패스 1 에서 함수 정의에 대한 정보는 이미 수집되어 있는 상태이기 때문에 패스 2 과정에서 함수 호출을 발견하면 호출된 함수 정의에 대한 레퍼런스와 실인자를 포함하는 함수 호출 문에 대한 정보를 저장한다.

(그림 2)의 클래스 다이어그램은 패스 1과 패스 2 과정에서 사용되는 클래스들 간의 상호 관계를 모델링한 것으로 이 그림에 속하는 클래스들의 역할을 개략적으로 설명하면 다음과 같다.

• Reverser : 이 객체는 역공학 도구 서브 시스템의 관점에서 메인 프로그램 역할을 하는 객체이다. 이 객체의 임무는 크게 두 가지인데 그 하나는 통합 GUI의 메뉴 선택에 대한 해당 기능을 수행하는 인터페이스를 제공하는 것이고 다른 하나는 자바 코드 분석 이후에 통합 GUI가 제공하는 인터페이스를 이용하여 다이어그램들을 작성하는 것이다.

• Analyzer : 이 객체는 첫 번째 패스를 위한 구문 분석을 수행하는 파서 객체이다. 이 객체의 주된 목적은



(그림 2) 역공학 구현을 위한 클래스 다이어그램

모든 클래스를 파싱하여 각 클래스를 위한 데이터 멤버와 멤버 함수 정의를 추출하는 것이다. 이 객체의 수행 결과로부터 클래스 다이어그램을 생성할 수 있으며 외부 시스템에 정보를 제공하는 인터페이스를 갖추고 있다.

• CallGraphAnalyzer : 이 객체는 두 번째 패스를 위한 구문 분석을 수행하는 파서 객체이다. 이 객체는 첫 번째 패스의 결과를 이용하여 멤버 함수 본체로부터 시퀀스 다이어그램을 추출할 수 있는 함수 호출 정보들을 얻는다.

• ClassInfoNode : 이 객체는 하나의 클래스에 대하여 분석된 모든 정보를 관리한다.

• ClassInfoNodeList : 이 리스트는 분석된 모든 클래스 객체들을 관리하는 컨테이너 객체이다. 그림에서와 같이 이 객체는 Analyzer 객체의 부속 객체로 존재하기 때문에 외부 부시스템에서 특정 클래스에 대한 정보를 얻고자 할 때는 Analyzer 객체를 통해 이 리스트 객체를 traverse하여 특정 클래스를 찾은 후 그 클래스의 ClassInfoNode 객체 내용에 접근할 수 있다.

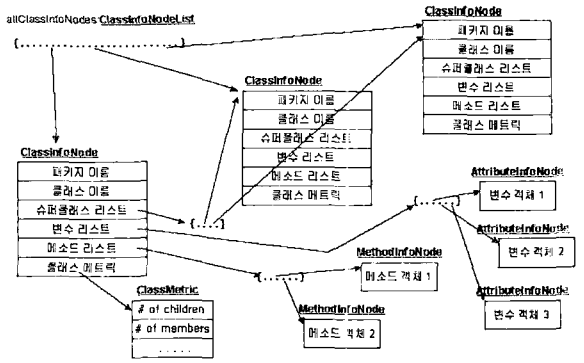
• AttributeInfoNode : 이 객체는 특정 클래스에 속하는 하나의 데이터 멤버에 대한 모든 정보를 관리한다.

• MethodInfoNode : 이 객체는 특정 클래스에 속하는 하나의 멤버 함수에 대한 모든 정보를 관리한다. 특히 시퀀스 다이어그램을 생성하기 위한 객체간의 호출 정보도 이 객체에 의해 관리된다.

• ParameterInfo : 이 객체는 특정 멤버 함수에 속하는 형식 인자에 대한 정보를 관리한다.

• InstructionBlock : 이 객체는 모든 멤버 함수의 부속 객체로 소속되며 시퀀스 다이어그램 추출을 위한 함수 호출 정보를 관리한다.

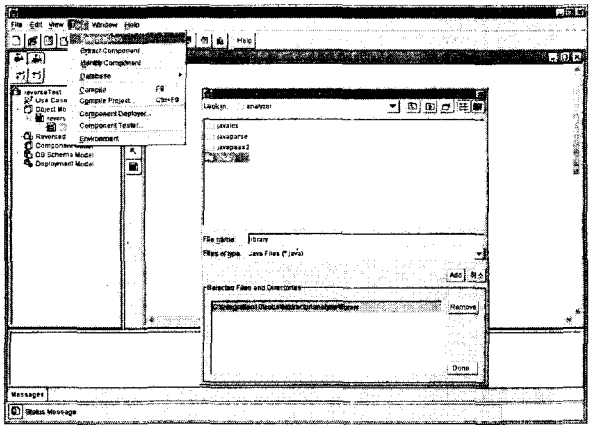
자바 코드 분석 과정을 실행하여 두 패스가 수행되고 나면 원시 코드의 내용은 파스 트리로 변환된 자료구조에 저장된다. (그림 3)은 코드 분석 이후에 저장되는 자료 구조 양식을 보여 준다.



(그림 3) 분석된 클래스 정보들을 저장하는 자료구조

3. 시스템 구현

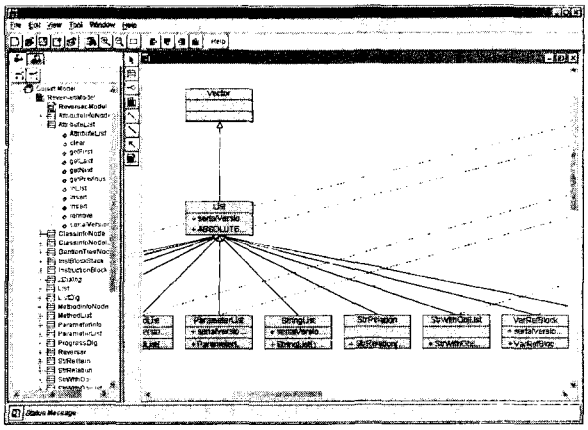
본 장에서는 구현된 기능을 실행 화면 위주로 제시한다. 우선 (그림 4)의 내용은 역공학 기능을 실행하기 위한 GUI 환경을 보여 준다.



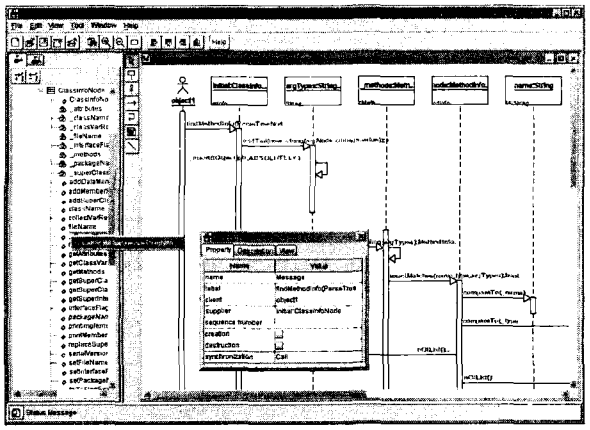
(그림 4) 역공학 실행 초기 화면

(그림 4)에서와 같이 도구의 메뉴에서 <Reverse Java> 버튼을 실행하면 화면 중앙의 대화 상자가 나타나며 이 대화 상자를 통해 분석하고자 할 자바 코드 디렉토리를 설정한 후 도구를 실행시킨다. 도구의 실행이후에 우선적으로 추출되는 결과물은 (그림 5)와 같은 클래스 다이어그램이다.

다음으로 (그림 6)은 시퀀스 다이어그램의 추출 결과를 보여준다. 시퀀스 다이어그램을 생성하기 위해서는 팝업 메뉴를 이용하는데 (그림 6)의 모델 브라우징 윈도에서 추출하고자하는 객체의 멤버 함수를 선택하여 팝업 메뉴를 띄우면 그림에서와 같이 <Generate Sequence Diagram> 이라는 버튼이 나타나게되고 그 버튼을 선택하면 해당 함수를 출발점으로 하는 시퀀스 다이어그램이 생성된다. (그림 6)에서 추출된 다이어그램은 ClassInfoNode 클래스의 findMethodInfo() 함수를 출발점으로 하여 추출된 시퀀스 다이어그램이다.



(그림 5) 추출된 클래스 다이어그램 예



(그림 6) 추출된 시퀀스 다이어그램 예

3. 결론

본 논문에서는 자바 코드로부터 시퀀스 다이어그램을 추출하는 기능의 설계 및 구현에 대하여 기술하였다. 본 기능은 EJB 기술을 이용한 컴포넌트 기반 기술의 개발 환경을 구축을 목표로 구현된 COBALT 시스템의 부속 기능으로 구현되었다. 본 논문에 의해 구현된 기능은 기존의 자바 소프트웨어로부터 컴포넌트를 추출하고자 할 때 시스템의 이해를 돕는 도구로 사용되어 재사용성을 향상시킬 수 있다. 본 연구의 차후 과제로 신뢰성 향상에 대한 연구가 필요하며 메트릭 정보 추출 기능의 구현이 계획되어 있다.

참고문헌

1. Martin Fowler, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, 1997
2. Scott E. Hudson, *CUP User's Manual*, http://www.cs.princeton.edu/~appel/mordern/java/CUP/manual.html