JML을 이용한 Java 원시 코드의 역공학/순공학적 접근

(Forward/Reverse Engineering Approaches of Java Source Code using JML)

장 근 실 † 유 철 중 †† 장 옥 배 †††
(Jang Geun-Sil) (Yoo Cheol-Jung) (Chang Ok-Bae)

요 약 웹상에서 문서의 표준으로 인정받고 있는 XML을 기반으로 전자상거래, 무선통신 및 멀티미디어 기술 등 많은 연구가 활발하게 이루어지고 있다. JML(Java Markup Language)은 Java로 작성된 원시 코드를 이해하고 재사용하는데 도움이 되는 정보를 다양한 목적으로 이용하는데 적합하도록 작성된 XML 응용으로 클래스 계층구조나 클래스 관계성 및 메소드 등에 관련된 다양한 정보를 효과적으로 표현할 수 있는 DTD이다.

본 논문은 역공학 측면에서 JML을 이용하여 Java 원시 코드로부터 주석정보를 추출하고, 그 외에 이해 및 재사용에 도움이 되는 정보를 추출하여 JML 문서를 생성하는 도구와 순공학 측면에서 사용자가 수작업으로 생성하거나 자동으로 생성된 JML 문서에 포함된 문서 정보로부터 Java 응용 프로그램의 골격 코드를 생성하는 도구를 설명한다. 본 연구의 결과를 이용하여 원시 코드의 이해나 분석 또는 유지보수에 유용하고, 필요한 정보를 쉽게 얻을 수 있고, XML 형식의 문서로 인해 개발자들이나 팀 구성원들 사이의 정보의 공유 및 가공을 쉽게 할 수 있다. 또한 JML 문서로부터 생성된 Java 골격 코드는 신뢰성이 있는 강건한 코드로 완전한 원시 코드를 개발하는데 도움을 제공하며, 마지막으로 프로젝트의 비용 및 시간을 절감할 수 있도록 해준다.

키워드 : XML, 정보공유, 문서화, 코드생성, 역공학, 순공학

Abstract Based upon XML, a standard document format on the web, there have been many active studies on e-Commerce, wireless communication, multimedia technology and so forth. JML is an XML application suitable for understanding and reusing the source code written using JAVA for various purposes. And it is a DTD which can effectively express various information related to hierarchical class structures, class/method relationships and so on.

This paper describes a tool which generates JML document by extracting a comment information from Java source code and information helpful for reusing and understanding by JML in terms of the reverse engineering and a tool which generates a skeleton code of Java application program from the document information included in the automatically or manually generated JML document in terms of the forward engineering. By using the result of this study, the information useful and necessary for understanding, analyzing or maintaining the source code can be easily acquired and the document of XML format makes it easy for developers and team members to share and to modify the information among them. And also, the Java skeleton code generated from JML documents is a reliable robust code, which helps for developing a complete source code and reduces the cost and time of a project.

Key words : XML, Information Sharing, Documentation, Code Generation, Forward Engineering Reverse Engineering

† 정 회 원 : 광양보건대학 컴퓨터정보과 교수
jgs@kwangyang.ac.kr
†† 종신회원 : 전북대학교 자연과학대학 컴퓨터과학과 교수
cjyoo@moak.chonbuk.ac.kr
††† 종신회원 : 전북대학교 공과대학 전자정보공학부 교수
okjang@moak.chonbuk.ac.kr
논문접수 : 2002년 3월 29일
심사완료 : 2002년 10월 14일

1. 서 론

소프트웨어공학의 목적인 제한된 개발비용으로 최적의 생산물을 만들어 내는 것을 효과적으로 달성하기 위해서 지난 50여년간 소프트웨어 공학자들은 끊임없는 노력으로 많은 기법들과 방법론들을 만들어왔다. 이들 결과물들 중에서 순공학은 일반적인 개발과정(소프트웨어 개발 주기)을 통해 목적을 달성해 나가는 분야로서 높은 정보의 추상화 레벨로부터 추상수준이 낮은 레벨로 구체화를 시켜나가는 과정을 의미하고, 역공학은 순공학의 반대개념으로서 결과물(원시 코드)을 이용하여 개발과정의 부산물들을 이끌어 내는 분야를 의미한다. Bohner의 소프트웨어 재성 분류법에서도 역공학이란 코드로부터 설계와 명세서를 생산하고, 표현법들을 관리하는 것이라 분류하고 있으며, 구현된 원시 코드로부터 명세서를 추상화하는 기능을 그 핵심사항이라 기술하고 있다[1]. 한편, 하드웨어의 급속한 발전과 인터넷이나 인트라넷과 같은 기반환경의 급속한 보급은 소프트웨어 개발 측면에서도 많은 영향을 미치고 있다[2]. 특히 개발자나 이용자 측면에서 기반환경의 변화는 과거의 고정된 장소에서의 팀단위 작업환경이 개별적인 장소로 이동됨을 의미한다. 또한 프로젝트의 기본적인 환경이 클라이언트/서버나 분산컴퓨팅을 지원하도록 요구하고 있다. 이런 다양한 요인들로 인해 개발자들은 과거에 비해 더욱 큰 부담을 느끼고 있다. 때문에 이러한 환경을 효과적으로 극복하기 위한 많은 노력들이 이루어지고 있는데, 역공학 측면에서 원시 코드 문서화의 자동생성과 순공학 측면에서 대상 언어의 골격 코드를 자동생성하는 [3]과 같은 분야가 대표적이다.

원시 코드 문서화는 코드의 이해용이성과 가독성을 향상시키고 개발자들이 유지보수를 하는데 큰 도움을 주기 때문에 매우 중요하게 인식되고 있다[4, 5, 6]. 하지만 문서화가 중요한 역할을 하는 소프트웨어의 경우 전체 소프트웨어 개발비용의 50% 이상을 차지할 만큼 [7, 8] 많은 시간과 노력이 요구되는 단계로 이를 효과적으로 해결하기 위해 많은 연구들이 진행되었고, 현재도 진행 중에 있다. 또한 코드 자동 생성은 중/대형 소프트웨어 개발에서 부분적인 코드를 개발자와 다양한 표준의 수용 및 사실상의 표준 응용 프로그래밍 인터페이스로부터 복잡성을 감추기 위한 노력의 결과로 소프트웨어 개발에서 보편적으로 증가하고 있다[9]. 코드의 자동 생성은 대부분 고가의 CASE 도구나 모델링 도구를 통해 이루어지는데, 시각적인 도구를 이용하여 원시 코드의 구조를 만들어 낸다. 이는 개발자들에게 코딩에 소비되는 시간을 절약할 수 있게 하고, 생성된 코드의 논리적이고 물리적인 에러가능성을 제거해 주기 때문에 코드의 신뢰성을 높이는 방법으로 인식되고 있다.

XML(eXtensible Markup Language)은 1996년 W3C에서 제정한 마크업 언어로 기존의 웹 표현언어인 HTML이 갖는 제한적인 문제점과 이의 모태가 되는 SGML의 복잡성을 효과적으로 표현하기 때문에, 웹 환경에서 정보의 표현과 공유를 제안하는 표준으로 인정받고 있다. JML은 XML 응용으로 Java를 대상언어로, 개발자들이나 이용자들이 코드를 이해하는데 필요한 정보를 웹을 통하여 효과적으로 제공하고, 공유할 수 있도록 제안된 DTD로 대표적인 Java 문서화 프로그램인 Javadoc의 문서화 태그(documentation tag)를 지원하고, 원시 코드의 이해에 도움이 되는 클래스와 클래스, 객체와 객체, 메소드 호출관계 등의 정보를 제공한다.

본 연구의 내용은 JML을 이용하여 Java 원시 코드 내에 포함된 주석을 포함하여 그 외에 코드의 이해에 필요한 다양한 정보들을 추출하여 이를 XML 형식의 문서로 제공하는 역공학적 측면에서의 연구와 도구를 통하여 자동생성된 JML 문서나 사용자들이 수작업으로 작성한 JML 문서를 이용하여 관련된 Java 원시 코드를 자동으로 생성해주는 순공학적 측면에서의 연구로 이루어진다. 역공학적 측면에서의 문서자동생성은 개발시간을 단축해주고, 기존의 연구물들에 비해 더욱 풍성한 정보를 제공해주고, XML 형식으로 인한 정보의 추출이나 가공이 보다 용이해지는 장점을 제공하며, 알고리즘 단계의 컴포넌트의 이해보다 아키텍쳐의 구조적 정보를 더욱 용이하게 이해할 수 있다[7]. 또한 순공학적 측면에서의 코드자동생성은 JML 문서가 포함하는 다양한 정보를 에러-프루프한 원시 코드로 생성해주기 때문에 코딩 시간의 감소와 에러발생 가능성을 감소시키며, 생산성 및 품질을 증가시키는 장점을 제공한다. 또한 관련된 원시 코드가 존재하지 않을 경우 프로그램의 골격을 이루는 기본 코드를 생성하여 이해를 도모할 경우에도 큰 도움이 된다.

2. 관련연구

2.1 DTD

DTD는 XML 1.0 Recommendation에 소개된 SGML DTD를 간소화한 버전으로[10] XML이 지원하는 사용자 정의 태그와 문서의 구조 정보를 표현할 수 있는 정의서이다[11]. DTD는 정의하는 구문의 이용 목적이나 대상에 의해 속성 기반(Attribute-Centric)의 DTD와 요소 기반(Element-Centric)의 DTD로 나누어

볼 수 있다[12]. 속성 기반 DTD는 이용자 측면에서 효과적인 가독성을 제공하는 반면에 응용프로그램 기반의 적용(처리, 조회)은 어려운 특징이 있고, 반대로 요소 기반 DTD는 가독성은 낮지만, 프로그램 적용이 용이한 특징이 있다. 본 연구에서는 데이타 수집, 가공 및 처리에 용이하도록 JML에 요소 기반 DTD를 적용하였다.

2.2 문서 자동 생성

첫 번째 관련연구는 문서화에 대한 연구물들이다. 원시 코드를 기반으로 주석의 내용을 문서화하는 연구물들은 다양한 형식의 문서 포맷들(Text 형식, Postscript 형식, Unix의 매뉴얼 형식 및 Windows 계열의 도움말 형식, HTML 등)을 지원하고 있지만, 웹과 같은 분산된 환경에서는 인터넷에서의 이용가능성에 중심을 두어야 한다. 인터넷 환경을 고려하면, 문서의 형식은 웹에서의 HTML과 근래의 인터넷 표준 언어로 인정받는 XML 형식을 들 수 있다. 대부분의 기존 연구물들은 코드가 완성된 후 사용자나 개발자 측면에서 개발에 이용할 수 있는 API 형식의 문서를 HTML 형식으로 제공하는 것이 대부분이다. 기존의 연구물들 중에서 가장 널리 사용되는 것은 Java에 포함되어 배포되는 Javadoc와 문서화 API인 Doclet를 들 수 있다. Javadoc는 실행파일 형태로 제공되며, 생성된 원시 코드 내에 포함되어 있는 주석정보를 기반으로 메소드와 클래스에 대한 프로그램 코딩 측면의 API를 HTML 형식의 문서로 생성한다. Doclet는 HTML 이외의 다른 형식의 문서를 생성할 때 이용할 수 있는 API이다.

이들 두 가지는 이용관점이나 시각에 따라 동일하게 간주되는데, 그 이유는 Javadoc를 이용하여 API HTML 문서를 생성할 때 명령행 인수 -doclet를 생략하면(사실 대부분의 Javadoc 이용에서 이 방법을 이용한다.) "standard" Doclet이 적용된다. 만일 이용자(개발자나 최종사용자) 입장에서 다른 형식의 API를 생성할 때에는 각각 다른 프로그램을 작성해야 한다. 여기서 주목해야 할 것은 서로 다른 문서형식들이라 하더라도, 결국 내용은 HTML 문서와 동일한 API라는 데 있다. 즉, HTML 문서를 해당 포맷으로 변환한 것에 불과하다. 그리고, 사용자정의 태그(Javadoc가 인식하는 표준 태그 외의 태그들)를 지원하기 위해서는 Doclet 자체를 수정하여 자신만의 Doclet 파일을 생성해야 하는데, 이 때 XML 형식의 결과물을 생성할 때 DTD를 이용할 수 없다. DTD는 XML의 가장 큰 장점인 정보의 구조를 지원하는데 반드시 필요한 필수적인 요소이다. 결국, DTD가 존재하지 않는 상황에서 XML 형식의 문서를 생성한다는 것은 정보의 구조를 지원할 수 없음을 의미하며, 이는 많은 양의 정보들이 다양한 계층구조 및 관계성으로 이루어져 있는 XML 문서를 생성하는데는 한계가 있다는 것과 개발자나 최종 사용자들에게 필요한 정보를 효과적으로 제공할 수 없다는 상황에 이르게 된다. Javadoc 및 Doclet에 비해 본 연구가 갖는 특징들로는 다음과 같다.

- 문서의 내용 : 본 연구는 API 형식의 문서와 API 형식 외에 논문에서 주장한 다양한 정보들을 복합적으로 포함하는 문서를 생성한다.
- 문서의 포맷 : XML 형식의 문서를 만들기 위해 복잡한 프로그램 개발이 요구된다. Javadoc(Doclet)의 경우 개별적인 프로그래밍이 요구되고, XML 형식을 지원할 때 정보의 구조를 나타내는 DTD의 요소와 속성들의 정보를 효과적으로 기술할 수 없다.
- 범용성 및 재사용성 : HTML 형식의 문서에 포함된 내용이 단순히 API 참고 매뉴얼과 같은 성격으로 HTML이 갖는 형식상의 제한사항으로 인해 문서들의 재사용 가능성 및 문서 정보의 구조 분석이나 내용의 가공에 어려움을 가지게 된다. 하지만, 본 연구에서 제안하는 XML 형식의 문서는 HTML의 어려운 사항들을 자체적으로 해결할 수 있으므로, 문서에 포함된 정보의 재사용이나 정보의 구조 분석 등이 용이하다. JML 문서로부터 원시 코드의 골격 코드를 생성하는 부분은 XML 문서의 재사용성 및 범용성의 범위를 보여준다.

2.3 코드 자동 생성

두 번째 관련연구는 코드의 자동생성에 대한 연구물들을 들 수 있다. 이 부분의 연구 역시 많은 발전과 시행착오를 거듭하여 현재는 상용화된 고가의 CASE 도구들이 많이 존재하고 있다. 이들 도구들은 대부분의 그래픽 유저 인터페이스를 기반으로 하는 드래그 앤 드롭 방식으로 비쥬얼한 개발환경을 제공하며, 도구 자체에 포함된 문법검사기를 통하여 에러-프루프한 코드를 생성한다. 이들 두 부류의 관련연구들은 각각이 개별적인 연구 분야로서 서로 다른 영역을 차지하고 있기 때문에 각각의 결과물들에 대한 호환성이 없고, 결국은 이들을 이용하기 위해 많은 비용 및 복잡한 등록과정이 필요하다.

대표적인 객체지향 모델링 도구인 *Rational Rose*를 중심으로 원시 코드 생성에 대해 살펴보면, Rational Rose의 경우 모델링 과정 전반에 걸쳐 생성된 다양한 모델에 대한 주석정보를 제공할 수 있다. 이 때 Javadoc와의 연계사용을 위해 문서화 과정에서 Javadoc의 doc tag들을 지원한다. 하지만, 그 외에 사용자정의 태그와 같은 형식은 지원되지 않는다. 이와 같은

과정을 통해 생성된 Javadoc 문서를 분석하여 자바 골격 코드를 구축하는 부분은 Rational Rose의 범주와는 일치하지 않는다. 다른 개발 도구들 역시 시각적인 모델들을 제공하여 골격 코드를 구축하지만, 이들 역시 생성된 결과물(원시 코드의 내용을 HTML 또는 XML형식의 문서)로부터 반대로 골격 코드를 생성하는 과정은 고려하고 있지 않다. 그 이유는 이들 도구들이 갖는 성질 때문이다. 즉, 이들 도구들은 다양한 그래픽 표기법을 이용하여 분석과정 및 설계 과정을 유도하고, 이를 통하여 골격 코드를 생성하기 때문에 이를 고려하지 않는 원시 코드의 문서화를 통해 생성된 문서로부터 골격 코드를 생성하는 것은 도구들 자체의 본질과는 관계가 없다.

3. Java Markup Language

문헌 [13]에서 정의한 JML은 Java 프로그램 언어의 구성 컴포넌트인 클래스와 메소드 및 필드를 대상으로 코드의 이해에 필요한 정보의 구문을 정의한 DTD이다. JML의 초기 버전은 속성 기반의 DTD였지만, 자체적으로 표현하는 정보의 종류와 대상이 부족하여 새로운 정보로 추가/변경한 요소 기반 DTD로 변경하였다.

JML은 크게 클래스와 메소드 및 필드와 같이 3개의 컴포넌트로 구성된다. 각각의 컴포넌트에 포함되는 내용들은 대부분의 연구물들에서 원시 코드의 이해나 유지보수와 같은 다양한 소프트웨어공학 활동에 도움이 되는 것들과 그 동안의 개발과정 및 코드 분석 과정을 거치면서 경험한 여러 가지 어려운 부분들로 구성된다.

- 사용자 정의 메소드 : 간혹 클래스 내부에 사용자가 정의한 메소드가 있는데, 이것이 레퍼런스 라이브러리에 이미 존재하는 메소드와 혼동될 때가 있다. 이와 같은 메소드들은 개별적으로 표시해주면 편리하다.
- 클래스 관계성 : 상속 및 재사용으로 인한 클래스들 사이의 관계성이나 객체의 빈번한 호출로 인해 이해가 어렵다. 문헌 [14]를 살펴보면 여러 가지 관계성이 존재하는데, 일반화(generalization), 특수화(specialization) 관계성은 implements 구문에 해당하며, 상속관계에서 부모와 자식을 의미한다. 또한 집약(aggregation), 복합(composition), 연관(association) 관계성은 클래스 몸체 내에서 호출 또는 구성하는 다른 클래스 멤버들로 사상된다.
- 메소드 오버라이딩 : 빈번한 메소드 오버라이딩은 코

표 1 각 컴포넌트별 정보

클래스/인터페이스 포함	메소드/생성자 포함
● 클래스의 목적 • 유용한 부분 • 사용방법(서브클래싱, 인스탄싱) • 작업방식 ● 클래스 정의 • 식별자 : abstract, final, public • 상속구조(부모클래스) • 인터페이스 상속구조 ● 클래스 인스탄스화 • 셋업과정을 포함하는 클래스의 인스탄스 방법 • 결과로 자동생성되는 다른 클래스들 ● 서브클래스화 • 서브클래스화되는 시점 • 오버라이드 메소드 ● 메모리 해제 • 해제되어야 할 인스탄스에 대한 참조	● 메소드의 목적 • 적용분야, 사용법, 효과 • 반환값, 부작용 또는 두가지 모두에 대한 효과 ● 메소드의 정의 • static (class method / instance method) • 처리모드(public / protected / friendly / private) • 인수들의 개수 및 형 • 반환값의 형 • abstract (abstract / concrete) • final (unredefinable / redefinable) • synchronized • native • throws (exceptions) ● 메소드 호출 • 메소드가 호출 : 명시적/묵시적 • 메소드 호출방법 • 각 인수의 유효범위 • 반환값의 유효범위 • 메소드의 부작용 ● 메소드 오버라이딩 • 부모/자식 클래스에서의 오버라이딩
필드	프로젝트
● 필드의 목적, 적용 ● 필드 정의 • static (class variable / instance variable) • 필드의 형 • final (constant / variable) • transient (transient / persistent) • volatile • read / write	● 프로젝트명 ● 저자 ● 버전정보 ● 날짜 ● 전자우편 ● import 문장 ● package 문장

드의 이해와 가독성을 감소시킨다.

- 메소드 오버로딩 : 빈번한 메소드 오버로딩은 반환값 및 인수들, 다른 메소드들과의 명확한 구분을 위해 정확한 정보가 필요하다.
- 복잡한 제어구조 : 너무 많이 중첩된 제어문 내의 제어문은 너무 복잡한 구조를 이루기 때문에 분석과 이해를 어렵게 한다.

표 1은 이상에서 기술한 내용들을 기반으로 각각의 컴포넌트에 필요한 정보들[13, 15, 16, 17]을 정리한 것으로, 열거한 컴포넌트들 가운데 프로젝트와 클래스에 대한 정보 및 이에 대응하는 JML 코드는 표 2에 나타나고, 실제 생성된 정보와 JML 문서로부터 위 사항들을 획득하는 과정 및 결과는 5장에서 상세하게 설명한다.

4. JML Tool

4.1 JML Tool의 구성

JML Tool은 크게 문서 자동생성 부분과 골격 코드 자동생성 부분으로 나뉘어 진다. 문서 자동생성 부분은 다시 정보 추출(DIE : Doc Information Extractor) 모듈과 정보 사상(DIM : Doc Information Mapping) 모듈 및 문서 생성(DG : Document Generator) 모듈로 이루어진다. 골격 코드 자동생성 부분은 JML 문서로부터 원시 코드 생성에 필요한 정보를 추출해내는 정보 추출 부분(SIE : Source Information Extractor)과 정보 사상(SIM : Source Information Mapping) 모듈 및 원시 코드 생성(SCG : Source Code Generator) 부분으로 이루어진다. 그림 1은 문서 자동생성 부분에 대한 모듈 다이어그램으로 각 모듈이 포함하는 서브 모듈을 포함하고 있다. DIE는 입력된 Java 원시 코드를 분석하여 필요한 정보를 추출해 낸다. 이 모듈은 "comment", "class", "method" 및 "field" 모듈로 나뉘어져 있다. 이 과정을 통해 생성된 정보는 정보저장소에 저장된다. DIM은 정보저장소에 저장되어 있는 정보들을 정의한 JML DTD에 맞추어 가공하는 단계로 원시 코드로부터 추출되는 정보들 중에서 직접적인 의미를 얻기 위해 정보를 수집해야 하는 정보들(즉, 클래스, 메소드 등의 관계)을 구성하는 모듈이다. 마지막으로 DG는 수집되고, 가공된 정보들을 이용하여 JML 문서를 생성하는 모듈이다.

표 2 프로젝트 및 클래스 컴포넌트의 DTD

컴포넌트	정 보	DTD
프로젝트	· 프로젝트 요소의 구성: 프로젝트 명, 저자, 버전, 생성일, import 및 package 키워드 · Name: 프로젝트 이름 · Author: 저자의 이름 · Version: 버전에 대한 정보 · Date: 주요 날짜(생성/보급일 등) · Email: 전자우편 주소 · Tel: 전화번호 · Import: 임포트 문장 · Package: 패키지 문장	<!ELEMENT Project (Name?, Author?, Version?, Date?, Email?, Tel?, Import*, Package*, Class*, Method*, Field*)> <!ELEMENT Name (#PCDATA)> <!ELEMENT Author (#PCDATA)> <!ELEMENT Version (#PCDATA)> <!ELEMENT Date (#PCDATA)> <!ELEMENT Email (#PCDATA)> <!ELEMENT Tel (#PCDATA)> <!ELEMENT Import (#PCDATA)> <!ELEMENT Package (#PCDATA)>
클 래 스	· 각각의 클래스별로 정보 구성 클래스 명, 액세스 모드 및 식별자, 상위/하위 클래스, 각종 필드 및 메소드, 예외처리 등의 정보 · Name: 클래스의 이름 · Type: 클래스의 종류(형) · Access: 액세스 모드 · InnerClass: 내부 클래스 · Parent: 상위 클래스 · Childs: 하위 클래스들 · Interfaces: 관련된 인터페이스 · Methods: 포함된 메소드 기본정보 · Instances : 인스탄스 목록 · Fields: 적용되는 필드들 · StaticMembers: 정적멤버 기술 · Purpose: 클래스의 목적 · Note: 기타 정보 기술 · Precondition: 사전조건 기술 · Constraint: 클래스 제한사항 · Exception: 예외처리 정보	<!ELEMENT Class (Name, Type?, Access?, Parent?, Childs*, Interfaces*, InnerClass*, Methods*, Instances*, Fields*, StaticMembers*, Purpose?, Note?, Precondition?, Constraint?, Exception?)+> <!ELEMENT Name (#PCDATA)> <!ELEMENT Type (#PCDATA)> <!ELEMENT Access (#PCDATA)> <!ELEMENT InnerClass (Name, Type)*> <!ELEMENT Parent (#PCDATA)> <!ELEMENT Childs (#PCDATA)> <!ELEMENT Interfaces (#PCDATA)> <!ELEMENT Methods (#PCDATA)> <!ELEMENT Instances (#PCDATA)> <!ELEMENT Fields (#PCDATA)> <!ELEMENT StaticMembers (#PCDATA)> <!ELEMENT Purpose (#PCDATA)> <!ELEMENT Note (#PCDATA)> <!ELEMENT Precondition (#PCDATA)> <!ELEMENT Constraint (#PCDATA)> <!ELEMENT Exception (#PCDATA)>

그림 2는 Java 원시 코드를 자동으로 생성하는 코드 생성기의 모듈 다이어그램이다. 모듈의 구성은 문서 생성기와 거의 유사하다. JML DE 모듈은 입력된 JML문서로부터 원시 코드를 구성하는데 필요한 정보를 추출하는 역할을 한다. JML DM 모듈은 추출된 정보를 가공하여 Java 원시 코드를 구성하는 클래스와 메소드 등의 구조를 형성하는 역할을 한다. 이와 같은 과정을 통해 수집되고, 형성된 정보는 마지막 모듈인 SCG를 통하여 Java 골격 코드로서 생성된다.

4.2 구현 환경 및 정보저장소 구조

XML 문서를 이용한 프로세스에서는 DOM(Document Object Model)이나 SAX(Simple API for XML)를 이용한다. DOM은 1998년 W3C에서 제안한 트리 구조 형태의 API로 메모리 내부에 트리 형태의 XML 구조를 생성하고, XML 문서 전체를 다루는 API에 적합하다. 이와는 반대로 SAX는 데이타 구조를 생성하지 않고, 요소의 시작이나 끝과 같은 이벤트를 생성한다.

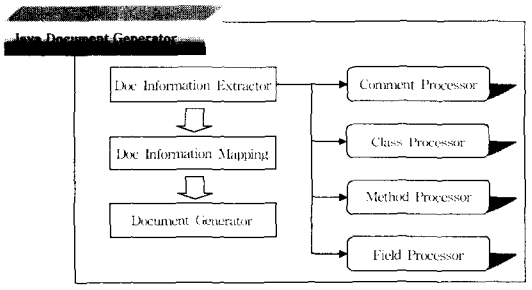

그림 1 Java 문서 생성기의 모듈 다이어그램

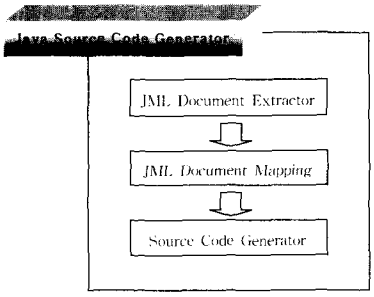

그림 2 Java 원시 코드 생성기의 모듈 다이어그램

하지만, XML 관련 연구인 본 논문의 구현에서는 DOM과 SAX를 사용하지 않고, Tcl/Tk를 이용하여 전용 프로세서를 구현한다. 그 이유는 Tcl extension으로 배포되는 DOM과 SAX 프로세서들이 대부분 Unix 환경에 종속적이므로, Tcl/Tk의 장점(플랫폼 독립적인 실행)을 살리지 못하기 때문이다. 또한 Java와 비교했을 때 Tcl/Tk는 개발이 훨씬 간편하고 명료하며, 플랫폼 독립적이고 가벼운 투자로 많은 효과를 얻을 수 있다. 물론 Java와의 막연한 비교는 자제되어야 하지만, 복합 자료형이나 문자열처리 및 정규표현식 등 많은 부분에서 훌륭한 성능을 보여주고 있으며, 또한 GUI의 개발부분에서도 Java에 비해 많은 장점을 가지고 있다[18]. 또한 논문에서 Java 원시 코드를 파싱하고 분석하여 필요한 정보를 생성한 후 이를 처리하여 JML 문서를 생성하므로, 이 부분에서는 DOM과 SAX를 이용할 이유는 없었고, XML 형식의 JML 문서로부터 Java 골격 코드를 생성할 때 DOM과 SAX의 이용가능성이 발생하였지만, 이들을 이용하게 되면, 시간 및 비용의 중복 투자가 발생하게 되고, 또한 전처리 부분에서 구현된 분석기의 적용을 통해 필요한 정보들을 효과적으로 추출할 수 있었으므로, 그 사용을 고려하지 않았다.

정보저장소는 데이타 구조를 의미한다. Tcl/Tk는 위에서 언급한 것처럼 C나 C++ 또는 Java 외에 많은 언어들에 비해 문자열 처리 및 정규표현식의 처리가 매우 용이하고 강력하다. 또한 기본 자료형 외에 복합 자료형으로 배열형(array type)과 리스트형(list type)을 지원한다. 배열은 일반적인 프로그래밍 언어와는 달리 배열의 요소를 가리키는 첨자에 정수뿐 아니라 의미 있는 데이타의 이름을 지정할 수 있다. 이로 인해 연상배열(associative array)이라고도 한다[19]. 리스트형은 일반 변수에는 하나의 데이타만이 저장되는 특성을 없애고, 특정 문자를 기준으로 여러 개의 데이타를 구분하여 저장할 수 있는 자료형을 의미한다.

자료구조를 설계하는 데는 많은 방법이 있지만, 위에서 언급한 자료형을 기반으로 설계하기 위해서는 다음과 같은 전제조건을 설정하였다.

- ▶ 클래스명을 배열의 이름으로 한다.
- ▶ 다른 코드모듈과의 구분을 위해 클래스는 "CL"로 배열명을 시작한다.
- ▶ 다른 코드모듈과의 구분을 위해 메소드는 "ME"로 배열명을 시작한다.
- ▶ 다른 코드모듈과의 구분을 위해 필드변수는 "VA"로 배열명을 시작한다.
  - 예) 클래스명이 myClass이면 생성될 배열명은 CLmyClass
  - 예) 메소드명이 myFunction이면 생성될 배열명은 MEmyFunction
- ▶ 추출된 정보가 Null이면 Empty처리한다.

▶ 배열의 요소가 반복되어 발생되는 정보이면 리스트 처리한다.
▶ 배열의 요소명은 해당 엘리먼트의 이름으로 한다.
▶ 프로젝트에 존재하는 클래스들의 이름을 개별적으로 관리한다. 이는 여러 클래스로 구분되는 프로젝트에서 배열명으로 사용되는 클래스의 이름을 동적으로 적용하기 위해서이다. 메소드와 필드의 경우에도 클래스 처리와 동일한 방식을 이용한다.

위와 같은 전제조건을 기반으로 생성되는 클래스 모듈에 대한 자료구조는 표 3과 같다. 이번 절에서 언급하지 않은 메소드나 데이타 멤버의 경우도 위와 동일한 과정으로 처리된다. 정의된 자료구조를 바탕으로 데이타를 실제 추출하는 부분은 다음 절에서 언급한다.

4.3 역공학적 접근 - JML 문서생성

JML 문서를 생성하기 위해서는 위에서 기술한 정보저장소의 내용을 만족하는 정보를 원시 코드로부터 정확하고, 바르게 추출하는 프로세스가 필요하다. 이를 위해 문서정보 추출기가 필요하고, 추출된 문서로부터 DTD에 일치하는 문서를 자동으로 생성하는 문서 생성기가 필요하다. 추출된 정보들은 각각의 원시 파일명을 따라서 "info"라는 확장자로 생성된다. 생성된 정보저장소 파일은 텍스트 파일 형식으로 다른 활동에서도 이용할 수 있다.

Java로 작성된 원시 파일은 확장자가 "java"이고, 일반적으로 하나의 클래스 문으로 이루어진다. 물론 여러 개의 클래스로 구성되는 경우도 있다. 여러 개의 클래스일 경우라 하더라도 정보를 추출하여 해당 JML문서를 생성하는 데는 아무런 제한이나 문제가 없다. 원시 파일을 컴파일하게 되면 원시 파일 내에 정의된 클래스명과 "class" 확장자로 이루어진 클래스 파일이 생성된다.

원시 코드로부터 데이타를 바르게 추출하기 위해서는 Java의 구문을 정확하게 파악하여 올바른 구문분석기(syntax analyzer)를 구현해야 한다. 구문분석기를 제작하는 데는 주로 Lex와 같은 프로그래밍 언어를 사용하지만[20], 본 연구에서는 Tcl/Tk를 이용하여 직접 구문분석 코드를 구현하였다. 표 4는 정보를 추출하는 개괄적인 알고리즘을 보여준다.

표 4 정보추출을 위한 알고리즘

```
Input:
   Java source file
Output:
   JML file
Begin algorithm
1: open source file
2: if file == empty or EOF, then exit
3: store current line
     // import, class, method, field, and project
        information
4: keyword search
   if keyword == "import",
         call import information extract module
    elseif keyword == "project",
         call project information extract module
    elseif keyword == "class",
         call class information extract module
    elseif keyword == "method",
         call method information extract module
    endif keyword == "field",
         call field information extract module
5: current line + 1
   End algorithm
```

위의 알고리즘에서 항목 4번이 실질적인 정보추출 모듈인데, 읽어 들인 라인에서 키워드를 매칭하여 매칭된

표 3 클래스 모듈의 자료구조

배열명	CL("ClassName"	클래스명	myClass
요소명	요소형	내용구분자	적용례
• Name	list	::	CLmyClass(Name)
• Type	string		CLmyClass(Type)
• Access	string		CLmyClass(Access)
• InnerClass	string		CLmyClass(InnerClass)
• Parent	string		CLmyClass(Parent)
• Childs	list	::	CLmyClass(Childs)
• Interfaces	list	::	CLmyClass(Interfaces)
• Methods	list	::	CLmyClass(Methods)
• Instances	list	::	CLmyClass(Instances)
• Fields	list	::	CLmyClass(Fields)
• StaticMembers	list	::	CLmyClass(StaticMembers)
• Purpose	string		CLmyClass(Purpose)
• Note	string		CLmyClass(Note)
• Precondition	string		CLmyClass(Precondition)
• Constraint	string		CLmyClass(Constraint)
• Exception	string		CLmyClass(Exception)

결과에 의해 해당 서브모듈을 호출하여 연속적인 데이타를 처리한다. 표 5는 클래스 정보를 위한 서브모듈에 대한 개괄적인 알고리즘을 나타낸다. 알고리즘에서 항목 4-1과 항목 4-2는 클래스 내부를 구성하는 부분을 추출하기 위한 부분이다. "{"와 "}"내부에 다양한 코드가 나타나는데, 이 중에는 메소드나 정적변수 또는 필드 변수 등도 포함된다. 여기서 메소드의 경우 역시 "{" 와 "}"를 이용하여 몸체를 표현하기 때문에 각각의 개수를 비교하여 모듈의 혼동을 피한다.

표 5 클래스 정보를 위한 서브모듈 알고리즘

```
Input:
   Java source file content
Output:
   Class information
Begin algorithm
1: store line position
2: extract the class declaration information
3: current line ++
// extract information of class module
4: keyword search
  4-1: if keyword == "{", then
                current line ++
                goto 4
       endif
  4-2: if keyword == "}", then
                if compare a count of "{" and "}", then
                  current line ++
                  store current line to old line
                  exit
                else
                  current line ++
                  goto 4
                endif
              endif
  5: extract the class body information
  6: goto 4
End algorithm
```

4.4 순공학적 접근 - Java 골격 코드 생성 부분

위에서 정의한 DTD는 클래스와 메소드 및 필드 모듈과 프로젝트에 대한 정보를 담고 있다. 각각의 모듈들이 갖는 정보들은 직접 정보(direct information)와 간접 정보(indirect information)로 구분된다. 직접 정보란 원시 코드로부터 직접 추출한 정보들을 의미하고, 간접 정보란 개발자들 또는 프로그래머들이 해당 모듈이나 특정 코드 부분에 유지보수나 코드 이해에 도움을 주기 위해 기술한 정보들을 의미한다. 프로그래밍 언어로 구현된다는 것은 분석이나 설계와 같은 개발주기의 단계를 거쳐 결과를 얻어냈다는 것을 의미하지만 이와 같은 개발주기의 단계에서 얻어진 각각의 정보들은 추상화되어 코드로 표현되기 때문에 분석이나 설계 단계에서 얻어낸 정보들을 유지보수나 코드의 이해를 위해 효과적으로 코드상에 표현할 수는 없다. 각각의 프로그래밍 언어들이 제공하는 주석은 이와 같은 어려움을 해결하여 추상화되지 못하는 정보들을 제공할 수 있는 형식이다.

골격 코드 생성에 이를 적용시키면, 원시 코드로부터 추출되는 정보들 중에서 직접 정보는 Java 골격 코드를 생성할 때 Java 구문에 의해 해당 키워드를 동반하여 정확한 위치에 삽입되어야 하는 정보들을 의미하고, 간접정보는 원시 코드의 해당 부분에 주석의 형식을 이용하여 삽입될 정보들을 의미한다. DTD의 프로젝트 모듈에 나오는 정보들은 생성될 Java 골격 코드의 머리 부분에 위치하고, 나머지 3개의 모듈에서 "Purpose", "Note", "Precondition", "Constraint" 및 "Exception" 요소에 적용되는 내용들은 해당 모듈이 시작하기 전에 주석으로 표현된다.

만일 해당 원시 코드가 존재하고, 이로부터 JML 문서를 생성하였다면, 정보저장소 내에 표현되는 내용들은 해당 파일(원시 코드명.info)로 존재하게 된다. 이는 이미 원시 코드가 존재한다는 의미이므로 역공학적 측면에서 Java 골격 코드를 생성한다는 것이 큰 의미를 갖지 못한다. 하지만, 대응되는 Java 원시 코드가 존재하지 않고, JML 문서만이 존재하는 경우나, 사용자가 JML DTD에 맞추어 JML 문서를 만들었을 경우에는 원시 코드의 다양한 정보(클래스나 메소드의 구조적인 정보 및 개별적인 코드 정보, 설계 및 분석 단계에서 산출된 다양한 정보)를 제공하는 골격 코드의 생성이 큰 의미를 갖게 된다.

코드를 생성하는 과정 중에서 클래스에 대한 정보를 추출하기 위한 전반적인 알고리즘은 표 6과 같다.

표 6 클래스 정보 추출을 위한 알고리즘

```
Input:
   Line context
Output:
   Extracted information about class module
Begin algorithm
1: store line position
2: current line ++
3: extract tag name
4: check class end tag
   if check result == true, find end tag
              if end tag exists in the same line,
                    extract the content
                  store the content to array
                    goto 4
            elseif
              current line ++
              extract the tag
              match the tag and the keyword
              extract the content
              store the content to array
              goto 4
            endif
     endif
5: return main routine
End algorithm
```

위 알고리즘은 입력 파라메터로 현재의 라인 번호와 추출한 키워드(여기에서는 class)를 요구하며, 반환값은 현재의 라인 번호이다. 전달된 라인 번호는 전체적인 문서의 제어를 위해 필요하고, 키워드의 경우 해당 모듈의 데이타를 구성하기 위해 필요하다. 항목 ④부터 필요한 정보를 추출해 내는데, 먼저 클래스의 이름을 추출해 낸다. DTD에 정의된 내용에 맞추어 추출한 클래스 이름은 해당 배열의 이름으로 작용되고, 각 엘리먼트 이름은 해당 배열에서 인덱스로 이용이 된다. 이와 같은 과정을 통하여 하나의 클래스 모듈을 완성하게 되면, 클래스에 필요한 구문 정보와 주석으로 표현되어야 할 정보들을 포함하는 배열이 완성된다.

실제로 Java 골격 코드를 생성하는 부분은 각각의 배열을 탐색하면서 각 요소마다의 인덱스와 내용을 Java 클래스 구문에 맞추어 조립을 한다.

5. 적용사례

이 장에서는 JML을 구성하는 각 요소 및 속성들을 이용하여 실제 정보를 획득하는 과정과 실제 코드를 적용시켜 생성된 각종 정보 및 이들의 표현을 기술한다. 먼저 3장에서 언급한 각 컴포넌트별 정보와 이를 적용한 JML을 중심으로 원시 코드의 이해나 재사용에 필요한 정보를 획득할 때 클래스들 사이의 관계나 객체들 사이의 관계 및 메소드 사이의 호출관계 및 다형성과 같은 정보들을 이용자가 획득하는 방법은 다음과 같다. 이미 언급한 것처럼 일반화와 특수화 관계성은 JML에서 각 Class 요소의 자식 요소인 Parent 요소를 이용하여 확인이 가능하다(그림 4의 ①). 각 Class 요소에 존재하는 Interfaces 요소는 다중상속을 지원하는 Java의 객체지향 메커니즘으로 현재의 클래스에서 상속받는 인터페이스를 의미한다. 또한 현재 클래스를 부모 클래스로 하는 상속관계를 갖는 자식 클래스들은 Childs 요소를 통해 확인할 수 있다.

또한 연관관계에 포함되는 집약과 복합의 관계성은 어떤 클래스 내부에서 클래스의 일부로 구성되는 다른 클래스들과 그들의 인스턴스로 표현된다. 하지만, 집약과 복합의 차이점은 클래스나 객체들 상호간의 관계가 어느 정도의 밀접성으로 유지되는가에 있으며, 코드 내부에서 이들을 구분하는 것은 설계나 분석 단계와는 다른 구현 단계 혹은 최종 결과물 상에서는 큰 의미가 없으므로 본 연구에서는 이 둘의 의미를 구분하지 않고, 연관관계로 통일한다. 클래스들의 연관관계를 JML을 이용하여 구성하는 방법은 다음과 같다. JML에서 Class 요소는 여러 개의 Fields 요소를 가질 수 있다. 이들은 해당 클래스 내부에 존재하는 일반 원시 변수(primitive variable, 정수, 실수, 문자형 등)들과 인스턴스들로 구성되며, 이들의 차이는 본 연구에서 구현된 분석기를 통해 구분 가능하다(그림 3의 ①). 이들 중에서 클래스 내부에 존재하는 인스턴스들이 위에 언급한 연관관계에 해당하는 클래스들이 된다.

메소드들 간의 호출 관계의 경우는 JML에 존재하는 Method 요소에 존재하는 Polymorphism 요소와 IncludeClass 요소 및 Messages 요소들을 이용하여 획득할 수 있다. 먼저 Polymorphism 요소는 메소드의 형식과 클래스의 이름을 요소들로 가지기 때문에 해당 클래스 자료구조의 메소드 정보와 형식을 추적하여 다형성 정보를 획득할 수 있고(그림 3의 ②), IncludeClass는 현 메소드를 포함하는 클래스의 이름을 제공한다. 마지막 요소인 Messages는 메시지 호출에 관한 정보를 제공하기 위해 Sender 요소와 Receiver 요소로 구성되고, 이들 두 요소는 각각 클래스 이름과 메소드 이름을 그 값으로 취한다. 이중에서 Sender 요소는 현재 메소드에 메시지를 전달하는 메소드와 이를 포함하는 클래스를 나타내며, Receiver 요소는 현재 메소드가 메시지를 전달하는 메소드와 이를 포함하는 클래스를 나타내게 되며, 이들 Messages 요소에 포함된 Sender와 Receiver 요소의 값들을 추적함으로써 메소드들의 호출관계를 획득할 수 있다(그림 4의 ③). 또한 메소드 내부에서 포함된 객체 및 클래스는 각 Method 요소의 Members 요소를 통해 확인할 수 있다. Members 요소는 MemberType과 MemberName의 쌍들을 포함할 수 있는데, 이들이 각각 클래스의 이름과 객체의 이름을 나타낸다. 이 정보를 이용하여 메소드와 인스턴스의 관계성 및 연관관계를 획득할 수 있다(그림 4의 ②).

본 연구에서는 JDK 1.3 SE 버전에 포함되어 설치되는 데모 파일들을 대상으로 테스트를 실행하였다. 설치된 데모 폴더는 여러 개의 서브 폴더들을 포함하고 있다. 이중에서 applet 폴더는 다시 각각의 프로젝트 별로 개별적인 폴더로 구성되며, 그 중에서 ArcTest 프로젝트를 대상으로 추출한 정보 파일과 생성된 JML 파일을 제시한다. 그리고, 생성된 JML 파일로부터 생성한 Java 골격 코드를 제시한다. 표 7은 ArcTest에 대한 개괄적인 정보로 전체 라인수(주석 포함), 포함된 클래스의 개수 및 이름, 개별적인 클래스 별로 포함된 필드의 개수와 메소드의 개수 및 전체의 개수를 나타내고, mLOC는 변경된 라인수를 의미한다. mLOC의 변경된 라인수의 의미는 ArcTest.java에 포함된 주석정보 중에서 저작권에 관련된 정보를 제외한 것을 나타낸다. ArcTest.java

표 7 예제 프로그램의 개괄적인 정보

프로젝트명	파일명	LOC	mLOC	클래스명	필드 개수	메소드 개수
ArcTest	ArcTest	159	130	ArcTest	2	7
				ArcCanvas	4	2
				ArcControls	3	2
총계	1			3	9	11

에는 총 4개의 주석정보가 제공되는데, 그 중 2개는 "/* */"를 이용한 다중라인 주석으로 각각 저작권에 대한 정보와 ArcTest 클래스의 정의에 앞서 제공되는 실행에 관련된 정보이다. 나머지 2개는 ArcTest 클래스에서 이용되는 두 개의 객체형에 대한 정보로 "//"를 이용한 단일라인 주석 형식으로 정의된다.

그림 3은 ArcTest로부터 추출한 정보들을 나타낸다. 추출된 내용은 위에서 정의한 내용을 기반으로 Arc Test.info라는 텍스트 파일에 저장된다. 93라인부터 세 번째 클래스인 ArcControls에 대한 정보가 나타난다. 화면을 보면 데이타 멤버형/클래스명에 3개의 클래스명이 있고, TextFiled의 인스턴스로 "s"와 "e"가 있고, ArcCanvas의 인스턴스로 "canvas"가 나타나는 것을 확인할 수 있다. 또한 2개의 메소드를 가지고 있는데, 하나는 생성자이고, 다른 하나는 일반 메소드이다. 각 생성자와 메소드명 아래 부분에 개별적인 정보들(전체 인수 목록, 메소드 인수가 클래스일 경우에는 클래스명 및 해당 객체명)이 나타난다. 그리고, 마지막 라인에 전반적인 정보로 static, final, synchronous, 반환형 등의 정보가 나타난다. 메소드 actionPerformed를 대상으로 부연 설명을 하면, 전체 인수 목록은 "ActionEvent ev"이고, 이로부터 "메소드인수 클래스명"과 "객체명"에 대한 정보를 확인할 수 있고, 반환형이 "void"임을 알 수 있다.

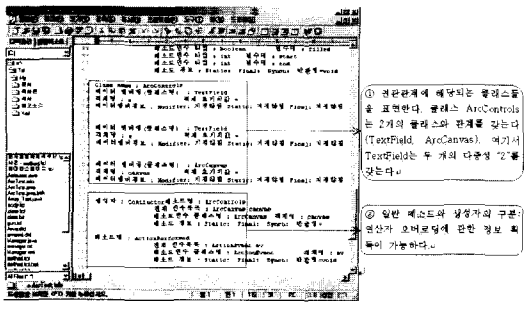

그림 3 정보분석기로부터 추출된 정보

그림 3에서 나타난 정보들(ArcTest.info)을 대상으로 자동 생성된 ArcTest.xml 문서의 내용을 인터넷 익스플로러를 이용하여 브라우징한 화면은 그림 4와 같다. 생성된 문서의 크기는 7KByte이고, 총 라인의 수는 251라인에 이른다. 하나의 화면에 메소드에 대한 정보를 출력하기 위해 3개의 클래스 중에서 2개를 펼치지 않았다. 펼친 클래스의 이름은 ArcTest이고, 나타난 메소드는 ArcTest 클래스의 init 메소드이다. 문서 시작 부분에 3개의 import문이 나타난다.

표 8에 나타나는 코드는 생성된 XML 문서로부터 생성된 Java 골격 코드로, XML 문서에서 Import 요소에 해당하는 부분과 Class 요소들 중에서 첫 번째 클래스인 ArcTest에 해당하는 부분으로 구성되어 있다.

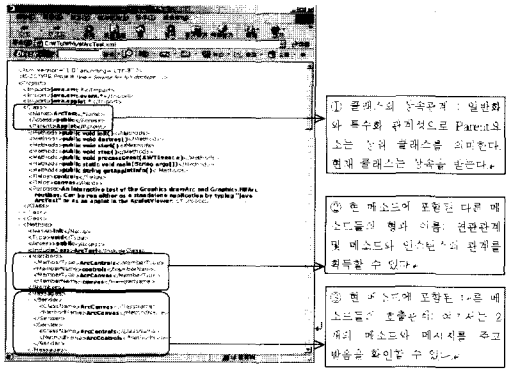

그림 4 생성된 JML 파일의 브라우징 결과

6. 결 론

문서화의 목적은 문서 생성자와 문서의 사용자들 사이의 의사소통을 용이하게 하고, 설계 단계의 의도, 적용사례들 및 잠재적인 문제점들에 대한 통찰력을 제공해야 한다. 본 연구에서는 요소 지향 방법을 이용하여 개발한 DTD인 JML을 기반으로 문서화의 목적을 달성하는 문서를 생성하는 부분과 생성된 JML 문서나 사용자가 JML을 기반으로 작성한 JML 문서로부터 Java 골격 원시 코드를 생성하는 코드 자동생성 부분에 대해 기술하였다.

역공학적 측면에서 Java 문서 자동생성은 Javadoc가

표 8 생성된 Java 골격 코드

```
// Project part

// Import & Package part
import java.awt.*
import java.awt.event.*
import java.applet.*

// Class part
/* Purpose: An interactive test of the Graphics.drawArc and
 Graphics.fillArc
routines. Can be run either as a standalone application by
typing "java ArcTest" or as an applet in the AppletViewer. */
public class ArcTest extends Applet {
// Field part
  ArcControls controls;
  ArcCanvas canvas;

// Method prototype part
  public void init(){
    ArcCanvas canvas = new ArcCanvas();
    ArcControls controls = new ArcControls(canvas));
  }
  public void destroy(){
  }
  public void start(){
  }
  public void stop(){
  }
  public void processEvent(AWTEvent e){
  }
  public static void main(String args[]){
    Frame f = new Frame("ArcTest");
    ArcTest arcTest = new ArcTest();

  }
  public String getAppletInfo(){
    return "An interactive test of the Graphics.drawArc and
Graphics.fillArc";
  }
}
```

이용하는 주석형식을 지원하고, Javadoc이 생성하지 못하는 정보들을 생성하여 개발자나 이용자들에게 보다 풍부한 정보를 제공한다. 이와 같은 정보들은 객체와 객체의 관계, 클래스와 클래스의 관계, 메소드들 사이의 호출관계 등 이해하기 어려운 부분들을 쉽게 해결할 수 있게 한다. 또한 XML 형식의 문서를 생성하기 때문에 분산된 개발환경이나 이용환경에서 정보의 공유 및 가공의 용이성을 제공한다. 그리고, 문서생성에 소비되는 많은 시간과 비용 및 인력을 절약하여 프로젝트 개발과정에 많은 도움을 줄 수 있다.

결과물로부터 개발과정의 산출물들의 유도해내는 순공학 측면에서 코드 자동생성이 갖는 의미는 다양하고 강력하다. JML을 구성하는 문법이 매우 간단하기 때문에 쉽게 Java 코드에 필요한 정보를 구성하는 문서를 작성할 수 있다. 이렇게 작성된 문서로부터 생성되는 골격 코드는 반복적이고, 소비적이며, 에러가능성이 높은 문제를 해결하며, 신뢰성 및 품질이 높은 코드가 된다. 또한 상업적으로 존재하는 도구들이 갖는 문제점들 즉, 높은 하드웨어 성능, 복잡한 사용법, 무거운 시스템의 필요성을 극복할 수 있다.

향후 연구 분야로는 생성된 정보문서의 효과적인 이용을 위한 사용자 정의 정보 추출이 필요하다. 포괄적으로 생성되는 문서는 사용자의 입장에 따라서 부담스러울 수 있다. 이런 부담을 감소하기 위해서는 사용자들이 필요한 정보를 지정하고, 이를 인터넷 브라우저나 전용 도구에서 디스플레이할 수 있어야 한다. 이 부분에 대해서는 문서 정보 템플릿을 이용하여 연구를 진행하고 있다.

또한 JML을 구성하는 각 요소들을 사용자들이 보다 쉽게 이용할 수 있도록 하기 위한 시각적인 인터페이스의 제공이 필요하고, Java 외에 다양한 객체지향 언어를 지원하도록 다양한 DTD를 개발하고, 개발된 DTD들을 지원하여 문서를 생성하고, 반대로 생성된 문서로부터 해당 언어로 구성된 코드를 생성하는 연구가 필요하다.

참 고 문 헌

[1] 장옥배 외 5인, 소프트웨어공학 이론과 실제, p.495, 도서출판 한산, 서울, 2001.

[2] Frank, M., and Gail, K., "Software Engineering in the Internet Age," IEEE Internet Computing, pp. 22-24, Sept.-Oct. 1998.

[3] Pankaj, K. G., and Walt, S., "A Hypertext System to Manage Software Life Cycle Documents," IEEE Software, pp. 90-98, May 1990.

[4] Marry, H., "Using Documentation as a Life Cycle Tool," Software Magazine, Dec. 1992.

[5] Marcello, V., and Curtis, C., "Software System Documentation Process Maturity Model," Dept. of CS, Oregon State University, Corvallis, 1992.

[6] Larry, S., "Trends In Automating Document Generation," IEEE Software, Vol.12, Is.5, pp. 116-118, Sept. 1995.

[7] Kenny, W., Scott, R., Tilley, Hausi, A. M., and Margaret-Ane, D. S., "Structural Redocumentation: A Case Study," IEEE Software, Vol.12, Is.1, pp. 46-54, Jan. 1995.

[8] Capers, J., Applied Software Measurement, Assuring Productivity and Quality, McGraw Hills, 1991.

[9] Mark, P., "Code Generation Using Javadoc," Javaworld, Aug. 2000, http://www.javaworld.com/javaworld/jw-08-2000/jw-0818-javadoc_p.html.

[10] W3C, Extensible Markup Language(XML) 1.0, 2nd Ed., W3C Recommendation, Oct. 2000, http://www.w3.org/TR/REC-XML.

[11] Eric, V. D. V., "Comparing XML Schema Languages," XML.com, Dec. 2001, http://www.xml.com/pub/a/2001/12/12/schemacompare.html.

[12] Michael, D., "Are Elements and Attributes Interchangeable?," XML-Journal, Vol2, Is7, pp.

42-47, July, 2001.
[13] Jang, G. S., Yoo, C. J., and Chang, O. B., "Information Sharing of Java Program Using XML," ACIS 1st Int. Conf., SNPD 00, Reims in France, pp. 384-391, May. 2000.
[14] Grady, B., and et. al., The Unified Modeling Language User Guide, Addison Wesley, 1999.
[15] Javadoc, http://java.sun.com/javadoc.
[16] Doc++, http://www.zib.de/Visual/software/doc++/index.html.
[17] 김재웅, 유철중, 장옥배 "Java 프로그램에 대한 복잡도 척도들의 실험적 검증", 정보과학회논문지: 소프트웨어 및 응용, 27권 12호, pp. 1141-1154, Dec. 2000.
[18] John K. O., "Scripting: Higher Level Programming for the 21st Century," http://www.scriptics.com/doc/scripting.html.
[19] Brent, B. W., Practical Programming in Tcl and Tk, 2nd Ed., Prentice Hall, 1997.
[20] Junichi, S., and Yoshikazu, Y., "Managing the Software Design Document with XML," ACM SIGDOC, 1998.

장 근 실

1995년 전북호원대학교 전자계산학과 졸업(이학사). 1997년 전북대학교 대학원 전산통계학과 졸업(이학석사). 1999년 전북대학교 대학원 전산통계학과 박사수료. 1996년 9월 ~ 1997년 3월 전북대학교 전자계산소 조교. 1997년 3월 ~ 현재 광양보건대학 컴퓨터정보과 조교수. 관심분야는 소프트웨어공학, 컴포넌트기술, 웹공학

유 철 중

1982년 전북대학교 전산통계학과 졸업(이학사). 1985년 전남대학교 대학원 계산통계학과 졸업(이학석사). 1994년 전북대학교 대학원 전산통계학과 졸업(이학박사). 1982년 9월 ~ 1985년 3월 전북대학교 전자계산소 조교. 1985년 4월 ~ 1996년 12월 전주기전여자대학 전자계산과 부교수. 1997년 1월 ~ 현재 전북대학교 자연과학대학 컴퓨터과학과 조교수. 관심분야는 소프트웨어공학, 에이전트공학, 컴포넌트기술, 분산객체기술, GNSS(GPS), GIS, 멀티미디어, 인지과학 등

장 옥 배

1966년 고려대학교 수학과 졸업(이학사). 1973년 고려대학교 교육대학원(교육학석사). 1980년 죠지아 주립대(박사과정수료). 1987년 산타바바라대학교 졸업(Ph.D). 1990년 ~ 1991년 영국에딘버러대학교 객원교수. 1980년 4월 ~ 현재 전북대학교 공과대학 전자정보공학부 교수. 관심분야는 소프트웨어공학, 전산교육, 수치해석, 인공지능 등