프로그램 변환을 이용한 자바 프로그램의 동적 예외 분석*

오희정° 창병모
숙명여자대학교 컴퓨터과학과
{lutino°, chang}@sookmyung.ac.kr

Dynamic Exception Analysis for Java Program by Program Transformation

Hee-Jeong Ohe, Byeong-Mo Chang
Dept. of Computer Science, Sookmyung Women's University

요 약

프로그램 내에서 심각하지 않은 에러를 예외라고 한다. 자바에서는 프로그래머가 이러한 예외를 명시적으로 처리할 수 있으며 신뢰성 있는 프로그램 개발을 위해서는 발생 가능한 예외에 대한 적절한 처리가 중요하다. 이러한 예외의 처리 여부에 대한 연구는 주로 정적 분석을 중심으로 이루어져 왔으나 프로그램 실행 중에 예외 발생, 전파 경로에 대한 정확한 정보는 제공해주지 못한다. 본 연구에서는 프로그램 변환을 이용해 실행 중에 발생, 처리, 전파되는 예외 정보를 실시간으로 제공할 수 있는 시스템을 설계 개발한다.

1. 서 론

자바 언어는 신뢰성 있는 프로그램의 개발을 위해 예외를 처리할 수 있는 명시적인 예외 처리 메커니즘을 제공한다. 자바의 예외 메커니즘은 모든 예외 타입이 Throwable 클래스나 이것의 서브 클래스를 상속받아야 하며, 예외를 발생시키는 throw문과 발생된 예외를 처리해주는 try-catch 문을 제공한다.

이러한 예외 메커니즘을 통해 프로그램의 신뢰성을 향상시키기 위해서 프로그래머는 발생 가능한 예외를 적절히 처리해야 한다. 만약 발생된 예외에 대해서 적절한 처리가 이루어지지 못하면 프로그램의 실행이 멈출 수 있으므로 실행 시간에 처리 되지 못하는 예외들을 분석하는 일은 매우 중요하다.

그 동안 이러한 예외 처리 여부에 대한 연구는 정적 분석을 중심으로 이루어져 왔으며 이러한 방법은 프로그램 실행 중에 예외 발생, 처리, 전파에 대한 정확한 정보를 제공해주지 못하는 문제점이 있다[3][4].

본 연구의 대상인 J2ME같은 내장형 소프트웨어에서는 신뢰성 있는 프로그램 개발이 매우 중요하다. 이를 위해 개발 과정에서 실제 실행 중에 발생하는 예외와 이들의 처리 과정을 살펴볼 수 있는 개발 환경이 필요하며 이를 통해 보다 신뢰성 있는 프로그램 개발이 가능할 것이다. 현재 이 부분에 대한 연구는 시작 단계에 있다[2].

본 연구에서는 이를 위하여 프로그램 변환을 이용해 실행 중에 발생, 처리, 전파되는 예외 정보를 실시간으로 제공할 수 있는 시스템을 설계, 개발한다. 이 시스템은 사용자가 동적 예외 분석을 위한 옵션을 선택하면 이를 바탕으로 예외 트레이스 정보를 실행 중에 실시간으로 제공한다. 이를 이용하면 좀 더 세부적이고 동적인 예외 분석이 가능하다. 또한 실행 직후에는 실행 과정에서 발생, 처리, 전파된 예외들을 요약한 프로파일 정보를 제공한다.

본 연구에서는 동적 분석에 의한 실행 시간 부담을 줄이기 위해 옵션에 따라 해당 예외 트레이스 정보나 프로파일 정보만을 제공하도록 입력 프로그램을 변환하고 변환된 프로그램을 실행함으로써 동적 분석 정보를 얻을 수 있도록 하였다. 대상 언어는 J2SE뿐만 아니라 J2ME에도 가능하며 자바 컴파일러의 전단부인 Barat[1]을 이용하여 입력 프로그램을 변환하였다.

2장에서는 우리가 구현한 시스템의 전반적인 설계를 기술하고, 3장에서는 실질적인 구현과 실험 결과에 대해서 기술할 것이다. 마지막으로 4장에서는 결론과 향후 연구 과제에 대해 살펴본다.

2. 설 계

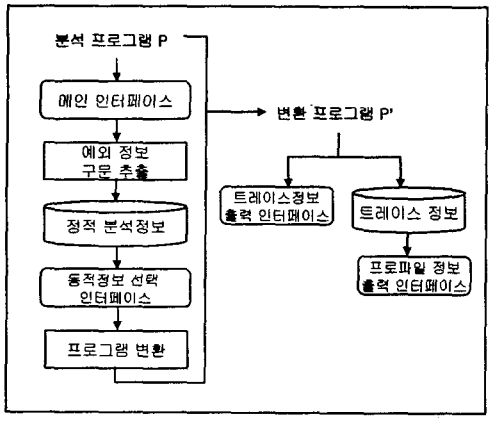
본 논문에서는 예외 처리에 대한 분석 정보를 크게 세 가지로 나누고 있다. 첫째는 컴파일 시간에 분석 가능한 정적 정보로 주로 예외 관련 프로그램 구조를 보여준다. 이 정보를 이용하여 사용자는 관심 있는 부분 혹은 정보만을 선택함으로써 옵션을 설정할 수 있다. 둘째는 프로그램 실행 중의 예외 발생과 처리에 대한 트레이스 정보로 이 정보는 옵션에 따라 변환된 프로그램을 실행하면 얻을 수 있는 정보이다. 세 번째는 프로그램 실행 후에 실행에 대한 요약 정보인 프로파일 정보로 이 역시 변환된 프로그램을 실행하면 실행 직후에 제공된다.

이러한 정보를 제공할 수 있도록 시스템 구조를 크게 세 부분으로 나눴다. (그림 1)은 시스템의 전체적인 구조이다.

첫 단계인 메인 인터페이스는 일종의 시작화면으로 대상 프로그램을 열고 옵션을 설정할 수 있다. 옵션 설정을 위해 입력 프로그램 P를 간단하게 정적 분석하여 프로그램 내의 예외관련 구문들에 대한 정보를 추출한다. 이 정보들을 보고 사용자는 관심 있는 예외, 메쏘드 등

* 본 연구는 한국과학재단 특정기초과제 "정적 분석을 이용한 모바일 자바 프로그램의 효율적인 자원 사용을 위한 환경 연구"(R01-2002-000-003630-0)의 지원에 의해 수행되었음.

에 대한 옵션을 선택을 할 수 있다. 두 번째 단계로 사용자가 옵션을 선택하면 기존의 프로그램과 똑 같이 동작하면서 예외 관련 동적 정보를 출력할 수 있도록 코드를 삽입하여 새로운 프로그램 P'으로 변환한다. 이 부분은 프로그램 변환기에 의해서 이루어진다. 마지막으로 이렇게 변환된 프로그램 P'을 실행시키면 실행 중 혹은 실행 후에 삽입된 코드를 통해서 트레이스 정보와 프로파일 정보가 출력되게 된다.



(그림 1) 프로그램 구조

3. 구현 및 실험 결과

3.1 구현 환경

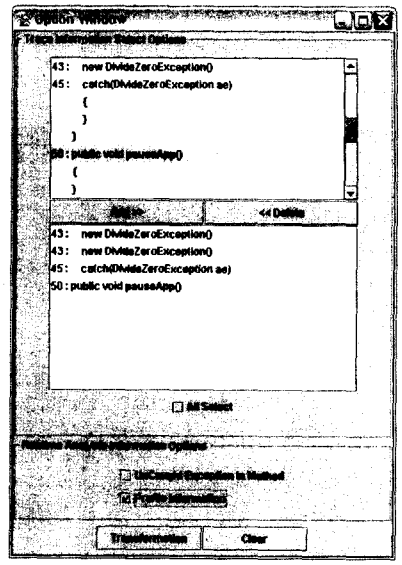
본 연구에서는 프로그램 변환을 위해 자바 컴파일러 전단부인 Barat을 이용하였다. Barat은 자바 소스 파일이나 클래스 파일을 입력 받아 이름과 타입 정보를 포함하는 AST (Abstract Syntax Tree)를 구성한다[1].

이렇게 구성된 AST의 각 노드를 비지터(Visitor)라는 디자인 패턴을 이용한 트리 횡단 루틴을 이용하여 방문하면서 필요한 연산을 수행할 수 있다. Barat이 기본적으로 제공하는 비지터에는 AST의 모든 노드들을 깊이 우선탐색을 해주는 DescendingVisitor, 모든 노드를 검색한 후 AST 정보를 기반으로 소스 코드를 다시 생성해주는 OutputVisitor 등이 있다.

따라서 이러한 비지터들의 메쏘드를 재정의함으로써 Barat이 AST노드를 방문 시 원하는 연산을 수행하도록 재구성할 수 있다. 본 논문에서는 Barat의 Descending-Visitor와 OutputVisitor의 재 정의를 통해서 예외 관련 구문 정보를 추출하거나 프로그램을 변환한다.

3.2 구 현

본 시스템의 구현은 크게 분석 세 단계로 구성된다. 첫 번째인 예외관련 구문 추출은 Barat이 제공하는 DescendingVisitor를 확장하여 StaticVisitor라는 이름으로 구현하였다. DescendingVisitor를 이용하여 프로그램의 AST를 탐색하면서 프로그램 내에 예외관련 구문 구조(예외 발생, 처리, 관련 메쏘드 등)에 대한 정적 정보를 추출한다. 이렇게 분석된 정보를 토대로 (그림 2)와 같이 동적 예외 분석을 위한 옵션 정보를 제공한다.



(그림 2) 동적 정보 분석을 위한 옵션 선택 화면

사용자는 이 정보를 이용하여 관심 있는 예외, 메쏘드 등을 선택함으로써 실행 중 해당 예외 정보만을 선택적으로 추적할 수 있다. 또한 실행 후 보여줄 수 있는 프로파일 정보에 대한 옵션도 제공하여 사용자가 관심 있는 정보만을 요약한 프로파일 정보를 볼 수 있다.

Transformation버튼을 누르게 되면 선택한 옵션에 따른 트레이스 정보만을 출력할 수 있도록 기존의 프로그램에 새로운 코드를 삽입한다. 이 변환 부분은 TransformVisitor라는 이름으로 OutputVisitor를 확장하여 구현하였다. (그림 3)은 TransformVisitor의 간략한 구조로 OutputVisitor를 상속받아서 예외 관련 메쏘드를 재정의함으로써 throw 문, try-catch 문, 메쏘드 등에서 발생, 처리, 전파되는 예외를 옵션에 따라 추적할 수 있도록 코드를 삽입하여 입력 프로그램을 변환한다.

```
Class TransformVisitor extends OutputVisitor{
  visitThrow{
        // Exception 발생 위치, 발생 Exception타입 정보
  }

  visitTry{
        // Try문 내의 Exception발생 메소드의 위치, 메소드 이름

  visitCatch{
        // Exception 처리 위치, 처리한 Exception의 type
          처리된 Exception 오브젝트를 통한 printStackTrace()호출
  }

visitConcreteMethod{
        //Exception을 발생시킨 메소드의 전파과정 분석
        // return propagation
  }
}
```

(그림 3) TransformVisitor의 구조

이 변환 된 소스 코드를 J2SE프로그램일 경우에는

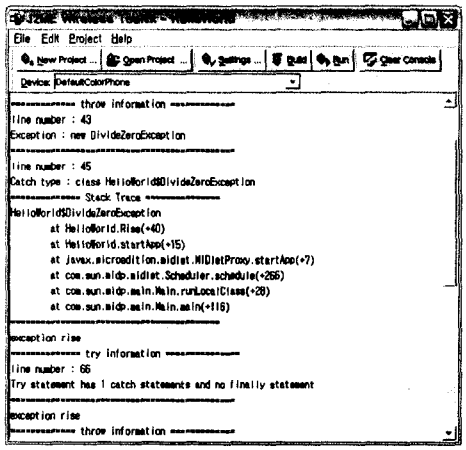
SDK 내에서 J2ME 프로그램일 경우에는 Wireless Toolkit에서 실행시킨다. 사용자가 선택한 트레이스 정보는 프로그램 실행 중에 실시간으로 출력되게 되며 실행 종료 후에는 전체 프로파일 정보가 출력된다.

3.3 실험

위에서 언급했듯이 StaticVisitor를 통해 소스 코드내의 예외 관련 구문들을 분석하여 (그림 2)와 같이 옵션을 받고 TransformVisitor를 통해 아래 (그림 4)와 같이 기존의 입력 프로그램을 변환하였다. 이러한 소스 변환을 통해 트레이스 정보뿐만 아니라 프로그램 실행 종료 후에 실행 중에 예외 처리에 대한 요약 정보인 프로파일 정보까지도 제공할 수 있다.

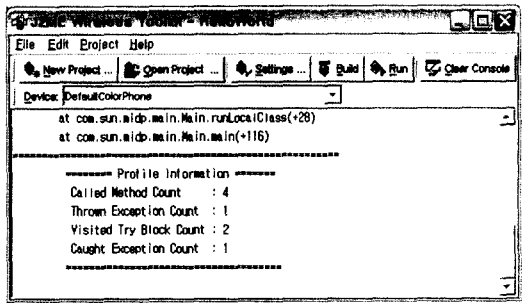
```
public class HelloWorld extends javax.microedition.midlet.MIDlet
  implements javax.microedition.lcdui.CommandListener {

  Profile_Info pi = new Profile_Info();
        :
        :
  public void Rise() throws HelloWorld.DivideZeroException {
     pi.method_prof();
     try {
     pi.try_prof();

     System.out.println("============= throw information =============");
     System.out.println("line number : 43");
     System.out.println("Exception : new DivideZeroException");
     System.out.println("=============================================");
     pi.throw_prof();
     throw new DivideZeroException();
     } catch (DivideZeroException ae) {

     System.out.println("============= Catch Information =============");
     System.out.println("line number : 45");
     System.out.println("Catch type : " +ae.getClass());
     System.out.println("                < Stack Trace >              ");
     ae.printStackTrace();
     System.out.println("=============================================");

     pi.catch_prof();
     }
     public void destroyApp(boolean unconditional) {
        pi.method_prof();
        pi.printProfile();
     }
  }
```

(그림 4) TransformVisitor로 변환된 프로그램



(그림 5) 변환된 소스코드 실행 결과 화면

(그림 4)와 같이 변환된 소스 코드를 Wireless Toolkit을 통해 실행시키면 KToolBar의 커맨드 라인 창에 (그림 5)와 같이 사용자가 선택한 정보가 보여 지게 된다. 프로그램이 종료되고 사용자가 옵션 선택에서 프로파일 정보를 선택했다면 (그림 6)과 같이 그 프로그램의 실행 결과에 대한 프로파일 정보를 보여준다.



(그림 6) 프로파일 결과 화면

4. 결론 및 향후 과제

(그림 4)에서 보았듯이 현재 구현된 시스템은 기존의 프로그램에 예외 관련 동적 분석을 위한 코드를 추가하여 실행한다. 이러한 동적 분석을 통해 기존의 정적 분석에서 제공할 수 없는 보다 정확한 예외 정보나 예외 전파 과정 등을 실시간으로 살펴볼 수 있으며 프로그램 실행 후에 실행 중 예외 발생, 처리 등에 관한 요약 정보를 얻을 수 있다. 또한 본 시스템은 J2SE뿐만 아니라 J2ME 프로그램의 분석도 가능하게 하였다.

현재는 프로그램의 변환 관련 부분만이 구현되었으며 변환된 프로그램은 수동으로 실행해야 한다. 앞으로 변환된 프로그램을 사용자에게 보이지 않으면서 자동적으로 실행하고 실행 결과인 트레이스 정보나 프로파일 정보를 별도로 시각화해서 보여주는 등의 연구를 계속 수행할 계획이다.

5. 참고문헌

[1] Boris Bokowski, André Spiegel. Barat A Front-End for Java. Technical Report B-98-09
December 1998

[2] Bruno Dufour, Karel Driesen, Laurie Hendren and Clark Verbrugge. Dynamic Metrics for Java. ACM OOPSLA '03, October, 2003, Anaheim, CA.

[3] B.-M. Chang, J. Jo, and S. Her, Visualization of Exception Propagation for Java using Static Analysis, Proceedings of IEEE Workshop on Source Code Analysis and Manipulation, Oct. 2002.

[4]B. G. Ryder, D. Smith, U. Kremer, M. Gordon, and N. Shah, "A static study of Java Exceptions using JESP," Tech. Rep. DCS-TR-403, Rutgers University, Nov. 1999