신뢰성 높은 Java 프로그램 개발을 위한 예외 모니터링 시스템

오 희 정[†] · 창 병 모[††]

요 약

자바 언어는 신뢰성 있는 프로그램의 개발을 위해 예외를 처리할 수 있는 명시적인 예외 처리 메커니즘을 제공한다. 따라서 신뢰성 있는 프로그램 개발을 위해서는 실행 시간에 발생 가능한 예외에 대한 적절한 처리가 매우 중요하다. 본 논문에서는 신뢰성 있는 프로그램 개발을 위한 동적 예외 모니터링 시스템을 개발하였다. 이 시스템은 프로그래머가 효과적으로 실제 발생된 예외의 처리, 전파과정을 모니터링 할 수 있으며 이러한 기능을 이용하여 좀더 적절한 예외 처리가 가능하도록 하며 결과적으로 개발된 프로그램의 신뢰성을 향상시킬 수 있다. 본 시스템은 사용자 옵션을 통하여 관심 있는 예외들만을 모니터링 할 수 있으며 또한 시스템의 성능 부하를 줄이기 위해 기존의 코드에 필요한 모니터링 코드만를 삽입하는 방식으로 시스템을 설계하고 구현하였다. 또한 실험을 통해서 그 효과를 보였다.

키워드 : Java, 예외, 모니터링, 프로파일

An Exception Monitoring System for Developing Reliable Java Programs

Heejung Ohe[†] · Byeong-Mo Chang[††]

ABSTRACT

Exception mechanism in Java allows programmers to define, throw and catch exceptional conditions. It is important for the development of reliable programs to make sure that exceptions are handled appropriately at run-time. In this paper, we develop an exception monitoring system, which can help programmers trace and handle exceptions effectively. Using this system, programmers can examine exception handling process in detail by tracing only interesting exceptions, and can handle exceptions more effectively. To reduce performance overhead, we design the system based on code instrumentation. Moreover, we show the effectiveness of the system by experiments on benchmark programs.

Key Words : Java, Exception, Monitoring, Profile

1. 서 론

자바에서의 예외처리 메커니즘은 프로그래머가 예외에 대한 발생과 처리를 가능하게 한다[6]. 자바 프로그램에서 발생된 예외가 적절히 처리되지 못하면 실행중인 프로그램을 중단시키는 원인이 되므로 신뢰성 있는 프로그램의 개발에 있어서 실행시간에 발생되는 예외에 대한 적절한 처리는 매우 중요하다. 하지만 프로그램 내의 발생되는 모든 예외에 대해서 프로그래머가 효율적으로 분석하고 처리하는 것은 쉬운 일이 아니다. 그 동안 이러한 예외에 대한 연구는 정적 분석을 중심으로 이루어져 왔으며 이러한 연구들은 정적 분석을 통해 모든 발생 가능한 예외에 대한 정보를 제공 한다[1, 2, 7, 8]. 그러나 이 방법은 프로그램 실행 중에 실제 발생하는 예외의 처리, 전파 등에 대한 정확한 정보를 제공해 주지 못할 뿐만 아니라 실행시간 예외(runtime exception)와 같은 비검사 예외(unchecked exception)에 대한 정보를 제공해주지 못하는 단점이 있다.

본 연구에서는 프로그래머에게 좀더 효율적으로 실제 발생된 예외의 처리 및 전파 과정에 대한 정보를 제공해 줄 수 있는 예외 모니터링 시스템을 개발한다. 이 시스템은 프로그램 내의 사용자가 관심 있는 예외만을 옵션을 통해 선택할 수 있으며 이를 바탕으로 선택된 예외의 실제 발생, 처리, 전파 등을 실시간으로 모니터링 한다. 이 시스템을 이용하면 좀 더 세밀하고 동적인 예외 분석 및 처리가 가능하며 이를 통해 개발된 프로그램의 신뢰성을 향상할 수 있다. 또한 실행 직후에는 프로그램이 실행되는 동안 호출된 메소드별로 각 메소드 내에서 처리된 예외들의 정보를 요약한 프로파일 정보를 제공한다.

본 연구에서는 동적 분석에 의한 실행 시간의 부담을 줄

† 정 회 원 : LG전자 연구원
†† 정 회 원 : 숙명여자대학교 컴퓨터과학전공 교수
논문접수 : 2005년 6월 14일, 심사완료 : 2005년 9월 12일

이기 위해 코드 삽입(code inlining) 기법을 기반으로 시스템을 구현하였으며 사용자 옵션에 따라 관심 있는 예외들만을 모니터링 하는 코드를 삽입하고 변환된 프로그램을 실행함으로써 동적으로 모니터링 할 수 있도록 구현하였다. 이 방법은 사용자가 관심 있는 부분만 모니터링 할 수 있을 뿐만 아니라 모니터링으로 인한 성능 저하도 크게 줄일 수 있다.

2장에서는 연구 배경에 대해서 기술한다. 3장에서는 시스템의 전반적인 구조를 4장에서는 시스템의 구현에 대해서 기술한다. 5장에서는 실험결과를 정리 하였으며 6장에서는 관련 연구와 비교한다. 7장은 결론과 향후 연구 과제 등에 대한 논의한다.

2. 연구 배경

예외 전파를 보여주는 (그림 1)의 간단한 예를 살펴보자. 메쏘드 m2로부터 발생된 예외 E1은 m2와 m1를 거쳐 main 메쏘드까지 전파되며 main 메쏘드 내에서 처리된다. 예외 E2는 메쏘드 m3로부터 발생될 수 있으며 만약 발생되면 main 메쏘드까지 전파되며 끝내 처리되지 않는다. 메쏘드 m3는 또한 자기 자신에 대한 호출을 가지고 있어서 발생된 예외 E2는 자기 호출을 따라 역으로 전파된다.

처리되지 못한 예외는 프로그램 실행을 중단시키므로 신뢰성 있는 프로그램 개발을 위해 실행시간에 발생되는 예외에 대한 적절한 처리는 매우 중요하다. 현존하는 정적 예외 분석 방식은 각 메소드 내에 처리 되지 못하는 예외에 대한 모든 가능한 정보를 제공한다. 그러나 이러한 정적 분석 방식은 실행시간에 실제로 발생된 예외에 대한 처리 및 전파 정보를 제공할 수 없다[1, 2, 7, 8].

J2ME WTK와 같은 기존의 동적 분석 툴의 경우도 단지 예외가 발생할 경우 발생된 예외의 이름 정보만 제공해주기

```
class Demo{
   public static void main(String[] args ) throws E2
   {
         try {
                  m1( );
         } catch (E1 x) { ; }
                  . . .
                  m3( );
         }
   void m1( ) throws E1{
         m2( );
   }
   void m2( ) throws E1{
         if (...) throw new E1();
   }
   void m3( ) throws E2 {
         if (...) throw new E2();
         if (...) m3( );
   }
}
```

(그림 1) 예외 전파 예

때문에 발생된 예외에 대한 트레이스 정보나 예외 전파 과정에 대한 정보를 제공해주는 기능이 없다. 신뢰성 있는 자바 프로그램의 개발을 위해서는 이러한 예외 모니터링 기능을 제공하는 적합한 툴이 필요하며 이러한 모니터링 기능을 이용하여 실제 발생한 예외에 대한 보다 상세한 추적이 가능하며 이를 통해 개발된 프로그램의 신뢰성을 향상시킬 수 있을 것이다.

따라서 본 연구에서는 프로그래머가 좀더 효율적으로 예외를 처리할 수 있도록 도와주는 동적 예외 모니터링 시스템을 개발하였다. 이 시스템은 실행시간에 예외 처리 과정을 모니터링하며 발생된 예외의 전파 경로를 보여준다. 또한 성능 부하를 줄이기 위해 JVMPI를 사용하는 대신에 분석하고자 하는 프로그램의 코드에 모니터링을 위한 추가 코드를 삽입한다. 사용자는 옵션 선택을 통해 관심 있는 예외만을 선택해서 트레이스 할 수 있다. 또한 실행 후에 실행 시간 동안 처리된 예외 정보를 요약하여 프로파일 정보를 제공한다. 이 시스템이 제공하는 기능들은 프로그래머가 좀더 효과적으로 예외를 처리할 수 있도록 도와주며 결과적으로 프로그램의 신뢰성을 향상시킬 수 있다.

3. 설 계

신뢰성 있는 자바 프로그램 개발을 위한 동적 예외 모니터링 시스템 설계에서는 다음 사항들을 고려하였다.

3.1 사용자 옵션

사용자가 관심 있는 예외를 선택할 수 있는 사용자 옵션을 제공한다. 이 시스템은 사용자가 코드를 변환하기 전에 관심 있는 예외 혹은 메소드를 선택함으로서 프로그램 내의 관심 있는 부분만을 모니터링 할 수 있다. 따라서 이러한 옵션 기능은 모든 예외를 모니터링하지 않고 사용자가 원하는 부분만 모니터링 함으로써 성능 부하를 줄이는 데 기여할 수 있다.

3.2 모니터링 오버헤드 최소화

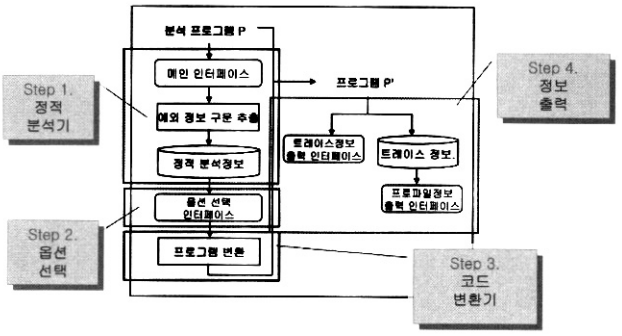
실행 시간 부하를 줄이기 위하여 JVMPI를 이용하는 대신에 추가 코드를 기존 코드에 삽입하는 코드 삽입 방법을 사용하였다. 시스템은 입력 프로그램 P를 받아서 사용자 옵션에 따라 관심 있는 예외만을 모니터링 하는 코드를 삽입하여 새로운 프로그램 P'으로 변환한다.

3.3 모니터링 및 프로파일 생성

변환된 프로그램을 실행하면 프로그램이 실행 중에 실제 발생된 예외의 처리 및 전파 과정을 시각적으로 보여준다. 또한 프로그램 실행 후에는 각 메쏘드별로 발생, 처리, 전파된 예외들에 대한 통계 자료인 프로파일 정보를 제공한다.

이 시스템은 (그림 2)처럼 4단계로 구성되며 각 단계의 기능은 아래와 같다.

• 첫 번째 단계는 정적분석을 통해 예외 관련 구문들을



(그림 2) 시스템 구조

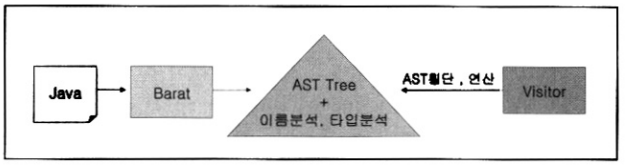
추출해 낸다. 이렇게 정적으로 분석된 예외 관련 정보를 이용하여 옵션을 제공한다.

- 두 번째 단계는 전 단계의 정적분석 정보를 이용하여 옵션 선택 기능을 제공하며 사용자는 관심 있는 예외나 메소드를 선택할 수 있다. 따라서 사용자는 이 단계에서의 옵션 선택을 통해 관심 있는 예외만을 모니터링하도록 할 수 있다.
- 세 번째 단계는 사용자가 선택한 옵션을 바탕으로 입력 프로그램 P에 모니터링을 위한 코드를 삽입하여 새로운 프로그램 P'을 생성한다.
- 네 번째 단계는 변환된 프로그램 P'을 컴파일하고 실행하는 과정이다. 실행은 SDK에서 이루어진다. 변환된 프로그램의 실행을 통해 실행시간동안 발생된 예외의 처리 및 전파 과정에 대한 정보를 실시간으로 모니터링 할 수 있으며 실행 후에는 프로파일 정보가 제공된다.

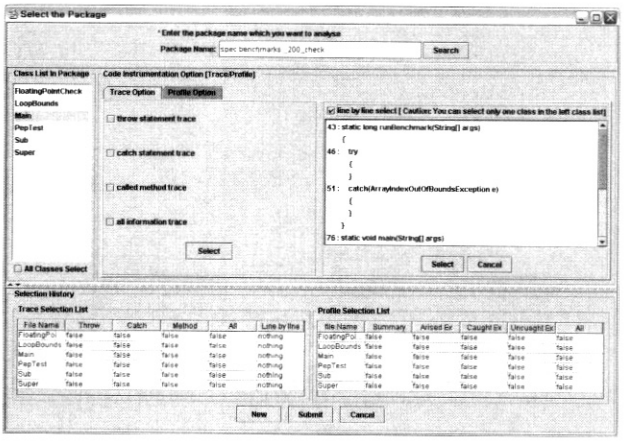
4. 구 현

본 연구에서는 이 시스템을 Barat이라는 프로그램 분석기 구현 툴을 기반으로 하여 구현하였다[3]. 이 툴을 이용하여 프로그램 내의 예외 관련 구문을 추출하는 정적 분석기를 구현하였으며 모니터링 코드를 삽입하는 프로그램 변환 단계를 구현한다. Barat은 자바 컴파일러의 전단부로 자바 언어를 위한 정적 분석기 구현을 지원한다[3]. Barat은 자바 소스나 바이트 코드 프로그램을 입력으로 받아서 이름과 타입 분석 정보를 포함하는 AST(Abstract Syntax Tree)를 구성한다.

이렇게 구성된 AST의 각 노드를 비지터(Visitor)라는 디자인 패턴을 이용한 트리 횡단 루틴을 이용하여 방문하면서 필요한 작업을 수행할 수 있다. 본 연구 에서는 Barat이 제공하는 기본적인 비지터를 확장하여 예외 관련 구문을 추출



(그림 3) Barat 구조



(그림 4) 옵션 선택

하는 정적 분석기를 구현하였으며 또한 실행시간에 실제 발생된 예외의 처리 및 전파 과정을 모니터링 하도록 입력 프로그램을 변환하는 변환기를 구현하였다.

이 시스템은 (그림 2)에서처럼 4단계로 구성되어있다. 본 연구에서는 앞의 3단계를 구현하였다. 마지막 단계는 Java SDK 상에서 실행된다. 첫 번째 단계를 위해 Barat이 제공하는 DescendingVisitor를 확장하여 프로그램 내의 예외 관련 구문 정보를 추출하는 정적분석기를 구현하였다. 이 단계에서는 정적 분석을 통하여 발생 가능한 예외와 메소드에 대한 정적 정보를 추출해 낸다. 특히 프로그램 내의 예외 발생, 처리 및 메소드에 대한 정적 정보를 추출하며 이 정보는 옵션을 제공하는데 사용된다.

두 번째 단계를 위해 (그림 4)에서처럼 옵션 선택을 위한 메뉴 창을 구성한다. 이 메뉴 창은 정적 분석 정보를 기반으로 예외, 처리기, 메소드 등을 보여준다. 사용자는 이 중에 관심 있는 예외나 처리기, 메소드를 선택할 수 있다. 옵션 선택을 통해 사용자는 관심 있는 예외 정보만을 선택할 수 있다.

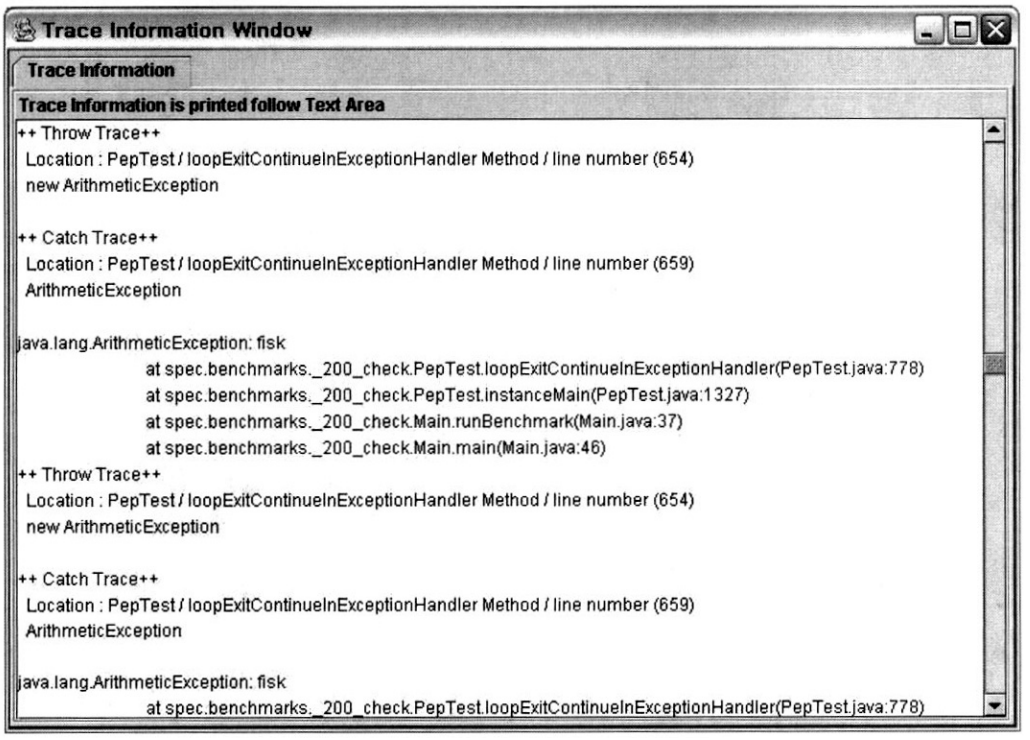
세 번째 단계를 위해서 모니터링 코드를 삽입하여 새로운 프로그램을 생성하도록 OutputVisitor를 확장하여 Transform-

```
class TransformVisotor extends OutputVisitor {
                                    ...
    visitThrow {
            // 예외 타입, 발생 위치 등 출력
    }
    visitTry {
            // try 문 내에서 발생된 예외 타입, 위치, 메쏘드 이
름 등 출력
    }
    visitCatch {
            // 처리된 예외 타입,  위치, 스택 트레이스 등 출력
    }
    visitConcreteMethod {
            // 발생된 예외의 메쏘드를 통한 전파과정 출력
    }
                                    ...
}
```

(그림 5) TransformVisitor의 구조

```
    synchronized int syncMethod2(int y) throws ArithmeticException {

        pi.print_methodtrace("syncTest","syncMethod2","27","synchronized null syncMethod2(int y) throws Arithmet
        pi.methodcollect("syncTest","syncMethod2","|ArithmeticException|");
      this.x = this.x + y;

      if (this.x == 99){
        pi.print_throwtrace("syncTest","syncMethod2","30","new ArithmeticException");
        pi.set_ThrowInfo("syncTest" , "syncMethod2" , "30" ,"new ArithmeticException");
          throw (new ArithmeticException("fisk"));
      }

      return this.x;
    }

    public static void main(String[] args) {
          try{

        pi.print_methodtrace("syncTest","main","34","static void main(String[] args)");
        pi.methodcollect("syncTest","main","|");
      syncTest sy = new syncTest();
      int xx = sy.syncMethod(4);
      xx = sy.syncMethod2(4);
          }catch(Exception apple)  {
              apple.printStackTrace();
          }
          finally{
              pi.print_Profile();
          }
    }
```

(그림 6) 변환된 프로그램의 예

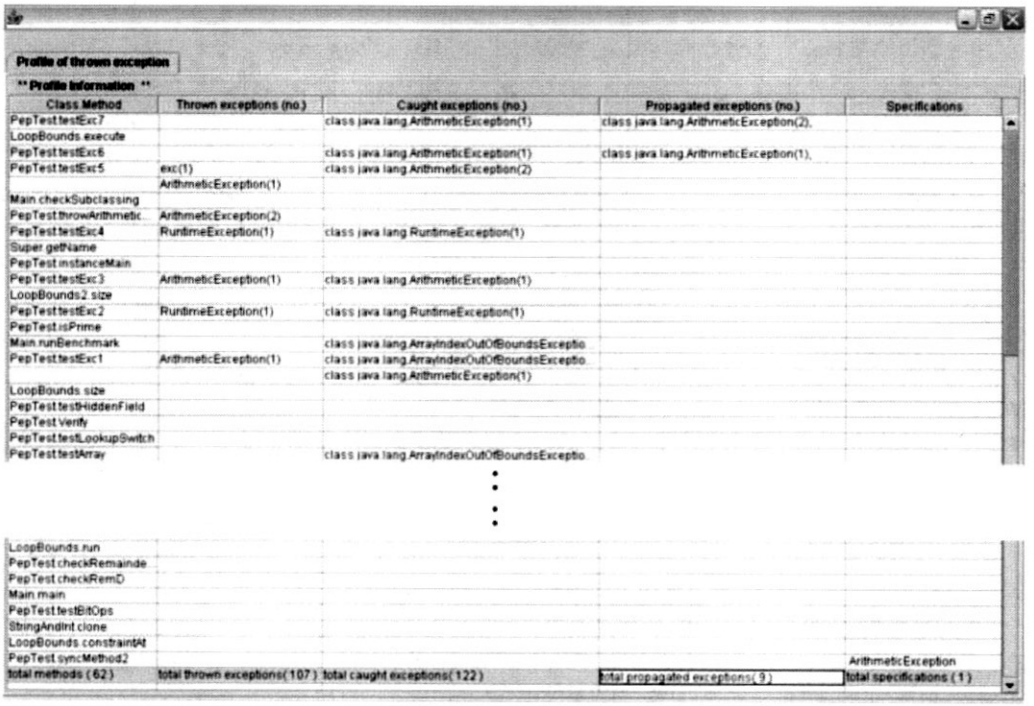


(그림 7) 모니터링 정보 출력 화면

Visitor를 구현한다. TransformVisitor는 분석하려는 프로그램 P를 입력 받아 옵션 선택에 따라 사용자가 관심 있는 예외들만을 모니터링할 수 있도록 추가적인 코드를 삽입하여 새로운 프로그램 P'을 생성한다. (그림 5)에서는 프로그램 변환기의 간략한 구조를 보여준다.

(그림 6)은 TransformVisitor를 통해 변환된 코드의 예이다. 박스 안에 있는 코드가 TransformVisitor를 통해 추가적으로 삽입된 코드로 예외의 발생, 처리, 전파과정을 모니터링 한다. 이렇게 변환된 프로그램은 Java SDK 상에서 자동으로 실행된다.

이렇게 변환된 프로그램은 실행 중에 옵션에서 선택된 예외들 중 실제 발생되는 예외들에 대한 처리와 전파 과정을 모니터링 해준다. 또한 프로그램의 실행이 종료된 후에는 실행 시간동안 각 메소드별로 발생된 예외의 처리 정보를 요약한 프로파일을 제공한다. 프로파일은 프로그램이 실행되면서 호출된 각 메소드별로 예외 발생, 처리 및 메소드



(그림 8) 프로파일 정보 출력화면

내에서 처리 되지 않고 전파된 예외들에 대한 통계 정보를 제공한다.

5. 실험 결과

본 연구에서 구현은 Pentium 4 프로세서/Window XP 기반의 SDK 1.4.2 버전을 기반으로 하였으며 실험을 통해 구현된 시스템의 유용성을 실험하였다. 먼저 specjvm98의 Check 프로그램을 이용해 구현된 시스템을 시험하였다. (그림 7)과 같이 옵션에서 선택된 예외들을 중심으로 실제로 발생된 예외들에 대한 처리 및 전파과정을 모니터링 할 수 있다. 이 그림은 발생된 예외의 위치와 타입을 보여준다. 또한 전파된 예외가 catch-절에 의해 처리되면 전파 경로를 보여준다. (그림 7)은 예를 들어 659번째 줄에서 처리된 ArithmeticException의 전파경로를 보여주고 있다.

프로그램이 종료되면 (그림 7)과 같이 프로그램 실행시간 동안 각 메소드별로 발생된 예외, 처리 혹은 처리 되지 못하고 전파된 예외에 대한 프로파일 정보를 제공한다.

본 연구에서는 5개의 자바 벤치마크 프로그램을 이용하여 실험하였다. 첫 번째 프로그램은 서버 연결 프로그램인 Server-Stuff이다. 두 번째는 1차 방정식 해를 구하는 Linkpack 벤치마크 프로그램이다. 세 번째 프로그램은 Specjvm98의 Ckeck 프로그램으로 JVM이 벤치마크의 요구에 적합한지를 검사한다. 네 번째는 Specjvm98의 Jess 프로그램으로 전문가 시스템 쉘이다. 다섯 번째는 gnu의 Rex 프로그램으로 입력 스트링을 정규 표현식에 매치시킨다.

<표 1>의 테이블에서 첫 번째 열은 벤치마크 프로그램의 코드 변환 전의 크기와 코드 변환후의 크기를 비교해 놓았다. 두 번째 열은 발생된 예외의 개수, 세 번째 열은 프로그램 내에서 처리된 예외의 수, 마지막 열은 발생된 메소드 내에서 바로 처리되지 못하고 전파된 예외의 수를 각 벤치마크 프로그램 별로 정리하였다.

〈표 1〉 벤치마크 프로그램 실험 결과

프로그램	줄 수		throw 예외 개수	caught 예외 개수	uncaught 예외 개수
	변환 전	변환 후			
Server Stuff	71	100	2	2	4
Linpack	1057	1103	1	1	17
Check	1817	2429	107	121	9
Jess	542	574	1	1	10
Rex	3198	3308	1	1	6

만약 예외가 발생되고 전파 되었다면 예외가 전파된 각 메소드에서의 *uncauaght exception*의 수가 증가된다. *Jess*의 실험 결과의 경우 올바르지 않은 문장이 입력 값으로 들어왔을 경우 예외를 발생시키고 발생된 예외에 대한 전파 결과를 보여주고 있다. *Rex*의 경우도 잘못된 옵션을 입력한 경우 발생된 예외에 대한 전파 결과를 보여주고 있다. 벤치마크 프로그램 중 *Check*의 경우 프로그램 내에 많은 수의 예외 발생과 발생된 예외의 처리 및 전파과정을 볼 수 있다.

6. 관련 연구

예외 분석에 대한 기존의 연구들은 정적 분석을 중심으로 한 연구가 대부분이다[1, 2, 7, 8]. 정적 예외 분석은 실행 전

에 입력 프로그램을 분석하여 각 메소드별로 발생될 수 있는 모든 가능한 예외를 분석함으로서 개략적인 정보를 제공한다. 자바 컴파일러도 프로그래머가 선언한 처리되지 않는 예외에 대한 명세를 중심으로 처리되지 않는 예외를 정적 분석한다[6]. [1, 2, 7]에서는 프로그래머가 선언한 예외 명세와는 독립적으로 처리되지 않은 모든 가능한 예외를 분석하는 프로시저-간 예외 분석 방법이 제안되었다.

그러나 정적 분석 기술은 프로그램 실행 중에 실제 발생된 예외의 처리 및 전파 과정에 대한 정보를 제공해 주지는 못한다. 따라서 실제 실행시간에 발생되는 정보를 제공해 주기 위한 동적 분석 기술이 연구되어 왔다[4, 9, 10]. 그동안 J2ME WTK[11]과 AdaptJ[9]을 포함하여 자바를 위한 동적 분석 툴들이 개발되어 왔다. 최근 J2ME WTK은 JVMPI를 이용하여 메소드 호출, 예외, 메모리 사용 등과 같은 트레이스 정보를 제공한다. 하지만 이 시스템은 예외가 발생되면 단지 발생된 예외의 이름만을 제공해준다. 또한 JVMPI는 성능 면에 있어서 많은 부담을 발생시켜 실행속도를 매우 저하 시키고 라이브러리들을 포함하여 모든 코드가 트레이스 정보에 포함되므로 프로그램 내의 관심 있는 부분만을 모니터링 하기 매우 어렵다. AdaptJ는 실행 시간동안 JVMPI를 통해 동적 정보를 수집하고 프로그램 실행 후에 동적 정보를 파일에 저장한다. 그러나 예외와 관련된 모니터링 기능은 제공하지 않는다.

7. 결 론

본 논문에서는 사용자가 관심 있는 예외들을 중심으로 실제 발생된 예외의 처리 및 전파 과정을 모니터링 할 수 있는 동적 예외 모니터링 시스템을 개발하였다. 이 시스템을 이용하여 프로그래머는 관심 있는 예외만을 모니터링 함으로써 예외 처리 과정을 보다 자세히 살펴 볼 수 있으며 이를 통해 보다 효과적으로 예외를 처리할 수 있을 것이다. 이는 결과적으로 개발된 프로그램의 신뢰성 향상에 기여할 수 있을 것이다. 이 시스템은 두 가지 방향으로 확장될 수 있을 것이다. 하나는 예외 모니터링과 프로파일 정보를 좀 더 사용자들이 쉽게 이해할 수 있도록 시각적으로 표현하는 것이다. 또 하나는 이 시스템을 모바일 환경에서 널리 사용되고 있는 J2ME 프로그램도 처리할 수 있도록 확장하는 것이다.

참 고 문 헌

[1] 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.

[2] B.-M Chang, J. Jo, K. Yi, and K. Choe, "Interprocedural Exception analysis for java," Proceedings of ACM Symposium on Applied Computing, pp.620-625, Mar., 2001.

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

[4] B. Dufour, K. Driesen, L. Hendren and C. Verbrugge. Dynamic Metrics for Java. ACM OOPSLA '03, Anaheim, CA, October, 2003.

[5] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns:Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[6] J. Gosling, B. Joy and G. Steele, "The Java Programming Language Specification," Addison-Wesley. 1996.

[7] M. P. Robillard and G. C. Murphy, Analyzing exception flow in Java programs, ACM SIGSOFT Symposium on Foundation of Software Engineering, pp.322-337.

[8] 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.

[9] AdaptJ:A Dynamic Application Profiling Toolkit for Java, http://www.sable.mcgill.ca/~bdufou1/AdaptJ.

[10] Sun Microsystems, J2ME Wireless Toolkit, http://java.sun.com.

오 희 정

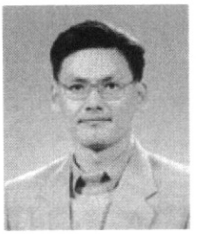
e-mail : lutino@sookmyung.ac.kr

2003년 숙명여자대학교 컴퓨터과학과 (이학사)

2005년 숙명여자대학교 컴퓨터과학과 (이학석사)

2005년~현재 LG전자 연구원

관심분야 : 프로그래밍 시스템



창 병 모

e-mail : chang@sookmyung.ac.kr

1988년 서울대학교 컴퓨터공학과(공학사)

1990년 한국과학기술원 전산학과(공학석사)

1990년 한국과학기술원 전산학과(공학박사)

1993년~1994년 한국전자통신연구원 박사후 연구원

1994년~현재 숙명여자대학교 컴퓨터과학전공 교수

관심분야 : 프로그래밍 시스템, 프로그램 분석