폴트 삽입 방식을 통한 자바 가상 기계의 강건성 테스팅(Robustness Testing) 기법

이성민,김상운,강제성,권용래
한국과학기술원 전자전산학과
{smlee,swkim,jskang,kwon}@salmosa.kaist.ac.kr

Robustness Testing of Java Virtual Machine using fault injection

Sung-Min Lee[0], Sang-Un Kim,Jae-Sung Kang, Yong-Rae Kwon
Dept. of Electronic Engineering and Computer Science,KAIST

요 약

Java의 경우 기존의 강건성 테스팅 방법인 ballista로는 객체 타입의 인자 및 파일 시스템의 변화를 시뮬레이션 할 수 없다. 따라서 객체에 대한 특별한 접근을 필요로 한다. 본 논문에서는 객체에 폴트를 삽입하는 방식을 통하여 자바 가상 기계의 강건성(Robustness)을 테스팅하는 방법을 제안한다. 자바 디버깅 툴인 JPDA를 사용하여 자바 가상 머신에 대한 직접적인 접근 및 의도한 상태 및 환경 변경을 가능하게 하고 이를 통해 자바 가상 기계에 폴트가 심어진 객체를 수행시킬 수 있다. 객체를 수행시키고 의도한 예외상황의 발생 유무를 관찰함으로써 자바 가상 기계의 강건성을 테스트 할 수 있다.

1. 서론

자바 가상 기계는 윈도우나 리눅스와 같은 OS 레벨에서 볼 때 응용프로그램 레벨에서 실행되는 하나의 프로그램이라고 할 수 있다. 하지만 JVM은 상위에서 또 다른 응용프로그램을 수행시키는 것이 주목적이기에 다른 프로그램에서보다 훨씬 높은 강건성(Robustness)을 필요로 한다.

하지만 아직까지 JVM에 대한 강건성을 측정하는 시도가 없었다. 특히 Java 언어 특성상 자체적인 파일 처리 방법을 가지고 있지 않고 JNI로 OS의 파일 처리 루틴에 접근하기 때문에, JVM 과 OS 간의 interaction에서 오는 문제들은 명확하게 예외상황 처리(exception handling)를 하기 힘들다. 따라서 JVM의 여러 구성요소 중 예외상황(exception)이 많이 발생하는 IO 패키지에 대한 강건성을 살펴보는 것이 중요하다고 할 수 있다. 이 논문에서는 JVM에 직접 접근, 객체에 폴트를 삽입하는 방식을 통하여 JVM I/O 패키지의 강건성을 테스트하는 방법을 제시하고자 한다.

1.1 강건성 테스팅 (Robustness Testing)

강건성(Robustness)이란 예외적인 입력(exceptional inputs) 또는 스트레스가 강한 환경(stressful environment) 속에서 소프트웨어 컴포넌트가 올바르게 작동하는 정도[1]를 나타낸다. 따라서 강건성 테스팅은 예외상황 및 스트레스 환경 속에서 프로그램이 정상작동을 하는지, 오류가 발생 하는지를 측정하는 테스팅 방법이다. 또한 이 테스팅은 미리 규정된 요구사항에서 제시하는 상황이 아닌 특정 시나리오 상황에서 프로그램을 테스트 한다.

강건성 테스팅이 중요한 이유는 개발한 프로그램의 강건성 부족이 해당 프로그램뿐만이 아닌 시스템 전체를 복구불능 상태로 만들 수 있기 때문이다.

2. 관련 연구

Ballista[2][3][4][5]는 Carnegie Mellon 대학에서 개발한 테스팅 자동화 도구로, 소스 코드에 대해 접근하지 않고 POSIX (Portable Operating System Interface) 기반의 OS API를 테스팅하는 도구이다.

Ballista에서 사용한 강건성 테스팅 방법은 테스트하고자 하는 API에 유효한(valid)한 값과 유효하지 않은(invalid) 값을 조합하여 인자로 사용한다. 또한 테스트 케이스마다 테스트할 모듈(module under test: MUT)을 한 번씩 호출하여 모듈의 인자가 가지는 값에 대하여 강건한 예외상황 처리를 해 주는지 테스트한다. 이때 인자에 사용되는 값은 모듈에 건네지는 인자의 자료 형에 따라 미리 정의한 값들에서 얻는다.

즉, Ballista에서 사용하는 테스팅 방식은 하나의 test value를 테스트할 모듈의 인자로 넣고 모듈을 호출하여 강건성 테스팅하고, 다시 인자 값을 바꿔 가면서 모듈을 계속 호출함으로써 전체 시스템이 올바로 동작하는지

테스트한다. 이때 모듈이 제대로 작동하는지에 대해서는 판단하지 않고 주어진 테스트 케이스를 실행했을 때 시스템이 정상작동 하는지 혹은 비정상적으로 종료하는지 알아내는 것이다. 시스템의 오류 타입은 다음의 치명적 오류 (Catastrophic failure), 시스템 재 시작 (Restart failure), 실행 중단 (Abort failure), 예외 상황에 대한 error 메시지를 표시하지 않음 (Silent failure), error의 원인을 잘못 기술함 (Hindering failure) 등의 5가지로 규정한다.

3. JVM 테스팅의 문제점과 해결 방법 (제안)

3.1 발리스타의 한계

CMU의 발리스타 강건성 테스팅 방법은 근본적으로 OS 레벨의 POSIX API만을 대상으로 하며 이 API에 primitive data type만을 인자로 전달한다. 하지만 응용프로그램 레벨인 JVM 상위의 Java API[7]에 대하여 발리스타 방식을 적용하기에는 크게 두 가지의 문제점이 발생한다.

첫째, Java의 경우 primitive data type 뿐만이 아닌 object data type 이 존재한다. 발리스타 방법으로는 객체를 인자의 값으로 전송할 때, 이 객체 인자에 적절한 값을 설정하기 힘들다. 따라서 의도한 예외상황을 발생시키기 힘든 상황이 생긴다. 사전에 결정해 놓은 primitive type과 달리 객체는 각 상황에 따라 상태가 달라지고 그 내부의 변수 값들도 변하게 된다. 또한 해당 객체 내에 또 다른 객체를 가지고 있을 수 있기 때문에, 객체를 사전에 인자 타입으로 정의해놓는 작업은 거의 불가능한 일이라 할 수 있다.

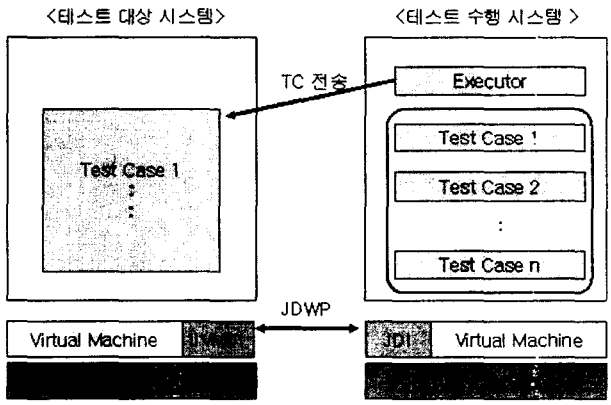
둘째. Java의 경우, 파일처리를 JVM 자체에서 처리하는 것이 아닌 JNI를 통해 OS 레벨에서 처리한다. 따라서 일반 어플리케이션이 OS의 파일 관련 API를 직접 접근하는 것보다 한 단계 더 거치기 때문에 OS 레벨에서의 변화가 응용프로그램의 레벨인 JVM에 예상하지 못한 예외상황을 발생시킬 수 있다. Java에서는 파일 관련 작업이 객체로 이루어 지기 때문에 이와 같은 상황은 발리스타의 파일 테스트로는 확인하기 힘들다. 특히 발리스타에서는 Java가 JNI를 통해 파일 포인터를 얻어 파일을 작성 하는 중이라도 OS 레벨에서 파일의 퍼미션 변동 등에 의한 예외상황이 발생하는 경우를 시뮬레이션화 하기 힘들다.

따라서 인자로 사용되는 객체 및 파일시스템의 변화를 시뮬레이션화 하기 위해서는 JVM으로 테스트하고자 하는 대상이 로딩되어 수행되기 직전에 정지 및 접근하여 그 객체의 상태 및 변수 값 등을 바꿔줘야 한다. 또한 I/O 객체의 경우 JVM이 파일을 생성하거나 열기 위한 작업 등을 수행 중일 때 파일 삭제 및 쓰기 권한의 변경 같은 작업을 허용하는 메커니즘이 필요하다. 이 논문에서는 JVM을 실시간으로 모니터링하고 상태 및 환경을 변경 시킬 수 있는 JPDA를 사용하며, 이를 통해 파일 객체에 대한 접근 및 컨트롤 방법을 살펴본다.

3.2 JPDA를 사용한 문제 접근 방법

JPDA(Java Platform Debugger Architecture)[8]는 가상기계를 컨트롤 하기 위해 만든 다중 계층 디버깅 아키텍처로, 디버깅 대상이 되는 부분(VM)에서 필요한 JVMDI, 디버깅 할 주체에서 필요한 JDI, 대상과 주체의 프로세스를 연결해주는 JDWP, 이렇게 3부분으로 구성되어 있다. 즉 JPDA는 JDI 와 JVMDI간의 프로세스 통신(inter-process communication)을 통한 VM의 전반적인 컨트롤 가능하게 해주는 아키텍처라 할 수 있다.

이 논문에서는 이 디버깅 툴을 실행 순서를 추적하여 변수들의 값을 확인 및 분석하기 위한 도구가 아닌, JVM을 사전에 미리 설정해 놓은 상태로 만들어 놓고, 원하는 부분에서 폴트를 삽입하기 위한 도구로써 사용한다. <그림 1>은 JPDA를 사용한 파일 객체 강건성을 테스팅 하기 위한 시스템이다. 호스트 시스템에서 테스트 대상 시스템을 테스트 케이스와 함께 동작시킨다. 그 후 호스트 시스템은 로컬 혹은 원격으로 테스트 대상 시스템에 JPDA를 통해 접근하여 VM에서 수행하는 테스트 케이스를 실시간으로 조정한다.



<그림 1. java fault injection system >

3.3 파일 객체 테스팅 방법

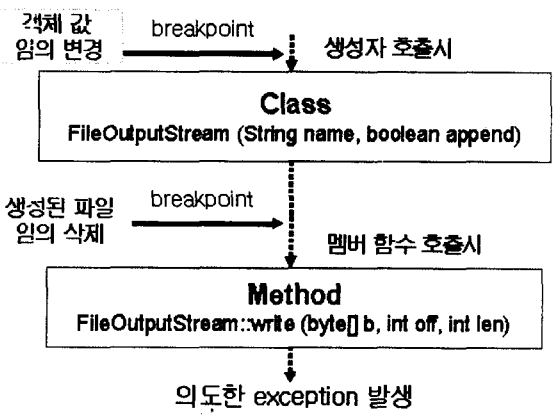
파일 객체에 대한 테스트 범위는 객체의 생성자 및 객체를 생성시키고 난 후 그 객체의 멤버함수 하나만을 대상으로 한다. 테스트 케이스에서는 객체의 생성자를 호출하여 테스트하거나, 객체를 생성한 후 멤버함수를 호출하여 테스팅한다. 이때 객체의 생성자와 멤버함수에는 여러 가지 방식으로 조합된 primitive 및 객체 타입의 인자가 존재한다. 또한 파일 객체는 OS 파일 시스템의 변화에 영향을 받을 수 밖에 없으므로 이를 시뮬레이션하는 방법이 필요하다. 이 모두를 분리하여 각각 테스트 케이스로 구성해야 하는데 <도표 1>은

테스트 케이스에 적용해야 할 테스팅 방법을 나타내고 있다.

대상	Parameter type	테스팅 방법
생성자	X	
	Primitive,[primitive]	사전에 정의된 값을 삽입
	Object , [Object | primitive]	VM에 직접 접근 후 객체 값 변경
멤버함수	X	
	Primitive,[primitive]	사전에 정의된 값을 삽입
	Object ,[Object | primitive]	VM에 직접 접근 후 객체 값 변경. 호스트 파일 시스템의 환경 변경.

< 도표 1. 테스팅 대상 분류 및 테스팅 적용 방법>

테스팅 대상은 생성자와 대상 객체의 멤버함수로 구분한다. 또한 각 함수의 인자 타입에 따라 테스팅 적용 방법이 달라져야 한다. 특히 각 생성자와 멤버 함수에서 객체가 인자로 존재할 경우에 JVM에 대하여 특정 순간에 객체의 입력 값을 변경하는 것과 객체에 영향을 미치는 주변환경을 임의 변경하는 두가지의 직접적인 컨트롤이 절대적으로 필요하다.. <그림 2>는 JVM에 접근하여 어떻게 의도한 예외상황을 발생시키는지를 보여주는 그림이다.



< 그림 2. 테스팅 진행 방식 >

생성자가 인자로 객체를 가지고 있는 경우에는 사전의 설정보다는 breakpoint를 통해 생성자 호출 직전 JVM에 접근 후 객체 인자의 내부 값들을 변경시켜서 의도한 exception을 발생시키기가 훨씬 쉽다. 특히 파일 객체의 경우에는 파일 처리 동작 중에 JVM에 접근하여, JVM의 상태를 정지시켜 해당 파일의 상태를 변경시킨 후 다시 JVM을 동작시키면 수월하게 예외상황을 발생시킬 수 있다.

4. 결론 및 향후 과제.

Java의 경우 객체 타입의 인자가 많기 때문에 기존의 primitive 타입에 기반을 둔 ballista 방법으로 JVM의 강건성 테스팅을 수행하기엔 한계가 있다. 여기서 제시한 필요한 객체에 대한 폴트 입력 방법을 사용하면 이 문제를 해결 할 수 있다.

차후 연구로는 하나의 클래스에 있는 모든 멤버함수를 한번의 객체 생성 후 모두 테스트 할 수 있는 방법을 수행하고자 한다. 이를 위해서는 각 멤버함수가 예외상황 처리를 수행하기 직전과 직후의 객체 상태가 변화하는지를 판단할 수 있는 메커니즘[5]이 필요하다.

5. 참고 문헌

[1] IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990, IEEE CS, Dec. 10,1990.

[2] PhilipKoopman,JohnDevale "The Exception Handling Effectiveness of POSX Operating Systems", IEEE Transactions On Software Engineering", 2000.

[3] ballista testing group 홈페이지 "http://gato.ece.cmu.edu/ostest/"

[4] Nathan P.Kropp, Philip J. Koopman JR., and DanielP. Siewiorek. "automated robustness testing of off-the shelf software components". In Symposium on Fault-Tolerant Computing (FTCS), pages 230-239,1998.

[5]Christof FETZER, Karin HOGSTEDT, Pascal FELBER, "Automatic Detection and Masking of Non-Atomic Exception Handling", Proceedings of the IEEE International Conference on Dependable Systems and Networks, June, 2003,

[6] Joao Carreir, Henrique Madeira,Joao Gabriel Silva, "Xception : A Technique for the Xperimental Evaluation of Dependability in Moden Computers", IEEE Transactions On Software Engineering", 1998

[7] JavaTM 2 SDK, Standard Edition Documentation Version 1.3.1,http://java.sun.com

[8] JPDA (Java Platform Debugger Architecture) http://java.sun.com/j2se/1.3/docs/guide/jpda/jdi/index.html