JPF Project

ABSTRACT

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference format:
. 1997. JPF Project. In *Proceedings of ACM Woodstock conference, El Paso, Texas USA, July 1997 (WOODSTOCK'97),* 2 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

In practice, program analysis relies on the existing technique tools. Many of Java program tools which play an important role in program analysis are designed to run on Java virtual machines, e.g., Java PathFinder.

2 MIXED EXECUTION

The mixed execution is based on two key insights. First, Android applications are typically structured as collections of screens where various events, e.g., user interactions, trigger transitions from one screen to another. As a series of events occur, the applications change and update internal states, bringing what a user desires. The major part of an application is to handle the events triggered by a user or from other applications and devices. The code of these event handlers are of most interest in the testing and analysis of Android applications since the process of user data and the implementation of designed functions are completed here.

Second, Android applications are written in Java and able to be executed on Java virtual machines logically. In practice, an application often interacts with underlying Android system using a framework API provided by the Android platform. The invoked components and methods are system-dependent, even device-dependent and cannot be executed on Java virtual machines. In practice, this part of code is assumed to be well tested and less concerned in the analysis and testing of applications.

With the two insights, we propose the concept of mixed execution, under which the code of an android application is executed on different platforms. The code which often requires intensive analysis and testing, e.g., event handlers, is executed on Java virtual machines. The code which cannot be run on Java virtual machines, i.e., code invoking the Android framework, is executed on the Android platforms. This allows the existing Java analysis tools to be directly applied to Android applications, avoiding the effort of extending these tools for Android platforms.

In the mixed execution, an android program is considered as a set of statements, from which the Android-platform-dependent statements are distinguished. These statements do not directly invoke the components or libraries in the Android platform like on mobile devices. Instead, they remotely interact with an Android framework, on which the invoked components and libraries are executed and in the end the desired results are returned to the Java virtual machine. With the mixed execution, an Android application can be run on a Java virtual machine.

2.1 Overview

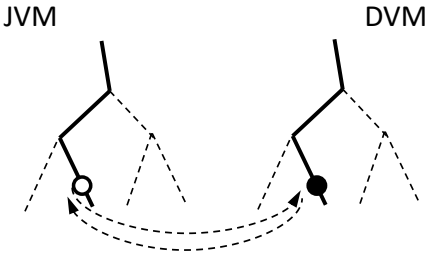

Figure 1: The overview of mixed execution

The mixed execution framework maintains two execution environments: JVM and DVM (see Figure 1). Given an Android application, the framework executes the application in the both environments simultaneously. Taking the same input and entry point, the application runs with the identical execution paths in the both environments. When the execution in JVM comes to a statement invoking Android libraries, the framework stops the execution and collects the information of the execution point. With this information, the framework controls the execution in DVM to reach the point where the execution in JVM stopped, making sure both executions have identical program states. Then the corresponding statement invoking the Android libraries is executed in DVM since all Android libraries are provided in this execution environment. The framework fetches the results of this statement from the DVM environment and returns them to the JVM environment. With the returned results, the execution in JVM is restarted and continues. For the Android-platform-dependent statements in the following execution, the framework will repeat the process above until the execution in JMV completes.

In the mixed execution, the Android-platform-dependent statements in JVM are handled as the input/output events of a program.

WOODSTOCK'97, July 1997, El Paso, Texas USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

Instead of invoking the Android libraries, these statements communicate with the DVM and get results computed by the invoked Android libraries in DVM. The crucial problem is to ensure the returned results are computed under the circumstance where the application has exact program states with the application in JVM. In the following, we present the approach to solve the problem.

2.2 Lock-step Synchronization

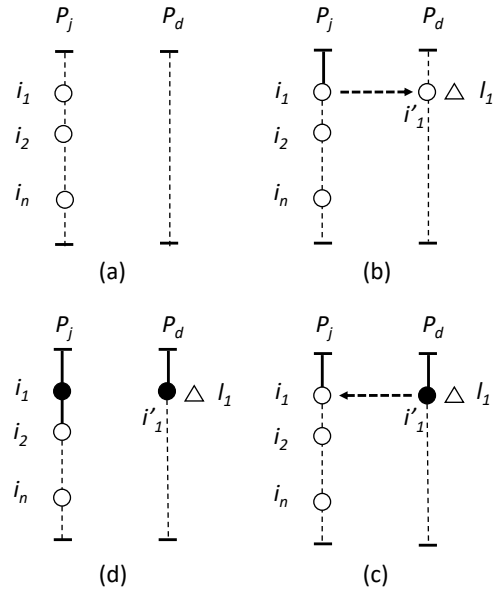

Figure 2: Lock-step synchronization execution between JVM and DVM

An Android application often intensively interacts with Android libraries. The invocation of Android libraries can be distributed in the whole program. When the application is executed in JVM and DVM environments, it requests to communicate with each other at each invocation of Android libraries. In order to synchronously execute the application in JVM and DVM, we propose the lock-step mechanism.

Given an Android application, a dummy main method is built in JVM environment, which is able to simulate the activities in launching the application in DVM environment. The application is provided with the same input in both environments, making both executions follow the identical paths.

Figure 2 shows the work flow of the lock-step synchronization mechanism. Given an application with the specific input, we assume its execution path is fixed. The path is indicated by P_j in JVM environment and P_d in DVM environment, as shown in Figure 2a. On the path, there exist n statements invoking Android libraries, which are represented by i_1, i_2...i_n. Now we demonstrate the execution in the lock-step manner.

At the initial stage, the application in JVM should be started first. The application in DVM waits to be started. When the execution on JVM comes to the first statement invoking Android libraries i_1, this part of the execution is considered as one step, indicated by s_1. In DVM environment, the application should run step s_1 and complete the invocation of the Android libraries, offering the results to the execution in JVM.

The statement i_1 is considered as an I/O event. JVM collects the location information of i_1 and sends it to DVM, and waits for results (see Figure 2b). With the location information of i_1, the mixed execution framework identifies the statement $i_1^{'}$ in DVM corresponding to i_1, at which a locker l_1 is set.

Next, the execution in DVM is started by the framework and stops at statement $i_1^{'}$ due to the locker l_1. In this execution, the statement $i_1^{'}$ is executed (marked by a solid circle in Figure 2c). The results returned by statement $i_1^{'}$ is sent back to JVM. With the results, JVM completes the execution of statement i_1 and starts the next step. For the next step, the locker l_1 will be removed when conducting the execution from statement $i_1^{'}$ to $i_2^{'}$. The process will be repeated until the application completes. Note that for the case that a statement invoking Android libraries is located in a loop, the process above still works because the locker l_m set in step s_m is removed in step s_{m+1}.

2.3 Events Transition From Android to JVM

2.4 Android Objects Migration

REFERENCES