Computer Languages, Systems & Structures 44 (2015) 283–298

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

A lightweight approach to component-level exception mechanism for robust android apps

Kwanghoon Choi[a,1], Byeong-Mo Chang[b,*,2]

[a] Computer & Telecommunication Engineering Division, Yonsei University, Wonju, Gangwon-Do 26493, Republic of Korea
[b] Department of Computer Science, Sookmyung Women's University, Seoul 04310, Republic of Korea

ARTICLE INFO

Article history:
Received 1 June 2015
Received in revised form
26 August 2015
Accepted 26 August 2015
Available online 10 September 2015

Keywords:
Android
Java
Exception
Component
Semantics

ABSTRACT

Recent researches have reported that Android programs are vulnerable to unexpected exceptions. One reason is that the current design of Android platform solely depends on Java exception mechanism, which is unaware of the component-based structure of Android programs. This paper proposes a component-level exception mechanism for programmers to build robust Android programs with. With the mechanism, they can define an intra-component handler for each component to recover from exceptions, and they can propagate uncaught exceptions to caller component along the reverse of component activation flow. Theoretically, we have formalized an Android semantics with exceptions to prove the robustness property of the mechanism. In practice, we have implemented the mechanism with a domain-specific library that extends existing Android components. This lightweight approach does not demand the change of the Android platform. In our experiment with Android benchmark programs, the library is found to catch a number of runtime exceptions that would otherwise get the programs terminated abnormally. We also measure the overhead of using the library to show that it is very small. Our proposal is a new mechanism for defending Android programs from unexpected exceptions.

1. Introduction

Exception handling in Java is an important feature to improve the robustness of Java programs. For example, Fig. 1 shows a simplified Java program that may throw one of two exceptions, `NoSuchOperator` and `ArithmeticException` (*divide by zero*), when users try to do a calculation with an unsupported operator or with zero as a divisor. Once the *calc* method throws such an exception, it is propagated along the call stack to a caller, the *main* method, where it is handled by the catch block.

Many Android programs are written in Java with Android APIs, presumably using exception handling. Android is Google's open-source platform for mobile devices, and it provides the APIs (Application Programming Interfaces) necessary to develop applications for the platform in Java (http://developer.android.com). An Android program consists of *components*

[*] Corresponding author.
E-mail addresses: kwanghoon.choi@yonsei.ac.kr (K. Choi), chang@sookmyung.ac.kr (B.-M. Chang).

[1] This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF-2014R1A1A2053446).
[2] This research was supported by the Sookmyung Women's University (Research Grant 1-1403-0212).

http://dx.doi.org/10.1016/j.cl.2015.08.010
1477-8424/© 2015 Elsevier Ltd. All rights reserved.

```
class Calculator {
 void main() {
    int a, b, r;  char op;
    // read a, op, and b
    try {
       r = calc(a, op, b);
       // display the result
    }
    catch(NoSuchOperator e) {
       // Not support the operator
    }
 }
```

```
int calc(int a,char op,int b) {
  if(op=='+' ) return a+b;
  else if(op=='-' ) return a-b;
  else if(op=='*' ) return a*b;
  else if(op=='/' ) return a/b;
  else
    throw new NoSuchOperator();
 }
}
```

Fig. 1. A Java program using exceptions.

such as activities, services, broadcast receivers and content providers. Android components communicate with another by sending messages called *Intents*. For example, an activity can start other activities by sending Intents to Android platform, which invokes methods of callee activities.

How vulnerable Android components are due to Intents have been reported by experiments in [1–4]. With Intent fuzzing, they generated random and semi-valid intents to test how components react to these exceptional conditions, focusing on uncaught exceptions that result in the crashes. One experiment by [2] measured the number of failed components for various types of components, reporting that 29(8.7%) out of total 332 activities crash with generated semi-valid intents. The distribution of exception types are also measured to understand how components fail due to uncaught exceptions, showing that `NullPointerException` makes up the largest share of all the exceptions, and other exceptions like `ClassNotFoundException` and `IllegalArgumentException` are next significant ones.

We have also found by examining source code of programs that Android programs can be very vulnerable to exceptions. We examined 9 programs and found that 41 activities (51%) out of total 80 activities have no exception handlers like try-catch, as will be shown in our experiment later. Activities without exception handlers cannot handle any thrown exceptions, and so result in the crashes when any exceptions are thrown.

From these observations, we can be sure that it is necessary for developing more robust Android programs to handle uncaught exceptions from components. Currently, programmers only resort to the conventional Java exceptions: the try-catch construct to defend statements and the thread-level uncaught exception handler interface (`Thread.UncaughtExceptionHandler`) to catch exceptions escaping from a thread. They are still crucial for Android programs, but they only address too fine-grained level in statements or too coarse-grained level in a thread. They are not immediately useful for addressing defending Android components.

Our design of *component-level exception mechanism* naturally follows that of the conventional Java exceptions of separating error-handling code from "regular" code and of propagating exceptions up the call stack. First of all, our mechanism is designed for each Android component to have a designated "catch" facility to defend itself from any (unexpected) uncaught exceptions thrown by the "regular" code of the component. This feature will allow programmers to focus more on the main flow of components, never missing any exceptions attempting to escape the components. Second, our mechanism is designed to support the propagation of exceptions following up a component activation stack. This facility will make components more resilient even by catching exceptions propagated from other components. Particularly, many components in Android programs have a relationship on "*who activates whom*", which is very similar to a caller–callee relationship in method invocation. Also, Android platform already has an activity stack internally, which is the same as the call stack of method invocation, to maintain the who-activates-whom relationship.

In this paper, we propose a mechanism for component-level exception handling and propagation in Android programs, which can be used to make them more robust by defending themselves from unexpected events. We take a lightweight approach by providing new component APIs (e.g., ExceptionActivity class), which extends the existing Android components (e.g., Activity class) with the component-level exception mechanism. No Android platform needs to be modified to use our approach. Programmers can utilize component-level exception handling and propagation by writing components with the new extended APIs. This use of the exception mechanism preserves the structure of classes and methods in original programs. Our approach is also flexible in that programmers can take full control of deciding which components handle what exceptions and how they are recovered.

Following an overview of Android programs and our motivation in Section 2, we present our idea of the Android component-level exception mechanism in Section 3. We give a theoretical account on the mechanism by an Android semantics with exceptions to prove the robustness of the mechanism in Section 4. We also perform experiments in practice to show that Android programs can be more robust with the new API in Section 5. We count how many exceptions are caught with the new API. We also measure the marginal cost of the mechanism by changed lines of code, increased binary size, and startup time due to the adoption of the mechanism. Finally, after discussing related work in Section 6, we conclude in Section 7.