Information Processing Letters 114 (2014) 620–627

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A type and effect system for activation flow of components in Android programs

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

[a] Yonsei University, Wonju, Republic of Korea
[b] Sookmyung Women's University, Seoul, Republic of Korea

ARTICLE INFO

Article history:
Received 1 August 2013
Received in revised form 23 March 2014
Accepted 22 May 2014
Available online 27 May 2014
Communicated by M. Yamashita

Keywords:
Android
Java
Program analysis
Control flow
Formal semantics

ABSTRACT

This paper proposes a type and effect system for analyzing activation flow between components through intents in Android programs. The activation flow information is necessary for all Android analyses such as a secure information flow analysis for Android programs. We first design a formal semantics for a core of featherweight Android/Java, which can address interaction between components through intents. Based on the formal semantics, we design a type and effect system for analyzing activation flow between components and demonstrate the soundness of the system.

1. Introduction

Android is Google's new open-source platform for mobile devices, and Android SDK (Software Development Kit) provides the tools and APIs (Application Programming Interfaces) necessary to develop applications for the platform in Java [1]. An Android application consists of components such as activities, services, broadcast receivers and content providers. In Android applications, components are activated through intents. An intent is an abstract description of a target component and an action to be performed. Its most significant use is in the activation of other activities. It can also be used to send system information to any interested broadcast receiver components and to communicate with a background service.

Many static analyses of Android programs [2–5] have adopted the existing Java analyses unaware of Android-specific features like components or intents, which are, however, essential for correctness of the Android program analyses. Such Android features make implicit the flow of execution, hiding it under Android platform. Therefore, the plain Java analyses cannot figure out all sound properties from Java programs running on Android platform. To address this problem, some Android analysis [5] attempted to introduce "wrapper"s modeling the Android features. However, one has never formalized the soundness of the existing Java analyses with such an Android extension.

The main contribution is to introduce a new featherweight Android/Java semantics, an analysis system for activation flow between components through intents, and its soundness proof with respect to the semantics. This activation flow analysis is important because the flow information is necessary for all Android analyses such as a secure information flow analysis. This formalization can be a basis for proving the soundness of the existing Android analyses.

* 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, Science and Technology (NRF 2011-0009225).
[2] This research was supported by the Sookmyung Women's University (Research Grants 1-1403-0212).

http://dx.doi.org/10.1016/j.ipl.2014.05.011
0020-0190/

We present our activation flow analysis as a type and effect system [6], and the key idea is to regard as an effect each occurrence of component activation through an intent. We first design a formal semantics for a core of featherweight Android/Java, which can address interaction between components through intents (Section 3). We design a type and effect system for analyzing activation flow between components (Section 4). Our system extends a Java type-based points-to analysis [7] with Android features, which demands us to introduce a simple string analysis [8] and the notion of effects [6]. The system records in the effects the name of Android components to activate, which the string analysis extracts from intents. We demonstrate the soundness of the system based on the formal semantics (Section 5). We discuss related work and sketch an implementation of our system (Section 6).

2. Overview of Android/Java

An Android program is a Java program with APIs in Android platform. Using the APIs, one can build mobile device user interfaces to make a phone call, play a game, and so on. An Android program consists of components whose types are Activity, Service, Broadcast Receiver, or Content Provider. Activity is a foreground process equipped with windows such as buttons and text areas. Service is responsible for background jobs, and so it has no user interface. Broadcast Receiver reacts to system-wide events such as notifying low power battery or SMS arrival. Content Provider is an abstraction of various kinds of storage including database management systems.

Components interact with each other by sending events called *Intent* in Android platform to form an application. The intent holds information about a target component to which it will be delivered, and it may hold data together. For example, a user interface screen provided by an activity changes to another by sending an intent to the Android platform, which will destroy the current UI screen and will launch a new screen displayed by a target activity specified in the intent.

The following table lists component types and some of the methods for activating components of each type [1].

Component type	Method for launching
Activity	startActivity(Intent)
Service	startService(Intent)
Broadcast Receiver	sendBroadcast(Intent)

Note that each occurrence of the above methods in an Android program is evidence for an interaction between a caller component and a callee component to be specified as a target in the intent parameter.

This paper focuses more on Activity than the other types because Activity is the most frequently used component type in Android programs. The proposed methodology in this paper will be equally applicable to the other types of components.

This paper uses an Android-based game program shown in Figs. 1 and 5 using the APIs shown in Fig. 2, whose details will be explained later.

```
class Score extends Activity {
    void onCreate() {
        this.addButton(1);
        // display the score screen
    }
    void onClick(int button) {
L1:     Intent i = new Intent();
L2:     i.setTarget("Main");
L3:     this.startActivity(i);
    }
}
```

Fig. 1. Score class in a game program.

Let us examine a Java class of the Android program in Fig. 1.

- Activity is a class that represents a screen in the Android platform, and Score extending Activity is also a class representing a screen.
- Once the Android platform creates a Score object, it invokes the *onCreate* method to add a button whose integer identifier is 1.
- Now a user can press the button 1, and then the *onClick* method is invoked to perform some action for the button.
- Intent is a class that represents an event to launch a new screen. It specifies the name of an activity class that represents the new screen.
- The *onClick* method sets "Main" as a target activity in the new intent object and requests launching by invoking *startActivity*.
- Android accepts the request and changes the current UI screen from Score to Main, which we call an *activation flow of components*.

The purpose of our type and effect system is to collect from an Android program all activation flows such as the above one from Score to Main by regarding Main as the *effect* of Score. The system needs to employ a form of string analysis [8] to infer classes (*Main*) from strings ("*Main*") stored in intents.

3. A semantic model for the Android platform

The syntax of a featherweight Android/Java is defined by extending the featherweight Java [9].

$$N ::= \text{class } C \text{ extends } C\ \{\bar{C}\ \bar{f};\ \bar{M}\}$$

$$M ::= C\ m(\bar{C}\ \bar{x})\ \{\ e\ \}$$

$$e ::= x \mid x.f \mid \text{new } C() \mid x.f = x \mid (C)x \mid x.m(\bar{x})$$

$$\mid \text{if } e \text{ then } e \text{ else } e \mid Cx = e;\ e \mid prim(\bar{x})$$

A list of class declarations $\bar{N}$ denotes an Android program. A block expression $C\ x = e; e'$ declares a local binding of a variable x to the value of e for later uses in e'. It is also used for sequencing $e;\ e'$ by assuming omission of a dummy variable $C\ x$. The conditional expression may be written as ite $e\ e\ e$ for brevity. We write a string object as a "string literal." Also, x.m("...") means *String* $s =$ "..."; $x.m(s)$ in shorthand. A recursive method offers a form of loops.