Droidel: A General Approach to Android Framework Modeling

Sam Blackshear
Alexandra Gendreau
Bor-Yuh Evan Chang

University of Colorado Boulder
Implementing an Android app

How do the app and framework communicate?

Extending special classes such as Activity

Overriding known methods such as onCreate()
How does the app hook into the Android framework?

@Override void onCreate()

Events in the Android framework trigger callbacks on the app
Execution of an Android app

Framework invokes app-defined callbacks via reflection

Call to `onCreate` is not seen in program analysis since callbacks are called via reflection

Entry point

`ActivityThread.main`

`App`

Reflective call

`@Override void onCreate()`
Models summarize reflective calls

**Explicit Model:**
Replace all reflective calls with explicit, app specific calls

```java
void loginActivityHarness() {
    Activity a = new LoginActivity();
    ...
}
void androidMain() {
    ...
    loginActivityHarness();
    ...
}
```
The trouble with modeling

The Android framework is **big**

The Android framework is **complex**

Client Specific Models

Requires careful modeling of execution context
Android Framework

Models are client specific and thus only summarize reflective calls relevant to a particular analysis making it difficult to reuse models.

Summarizes the behaviors of interest

These behaviors are abstracted away in the model.
Framework is Complex

To be sound, the harness must invoke `l.onCancel()` with respect to this `Activity`

Needs to over approximate behaviors of interest

Every harness model must soundly set up the execution context
Goal: a general purpose modeling approach

The problem is completely replacing the framework with a model

We present Droidel, a framework model for Android, built using these philosophies

A Different Approach

- Independent of the client analysis
- Avoids modeling the execution context

Model and augment the Android framework
Contribution of Droidel: model and augment

Explicated framework

App

App specific stubs

Android Framework
One time manual explication of the Android framework

public interface DroidelStubs {
    ...  
    Activity getActivity(String cls);
    ...
}

Replace with an explicit call to DroidelStubs

Activity a = (Activity) clazz.newInstance();

One time manual identification of the uses of reflection in the Android framework and replace those calls with explicit calls to DroidelStubs
Contribution of Droidel: model and augment

Explicated framework

App

App specific stubs

Android Framework
Contribution of
Droidel: model and augment

Explicated framework

App specific stubs

Android Framework
Automatic app specific stub generation

public interface DroidelStubs {
    ...
    Activity getActivity(String cls);
    ...
}

getter method for Activities

Droidel generates an implementation of DroidelStubs for each app

Dispatches based on the name and calls the zero argument constructor based as per the instructions in the documentation for newInstance

AppStubs implements DroidelStubs {
    Activity getActivity(String cls) {
        if (cls == "Activity A") {
            return new ActivityA();
        } else if (cls == "Activity B") {
            return new ActivityB();
        } else { return new Activity(); }  
    }
}
Droidel does not model the execution context. By explicating reflection, `AndroidThread.main` can be the entry point for analysis.
Empirical Evaluation
Experimental methodology

“The fundamental law of bug finding is No Check = No Bug. If the tool can’t check a method, then it won’t find bugs in it.”

Evaluate the percentage of concretely reachable methods in the call graph.

Experimental setup

1. Manual exploration of a set of 7 android apps

2. Compute the number of concretely reachable methods

3. Compare the number of concretely reachable methods in the call graphs generated using Droidel and FlowDroid (a taint analysis framework model).
## Experimental results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dynamic Exploration of App Methods</th>
<th>Reachable methods (FlowDroid)</th>
<th>Reachable methods (Droidel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>Total</td>
<td>Reached</td>
<td>Reached</td>
</tr>
<tr>
<td></td>
<td>Visited</td>
<td>% Visited</td>
<td>% Missed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>drupaleditor</td>
<td>325</td>
<td>90</td>
<td>28</td>
</tr>
<tr>
<td>spycamera</td>
<td>254</td>
<td>156</td>
<td>61</td>
</tr>
<tr>
<td>npr</td>
<td>76</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>352</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>textsecure</td>
<td>925</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>wordpress</td>
<td>5796</td>
<td>2042</td>
<td>35</td>
</tr>
<tr>
<td>k9</td>
<td>5357</td>
<td>1905</td>
<td>36</td>
</tr>
<tr>
<td>Summary</td>
<td>17467</td>
<td>6173</td>
<td>38</td>
</tr>
</tbody>
</table>

FlowDroid misses more concretely reachable methods than DROIDEL.
Droidel produces code that can be read by any Java analyzer
How can you help us?

Please use your Android expertise to contribute to Droidel

(https://github.com/cuplyv/droidel)
EXTRA SLIDES
Current Limitations of DROIDEL

- Not all uses of reflection have been explicated yet (i.e. Reflective allocation of Preferences objects)
- No generated stubs for summarizing native methods in Android

Not a problem with our approach but a limitation of the current implementation
Issues with this approach

- Client analysis specific
- Targeting another client analysis causes soundness issues
- Extensive manual effort
DROIDEL Outputs

Java Bytecode

Java Source Code

ActivityThread.main

Entry Point for Analysis

Java Program Analysis

DROIDEL Output

DROIDEL Output
• Manually explicate each version of the Android Framework

The current model and replace approach suffers this problem as well