ICCBot: Fragment-Aware and Context-Sensitive ICC Resolution for Android Applications

Jiwei Yan
Institute of Software, CAS, China
University of Chinese Academy of Sciences

Shixin Zhang
Beijing Jiaotong University, China

Yepang Liu
Southern University of Science and Technology, China

Jun Yan
Institute of Software, CAS, China
University of Chinese Academy of Sciences

Jian Zhang
Institute of Software, CAS, China
University of Chinese Academy of Sciences

ABSTRACT

For GUI programs, like Android apps, the program functionalities are encapsulated in a set of basic components, each of which represents an independent function module. When interacting with an app, users are actually operating a set of components. The transitions among components, which are supported by the Android Inter-component communication (ICC) mechanism, reflect the skeleton of an app. To effectively resolve the source and destination of an ICC, both the correct entry-point identification and the precise data value tracking of ICC fields are required. However, with the wide usage of Android Fragment, the entry-point analysis usually terminates at an inner fragment but not its host component. Also, the simply tracked ICC field values may become inaccurate when data is transferred among multiple methods. In this paper, we design an ICC resolution tool *ICCBot*, which resolves the component transitions that are connected by fragments to help the entry-point identification. Besides, it performs context-sensitive inter-procedural analysis to precisely obtain the ICC carried data values. Compared with the state-of-the-art tools, *ICCBot* achieves both a higher success rate and accuracy. *ICCBot* is open-sourced at **https://github.com/hanada31/ICCBot**. A video demonstration of it is at **https://www.youtube.com/watch?v=7zcoMBtGiLY**.

KEYWORDS

Android, ICC Resolution, Fragment, Component Transition Graph

ACM Reference Format:
Jiwei Yan, Shixin Zhang, Yepang Liu, Jun Yan, and Jian Zhang . 2021. ICCBot: Fragment-Aware and Context-Sensitive ICC Resolution for Android Applications. In *Proceedings of The 44th International Conference on Software Engineering (ICSE 2022).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

ICC is a framework-related mechanism, which plays an important role in building implicit control flow as well as transferring data among Android app components. The comprehensive extraction of ICC links can help to understand GUI structure and detect inter-component or inter-app risks. Many existing works [6, 10, 11] concentrate on the resolution of ICC links or the construction of component transition graphs (CTGs). However, most of them only consider the scenarios that one component directly launches another, but do not take the fragment [9] behaviors into consideration. According to existing research [8], nearly 91% of 217 top downloaded apps from Google Play use fragments. Also, some approaches adopt the context-insensitive strategy to improve efficiency, which leads to imprecise tracking of data transferring among method calls.

In this paper, we aim to resolve ICC links with consideration of both precision and efficiency. We implement a fragment-aware and context-sensitive ICC resolution tool, called *ICCBot*. For precision improvement, it first models the behaviors of fragments and extracts the fragment-loading graph, then connects the components linked through fragments. During analysis, it constructs summaries for each method according to the bottom-up order in the call graph and analyzes the relationship between the callee's summary and values of calling context. When analyzing the caller of a method, only specific context values will be tracked for summary updating. If there is no summary-related context value, the method summary of callee will be reused directly. According to the evaluation results, ICCBot achieves a 95% precision on two popular benchmarks and greatly outperforms the state-of-the-art tools on a benchmark containing fragment and context related ICCs. On large-scale real world apps, it resolved 28,584 ICC edges and 11,871 fragment-loading edges on 2,000 real-world apps, in which the fragment and context-sensitive strategies bring over 8% and 25% ICC links.

2 BACKGROUND OF ICC RESOLUTION

In the Android system, the basic function of ICC mechanism is achieved by constructing and sending `Intent` objects in apps. Each Intent object carries a group of data items that will be delivered to the Android system, parts of which are used by the system to determine the target component. Thus, the destination of an ICC transition can be obtained by precise ICC field extraction. Normally, developers send an ICC message in the lifecycle or user-callback methods of a basic component, like an `Activity`. Thus, the source

```
1  public class MainActivity extends ActionBarActivity {...
2      protected void onCreate(Bundle savedInstanceState) {
3          Button btn = (Button)findViewById(R.id.button);
4          btn.setOnClickListener(new OnClickListener(){
5              //Callback Entry of Component
6              public void onClick(View v){
7                  getSupportFragmentManager().beginTransaction()
8                  .replace(R.id.fragment, new OneFragment())
9                  .commit();}    //Fragment Loading
10 });}}
11 public class OneFragment extends Fragment {...
12     public void onAttach(Activity activity) {
13         //Callback Entry of Fragment
14         if(...){ Utils.sendICC("action.first");}
15         else{ Utils.sendICC("action.second");}
16 }}
17 public class Utils{
18     public static void sendICC(String mAction){
19         //Call Context Related ICC
20         Intent i = new Intent(mAction);
21         addCategory(i);
22         getActivity().startActivity(i); }
23     public static void addCategory(Intent i){
24         i.addCategory("category");}
25 }
```

Figure 1: Motivating Example

of an ICC transition can be obtained by analyzing the entry-point method of Intent sending. Nowadays, the `Fragment` [9] component, which is hosted by an activity or another fragment for reusable UI, is becoming more and more popular. It has its own lifecycle callback methods. Many developers choose to load fragments first and then send ICC messages in their callback methods. Therefore, to identify the actual source component of an ICC, the control flows between activities and the inner fragments should be built first, based on which we could extract the activity-level hosts of the fragments that actually trigger the Intent sending behavior.

Fig. 1 displays a motivating example with fragment usage and context-related ICC data assignments. In lines 7-9, the fragment `OneFragment` is created and loaded in component `FirstActivity`. For the Intent object i in line 20, without fragment modeling, the entry-point tracking analysis will terminate at the fragment lifecycle method `onAttach()` but miss the actual entry in its host activity. In lines 14-15, when the fragment `OneFragment` is attached, the method `sendICC()` will be invoked with a specific action value. This context value is used to construct an `Intent` object, while different values passed to the callee will lead to different ICC targets. In line 21, the Intent object is passed into method `addCategory()` for category adding. Considering both the Intent object itself and values of data fields can be passed among methods, context-sensitive inter-procedural analysis should be performed. Finally, in line 22, the updated Intent is sent to Android system by API `startActivity()`. For this example, ICCBot can detect one fragment-loading edge and two component-transition edges, while the related tools IC3 [10], IC3DIALDroid [7], and Gator(ATG client) [2] report no ICC. If the fragment is removed, these existing tools all generate FP (non-existing) ICCs when method `sendICC()` is called in multiple components, due to their context-insensitive analysis.

3 ICCBOT: FRAGMENT-AWARE AND CONTEXT-SENSITIVE ICC RESOLUTION

Fig. 2 displays the overview of our ICC resolution approach, which contains four modules: 1) callback-related control-flow analysis, 2)

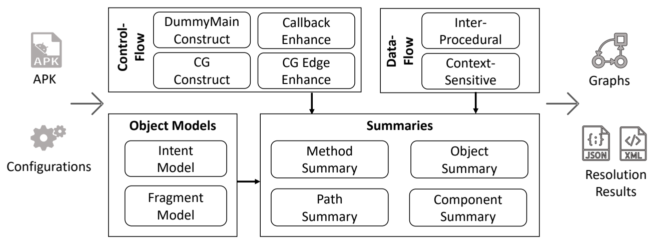


Figure 2: Framework of ICCBot (CTGClient)

fields-related data-flow analysis, 3) Intent/Fragment object modeling, and 4) summary construction. For the control-flow analysis, we first use FlowDroid [5] to get the initial call graph (CG) and callback entries. Based on it, we enhance the CG by adding asynchronous and polymorphic call-edges and extend the callback set by analyzing the user-customized callback listeners. For data-flow analysis, we adopt the intra-procedural reaching definition analysis to get the target variable related *du-chains* in each method and perform an inter-procedural analysis to track the data values transferred among methods. For the object modeling module, both the Fragment and Intent objects are concerned. The fragment model describes the fragment creating, adding and replacing behaviors, and the Intent model describes the Intent creating, packaging and sending behaviors. We model the behaviors of the related APIs according to their semantics and identify the objects whose behavior successfully triggers fragment loading and Intent sending. Finally, the above analysis can help to generate context-sensitive summaries that are used for ICC resolution. In the following, we mainly introduce the extraction of fragment-aware transitions and the construction of context-sensitive summaries.

3.1 Fragment-Aware Transition Extraction

There are two ways to start a fragment component in Android apps. For the static loading, the activity component can declare the fragment inside its layout file. Thus, by analyzing the layout declarations and the view inflating statements, we can get corresponding $\langle act, frag\rangle$ pairs. Besides, the activity component can add fragments into an existing `ViewGroup`. For example, we can obtain the `FragmentManager` and submit a transaction to it, which includes a set of fragment-related operations, like add, replace, etc. The actual fragment loading operations are executed by the Android framework, which is invisible in CG. To extract these edges, we model the fragment operating APIs and then infer the fragment behaviors by tracking the transaction sequences committed into the `FragmentManager`. A valid fragment transaction sequence starts from the creation of an `FragmentTransaction`, contains at least one fragment modification operation, and ends with the transaction commit operation. By further analyzing the modification targets, we can get either $\langle act, frag\rangle$ or $\langle frag, frag\rangle$ pairs. And the invalid operation sequences, e.g., not committed transactions, will be dropped. Finally, by analyzing the transitive relation among activities and fragments, we point out the activity-level host set of each fragment in pair $\langle frag, S_{act}\rangle$. Based on these analysis, when the entry callback that belongs to a fragment is reached, ICCBot could quickly identify its actual activity-level host component.

3.2 Context-Sensitive Summary Analysis

To improve efficiency, we consider building a set of lightweight context-sensitive summaries that reduce unnecessary computation.

Method Summary. The analysis is in a bottom-up order according to an acyclic CG, in which the cycles are removed by preprocessing. For method invocation, the callee is analyzed earlier than its caller. The summary of a method can be formally defined as a triple $\mathcal{MS} = \langle mtd, S_{os}, S_{ms} \rangle$, where

- mtd is the unique signature of current method.
- S_{os} is a set of object summaries. Each $os \in S_{os}$ is an object summary whose object $os.obj$ is created or received in mtd.
- S_{ms} is a set of method summaries. Each $ms \in S_{ms}$ denotes a callee method that is invoked by mtd. The inner object summaries in $ms.S_{os}$ are all context-irrelevant with its caller mtd.

Each method summary contains a set of object summaries and a set of callee method summaries. The object summary set S_{os} describes the created and received Intent or Fragment objects in the current method. For the callees whose object summaries have relationships with the invocation context, we efficiently build new object summaries into S_{os} according to callee's method summary and their relationships. And for other callees whose object summaries are context-irrelevant with method mtd, their method summaries will be directly added into S_{ms} to avoid re-analysis. For each method, the summaries are path-sensitively built on the Intent or Fragment-related code slices, and the maximum number of paths under analysis is limited by a threshold to avoid path exploration.

Object Summary. The object summary models the behaviors of an Intent or Fragment object by analyzing the operating statements of an object in one path. Both the Intent and Fragment-related statements are grouped into three categories, i.e., *create*, *update* and *send*. For a valid object summary, each category must contain at least one statement. We manipulate the object instance according to the semantic of each statement. For example, the statement `i = new Intent()` creates Intent object $\mathtt{I}_1$ and the statement `i.addAction(s)` updates the action field of $\mathtt{I}_1$. The valid summary of an Intent or Fragment object can be formally defined as a 4-tuple $\mathcal{OS} = \langle obj, mtd, L_{stmt}, M_{ctx} \rangle$, where

- obj is an object instance to be modeled, which contains a set of fields according to its concrete type.
- mtd is the signature of method where obj is created or received.
- L_{stmt} is a list of Intent/Fragment related statements, where $L_{stmt} = L_c \cup L_u \cup L_s$. L_c is a list of object creating or receiving statements that creates object obj, $|L_c| = 1$; L_u is a list of object updating statements that updates the fields of object obj, $|L_u| \geq 1$; L_s is a list of object sending statements, which delivers object obj to Android framework, $|L_s| = 1$. When a new statement s is observed and added into L_{stmt}, object obj will be modified according to the extracted target field and the filed value in s.
- M_{ctx} records the map from object fields to context locations, e.g., for the Intent summary created in method `sendICC()`, pair $\langle action, sendICC_1 \rangle$ means the value of action field is influenced by the first formal parameter of `sendICC()`. This pair will be transitively updated when its caller is analyzed and the first actual parameter of `sendICC()` is also context-related.

Component Summary. Based on object summaries, we can obtain the Fragment loading edges to enhance CG and extract the source/destination information to resolve ICC links. Then we analyze all the Fragment and Intent object summaries to further build component summaries, which can be formally defined as a tuple $CS = \langle source, S_{destination} \rangle$, where

- $source$ is the component that sends out the target object.
- $S_{destination}$ is a set of components to be loaded or launched.

4 USAGE

Multiple Clients. ICCBot provides a group of clients as follows. Users can customize their own clients based on them or just use them as separated functionalities.

- **CallGraphClient**. Output the method call graph of an app, including the enhanced asynchronous and polymorphic edges.
- **ManifestClient**. Extract and analyze the manifest file in app.
- **IROutputClient**. Run Soot and output the generated IR files.
- **FragmentClient**. Extract the fragment-loading relationships, including visualized graphs and the detailed summary files.
- **CTGClient**. The default client. Analyze the component transition relationships, including visualized graphs and the summary files of each Intent-sending and fragment-loading edge.
- **ICCSpecClient**. Output the inferred ICC sending and receiving specification by extracting the component declaration and analyzing the sent and received ICC messages.

Customized Configuration. ICCBot provides various configuration items to customize the analysis process. For the default *CTGClient*, the normal configuration items such as *maxPathNumber* (for intra-procedural path-sensitive analysis) and *maxAnalyzeTime* could be adjusted for the scalability. Meanwhile, the analysis can be performed while including or excluding specific Android characteristics, like fragment loading, context-sensitive analyzing, CG enhancing, String API analysis, etc.

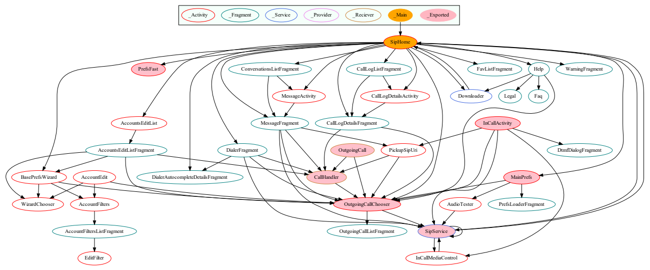
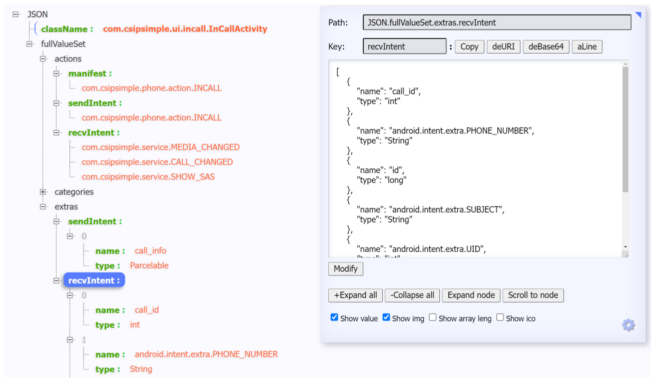


Figure 3: The Output of CTGClient


```
JSON
  className : com.csipsimple.ui.incall.InCallActivity
  fullValueSet
    actions
      manifest :
        com.csipsimple.phone.action.INCALL
      sendIntent :
        com.csipsimple.phone.action.INCALL
      recvIntent :
        com.csipsimple.service.MEDIA_CHANGED
        com.csipsimple.service.CALL_CHANGED
        com.csipsimple.service.SHOW_SAS
    categories
    extras
      sendIntent :
        0
          name : call_info
          type : Parcelable
      recvIntent :
        0
          name : call_id
          type : int
        1
          name : android.intent.extra.PHONE_NUMBER
          type : String
```

Path: JSON.fullValueSet.extras.recvIntent

Key: recvIntent

```
[
  {
    "name": "call_id",
    "type": "int"
  },
  {
    "name": "android.intent.extra.PHONE_NUMBER",
    "type": "String"
  },
  {
    "name": "id",
    "type": "long"
  },
  {
    "name": "android.intent.extra.SUBJECT",
    "type": "String"
  },
  {
    "name": "android.intent.extra.UID",
```

[✓] Show value [✓] Show img ☐ Show array leng ☐ Show ico

Figure 4: The Output of ICCSpecClient

Friendly Output. The outputs for both the *CTGClient* and *FragmentClient* are visualized with dot graph. Fig. 3 gives parts of the CTG output of app *CSipSimple*, in which both the fragments-loading edges and the component transitions are displayed. And Fig. 4 gives a snippet of report generated by the *ICCSpecClient*, which is in JSON format and displayed with the help of a JSON viewer here.

5 EVALUATION

In this section, we evaluate ICCBot on two benchmarks. One is formed by hand-made apps, and the other contains large-scale real-world apps. After filtering out the tools that are publicly unavailable or failed to be executed on our circumstance, we find three state-of-the-art tools *IC3*, *IC3-DIALDroid* and *Gator (ATG client)* can be used for results comparison with *ICCBot*.

First, we collect two well-known benchmarks, the *DroidBench* [1] (ICC related) and the *ICC-Bench* [3], which contain 38 ICC links in total. In addition, we design a compact self-made benchmark *ICCBotBench* [4], which contains two fragment-related ICCs that cover the basic static and dynamic fragment loading usages, seven context-related ICCs that try to pass the target objects or related values among methods, and two ICCs that involve both the fragment and context characteristics. The evaluation results are given in Table 1. The upper part presents the behaviors of tools on different benchmarks, in which the second column is the number of ground truth ICC links in each benchmark. The following columns are the number of reported ICCs and FPs (after -) of each tool, i.e., the subtraction results are TPs. Similarly, the lower part presents the behaviors of each tool about the summarized characteristics, in which the second column is the number of ground truth ICCs around each characteristic. According to the results, the state-of-the-art tools suffer from both the FN and FP, where ICCBot can correctly resolve ICC in almost all cases. By further investigation, we find that the dismissing of characteristics like fragment and incomplete callback handling are the key reasons leading to FN. The context-insensitive analysis is the main reason for FPs in three tools, especially for *IC3-DIALDroid* that context-related FPs are caused by the side effects of entry analysis optimization.

To further validate the effectiveness of tools on real-world apps, we perform evaluations on 2,000 apps from F-droid. For all the tools, we set 30 minutes as the analysis time threshold for each app. Table 2 shows the number of apps that succeeded and failed (crash or timeout) to be analyzed, the average analyzing time for all the successfully analyzed apps, the number of the detected ICC links, and the number of fragment-loading edges. According to the results, ICCBot achieves a higher success rate than other tools with acceptable efficiency. Considering the existence of FP and FN in the state-of-the-art tools, the number of ICC links is only provided for reference only. As ICCBot is configurable, we compare its default configuration with the ones that exclude fragment and context-sensitive data analysis. Our context-insensitive strategy omits to track the data transferring among methods instead of merging data from all the contexts, so that will not bring FP. According to the results, the fragment analysis added 11,871 edges in the CTG for fragment displaying. And the two strategies contribute with 2284 (8.68%), and 5769 (25.29%) ICCs, respectively, which greatly improves the completeness of the original CTG.

Table 1: Evaluation on Hand-made Benchmarks

Benchmark	**#GT**	***IC3***	***IC3$_{dial}$***	***Gator***	***ICCBot***
DroidBench	12	9	9	7-1	10
ICC-Bench	26	10	26	4	26
ICCBotBench	11	10-3	34-25	10-3	11
Sum	49	29-3	69-25	21-4	47
Characteristic	**#GT**	***IC3***	***IC3$_{dial}$***	***Gator***	***ICCBot***
Fragment	4	0	9-6	0	4
Call Context	9	10-3	25-18	10-3	9
Callback Entry	14	0	14	0	14
Other	32	19	29	16-1	30
Sum	59	29-3	77-24	26-4	57

Table 2: Evaluation on Real-world Benchmarks

Tool	**#Succ**	**#Fail**	**Time$_{succ}$**	**#ICC**	**#Frag**
IC3	1,719	281	57s	10,860	-
IC3$_{dial}$	1,851	149	115s	8,601	-
Gator	1,882	118	20s	27,297	-
ICCBot	2,000	0	54s	28,584	40,455

6 RELATED WORK

For the task of ICC resolution, *Epicc* [11] and *IC3* [10] model the Intent-related APIs and resolve the ICC field values by data-flow analysis. Tool *IC3-DIALDroid* [7] implements incremental callback analysis and extends the original *IC3*. *Gator* [2] provides an analysis client to generate the activity transition graph of an app. However, these works dismiss the components connected by fragments or adopt context-insensitive analysis, which will decrease the precision of generated CTG. Another tool *FragDroid* [8] presents the activity-fragment transition graph. However, their tool is not publicly available. Also, its transition analysis only tracks several object-creating APIs while not modeling the ICC behaviors completely.

7 CONCLUSION

Android ICC resolution is widely used in various scenarios. For precise resolution, it faces challenges of fragment analysis and data extraction. In this paper, we present a fragment-aware and context-sensitive ICC resolution tool *ICCBot*, which can efficiently extract CTG with higher accuracy and generate friendly outputs.

REFERENCES

[1] DroidBench. https://github.com/secure-software-engineering/DroidBench.
[2] Gator. http://web.cse.ohio-state.edu/presto/software/gator/.
[3] ICC-Bench. https://github.com/fgwei/ICC-Bench.
[4] ICCBot, ICCBotBench. https://github.com/hanada31/ICCBot, 2021.
[5] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. McDaniel. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In *PLDI 2014*, page 29, 2014.
[6] T. Azim and I. Neamtiu. Targeted and depth-first exploration for systematic testing of Android apps. In *OOPSLA 2013*, pages 641–660, 2013.
[7] A. Bosu, F. Liu, D. D. Yao, and G. Wang. Collusive data leak and more: Large-scale threat analysis of inter-app communications. In *AsiaCCS*, pages 71–85, 2017.
[8] J. Chen, G. Han, S. Guo, and W. Diao. Fragdroid: Automated user interface interaction with activity and fragment analysis in Android applications. In *DSN 2018*, pages 398–409, 2018.
[9] Fragment. Fragment. https://developer.android.com/guide/fragments, 2021.
[10] D. Octeau, D. Luchaup, M. Dering, S. Jha, and P. McDaniel. Composite Constant Propagation: Application to Android Inter-Component Communication Analysis. In *ICSE 2015*, pages 77–88, 2015.
[11] D. Octeau, P. D. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. L. Traon. Effective inter-component communication mapping in Android: An essential step towards holistic security analysis. In *USENIX Security Symposium, 2013*, pages 543–558, 2013.