Static Control-Flow Analysis of User-Driven Callbacks in Android Applications

ICSE’15

Shengqian Yang, Dacong Yan, Haowei Wu, Yan Wang, and Atanas Rountev Ohio State University

Presenter: Zheng Song

About the authors

- Shengqian Yang: PhD student since 2010.
- Dacong Yan, Phd 2009~2014, now at Google
- Haowei Wu, Phd student since 2013
- Yan Wang, …
- Atanas Rountev Ohio: h-index 30
    - 1995-2002 PhD from Rutgers University
    - OSU since then, now holds a professor position.

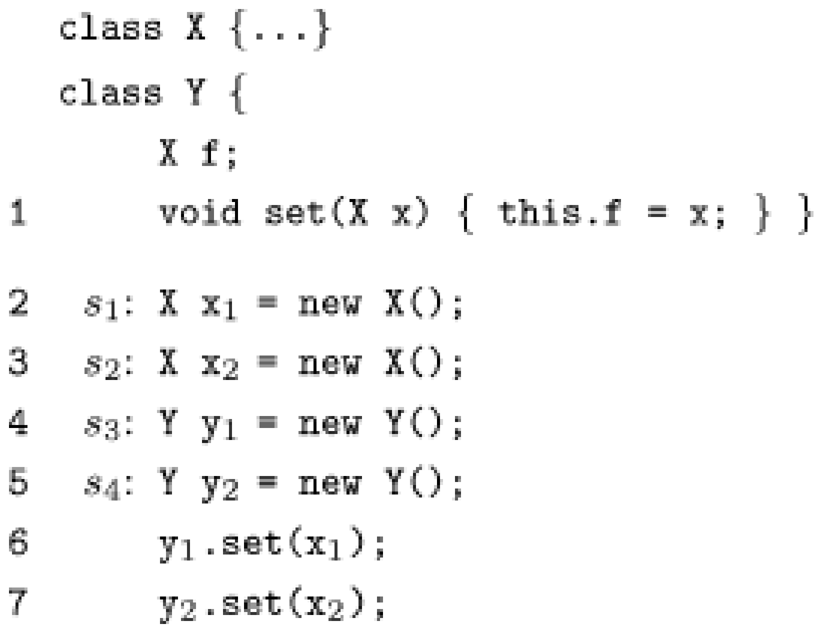
Program Analyses and Software Tools (PRESTO) Research Group

Key Contributions

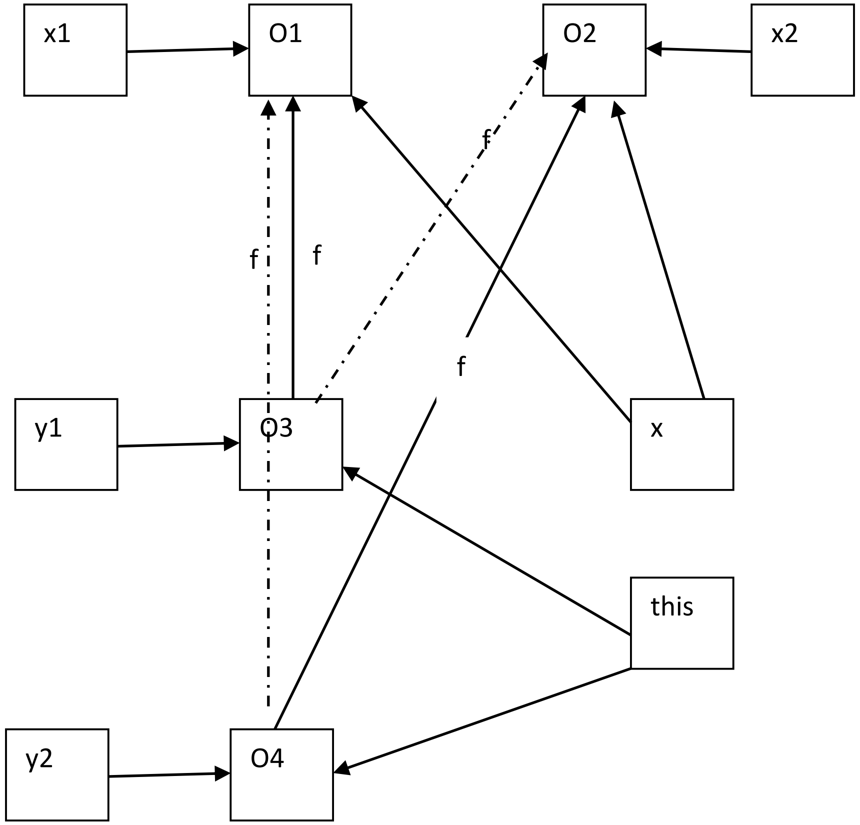
- 1. User-driven callbacks (lifecycle & event handler)
  - Traditional analyses cannot fit Android, framework-based and event-driven.
  - We consider user-event driven components and the related sequences of callbacks from the Android framework to the application code, [both for lifecycle callbacks and for event handler callbacks]
- 2. a program representation to capture such callback sequences.
  - using context sensitive static analysis of callback methods.

  Q: Context-sensitive??? (context-sensitive point-to analysis…)

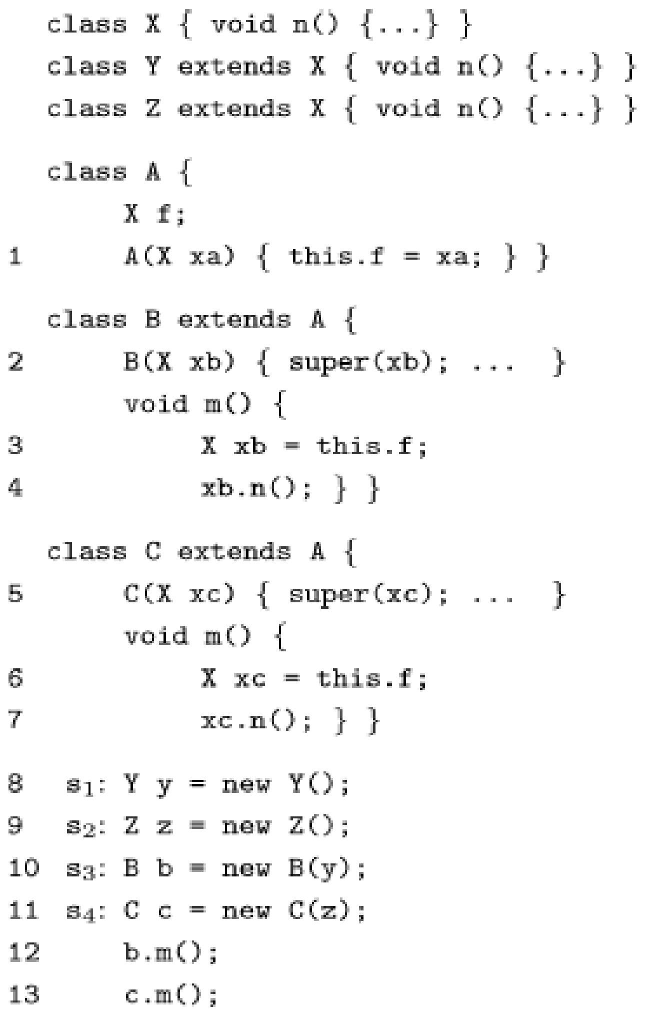
Class-sensitive point-to analysis: Encapsulation



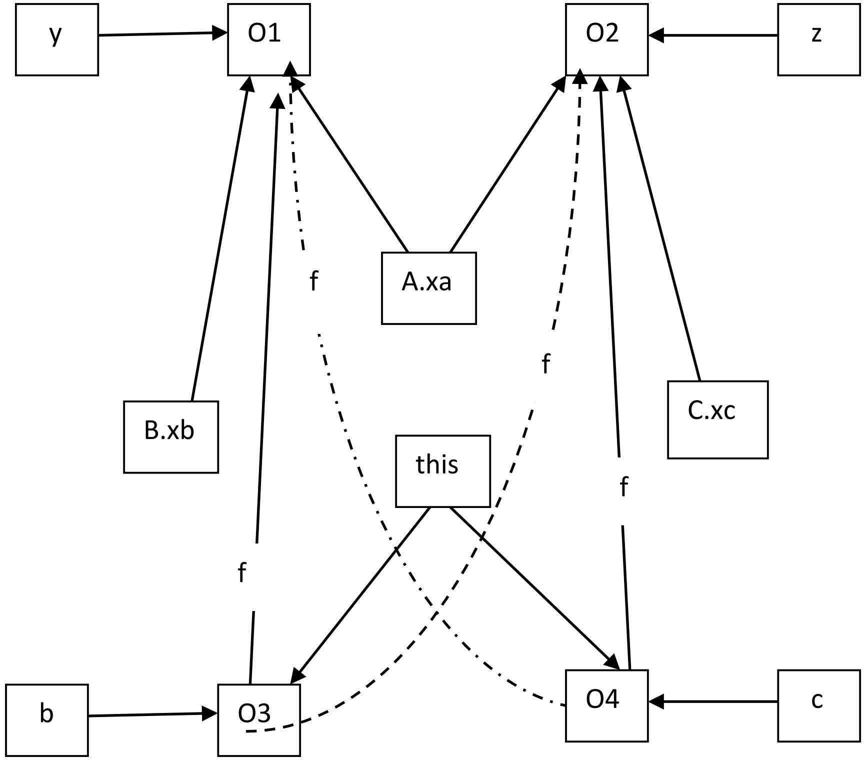
```
class X {...}
class Y {
     X f;
     void set(X x) { this.f = x; } }

 s1: X x1 = new X();
 s2: X x2 = new X();
 s3: Y y1 = new Y();
 s4: Y y2 = new Y();
     y1.set(x1);
     y2.set(x2);
```



Class-sensitive point-to analysis: Inheritance



```
     class X { void n() {...} }
     class Y extends X { void n() {...} }
     class Z extends X { void n() {...} }

     class A {
         X f;
1        A(X xa) { this.f = xa; } }

     class B extends A {
2        B(X xb) { super(xb); ...  }
         void m() {
3            X xb = this.f;
4            xb.n(); } }

     class C extends A {
5        C(X xc) { super(xc); ...  }
         void m() {
6            X xc = this.f;
7            xc.n(); } }

8     s1: Y y = new Y();
9     s2: Z z = new Z();
10    s3: B b = new B(y);
11    s4: C c = new C(z);
12       b.m();
13       c.m();
```



Outline

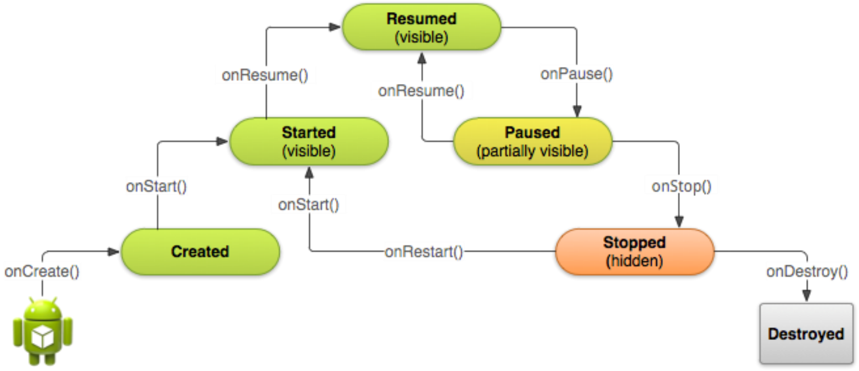
- 1. Introduction & definations
- 2. Example
- 3. Algorithm
- 4. Usage
- 5. Evaluation
- 6. Discussion

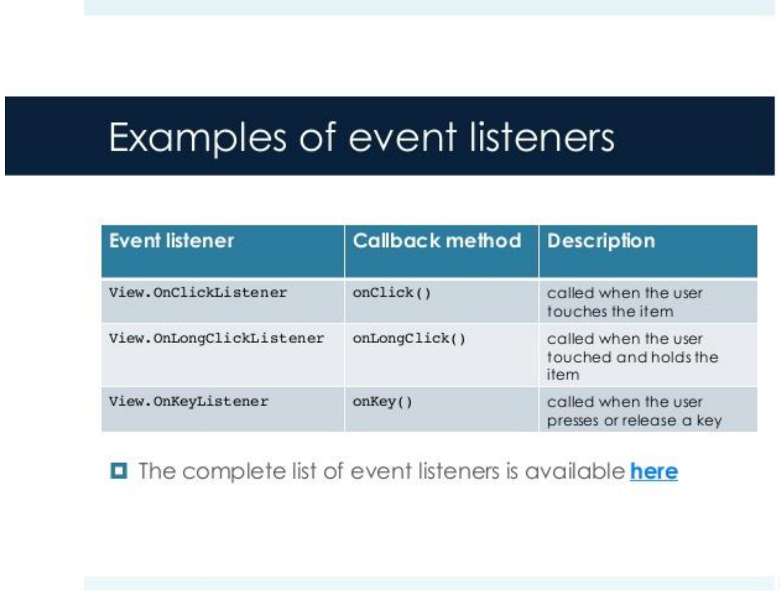
1. Introduction and Definations

- Android CallBacks: Don’t call us, we’ll call you☺

calls from the platform’s event processing code to the relevant callback methods defined in the application code.

Q: lifecycle callbacks; user event handler; other callbacks?

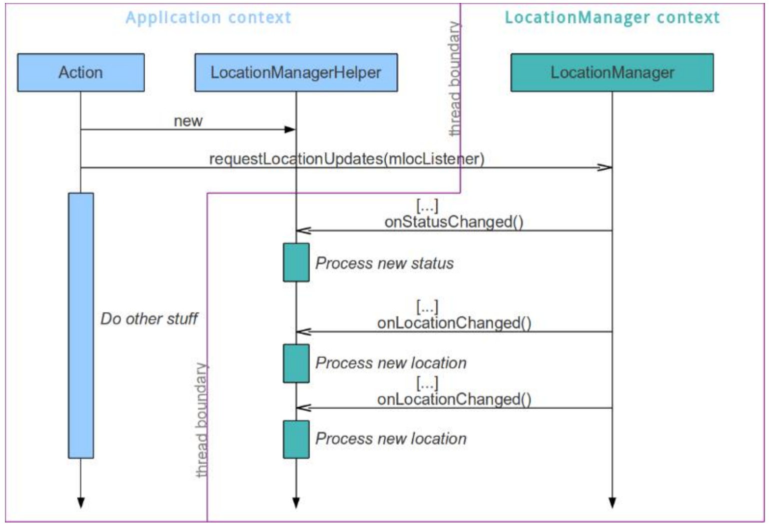




Examples of event listeners

Event listener	Callback method	Description
View.OnClickListener	onClick()	called when the user touches the item
View.OnLongClickListener	onLongClick()	called when the user touched and holds the item
View.OnKeyListener	onKey()	called when the user presses or release a key

- The complete list of event listeners is available **here**



1. Introduction:

- Procedure:
- 1. In essence, the control flow analysis problem can be reduced to modeling of the possible sequences of callbacks.
- 2. captures such callback sequences as *callback control-flow graph* (CCFG) [The analysis of each callback method (and the code transitively invoked by it) determines what other callbacks may be triggered next.]

- technical insight: a callback method must be analyzed *separately for different invocation contexts associated with it* =>context sensitivity

- Why is useful: the automated generation of static GUI models

1. Definition:

- 1. CFG, ICFG, CCFG
- 2. The CFG for a procedure *p* has a dedicated start node *sp* and a dedicated exit node *ep*. Each call is represented by two nodes: a call-site node *ci* and a return-site node *ri*. There is an interprocedural
-

  edge *ci*→*sp* from a call-site node to the start node of the called procedure *p*; there is also a corresponding edge *ep* → *ri*.

- Thus, the abstracted controlflow paths are always of the form *ci* → *smi, emi* → *ri, cj* →*smj , emj* → *rj, ck* → *smk, emk* → *rk, . . .* and will be represented simply as *mi mj mk . . .* where *mi* is the callbackmethod invoked by *c*
- set *L* of lifecycle methods for activities, dialogs, and menus, as well as set *H* of GUI event handler methods.

2. Motivating Example

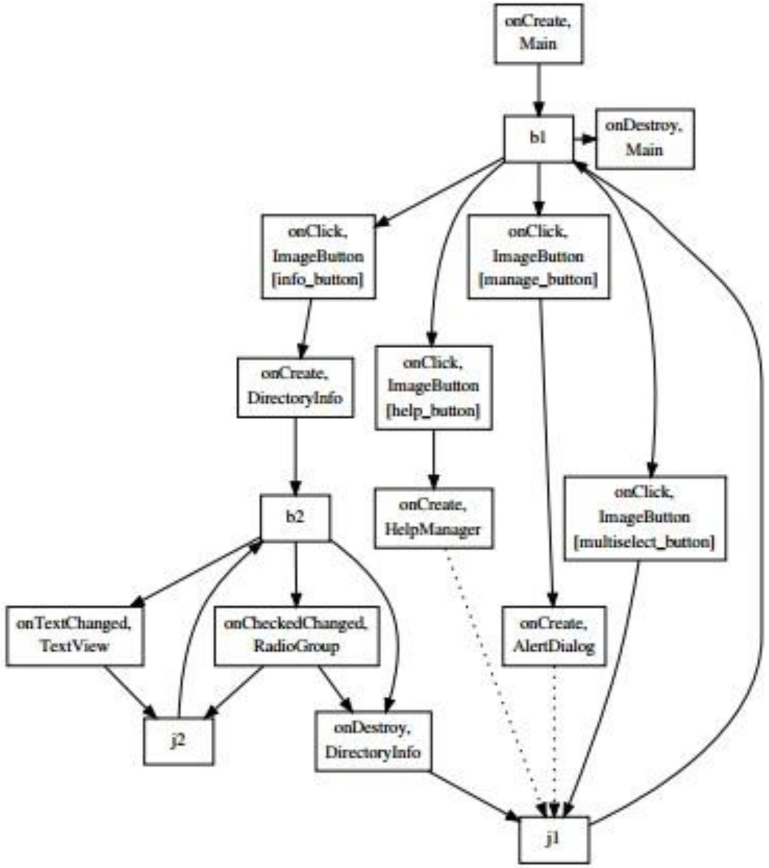
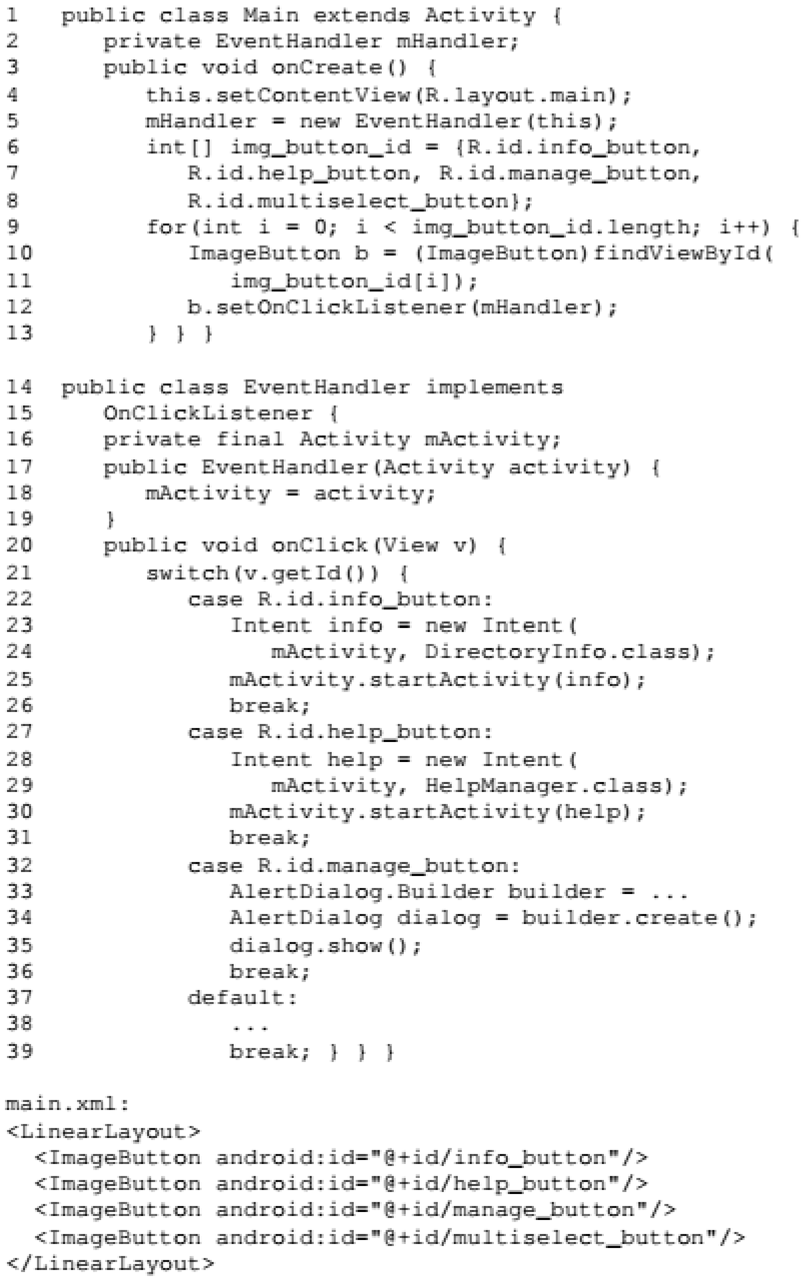

onCreate, Main
b1
onDestroy, Main
onClick, ImageButton [info_button]
onClick, ImageButton [manage_button]
onCreate, DirectoryInfo
onClick, ImageButton [help_button]
b2
onCreate, HelpManager
onClick, ImageButton [multiselect_button]
onTextChanged, TextView
onCheckedChanged, RadioGroup
onCreate, AlertDialog
j2
onDestroy, DirectoryInfo
j1

Fig. 2: Callback control-flow graph



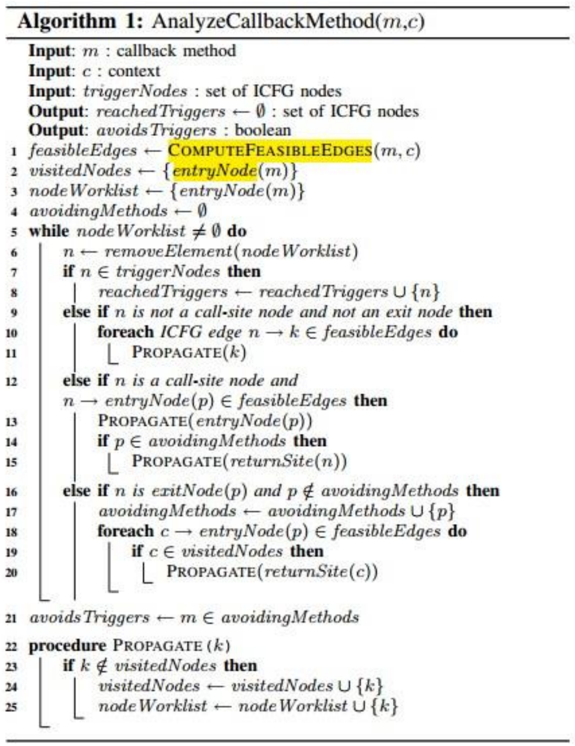
```
public class Main extends Activity {
   private EventHandler mHandler;
   public void onCreate() {
      this.setContentView(R.layout.main);
      mHandler = new EventHandler(this);
      int[] img_button_id = {R.id.info_button,
         R.id.help_button, R.id.manage_button,
         R.id.multiselect_button};
      for(int i = 0; i < img_button_id.length; i++) {
         ImageButton b = (ImageButton)findViewById(
            img_button_id[i]);
         b.setOnClickListener(mHandler);
      } } }

public class EventHandler implements
   OnClickListener {
   private final Activity mActivity;
   public EventHandler(Activity activity) {
      mActivity = activity;
   }
   public void onClick(View v) {
      switch(v.getId()) {
         case R.id.info_button:
            Intent info = new Intent(
               mActivity, DirectoryInfo.class);
            mActivity.startActivity(info);
            break;
         case R.id.help_button:
            Intent help = new Intent(
               mActivity, HelpManager.class);
            mActivity.startActivity(help);
            break;
         case R.id.manage_button:
            AlertDialog.Builder builder = ...
            AlertDialog dialog = builder.create();
            dialog.show();
            break;
         default:
            ...
            break; } } }

main.xml:
<LinearLayout>
  <ImageButton android:id="@+id/info_button"/>
  <ImageButton android:id="@+id/help_button"/>
  <ImageButton android:id="@+id/manage_button"/>
  <ImageButton android:id="@+id/multiselect_button"/>
</LinearLayout>
```

3. Algorithm

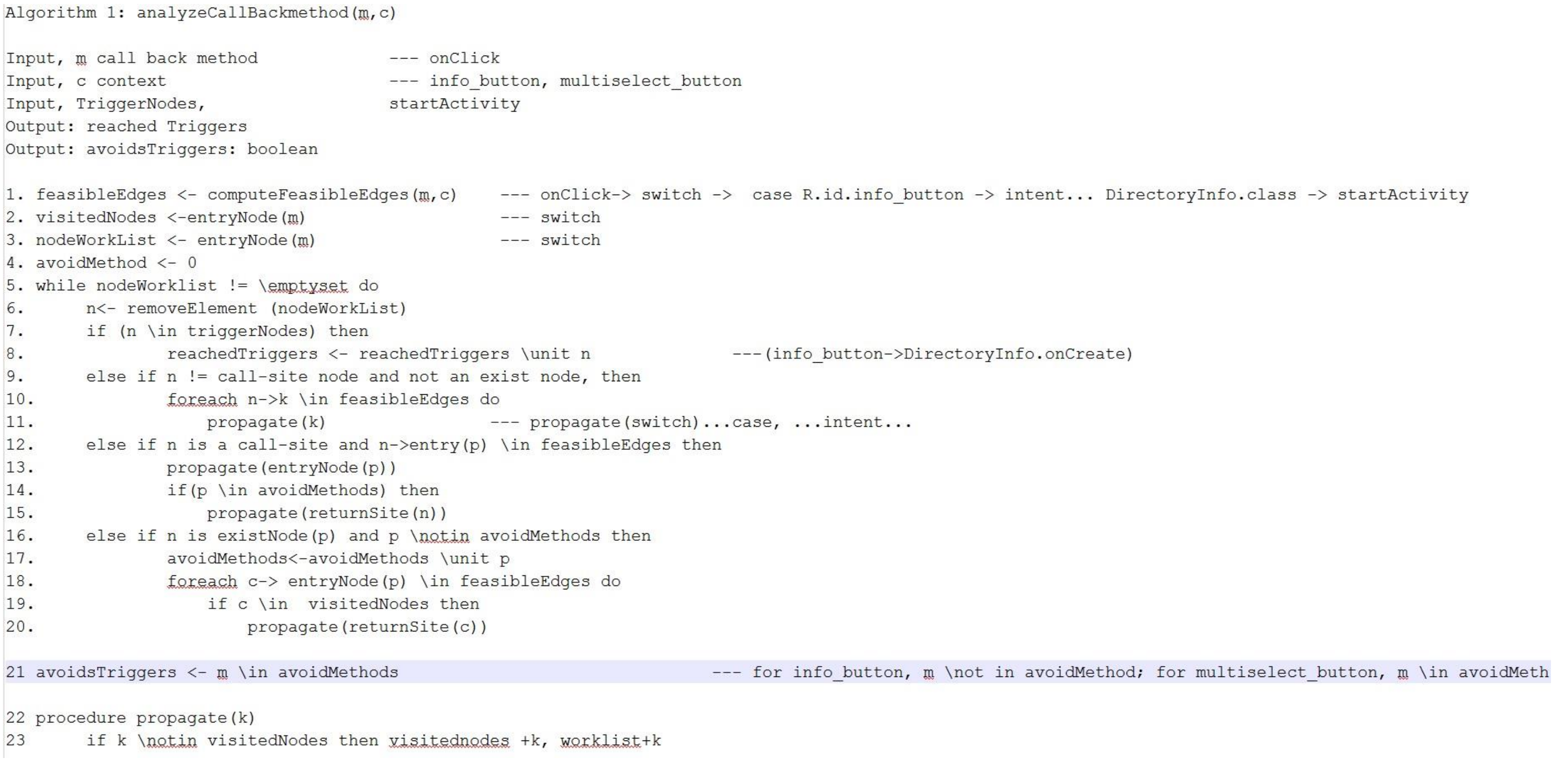
- 1. Control-flow analysis of a callback method:

- To indicate that event handlers could be executed in any order, branch nodes *bi* and join nodes *ji* are introduced, together with edges *ji* → *bi*.

-



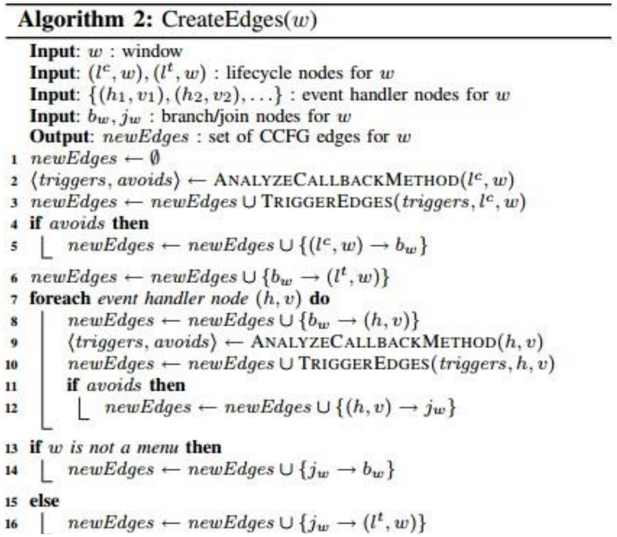
```
Algorithm 1: AnalyzeCallbackMethod($m$,$c$)
Input: $m$ : callback method
Input: $c$ : context
Input: $triggerNodes$ : set of ICFG nodes
Output: $reachedTriggers \leftarrow \emptyset$ : set of ICFG nodes
Output: $avoidsTriggers$ : boolean
$feasibleEdges \leftarrow$ COMPUTEFEASIBLEEDGES$(m, c)$
$visitedNodes \leftarrow \{entryNode(m)\}$
$nodeWorklist \leftarrow \{entryNode(m)\}$
$avoidingMethods \leftarrow \emptyset$
while $nodeWorklist \neq \emptyset$ do
    $n \leftarrow removeElement(nodeWorklist)$
    if $n \in triggerNodes$ then
        $reachedTriggers \leftarrow reachedTriggers \cup \{n\}$
    else if $n$ is not a call-site node and not an exit node then
        foreach ICFG edge $n \rightarrow k \in feasibleEdges$ do
            PROPAGATE$(k)$
    else if $n$ is a call-site node and
    $n \rightarrow entryNode(p) \in feasibleEdges$ then
        PROPAGATE$(entryNode(p))$
        if $p \in avoidingMethods$ then
            PROPAGATE$(returnSite(n))$
    else if $n$ is $exitNode(p)$ and $p \notin avoidingMethods$ then
        $avoidingMethods \leftarrow avoidingMethods \cup \{p\}$
        foreach $c \rightarrow entryNode(p) \in feasibleEdges$ do
            if $c \in visitedNodes$ then
                PROPAGATE$(returnSite(c))$
$avoidsTriggers \leftarrow m \in avoidingMethods$
procedure PROPAGATE $(k)$
    if $k \notin visitedNodes$ then
        $visitedNodes \leftarrow visitedNodes \cup \{k\}$
        $nodeWorklist \leftarrow nodeWorklist \cup \{k\}$
```



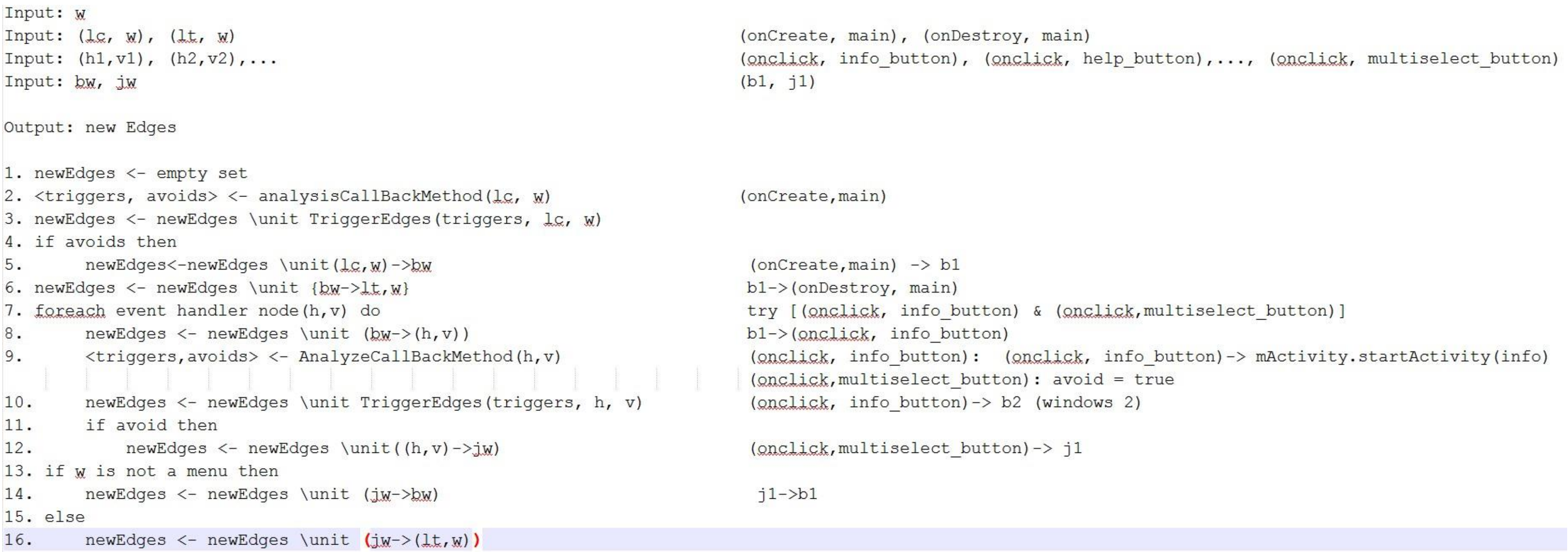
```
Algorithm 1: analyzeCallBackmethod(m,c)

Input, m call back method             --- onClick
Input, c context                      --- info_button, multiselect_button
Input, TriggerNodes,                  startActivity
Output: reached Triggers
Output: avoidsTriggers: boolean

1. feasibleEdges <- computeFeasibleEdges(m,c)    --- onClick-> switch ->  case R.id.info_button -> intent... DirectoryInfo.class -> startActivity
2. visitedNodes <-entryNode(m)                   --- switch
3. nodeWorkList <- entryNode(m)                  --- switch
4. avoidMethod <- 0
5. while nodeWorklist != \emptyset do
6.      n<- removeElement (nodeWorkList)
7.      if (n \in triggerNodes) then
8.              reachedTriggers <- reachedTriggers \unit n              ---(info_button->DirectoryInfo.onCreate)
9.      else if n != call-site node and not an exist node, then
10.             foreach n->k \in feasibleEdges do
11.                 propagate(k)                --- propagate(switch)...case, ...intent...
12.     else if n is a call-site and n->entry(p) \in feasibleEdges then
13.             propagate(entryNode(p))
14.             if(p \in avoidMethods) then
15.                 propagate(returnSite(n))
16.     else if n is existNode(p) and p \notin avoidMethods then
17.             avoidMethods<-avoidMethods \unit p
18.             foreach c-> entryNode(p) \in feasibleEdges do
19.                 if c \in  visitedNodes then
20.                     propagate(returnSite(c))

21 avoidsTriggers <- m \in avoidMethods                              --- for info_button, m \not in avoidMethod; for multiselect_button, m \in avoidMeth

22 procedure propagate(k)
23      if k \notin visitedNodes then visitednodes +k, worklist+k
```

3. Algorithm

- 2. CCFG Construction



```
Algorithm 2: CreateEdges($w$)
  Input: $w$ : window
  Input: $(l^c, w), (l^t, w)$ : lifecycle nodes for $w$
  Input: $\{(h_1, v_1), (h_2, v_2), \ldots\}$ : event handler nodes for $w$
  Input: $b_w, j_w$ : branch/join nodes for $w$
  Output: $newEdges$ : set of CCFG edges for $w$
1  $newEdges \leftarrow \emptyset$
2  $\langle triggers, avoids \rangle \leftarrow$ AnalyzeCallbackMethod($l^c, w$)
3  $newEdges \leftarrow newEdges \cup$ TriggerEdges($triggers, l^c, w$)
4  if $avoids$ then
5      $newEdges \leftarrow newEdges \cup \{(l^c, w) \rightarrow b_w\}$
6  $newEdges \leftarrow newEdges \cup \{b_w \rightarrow (l^t, w)\}$
7  foreach event handler node $(h, v)$ do
8      $newEdges \leftarrow newEdges \cup \{b_w \rightarrow (h, v)\}$
9      $\langle triggers, avoids \rangle \leftarrow$ AnalyzeCallbackMethod($h, v$)
10     $newEdges \leftarrow newEdges \cup$ TriggerEdges($triggers, h, v$)
11     if $avoids$ then
12         $newEdges \leftarrow newEdges \cup \{(h, v) \rightarrow j_w\}$
13 if $w$ is not a menu then
14     $newEdges \leftarrow newEdges \cup \{j_w \rightarrow b_w\}$
15 else
16     $newEdges \leftarrow newEdges \cup \{j_w \rightarrow (l^t, w)\}$
```



```
Input: w
Input: (lc, w), (lt, w)                                    (onCreate, main), (onDestroy, main)
Input: (h1,v1), (h2,v2),...                                (onclick, info_button), (onclick, help_button),..., (onclick, multiselect_button)
Input: bw, jw                                              (b1, j1)

Output: new Edges

1. newEdges <- empty set
2. <triggers, avoids> <- analysisCallBackMethod(lc, w)     (onCreate,main)
3. newEdges <- newEdges \unit TriggerEdges(triggers, lc, w)
4. if avoids then
5.     newEdges<-newEdges \unit(lc,w)->bw                   (onCreate,main) -> b1
6. newEdges <- newEdges \unit {bw->lt,w}                   b1->(onDestroy, main)
7. foreach event handler node(h,v) do                      try [(onclick, info_button) & (onclick,multiselect_button)]
8.     newEdges <- newEdges \unit (bw->(h,v))              b1->(onclick, info_button)
9.     <triggers,avoids> <- AnalyzeCallBackMethod(h,v)      (onclick, info_button):  (onclick, info_button)-> mActivity.startActivity(info)
                                                            (onclick,multiselect_button): avoid = true
10.    newEdges <- newEdges \unit TriggerEdges(triggers, h, v)  (onclick, info_button)-> b2 (windows 2)
11.    if avoid then
12.        newEdges <- newEdges \unit((h,v)->jw)            (onclick,multiselect_button)-> j1
13. if w is not a menu then
14.    newEdges <- newEdges \unit (jw->bw)                   j1->b1
15. else
16.    newEdges <- newEdges \unit (jw->(lt,w))
```

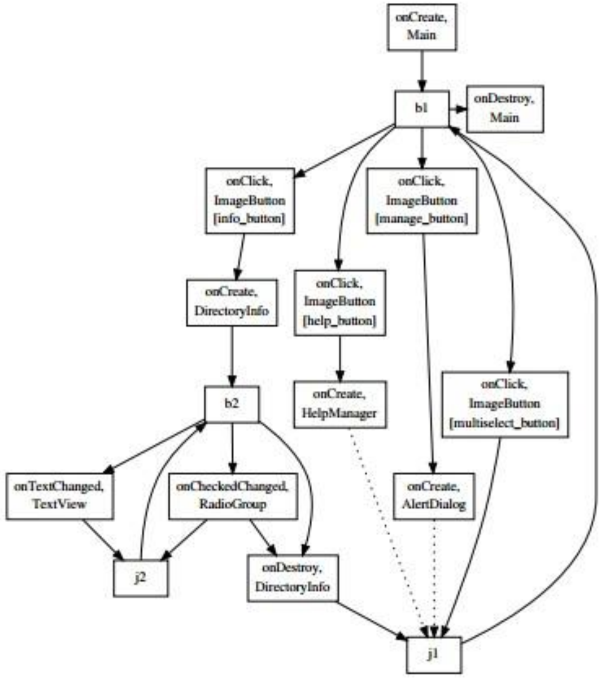

onCreate, Main
b1
onDestroy, Main
onClick, ImageButton [info_button]
onClick, ImageButton [manage_button]
onCreate, DirectoryInfo
onClick, ImageButton [help_button]
b2
onCreate, HelpManager
onClick, ImageButton [multiselect_button]
onTextChanged, TextView
onCheckedChanged, RadioGroup
onCreate, AlertDialog
j2
onDestroy, DirectoryInfo
j1

Fig. 2: Callback control-flow graph

4. Usage

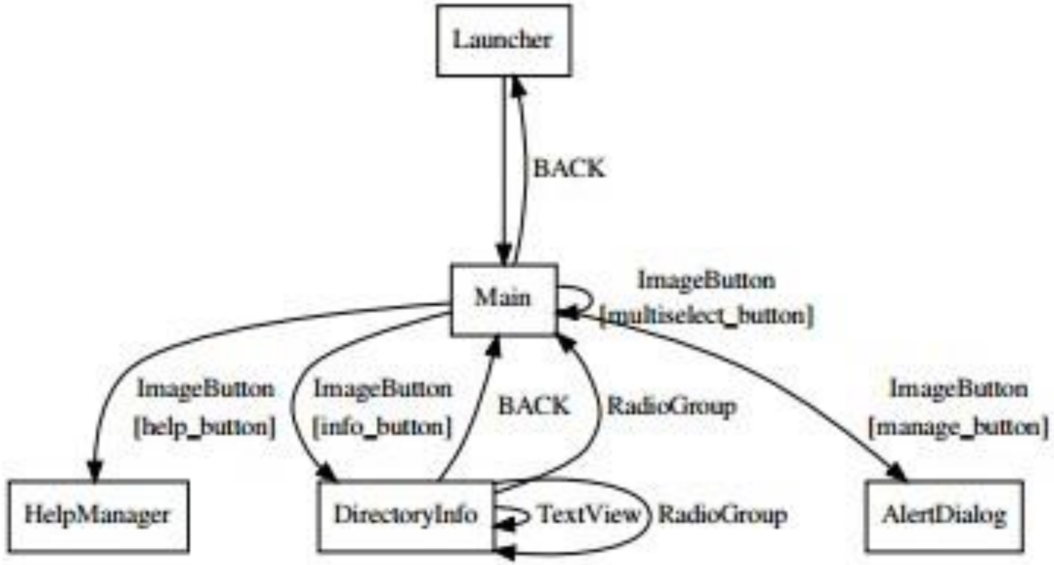
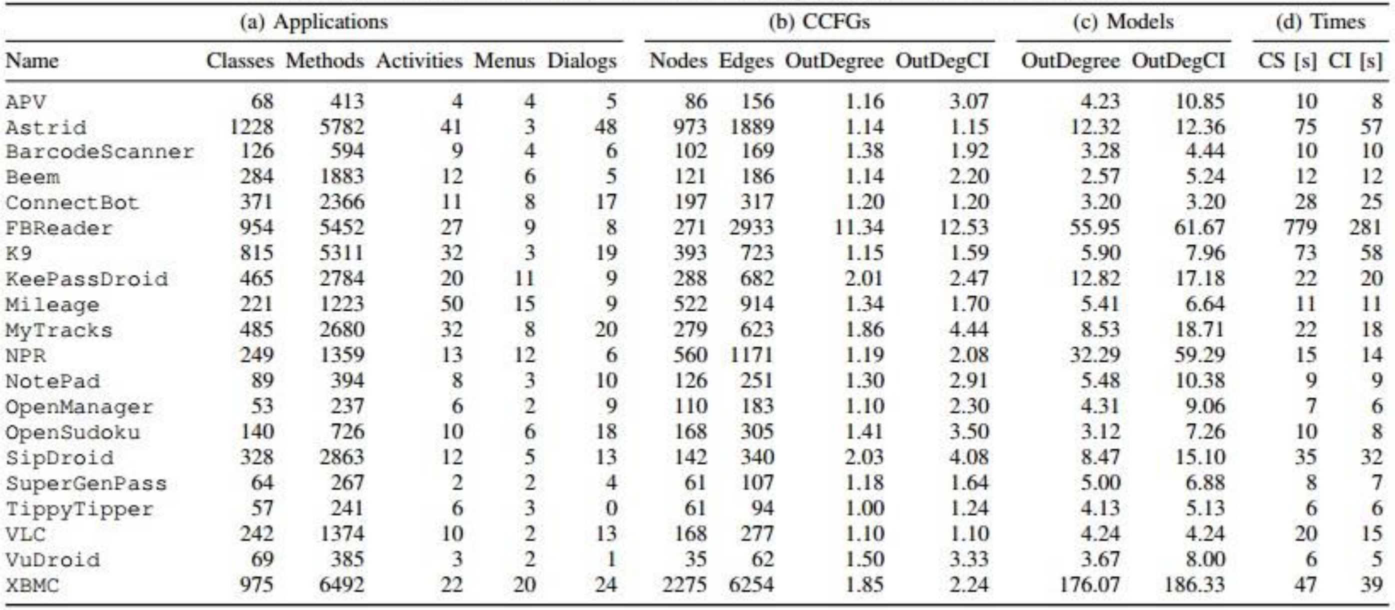


Fig. 3: GUI model for the running example

5. Evaluation

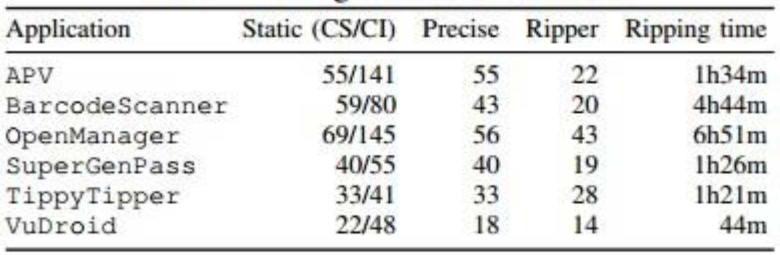
- (1) characterize the size and complexity of the CCFG,
- (2) measure the benefits of context sensitivity in the analysis of event handlers,
- (3) evaluate the precision of the GUI models derived from the CCFG.

TABLE I: Characteristics of the analyzed applications and their CCFGs.



(a) Applications						(b) CCFGs				(c) Models		(d) Times	
Name	Classes	Methods	Activities	Menus	Dialogs	Nodes	Edges	OutDegree	OutDegCI	OutDegree	OutDegCI	CS [s]	CI [s]
APV	68	413	4	4	5	86	156	1.16	3.07	4.23	10.85	10	8
Astrid	1228	5782	41	3	48	973	1889	1.14	1.15	12.32	12.36	75	57
BarcodeScanner	126	594	9	4	6	102	169	1.38	1.92	3.28	4.44	10	10
Beem	284	1883	12	6	5	121	186	1.14	2.20	2.57	5.24	12	12
ConnectBot	371	2366	11	8	17	197	317	1.20	1.20	3.20	3.20	28	25
FBReader	954	5452	27	9	8	271	2933	11.34	12.53	55.95	61.67	779	281
K9	815	5311	32	3	19	393	723	1.15	1.59	5.90	7.96	73	58
KeePassDroid	465	2784	20	11	9	288	682	2.01	2.47	12.82	17.18	22	20
Mileage	221	1223	50	15	9	522	914	1.34	1.70	5.41	6.64	11	11
MyTracks	485	2680	32	8	20	279	623	1.86	4.44	8.53	18.71	22	18
NPR	249	1359	13	12	6	560	1171	1.19	2.08	32.29	59.29	15	14
NotePad	89	394	8	3	10	126	251	1.30	2.91	5.48	10.38	9	9
OpenManager	53	237	6	2	9	110	183	1.10	2.30	4.31	9.06	7	6
OpenSudoku	140	726	10	6	18	168	305	1.41	3.50	3.12	7.26	10	8
SipDroid	328	2863	12	5	13	142	340	2.03	4.08	8.47	15.10	35	32
SuperGenPass	64	267	2	2	4	61	107	1.18	1.64	5.00	6.88	8	7
TippyTipper	57	241	6	3	0	61	94	1.00	1.24	4.13	5.13	6	6
VLC	242	1374	10	2	13	168	277	1.10	1.10	4.24	4.24	20	15
VuDroid	69	385	3	2	1	35	62	1.50	3.33	3.67	8.00	6	5
XBMC	975	6492	22	20	24	2275	6254	1.85	2.24	176.07	186.33	47	39

TABLE II: Edges in the GUI model.



Application	Static (CS/CI)	Precise	Ripper	Ripping time
APV	55/141	55	22	1h34m
BarcodeScanner	59/80	43	20	4h44m
OpenManager	69/145	56	43	6h51m
SuperGenPass	40/55	40	19	1h26m
TippyTipper	33/41	33	28	1h21m
VuDroid	22/48	18	14	44m

Questions:

- 1. Why cannot such method be used in onNewLocation?
- 2. What’s the strength of this paper? The weakness?
- 3. About the writing pattern