SOFTWARE MAINTENANCE AND EVOLUTION
--- REFACTORING FOR ASYNC : PART 2---

CS563
WEEK 3 – THU

Danny Dig
Oregon State University OSU
Overview of Our Transformations for Asynchronous Programming

- sync
- async
- modern async
- performant async
- long-running async
Refactorings for Android Asynchronous Programming

Yu Lin, Semih Okur, Danny Dig
Async Constructs in Android

<table>
<thead>
<tr>
<th>Thread 1996</th>
<th>AsyncTask 2008</th>
<th>IntentService</th>
<th>AsyncTaskLoader 2011</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ oldest</td>
<td>+ interact UI</td>
<td>+ interact UI</td>
<td>+ interact UI</td>
</tr>
<tr>
<td>- interact UI</td>
<td>- short-running</td>
<td>+ long-running</td>
<td>+ long running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- distributed</td>
<td>- Android 3+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>programming</td>
<td>- limited uses</td>
</tr>
</tbody>
</table>

DEMO refactoring.info/tools/asynchronizer
refactoring.info/tools/asyncdroid
AsyncTask can lead to memory leaks, lost results, and wasted energy.
Outline

Formative study to map the async programming landscape (avoid wrong assumptions)

Refactoring between async variants

Empirical evaluation
A Formative Study to Understand Async Programming in Android Apps

RQ1: How do developers introduce async constructs?

RQ2: Which async constructs are used most?

RQ3: How do expert developers interpret the disparity in usage of async constructs?
RQ1: How do Android developers introduce async constructs?

93 Android apps (1.5M SLOC)

We searched commits where async constructs are introduced first time, then manually analyzed them.

<table>
<thead>
<tr>
<th>Action</th>
<th>#Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newly added AsyncTask</td>
<td>277</td>
</tr>
<tr>
<td>Refactor sync code to AsyncTask</td>
<td>103</td>
</tr>
<tr>
<td>Refactor Thread to AsyncTask</td>
<td>18</td>
</tr>
<tr>
<td>Newly added IntentService</td>
<td>205</td>
</tr>
<tr>
<td>Refactor sync code to IntentService</td>
<td>13</td>
</tr>
<tr>
<td>Refactor Thread to IntentService</td>
<td>18</td>
</tr>
<tr>
<td>Refactor AsyncTask to IntentService</td>
<td>9</td>
</tr>
</tbody>
</table>
RQ2: Which async constructs are used most?

- 500 most popular Android projects from Github, comprising 4.7M SLOC
  - AST-based static analysis

<table>
<thead>
<tr>
<th></th>
<th># Instances</th>
<th># Apps</th>
<th>App %</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsyncTask</td>
<td>938</td>
<td>97</td>
<td>19%</td>
</tr>
<tr>
<td>Thread</td>
<td>655</td>
<td>110</td>
<td>22%</td>
</tr>
<tr>
<td>IntentService</td>
<td>182</td>
<td>30</td>
<td>6%</td>
</tr>
<tr>
<td>AsyncTaskLoader</td>
<td>118</td>
<td>14</td>
<td>3%</td>
</tr>
</tbody>
</table>
RQ3: How do expert developers interpret the disparity?

- We contacted 10 experts on StackOverflow – top 10 users for answering “android async” related questions – on average, each one answered 2095 questions

- Q1: Why are there still lots of legacy-style Thread uses?
  - familiarity with Java Thread
  - broader applicability: parallelism, scheduled tasks
RQ3: How do expert developers interpret the disparity?

Q2: Why is AsyncTask used much more than IntentService and AsyncTaskLoader?
   - AsyncTask “is overused” at the expense of other two
   - advertised more
   - developers not familiar with Android memory management
   - AsyncTask should only be used for short tasks < 1 sec

Q3: Would an automated refactoring from AsyncTask to IntentService be useful?
   - “It would be quite a challenge to do automatically”
   - “It may educate beginners to understand the construct”
Outline

Formative study

Empirical evaluation
AsyncTask to IntentService Refactoring

• Observations from the study
  – *AsyncTask* “is being overused” at others’ expense
  – *AsyncTask* can lead to memory leak, lost results and wasted energy, especially for long-running tasks
  – Manual refactorings exist
  – Automated refactoring can be beneficial

• *AsyncDroid* tool
IntentService execution

startService

register BroadcastReceiver

send Broadcast

onReceive

onHandleIntent

background task

GUI update

GUI

Broadcast Receiver

Intent Service

background thread

OS/Library

CPU/IO operations
AsyncDroid Code Transformation

```java
static String FILTER = "RS FILTER";

void onCreate() {
    final String query = "select ...";
    registerReceiver(new ProcessRoutes(), new IntentFilter(FILTER));
    Intent intent = new Intent(this, RS.class);
    intent.putExtra("query", query);
    startService(intent);
}

class ProcessRoutes extends BroadcastReceiver {
    void onReceive(Intent result) {
        Cursor mCsr = result.getExtra("result");
        textView.setText(mCsr.getCount());
    }
}

cursor doInBackground(String... args) {
    String query = args[0];
    Cursor mCsr = db.rawQuery(query);
    return mCsr;
}

class RS extends IntentService {
    void onHandleIntent(Intent intent) {
        String query = intent.getStringExtra("query");
        Cursor mCsr = db.rawQuery(query);
        Intent result = new Intent(FILTER);
        result.putExtra("result", mCsr);
        sendBroadcast(result);
    }

    static class FILTER = "RS_FILTER";
}
```

- Setup Filter
- Register Receiver
- Establish comm
- Start service
- Create Receiver
- Update GUI
- Create IntentService
- Background work
- Input/Output args
Challenge #1: Find which Objects to Transfer from/to IntentService

Incoming Variables (IV)
- variables passed into `AsyncTask doInBackground`

Outgoing Variables (OV)
- variables sent from `AsyncTask` to GUI
- return value of `AsyncTask doInBackground`
- outer class’ fields modified by `AsyncTask doInBackground` (IntentService holds a different copy of the fields)

Collecting IV and OV needs interprocedural analysis
- Mark IV and OV as serializable

```java
void onCreate() {
    final String query = "select ...",
    new ProcessRoutes().execute(query);
}

class ProcessRoutes extends AsyncTask{
    Cursor doInBackground(String... args){
        String query = args[0];
        Cursor mCsr = db.rawQuery(query);
        return mCsr;
    }
}
```
Challenge #2: Where to Register the BroadcastReceiver?

- **Naïve approach**
  - The place where `IntentService` is started
  - May lead to lost task result

- **Our approach**
  - Register in lifecycle event handler, guaranteed to be invoked during GUI recreation (e.g., `onCreate`)

- **Analysis:**
  - Variables accessed by the Receiver are still visible after moving to lifecycle handler (i.e., no syntax error)
  - Variables used by the Receiver are not redefined in other lifecycle event handlers (i.e., `def-use` is not changed)

- **GUI recreation occurs while the task is running**
  - May lead to lost task result
  - The new GUI cannot receive the result

```java
void onClick() {
    registerReceiver(new Receiver(), ...);
    startService(intent);
}
```
Refactoring Preconditions

• **P1**: Incoming and outgoing variables are serializable
  – *Intent* can only carry serializable objects
• **P2**: All the methods invoked in *doInBackground* should also be visible by *IntentService*
  – *IntentService* is *static* while *AsyncTask* is usually not
• **P3**: The refactored task is directly extended from *AsyncTask* and is not subclassed
  – Do not break inheritance relations
• **P4**: An *AsyncTask* instance is only used when task is started (i.e., invoking *AsyncTask.execute*)
Outline

Formative study

Refactoring between async variants
Experimental Evaluation

RQ1  Applicability: refactorings pass the preconditions

RQ2  Effort saved: number of difficult changes

RQ3  Accuracy and Value: refactorings accepted by developers

Methodology

– 9 popular Android apps from Github (0.4M SLOC)
– Use AsyncDroid to refactor *all* 97 AsyncTask tasks that hold references to GUIs
Refactoring is Broadly Applicable

<table>
<thead>
<tr>
<th>Applicability</th>
<th>Passed Precs</th>
<th>Conditional Passed</th>
<th>Failed Precs</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td><strong>44</strong></td>
<td><strong>10</strong></td>
<td><strong>43</strong></td>
<td><strong>41</strong></td>
<td><strong>7</strong></td>
<td><strong>1</strong></td>
<td><strong>21</strong></td>
</tr>
</tbody>
</table>

44 of 97 (45%) passed preconditions

10 (10%) conditional passed precondition with small edits
- some edits are automatable, e.g., converting fields into local variables is supported by Eclipse’s refactoring

41 fail P1: incoming/outgoing vars are not serializable
## Automated Refactoring Saves Programmer Effort

<table>
<thead>
<tr>
<th>#IV</th>
<th># OV</th>
<th>Moved Methods</th>
<th>Serialized Types</th>
<th>Moved receivers</th>
<th>SLOC Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>81</td>
<td>67</td>
<td>14</td>
<td>18</td>
<td>30</td>
</tr>
</tbody>
</table>

On average, each refactoring changes 63 SLOC

Non-trivial changes require deep analysis

Computing $IV$, $OV$, moved methods, serialized types needs traversing Call Graphs and Type Hierarchy
AsyncDroid is Accurate and Valuable

45 patches in 7 apps

Accepted:
15 in 2 apps

WhatAndroid: “These tasks are good fit for migration. I’ll migrate over to IntentServices”

Rejected:
16 in 2 apps

AntennaPod: “most of our tasks are short and should finish well under 100ms”
Summary: Improve Responsiveness via Refactorings for Asynchrony

Formative study on Android async constructs
  – Several constructs, with tradeoffs
  – AsyncTask only fit for short-running tasks (memory leaks, lost results and wasted energy)
  – Need refactorings to convert among constructs

AsyncDroid
  – Convert AsyncTask (shared-memory style communication) to IntentService (distributed-style communication)
  – Evaluation shows the refactoring is useful

http://refactoring.info/tools/AsyncDroid
Making a Broader Impact

• Integrate *CTADetector* and *AsyncDroid* with Google *ShipShape*
  – *ShipShape* is an open-source static analysis platform
Discussion

• What was the most unexpected result?
• What did the authors do right?
• What could have been improved?
• New ideas for research?
Paper Discussion

New research ideas

- Miguel: relaxing P1 (type defined in the app, not in the library) to increase usage
- Braden: refactoring from Thread to Async
-  - generate code for serialization to further increase usage (P1)
- Deval, Malinda: extend to other languages, JavaScript, Kotlin
- Malinda: migrate bytecode in APK
- Dan: compare AsyncDroid with alternative tools on performance and usability
- Vijay: integrate with AndroidStudio
Open questions

Paper Discussion

- Miguel: how did the authors select Corpus 2 – 3 to be mutually exclusive.
- Braden: Does AsyncDroid held when preconditions fail?
  - tradeoffs in analyzing plain txt vs AST

Deval: why only 45 patches from 611 apps?

Dan: how do we help novice programmers?

Malinda: why not handle Content provider?

Vijay: AsyncDroid-like tools for Desktop software?