Android Wear 应用程序中的通知分析和测试

Hailong Zhang and Atanas Rountev
Ohio State University, Columbus, OH, USA
Email: {zhanhail,rountev}@cse.ohio-state.edu

摘要 - Android Wear（AW）是开发可穿戴设备应用程序的 Google 平台。 我们的目标是迈向 AW 应用程序分析和测试基础的第一步。 我们关注这些应用程序的一个核心功能：手持设备（例如，智能手机）发布并显示在可穿戴设备（例如智能手表）上的通知。 我们首先定义 AW 通知的正式语义，以便捕获通知机制的核心功能和行为。 接下来，我们描述一个基于约束的静态分析来构建这种运行时行为的模型。 然后，我们使用这个模型为 AW 应用程序开发一个新颖的测试工具。 该工具包含一个测试框架和组件，以支持 AW 特定的覆盖标准，并自动生成可穿戴设备上的 GUI 事件。 这些贡献推动了可穿戴设备软件日益重要的领域的发展。

Ⅰ.介绍

可穿戴设备

电子可穿戴设备被设计为佩戴在身体上以便实现移动性和免提/无眼睛活动。虽然智能手表和健身腕带是目前使用最广泛的此类设备，但其他设备类别也有望越来越受欢迎，其中包括头戴式显示器，智能珠宝，身体相机和智能服装。

传统的移动设备需要直接操作，导致高的认知和感知负载，导致用户分心。可穿戴设备应该减少这种负载，并允许嵌入式的交互，上下文感知，个性化，适应性和预期性。长期趋势是朝向具有环境和生理传感器（例如，GPS，加速度计，心率）丰富的设备，其在医疗保健，健身，娱乐，制造，建筑，现场工作等方面具有广泛的用途。预计可穿戴设备成为计算领域发展最快的市场之一。最近的一份行业报告预测，2020 年将有超过 7600 万智能可穿戴设备出货，其中 2280 万台将以 Android

为基础[1]。

为可穿戴设备编写的软件应用程序为软件工程研究人员提出了各种有趣的挑战 - 例如，安全/隐私，功耗，针对设备限制进行优化的 UI 以及由于快速发展的市场导致的软件演进。在此背景下，开发一系列关于静态和动态分析以进行程序理解，测试，调试，优化和演进的工作将是至关重要的。

Android Wear

Android Wear（AW）是 Google 开发面向可穿戴设备应用的软件平台[2]。在高层次上，有两类 AW 应用程序。首先，可穿戴设备可以与通常为智能手机或平板电脑的配套手持设备结合使用。可穿戴设备上的软件和手持设备上的软件通过平台 API 进行交互。第二种情况是独立可穿戴设备包含独立运行的软件。AW 1.x 不支持独立应用程序，但由于 AW 2.0 中的更好支持（2017 年 2 月正式发布），预计它们将更受欢迎。对于本文的其余部分，我们考虑 AW 应用程序，其中软件既可以在可穿戴设备上运行，也可以在同伴手持设备上运行。

我们的工作专注于 AW 应用程序的核心功能：手持设备发布并显示在可穿戴设备上的通知。通知的构建和发布是 Google AW 开发人员指南[3]介绍的第一个主题。当显示通知时，用户可以执行操作，将控制流返回给掌上电脑。

我们的贡献

据我们所知，AW 应用行为的这个关键方面在以前的工作中还没有被研究过。鉴于可穿戴设备的重要性日益增加，建立可穿戴应用程序分析和测试基础非常重要。我们的工作贡献可以总结如下。首先，我们定义 AW 通知的正式语义。使用抽象语法和操作语义，我们捕获通知机制的核心行为。其次，我们描述一个静态分析来建立这种运行时行为的静态模型。分析基于相关运行时实体的静态抽象，以及这些实体之间重要关系的约束表示。第三，我们用这个模型来开发一本小说 AW 应用程序的测试工具。该工具包含（1）用于定义和执行跨两个设备的测试的测试框架，（2）用于测量 AW 特定覆盖标准的运行时覆盖的组件，以及（3）用于自动生成 GUI 事件的组件在可穿戴设备上。最后，我们提出实验结果和案例研究来评估所提出的技术。我们计划在不久的将来公开我们的实施和实验主题。

Ⅱ. 背景和例子

我们的重点是定义并在手持设备（例如智能手机）上运行的 Android Wear 应用程序，但使用可穿戴设备（例如智能手表）向用户显示通知并接收用户反馈。实质上，可穿戴设备成为手持设备的 GUI 的扩展。实际上，这意味着有一个应用 APK（在手持设备上运行），并且在此 APK 中发出 API 调用以在可穿戴设备上触发某些行为。在对 Google Play 应用的探索性研究中，我们考虑了每个应用类别中的前 100 个应用，并确定了包含可穿戴专用代码的 283 个应用。其中，57％有这种结构。另外两种替代方案也是可能的。首先，可以在手持设备上运行 APK，并在可穿戴设备上运行另一个 APK，与相关 API 提供的设备间通信。其次，在可穿戴设备上可能会有独立的 APK，而不需要配套的手持设备。虽然这两种情况对未来的工作都很有意思，但在此不予考虑。

```
1  class MyNotificationManager {
2    void create() {
3      Builder builder = new Builder();
4      Intent mainIntent = new Intent(MainActivity.class);
5      PendingIntent mainPI = PendingIntent.getActivity(mainIntent);
6      WearableExtender extender = new WearableExtender();
7      if (...) {
8        Notification chatPage = new Builder().build();
9        extender.addPage(chatPage);
10     }
11     Intent replyIntent = new Intent(RemoteMessagingReceiver.class);
12     PendingIntent replyPI = PendingIntent.getBroadcast(replyIntent);
13     Action replyAction = new Action.Builder(replyPI).build();
14     extender.addAction(replyAction);
15     Intent readIntent = new Intent(MarkReadReceiver.class);
16     PendingIntent readPI = PendingIntent.getBroadcast(readIntent);
17     Action readAction = new Action.Builder(readPI).build();
18     extender.addAction(readAction);
19     builder.setContentIntent(mainPI).extend(extender);
20     NotificationManager.notify(builder.build());
21   }
22 }
```

图 1. 来自 QKSMS 的简化代码

通知显示为可穿戴设备上的一系列屏幕。向左和向右滑动允许用户在屏幕之间导航。有两类屏幕。一个页面显示通知的内容，包括标题，文本和图标。它是一个被动的实体 - 用户观察信息但不与其交互。一个动作是含有一个标题和动作按

钮的屏幕； 用户可以通过触发在手持设备上执行的代码来点击该按钮来执行某些期望的功能。

A.示例 Android Wear 应用程序

图 1 展示了来自 QKSMS 开源 Android Wear 应用程序的简化版代码。为了清楚起见，删除或简化了非必要的细节。此消息传递应用程序与智能手表进行交互以发出通知。在第 20 行通知的呼叫会在智能手表上显示多个屏幕，如图 2 所示。主页面首先显示。该页面的标题是“测试帐户”（消息发送者标识符），页面文本“Aloha”是消息的内容。如果用户向左滑动，则会显示另一个嵌套页面，其中包含此消息发件人的聊天记录。从右向左滑动显示“回复”动作。通过额外的滑动，用户可以访问另外三个动作。最后一个（“Block 应用程序”）是一个默认的 AW 动作，阻止来自这个应用程序的进一步通知。

通知至少包含一个页面（主页面）以及“Block 应用程序”操作。主页面之后可能还有其他页面。这些页面后面是一系列操作。当用户触摸操作按钮时，AW 框架会在手持设备上执行代码。例如，对于“在手机上打开”操作，手持设备上将打开一个屏幕以显示对话列表。执行的代码位于 MainActivity 类中，并使用图 1 中第 4 行的 Intent 对象触发。

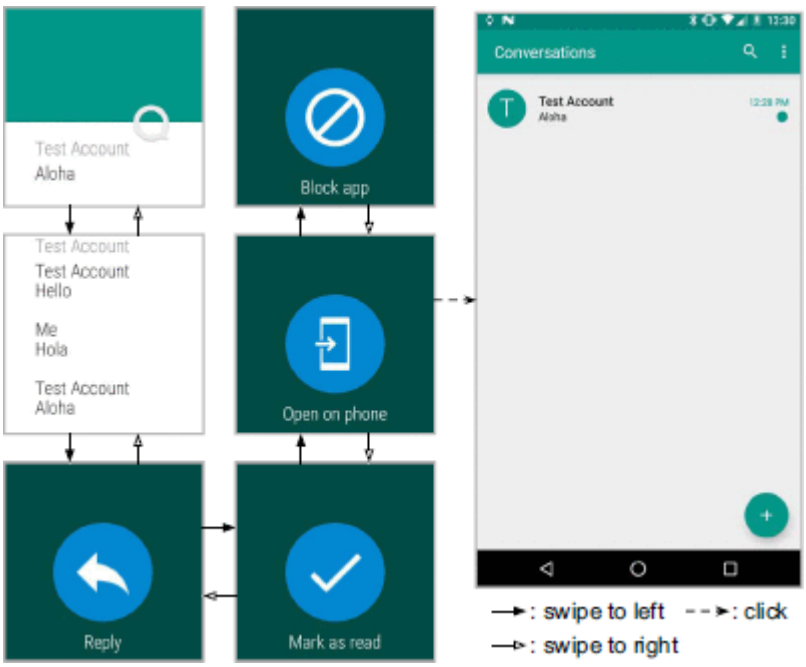

图 2. 智能手表上的屏幕

B.主要概念和 API

通知机制的关键概念是：（1）通知构建器对象用作通知对象的工厂；（2）可穿戴扩展器是辅助对象，当它应用于通知构建器时，会导致创建可穿戴专用通知；（3）可以在通知中包括几个动作以允许可穿戴设备的用户作出响应；（4）动作内部的意图决定哪个手持应用程序组件作为响应被调用；（5）嵌套页面也可以包含在扩展器/构建器/通知中。

图 1 中的第 3 行和第 8 行创建了通知构建器对象。这些是 NotificationCompat.Builder 类的实例，缩写为 Builder 中的示例。第 6 行创建了一个可穿戴的扩展器。构建器和扩展器最终用于创建通知对象（调用在第 20 行构建），并将其显示在可穿戴设备上（调用以通知第 20 行）。

通常，通知可以显示在手持设备和可穿戴设备上。穿戴式专用通知是使用可穿戴式扩展器对象创建的。扩展器为构建器添加更多功能。例如，在第 19 行的扩展调用将扩展器的动作和嵌套页面添加到构建器中。早期的 API 调用通过这些操作（第 13 行和第 18 行，调用 addAction）和嵌套页面（第 8 行，调用 addPage）填充扩展器。

动作对象描述了要在可穿戴设备上显示的屏幕。屏幕包含一个标题（例如，“标记为已读”）并具有基本意图。当用户滑动到此屏幕并触摸图标时，意图用于触发手持设备上的应用程序组件。对于正在运行的示例，使用助手操作构建器对象在第 13 行创建了“回复”的操作对象。此操作与执行 RemoteMessagingReceiver（第 11 行）的意图有关，该应用程序组件在手持设备上运行。该组件是广播接收器的一个例子，一种标准的 Android 组件类型，在后台运行并响应通过意图发送的请求。在第 15 行创建的另一个意图用于在手持设备上触发广播接收器 MarkReadReceiver，以响应在第 17 行创建的操作。这两个操作都添加到扩展器，然后复制到构建器（通过扩展），然后到通过构建在第 20 行创建的通知。

类 Intent 的实例包含要执行的操作的抽象描述。这是触发应用程序组件的通用 Android 机制。例如，如果手持设备应用中的一个活动（Android 中的另一个标准组件类型）想要在同一应用中触发另一个活动，它通常会调用 startActivity 并提供一个描述目标活动的意图参数。同样，对 sendBroadcast 的调用会根据给定的意图触发广播接收器。由于这种机制的广泛使用，之前的工作（例如，[4] - [5]

[6]）已经考虑了意图的语义及其静态建模。

为了将其用作我们工作中分析的通知机制的一部分（其可在两个设备上工作而不是在单个设备内部工作），它必须由助手 PendingIntent 对象包装。出于安全原因，意图应该几乎总是明确的[7]。第 12 行和第 16 行显示了这些辅助对象的创建。挂起的意图作为动作对象的一部分提供给 Android 通知管理器，并且当动作实际执行时（即用户触摸动作图标），挂起的意图用于访问基础“常规”意图。那时，诸如 startActivity 或 sendBroadcast 之类的调用的概念等价物就是使用该意图对象进行的。

在第 19 行对 setContentIntent 的调用用于向构建器添加默认的“在手机上打开”操作。此操作是作为此 API 调用的一部分隐式创建的。在此示例中，此操作的目标是 MainActivity（通过在第 4 行创建的意图）。此活动在手持设备上执行以显示对话列表。第 8 行创建一个通知对象，第 9 行使用 addPage 将其添加到扩展器，从而添加到第 20 行构建的通知中。请注意，第 8 行和第 20 行都会调用通知构建器上的构建，并生成 Notification 实例。在这种情况下，其中一条通知（第 20 行）对应于主通知页面，另一条通知（第 8 行）对应聊天记录的嵌套页面。

下一部分将 AW 通知的关键抽象形式化并定义它们的运行时效果。这种形式化作为所提出的静态分析的基础。

Ⅲ. AW 通知的形式化语义

AW 应用程序中通知的运行时语义的正式定义基于“简单”Java（基于[8]，[9]）和“简单”Android 的语义定义（源自我们之前的工作[10] - [11] [12]）以及我们针对 AW 应用程序新开发的一个正式化程序。

A.普通 Java 和普通 Android

1.普通的 Java

我们的讨论集中在方法体内个别语句的语义上。对类型系统的建模以及由于调用和返回而导致的行为是很好理解的（例如，[8]，[9]，[13] - ），并且为简单起见而被省略。

一个 Java 程序包含一组 Java 类。每个类定义一组字段 f∈Field 和一组

方法和构造函数。一个方法体包含局部变量的声明 x∈Var。以及其中节点是语句的控制流图。这些语句的语法由<语言>定义

$$s ::= x = \text{new } C \mid x = y \mid x = y.f \mid x.f = y$$

包含方法调用和其他 Java 特性的泛化是众所周知的，不予讨论。相应的语义基于堆对象的集合对象，定义局部变量如何引用这些对象的映射存储以及表示对象字段的值的映射堆。

o∈Obj 堆对象

σ ∈Store=Var→Obj 变量值

H∈Heap=(Obj×Field)→Obj 字段值

商店和堆的语义效应是

$$\begin{array}{lcl} \langle x = \text{new } C, \sigma, \mathcal{H}\rangle & \rightarrow & \langle \sigma[x \mapsto o], \mathcal{H}\rangle \\ \langle x = y, \sigma, \mathcal{H}\rangle & \rightarrow & \langle \sigma[x \mapsto \sigma(y)], \mathcal{H}\rangle \\ \langle x = y.f, \sigma, \mathcal{H}\rangle & \rightarrow & \langle \sigma[x \mapsto \mathcal{H}(\sigma(y), f)], \mathcal{H}\rangle \\ \langle x.f = y, \sigma, \mathcal{H}\rangle & \rightarrow & \langle \sigma, \mathcal{H}[(\sigma(x), f) \mapsto \sigma(y)]\rangle \end{array}$$

规则显示更新的商店/堆；a[b↦c] 表示该地图 a （重新）映射更新 b 至 c。对于 x=new C,o∈Obj 表示类的新堆对象 C；我们假设初始化 o 字段由表单的单独声明表示 x.f=y。

2.纯 Android

我们之前关于 Android GUI 分析的工作[10],[14]定义了几个重要 Android 功能（例如，活动，菜单，对话框，小部件，布局定义，事件监听器等）的与 GUI 相关的语义。这些定义与本文所考虑的问题没有直接关系，但下面描述的 AW 语义可以被视为这些现有定义的扩展。

B.Android Wear 中的通知

甲通知是一个应用程序的正常 GUI 以外显示一条消息。对于我们考虑的 AW 应用程序，运行在手持设备上的应用程序使用通知在伴随可穿戴设备上显示信息。

相关 AW 类的实例和所有这些实例的集合将被表示如下：

no∈Notif⊂Obj 通知　　nb⊂NotifBuilder⊂Obj 通知建设者

we∈WearExtender⊂Obj 可穿戴的扩展器　ac∈Action⊂Obj 动作

in∈Intent⊂Obj 意图　pi∈PendingIntent⊆Obj 待定意图

在手持设备应用程序中创建通知后，它可以触发可穿戴设备上的新屏幕。这是通过呼叫来完成的，如图 1 中的第 20 行所示。出于控制流和数据流分析的目的，通知会导致可穿戴设备上的事件处理逻辑的执行，其然后触发手持设备中的事件处理代码在诸如活动或广播的组件中接收器。

分析 Android 应用程序中的组件间控制流和数据流是非常重要的，并且一直是许多现有分析的目标（例如，[4] - [5] [6]，[11]，[12] - ）。对于 AW 应用程序，notify 是控制流出口点，必须与手持应用程序代码中的后续重新入口点匹配。实质上，通知机制为组件间控制/数据流提供了新路径，但是这次涉及两个设备。我们的静态分析是模拟这种组件间交互的第一种方法。控制流出口点和重入点的匹配是分析输出的一部分。这些信息可能会被其他静态分析及其客户端使用（例如，测试，调试，安全性分析和分析）。

C.建设者，扩展者和通知

几类 API 调用与构建器，扩展器和从它们创建的通知相关。与我们的目的相关的 API 调用的子集由语句的抽象语法的以下定义捕获 s：

$$\begin{aligned} s \ ::= \ & x = \text{addaction}(y, z) \mid x = \text{setaction}(y, z) \mid \\ & x = \text{extend}(y, z) \mid x = \text{build}(y) \mid \text{notify}(x) \end{aligned}$$

1.添加操作

抽象操作 addaction 表示一个 API 调用，它将操作添加到可穿戴扩展器，并因此添加到在此扩展器帮助下创建的可穿戴专用通知。参数 y 指的是扩展器，而 z 指的是被添加的动作。addaction 的返回值是对更新的扩展器的引用（即，x 和 y 是别名）。

2.默认操作

通知构建器可以具有默认的可穿戴特定操作“在手机上打开”，如运行示例中所示。如果在构建器上调用 setContentIntent（图 1 中的第 19 行），则会隐式创建此类默认操作并将其与构建器关联。我们使用抽象操作为这些效果建模

x=setaction(y,z) 哪里 y 指构建器和 z 指的是动作。一个字段默认存储这个关联。

setaction 的语义是映射 H(σ(y)，默认）为 σ(z) 并复制该值 y 至 x。

3.扩展 Builder

抽象操作 x=extend(y,z) 以输入为参考的通知构建器 y 和一个可穿戴的扩展器 z。返回值是对同一个构建器对象的引用。执行扩展时，扩展器当前状态的快照将存储在构建器中。在我们的定义中，这可以通过将扩展器的动作列表复制到构建器来建模。因此，我们在建造者中引入一个领域动作，并进行设定 H(σ(y)，weactions）具有的价值 H(σ(z)，weactions)。

4.Building 通知

一个操作 x=build(y) 使用引用的构建器的状态 y 创建并初始化通知对象 no∈NOTIF。局部变量 x 被分配了一个参考 no。对象状态的关键属性是动作列表，它需要以下堆扩展名：

$$\text{Heap} = \ldots \cup (\text{Notif} \times \{\text{actions}\} \to \text{Action}^*)$$

特定 nb=σ(y)，新通知的操作定义如下。如果我们采取动作 nb 不是空的，则新通知的操作字段被设置为 H(nb,weactions)○ H(nb,默认)。但是，如果 weactions 为空，则操作设置为 H(nb，nbactions） ◦H(nb，默认)。这种行为对应于两种情况。首先，如果 nb 被一个非空动作列表的扩展器所扩展，这些动作是在可穿戴设备上显示的动作（后面是 nb 的默认动作)。扩展器也可以不提供任何操作，而是设置其他选项 - 例如显示样式。在这种情况下，可穿戴设备显示直接添加到构建器的操作。

另外，在动作列表的末尾添加预定义的“Block 应用程序”操作，以阻止进一步的通知。图 2 说明了结果的一系列操作。

D.动作和意图

为了模拟与意图，未决意图和动作相关的 API 调用，我们定义了以下抽象语法：

$$s ::= x = \text{buildpending}(y) \mid x = \text{buildaction}(y)$$

操作 buildpending 摘要 API 调用将构建一个包含在引用的常规意图中的未决意图 y 如图 1 中的线 5,12 和 16 所示。等待的意图可以在创建新的操作对象时使用:在第二个生产中,y 指的是这个未决的意图。操作 buildaction 表示两种情况:(1)在新的 Action 表达式中构造一个构造函数,以及(2)使用动作构建器,如图 1 中的第 13 行和第 17 行所示。与通知构建者创建通知的方式类似,动作构建者可以创建操作。为简单起见,我们忽略了相关细节,但我们的实现处理了这两种情况。

无论动作对象是如何创建的,其内部状态的一部分都是未决意图。我们需要堆概括:

$$\begin{aligned}\text{Heap} = \ldots \cup (\text{PendingIntent} \times \{\text{intent}\} \to \text{Intent}) \\ \cup (\text{Action} \times \{\text{pending}\} \to \text{PendingIntent})\end{aligned}$$

buildpending 和 buildaction 的语义和预期一样,并没有详细显示。

E.嵌套页面

每个通知对象都显示一个主要通知页面。有时在嵌套页面上显示附加信息,当用户向左滑动时可访问。可以通过创建附加通知对象并将它们附加到主通知对象来添加此类页面。图 1 中的聊天 chatPage 是一个嵌套页面的例子。

抽象语法是 s::=x=addpage(y,z),在哪里 y 指的是可穿戴的扩展器和 z 指的是嵌套的通知对象。添加到扩展程序的页面序列可以由字段页面表示:

$$\text{Heap}=\cdots\cup(\text{WearExtender}\times\{\text{pages}\}\to\text{Notif}*)$$

H(σ(y),pages) 通过追加更新 σ(z);此外,y 被复制到 x。我们还需要将建设者和通知概括为类似的字段页面。扩展和构建的语义分别包括将页面的值复制到构建器或通知。

应该注意语义的另外两个方面。首先,假设有通知 no 包含一个嵌套页面 no′。即使 no′ 可能有自己的行为,他们不会影响行为 no。换一种说法,H(no,actions),是独立的 H(no,pages)。其次,什么时候 no 实际上显示在可穿戴设备上,重复向左滑动将首先显示其嵌套页面的顺序,然后显示其操作顺序。这一行为如图 2 所

示。

Ⅳ.静态分析

本部分描述了一个静态分析，该分析对通知相关对象的传播进行建模并确定它们之间的重要关系。使用约束图可以解决普通 Java 的类似参考传播问题。图节点对应于一个变量 x∈Var 一个领域 f∈Field，或者分配新的 C。边缘对值进行约束编码。例如，分配 x=y 由边表示 y→x，显示了这组值 y 是一组值的一个子集 x。新的向前可达性 C 确定哪些变量和字段引用 C 实例。这样的分析被分类为流量不敏感，上下文不敏感，基于现场的参考分析[15]，[16]。我们对 AW 应用程序的分析概括了这种方法。可以定义各种精确的扩展（例如，[9]，[16]，[17] - ），并可以与我们的 AW 特定分析相结合。

分析的概念输入是基于前面提出的抽象语义的程序表示。图 3 显示了这个运行示例的表示。分析实现对来自 Soot 分析框架的三地址 Jimple 表示[18]起作用，并在概念上将调用语句映射到这些抽象操作。

约束图

1. 操作节点

除了上面列出的标准约束图节点之外，我们还使用操作节点的集合 OP。上一

```
1  Builder a = new Builder();
2  Intent b = new Intent(MainActivity.class);
3  PendingIntent c = buildpending(b);
4  WearableExtender d = new WearableExtender();
5  Builder e = new Builder();
6  Notification f = e.build();
7  addpage(d,f);
8  Intent g = new Intent(RemoteMessagingReceiver.class);
9  PendingIntent h = buildpending(g);
10 Action i = buildaction(h);
11 addaction(d,i);
12 Intent j = new Intent(MarkReadReceiver.class);
13 PendingIntent k = buildpending(j);
14 Action l = buildaction(k);
15 addaction(d,l);
16 Action m = buildaction(c);
17 Builder n = setaction(a,m);
18 extend(n,d);
19 Notification o = build(a);
20 notify(o);
```

节中定义的抽象操作由这些节点表示。对于 x=op(y)，相应的节点 n 有一个来自变量节点的传入边缘 y，以及节点的输出边缘 x。如果该操作具有两个参数，则会有第二个传入边缘。图 4 显示了运行示例的约束图。数字后缀对应于图 3 中的行号。

图 3. 抽象程序表示

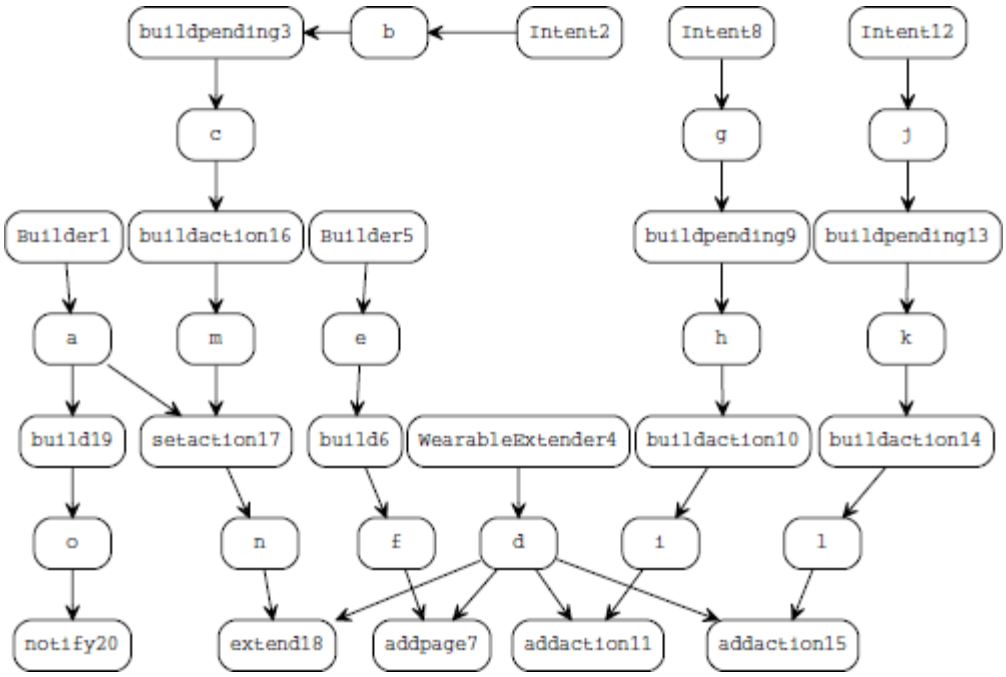

图 4. 运行示例的约束图。

2. 对象创建

节点集 NB，WE，IN，BN，BP 和 BA 表示由程序语句创建的对象。设 NB 为与通知构建器的新表达式（例如，图 4 中的节点 Builder2 和 Builder5 ）相对应的分配节点集合。同样，让 WE 为可穿戴扩展器新表达式的节点集合，IN 为意图的相似集合。

除了新的表达式，操作节点可能会创建新的对象。通知是使用创建的 x=build(y)。每个这样的操作对应于约束图节点 $n \in BN \subseteq OP$。在这个例子中，BN={build6,build19}。同样的，x=buildpending(y) 创建一个挂起的意图，并由节点表示 $n \in BP \subset OP$。最后，可以使用新表达式创建动作对象，也可以在动作构建器上创建构建调用。这两种情况都是抽象出来的 x=buildaction(y)，为此我们有一个节点 $n \in BA \subset OP$。在这个例子中，BA 包含三个 buildaction 节点。

A. 基于约束的分析

我们根据几种关系来定义分析。使用推理规则以声明方式描述这些关系。稍后我们将描述关系是如何计算的。

对象引用的流程由 flowsto 表示 $\subseteq(NB \cup WE \cup IN \cup BN \cup BP \cup BA) \times (Var \cup Field \cup OP)$。一双 n flowsto n′ 表明一个对象表示 n 传播到操作的变量，字段或参数。标准传播的推理规则很简单。首先，对于一个节点 $n \in NB \cup WE \cup IN \cup BN \cup BP \cup BA$ 左侧变量 x,n→x 暗示 n flowsto x。此外，传递性被定义为预期的：任何 n,n′,n′′，我们有这个 n flowsto n′ 和 n′→n′′ 意味着 n flowsto n′′。

1. 建设者和扩展者

其他关系用于捕捉第三节介绍的 AW 特定抽象。例如，x=setaction(y,z) 作为输入的建设者 y 和一个动作 z。关系默认$\subseteq NB \times BA$ 代表相应节点的影响 n 定

$$\frac{nb \text{ flowsto}_1 \ n \quad ac \text{ flowsto}_2 \ n \quad n \to x}{nb \text{ flowsto } x \quad nb \text{ default } ac}$$

义如下：

这里下标表示流量是对操作的第一个还是第二个参数。扩展器（或构建器）上的 addaction 规则类似：它为二元关系过程添加一对$\subseteq WE \times BA$ （或 nbactions $\subseteq NB \times BA$）。

代表影响 x=extend(y,z) 我们使用关系扩展 $\subseteq WE \times NB$

$$\frac{nb \text{ flowsto}_1 \ n \quad we \text{ flowsto}_2 \ n \quad n \to x}{nb \text{ flowsto } x \quad we \text{ extends } nb}$$

3. 通知

操作 x=build(y) 基于构建器创建新的通知 y。此构建器的状态以及其关联的扩展器的状态决定了通知的内容。因此，我们希望将构建调用站点，构建器和扩展器的三

$$\frac{nb \text{ flowsto } bn \quad we \text{ extends } nb}{(bn, nb, we) \in \text{NO}}$$

元组记录为运行时通知对象的静态抽象。让 $NO \subseteq BN \times NB \times WE$ 表示所有这些记录的三元组的集合。对于一个节点 bn 代表一个建造操作，我们有

设置 NO 是我们分析的输出之一。此外，每个三重 no∈NO，我们需要确定一组相关的动作。关系操作⊆NO×BA 捕获这些信息： no 动作 n 显示由节点创建的动作 n （这是一个 buildaction 网站）位于动作列表中 no。构建节点的三条规则 bn 代表这种关联。

首先，扩展器的任何操作都被复制到通知中。

$$\frac{no = (bn, nb, we) \in \mathrm{NO} \quad we \text{ weactions } ac}{no \text{ actions } ac}$$

其次，添加构建器的默认操作。

$$\frac{no = (bn, nb, we) \in \mathrm{NO} \quad nb \text{ default } ac}{no \text{ actions } ac}$$

最后，如果没有来自扩展器的动作，则添加构建器的动作。

$$\frac{\begin{array}{c} no = (bn, nb, we) \in \mathrm{NO} \\ nb \text{ nbactions } ac \quad \not\ni we \text{ weactions } ac' \end{array}}{no \text{ actions } ac}$$

除了通知及其操作之外，分析输出还有三倍 no∈NO 流向哪个呼叫通知。对于任何这样的 no =（bn; ... :)，如果 bn 流动 n 其中 n 是通知的呼叫，则通过分析报告该对（no; n)。

4. 动作和意图

对于一个节点 n∈BA 对应 x=buildaction(y)，传入边缘 y→n 代表未决意图的流向。外向边缘 n→x 传播动作的静态抽象（即节点 n）变量 x。动作与未决意图之间的关联由关系待定表示⊆BA×BP。推理规则如预期：pi flowsto n 暗示 n 有待 pi。的建模 x=buildpending(y) 是类似的：它更新关系意图 ⊆BP×IN 它将待决意图与潜在的真实意图联系起来。

5. 嵌套页面

由 x = addpage（y; z）创建的嵌套页面建模与动作建模类似。 关系页面⊆WE×BN 记录哪些通知添加到 addpage 节点 n 的哪些扩展器

请注意，在 addpage 中使用的通知的操作不会影响与我们一起构建的其他通知。 因此，我们仅使用其构建站点 bn ∈ BN 来抽象嵌套通知，并且不对 bn 中使

用的特定构建器/扩展器建模。

在构建的调用中，扩展器的页面列表被复制到新的通知中。

$$\frac{no = (bn, nb, we) \in \mathrm{NO} \qquad we \text{ pages } bn'}{no \text{ pages } bn'}$$

这里的页面被扩展为包含一个子集 NO×BN。

6. 分析算法

计算约束系统的解决方案分几个阶段完成。首先，约束图是从程序表示中构建的。接下来，转发可达性 n∈NB∪WE∪BA 使用 addaction，setaction 和 extend 节点来计算关系 weactions，nbactions 和 extends。然后，根据通知构建者到构建节点的可达性以及关系动作来确定 NOno∈NO 被计算。最后，检查从构建到通知节点的可达性。addpages，buildaction 和 buildpending 的处理以类似的方式完成。通过我们之前的工作[11]的意向分析来分析到达增量节点的 Intent 站点，以确定它们的目标。

分析输出

四类信息由静态分析产生。输出的第 1 部分是静态抽象的 NO 集合，它表示在可穿戴扩展器的帮助下通过构建调用创建的运行时通知对象。每(bn,nb,we)∈ NO 是程序语句的三倍：构建调用站点 bn，一个新的表达 nb 创建通知构建器和新表达式 we 为可穿戴的扩展器。在这个例子中，NO 包含 no1=(build19,Builder1,WearableExtender4)。尽管这里的构建站点只有一个可能的构建器/展开招标，但我们已经看到了一些真实代码中的示例，其中几个构建器或扩展器可以实现相同的构建。

输出的第 2 部分为每个描述 no∈NO，通知它到达的 call。这可以用来确定这些控制流出口点的行为。例如，no1 达到通知 20。在输出的第 3 部分中，对每一部分 no 有关于在可穿戴设备上可能触发的屏幕的信息。任何一双 no 动作 ac 和 no pages bn 对应于一个屏幕。在我们的例子中 no1 动作 buildaction i 对于 i∈{10,14,16} 和 no1 页面 build6。

操作将控制流转回掌上应用程序。在输出的第 4 部分，每个 no 与定义这些重入点的 Intents 的新站点相关联。结合众所周知的意图分析技术(例如,[4],[5])，

这可以消除通知呼叫时的控制流。对于任何组合 no 动作 ac,ac 有待 pi，和 pi 意图 in，调用 no 的通知可以与意图的目标 in 匹配来进行进一步分析。在这个运行中的例子里，对于 (i,j)∈{(10,9),(14,13),(16,3)}，我们有取决于 buildpendingj 的 buildActioni。对于 (j,k)∈{(9,8),(13,12),(3,2)}，y 有 buildPendingj 意图 Intentk。因此，对于这三个动作中的每一个，可以通过考虑来自 {Intent2， Intent8， Intent12} 及其目标（即， MainActivity，RemoteMessagingReceiver 或 MarkReadReceiver）的相应意图来确定 notify20 的控制流重入点。

V．测试工具

分析可能会被不同的客户使用。我们在由我们开发的测试工具的上下文中说明了这种用法。该工具的结构如图 5 所示。

A.测试框架

由于 Android 平台在两个设备上处理通知，Robotium [19]和 Espresso [20]等框架不能用于编写 AW 测试用例。我们开发并制作了 AW UIAutomator Server [21]，这是 AW 应用程序的测试框架。该实现为其他人开发的现有 UIAutomator 服务器添加了 AW 特定的功能[22]。我们的 AW UIAutomator 服务器在每个设备上创建一个 JSON-RPC 服务器来监听传入事件。开发人员可以编写简单的 Python 脚本来发送诸如滑动和点击等事件，在输入字段中输入文本以及模拟发送 SMS。该方法适用于仿真和真实可穿戴设备。

该框架包含一个带有 Android 模拟器的 GUI 小部件层次结构爬行器的库。给定一个已经显示在模拟器中的通知，库与 GUI 小部件服务器通信以在可穿戴屏幕上记录当前小部件（包括字符串标题），然后将这些信息解析为抽象对象。对于真实的设备，由于默认情况下禁用了 GUI 小部件服务器，因此我们无法使用此爬网程序。该库包含一个基于 pytesseract OCR 工具的替代爬虫[23]。抓取工具通过 OCR 抓取屏幕截图并识别字符，以便构建可穿戴设备屏幕的表示形式。如下所述，我们使用这些功能来标识通知，页面和操作的静态 ID，以检查覆盖范围。

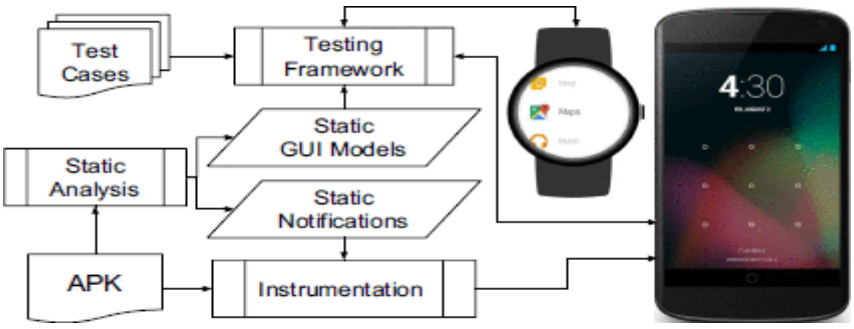

图 5.测试工具概述

B.覆盖标准

考虑一组使用我们的框架编写和执行的测试用例。一个有趣的问题是这个集合是否全面地执行通知相关的运行时行为。此行为由 Android 平台代码跨两个不同的 JVM 执行。诸如手持应用程序代码的声明或分支机构覆盖范围等传统覆盖不足以确保运行时行为中的可能变化得以实施。例如，在 QKSMS 中，创建和更新对话都会触发通知。这两个操作的通知构建器都可以访问包含多个分支的函数 multipleSenders，并且可以传递扩展，构建和通知。声明/分支覆盖无法确保在运行时涵盖所有可能的行为。我们提出以下 AW 特定的覆盖目标，

1. 通知站点

回想一下，分析计算了一组三元组 no=(bn,nb,we)代表通知对象。每个 no，分析确定哪些呼叫通知已到达 no。让 n 表示这样的呼叫站点。我们为每个定义通知网站的覆盖范围如下：n 和 no 达到它，执行至少一个调用的测试用例 n 与从中建立的通知 nb，通过扩展 we，并建于 bn。

该标准涵盖了通知的所有静态抽象以及发布它们的每个可能的通知呼叫站点。我们已经看到多个构建器和扩展器流向单个构建站点的应用程序，并且单个构建器流向多个构建站点。所有这些场景都被这个定义所捕获。对于图 3 中的示例，测试用例应该覆盖（build19，Builder1，WearableExtender4，notify20）。

2. 嵌套页面

嵌套页面可选地用于通知以显示补充信息。运行示例演示了这种情况：仅当图 1 中第 7 行的条件为真时才添加聊天页面。为了行使与这些页面相关的运行时行

为，我们定义下面的嵌套页面覆盖标准：对于每个页面 no 网页 bn，这样 no 到达通知站点 n，执行至少一个测试用例 no 由...颁发 n 以及由构建站点创建的页面 bn 作为运行时通知的一部分显示在可穿戴设备上。图 2 显示了这样一个执行：当由 build19 创建的通知达到 notify20，然后在可穿戴设备上发生“向左滑动”事件时，在 build6 上创建的嵌套页面将显示在可穿戴设备上。

3. 操作

我们还考虑行动的覆盖范围：每个 no 行动 ac 并通知每个网站 n 到达 no，至少会触发一个测试用例 n 发行 no 并在运行时通知中表示一个动作 ac 显示在可穿戴设备上。另外，每个可能的静态目标都是 ac 应该通过点击可穿戴设备上的操作按钮重新进入手持设备。对于运行示例，需要三个测试用例，每个操作一个（“Block 应用程序”没有任何兴趣）。他们应该在手持设备上触发相应的意图目标，即执行“回复”操作并输入 RemoteMessagingReceiver，执行“标记为已读”操作并输入 MarkReadReceiver，然后执行默认的“在手机上打开”操作并输入 MainActivity。

C.测量测试覆盖率

1. 静态分析

分析会生成静态通知列表，每个静态通知都由构建调用站点，通知调用站点以及通知构建器和可穿戴扩展器的新表达式的网站定义。这些信息可在分析输出的第 2 部分中找到（第 IV-C 节）。我们使用基于相应的 Soot 语句，周围方法的签名和语句行号的哈希代码为每个站点分配一个整数 ID。四个站点的 ID 用于计算通知的 ID。这个 ID 然后被我们的测试框架用来识别可穿戴设备的屏幕上的 GUI 小部件，并计算覆盖范围，如稍后所述。我们还根据其构建活动站点为每个操作分配一个整数 ID。

分析输出的第 3 部分为每个静态通知定义了一个 GUI 模型。这个模型简单地说就是关系动作和页面所定义的通知动作和嵌套页面的静态抽象集合。

2. 仪表

仪器组件将 APK 文件和静态分析的输出作为输入。对于通知构建器和可穿戴

扩展器的每个新表达式，检测记录该站点的整数 ID 并将其与运行时对象相关联。我们还记录要构建的呼叫的 ID 并将其与其创建的通知相关联。在测试期间，在每次通知通知之前,检测工具将检查运行时通知的三个站点的 ID 以及通知站点的 ID。如果它们与静态通知的网站相匹配，则我们记录测试涵盖了这部分通知网站标准。我们还检查通知的嵌套页面。如果网页的网站与静态通知中的页面匹配，我们会将静态 ID 添加到页面的标题中。例如，聊天页面的标题将从“测试帐户”更改为“1859080457 测试帐户”。如果我们可以在测试执行期间在可穿戴设备的屏幕上的页面的字符串标题中观察到此 ID，我们会记录嵌套页面的覆盖范围。

为了识别一个动作，检测插入其静态 ID 作为其标题的前缀。我们还将动作 ID 作为额外字符串添加到其目标 Intent 中，并在手持设备上检测相应静态目标的入口点。如果一个目标是一个活动，我们使用 onCreate 方法。如果目标是广播接收器，则入口点是其 onReceive 方法。如果目标是一个 intent 服务，入口点是 onHandleIntent; 对于正常服务，该条目是 onBind 或 onStartCommand。如果我们可以在发布通知时从可穿戴设备的屏幕上的标题中检索到动作 ID，并且它与执行动作后我们从掌上电脑的意向中获得的 ID（onCreate 中提供的 Intentis 等）相匹配，我们会记录动作范围。通过标准的 API）。

D.自动生成可穿戴 GUI 事件

使用测试框架，工具用户可以编写一个测试用例，其中包含手持设备和可穿戴设备上的 GUI 事件组合，例如触发手持应用程序中的通知站点，在可穿戴设备上发出滑动事件以获得特定操作，然后单击操作按钮。如果目标是编写很多这样的测试用例来实现可穿戴设备上的 GUI 结构的高覆盖率，那么可以使该过程的一部分自动化。假设一个工具用户写一个测试用例 Tinitial（用于手持应用程序），它在通知站点发布通知对象。可以自动触发此通知的 GUI 模型的元素（即其嵌套的页面和操作）。对于每个嵌套页面，我们可以附加到 Tinitial 一系列滑动事件在到达该嵌套页面时停止。同样,对于 GUI 模型中的任何操作,我们都可以附加到 Tinitial 滑动事件以触发该操作，然后触发点击事件。GUI 模型不表示动作/页面的排序，并且达到特定动作/页面所需的滑动事件的数量不是静态的。因此，在每次滑动事件之后，运行时检查（类似于用于覆盖范围跟踪的检查）确定是否达到目标操作/

页面。这种自动化允许自动生成和执行几个增强的测试用例，从一个开始 Tinitial 由工具用户编写。

VI. 实验评估

A.研究对象

我们评估了 F-Droid 提出的八个开源 AW 应用的静态分析[24]。他们之所以被选中，是因为他们是唯一的 F-Droid 应用程序，在其反编译的代码中具有字符串“WearableExtender”，并允许在实际的 AW 智能手表上进行安装。我们编写测试用例来实现上一节中介绍的标准的高覆盖率。然后，我们将结果运行时通知与我们分析报告的静态通知进行比较。

除了这些开放源代码应用程序之外，我们还希望证明对闭源应用程序的适用性。只有 APK 可用于此类应用程序。在没有源代码的情况下，触发必要的运行时间条件以实现高覆盖率并推断静态解决方案的可行性是非常具有挑战性的。对于 4 个应用程序，我们能够获得足够的理解，以便能够编写有意义的测试案例，并对解决方案的可行性做出高度可信的判断。

研究对象的特征见表 I ； 表格底部列出了封闭源代码应用程序。类的数量显示在“类”列中。这包括除 android.support 库之外的 APK 中的所有类。Jimple 是 Soot 的中间代表； 该表显示了此 IR 中的语句数量。对于开源应用程序，我们手动识别并过滤掉所有第三方库，然后计算应用程序类，方法和 Jimple 语句的数量。这些数字显示在相应列中的括号中。我们的分析不会将应用程序类别与第三方库类别区分开来。

Application	Classes	Methods	Jimple Stmts	Time (sec)	notify calls (AW/All)	build calls (AW/All)	extend calls	$\|NO\|$	(no, n)	(no, n, bn)	(no, n, ac, t)
QuickLyric	1139 (100)	7584 (471)	121772 (8876)	12.24	2/5	2/5	2	2	2	0	2
WhatsappBetaUpdater	387 (36)	1891 (146)	29832 (1930)	2.41	1/1	1/1	1	1	1	0	1
QKSMS	1592 (672)	9193 (4360)	140234 (73896)	11.67	1/1	3/5	3	7	6	6	18
Loop	555 (201)	5070 (1373)	72796 (22808)	6.38	1/1	1/1	1	1	1	0	3
Silence	4898 (948)	33860 (5782)	523060 (74000)	68.74	2/6	2/9	3	3	3	0	7
Tasks	1357 (956)	6249 (4409)	83602 (55223)	7.67	1/1	1/4	1	1	1	0	3
Telegram	4363 (3778)	23405 (19227)	510145 (447549)	28.42	1/4	1/6	2	1	1	0	2
org.toulibre.cdl	172 (146)	896 (753)	10582 (8509)	1.04	1/1	1/1	1	1	1	0	2
ArcusWeather	6361	36805	485714	92.19	1/5	2/8	1	2	1	3	1
GroupMe	4699	26937	362634	40.96	1/3	2/8	2	2	1	1	5
Slack	5697	34867	418256	152.06	3/12	5/14	3	5	3	2	3
Signal	5987	41008	588386	68.91	2/9	2/12	3	3	3	0	7

表 I 研究对象的特征

列“时间”显示静态分析的运行时间。平均而言，分析的成本大约为每 10K Jimple 语句 1.5 秒，配备 3.40GHz CPU 和 16GB 内存的 PC。第 6 和第 7 列显示了所有通知和构建调用的数量以及那些至少有一个用于通知和构建器的可穿戴扩展器的数量。总共有 49 个通知呼叫和 74 个呼叫，其中 17 个和 23 个分别是可穿戴的。分析所有这些呼叫以确定哪些子集适用于可穿戴设备，哪些适用于手持设备。第 8 列显示了扩展调用的数量。

列“| NO |”显示了设置 NO 的大小，其中包含通知的静态抽象。最后三列与部分 VB 中定义的覆盖标准相对应。列“(no,n)“对应于通知站点标准。这里 n 是通过呼叫通知 no∈NO。在某些情况下（例如，QKSMS），这些对的数量小于 NO 的大小，因为某些通知用作嵌套页面而不是通知的参数。列“（no,n,bn）“对应于嵌套页面标准。现场 bn 是创建一个添加到嵌套页面的构建调用 no 通过一些可穿戴的扩展器。列“(no,n,ac,t)“对应于行动报道。这里 ac 是一个行动 no 和 t 是由此操作触发的手持应用重新入口点。少量的嵌套页面意味着 AW 通知的 GUI 结构的简单性。这是因为基于微相互作用的设计原则建议“尽量减少细节卡的数量”[25]。

B.案例研究

对于每个应用程序，我们编写测试用例，试图根据之前定义的标准实现完全覆盖。应用程序的源代码在可用时进行了检查，以确保我们确实实现了最大可能的覆盖范围。（1）验证我们的测试工具的工作，以及（2）评估静态分析的精度，因为任何无法实现的覆盖目标都表明分析不准确。

这些案例研究的结果如表 2 所示。一般而言，覆盖率非常高，表明静态分析解决方案在运行时通常是可行的。对于 12 个应用程序中的 9 个，观察到了完美的分析精度。这些研究的其他观察结果如下。

Application	notification site coverage	nested page coverage	action coverage
QuickLyric	2/2	0	2/2
WhatsappBetaUpdater	1/1	0	1/1
QKSMS	5/6	5/6	15/18
Loop	1/1	0	3/3
Silence	3/3	0	7/7
Tasks	1/1	0	3/3
Telegram	1/1	0	2/2
org.toulibre.cdl	1/1	0	2/2
ArcusWeather	1/1	3/3	1/1
GroupMe	1/1	1/1	3/5
Slack	1/3	0/2	1/3
Signal	3/3	0	7/7

表 II 实现的运行时覆盖

1. QKSMS

每当有消息到达时，这个应用程序就会在可穿戴设备上发出通知。我们无法触发六个静态通知中的一个。这种情况发生在用户从多个发件人收到多封邮件时。这种情况的处理逻辑很复杂，并且据我们所知，发布通知的代码是死代码

2. 电报

这是一个流行的聊天应用程序，Google Play 商店的下载量接近 200 万次。烟尘无法为其生成有效的检测 APK 文件。因此，我们手动检测代码。该应用程序需要两个手持设备进行测试：一个用于发送消息，另一个用于接收消息并将通知桥接到可穿戴设备。我们利用我们的测试框架同时管理三台设备，并实现了全面覆盖。

3. GroupMe

这是一个群聊和分享的应用程序。它还需要两个手持设备进行测试。有 5 个元组(no,n,ac,t)静态分析报告，但其中只有三个是可行的。不可行性的原因是超类 BaseNotification 包含用于构建和发布通知（无论是在掌上电脑还是可穿戴设备上）的代码，并且只有其中一个子类与可穿戴通知相关。虚假的目标 t 来自 BaseNotification 的其他子类。有标准的技术来处理这种不精确性（例如，对象敏感性[9]，[26]），它们可以很容易地与我们的方法相结合。

4. 松弛

此业务应用程序用于团队沟通，文件共享，归档，云集成等 (no,n)，只有一个是可行的。另外两个覆盖标准也受到这种不精确性的影响。我们确定这两个不可

行的通知是由运行时永远不能执行的代码发出的。这个死代码可以通过过程间常量传播分析发现。

C.自动生成 GUI 事件

如 VD 部分所述如果工具用户编写发布特定通知的测试用例，我们的方法可以通过在可穿戴设备上添加滑动事件和点击事件来生成多个增强的测试用例。一个问题是这种方法与随机测试的比较。为了获得关于这种比较的见解，我们使用 ArcusWeather 进行了案例研究。此应用程序中的通知具有相对复杂的 GUI 结构，包含三个嵌套页面和一个操作。使用静态 GUI 模型来指导探索，我们的方法会生成四个增强的测试用例，每个页面和动作都有一个。覆盖第一个嵌套页面的测试用例需要两个事件：向上滑动以激活通知，向左滑动以触及页面。为了覆盖第二页，测试用例需要三个事件。需要四个事件来覆盖最后一页。

可以尝试用随机测试来实现相同的覆盖率。为了探索这种可能性，我们使用 Google 的 Monkey 工具[27]进行随机测试。我们将 Monkey 配置为探索包 com.google.android.wearable.app 和类别 android.intent.category.HOME，以避免打开不相关的应用程序并触发无用的事件。由于目标事件是轻扫和点击，我们还将该工具配置为仅执行动作和触摸来模拟这两个操作。触发随机事件最常见的情况是打开应用程序列表或导航栏并开始探索它们。在这种情况下，实际的通知从屏幕丢失。附加事件不太可能会导致通知并达到期望的页面或操作。我们认为这是一个“卡住”状态。在我们的研究中，我们每 30 个事件检查一次卡住状态，并在达到这种状态后重新启动猴子。此过程会重新启动多次，始终从原始用户定义的测试用例开始触发通知。成功运行后即停止执行，即达到/触发期望的页面/操作时。我们记录所有不成功运行和最终成功运行中的事件总数。由于随机性，我们执行此过程的 10 次独立执行并测量事件的平均数量。

为了覆盖第一个嵌套页面，Monkey 需要 156 个事件。为了达到第二页和第三页，猴子分别需要 204 和 280 个事件。为了触发这个动作，猴子需要 291 个事件。大量事件有两个原因。首先，随机测试很可能会打开应用程序列表并调用其他默认应用程序，而不是探索通知的 GUI 元素。其次，随机生成的事件会产生弹跳效果，产生许多无用的滑动效果，例如，从第 1 页的第 2 页着陆，然后返回第 1 页。这突

出显示了基于静态分析模型生成 GUI 事件的好处，而不是随机生成 GUI 事件。

VII. 相关工作

对于 Android 的静态分析和测试，有大量工作（例如[4]，[5]，[10] - [11] [12]，[14]，[28] - [29] [30] [31] [32] [33] [34] [35] [36]），但 Android Wear 的工作量很小。

Android Wear

Min 等人 [37]对智能手表的电池使用进行了探索性调查，并强调“智能手机通知检查”是智能手表最常见的用法。Chauhan 等人。[38]表征 AW 和其他可穿戴 OS 的应用程序的各种属性（例如，域类别，外部跟踪，信息泄露）。Liu 和 Lin [39]检查 AW 设备的 CPU 使用情况，空闲情节和线程级并行性。它们提供了 AW 平台中执行效率低下和设计缺陷的证据。其他研究人员已经考虑过在诸如医疗[40]，文本识别[41]和移动生物识别[42]等领域使用 AW 设备，。在 AW 通知机制建模和在测试工具中使用这种建模方面没有任何工作，这是我们工作的目标。

简单 Android 的测试和 GUI 探索

Choudhary 等人。[43]总结了许多针对 Android 应用程序的现有测试和 GUI 探索方法。Dynodroid [31]使用指导性的随机 GUI 探索。GUIRipper [32]生成一个动态构建的 GUI 模型。MobiGUITAR [33]使用 GUIRipper 的增强版本，并将测试充分性标准应用于其中以生成测试用例。A 3 E [30]使用基于静态分析的控制流模型的 GUI 探索。PUMA [34]是一个框架，它分离了探索应用程序执行的逻辑和分析应用程序属性的逻辑。ACTEve [35]是一个 concolic 测试工具，它象征性地追踪他们这一代到他们处理的事件。这些工具都不是为 AW 应用程序设计的，它们不能直接用于 AW 通知的分析和测试覆盖。

Ⅷ.结论和未来工作

预计未来十年，可穿戴设备的普及将急剧增加。这种增长给软件工程研究人员

带来了有趣的挑 我们的工作专注于 Android Wear 及其核心交互机制之一：通知导致的控制流。我们抽象出机制的基本概念，并定义一个分析来静态模拟它们。由此产生的信息为进一步的客户分析提供了一个起点。我们的评估表明，分析具有实际成本和高精度。

AW 1.x 和更复杂的 AW 2.0 都有很多这方面的问题。预计两款 APK（一款在手持设备上，一款在可穿戴设备上）以及带有可穿戴 APK 的应用程序都将越来越受欢迎。有趣问题的例子包括可穿戴设备和手持设备之间的数据同步，可穿戴设备上的自定义用户界面，减少电池消耗的技术，安全分析以及对 AW 演进的支持。

致谢

我们感谢 ICSE 评论员的宝贵意见。本材料基于美国国家科学基金会的支持下的奖项 1319695/1526459 和谷歌学院研究奖。

REFERENCES

[1] IDC Research Inc., “Worldwide smartwatch market will see modest growth in 2016 before swelling to 50 million units in 2020,” http://www.idc.com/getdoc.jsp?containerId=prUS41736916, Sep. 2016.

[2] “Android Wear,” https://developer.android.com/wear.

[3] “Building apps for wearables,” https://developer.android.com/training/building-wearables.html.

[4] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. le Traon, “Effective inter-component communication mapping in Android with Epicc,” in USENIX Security, 2013.

[5] D. Octeau, D. Luchaup, M. Dering, S. Jha, and P. McDaniel, “Composite constant propagation: Application to Android intercomponent communication analysis,” in ICSE, 2015, pp. 77–88.

[6] “SCanDroid: Security Certifier for anDroid,” http://spruce.cs.ucr.edu/SCanDroid.

[7]“PendingIntent,”https://developer.android.com/reference/android/app/PendingIntent.html.

[8] F. Nielson, H. R. Nielson, and C. Hankin, Principles of Program Analysis. Springer, 2005.

[9] Y. Smaragdakis, M. Bravenboer, and O. Lhoták, “Pick your contexts well: Understanding object-

sensitivity," in POPL, 2011, pp. 17–30.

[10] A. Rountev and D. Yan, "Static reference analysis for GUI objects in Android software," in CGO, 2014, pp. 143–153.

[11] S. Yang, D. Yan, H. Wu, Y. Wang, and A. Rountev, "Static controlflow analysis of user-driven callbacks in Android applications," in ICSE,2015, pp. 89–99.

[12] S. Yang, H. Zhang, H. Wu, Y. Wang, D. Yan, and A. Rountev, "Static window transition graphs for Android," in ASE, 2015, pp. 658–668.

[13] A. Igarashi, B. C. Pierce, and P. Wadler, "Featherweight Java: A minimal core calculus for Java and GJ," TOPLAS, vol. 23, no. 3, pp. 396–450,May 2001.

[14] D. Yan, "Program analyses for understanding the behavior and performance of traditional and mobile object-oriented software," Ph.D.dissertation, Ohio State University, Jul. 2014.

[15] B. G. Ryder, "Dimensions of precision in reference analysis of objectoriented programming languages," in CC, 2003, pp. 126–137.

[16] O. Lhoták and L. Hendren, "Scaling Java points-to analysis using Spark,"in CC, 2003, pp. 153–169.

[17] M. Sridharan and R. Bodik, "Refinement-based context-sensitive pointsto analysis for Java," in PLDI, 2006, pp. 387–400.

[18] R. Vallée-Rai, E. Gagnon, L. Hendren, P. Lam, P. Pominville, and V. Sundaresan, "Optimizing Java bytecode using the Soot framework: Is it feasible?" in CC, 2000, pp. 18–34.

[19] "Robotium: User scenario testing for Android," https://github.com/robotiumtech/robotium.

[20] "Testing UI for a single app," https://developer.android.com/training/testing/ui-testing/espresso-testing.html.

[21] "Android Wear (AW) UIAutomator server," https://github.com/presto-osu/aw-uiautomator-server.

[22] "Android UIAutomator server," https://github.com/xiaocong/android-uiautomator-server.

[23] "Python-tesseract: A Python wrapper for Google's Tesseract OCR, " https://pypi.python.org/pypi/pytesseract.

[24] "F-Droid application market," https://f-droid.org.

[25] "UI patterns for Android Wear," https://developer.android.com/design/wear/patterns.html.

[26] A. Milanova, A. Rountev, and B. G. Ryder, "Parameterized object sensitivity for points-to

analysis for Java," TOSEM, vol. 14, no. 1, pp.1–41, 2005.

[27] "UI/Application exerciser Monkey," https://developer.android.com/tools/help/monkey.html.

[28] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein,Y. Le Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise context,flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," in PLDI, 2014, pp. 259–269.

[29] W. Yang, M. Prasad, and T. Xie, "A grey-box approach for automated GUI-model generation of mobile applications," in FASE, 2013, pp. 250–265.

[30] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of Android apps," in OOPSLA, 2013, pp. 641–660.

[31] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for Android apps," in FSE, 2013, pp. 224–234.

[32] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. Memon, "Using GUI ripping for automated testing of Android applications," in ASE, 2012, pp. 258–261.

[33] D. Amalfitano, A. Fasolino, P. Tramontana, B. Ta, and A. Memon, "MobiGUITAR: Automated model-based testing of mobile apps," IEEE Software, pp. 53–59, 2015.

[34] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "PUMA:Programmable UI-automation for large-scale dynamic analysis of mobile apps," in MobiSys, 2014, pp. 204–217.

[35] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in FSE, 2012, pp. 1–11.

[36] H. Zhang, H. Wu, and A. Rountev, "Automated test generation for detection of leaks in Android applications," in AST, 2016, pp. 64–70.

[37] C. Min, S. Kang, C. Yoo, J. Cha, S. Choi, Y. Oh, and J. Song, "Exploring current practices for battery use and management of smartwatches," in ACM Int. Symp. Wearable Computers, 2015, pp. 11–18.

[38] J. Chauhan, S. Seneviratne, M. A. Kaafar, A. Mahanti, and A. Seneviratne, "Characterization of early smartwatch apps," in Workshop on Sensing Systems and Applications Using Wrist Worn Smart Devices,2016, pp. 1–6.

[39] R. Liu and F. X. Lin, "Understanding the characteristics of Android Wear OS," in MobiSys, 2016, pp. 151–164.

[40] H. Dubey, J. C. Goldberg, M. Abtahi, L. Mahler, and K. Mankodiya, "EchoWear:Smartwatch

technology for voice and speech treatments of patients with Parkinson's disease," in Proceedings of the Conference on Wireless Health, 2015, p. 15.

[41] L. Arduser, P. Bissig, P. Brandes, and R. Wattenhofer, "Recognizing text using motion data from a smartwatch," in Workshop on Sensing Systems and Applications Using Wrist Worn Smart Devices, 2016, pp. 1–6.

[42] A. H. Johnston and G. M. Weiss, "Smartwatch-based biometric gait recognition," in Int. Conf. Biometrics Theory, Applications and Systems, 2015, pp. 1–6.

[43] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for Android: Are we there yet?" in ASE, 2015, pp. 429–440.