基于Smali Code的移动应用行为模型的自动构建方法

余 勇 郭 骞
(全球能源互联网研究院信息通信研究所 南京 210003) (信息网络安全国网重点实验室 南京 210003)

摘 要 移动应用数量的快速增长,以及移动应用开发周期短、迭代速度快等特点,使得移动应用的测试面临挑战,许多应用没有经过充分测试就被投放到市场,其中可能包含程序缺陷,从而影响用户体验。模型驱动的测试方法是最有效的测试方法之一,在功能、性能、可用性、安全等测试方面均有应用,能在一定程度上提高测试的自动化程度。移动应用领域与传统研究领域的模型驱动测试方法的最大区别在于模型构建方法的不同,因为移动应用是事件驱动的。提出了一种构建移动应用行为状态机模型的方法。首先通过逆向工程的方法得到移动应用的中间码;然后在中间码上通过动静态结合的方法生成事件表;最后,建模时通过在系统层扫描屏幕变化来判断是否出现新状态,并定义所有出现的状态,从而组成最终的模型。该方法一方面避免了源代码的限制,另一方面也提高了模型的覆盖度。实验结果表明,所提方法可以准确有效地构建移动应用的行为状态机模型,解决现有移动应用测试中模型构建存在的部分问题。

关键词 移动应用测试,模型驱动测试,模型构建,移动应用行为模型,逆向工程,状态机
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.032

Behavioral Model Construction Method for Mobile Applications Based on Smali Code

YU Yong GUO Qian
(Information Communication Research Institute, Global Energy Interconnection Research Institute, Nanjing 210003, China)
(State Grid Key Laboratory of Information & Network Security, Nanjing 210003, China)

Abstract With the rapid growth in the number of mobile applications and the speed requirement of mobile application development, mobile application testing faces big challenge. In this case, many applications have been put into the market without adequate testing, which may contain bugs and impact the user's experience. Model-driven testing method is one of the most effective testing methods, which is widely used in function, performance, availability, security testing, and can greatly improve the automation of testing. The biggest difference between the model-driven testing methods for mobile applications and traditional applications is model building method, because mobile applications are event-driven. In this paper, we proposed a method to build mobile application behavior state machine model. First, the intermediate code of the application is obtained through reverse engineering method. Then intermediate code is used to generate event table by static and dynamic methods. At last, the model is built by scanning the screen in the system layer to discover new states and define all states appeared. Experimental results show that our method can effectively and accurately build mobile application behavior model and solve some of the issues which exist in the mobile application model building.

Keywords Mobile application testing, Model-driven testing, Model building, Mobile application behavior model, Reverse engineering, State machine

1 引言

计算机的高速发展正在逐渐改变人类的生活方式,它渗入生活的各个方面,尤其是在移动互联网时代,人类的生活方式正在日新月异的变化。移动互联网处于高速发展期,据CNNIC发布的第37次《中国互联网络发展状况统计报告》[1],截止到2015年12月,中国网民规模达6.88亿,互联网普及率为50.3%,手机网民规模6.20亿,占比提升至90.1%。随着网络环境的逐渐完善,移动互联网应用的需求不断增长,在2015年,网络金融、娱乐、交易等个人应用发展迅速,尤其是支付应用,到2015年12月,手机网上支付用户规模已达到3.58亿,增长率为64.5%。由于需求在不断增加,移动应用的开发周期通常较短且迭代较快,测试人员的测试时间很少,同时移动测试框架和工具都十分有限,自动化程度不高,以致于很多应用没有经过充分的测试就被投放到市场。不同于传统的软件,移动平台应用具有丰富的图形用户界面,增加了应

到稿日期:2016-10-07 返修日期:2016-12-25 本文受面向电力移动终端的应用测试技术研究(5455HT150029)资助。
余 勇(1970-),男,高级工程师,主要研究方向为电力行业的信息安全技术,E-mail:yuyong@geiri.sgcc.com.cn;郭 骞(1983-),男,高级工程师,主要研究方向为电力行业的信息安全技术,E-mail:guoqian@geiri.sgcc.com.cn。

用的复杂性。移动应用程序属于事件驱动图形界面程序，具有一些新的特征，比如规模小、事件驱动、直观的 GUI 等，需要特别的自动化测试技术[2]。

在 Web 和 Java 等 PC 软件领域，很多学者研究了模型驱动的测试方法。模型驱动测试方法针对现实问题构建出抽象的验证模型，系统的结构和行为可以通过有限状态机、UML 模型、Petri 网等建模，系统属性可以用时序逻辑公式(CTL，LTL 等)来描述。通过维护测试模型，不再需要维护大量的脚本；另外，基于模型能自动生成测试用例，这是实现测试自动化的基础。在移动应用的测试领域，模型驱动的测试方法受到广泛关注。模型驱动测试方法研究的是通过构建模型来生成测试用例，该方法在 C，C++，Java 语言上的成熟应用为移动应用测试提供了很多经验。然而，由于移动应用程序主要是事件驱动的，而非传统的路径敏感，符号执行等研究方法还面临着较大压力，因此模型测试的优势非常明显。移动应用的模型驱动特征主要体现在模型构建上，一方面，移动应用的事件类型多，例如触控、手势、传感器等，在提取建模需要的事件时比较困难；另一方面，状态的定义较为困难，很多研究通过手工建立状态机模型，依赖测试工程师找出所有的事件并定义状态机的状态。基于模型的测试方法是移动应用领域最有效的测试方法之一，然而现有工作面临的棘手问题是如何有效地构建模型。

模型构建成为了移动应用测试的关键环节。在模型构建的研究领域中，状态机是使用最广泛的模型。该模型要求设计者考虑所有可能的状态，以及在所有可能输入条件下可能进行的所有状态转移。状态机模型适用于以控制为主的系统，在移动应用测试中，移动应用的行为模型是基于事件的控制类型系统。建立移动应用行为模型的难点在于覆盖度和成本之间的权衡，降低成本通常会降低模型覆盖度，而较高的覆盖度通常意味着较高的建模成本。

本文研究面向移动应用测试的模型构建方法，以 Android 平台作为研究方法的实现载体，提出了移动应用行为状态机模型的构建方法。该方法首先通过逆向工程得到应用可执行文件的中间码 Smali Code；然后，在中间码的基础上构建状态机模型，从 Intent 入手，通过动静态结合的方法生成事件表，即所有 Activity 以及它们之间的关联关系；在建模时通过在系统层扫描屏幕变化来判断是否出现新的状态，自动地生成所有的状态向量，无需人工定义，将状态和事件关联起来即可得到状态机模型。该方法可以有效地提高覆盖度，并摆脱建模对源代码的依赖。

本文第 2 节主要介绍技术背景；第 3 节详细阐述移动应用行为状态机模型的生成方法；第 4 节介绍原型工具和相关的实例研究；第 5 节为相关工作；最后是方法的总结与对未来工作展望。

2 背景知识介绍

本节对相关背景和技术进行介绍。首先介绍基于模型的测试方法的应用和发展状况；然后阐述 Android 系统架构和核心功能中与本文方法直接相关的部分；最后概述 Android 系统的视图结构和事件驱动特征。

2.1 基于模型的测试方法

在本文的方法中，模型主要指状态机模型，状态机模型是一种描述系统对内部或外部事件响应的行为模型。它关注系统状态和事件，事件引发系统在状态间的转换，而不是系统中数据的流动。状态机模型可以解释为：在任意时刻，系统处于有限个可能的状态中的一个状态中，当某一个激励到达时，它将激发系统转移到一个新的状态。有限状态机包含以下几个重要的基本概念。

1)状态(State)：状态机处在其生命周期的一种状态，处于某个特定状态中的对象必然会满足某些条件、执行某型动作或者等待某些事件。

2)事件(Event)：在时间和空间上占有一定位置，并且于对状态机而言是有意义的事情。事件会引起状态迁移，促使状态机从一个状态切换到另一个状态。

3)转换(Transition)：两个状态之间的变化关系，对象在前一个状态上执行了某个动作并在事件发生的同时满足特定条件就会迁移到下一个状态。

4)动作(Action)：状态机中可以执行的那些原子操作。原子操作指的是它们在运行过程中不能被其他消息中断，必须一直执行下去。

在软件工程的相关领域中，基于模型的测试方法一直都发挥着重要作用[3]。基于模型的测试方法一般包括 5 个步骤，具体流程如下(见图 1[4])：1)建立待测试程序的模型；2)从模型上生成测试用例；3)抽象测试用例的实例化；4)在待测试程序上执行这些测试用例；5)分析测试结果。该方法对状态机模型的应用分为两个阶段：建立模型和遍历模型，从而减少了手动生成测试用例的工作量。

对于基于模型的测试方法而言，核心工作是如何获得模型的？模型是模拟软件最直观的方法，而模型的质量影响软件模拟的质量，好的模型既要保持测试的特点又不能太复杂，这也是本文工作的重点研究内容。

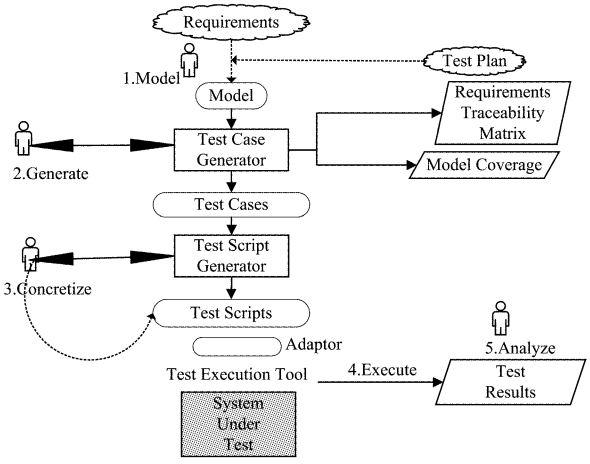


图 1 基于模型的测试方法流程图

由于维护模型的时间与生成测试用例的时间的总和往往比手工设计测试用例的时间少，因此基于模型的测试方法能够有效减少开销。同时，很多基于模型的测试工具能够生成发现缺陷的最短的测试序列，特别是当遇到一个特殊的测试失败情景时，可以通过抽象测试用例来查看执行路径而不需

要阅读源代码,因此在错误分析阶段也可以节省大量时间。基于模型的测试方法生成的测试用例是可追踪的,因而可以知道测试用例是如何产生的。当模型进化时,可以用可追踪性来优化测试执行,例如我们只需要执行受模型改变影响的测试用例。

当然,基于模型的测试方法也存在缺陷,该方法不能保证模型与程序完全相同,即使我们在模型上生成无数的测试用例,多次运行调整,也不能保证模型没有缺陷。另外,基于模型的测试方法只能用来做功能测试,偶尔可以用于压力测试,但效果不明显。模型测试发现错误的能力仍然依赖于测试人员的技术和经验。

大型公司很早就开始使用基于模型的测试方法。IBM 最早开发了模型测试驱动内核,给内核生成 FSM 模型。IBM 使用该方法测试了 Java 垃圾回收器的部分内容,测试内容包括 7000 行代码,这些代码分布在 2700 个基本块中。IBM 使用该方法建立了两个有限状态机模型,一个 FSM 根据 Java 语言的语义建立,另一个 FSM 根据 catch,try 和 finally 的语法建立。通过运行模型上生成的测试用例,IBM 发现了 4 个新的缺陷,代码覆盖了 83%的语句,建模和测试花费的时间只是类似测试系统的一半。IBM 也发现,虽然手动生成模型需要有技术基础,但有编程知识背景的人能够很快地学习完这些方法,IBM 的建模课程只需要 2 天就能学完。

微软的基于模型测试方法工具已经经历了三代。第一代是微软 IE 团队开发的 Test Model Toolkit,TMT 也使用 FSM 模型,支持 6 种测试生成算法,生成的测试用例以 xml 的方式保存,同时有一个驱动程序负责执行这些测试用例。第二代和第三代分别是 ASML 和 SpecExplorer,ASML 采用了抽象状态机理论,SpecExplorer 使用的建模语言是 C＃的扩展。由于 ASML 缺少文档说明,学习曲线太长,最终导致了 SpecExplorer的出现。有几个微软的产品开发团队几乎每天都在使用 SpecExplorer,在相同的时间消耗下,SpecExplorer 发现的错误数量是手动测试的 10 倍。基于模型的测试代码覆盖度也能达到 70%。

2.2 Android 系统

2.2.1 架构与组成

Android 系统架构由 4 个部分组成,最下层是 Linux 内核,Android 是从 Liunx 系统上发展起来的,该层提供最底层的系统服务,例如内存管理、网络通信、进程管理等。第二层是库和运行环境,该层也不直接涉及应用软件,同样是为应用层提供服务,其包含 C/C++库、SQLite 数据库、图形引擎和 Java 核心库等,以及最关键的 Dalvik 虚拟机。第三层是应用程序框架层,该层提供了大量的 API 以供开发者调用,也体现出了 Android 开发区别于 Java 和 C＃等面向对象程序开发的特征。应用程序框架层封装了大量的对象,提供了大量功能以辅助测试人员完成测试任务,本文研究工作通过结合该层来提高测试效率。最上层是应用程序层,所有的应用程序都位于这一层。开发人员可以通过自己开发程序来替换掉原有的内置程序,用户可以与应用层上的程序交互,应用程序层再与下面几层交互完成用户任务。

每一个 Android 应用程序都由组件组成。Android 系统共有四大基本组件:Activity、服务、广播接收器、内容提供商。Activity 是 Android 系统最基本、最重要的组件,几乎所有的应用程序都包含 Activity。Activity 直接负责人机交互,即 Activity 是应用可视化界面的载体,是移动测试中最重要的环节,能够接受用户输入。Activity 共有 4 个基本状态:1)活动,Activity 处于屏幕的最前方以获取用户焦点;2)暂停,Activity 失去焦点后仍然可见;3)停止,被另外一个 Activity 完成遮住,这时原 Activity 的窗口是隐蔽的;4)待用,Activity 处于暂停或停止状态后,系统可能直接杀死它,需要时必须重新启动以恢复到以前的状态。服务也可以单独作为 Android 的组件,但不同于 Activity 的是,服务在后台运行时没有可视化界面,它也拥有独立的生命周期,服务组件通常起到辅助作用,例如在后台提供服务、监控其他组件等。

2.2.2 Intent 机制

Intent 是一种消息传递媒介,可以在一个应用的不同组件间或者不同应用之间传递消息,同时也是一种数据结构。Android 系统中的三大组件(Activity、服务、广播接收器)可以通过 Intent 触发。潜在地,通过分析程序中的 Intent 可以很好地了解应用程序的结构。Intent 有两种类型,即显式 Intent 和隐式 Intent。

1)显式 Intent:在应用中创建 Intent 对象时已经指定了消息的接收者。显式 Intent 往往不需要携带额外的匹配信息,正是由于这个原因,其只能在同一个应用的不同组件间完成通信。

2)隐式 Intent:这种 Intent 不明确指定消息接收者,而是携带了匹配信息,系统会帮助应用筛选符合条件的组件,这时的 Intent 可以发送给其他进程,大多数时候符合条件的信息接收者不止一个。

2.2.3 Smali Code

Android 系统运行在虚拟机 DVM 上,该虚拟机与 Java 的虚拟机 JVM 完全不同。DVM 不会直接运行 Java 程序的二进制文件,DX 工具会将程序中的 class 文件全部转换为dex 文件,dex 文件的运行环境就是 DVM。实际上,dex 就是 DVM 的二进制文件。dex 文件是完全由 Java 源代码转换过来的,从理论上讲,dex 文件也可以是逆向工程分析的目标。但是 dex 文件是二进制的,内部的语法非常复杂,而且打开后可读性极差,在这种情况下 dex 并不是最好的选择。

Smali Code 是 DVM 虚拟机上 dex 文件反汇编后的代码,对应的 Smali 语言就是 DVM 的反汇编语言,并支持 dex 的全部功能。Smali Code 成为了逆向工程的新目标。Smali Code 有单独的语法,且比较简单,相比 dex 语法具有明显的优势;另外,smali文件可以直接以文本形式打开,可读性极高。下面介绍本文使用的 Smali Code 的语法结构。

(1)数据类型。DVM 字节码可以分为原始类型和引用类型,除了对象和数组属于引用类型外,其他都属于原始类型。Smali 用一个字母表示数据类型,不同的数组元素类型使用不同的大写字母。其实,Smali 数据类型的字母与 Java 基本数据类型首字母一一对应,boolean 类型例外(用大写的“Z”表示),对应关系如表 1 所列。

表 1 数据类型表示方法

void	boolean	byte	short	char	int	long(64)	float	double
V	Z	B	S	C	I	J	F	D

对象类型的表示格式为：Lpackage/name/ObjectName;。前面的L表示这是一个对象类型，package/name/是指对象所在的包，ObjectName 是对象的名字，";"表示对象名称的结束，不能丢弃。类似于 Java 的 package. name. ObjectName，Ljava/lang/String;代表 java. lang. String。

(2)方法。方法必须详细地指定方法类型、方法名、参数类型和返回类型。方法的表示格式为：Lpackage/name/ObjectName;->MethodName(III)Z。它虽然从表面看起来较复杂，但拆分后非常容易理解：Lpackage/name/ObjectName 表示类型，MethodName 代表方法名，III 代表该方法有 3 个整型参数，方法的参数是连接在一起的，中间不分隔。"public abstract Context getViewContext();"被表示成". method public abstract getViewContext() Landroid/content/Context;"。

(3)字段。字段在程序中的使用方式和表示格式都非常简单，由包名、字段名、各字段类型组成。表示格式为：Lpackage/name/ObjectName;->FieldName:Ljava/lang/String，示例如表 2 所列。

表 2 字段的表示形式

Java 代码	Smali 代码
public boolean isLarge;	. field public isLarge:Z
public boolean resize;	. field public resize:Z
public int thumbnailSize;	. field public thumbnailSize:I
public String url;	. field public url:Ljava/lang/String;

2.2.4 设备通信

Android Debug Bridge(ADB)是 Android SDK 下的一款强大工具。我们可以通过 ADB 直接对设备发送命令，甚至直接切换到 Android 系统中执行内核命令。ADB 是外界与设备通信的渠道，这为测试平台介入系统提供了通道。

ADB 有多种功能，例如直接更新设备上的代码、操作设备执行 shell 命令、管理设备端口和设备间的文件复制等，所有的功能都体现出 ADB 作为调试桥梁的特性。在本文的方法中，利用 ADB 与设备通信，实现了设备管理工作，同时，通过 ADB 发送 shell 命令辅助完成测试工作。

2.3 视图和事件

Android 系统中的所有 UI 类都是在 View 和 ViewGroup 这两个类的基础上建立的。Android 系统的视图结构采用组合模式设计思路，将 View 作为所有图形子类的基类，ViewGroup 也是 View 的继承视图容器类，由此形成了树形视图基本结构，如图 2 所示。Android UI 程序屏幕体系结构的组织遵循以下原则：一个屏幕可以包含一个视图；视图组本身也是一个视图；视图组可以包含若干个视图。

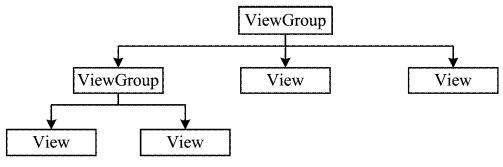


图 2 视图的树形结构

Android 事件在表现形式上有多种类型，包括点击、滑动、拖动、长按、手势、传感器等，在系统内的处理函数形式包括 onTach，onClick 和 onLongClick 等，在内核的表现形势有 action_down，action_move 和 action_up 等。尽管事件表现类型不同，但都要通过事件传递模型实现。Android 程序的事件传递模型与 JS 的事件传递模型有很多相似之处，二者都采用了先捕获后冒泡的处理原则。在捕获阶段，即发现阶段，外部的 View 发现事件后，将事件传递给内层的子 View，依次向内传递，逐渐缩小范围，直到发现拥有此事件的最小 View 单元，这是完整的事件捕获过程；冒泡阶段是事件处理阶段，事件从最小的 View 单元开始处理，若不能完全处理则向外冒泡，将事件对外传递。

捕获阶段和冒泡阶段是整个事件从发现到处理的整个流程，但实际的过程并不会这么简单，Android 系统的表现相对较复杂，主要原因为：1)控制每层事件是否需要继续传递(需要依赖事件分发、事件拦截的协同进行)；2)事件的具体消费情况复杂，原则上是事件响应进行使用，但很多时候事件分发自身也会使用事件。

Android 中含有多种控件类型，每类控件的事件分发、拦截和响应各有不同，这是因为 Activity 是界面载体而不是界面元素，Activity 本身没有事件拦截功能，只有 ViewGroup 的最小 View 单元(或者 View)才可能执行事件分发和拦截。如何管理事件传递机制和事件响应机制主要依赖以下方法的返回值类型。

(1)事件分发处理方法 dispatchTouchEvent(MotionEvent)。当监听器发现事件后，最初由 Activity 对其进行捕获，接着立刻进入事件分发处理流程。由于 Activity 和 View 都具有事件分发功能，同时自身也具有消费能力，因此如果事件分发方法返回 true，则表明事件在本层完全可以被处理，即分发的过程中同时消耗了事件，分发结束时事件已经完结。当然，我们可以消除 Activity 中的任何控件的事件响应能力，只需要重载 Activity 的 dispatchTouchEvent 方法，直接返回 true。如果事件分发方法返回 false，则表明事件在本层继续分发，并交由上层控件的 onTouchEvent 方法进行消费。如果当前控件是 Activity，事件直接交给系统处理。如果事件分发返回的是 super. dispatchTouchEvent(ev)，事件将分发给本层的事件拦截 onInterceptTouchEvent 方法进行处理。如果当前控件是 Activity，它没有事件拦截功能，则会直接把事件传递到子 View，子 View 的事件分发方法开始处理。

(2)事件拦截处理方法 onInterceptTouchEvent(MotionEvent)。当 onInterceptTouchEvent 返回的结果是 true 时，表明对事件进行了拦截，同时拦截的事件被交给本层控件的 onTouchEvent 方法处理；当返回结果是 false 时，表明不对事件进行拦截，事件在本次被分发到后续层的子 View，子 View 的 dispatchTouchEvent 处理事件。当返回 super. onInterceptTouchEvent(ev)时，默认不会拦截事件，将其分发给子 View 的 dispatchTouchEvent 处理。

(3)事件响应处理方法 onTouchEvent(MotionEvent)。当 onTouchEvent 返回值是 true 时，表明 onTouchEvent 消费了这个事件，到此事件完全结束，不再冒泡。如果 on-

TouchEvent 返回的是 false,那么事件在 onTouchEvent 中处理后继续冒泡,传递给子 View 的 onTouchEvent 进行处理。如果返回的是 super. onTouchEvent(ev),则默认处理方法与返回 false 时的情况相同。

上面分析了事件分发和事件消费流程,无论 dispatchTouchEvent 方法返回 true 还是 false,事件都不再进行分发,只有当其返回 super. dispatchTouchEvent(ev)时,才表明事件会向下层分发,但这并不代表分发就会成功,还需要事件拦截方法 onInterceptTouchEvent 的同意。事件是否继续冒泡,由 onTouchEvent 方法返回值的类型决定。

3　移动应用行为状态机模型的构建方法

本文提出了基于 Smali Code 的模型生成方法,该方法通过逆向工程得到 Smali Code,避免使用源码;同时,Smali Code 与源代码完全对应有利于提高模型的覆盖度和准确度。该方法的建模对象是可执行文件。生成 Smali 后将系统层的代码剥离,以减少开销。通过动态与静态结合的方法找到 App 中所有的控件,并识别出每个控件绑定的事件类型。另外,建模时通过在系统层扫描屏幕变化来判断状态是否发生变化,不再需要人工为屏幕定义状态向量。本方法的目的是探索有效的移动应用行为模型构建方法,在提高覆盖度的同时降低建模成本。

3.1　方法框架

本文研究方法的流程如图 3 所示。本文的测试模型生成方法与很多基于模型的测试方法的模型生成流程相似,不同之处主要表现在两大方面:1)基于 Smali Code 的模型生成方法;2)引入自动化的状态分析。

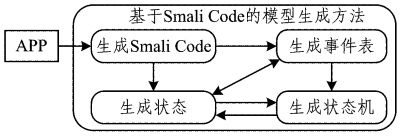


图 3　方法框架图

从流程图中可以看出,流程的入口就是 Android 的可执行文件 Apk(通常也被称为 App),不需要源代码,获取源代码的模型生成方法的适用性受到了很大的限制,避开源代码的建模优势显而易见。图 3 所示的框架会生成一个有限状态机模型,该模型可以被用来生成测试用例。

(1)生成 Smali Code 模块。这是预处理部分,负责从 App 中生成 Smali Code,以及对 App 的配置信息进行分析,主要是针对 AndroidManifest. xml 文件。

(2)生成事件表模块。Android 应用程序是基于事件驱动的,事件表包含了模型中所有的状态转换事件,所有的界面都对应了 Activity。该部分首先要找到 App 中的所有 Activity 以及 Activity 之间的关联关系;根据 Activity 之间的关系,有序地遍历 Smali Code 中的每个 Activity,找到每个控件绑定的事件;最后生成一个完整的事件表〈Activity,控件类型,变量,id,事件〉。

(3)生成状态模块。本文研究方法结合了系统层。阻碍现有移动应用模型生成方法实现自动化的一个最大难点在于需要手工定义状态机的状态,通过人工判断界面上的哪些属性是状态变化的依据。现有研究都是通过人工为每个界面定义一个向量,这是由于忽略了系统层而造成。本文将系统层划入平台的一部分,通过在系统层扫描屏幕的变化来判断是否发生了状态变化,不再需要人工干预,该部分负责系统层的工作。

(4)生成状态机模块。通过事件分析模块,找到了程序中所有的 Activity 以及每个 Activity 所附属的事件表,它们是组成模型的所有元素,但并不是一个模型。该部分负责建立一个有限状态机模型。

基于 Smali Code 的模型生成方法主要有如下 3 个特征:不需要源代码;通过逆向工程方法得到 Smali Code,通过静态分析从 Smali Code 中获取信息,针对性强;延伸到系统层,通过扫描屏幕发现状态变化,不必手动定义状态,提高了自动化程度。

3.2　生成 Smali Code

逆向工程是抽象定义,具体实施技术因分析对象的不同而不同。分析的对象可以是二进制流、可执行文件、源代码、中间码和设计文档等。最早的逆向工程用来帮助工程师理解系统,以便维护软件。目前,逆向工程已经发展到追求与源代码一致,利用逆向工程方法可以从可执行文件中抽取体系结构、类定义和变量等。以往为了解决建立准确模型的难题,测试人员需要根据文档、代码和肉眼观察进行建模。依赖文档或观察建立的模型准确度较低,根据代码建立的模型准确度较高,当没有源代码时,逆向工程的好处就显而易见,这也是本文研究工作选择逆向工程的原因。由于移动应用多数涉及用户隐私,通过逆向工程得到代码成为了 Android 平台测试的有效辅助手段。

虽然逆向工程是从可执行文件中反编译出重要信息,但由于目的和设计思路不同,最终逆向的结果在不同的工程中不尽相同。本文方法需要如下资源:

(1)Smali 代码:代码是整个可执行文件的核心,获得代码资源后可以建立程序的静态结构,同时能够得到对象、变量的定义信息。

(2)配置文件:Android 应用需要配置设备、版本、权限等,每个可执行文件都会携带配置信息。配置信息包括两部分,一部分是该应用自身的信息,如 Activity 等;另一部分是安装需要的配置信息,如权限等。

Smali Code 逆向工程方法通常都应用于安全领域,本文将它应用于测试建模领域,这是基于 Smali Code 的模型生成方法的第一步。在这个阶段,不对反编译出的 Smali Code 做任何处理,但需要分析配置信息,并从中提取出 Android App 运行的配置要求。

Apk 文件是 Android 代码打包后生成的安装程序,即本文提及的可执行文件。每个程序都有一个 AndroidManifest. xml 文件存储于根目录中。这个文件的清单中列出了 Apk 的基本信息,Android 系统在运行之前必须知道这些配置信息。Android 配置文件由 XML 语言描述,每个 XML 标签都有不同的含义,多数情况下配置参数放在标签属性中,找到了这些标签就相当于找到了配置要求。〈uses-permission〉标签包含的是 Android 应用的权限机制,Android 框架制定了一套

严格的权限系统。每一个程序都需要提前对使用的权限进行声明，声明后获取了权限才可以使用对应的功能。例如，若应用软件需要访问网络，则需要配置“android. permission. INTERNET”权限申请；同样，若要使用设备的相机功能，则必须在标签下申请“android. permission. CAMERA”权限。在〈uses-permission〉标签下，通过 android:name 属性来声明权限名。〈uses-permission〉标签定义的是使用的具体权限，而很多情况下不需要为应用程序声明某个权限。如果需要给其他应用程序提供可调用的代码，这时就需要使用〈permission〉标签。有时，还可以和〈permission-group〉标签、〈permission-tree〉标签配合使用，以构造出更有层次、更有针对性的权限系统。〈uses-configuration〉标签和〈uses-feature〉标签被用来定义需要的硬件和软件特性，避免程序安装在没有这些特性的设备上。这些重要的信息定义了软件的执行环境，在安装和运行时，可执行文件会依据这些信息对设备提出要求，如果设备不能满足全部要求，那么软件将无法在设备上安装或执行。

通过逆向工程得到 AndroidManifest. xml 文件后，生成一个 XML 文件的处理器，它可以将重要的标签全部提取出来并存放到统一的数据结构中；后续工作将访问该数据结构。

3.3 生成事件表

3.3.1 GUI 系统建模中的状态和事件

GUI 是用户和程序进行交互的重要方法，它虽然提供了可视化的用户图形界面，但是建模复杂度也很大。GUI 程序都是事件驱动机制，事件驱动程序拥有大量的状态，其弊端之一是生成的事件序列空间巨大。初略地认识，GUI 系统中的界面是状态机模型中状态的来源，而用户交互行为对应状态机的迁移事件，如图 4 所示。应用软件有 3 个界面，在状态机上存在与其对应的 3 个状态，状态机上的事件 e1，e2，e3，e4，e5 对应了软件上可以接受的点击动作〈click〉，〈fill〉，〈ok〉，〈fill＋ok〉。

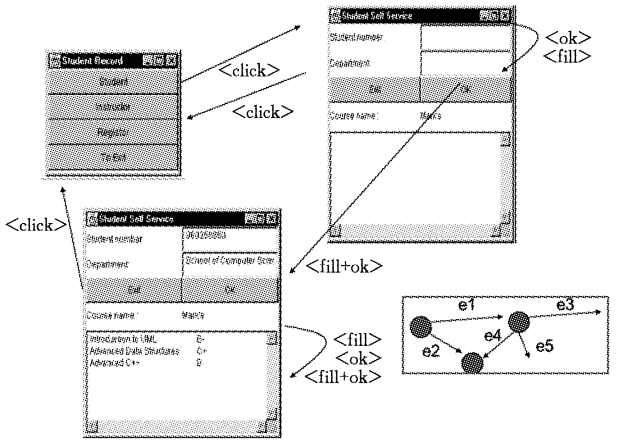


图 4 GUI 系统和状态机模型的对应关系图

Android 系统是典型的 GUI 系统，事件驱动特征明显。当然，Android 系统的状态和事件不仅仅来自界面和用户交互，例如系统事件也会导致状态变化。我们将 Android 应用的 GUI 结构建成有限状态机模型。虽然 GUI 程序通常都有大量的、无穷多的 UI 状态，但我们的目标非常明确：利用移动应用中简单且直观的 GUI 设计得到一个精简且高质量的模型。在 Android 应用的界面设计中，同一个 App 的不同界面往往都是完全不同的结构设计，绝不仅仅是格式上的差别，我们需要做的是抓住并反馈其中重要的差别（如一个按钮可用与否），一些支持用户动作的 GUI 控件的属性能够反映出这些差别。为了保持模型的简洁，忽略一些无关紧要的变化（例如用户输入的文字不同），以有效地缩减 UI 状态。

3.3.2 基于 Intent 的界面变化研究

找到 GUI 程序与模型的对应关系后，深入分析 Android 系统，目标是对如何定义有限状态机模型的状态和事件进行规划。在 Android 系统的四大组件中，Activity 是唯一的图形界面，它支撑了所有屏幕，也就是说，GUI 状态机中状态的源头就是代码中的 Activity。屏幕变化的唯一特征是屏幕上控件的变化。本文根据 Activity 是否改变，将界面变化分为两类。

(1)Activity 无变化。这种情况下，虽然屏幕状态改变，但是当前界面和上一个界面属于同一个 Activity，这是因为事件并没有触发 Intent，或者触发 Intent 指向自己（少见），如空白的文本框显示文字。如图 5 所示，界面 1、界面 2 和界面 3 属于同一 Activity，它们之间的变化不改变 Activity。

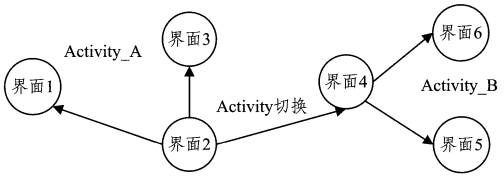


图 5 界面变化原理图

(2)Activity 切换。这种情况下，屏幕变化一般较大，这是因为事件触发了 Intent，跳转到了另外的 Activity。如图 5 所示，3 个界面 2 跳转到界面 4 时，代码中的 Activity 也发生了改变。

虽然 Intent 只是导致 Activity 变化的一类原因，但同一 Activity 上界面变化的前提是进入了这个 Activity，也就是说，Activity 和 Intent 可以组成状态变化的主干图，这个主干图就是 Activity 调用图（Activity Call Graph，ACG）。ACG 图的最大优点是可以有序地遍历到每个 Activity，为散乱的 Smali Code 形成次序。

为了建立模型，需要找到每个 Activity 上的所有控件和每个 Activity 控件上绑定的事件。通过逆向工程手段，能够找到 Apk 中所有的 Activity，但这只是开始，我们还不清楚 Activity 之间的关系。

ACG 就是 Activity 间的调用关系图。在 Android 系统中，Activity 之间的转换是靠 Intent 实现的，Intent 的功能就是启动新的组件。在代码中，一个 Activity 启动另一个 Activity 的方法是固定的，即通过调用 startActivity(Intent)函数和 startActivityForResult(Intent，Int)函数实现。Android 应用通过 Intent 在同一个应用间的不同组件和不同应用间传递信息。若要建立 ACG，除了需具有所有的 Activity，还必须具有所有的 Intent；另外，还需要分析出所有 Intent 的起始 Activity 和目标 Activity。

Android 系统中，Intent 被分为两类：显式 Intent 和隐式 Intent。显式 Intent 较容易处理，其在调用时指明了目标 Activity，大多数情况下调用发生在起始 Activity 中，那么就有一条路径：Activity A 经过 Intent 到达 Activity B。但是仅有 Intent 是不够的，因为包含 Intent 的控件才能触发事件。显式

Intent 分析方法如图 6 所示,在 Activity 类 TestActivity 中存在一个显式定义的 Intent(mIntent),mIntent 定义在 SerializeMethod()函数中,该函数在 sButton 绑定的事件监听器中被调用。分析方法是:在 Activity 中识别出是否有 Intent 的定义,若有,则需要根据 Intent 的定义形式分析出起始 Activity 和目标 Activity,然后采用回溯算法找到触发 Intent 的事件。

```
1. public class TestActivity extends Activity {
2.     @override
3.     Public void onCreate(Bundle savedInstance State) {
4.       sButton = (Button)findViewById(R. id. serButton);
5.       sButton. setOnClickListener(new
6.       View. OnClickListener(){
7.           @Override
8.           Public void onCLick(View v) {
9.               SerializeMethod();
10.          }
11.          });
12.    }
13.    Private void SerizlizeMethod(){
14.    Intent mIntent = new Intent(this,SerializableDemo,class);
15.    Bundle mBunble=new Bundle();
16.    mBundle. putSerizlizable(SER_KEY,mPerson);
17.    mIntent. putExtras(mBundle);
18. startActivity(mIntent);
```

图 6　显式 Intent 分析方法示意图

隐式 Intent 较为复杂,它没有明确指明目标 Activity,且符合条件的目标 Activity 经常有多个;还有一点比较特别,即隐式 Intent 代码在起始 Activity 中定义,但需要在 AndroidManifest. xml 中判断符合条件的目标 Activity,对系统的其他应用中符合条件的 Activity 无法判断,因此本文研究工作不支持跨应用。隐式 Intent 分析方法如图 7 所示,Intent 只携带了 action 信息,根据这条信息在配置文件中找到声明了该 action 的 Activity,其他分析过程与显式 Intent 的一致。

```
1. Intent intent = new Intent();
2. Intent. setAction("com. xiazdong. action");
3. intent. addCategory("com. xiazdong. category");
4. intent. setData(Uri. parse("xiazdong://www. xiazdong. com/xia");
5. startActivity(intent);
6. intent. addCategory("android. intent. category. DEFAULT");
7. <activity
8.     android:name=". OtherActivity"
9.     android:label="OtherActivity">
10.    <intent-filter>
11.      <action android:name="com. xiazdong. action"/>
12.      <category
13.    android:name="android. intent. category. DEFAULT"/>
14.      <category
15.    android:name="com. xiazdong. category"/>
16.      <data
17.        android:host="www. xiazdong. com"
18.        android:scheme="xiazdong"/>
19.    </intent-filter>
20.  </activity>
```

图 7　隐式 Intent 分析方法示意图

所有的目标 Activity 都可以通过分析 Intent 结构得到,但是起始 Activity 不一定总是这样。如果启动 Intent 的 startActivity(Intent)函数和 startActivityForResult(Intent,Int)函数是在 UI 控件的事件监听器中定义的,那么可以断定当前的 Activity 一定为起始 Activity,因为 UI 控件一定是附属于当前 Activity,这个 UI 控件所绑定的事件监听器就是从起始 Activity 到目标 Activity 的边。如果 Intent 不在事件监听器中定义,则必须根据函数调用图(FCG)逆向分析所有路径才能找到起始 Activity,由于这种情况非常少见,本文暂不处理这类 Intent。另外,整个 ACG 图起始的 Activity 就是 App 的主 Activity,它是启动时最先进入的界面,在配置文件中携带标签〈action android:name="android. intent. action. MAIN"/〉。

本文分析了 6 种 Intent 对象的定义形式,它们的特点和区别如下:

形式 1:Intent()。它定义了一个空 Intent,后续可以通过 setClass(),setComponent(),setActivity()等方式绑定目标 Activity。

形式 2:Intent(Intent)。这种形式是一种复制定义,由参数中的 Intent 决定,必须分析参数中 Intent 的起始 Activity 和目标 Activity。

形式 3:Intent(String)。这是隐式 Intent 定义方式,参数是 String 类型的 action,必须到 AndroidManifest. xml 寻找符合 action 的 Activity。

形式 4:Intent(String,Uri)。这种形式与形式 3 一样,也属于隐式 Intent 定义。

形式 5:Intent(Context,Class〈?〉)。一种显式 Intent 的定义方法,第二个参数直接定义了目标 Activity。

形式 6:Intent(String,Uri,Context,Class〈?〉)。这也是一种显式的 Intent 定义方法,最后一个参数说明了目标 Activity。

在 ACG 生成算法中(如算法 1 所示,表 3 中包含了其主要函数及说明),以 AndroidManifest. xml 中标记的 Main Activity 为入口,依次分析每个 Activity 中包含的 Intent。AndroidManifest 文件中包含了〈Activity〉标签,算法中的第一行将所有的 Activity 保存到链表中。从 ActivityList 开始循环,如果所有的 Activity 都被访问过,则算法结束;否则继续循环未被访问的 Activity。在处理每个 Activity 时,首先找到当前 Activity 下所有的 Intent 定义,如果没有 Intent 的定义,则从 ActivityList 中继续寻找还没被访问的 Activity,否则将当前 Activity 下所有的 Intent 保存在 IntentList 下。

算法 1　基于 Intent 的 Activity Call Group 生成算法

输入:中间码 Smali Code,配置文件 AndroidManifest. xml

输出:Intent 表,ACG 图(点 ACGNode,边 ACGEdge)

```
Begin
1. ActivityList=getActivityFromManifest(xml);
2. activityCurrent=getMainActivity(ActivityList);
3. While activityCurrent !=null do
4.     IntentList=findIntentFromCurrentActivity(activityCurrent);
5.     While IntentList !=NULL do
6.       intent=IntentList. getFirstIntent();
7.       Switch
8.         case intent == Explicit do //如果 intent 是显式定义的
9.           forSourceAndTargetActivityExplicit(intent);
10.          goBackForEvent(intent);
```

```
11.          update(IntentTable,ACGNode,ACGEdge);
12.        case intent == Implicit do
13.          forSourceAndTargetActivityImplicit(intent);
14.          goBackForEvent(intent);
15.          update(IntentTable,ACGNode,ACGEdge);
16.      EndSwitch
17.      IntentList.deleteFirstIntent();//结束当前的 intent 分析
18.    EndWhile
19.    activityCurren=getUnopenActivity(ActivityList);
20. EndWhile
```

表 3 算法 1 中的主要函数及其说明

主要函数	说明
getActivityFromManifest(Xml)	获取所有 Activity 列表
getMainActivity(ActivityList)	获取入口 Main Activity
findIntentFromCurrentActivity(Activity)	当前 Aciticity 包含的 Intent 列表
forSourceAndTargetActivityExplicit (Intent)	寻找 Intent 的起始 Activity 和目标 Activity
goBackForEvent(Intent)	寻找触发 Intent 的事件
update(IntentTable,ACGNode,ACGEdge)	更新 Intent 表

在处理 Intent 时，首先判断 Intent 的类型，显式定义和隐式定义的 Activity 处理函数不一样。Intent 如果是显式定义的，第 9 行会从 Intent 定义时的参数入手；如果是隐式定义的，则使用第 13 行的函数，还需要遍历 AndroidMainfest 文件，AndroidManifest 文件可以直接用 xml 解析器遍历所有子结点。无论 Intent 是显式定义的还是隐式定义的，goBackForEvent 函数和 update 函数的功能都相同。前者向后回溯找到可以触发该 Intent 的事件，分析方法前一小节，后者负责更新 ACG 图和 Intent 表。

3.3.3 Activity 事件表生成方法

上一节分析了界面变化的原理并生成了 ACG 图，本节将解决建模的两大要素之一——事件。上文生成 ACG 图的目的是找出 App 中的事件，只有找出 App 中所有的事件才可能到达 App 中包含的所有状态。

Android 系统可视界面上有各种控件，但是真正能够影响程序的是绑定了事件的控件，即必须注册了监听器。在图 8 中，定义了 View 类型对象 btn_delete，通过 findViewById() 查找布局文件中的指定 id 的组件，View 是父类，真正实例化后的 btn_delete 对象是 Button 类；紧接着的两段代码是 btn_delete 注册的两个事件监听器，分别是单击和长按。

```
1. View btn_delete=findViewById(R.id.btn_delete);
2. Btn_delete.setOnClickListener(new onClickListener()
3. {
4.       public void onClick(View v){
5.          removeBillAmount();
6.          FlurryAgent.onEvent("Delete Button");
7.       }
8. });
9. Btn_delete.setOnLongClickListener(new
10. onLongCLickListener(){
11.      public void onLongClick(View v){
12.          clearBillAmount();
13.          return true;
14.      }
15. });
```

图 8 组件绑定事件分析方法示意图

每一个 Activity 拥有的控件主要包括 3 种类型：View 类型、Menu 类型和 Dialog 类型。

(1)View 类型。对于这种类型，需要提前建立一个 View 子类库，这个子类库包含了所有 Activity 界面上可视并且可以绑定事件的控件类名，例如 Button，TextView，EditText 等，所有的这些内容都来自 Android 开发者官网，将所有出现过的控件种类都加入到 View 子类库中，这个过程是手动完成的。在寻找控件时，需要扫描每个 Activity 中的对象定义，如果有类型已包含在 View 子类库中，那么就发现了一个可能的控件，最终还要决定于是否绑定事件，仅仅定义而未绑定事件的控件不影响程序执行。另外，还要找到每个对象的 id 代码表示形式，如图 8 中的 R.id.btn_delete，但这种形式一般不能被测试用例执行工具支持，还必须找到数字代码。

(2)Menu 类型。Android 中共有 3 种菜单：选项菜单、上下文菜单和子菜单。选项菜单最常用，有两个固定的系统调用函数，在 onCreateOptionsMenu(Menu) 函数中创建，在 onOptionsItemSelected(MenuItem) 函数中处理。上下文菜单与选项菜单的形式一致，也由两个固定的系统调用函数(onCreateContextMenu() 和 onContextItemSelected(MenuItem item)) 来处理，而且这两个函数内部也是一致的。子菜单本质上属于以上两种菜单，只是在两类菜单的一个子菜单中再设置一层菜单，这种编程方式由于体验不好已经被淘汰，本研究中不处理子菜单和动态生成菜单。由于菜单的生成和处理通过系统函数定义，本研究跟踪的目标就是上述 4 个函数。

(3)Dialog 类型。Android 支持的 Dialog 种类较多，主要包括 AlertDialog，ProgressDialog，DatePrickerDialog，TimePickerDialog等，还可以自己扩展，其中 AlertDialog 是最普遍使用的。AlertDialog.Builder 类是对话框的定义函数。

除此之外，还有系统事件和手势事件等。系统事件包括电池、来电等，它可以在任意时候打断应用，这会增加建模的压力，本研究不考虑系统事件。手势事件与菜单类似，有固定的方法调用格式，通过扫描 Smali Code 观察这些方法是否被重载就可以判断一个 Activity 中有无手势事件。

本文方法分为两步：第一步寻找控件，第二步根据找到的控件查找绑定的事件。首先，找出 Activity 中定义的控件和 id，生成〈Activity，控件类型，变量，id〉表。仅仅有了变量名可能是无效的，考虑到后续研究可能展开测试用例生成，驱动工具只能识别 id。Android 中的 id 都保存在 gen 目录下的 R.class类文件的子类 id 中，形成 Smali Code 后，保存在单独的 RMYMid.smali 文件中。提取这些变量的 id 时需要使用对应代码，如 R.id.btn_delete。下一个阶段是寻找控件绑定的事件。寻找绑定的事件还需要扫描 Smali Code，需要以〈Activity，控件类型，变量，id〉表中的变量为依据找出每个变量绑定的事件。以图 8 为例，btn_delete 定义了两个事件，即〈Activity，Button，btn_delete，0x7f060006，OnClickListener〉和〈Activity，Button，btn_delete，0x7f060006，OnLongClickListener〉，最终生成一个事件表。事件表中包含了以下信息：每个 Activity 下控件的数量，每个控件上绑定的事件数量，每个事件的类型。

本方法生成事件表时采用了动态和静态相结合的方式(如算法 2 所示,表 4 包含了算法 2 的主要函数及其说明)。第一个阶段是动态执行的过程,这个阶段通过执行 App 获得每个 Activity 上绑定的控件,但这时还不知道每个控件绑定的事件。第二个阶段是静态执行的过程,这个阶段完全通过扫描 Smali Code 找到每个控件绑定的事件。

算法 2　事件表生成算法

```
输入:ACG 图,Intent 表,中间码 Smali Code
输出:事件表
Begin
1. ACGVisitedTable=get(ACGNode,0);
2. firstActivity=getAfterRunning();
3. getEventFromFirstActivity(EventTable,firstActivity);
4. setACGVisited(ACGVisitedTable,firstActivity,1);
5. Stack(firstActivity);
6. While Stack !=NULL do
7.     currentActivity=Stack.getTopValue();
8.     If hasNextActivity(ACG,currentActivity,next) and
9.           getValue(ACGVisitedTable,next) == 0 then
10.          action=find(IntentTable,currentActivity,next);
11.          executeAction(action);
12.          getViewEvent(EventTable,next);
13.          getOtherEvent(EventTable,Smali Code,next);
14.          setACGVisited(ACGVisitedTable,next,1);
15.          Stack.add(next);
16.    else
17.          Stack.pop();
18.          path=getPath(IntentTable,firstActivity,Stack.getTop-
             Value());
19.          RestartApp(path);
20.    EndIf
21. EndWhile
22. scanSmaliCodeForEvent(EventTable);
```

表 4　算法 2 中的主要函数及其说明

主要函数	说明
getAfterRunning()	获取第一个 Activity
getEventFromFirstActivity(EventTable, Activity)	获取第一个 Activity 的事件列表
setACGVisited(ACGVisitedTable, Activity,1)	更新访问情况,标注已访问内容
executeAction(action)	执行 action 到达下一个界面
getViewEvent(EventTable,Activity)	访问 Activity 上的 View 控件
getOtherEvent(EventTable, Smali Code, Activity)	访问 Activity 上的非 View 控件
getPath(IntentTable, Activity, Stack.getTopValue())	记录即将访问的 Activity 路径
RestartApp(path)	利用访问路径重启 App,达到需要的界面
scanSmaliCodeForEvent(EventTable)	扫描 Smali Code 获得控件绑定的事件

第一阶段是动态执行。整个过程即为 ACG 图的深度优先遍历算法,App 启动后第一个界面打开,算法首先访问第一个 Activity。在获取每个 Activity 上的控件时(第 12 行),扫描整个界面,通过遍历界面对应的层次树可以直接找到所有的可视控件。但并不是所有的控件都是当前可视的,例如菜单、对话框和手势,通过扫描 Smali Code 找出未在界面上显示的控件(第 13 行)。其实也可以不执行 App,全部通过扫描 Smali Code 来寻找控件,但是由于没有 Smali Code 的语法树分析工具,直接从运行的 App 中获得控件减少了很多工作量。算法中最关键之处是弹栈的处理,ACG 正在访问的 Activity 必须与当前界面的 Activity 同步,算法中的弹栈与 App 的返回不是对应的。弹栈后,我们就可以知道新的栈顶元素,通过 ACG 找到从程序启动到这个 Activity 的事件序列,重启 App 后直接到达这个 Activity。

第二阶段是静态扫描。这个阶段依赖 Smali Code 和第一阶段的结果。根据 Smali Code 语法设计了扫描函数(第 22 行),它扫描所有的控件和事件监听器的关键字,如果一段语法是方法的定义,同时包含了关键字,则认为控件绑定了一个事件,将监听器的关键字存入事件表中。这里的限制是只能分析已知类型的事件。

3.4　生成状态

生成模型的难点之一是定义状态。屏幕上的每一个界面一定存在对应的状态机状态,而状态由包含的属性和属性的值两部分组成,在 Android 系统的状态机中,属性就是屏幕上的控件,属性值就是控件中包含的元素,用于表达状态的变化。如图 9 所示,当前屏幕对应的状态中有一个属性是 ListView 类型,属性值是(A,B)两个子元素。需要注意的是,"Notes"也是一个控件,但是它由于没有绑定事件,因此在上一小节中已经被过滤掉,不再包含在事件表中。以往需要人工判断界面上的哪些属性是状态变化的依据,现有研究为每个界面定义一个向量,这种方法是完全从用户视角理解界面。本文方法将系统层划入平台的一部分,通过在系统层扫描屏幕的变化来判断状态是否变化,不再需要人工干预。界面变化的原因是界面上控件状态发生变化,屏幕上控件的变化才可能产生新的状态,其思路是先划分出界面上的控件后再关注变化,因此如何划分界面控件显得非常关键,但这一过程变化因素较多,无法实现自动化。

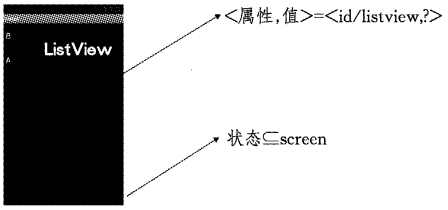


图 9　状态的属性与属性值

本文的思想是避开对界面上控件的划分,一个状态包含了整个界面,无需定义状态的控件集合。我们需要实时扫描设备来获取界面的底层表示,每个界面由许多字符串表示,只需通过比较这些字符串之间的差别就能判断界面是否变化,如果字符串表示已经出现过,则说明该状态是已经定义的状态,只需要添加事件即可。图 10 给出了 Note 程序的两个界面,在第一个界面中列表是空的;在第二个界面中列表有一个子元素,值为 abc。Android 设备在生成每一屏时都会在底层同步生成屏幕的特殊表示形式,可以通过 ViewServer 从外部

直接访问，并以树的遍历算法输出。第一个界面和第二个界面在屏幕上的不同可以通过字符串序列来反映，第二屏多了的一行，表示这是一个新的状态。

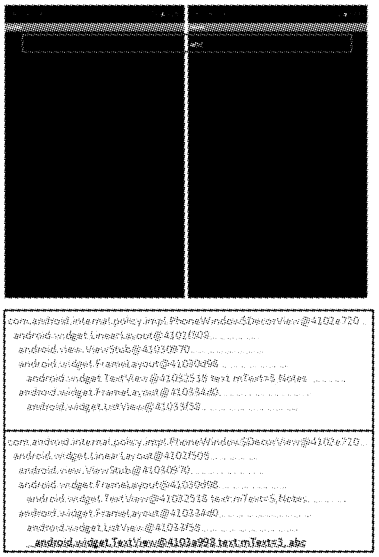


图 10 字符串比对分析界面变化示意图

3.5 生成状态机模型

遍历算法(算法 3)的目的是经过 App 中所有的状态，同时触发所有的事件。遍历算法从程序启动开始，遍历完所有的状态和事件后结束。这里采用了深度优先遍历算法。

算法 3 状态机生成算法

输入：事件表

输出：状态机模型

Begin

1. Model=Null;
2. s=getFirstScreen(App);
3. While s != null do
4. s=getNextState(s,App,Model,M);
5. s=backTrack(s,App);
6. if isInitialState(s) then
7. s=findNewOpenState(s,M,App);
8. EndIf
9. EndWhile

遍历算法主要是递归执行 3 个阶段。第一阶段(第 4 行)，从当前状态的所有未被访问事件中随机选择一个来执行，直到到达一个完全状态，完全状态下所有的事件都会被访问。此时，进入第二阶段(第 5 行)，它的功能是从当前的状态返回到非完全状态(还有事件没被执行)，或者直接返回到模型的初始状态。如果返回的是一个非完全状态，从这个状态开始继续深度优先遍历；如果返回的是模型的一个初始状态，则需找到一个非完全状态，并从它开始执行遍历算法(第 6—7 行)。算法中最核心的部分是 getNextState()函数(第 4 行)，在深度遍历的过程中，它会不停地触发事件，更新状态机模型。对于一个给定的状态，它会获取这个状态上的一个未被访问事件并执行，不停更新当前状态的事件表，将未被访问的事件标记为已访问，将新状态添加到状态机模型中。

4 原型工具和实验研究

针对所提出的方法，实现了原型工具并完成了相应的实验。本节将介绍工具的原理和实例的研究情况。

4.1 工具和实验 App 介绍

原型工具基于 Eclipse 平台开发，其框架如图 11 所示。

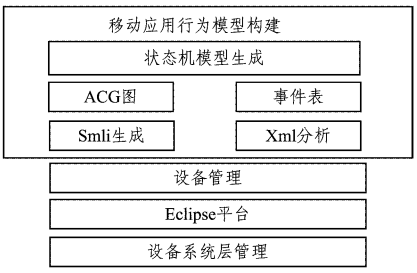


图 11 框架图

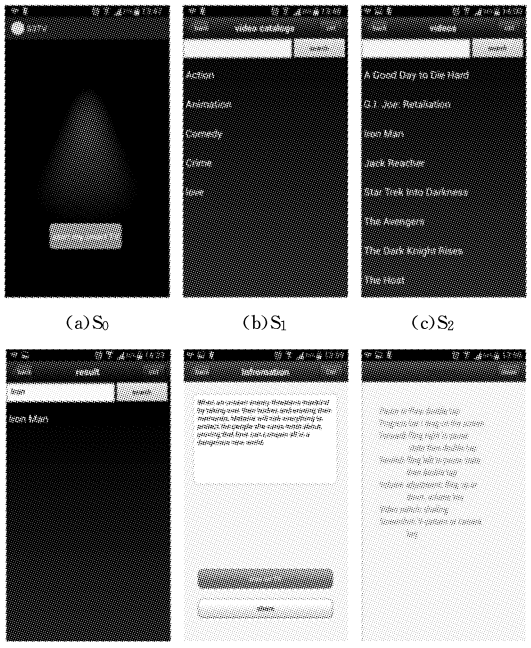
设备管理主要包括多设备控制和设备监视两大功能，这也是本文研究的基础工作。由于主要需求来自工业界，因此实用型的工具必须能够支持多设备部署。在本文的实现中，单个 PC 最多可以同时控制 16 个设备、模拟器或真机。在 PC 端可以并发地对所有设备发出命令，并与设备交互通信。特别地，管理多设备时，由于设备未必摆放在近处，使用人员也不能记住所有的设备，因此本文实现了对多设备的远程监控，在 PC 端可以打开设备列表，选择任意想要观看的设备并同步观看设备界面，还可以截图、录像、查看设备的内部文件等。

移动应用行为模型的构建主要是基于 Smali Code 的模型生成方法，包括静态分析、建立模型。静态分析功能块从反编译 Apk 开始，得到配置信息和 Smali Code，Smali Code 是后续建模的依据。在生成模型功能块中保存 Apk 的 ACG 图、事件表和模型。

本文实验的研究对象为 S3TV(Second-Screen Smart TV application)，它是我们研究智能电视人机交互时开发的一款软件。该 App 与智能电视建立连接，并从智能电视上下载本地视频库，用户可以选择喜欢的视频进行观看。智能电视与传统电视有很大的区别，其包含了网络设备功能，给人机交互模式带来了新的挑战；智能电视上有文本输入和用户定制内容，传统的电视遥控器不适用，因此考虑通过手机 App 控制智能电视的播放，S3TV 中包含了大量的人机交互事件。S3TV 用于真实的人机交互场景，而且我们拥有 App 的所有资料，本文的实验研究在 S3TV 上展开。该 App 主要由 6 个界面组成，包含了点击、长按、拖动、滑动、手势 5 类重要的用户动作。S3TV 的所有界面如图 12 所示。

S3TV 的主要功能介绍如下：S_0 是启动后的欢迎界面，点击“start my smart TV”按钮进入 S_1 界面。S_1 界面列出了当前可以播放的视频类型，如动作片、爱情片等。在 S_1 中选中一个菜单后可以打开该类电影下的所有可播放视频列表，即 S_2 界面，也可以在 S_1 和 S_2 的搜索框中搜索想要观看的视频名称，S_3 是搜索结果。在 S_2 和 S_3 中选择一个视频后可以查看到其详细信息，即 S_4 界面。在 S_4 界面上点击“play on TV”后，智能电视端开始播放视频，同时界面跳转到 S_5。S_5 是播放控制界面，主要的控制方式有：1)双击实现播放和暂停；2)水平拖动实现进度条功能；3)上/下快速滑动(或者音量键)

实现增大/减小音量;4)“V”型手势(或者照相机键)实现截屏功能;5)长按弹出选项菜单。实现 S3TV 的主要目的就是尝试用手势事件控制智能电视的播放。



(a)S_0　(b)S_1　(c)S_2

(d)S_3　(e)S_4　(f)S_5

图 12　S3TV 界面示意图

4.2　设备管理

设备控制是本文工作的基础。工具部署在 PC 端,设备可以是模拟器或者真机。多设备控制存在两方面的难点:1)完成一系列的远程控制命令,并与真人的操作效果一致。本文研究目前实现了绝大多数命令,还加入了对设备系统的特别控制。2)多设备控制。多设备控制存在并发问题和选址问题,目前本文在 32 位 PC 上能同时支持 16 个设备并发,但考虑到实验主机内存上限,运行 10 个以下设备时主机性能不会明显下降。同时,本文工作实现了 PC 端对设备屏幕的实时映射,在 PC 端直接点击就可以观看到设备界面的变化,类似于电视直播;另外还可以直接截图和录像。

4.3　移动应用行为模型实验

在图 12 中 S3TV 展示了 6 个界面,但出现 S_3 界面的情况有两种,在 S_1 或 S_2 中搜索想要播放的视频时,搜索结果展现出了 S_3 界面。虽然两种方式最终展示的界面一样,但是经历的是不同的路径。图 13 展示了 S3TV 的有限状态机模型,这个模型是我们为了对比结果而手动生成的,只包含了用户代码中的状态变化,系统事件 Back 和 Home 并没有涉及其中,因此在 SS5 状态只有进入,这是由于在播放界面没有设置退出,必须依赖 Back 键退出,但我们在 D_0-S_0 的状态引入了 Back 操作,这是因为必须通过 Back 才可以到达对话框状态 D_0。$S_{3\text{-}1}$ 和 $S_{3\text{-}2}$ 状态都对应图 12 中的 S_3 屏幕,$S_{3\text{-}1}$ 是指从 S_2 到达 S_3 的情况,$S_{3\text{-}2}$ 是指通过 S_1 到达 S_3 的情况。另外,该状态机中不包含系统事件打断时的状态。

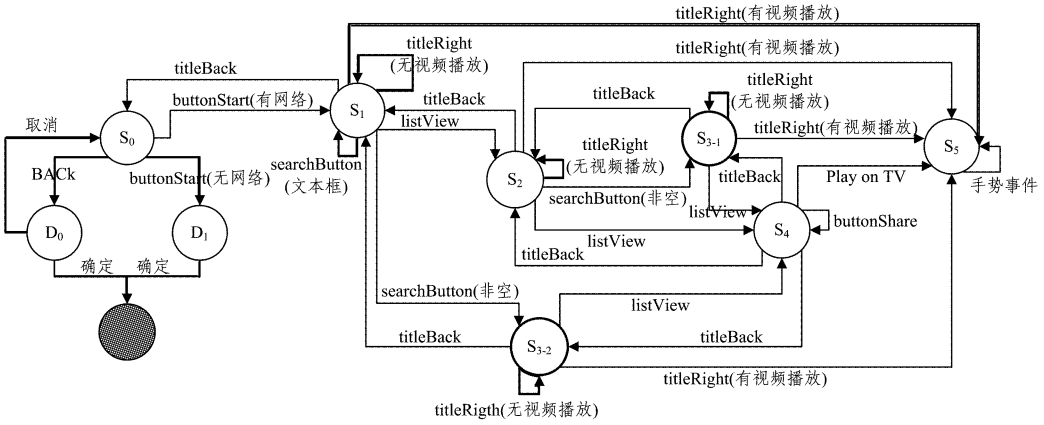


图 13　S3TV 的有限状态机模型

本文的实验对象共有 5 个 Activity 和 8 个 Intent,其中有一个隐式的 Intent。生成的 ACG 如图 14 所示。

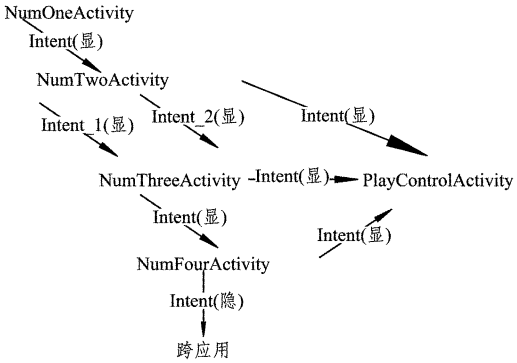


图 14　ACG 示意图

显式的 Intent 非常容易实现,即使两个 Activity 之间包含多个 Intent;但是,隐式的 Intent 处理起来比较麻烦,在第 3 节提到不处理跨应用,在 S3TV 中 NumFourActivity 包含了一个隐式的 Intent,action 是所有有分享功能的 Activity,由于 S3TV 没有自身的分享设置,响应的 Activity 全部来自其他应用(短信、微信、人人网等)。虽然这会导致 ACG 图不完整,但即便完整,我们依旧不能生成跨应用的模型,因此这并不妨碍后面的实验。

表 5 列出了原型工具生成的事件表,其包含了控件绑定的事件和手势事件,手势事件直接依附于 Activity,不依附于某个控件。需特别注意的是 EditText 类型的事件,与 TextView 类型不同,虽然它并没有绑定事件,但是 EditText 是否为空会改变程序的执行路径,本文将该类型纳入研究范围。

表 5 事件表

屏幕	Activity	对象类型	变量名	id	事件类型
S_0	NumOneActivity	Button	buttonStart	0x7f080017	setOnClickListener
S_1	NumTwoActivity	ListView	listView	0x7f080010	setOnItemClickListener
S_1	NumTwoActivity	Button	titleBack	0x7f08000b	setOnClickListener
S_1	NumTwoActivity	Button	titleRight	0x7f08000d	setOnClickListener
S_1	NumTwoActivity	EditText	editText	0x7f08000e	null
S_1	NumTwoActivity	Button	searchButton	0x7f08000f	setOnClickListener
S_2,S_3	NumThreeActivity	Button	titleRight	0x7f080006	setOnClickListener
S_2,S_3	NumThreeActivity	Button	titleBack	0x7f080004	setOnClickListener
S_2,S_3	NumThreeActivity	EditText	titleBack	0x7f080007	null
S_2,S_3	NumThreeActivity	Button	button	0x7f080008	setOnClickListener
S_2,S_3	NumThreeActivity	listView	listView	0x7f080009	setOnItemClickListener
S_4	NumFourActivity	Button	buttonPlay	0x7f080013	setOnClickListener
S_4	NumFourActivity	Button	buttonShare	0x7f080015	setOnClickListener
S_4	NumFourActivity	Button	titleBack	0x7f080019	setOnClickListener
S_4	NumFourActivity	Button	titleRight	0x7f080017	setOnClickListener
S_5	PlayControlActivity	GestureOverlayView	gestureView	0x7f080002	addOnGesturePerformedListener
S_5	PlayControlActivity				onDoubleTap
S_5	PlayControlActivity				onFling
S_5	PlayControlActivity				onLongPress
S_5	PlayControlActivity				onScroll
S_5	PlayControlActivity				onKeyDown

根据本文第 3 节提出的研究方法，算法生成了 S3TV 的有限状态机模型。为了便于说明，将生成的有限状态机与图 13 的有限状态机模型重合，最终的结果如图 15 所示。

图 15 中有 3 种类型的线：细实线，粗实线和虚线。细实线表示与图 13 一致的地方；粗实线表示图 13 中存在但在生成的状态机中丢失的线；虚线表示图 13 中不存在但在生成的状态机中出现的线。

从结果来看，本文方法能够生成有效的状态机模型，基本覆盖了大部分状态和事件，因此可以将 Smali Code 分析方法有效地应用于功能测试模型构建领域，从而摆脱对源代码的依赖。在最终的生成结果中，状态和事件还存在一些缺陷，这也是进一步改善实验结果的地方。

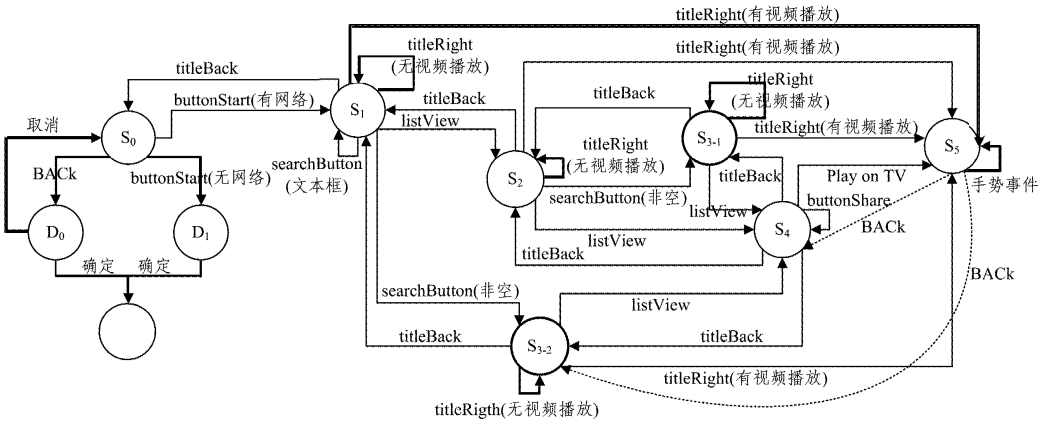


图 15 算法生成的 S3TV 状态机模型示意图

(1)状态缺失。在图 15 中，状态 S_{3-1} 和 S_{3-2} 用加粗标记，在最初的设计中，我们估计该算法的执行可能会导致状态冗余，由于 S3TV 界面不包含动态控件，因此没有出现状态数的增加，但出乎意料的是状态数出现减少。原因在于，模型生成算法判断 S_{3-1} 和 S_{3-2} 是同一个状态，这是由于我们通过扫描系统层来判断状态，而 S_{3-1} 和 S_{3-2} 来自于同一个 Activity，其内容相同但逻辑不同，算法没能识别出来；另外，由于状态机生成算法可以通过重启 App 来继续执行，因此掩盖了两个状态混乱对后续的影响。

(2)外部环境。由于 S3TV 与网络进行通信，因此很多环节会受到网络的影响，大部分情况下依靠异常来处理，但也有外部环境直接决定路径的情况。S_0-D_0 路径执行的条件是网络通信失败，因此网络正常时整个路径不能被访问到。

(3)EditText 类型。虽然 EditText 类型在多数情况下不会注册事件，但是由于文本框的内容会改变路径，因此本文将该类型加入建模方法。本文将 EditText 类型的状态分为非空和空两种，因此对每个 EditText 加入文本，但加入文本的时机和文本的内容仍然会对移动应用行为产生影响。为了方便，本文工具直接在第一次进入 Activity 时就填入文本，文本内容是视频库中存在的一个影片名。这种做法可以保证建模的高效性，避免一些琐碎的环节，但其弊端是有些路径不会被执行。如图 15 所示，S_1，S_2，S_3 的自环边都没有被执行，这是因为文本框为空的情况被屏蔽了。

(4)BACK 键。本文算法中，在没有回路时可以使用系统返回键，图 15 中两条虚线就是这样产生的。一条路径为 $S_0\rightarrow S_1\rightarrow S_2\rightarrow S_{3-1}\rightarrow S_4\rightarrow S_5\rightarrow S_4$，但是在理解模型时却可以生成一条 $S_0\rightarrow S_1\rightarrow S_2\rightarrow S_{3-1}\rightarrow S_4\rightarrow S_5\rightarrow S_{3-2}$ 路径，这是不合理的。

(5)条件改变。以 S_1 为例，开始执行 titleRight 按钮时没

有视频播放,有视频播放后执行 titleRight 的会产生一条新的路径,但是事件表中记录的 titleRight 已经被访问,这样也就丢失了一些路径。

本方法利用静态分析的结果指导最终模型生成。对比黑盒方法中执行 App 的方法可知,本文方法覆盖的状态和事件较为全面,同时减少了重复执行的次数及时间开销。

从工具的最终结果来看,基于 Smali Code 的模型生成方法可以达到与源代码一样的效果。虽然生成的模型还存在缺陷,但其中多数缺陷在其他方法中也存在且后续可以改善提高,况且摆脱了对源代码的依赖,因此在当前大量依赖第三方测试的情况下本文方法具有明显优势。

5　相关工作

模型驱动测试方法有很长的研究历史[5],该理论在软件工程领域形成了一套完善的解决问题的方法[6]。文献[7]采用了从软件抽取模型的技术手段,模型可以是 FSM,UML 图等。Dala[7]描述了模型测试方法的 3 个关键因素:描述程序行为的模型、测试用例生成算法和测试用例执行工具,其中模型构建是最关键的因素。Spec Explorer[8-9]是微软研究中心开发的面向对象语言的模型测试工具,它的最大特点在于基于动作表的建模方法。Spec Exploer 专门选择动作表作为模型,它的建模方法是将系统软件的输入和输出抽象表示为动作表[10-14]。由于在传统软件(C,C+,Java 等)上模型测试理论有一系列的成功应用案例,很多学者考虑为移动应用建模。Takala[15]研究了状态机模型在移动应用功能测试领域的应用可行性,其研究目的之一是提供真实的实验数据,将传统的 GUI 建模方法应用到移动应用领域;实验结果最终证实为移动应用建模以及基于模型的测试方法是可行的。他们还指出,在移动应用领域应用状态机模型的最大难点是模型的建立,其他问题甚至可以直接使用传统软件领域的工具来解决。

许多研究工作[16-23]都是通过构造领域模型来抽象程序行为。为移动应用程序建模与 Web 程序建模有很多相似之处,因为它们都是基于事件的。Marchetto[24]从 Ajax web 程序的 DOM 中提取一个有限状态机模型,通过静态分析技术提取信息,再根据这些信息动态地从 Ajax 应用程序中提取 FSM,这种动态与静态相结合的思想为移动应用建模提供了很好的参考。有学者提出了灰盒建模方法[25],即从源代码中提取模型元素,构建出完整、准确的状态机模型,其最大的缺点是完全依赖于程序源代码。除了状态机模型方法,为移动应用程序建模还包括 GUI 事件模型方法。对于基于 GUI 事件模型的方法,典型工作包括 Android GUITAR [26],Abbot[27],HP WinRunner[28],GUI crawler[29]等,它们通常依赖关键字(keyword action)重现用户界面的事件。在具体的 GUI 层次中将输入事件(input events)提取出来,然后用更高层次的表述来替代它,这些表述可以是 GUI 对象的句柄或者名字(通常系统与它们进行交互)。然而,并不是所有的应用都存在 GUI 元素,因此无法实现依赖关键字的建模[30]。根据爬虫原理,GUI crawling 技术可以执行崩溃测试和回归测试,crawler 创建的模型是一棵 GUI 树,每条结点代表一个用户界面,每个边代表界面间的转换,从而实现 GUI 建模[22,29]。基于 GUI 事件的建模方法多数是建模与测试同时进行的过程,这些方法的优点是比较直观,只要能驱动程序执行即可,缺点是路径覆盖度过低。

以上都是黑盒建模方法,这种方法大多依赖于静态分析,也有许多建模工作是基于动态分析和源代码的[31-33]。Saswat Anand 等[34]提出了建立 Android 系统的 GUI 树模型,该方法在源代码中建立每个 Activity 的树模型,首先找出程序中存在的所有事件,然后基于 GUI 树模型将获得的单个事件进行组装以生成事件序列。Wontae Choi 等提出了 SwiftHand 工具,它是一种提高测试覆盖度的补救措施,最大的特点是采用机器学习的方法构建程序模型。它根据收集的执行路径来学习一个模型,使用了机器学习的主动学习和被动学习算法,但这个模型是近似的,因为它只被用来指导生成部分测试用例,覆盖未被执行到的部分[35]。针对源代码的模型研究,还有许多学者尝试将其他平台的已有工具引入到移动应用领域,例如将著名的 Java 模型检验工具 JPF 引入 Android 领域。在符号执行的 Android 建模中,文献[36]提出了使用 JPF 自动为 Android 程序建模,通过得到 Java 字节码而不是使用 Android 虚拟机来实现 Java 编译器编译 APP,通过 JPF 为程序生成调用图模型,最后将调用图模型转换为上下文无关文法生成事件序列。另外,JPF 拥有强大的功能和扩展点,JPF-Android[37]企图扩展 JPF-AWT 包来实现 Android 平台的模型检验,由于 Android 在 DVM 上运行,有别于 Java 的 JVM,文献[20,37]逐步扩展 Android 包,保证 Android 程序能够运行在 Java 虚拟机上,对 Android 程序执行模型检验。

Smali Code[38]是 Google 开源项目,目前许多研究领域已从 Smali 着手开展工作[39-41],但主要还是集中在安全领域。文献[42-44]发现 Android 程序中的恶意代码,文献[45-46]研究 Android 系统上的权限和用户隐私问题。以上研究工作虽然侧重点和方法都各不相同,但涉及 Smali Code 时用法都是一致的,即在 Smali Code 上做静态分析。本次首次利用 Smali Code 重构移动应用的行为模型,并在此基础上研究基于模型的移动应用维护和测试。

结束语　本文研究面向移动应用的模型构建方法,构建移动应用行为模型。通过逆向工程得到可执行文件的中间表示 Smali Code,Smali Code 与源代码完全对应,在 Smali Code 上构建状态机模型可以提高模型的准确度和覆盖度;研究方法从 Intent 入手,通过动态与静态相结合的方法生成事件表;建模时通过在系统层扫描屏幕变化来判断是否出现新状态,不再需要人工为界面定义状态向量。该方法既可以提高模型的覆盖度,又可以摆脱建模对源代码的依赖。实例研究针对 S3TV 生成了有限状态机模型,该模型覆盖了大多数状态和事件,验证了基于 Smali Code 生成有限状态机的方法是完全可行的。

研究工作主要着眼于移动应用测试的模型构建,研究理论有明显的创新性,实验结果达到了预期的目标;但在理论和实现方面仍有很大的上升空间,未来工作主要围绕两方面进行。

(1)弥补不足。工作中暴露出了一些问题,这些问题在很大程度上制约了工具的生成。1)完整性问题:Smali Code 代码的分析还不能完整地包含所有语法,Intent 的分析还不能包含部分隐式类型,这会缩小工具的使用范围;2)产生的新问题:我们结合系统层来判断状态变化,虽然节省了很多手工工作量,但是也牺牲了场景判断能力,例如在实验中,最终生成的状态机混淆了状态 $S_{3\text{-}1}$ 和 $S_{3\text{-}2}$;3)自动化程度:在本文的可用性测试研究中,将事件记录系统插桩到源代码中还没有完全实现自动化。

(2)扩充工作。模型构建只是移动应用测试的一部分,未来还要有测试用例生成和测试用例执行,这两部分虽然几乎没有理论创新,但却是一套完整的工具中必不可少的部分。在可用性测试方面,本文的工具需要源代码,未来可以考虑直接在 Apk 上进行插桩。

参 考 文 献

[1] http://www.cnnic.cn.

[2] WASSERMAN A I. Software engineering issues for mobile application development[C]// Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. ACM, 2010: 397-400.

[3] RIDENE Y, BARBIER F. A model-driven approach for automating mobile applications testing[C]// Proceedings of the 5th European Conference on Software Architecture: Companion Volume. ACM, 2011: 9.

[4] UTTING M, LEGEARD B. Practical model-based testing: a tools approach[M]. Morgan Kaufmann, 2010.

[5] YOUNG M. Software testing and analysis: process, principles, and techniques[M]. John Wiley & Sons, 2008.

[6] UTTING M, LEGEARD B. Practical model-based testing: a tools approach[M]. Morgan Kaufmann, 2010.

[7] DALAL S R, JAIN A, KARUNANITHI N, et al. Model-based testing in practice[C]// Proceedings of the 21st International Conference on Software Engineering. ACM, 1999: 285-294.

[8] BARNETT M, GRIESKAMP W, NACHMANSON L, et al. Towards a tool environment for model-based testing with AsmL [M]// Formal Approaches to Software Testing. Springer Berlin Heidelberg, 2004: 252-266.

[9] GRIESKAMP W, GUREVICH Y, SCHULTE W, et al. Generating finite state machines from abstract state machines[J]. ACM SIGSOFT Software Engineering Notes, 2002, 27(4): 112-122.

[10] CAMPBELL C, VEANES M. State Exploration with Multiple State Groupings[M]// Abstract State Machines. 2005: 119-130.

[11] CAMPBELL C, VEANES M, HUO J, et al. Multiplexing of partially ordered events[M]// Testing of Communicating Systems. Springer Berlin Heidelberg, 2005: 97-110.

[12] GRIESKAMP W, TILLMANN N, VEANES M. Instrumenting scenarios in a model-driven development environment[J]. Information and Software Technology, 2004, 46(15): 1027-1036.

[13] NACHMANSON L, VEANES M, SCHULTE W, et al. Optimal strategies for testing nondeterministic systems[J]. ACM SIGSOFT Software Engineering Notes, 2004, 29(4): 55-64.

[14] PAIVA A C R, FARIA J C P, TILLMANN N, et al. A model-to-implementation mapping tool for automated model-based GUI testing[M]// Formal Methods and Software Engineering. Springer Berlin Heidelberg, 2005: 450-464.

[15] TAKALA T, KATARA M, HARTY J. Experiences of system-level model-based GUI testing of an Android application[C]// 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST). IEEE, 2011: 377-386.

[16] RIDENE Y, BARBIER F. A model-driven approach for automating mobile applications testing[C]// Proceedings of the 5th European Conference on Software Architecture: Companion Volume. ACM, 2011: 9.

[17] THOMPSON C, WHITE J, DOUGHERTY B, et al. Optimizing mobile application performance with model-driven engineering [M]// Software Technologies for Embedded and Ubiquitous Systems. Springer Berlin Heidelberg, 2009: 36-46.

[18] THOMPSON C, SCHMIDT D, TURNER H, et al. Analyzing mobile application software power consumption via model-driven engineering[J]. Advances and Applications in Model-Driven Engineering, 2013: 342.

[19] AMALFITANO D, FASOLINO A R, TRAMONTANA P, et al. MobiGUITAR: Automated Model-Based Testing of Mobile Apps[J]. Software, 2015, 32(5): 53-59.

[20] VAN DER MERWE H, VAN DER MERWE B, VISSER W. Execution and property specifications for JPF-android[J]. ACM SIGSOFT Software Engineering Notes, 2014, 39(1): 1-5.

[21] JHA A K, LEE W J. Capture and Replay Technique for Reproducing Crash in Android Applications[C]// Proceedings of the 12th IASTED International Conference in Software Engineering. 2013: 783-790.

[22] GIANAZZA A, MAGGI F, FATTORI A, et al. Puppetdroid: A user-centric ui exerciser for automatic dynamic analysis of similar android applications[J]. arXiv preprint arXiv: 1402. 4826, 2014.

[23] GRIEBE T, GRUHN V. A model-based approach to test automation for context-aware mobile applications[C]// Proceedings of the 29th Annual ACM Symposium on Applied Computing. ACM, 2014: 420-427.

[24] MARCHETTO A, TPMELLA P, RICCA F. State-based testing of ajax web applications[C]// 2008 1st International Conference on Software Testing, Verification, and Validation. IEEE, 2008: 121-130.

[25] YANG W, PRASAD M R, XIE T. A grey-box approach for automated GUI-model generation of mobile applications[M]// Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2013: 250-265.

[26] RAUT P, TOMAR S. Android Mobile Automation Framework [OL]. http://oaji.net.articles/2015/887-1427185326.pdf.

[27] WALL T. Abbot framework for automated testing of Java GUI components and programs[OL]. http://abbot.sourceforge.net/doc/overview.shtml.

(下转第 231 页)

[13] AGRAWAL R,SRIKANT R. Mining Sequential Patterns[C]// IEEE 29th International Conference on Data Engineering. 1995: 3-14.

[14] SRIKANT R,AGRAWAL R. Mining Sequential Patterns:Generalizations and Performance Improvements[C]// International Conference on Extending Database Technology:Advances in Database Technology. Springer-Verlag,1996:1-17.

[15] TAN L,ZHANG X,MA X,et al. AutoISES:Automatically Inferring Security Specifications and Detecting Violations[C]// Conference on Security Symposium. 2008:379-394.

[16] TAN L,YUAN D,KRISHNA G,et al. /* iComment:Bugs or Bad Comments? */[C]// ACM Symposium on Operating Systems Principles 2007 (SOSP 2007). Stevenson, Washington, USA,2007:145-158.

[17] TAN L,ZHOU Y,PADIOLEAU Y. aComment:Mining Annotations from Comments and Code to Detect Interrupt Related Concurrency Bugs[C]// Proceeding of International Conference on Software Engineering. 2011:11-20.

[18] YU X M,LIANG B,CHEN H,et al. Path-Sensitive Multi-Variable Access Correlations Mining and Related Source Code Defects Detection[J]. Pattern Recognition and Artificial Intelligence, 2012,25(4):691-698. (in Chinese)
于秀梅,梁彬,陈红,等. 路径敏感的源码关联变量模式挖掘及缺陷检测[J]. 模式识别与人工智能,2012,25(4):691-698.

[19] ENGLER D,CHEN D Y,HALLEM S,et al. Bugs as Deviant Behavior:a General Approach to Inferring Errors in Systems Code[J]. ACM Sigops Operating Systems Review,2010,35(5): 57-72.

[20] MAFFORT C,VALENTE M T,BIGONHA M,et al. Mining Architectural Patterns Using Association Rules[C]// International Conference on Software Engineering and Knowledge Engineering. 2013:375-380.

(上接第 220 页)

[28] MOLINARI D H,STAMBAUGH M A,CAIN P J. Apparatus for testing cellular base stations:U. S. Patent 6,308,065[P]. 2001.

[29] AMALFITANO D,FASOLINO A R,TRAMONTANA P. A gui crawling-based technique for android mobile application testing [C]// 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE,2011:252-261.

[30] WANG P,LIANG B,YOU W,et al. Automatic Android GUI Traversal with High Coverage[C]// 2014 Fourth International Conference on Communication Systems and Network Technologies (CSNT). IEEE,2014:1161-1166.

[31] CHUN B G,IHM S,MANIATIS P,et al. Clonecloud:elastic execution between mobile device and cloud[C]// Proceedings of the Sixth Conference on Computer Systems. ACM, 2011: 301-314.

[32] ENCK W,GILBERT P,HAN S,et al. TaintDroid:an information-flow tracking system for realtime privacy monitoring on smartphones[J]. ACM Transactions on Computer Systems (TOCS),2014,32(2):1-29.

[33] FELT A P,CHIN E,HANNA S,et al. Android permissions demystified[C]// Proceedings of the 18th ACM Conference on Computer and Communications Security. ACM,2011:627-638.

[34] ANAND S,NAIK M,HARROLD M J,et al. Automated concolic testing of smartphone apps[C]// Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM,2012:1-11.

[35] CHOI W,NECULA G,SEN K. Guided gui testing of android apps with minimal restart and approximate learning[J]. ACM SIGPLAN Notices,2013,48(10):623-640.

[36] MIRZAEI N,MALEK S,PĂSĂREANU C S,et al. Testing Android apps through symbolic execution[J]. ACM SIGSOFT Software Engineering Notes,2012,37(6):1-5.

[37] VAN DER MERWE H,VAN DER MERWE B,VSSER W. Verifying android applications using Java PathFinder[J]. ACM SIGSOFT Software Engineering Notes,2012,37(6):1-5.

[38] JESUS F. Smali,an assembler/disassembler for Android's dex format[OL]. http://code.google.com/p/smali.

[39] ZHENG M,LEE P P C,LUI J C S. ADAM:an automatic and extensible platform to stress test android anti-virus systems[M]// Detection of Intrusions and Malware,and Vulnerability Assessment. Springer Berlin Heidelberg,2013:82-101.

[40] APVRILLE A,STRAZZERE T. Reducing the Window of Opportunity for Android Malware Gotta catch'em all[J]. Journal in Computer Virology,2012,8(1/2):61-71.

[41] SPREITZENBARTH M,FREILING F,ECHTLER F,et al. Mobile-sandbox:Having a deeper look into android applications [C]// Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM,2013:1808-1815.

[42] BATYUK L,HERPICH M,CAMTEPE S A,et al. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications[C]// 2011 6th International Conference on Malicious and Unwanted Software (MALWARE). IEEE,2011:66-72.

[43] ZHENG C,ZHU S,DAI S,et al. Smartdroid:an automatic system for revealing ui-based trigger conditions in android applications[C]// Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. ACM,2012: 93-104.

[44] KOVACHEVA A. Efficient Code Obfuscation for Android [M]// Advances in Information Technology. Springer International Publishing,2013:104-119.

[45] BERTHOME P,FECHEROLLE T,GUILLOTEAU N,et al. Repackaging android applications for auditing access to private data[C]// 2012 Seventh International Conference on Availability,Reliability and Security (ARES). IEEE,2012:388-396.

[46] STEVENS R,GIBLER C,CRUSSELL J, et al. Investigating user privacy in android ad libraries[C]// Workshop on Mobile Security Technologies (MoST). 2012.