软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于扩展的 IFML 的安卓应用的建模与测试 *

陆一飞, 潘敏学, 张 天, 王林章, 李宣东

(南京大学 计算机软件新技术国家重点实验室, 江苏 南京 210023)

摘 要: 随着智能机以及平板电脑的普及,安卓终端逐渐成为日常生活中不可或缺的重要元素之一,安卓端应用也随之蓬勃发展,其应用复杂度也呈几何倍数增长.作为软件生命周期中最为耗时的步骤之一,测试一直是软件开发工作的一项重要工作.由于人工测试太过耗时难以满足工业要求,越来越多的人将目光转向了自动化测试.但目前的自动化测试自动化程度还比较低;尤其安卓应用大量的操作都集中在图形化界面上,加大了自动化测试的难度.现存在较多基于 UML 模型针对应用内部事务逻辑的测试用例生成工具,然而这些工具生成的测试用例通常为简单的数据输入操作,难以符合图形化界面测试的要求.故此,相应的安卓应用测试的自动化程度普遍比较低.本文采用交互流建模语言(IFML)来进一步提高测试的自动化程度,我们针对安卓平台的特点,对交互流建模语言进行了相应的扩展,提高了其可用性与对安卓应用的适用性.另外,本文以该扩展后的交互流建模语言为模型,采用近年来备受关注的基于模型测试方式,提出了基于扩展的 IFML 模型的测试方法,并在此基础上实现了相应的原型工具.该工具能够有效地使用 IFML 为安卓应用建模,按照所设定的测试覆盖准则生成符合要求的测试序列并自动执行. 实验结果表明,无论在测试覆盖度还是缺陷发现能力上该工具都有很好的表现.

关键词: 安卓应用;基于模型测试;交互流建模语言;测试用例生成
中图法分类号: TP311

中文引用格式: 陆一飞,潘敏学,张天,王林章,李宣东.基于扩展的 IFML 的安卓应用的建模与测试.软件学报. http://www.jos.org.cn/1000-9825/ 0000.htm

英文引用格式: Lu YF, Pan MX, Zhang T, Wang LZ, Li XD. Modeling and Testing for Android Applications Based on the Extended Interaction Flow Modeling Language (IFML), 2017 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

Modeling and Testing for Android Applications Based on the Extended Interaction Flow Modeling Language

LU Yi-Fei, PAN Min-Xue, ZHANG Tian, WANG Lin-Zhang, LI Xuan-Dong

(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China)

Abstract: Under the widespread adoption of smartphones and tablets, mobile termination gradually becomes one of the most important elements in our daily life. Along with it, mobile applications are now flourishing and developing, and so are their complexity. As the most time consuming phase in Software Development Life, test is one of the important works of the software engineering. Nowadays, more and more software engineers turn to the automated testing for the manual testing is time-consuming and not comprehensive for the software industry. However, the automation degree is still not enough for our need, especially in mobile respect for most mobile operations are concentrated on GUI which increases the difficulty. In recent years more and more UML-based test case generation tools spring up, which focus on the business logic inside the applications. Nevertheless, the test cases that these tools generate are usually simple data entry operations which have a few links with the GUI and cannot meet the need of the GUI test. As a result, corresponding test case generation is still man-made, labor intensive and time-consuming. This paper uses the Interaction Flow Modeling Language (IFML) to deal with the above-mentioned problems with extending the IFML for mobile application elements, especially Android ones to improve its availability and applicability for mobile. Whatsmore, this paper develop a model-based test case generator based on the IFML, which generally means

* 基金项目: 国家自然科学基金(00000000, 00000000);
Foundation item: National Natural Science Foundation of China (00000000, 00000000);
收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00
CNKI 在线出版时间: 0000-00-00

the automated generation of the test cases. This work finally paves the way for the continuing improving of automation and the new method for mobile automated testing, which is of far-reaching significance.

Key words: mobile application;Android;model based test;Interaction Flow Modeling Language;test case generation

随着智能机与平板电脑的日渐普及,安卓设备以及其应用市场蓬勃发展.近年来在谷歌公司的推广下,安卓(Android)这一开源操作系统被广泛应用于移动智能终端.根据 Gather 公司对手机市场的调研[1],截止至 2016 年第四季度,安卓系统在智能手机市场的占有率已经达到 8 成;此外,至 2017 年年初,单在 google play 应用商店中可获得的安卓应用已超过 270 万[2],月度活跃用户已达 20 亿.毫无疑问,安卓设备以及安卓应用逐渐成为移动计算市场的主流.安卓应用在拥有越来越丰富的功能的同时还需要应对因其独有的元素引起的各种各样的操作环境的变化[3].这导致其程序复杂度也在呈现指数级的增长;同时,为了能更好地抢占市场,安卓应用需要适配尽可能多的设备,而安卓设备和系统存在多样化和碎片化的现象.这些因素综合在一起,使得相关的开发与测试工作量大大增加了.

模型驱动工程(MDE)在一定程度上能较好地解决上述问题.MDE 的主要思想是将问题以模型的方式来描述,从而基于模型来实现系统.它不仅能帮助开发者将关注点置于系统本身的逻辑,以便更好地理解系统,还能降低软件产品对变化的敏感度.额外的建模工作则由后期有效的开发测试过程来补偿[5].MDE 的优势在安卓应用的开发更迭过程中尤为明显.安卓设备和系统存在多样化和碎片化的现象,同一个应用经常需要开发多个版本,使得主流的安卓系统版本和设备都能运行.而 MDE 天然地能将系统中变化与不变的部分相分离,使得应用的逻辑设计这一类不变的部分能得到充分复用.目前,国际上已有学者针对这一主题进行了相应的研究,如采用 UML 类图和时序图作为移动应用的高层抽象模型,并基于这些模型实现了代码的部分自动化生成[4],或是针对数据驱动的应用,基于 md^2 这一 mvc 结构模型进行移动端的模型驱动开发[5].然而这些模型都是为传统的 PC 应用设计的,没有考虑安卓应用本身的特点.与传统的 PC 应用不同,安卓应用的执行基本是事件驱动的,大量依赖于图形化用户界面(GUI)[6].这就需要能有效对 GUI 建模的语言.交互流建模语言(Interaction Flow Modeling Language,简称为 IFML)作为面向对象组织 OMG 的年轻标准,倡导以图形的形式描述软件系统前端设计中的展现内容及与其用户之间的交互.IFML 能够与其他基于 UML 的模型兼容交互,有良好的可扩展性与兼容性.它以事件流为驱动,重视用户和系统事件以及人机交互,适合安卓终端.因此,本文将研究使用 IFML 对安卓应用建模的方法.

另一方面,基于模型的测试是模型驱动工程中的重要一环.对于安卓应用来说,目前在工业界中实际使用较为普遍的测试办法仍然以人工测试为主,存在着资源消耗大,推迟产品发布,测试覆盖不完全等问题,为此实现测试自动化势在必行.尽管已有一些安卓应用的自动化测试工具,如 MonkeyRunner[7],Appium[8]和 Mobile Test[9]等,这些工具虽然实现了测试用例的自动化执行,但重要的测试用例仍然需要人工生成,这是一个消耗大量人力资源的工作,也是目前进一步提高自动化测试程度的瓶颈.而基于模型的测试能有效解决这些缺陷.基于模型自动生成的测试用例覆盖率更高,配合相应的自动化测试框架能够最大化地提高测试的自动化程度,有效地降低人工测试所带来的巨大消耗.然而目前许多基于模型的测试方法却无法适用于安卓应用.本文也将研究基于 IFML 模型对安卓应用测试的方法.

本文的工作与贡献主要包括四个方面. 首先, 本文在研究调查大量已有的安卓应用之后,总结了安卓前端界面中常用的控件和控件组,并以该知识为基础,提出了面向安卓平台的 IFML 扩展,以更好地对安卓前端进行建模.第二,对于 IFML 标准中非形式化的部分,以及本文提出的扩展部分,我们都给出了形式化的定义.这有利于 IFML 模型的准确分析和理解,也为基于 IFML 的基于模型的开发和测试奠定了理论基础.第三,基于扩展的 IFML 模型的形式语义,给出了遍历 IFML 模型的算法,从而能以不同的测试准则生成测试用例.最后,结合测试脚本执行工具,本文实现了安卓应用的基于模型的自动化测试.实验结果表明,该测试方法无论在测试覆盖度还是缺陷发现能力上都有很好的表现.

本文第 1 节从总体上介绍了 IFML 的特点和它的三个部分交互流模型,域模型以及视角,并介绍了交互流模型的主要组成.第 2 节以一个安卓应用为例描述了当前移动端自动化测试的不足,说明扩展 IFML 以用于基于模型测试的必要性.第 3 节通过数据统计,总结了安卓应用前端的总体特点,从而针对性地对 IFML 进行安卓平台

的扩展,并最终用形式化语言定义了扩展后的 IFML 模型.第 4 节在第 3 节定义的基础上提出了通过遍历 IFML 模型中的可执行路径,生成符合测试要求的测试序列并自动化化执行的测试方法.第 5 节则介绍针对上述测试方法进行的对比实验的设计,以及结果的分析.第 6 节列举模型驱动开发,移动端自动化测试以及对 IFML 扩展的已有工作进行评价.第 7 节总体全文,提出未来的研究方向和改进目标.

1 交互流建模语言

交互流建模语言(Interaction Flow Modeling Language,简称为 IFML),是以图形的形式描述软件系统前端设计中的展现内容,与用户之间的交互,以及行为控制等内容的可视化建模语言[10].IFML 在 2013 年 3 月被 OMG 协会纳为标准,并在 2015 年 3 月发表了 1.0 官方正式文档.

IFML 的主要目的是给软件工程师们提供用以描述软件应用前端的 IFML 模型以及其中的模型元素.根据 IFML 标准,一个图形化界面应用的前端可由一个 IFML 模型(IFMLModel)来表示.IFML 模型是 IFML 中最大的概念,其下又包含 3 个部分:交互流模型(InteractionFlowModel), 域模型(DomainModel)以及视角(ViewPoint).

1) 交互流模型(InteractionFlowModel)是 IFML 模型的重点,它用交互流模型元素(InteractionFlowModel-Element)抽象代表了应用中各前端元素,以及各元素之间的交互关系.
2) 域模型(DomainModel)代表了 IFML 模型中对应的逻辑域视图,实际为一个域元素(DomainElement)的集合.它们通常以 UML 模型元素的形式组成,用于表示 IFML 交互流模型中元素所使用到的数据结构以及具体的行为逻辑.
3) 视角(Viewpoint) 表示在不同环境下可获取的 IFML 模型的某一功能性的部分.一个 IFML 模型可拥有多个视角,每个视角由其环境元素(使用设备,位置,用户角色等)和该交互流模型中的一组相互关联的交互流模型元素所组成,

大体而言, 作为一个 IFML 模型最为重要的部分,交互流模型中存在多个视图容器(ViewContainer),代表应用/系统中的界面或视图.每个视图容器可以拥有属于自己的视图组件(ViewComponent)来表示该界面上出现的控件元素,也可以嵌套其它子视图容器用以表示该界面上的子界面.在视图组件上又可包含视图组件部件(ViewComponentPart)来对该视图组件进一步补充说明.在视图容器/视图组件中,可以拥有参数元素(Parameter)表示该部分所持有的参数,并且参数的数值可参与表达式(Expression)的计算(关于表达式的使用可见下述章节).此外,在视图容器/视图组件/视图组件部件上也可持有事件元素(Event),表示在其上可触发的事件.当事件被触发时,交互流(InteractionFlow,包含导向流 NavigationFlow 和数据流 DataFlow)会被从该事件导出,导向其他的元素,用于表示视图焦点或是数据的转移.与此同时,交互流上的参数绑定组(ParameterBindingGroup)将会被执行,进行参数数值的传递.特别地,当导向流指向行为元素(Action)时,行为中的逻辑将会被执行,完成后将会触发其上的行为事件(ActionEvent),从而引出新的交互流指向其它类型的元素.

在 IFML 标准中,交互流模型是 IFML 模型的主体,而域模型为其服务,视角则是在它基础上进行的扩展.显而易见, 交互流模型是用户用于建模的主要部分,因此本文重点研究面向安卓应用的交互流模型,不再对域模型和视角进行详细介绍,其具体内容可参阅 IFML 标准文档.

IFML 标准可以支持多种终端上的应用,如台式电脑,平板电脑,笔记本电脑,安卓手机,掌上电脑等.使用上述的 IFML 元素可根据选择的应用建立起其对应的 IFML 模型,该 IFML 模型从使用者的视角出发,描述的重点在于终端用户所理解的应用的前端结构及与之相关的前端交互行为.

2 驱动案例

本节将介绍一个来自 github 上的安卓开源应用 Good Weather,用于讨论对 IFML 标准进行面向安卓扩展,并在此基础上自动化测试的必要性.

Good Weather 是一个在安卓框架下开发的天气信息展示应用.该应用可在 github 上获取,其发布的 apk 应用文件也可在 google play/apkture 上获取.

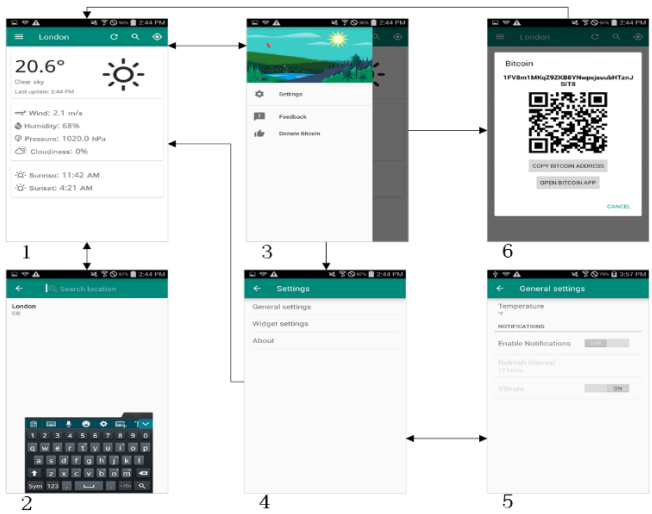

Fig.1 Six views of Good Weather and their relations

图 1 Good Weather 应用的 6 个视图及它们之间的关系

图 1 显示了 Good Weather 这一安卓应用 GUI 的实际视图,以及各视图之间的跳转关系,可以看到,Good Weather 应用总体而言虽然逻辑简单,但含有较多的设置以及相应的界面跳转,故采用手工测试会消耗一定量的人力资源.为此我们尝试简单的自动化测试工具 monkey 来对 Good Weather 进行测试:在长达 750 秒的测试下,共执行了 10000 条事件流.在执行过程中触发了 3 次应用崩溃,均体现在输入关键字进行搜索的情况下报出空指针的异常,为此我们认为在输入字符显示地点列表处存在缺陷.然而在观察 monkey 发送随机的事件流进行测试的过程中,我们也发现了该方法存在了一定缺点:

1. 应用测试过程中虽然执行了 10000 个事件,但其中存在将近 80%的无效操作,例如点击无事件的图片,图标,文字等,效率低下;而部分面积小却含有重要事件的图片/按钮/图标覆盖几率较低;
2. 应用中存在会跳转至其他应用的事件,例如在 Good Weather 应用中的 Feedback 功能(可见图 1-3)将会打开设备的邮件应用,只有当发送完邮件或是退出邮件应用后才能返回 Good Weather 进行进一步的测试,当触发这些事件时 monkey 很难再返回至原应用,更不用提完成期待的相应操作返回原应用;
3. 部分事件 monkey 难以触发,如 monkey 虽然设计了滑动的事件,但由于其随机性,部分特定界面的滑动事件难以触发,例如在 Good Weather 应用中划出侧边栏要求滑动事件起始点极端靠近屏幕边缘,在 monkey 随机事件流下触发概率极低,在实测的 10000 个事件过程中,并未能通过滑动触发侧边栏这一事件.此外 monkey 也有许多操作如长按,双击以及部分重要系统事件等未有进行设计.
4. monkey 测试由于其随机性,导致测试时较为集中在某些界面,例如在 Good Weather 中的主界面(图 1-1)上有大量文本内容,其上并没有可触发的事件,且其工具栏上的刷新与 GPS 定位按钮并不会引起界面的变化,所以在本测试中大量的测试时间被消耗在主界面上,这导致了测试覆盖分布不均,部分较深界面测试覆盖不够.

针对以上问题,一种很自然的想法就是使用模型驱动开发过程中的应用模型,来指导测试用例的生成.然而,若是直接以 IFML 标准为应用建模并在此基础上进行测试,会面临以下难点:

1. 安卓应用由于其物理屏幕面积较小的原因,其所使用的的视图容器,控件以及排版结构等往往与其他终端上的应用有较大差别.同时应用中界面元素种类繁多,如广域的界面区域有工具栏(Toolbar)等,小型的界面控件则有图像视图(ImageView),按钮(Button),本文视图(TextView)等,甚至还有独立于应用的界面如通知栏(Notification Area)等.仅仅使用 IFML 标准中的视图容器和视图组件(以及它们的若干扩展子类型)来建模难以区分各类控件,会给基于模型的测试中控件的识别和定位带来困难;

2. 安卓应用具备丰富的用户操作类型,导致应用中可触发的事件种类繁多.例如用户操作所触发的事件,在应用中有左右滑动,上下滑动,触摸,长按等不同类型,单一的视图元素事件(ViewElementEvent,为事件的一个子类型)难以准确地描述,在测试时更难以确定并进行操作模拟.
3. 安卓应用往往还需处理安卓系统多样的系统事件情况,如电量不足,无线连接中断,传感器信号等. 在 IFML 中,这些事件都用同一个类型系统事件(SystemEvent,事件的一个子类型)来表示,无论对于建模还是测试都极为不合适.

除此以外,安卓应用作为移动设备,能引起的特殊行为,如话筒,相机相关的行为,并且安卓平台的碎片化,即设备繁多,品牌众多,版本各异,分辨率不统一也说明使用原 IFML 模型对安卓应用的建模和测试是不足够的,并极力驱使我们对 IFML 进行安卓终端的扩展.

为此本文拟对 IFML 标准进行面向安卓端的扩展,并在此基础上对安卓应用进行基于模型的测试.该方法更利于实际生产,能以更高的效率以及覆盖率生成高质量的测试用例并自动化执行,有效降低资源的消耗.同时该方法也能与模型驱动开发相结合,合理复用人工建立的 IFML 模型.

3 面向安卓平台的 IFML 扩展

本文面向安卓平台对 IFML 标准进行了两方面的工作.首先是针对安卓平台特点进行的扩展,由于安卓平台上的应用与 PC 端上的有较大差异,因此需要针对这一部分的差异进行扩展,以便使 IFML 更好地适应安卓应用;其次是给出了面向安卓平台的 IFML 标准的形式化语义,由于原 IFML 标准采用了非形式化的描述方法,标准中的许多部分含义不够明确,为了更好地支持基于 IFML 模型的测试,我们对标准中模糊的部分进行了严格的定义,同时也对本文提出的面向安卓平台的扩展部分进行了形式化定义.

3.1 安卓应用前端的特点

作为移动应用的一大分支,安卓应用与其他的桌面应用,网页应用有着较大的不同.安卓应用依附于安卓设备,由于安卓设备便于携带,屏幕面积小的特征,它所采用的 GUI 控件和界面与其他平台应用类型差异较大.为了更好地实现对 IFML 模型进行面向安卓的扩展,我们调查分析了 50 个主流安卓商业应用,以获取科学数据的指导.这些应用来自 Google Play 热销榜上的 24 个应用分类,包含工具类,医疗类,教育类等.在 Ui automator viewer 的帮助下,对于每一个应用,我们抽取了 3-4 个主要界面,对这些界面上出现的控件(View)和控件组(ViewGroup)进行了统计和分析.表 1 中显示的是对控件组的统计.

Table 1 Statistics for usage of ViewGroups in 50 apps

表 1 50 个应用中控件组使用情况统计

父类型	子类型(本体)	总数	平均出现数	IFML 抽象
--	RecyclerView	68	1.36	List
AdapterView	AdapterView	5	0.1	
	ListView	16	0.32	
	Spinner	16	0.32	
LinearLayout	TableLayout(RadioGroup)	10	0.2	
	SearchView	24	0.48	Form
	TimePicker(DatePicker)	10	0.2	
	ScrollView	29	0.58	Scroller
	HorizontalScrollView	38	0.76	
--	DrawerLayout	22	0.44	Drawer
--	ViewPager	28	0.56	--
--	ToolBar	41	0.82	ToolBar
--	WebView	14	0.28	Web

统计结果基本按照安卓控件包内控件组的继承关系来展示,对于其中每个类型,我们标明了其父类,本身类型,在本次统计中出现的总数,以及在平均每个应用中的平均出现数.我们省略了在应用中平均出现数低于 0.1

的控件组,如 NestedScrollView, ViewAnimator 等.因此,上表中这些标有出现次数的元素即可认为是安卓应用常用的控件组,也是我们应当扩展的主体内容.然而过于细致的扩展会导致 IFML 复杂度大大提高,不利于用户使用和建模,为此我们也根据这些控件组的共性对其进行分类和抽象,并尽量使用 IFML 中已有的类型,即表 1 中最右边一栏——IFML 抽象.RecyclerView, AdapterView, ListView, Spinner, TableLayout 和 RadioGroup 共性在于均包含了若干相似的元素,可由 IFML 中的 List 类型来进行统一表示;TimePicker,DatePicker 和 SearchView 表示为可进行内容输入,提交的控件组,可由 IFML 中的 Form 类型来表示;ScrollerView 和 HorizontalScrollView 表示为可上下,左右滑动的控件容器,因此可以抽象为 Scroller 来统一表示;DrawerLayout 表示为可从左右滑出的侧边栏,鉴于其平均出现数较高且与其他元素有较大区别,将其抽象为 Drawer 来表示,类似的还有 ToolBar 和 WebView.另外 ViewPager 虽然有一定的出现次数,但它表示的是页面内容的切换,而这一特点可由 IFML 中的视图容器的 XOR 属性来实现,故在此没有必要进行额外扩展.

此外,我们以同样的方式统计了应用中独立控件出现的情况,如表 2 所示:

Table 2 Statistics for usage of independent Views in 50 apps

表 2 50 个应用中独立控件使用情况统计

父类型	子类型(本体)	总数	平均出现数	IFML 抽象
ProgressBar	ProgressBar	10	0.2	ProgressBar
	SeekBar	11	0.22	
TextView	TextView	1081	21.62	Text
	CheckedTextView	16	0.32	
	EditText	45	0.9	EditText
Button	Button(Text)	100	2	Button
	CheckBox	22	0.44	CompoundButton
	Switch	28	0.56	
	RadioButton	29	0.58	
	ToggleButton	20	0.4	
	Button(Icon)	27	0.54	Icon
ImageView	ImageButton	223	4.46	
	ImageView(Icon)	448	8.96	
	ImageView(Image)	161	3.22	Image
--	Other	10	0.2	CustomComponent

表 2 中依旧按照安卓视图控件的继承树顺序来进行展示(Button 原本继承于 TextView,但考虑到其独特性以及为了显示方便,我们将其单独分为一类进行分析),并省略了部分出现数量较少的控件元素.在该表中,ProgressBar 和其子类 SeekBar,TextView 和其子类 CheckedTextView 以及 EditText 在应用中的平均出现数均高于 0.1,并且它们与其他元素间的差异性比较显著,我们将其抽象为 ProgressBar,Text 和 EditText.同时,实际统计过程中 Button,ImageView 和 ImageButton 不仅使用率高,而且在使用范围上存在一定的重叠,即部分应用上这三类元素的使用情况较为相似.我们对这三类元素的外观进行了额外的统计和分类,具体可如下表 3:

Table 3 Statistics for appearance of Button, ImageView and ImageButton in these 50 apps

表 3 50 个应用中 Button,ImageView 和 ImageButton 外观展示统计

控件 \ 外观	Icon	Image	Text	Other
Button	27(20.6%)	1	100(76.3%)	3
ImageView	448(73.3%)	161(26.4%)	2	5
ImageButton	213(95.5%)	3	5	2

表 3 中,我们以用户的角度对这三类元素的外观进行了分类,其中:Icon 表示该元素以小型图标的形式展示,Image 表示该元素以大型图片的形式展示,Text 则表示以文字形式展示.可以看到 ImageButton 基本均以小型图标 Icon 的方式展示(95.5%),其他类型的比例基本可以忽略不计.而 Button 和 ImageView 在外观展示上则各自存在 2 种使用方式.Button 以文本按钮方式显示为主(76.3%),但也存在一定比例的(20.6%)图标形式,当他以小型图标方式展示时,其使用方式类似于 ImageButton.ImageView 则也类似,其 73.3%以图标方式展示,此时使用方式

也近似于 ImageButton,同时它仅有 26.4%的情况是以大型图片的方式展示.在 ImageView 中大型图片 Image 方式的比例较低,其很大程度上来源于安卓应用中大型图片的使用率远低于小型图标 Icon.我们将这一额外统计结果也纳入考量范围,将 Button 区分为带小型图标的 Button(Icon)和带文字的 Button(Text);将 ImageView 区分为带大型图片的 ImageView(Image)和带小型图标的 ImageView(Icon).最终我们将 Button(Text)抽象为 IFML 中的 Button 视图组件,ImageView(Image)则抽象为 IFML 中的 Image 视图组件;而 Button(Icon),ImageView 和 ImageButton(Icon)则抽象为 IFML 中的 Icon 视图组件.此外 Button 下的四个子类,CheckBox,Switch,RadioButton 和 ToggleButton 均表示为一个拥有 2 个状态的 Button,并且它们可通过按压来实现状态转换,最终我们将这四类元素用 CompoundButton 来统一表示.最后存在一些直接继承 View 的控件,它们的使用方式与统计中出现的常用控件有较大区别,例如地图和导航类应用中承载地图主体内容的控件,又比如相机摄影应用中显示镜头内容的控件等,我们在 IFML 中为这些使用率低或是只在某一特定类型应用中出现的元素设立了模板 CustomComponent,允许用户为其自定义视图组件.

3.2 面向安卓平台的IFML扩展

3.2.1 针对视图容器(ViewContainer)与视图组件(ViewComponent)的扩展:

视图容器(ViewContainer)代表的是一个整体性的界面,作为一个背景或是布局来显示,本身还可以嵌套包含其他的视图元素(ViewElement).我们为其添加了安卓应用视图容器(AndroidAppContainer)作为安卓端应用中所使用的视图容器子类型,并按照 3.1 章节中的数据统计,在其下添加滑动窗口(Scroller), 抽屉(Drawer),屏幕(Screen),工具栏(Toolbar)和网页(Web)这 5 个实例子类型.滑动窗口表示可上下或左右滑动的视图容器;抽屉指可从左/右滑出的侧边栏;工具栏表示安卓应用中具有导航,显示标题等其他辅助功能的工具栏容器;网页则表示以网页风格展示的视图容器;屏幕表示安卓应用中进行主体的信息展示,与用户交互的整体性界面容器,与其他 4 个元素进行区分.通过对这些安卓应用常用视图容器的特例化,使得 IFML 建模更加精确;同时,它们的使用也能帮助区分应用中的各块区域,从而降低测试过程中控件定位的难度.其具体关系结构可如图 2 所示.

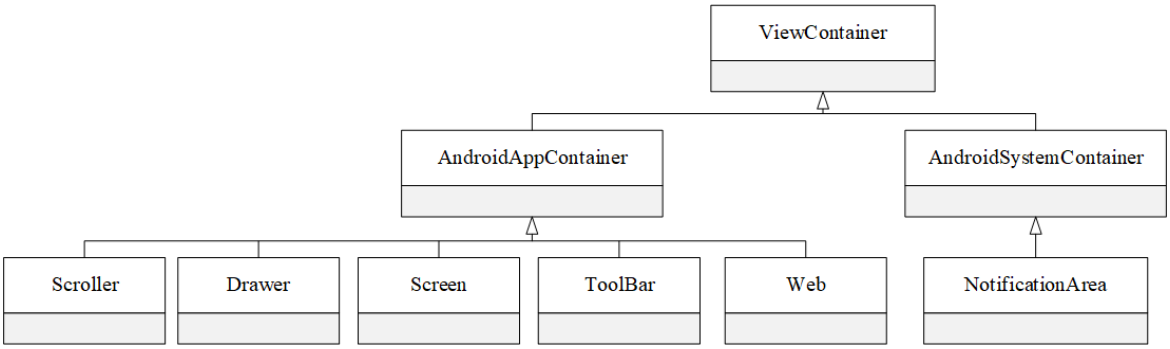

Fig.2 The android extension to ViewContainer

图 2 视图容器(ViewContainer)的安卓端扩展

视图组件(ViewComponent)代表的是视图容器(ViewContainer)界面上的用于展示信息或接受输入的,并可引发应用事件的组件.视图组件不能单独存在,必须依附某个视图容器而存在.原 IFML 标准中所有的组件均以视图组件来表示,过于笼统而显得定义模糊.因此,我们为其添加安卓应用视图组件(AndroidAppComponent)这一子类型表示安卓应用中使用到的视图组件.按照 3.1 节中对独立控件的统计,我们进行了相应的建模:按钮(Button)表示按钮类型的组件,其下有子类型状态改变按钮(CompoundButton)表示会引起状态改变的按钮;文本(Text)表示文本类型的组件;图标(Icon)表示图标组件;图片(Image)表示图片组件;列表(List)和表单(Form)在原 IFML 模型中已有扩展,此处便不再说明;编辑框(EditText)表示为文本输入框;进度条(ProgressBar)表示为进度条组件.为了应对用于特别用途,不属于上述几个分类的控件(即 3.1 中其他类型的控件),我们还添加了模板自定义组件(CustomComponent)允许用户自定义使用到的组件类型.其具体的关系结构可见下图 3.

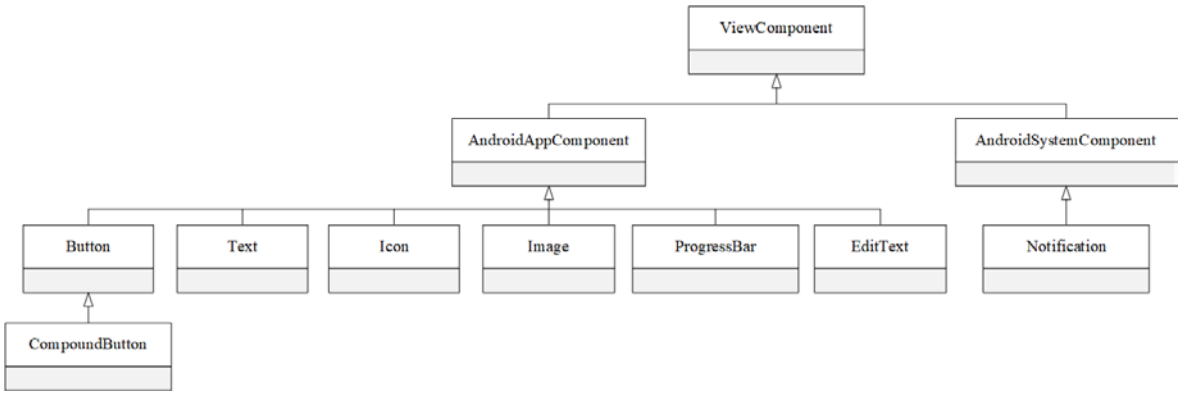

Fig.3 The android extension to ViewComponent

图 3 视图组件(ViewComponent)的安卓端扩展

此外,针对安卓系统上一些独有的界面元素,我们使用安卓系统视图容器(AndroidSystemContainer)与安卓系统视图组件(AndroidSystemComponent)进行表示, 安卓系统视图容器表示安卓端系统提供的视图容器,而安卓系统视图组件表示其系统独有的视图组件.我们亦在其下创建了 1 组实例:通知区域(NotificationArea)表示安卓应用中的通知栏区域元素,通知(Notification)表示其中的通知提示信息.该组实例能够减少对于安卓系统上通知区域建模的劳力,并且由于安卓系统中通知区域的唯一性,该组实例能够与之一一对应.

3.2.2 针对表达式(Expression)与事务行为(Action)的扩展:

表达式(Expression)定义了一个无副作用的语句,可以在执行后返回特定的结果,以便 IFML 模型进行逻辑判断.在 IFMl 标准中存在多种表达式的实用子类型,在本文中主要使用并介绍其中 2 类:交互流表达式(InteractionFlowExpression),它用以判断对应的事件(Event)在当前状况下最终触发的交互流(InteractionFlow)为哪一条;激活表达式(ActivationExpression),用于判断该表达式所属元素是否可见/可触发.

行为(Action)为交互流元素(InteractionFlowElement)的一个可直接使用的子类型,用于表示由事件(Event)所引起的事务逻辑.其拥有动态行为(dynamicBehavior)类型的属性用于表示一个有着可以进行事务逻辑或是返回内容的服务或者行为的系统.该动态行为属于域模型下的域元素,意味着行为的逻辑执行需要域模型的支持.

在 IFML 中,应用逻辑的执行以及是条件的判定是利用表达式以及域模型协同完成的,然而对于通常的安卓应用来说,域模型的使用太过庞大和繁琐,为此本文扩展了表达式的使用,允许将它作为域模型中逻辑的一个轻量级的表示.扩展主要有以下 2 个方面:

1. 定义有副作用的, 即会修改参数数值的表达式子类型默认表达式(DefaultExpression):作为表达式逻辑中重要的参与元素,参数(Parameter)存在表达式类型的属性 defaultvalue,允许其赋予该参数的最初值.然而 IFML 标准中并未定义其对应的表达式子类型,为此本文添加了该默认表达式专门用于以表达式形式直接赋予参数以最初的数值;
2. 定义有副作用的,即会修改参数数值的表达式子类型执行表达式(ExcutionExpression):由上可知行为的逻辑实际上是通过域模型中的 UML 元素,如 UML 活动图,序列图来表示和执行,然而对于轻量级的安卓应用来说太过复杂,增大了建模的范围和难度,为此本文为行为添加新的表达式类属性 exeutioncontent,以表达式类型简单地描述某行为中的具体逻辑,从而不用依靠繁琐的域模型即可完成 IFML 模型中行为的逻辑执行.通过执行 exeutioncontent,它会修改部分参数对应的数值,表示行为中逻辑所带来对模型状态的影响.

此外安卓终端也有许多其他终端所没有的或是不常使用的事务行为,如通话,摄像,照相等;本文还为事务行为添加了安卓终端的扩展.本文扩展了安卓行为(AndroidAction),用以表示安卓应用中的所引发的事务逻辑,由于安卓行为(Action)有行为事件(ActionEvent)与其一一对应,本文亦对行为事件进行了安卓端的扩展,添加了安

卓行为事件(AndroidActionEvent)作为安卓行为对应触发的行为事件.同时在安卓行为下添加了相机行为(CameraAction)这一实用子类型表示照相/录像的事务行为,在安卓行为事件下添加相机行为事件(CameraActionEvent)与其对应;在安卓事务行为下添加话筒行为(MicrophoneAction)表示通话事务逻辑,并在安卓行为事件下添加话筒行为事件(MicrophoneActionEvent)与其对应.这些安卓行为以及其对应的行为事件能够与安卓上的特有行为相对应,减少对该类行为进行建模的复杂过程,并且限定了该行为具体执行的内容以及执行完后对应的行为事件,提高了模型的可用性.

3.2.3 针对事件(Event)的扩展:

事件(Event)为交互流元素(InteractionFlowElement)的子类,用于表示可能会影响该应用系统状态的一个事件.由于安卓设备小巧便携的特点,安卓应用中可能产生以及接收的事件往往会比其他类型的应用更为丰富,应用也会针对设备外界情况变化引起的事件作出相应的响应,如各式各样的屏幕触摸事件,安置在安卓设备上的电池,传感器和 GPS 等引起的事件等.对于事件,本文主要关注其下捕获事件(CatchEvent)这一子分类,其可进一步分为行为事件(ActionEvent),系统事件(SystemEvent)以及视图元素事件(ViewElementEvent).本文对事件的扩展也集中在这三个类型上(对于行为事件的说明和扩展可见上一章节 3.2.2).

系统事件(SystemEvent)代表系统内部产生的事件,本文在其下扩展了新的子类型安卓系统事件(AndroidSystemEvent)用于表示安卓终端上系统产生的事件,同时还在该类型下添加 5 个实用子类型:电池事件(BatteyEvent),表示电池状况所引起的系统事件;存储事件(StorageEvent)表示系统存储引起的系统事件;传感器事件(SensorEvent)表示安卓终端传感器产生的系统事件;通知事件(NotificationEvent)表示安卓终端的信息通知产生的系统事件;连接事件(ConnectionEvent)表示系统连接产生的系统事件,最终的扩展结构可见下图 4:

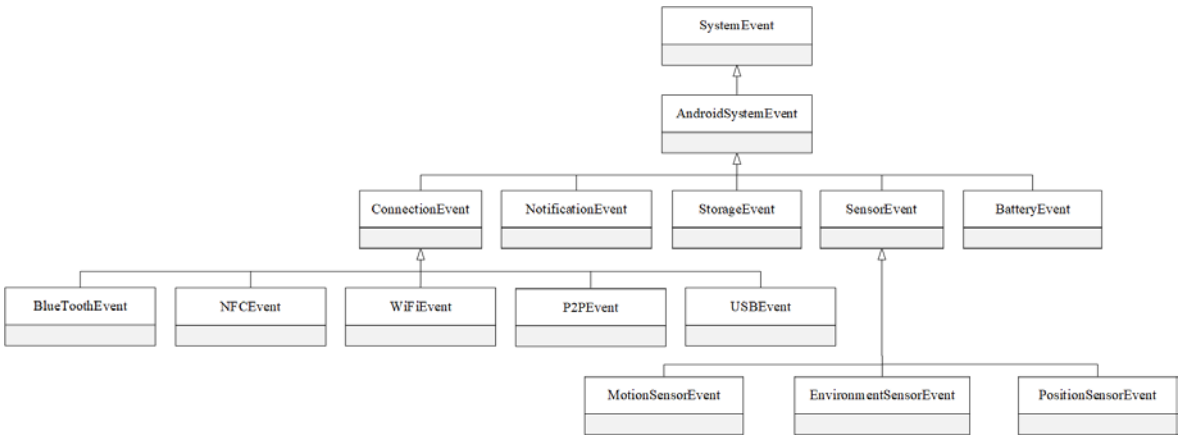

Fig.4 The android extension to SystemEvent

图 4 系统事件(SystemEvent)的安卓端扩展

同时我们还对其中的连接事件和传感器事件进行进一步的划分:连接事件可继续划分为 5 个子类型, 蓝牙事件(BlueToothEvent), NFC 事件(NFCEvent), WiFi 连接事件(WiFiEvent), P2P 事件(P2PEvent)以及 USB 事件(USBEvent);而传感器事件则可细分为 3 个类型,运动传感器事件(MotionSensorEvent),环境传感器(EnvironmentSensorEvent)和位置传感器事件(PositionSensorEvent),其下可继续进行划分,在此则不再详细描述.

视图元素事件(ViewElementEvent)代表用户直接与应用进行的交互事件.安卓端提供了通过压力,电容感知对屏幕的触摸捕捉的支持,与 PC 的鼠标和键盘控制相比有着更丰富的动作与手势,能监听更多类型的事件.为此,本文为视图元素事件添加子类型安卓视图元素事件(AndroidElementEvent)表示安卓终端上用户交互所触发的事件,在其下还增加 9 个实用子类型: 触摸事件(TouchEvent),表示用户轻量级触摸屏幕触发的事件;双击事件(DoubleTapEvent),表示用户在屏幕同一处位置快速点击两次所触发的事件;长按事件(LongPressEvent),表示用户长按所触发的事件;缩捏事件(PinchEvent),表示用户在屏幕上触摸由外向内缩捏或由内向外伸展的行为所触

发的事件;滑动事件(SwipeEvent),表示用户左右滑动所触发的事件;滚动事件(ScrollEvent),表示用户上下滑动所触发的事件,输入事件(OnInputEvent)表示为用户进行文本输入的事件,拖拽事件(DrapDropEvent)表示为用户拖拽行为触发的事件.同时与 3.2.1 中自定义组件相对应,本文也建立了自定义事件(CustomEvent)模板可供用户自定义可触发的事件以及具体的触发行为.最终的扩展结构可见下图 5:

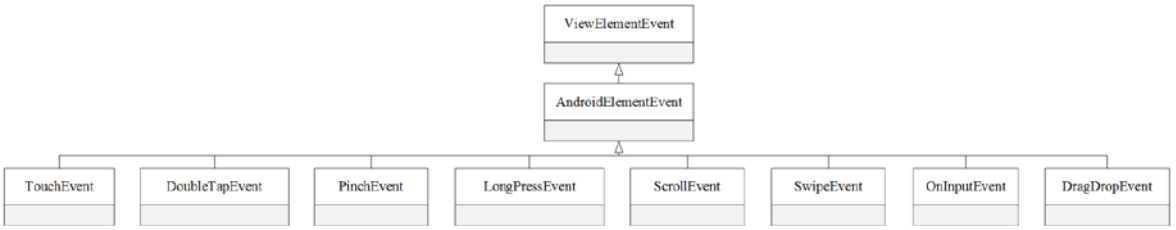

Fig.5 The android extension to ViewElementEvent

图 5 ViewElementEvent 的安卓端扩展

上述事件的安卓扩展,不仅方便复用,还对执行该事件所可能需要的参数进行了限制,初步进行了标准化,给 IFML 建模带来了极大的方便的同时减少了用非形式化语言描述时语义的不一致性和歧义,降低了测试时的难度.

最后,我们还针对 IFML 在 eclipse 上的建模插件进行的扩展.该扩展在修复了一些原本插件所存在的问题的基础上,增添了上述扩展内容,形成一个可用性易用性更高并有针对安卓端部分的 IFML 建模插件.本文于后面章节中均采用该插件工具来进行 IFML 建模.

3.3 IFML模型的形式化定义

基于 IFML 标准文档,一个完整的 IFML 模型均有且仅有唯一的应用或是系统与之对应,主要分为交互流模型,域模型和视角.作为其主要部分的交互流模型,域模型与视角情形均是为其服务,本文关注的是一般安卓应用基于模型的测试,因此在通过 IFML 模型进行安卓端的自动化测试时通常仅需要关注其中的交互流模型,因此下文所指 IFML 模型均只包含交互流模型.

由前文论述可知 IFML 模型较为复杂,故在此我们先给出其中各重要元素的定义.

定义 1. $X=\{x_1, x_2, \ldots, x_n\}$为 IFML 模型中被使用到的表达式集合.对于任意 $x\in X$,x 为一个三元组 x=(xt,l,b),其中 xt 用于标记该表达式的所属的具体类型,即其 4 个子类型;l 表示解析当前表达式所使用的语言,可选 JAVA,OCL 等;b 以文本形式存储了该表达式的表示.在上述 3.2.2 章节已知,X 包含 4 种类型,即 $X=X_A\cup X_I\cup X_D\cup X_E$, X_A,X_I,X_D,X_E 分别表示为激活表达式, 交互流表达式,默认表达式以及执行表达式的集合.

定义 2. $VE=\{ve_1,ve_2, \ldots, ve_n\}$为交互流元素中的视图元素的集合,表示能够展示内容的用户接口的元素.VE 可进一步分为视图容器和视图组件,记为:$VE=VCT\cup VCP$, VCT 表示该交互流模型中视图容器的集合,VCP 表示该交互流模型中视图组件的集合.无论对于视图容器还是视图组件,对任意 $ve\in VE$,均有 ve=(vet,ax,SE,vc),vet 标记了该视图元素的具体类型;$ax\in X_A$ 为该元素对应的激活表达式,SE 表示其上可以触发的视图元素事件的集合(见定义 5);$vc\in VCT$ 为视图容器,表示为该容器/组件的父容器.ax 与 SE 均可为空,且当 ax 为空时,此时该视图容器/组件默认活跃.仅当 $ve\in VCT$,时 vc 可能为空,表示该视图容器为最外层容器.

定义 3. $VP=\{vp_1,vp_2, \ldots, vp_n\}$为交互流元素中的视图组件部件的集合.视图组件部件用于辅助补充视图组件的内容部分,对于任意 $vp\in VP$,均有 vp=(vpt,ax,SE,pvp),vpt 表示该视图组件部件的具体类型;ax 表示该视图组件部分所对应的激活表达式,SE 表示其上可以触发的视图元素事件的集合,$pvp\in VCP\cup VP$,表示该组件部分所属的父视图组件或父视图组件部件.ax,SE 均可为空,特别地当 ae 为空时表示该视图组件部件默认活跃.

定义 4. $A=\{a_1, a_2, \ldots, a_n\}$为交互流元素中的行为的集合.行为代表了一块由事件所触发的可执行的事务逻辑.对于任意 $a\in A$,均有 a=(at,ax,SE,vc,ex),at 表示该行为的具体类型;ax 为该行为所对应的激活表达式;SE 表示从属于该行为并能触发的行为事件的集合;vc 表示为该行为所从属于的视图容器,即父容器,其值可以为空,表示该行为独立于所有的视图容器;$ex\in X_E$ 表示其内部逻辑对应的执行表达式.

定义 5. E={e_1, e_2, …, e_n}为交互流元素中的事件的集合.对于任意 e∈E,均有 e=(et,ax,ix,SI,pe),其中 et 表示为该事件的具体类型;ax 代表该事件对应的激活表达式;ix∈X_I,为交互流表达式(InteractionFlowEpression),其计算值表示该事件在当前情况下会触发哪些交互流;SI 表示为该事件可触发的交互流的集合;pe 表示为该事件所从属的视图容器/视图组件/视图组件部件/行为.其中 ax,ix,pe 均可为空.当 ax 为空表示该事件默认活跃,当 ix 为空时会触发 SI 中所有交互流并且这些交互流中有且仅有一条为导向流.由章节 3.2.3 可知 E 可被分为 3 个类型,记为:E=E_S∪E_A∪E_V,分别表示为系统事件, 行为事件以及视图元素事件的集合.当它为视图元素事件时,其 pe 为视图容器/视图组件/视图组件部件;当它为行为事件时,pe 恒为行为,即若有行为事件 ae=(et,ax,ix,SI,pe),则有 pe∈A.

定义 6. I={i_1,i_2, …, i_n}为交互流模型中使用到的交互流的集合.交互流可分为导向流与数据流,记为: I=N∪D,对于任一 i∈I,均有 i=(it,s,t,PBG).其中 it 表示该交互留具体为导向流还是数据流;s 为触发当前交互流的事件,表示为该交互流的起始点;t 为该交互流的终点,t∈VE∪VP∪A,为视图元素,视图组件部件或行为.特别地,一个行为事件触发的交互流不再导向另一个行为,即任意的行为事件 ae=(et,ax,ix,SI,pe),若有交互流 i=(it,s,t,PBG)使得 i∈SI,则 t∉A. PBG={pb_1, pb_2, …, pb_n}为参数绑定组(ParameterBindingGroup),表示该交互流所携带的数据绑定信息的集合.对于任意 pb∈PB, 其表示为一条参数绑定信息(ParameterBinding),有 pb=(sp,tp),sp 表示为源参数,tp 为目标参数,该一条 pb 表示为在当前状态下将 sp 的值传递给 tp.

定义 7. P 表示为 IFML 模型中被使用到的参数(Parameter)的集合,有 P={p_1, p_2, …, p_n},对于任意 p∈P,其可用一个三元组表示 p=(d, te, dx):d 以枚举类型表示该参数 p 为入参,出参或者两者均是;te 表示该参数的具体类型,dx∈X_D,以默认表达式的形式给定该参数在初始情况下的默认数值.

有了以上定义,我们可以给出 IFML 模型的形式化定义.

定义 8. 一个 IFML 模型为一个六元组 M = (V, A, E, I, X, P, v1),其中:

- V = {v_1, v_2, …, v_n} 为视图元素(视图容器和视图组件)和视图组件部件的集合,即 V=VE∪VP.
- A = {a_1, a_2, …, a_n} 为行为的集合.
- E = {e_1, e_2, …, e_n} 为事件的集合.
- I = {i_1, i_2, …, i_n} 为交互流的集合.
- X = {x_1, x_2, …, x_n} 为表达式的集合.
- P = {p_1, p_2, …, p_n} 为参数的集合.
- v_1∈V 为该 IFML 模型最初情况下显示的视图容器.

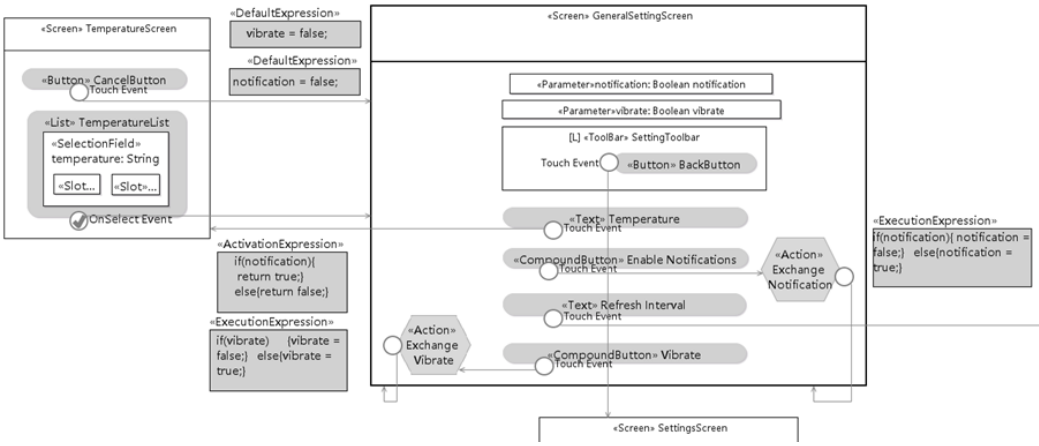

Fig.6 Part of the IFML model of GoodWeather app

图 6 GoodWeather 应用的部分 IFML 模型

图 6 是对第 2 章节中 GoodWeather 应用进行的 IFML 安卓端建模,由于建立后的 IFML 模型较为庞大,此处

仅显示其中的一个重要界面.图中展示的是该 GoodWeater 应用中 GeneralSetting 界面所对应的内容,即 1-5 图中所对应的 IFML 模型部分.该屏幕类型的视图容器 GeneralSettingScreen 即代表了该 GeneralSetting 的界面,其上拥有其对应的总是可见的工具栏 SettingToolbar,其上按钮类型的视图组件 BackButton 可触发触摸事件返回上一个界面 SettingScreen,即 1-4 图中内容.我们为屏幕 GeneralSettingScreen 添加了 2 个布尔型的参数 notification 和 vibrate,用于表示该应用是否启用通知和通知震动,它们的 defaultvalue 属性 (即图 6 左上 2 个 DefaultExpression)将它们的初始值均赋值为 false,表示原应用默认关闭通知和震动.我们可以触发切换按钮 Enable Notifications 和 Vibrate 上的触摸事件,从而执行相应的行为 Exchange Notification 和 Exchange Vibrate,进行对参数 notification 和 vibrate 的修改,以表示通知和震动的启用和关闭.而另外 2 个本文类型组件 Temperature 和 Refresh Interval 上的触摸事件则会进入新的界面,并提供列表类组件进行选择.

在给出 IFML 模型的语法后,为了能基于 IFML 模型生成测试用例,我们还需要给出 IFML 模型的语义,即对 IFML 模型中的执行路径进行定义.IFML 模型虽然与常用的控制流模型类似,使用交互流来表示状态的转化(视图焦点与数据的转移),但由于其复杂性,控制流模型中简单的"点—边—点"模式难以表现出 IFML 模型中交互执行的丰富含义.同时根据 IFML 的语义,是事件触发了交互流,随后导致了状态的转化,交互流无法脱离事件而单独存在,故本文以 IFML 模型中的事件和状态的集合来表示其执行路径.当处于一个 IFML 模型中时,它当前的运行时状态,记为 CS,表示一组激活的视图容器,视图组件,视图组件部件和行为,以及参数和它们当前值的键值映射集合.由此状态 CS 可定义为一个二元组(CM,CA), CM 为激活的视图容器,视图组件,视图组件部件,行为的集合,即 $CM \subset V \cup A$; CA 即为上述的参数和它们当前值的键值映射集合 $P \rightarrow U$,P,U 分别是参数以及实际值的集合,对于任意参数 $p_i \in P$,均有唯一实际值 $u_i \in U$ 与之对应.

为更好地描述状态 CS=(CM, CA),我们又有以下函数的定义:

- 定义函数计算激活表达式来判断元素是否被激活:对于任意元素 $v_i \in A \cup V \cup E$,其激活表达式 ax 有函数 eval(v_i.ax, CA)=b 来计算在 CA 所给定的参数值下该激活表达式的值, b 为布尔值类型,为真时表示该元素 v_i 被激活.
- 定义函数来判断在给定可见元素时其他元素是否可见:对于任意元素 $v_i, v_j \in V \cup A$,存在函数 Lv(v_i,v_j)=b,b 为布尔值类型,为真时表示在元素 v_i 可见时 v_j 也可见.当给定拥有当前状态的视图焦点的可见元素 $v_i \in V$,该元素必定已被激活,即对于 v_i 的激活表达式 ax_i 有 eval(ax_i, CA)=true.此时,v_i 的父容器也必然可见,即对于 v_i 的父容器 pv,有 Lv(v_i,pv)=true 成立.按此定义,该 Lv 函数可继续递归于 pv 的父节点.特别地,当可见元素 $v_i \in A$,有且仅有 v_i 自身可见,即仅有 Lv(v_i, v_i)=true.从而我们可以通过该函数获取视图焦点转移后可见元素的集合,即若 v_i 获得视图焦点(当前可见),则当前所有的可见元素可用集合{$x \in A \cup V$| Lv(v_i, x)=true }来表示.
- 定义函数来执行行为中的执行表达式:对于任意元素 a_i=(at,ax,SE,vc,ex)$\in$A,有函数 exec(ex, CA)=CA', 表示为通过执行 a_i 中的执行表达式内容使得 CA 中元素当前数值得到改变,最终得到新的映射集合 CA'来代替 CA,表示状态参数已经得到更新.
- 定义函数来计算事件中的交互流表达式获取触发的交互流集合:对于任意事件 e_i=(et,ax,ix,SI,pe) $\in$E, 其下交互流表达式 $ix \in X_I$,存在函数 eval(ix, CA)=SI', SI'表示为 ix 所对应的事件(Event)在当前状态下所触发的交互流(InteractionFlow)的集合,并有当 ix 为空时,eval(ix, CA)=SI.
- 定义函数来执行交互流中的参数绑定信息:对于任意交互流 i_i=(it,s,t,PBG),若其 PBG 存在,则有函数 exec(PBG, CA)=CA'表示执行这一值的传递过程,形成新的参数与其值的映射集合 CA'代替 CA 表示状态参数的更新.
- 定义函数计算默认表达式初始化参数的值:对于任意参数 p_i=(d, te, dx) $\in$P,有函数 exec(dx, CA)=CA'表示对该参数的数值进行初始化,CA'中参数 p_i 被赋予了其 dx 中给定的数值.并在初始状态下,给定 CA=$\varnothing$,可对于 IFML 模型中所有的参数 $p_i \in P$,执行 CA$\leftarrow$exec(dx, CA), 最终即可得到其初始的参数与其值的映射集合.

在当前状态 CS=(CM, CA)下,我们又可以定义函数 enable:CM→E, enable(CM)表示为当前状态下活跃元素可触发的事件集合,并对于其中任意 e=(et,ax,ix,SI,pe)∈enable(CM),均需满足 eval(ax, CA)=true,pe∈CM.通过从 enable(CM)中选择任意事件 $e_i=(et_e,ax_e,ix_e,SI_e,pe_e)$后,我们可以通过以下步骤从原状态 CS 变化到新状态 CS', 即 $CS=(CM, CA)\xrightarrow{e_i}CS'=(CM', CA')$:

1. 若 $pe_e\in A$,则需要先执行该行为(记为 $a=(at_a,ax_a,SE_a,vct_a,ex_a)$)中的执行表达式,更新状态,即 $CA_i' = exec(ex_a, CA_i)$.
2. 取 $SI'=eval(ix_e, CA_i')$,由上述定义可知集合 SI'表示为该 e_i 在当前状态下所触发的交互流集合,并且其中的导向流集合可记为 SNI,SNI⊂SI'.对于交互流 $i=(it_i,s_i,t_i,PBG_i)\in SI'$,需要依次执行 $CA'=exec(PBG_i, CA_i')$,即进行参数的传递.
3. 根据第 2 步中导向流集合可重新计算新状态 CS'的 CM':先将 CM 删除其中视图焦点转移过程中失去视图焦点的部分,即 CM'=CM-{ x∈A∪V| Lv(pe_e, x)=true && x∈CM }.随后,添加其获得视图焦点的元素,即遍历选择 SNI 中的导向流 $i_i=(it_i,s_i,t_i,PBG_i)\in SNI$,可知下一状态下获得视图焦点/直接控制流的元素记为 t_i,对其逐个进行 CM'= CM'∪{ x∈A∪V| Lv(t_i, x)=true && x∉CM'}.最终完成遍历后即可获得新状态 CS'下对应的 CM'.

最终我们可给出下述定义 9:

定义 9. 对于任意给定的 IFML 模型 M=(V, A, E, I, X, P, v_1),根据 M 生成的一条 n 步长(n > 0)的可执行路径 ρ 定义为以下一条序列:

$$\rho = CS_0 \xrightarrow{e_0} CS_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{n-1}} CS_n$$

其中,$CS_i(0\leqslant i\leqslant n)$即为上述定义的在该可执行路径 ρ 中的一个运行时状态.特别的,CS_0 表示为该 IFML 模型的初始状态,记为 $CS_0=(CM_0, CA_0)$. 为初始化 CA_0,可先给定 CA=∅,对该 IFML 模型中所有的参数 p=(d, te, dx)∈P,逐次执行 CA←exec(dx, CA), 最终即可得到其初始的映射集合 CA_0.而已知 v_1 表示为该 IFML 模型的初始视图容器,则有 CM_0={ x∈A∪V| Lv(v_1, x) = true && eval(v_1,CA_0) = true},可以获得最初的 CM_0.

从 IFML 模型的初始状态 $CS_0=(CM_0, CA_0)$开始,通过 enable 函数选择一个当前可触发事件执行并更新状态,不断重复上述步骤,我们便可获得一条可执行路径 $CS_0 \xrightarrow{e_0} CS_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{n-1}} CS_n$.

4 IFML 模型测试序列的自动化生成与执行

在对原 IFML 标准进行扩展以及对 IFML 模型中的状态和执行路径进行定义后,本文实现了针对已建立的 IFML 模型自动化测试过程,其具体包含两个过程:1,针对 IFML 模型自动化生成可读的测试序列的集合;2,根据上述生成的测试序列进行自动化测试执行.

4.1 测试序列的自动化生成

本文利用上述改进后的 IFML 模型制作插件,针对待测试安卓应用人工建立 IFML 模型,随后便可通过该 IFML 模型按照事先所选定的测试要求自动生成符合要求的测试序列,其生成过程如图 7 所示:

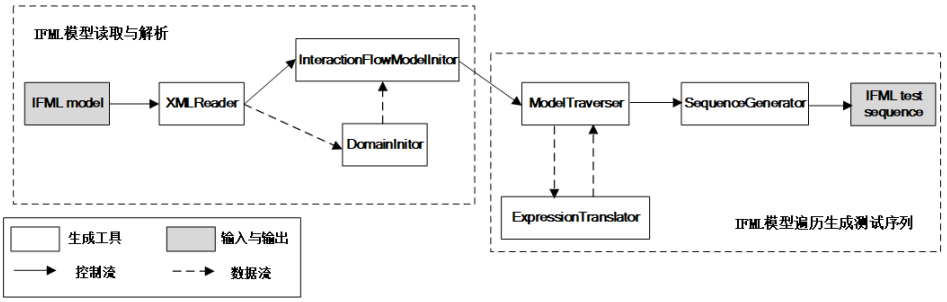

Fig.7 the overview of test sequence generation

图 7 测试序列生成步骤

此处主要分为两个部分:1,IFML 模型读取与解析;2,遍历 IFML 模型生成测试序列.

IFML 模型读取与解析主要为读取上述插件生成的以 XML 格式存储的 IFML 模型,并初始化其中的域模型(包含它可能依赖的 UML 模型,用于解析表达式中使用到的数据类型)与交互流模型,其过程较为简单故在此不详细描述.

本节主要部分即为遍历该 IFML 模型,生成测试序列,其大体步骤为:从 IFML 模型的初始状态开始,遍历选择其下可触发的事件到达新的状态,将新状态代入上述步骤进行递归,我们将会收集在此递归过程中所经历的可执行路径,并生成对应的测试序列.

然而,在上述遍历 IFML 模型过程中,存在状态爆炸,可执行路径数量过多的问题.为此,本文对执行路径的生成进行以下的约束,并按照特定的准则从中选取有较高价值的路径生成对应的测试序列:

1. 执行路径最大事件数(ME):本文将 ME 表示为执行路径 ρ 中其按序的事件序列{e_0, e_1, …, e_{n-1}}的最大长度,此参数的设立为最大程度避免状态爆炸带来的遍历时间太长的问题.
2. 执行路径最大重复事件数(MDE): 本文将 MDE 定义为允许在单条可执行路径 ρ 中某事件 e 重复出现,即执行同一步骤的最大次数.由上述定义 9 可知,执行路径 ρ 以 CS $\xrightarrow{e}$ CS 形式定义,由于检测状态 CS 是否等价难度远超过检测事件,故我们以路径 ρ 中的执行的事件序列来表示执行路径中的步骤,并以此为基准计算路径 ρ 中执行相同的步骤的数量.此参数的设立为防止执行同一重复事件太多次而带来的无意义行为.
3. 采用的覆盖准则(CC):由于在遍历 IFML 模型时可执行路径数较为庞大,在实际自动化测试时无法逐条执行,为此本文采用覆盖准则来从这些路径中选择具有代表性的路径来生成并自动化测试.我们采用的覆盖准则以事件为单元进行覆盖,包含边覆盖,即测试序列中必须覆盖所有的事件,以及边对覆盖,即要求在测试序列中覆盖所有两个邻接事件组合出现的情况.前者目标在用覆盖应用的所有交互事件,后者目的则在此基础上覆盖多交互事件组合触发的情况.

下提出本文从遍历 IFML 模型,并以单事件覆盖为准则生成测试序列的算法逻辑,图 8 所示为该遍历算法的伪代码表示:

```
Algorithm 1: Path Traversing Algorithm
Input:    1)CS   2)Paths   3)IS    4)SE
Output : 从状态 CS 出发遍历得到的可执行路径
1.  procedure pathsTraversing(CS, Paths, IS, SE)
2.      candidateEvents := enable(CS.CM);
3.      for event∈candidateEvents do
4.          SE.add(event);
5.          if reach end conditions then
6.              if uncovered(SE) && check(IS, SE) then
7.                  path := generatePath(SE);
8.                  Paths.add(path);
9.              end if
10.         else
11.             for SI ∈ possibleSIGroup(event) do
12.                 CS' := generateCS(CS, event, SI);
13.                 pathsTraversing(CS', Paths, IS, SE);
14.             end for
15.         end if
16.         SE.remove(event);
17.     end for
18.     return Paths;
19.end
```

Fig.8 the algorithm of generating test sequences via traversing the IFML model

图 8 遍历 IFML 模型生成测试序列算法

图 8 中算法 1 描述了从给定 IFML 模型 M 的初始状态下递归生成所有满足要求的测试序列的主要过程.其

共有 4 个入参,其中:IS 表示为 IFML 模型的初始状态,即定义 9 中的 CS_0,其值可由 M 中的初始试图容器 v_1 以及各参数的默认值生成,在此不详细描述;Paths 表示为记录所有覆盖的事件以及其对应的可执行路径的映射集合,初始时为空;CS 表示在执行该算法时的 IFML 模型状态;SE 表示为从初始状态达到 CS 所需要的事件的序列的集合.该算法以初始状态 CS_0 开始执行,遍历选择其状态下任何可能触发的事件,若覆盖到未曾覆盖的事件并且达到中止条件时,则对该路径进行可行性验证,若可行便会生成对应测试序列更新 Paths,若未达到中止条件则会生成对应的新状态,在新状态下继续遍历选择其下可能触发的事件,从而递归执行该算法.该算法将会一直持续下去直到所有遍历完成.该算法的具体说明如下:

1) 在第 2 行,首先根据上述说明的 enable 函数获取在当前状态 CS 下可触发的事件集合 candidateEvents,并在第 3 行遍历选择这些事件来执行.由于 CS 下的参数与其当前值的映射集合 CA 并非实时更新,故在对于事件 e∈candidateEvents,其激活表达式 ax 的值不明,即 eval(ax, CA)的值未知.故对于其中重要的事件(即最终被选择执行的事件),其 eval(ax, CA)=true 需要在下述 check 函数中作为约束进行验证.
2) 第 4 行在 SE 中加入了遍历中的一个事件 event,表示下面将执行该 event 并更新状态.在第 14 行该 event 从 SE 中被删除,消除本次循环对下次循环的不良影响.
3) 第 5 行检查该次遍历/递归是否达到中止条件,其具体有 2 条:SE 中最大重复事件数达到设定值或最大事件数达到设定值.当达到中止条件时, 在第 6 行 if 语句中将会检查 SE 中是否有未被覆盖的事件(即 uncovered 函数)并从初始状态检查 SE 中事件序列的可行性(即 check 函数).该 check 函数会从初始状态 IS 逐步执行 SE 中的事件序列,若存在无法执行的情况(如对于 SE 中某一事件 e_i=(et,ax,ix,SI,pe)在执行它的状态 CS_i=(CM_i,CA_i)时 eval(ax, CA_i)=false;又如对于 e_i,不存在能够使得其 eval(ix,CA_i)获得的交互流集合 SI'与要求相同的情况等),则说明该 SE 中的事件序列不可行,反之则证明其可行性.当上述条件满足时便会通过 SE 的事件序列生成对应的测试序列并加入 Paths 中,即第 7,8 行语句.该测试序列实为 SE 中事件的顺序排列,包含事件触发类型,触发事件元素,触发该事件所需额外参数以及可能需要的输入信息.
4) 若第 5 行语句并未达到中止条件,则会继续生成新状态 CS'进行递归,值得注意的是该新状态 CS'中只更新 CM 而不更新 CA,因为由于用户输入等问题会导致 CA 的不确定性,为此在我们将遍历所有的可能情况,仅当在上述 check 函数时才会逐步检测其可行性并生成用户应当进行的输入.由于当事件 event 的交互流表达式 ix 不为空时,该事件会有函数 eval(ix, CA)用以选择在该事件后触发的交互流集合,但在 CA 不实时更新的情况下该函数无法使用.故在第 11-14 行,我们利用 possibleSIGroup 函数获取 event 所有可能触发的交互流集合,并逐一进行遍历,利用 generateCS 生成新的状态 CS'并进行递归,新状态 CS'下的 CM'获取同定义 9 上方的理论:获取 SI'中的导向流集合 SNI⊂SI';从 CM 中删除其中视图焦点转移过程中失去视图焦点的部分,即 CM'=CM-{x∈A∪V| Lv(pe, x)=true&& x∈CM }(其中 pe 表示为选定事件的父容器);随后,添加其获得视图焦点的元素,即遍历选择 SNI 中的导向流 i_i=(s,t,PBG)∈SNI,可知下一状态下获得视图焦点/直接控制流的元素记为 t,逐个进行 CM'= CM'∪{ x∈A∪V| Lv(t, x) = true&& x 不属于 CM'}.最终完成遍历后即可获得新状态 CS'下对应的 CM'.同时,在选择了 SI 的情况下,对于 event 的交互流表达式 ix,需要将 eval(ix, CS.CA)=SI 作为约束,在 check 函数中进行验证.
5) 最终在完成所有遍历和递归或是提前覆盖完所有事件以后,Paths 即为所要求的测试序列的集合.

4.2 测试序列的自动化测试执行

在获得上述测试序列集合后,本文进一步从该测试序列集合生成测试用例并使用相应的自动化测试工具进行自动化执行.

对上述 4.1 生成的测试序列,为便于使用,本文中将其以 XML 格式进行存储作为中间产物,并对该中间产物进行转化后使用相应工具进行自动化测试.该测试序列的 XML 格式存储如图 9.

在该 XML 格式中,project 标签表示为一个自动化测试项目,即为 4.1 章节中获得的测试序列的集合.其下包含多个事件流(eventsflow)子元素;任一事件流代表一组按序的事件集合,即一条测试序列;编号(id)属性表示该

测试序列的顺序标识.其下又包含多个顺序排列的事件(event)子元素代表测试序列中的事件,其下各子元素即为触发该事件所需要的信息:

- 顺序(Order)子元素表示当前事件的顺序标识;
- 组件(Component)子元素表示触发该事件的实体元素组件,以属性(type),名称(text),编号(id)子元素辅助其进行定位;
- 操作类型(operationType)子元素代表执行的具体操作或发生的事件,如 Touch,LongPress,DoubleTap 等;
- 参数(reference)子元素表示具体操作可能需要的参数,如 LongPress 的时间,Swipe 的方向等;
- 数据输入集合(dataList)子元素表示该步操作可能包含的输入信息,其下包含多个数据(data)子元素,各数据子元素则表示一条信息输入,文本名称(text),编号(id),序号(index)用于对参数输入的元素(一般为 EditView)进行定位,数值(value)则为输入的具体信息;

```
<project>
    <eventsflow id=???>
        <event>
            <order/>
            <component>
                <type/>
                <text/>
                <id/>
            </component>
            <operationType/>
            <reference></reference>
            <datalist>
                <data>
                    <text/>
                    <id/>
                    <index/>
                    <value/>
                </data>
            </datalist>
        </event>
    </eventsflow>
</project>
```

Fig.9 test sequences saved as XML

图 9 以 XML 格式存储的测试序列

生成 XML 格式的中间产物后,即可采用各式自动化测试执行工具根据其进行自动化测试.由于 Robotium 自动化测试工具[11]使用便捷,本文最终使用 Robotium 工具进行测试序列的自动化执行.根据该类工具的特性,还需将 XML 格式的测试序列转化成能被 Robotium 工具所能接收的测试用例.为此我们使用 spoon 这一基于 Java 的程序分析和插桩工具来进行相应转化,由于篇幅限制本文不再详述.

最终,该测试用例集合可使用 Robotium 工具直接在待测应用上进行自动化测试,并能生成具有反馈意义的测试信息,实现从 IFML 模型至测试用例并自动化执行的过程.

5 实验研究与分析

为验证上述基于 IFML 扩展模型的安卓应用自动化测试方法可行性,本节将其阐述为 2 条具体的问题:

- RQ1:本文中的自动化测试方法是否真正能够实用于真实的应用?是否能够保证较高的覆盖率并发现应用中的 Bug?
- RQ2:该方法时间效率如何?与其他的主流测试方法比起来是否足够有效?

5.1 实验设计

针对 RQ1,本文将上述的自动化测试方法应用于 6 个安卓应用上.第 1 个即为应用 Good Weather,第 2 个则为 Remindly,一个事件提醒软件,允许用户事先输入事件发生的时间,并在时间到时进行通知.上述 2 个应用均为开源项目,可在 github 上找到相应的源码,也可在 Google Play 上找到相应应用.我们基于源码对这两个应用进行了建模.另 4 个应用则分别为 TipCalculator,一个小费计算工具; World Time,一个当前世界时间显示工具;Camera FV5 Lite,著名的相机应用的免费版本以及 Flower Player,一个音乐播放软件.它们均是 Google Play 和 Apkture 上发布的商业软件,我们基于使用经验对其进行了建模.表 4 显示了这些应用的基本信息和 IFML 模型信息.

Table 4 Testing result of test cases for the six apps

表 4 6 个应用的各项测试执行结果

应用	版本	类型	IFML 事件总数	主视图容器数	参数 ME	参数 MDE
TipCalculator	1.5	工具	12	1	6	2
World Time	2.1	工具	23	3	10	2
Remindly	2.0	工具	28	3	10	2
Good Weather	4.4	天气	33	3	8	2
Camera FV5 Lite	3.2	摄影	102	4	8	2
Flower Player	1.784	音乐	130	8	8	2

表 4 中展示了这 6 个应用的类型以及我们所采用的版本.第 4,5 列"IFML 事件总数"和"主视图容器数"表示了我们为该应用建立的 IFML 模型中事件的总数和主要的视图容器的数量,这可以在一定程度上反映应用的 GUI 复杂度,但也存在例外.在最后 2 项"参数 ME"和"参数 MDE"中,通过反复试验和 GUI 规模估算,我们确定了适用于这些应用的模型遍历参数——执行路径最大事件数 ME 和执行路径最大重复事件数 MDE,使得在当前取值下事件覆盖率能达到一个比较高的水准,并且其生成时间也在一个可以接受的范围内.同时为了回答上述问题 RQ2,本文也采用安卓终端主流的自动化测试工具 monkey 对上述 2 个应用进行测试.为使实验公平,我们在 monkey 测试中分别尝试了指定其执行事件数以及充分测试情况下的情况,来进行比较.本文的测试序列自动化生成执行在 windows 环境下,并且执行机器拥有 8GB 内存,i7 处理器以及 4 核心.安卓应用测试实验则均运行在同一台三星 S4 系列的安卓实体机上,并且在每次开启测试前清空该应用的数据.

5.2 实验结果分析

最终本文对上述 6 个应用进行的各项测试的实验结果记录在下表 5 中.

Table 5 Testing result of test cases for the six apps

表 5 对 6 个应用的各项测试执行结果

应用 :*TipCalculor*								
序号	事件总数	测试方法	覆盖准则	用例数	覆盖率	生成时间	执行时间	缺陷
1	12	本文方法(边覆盖)	单事件覆盖	12	100%	<1s	81.8s	--
2		Monkey(200 事件)	单事件覆盖	--	75%	--	14.6s	--
3		Monkey(1800 事件)	单事件覆盖	--	100%	--	137s	--
4	84	本文方法(边对覆盖)	事件对覆盖	84	100%	1.2s	586.8s	--
应用 :*World Time*								
序号	事件总数	测试方法	覆盖准则	用例数	覆盖率	生成时间	执行时间	缺陷
5	23	本文方法(边覆盖)	单事件覆盖	23	100%	21.5s	288.5s	--
6		Monkey(500 事件)	单事件覆盖	--	43.5%	--	40.4s	--
7		Monkey(10000 事件)	单事件覆盖	--	78.3%	--	723.4s	--
8	87	本文方法(边对覆盖)	事件对覆盖	82	94.2%	22.0s	1403.6s	--
应用 :*Remindly*								
序号	事件总数	测试方法	覆盖准则	用例数	覆盖率	生成时间	执行时间	缺陷
9	28	本文方法(边覆盖)	单事件覆盖	28	100%	34.8s	376.1s	--
10		Monkey(600 事件)	单事件覆盖	--	42.9%	--	48.2s	--
11		Monkey(4900 事件)	单事件覆盖	--	85.5%	--	357.6s	--
12	171	本文方法(边对覆盖)	事件对覆盖	137	80.1%	36.2s	1799.8s	--

应用 : *GoodWeather*								
序号	事件总数	测试方法	覆盖准则	用例数	覆盖率	生成时间	执行时间	缺陷
13	33	本文方法(边覆盖)	单事件覆盖	33	100%	3.5s	342.8s	1,2
14		Monkey(600 事件)	单事件覆盖	--	60.6%	--	40.2s	1
15		Monkey(6500 事件)	单事件覆盖	--	94.1%	--	474.1s	1
16	203	本文方法(边对覆盖)	事件对覆盖	196	96.5%	3.8s	2065.5s	1,2
应用 : *Camera FV 5 Lite*								
序号	事件总数	测试方法	覆盖准则	用例数	覆盖率	生成时间	执行时间	缺陷
17	102	本文方法(边覆盖)	单事件覆盖	100	98.0%	193s	1487.1s	3,4,5,6
18		Monkey(2000 事件)	单事件覆盖	--	37.3%	--	153.8s	3
19		Monkey(30800 事件)	单事件覆盖	--	76.5%	--	2390.1s	3,6
20	1483	本文方法(边对覆盖)	事件对覆盖	989	66.7%	201s	12818s	3,4,5,6
应用 : *Flower Player*								
序号	事件总数	测试方法	覆盖准则	用例数	覆盖率	生成时间	执行时间	缺陷
21	130	本文方法(覆盖)	单事件覆盖	130	100%	278.6s	1085.3	7
22		Monkey(2100 事件)	单事件覆盖	--	33.6%	--	143.3s	--
23		Monkey(40000 事件)	单事件覆盖	--	74.7%	--	3016.7s	--
24	2340	本文方法(边对覆盖)	事件对覆盖	1115	47.6%	292.2s	10605.8s	7

如上表 5 所示,其显示的是采用文本方法(包含事件边覆盖准则以及边对覆盖准则)以及 monkey 对该 6 个应用进行测试的结果.

表 5 中将针对 6 个应用的各项测试分开显示,每一行均表示为一项实验结果,每个应用均进行了四项实验,从上至下依次为:以单事件覆盖为准则的本文方法,等事件数的 monkey 测试,接近覆盖极限的 monkey 充分测试以及以事件对覆盖为准则的本文方法(下记为每个应用的第 1,2,3,4 项实验),表 5 中第 3 列"测试方法"对其进行了标注.其中前 3 者以单事件来计算覆盖率,而最后一个则以事件对来计算覆盖率,即第 4 列的"覆盖准则"内容,同时第 2 列的"事件总数"则表明了该应用中的单事件/事件对的总数.在第 2 项实验中,即等事件数的 monkey 测试,要求与第 1 项实验执行相同的事件数来进行对比.以应用 GoodWeather 为例,第一项实验中总事件数为(用例数)33*(每用例事件数)8=264,同时由于 monkey 测试中将一个触摸/滑动事件表示为 Action_Up 和 Action_Down 两个事件,我们又有 robotium 中的 264 事件≈monkey 中 264*2 事件,向上取整则为 600 事件数,即第 2 项的 monkey 测试中将执行 600 个事件.同理,对于其他 5 个应用,则分别对应于 200 事件,500 事件,600 事件,2000 事件和 2100 事件.在第 3 项实验中,我们则利用 monkey 对这些应用进行充分测试,在该实验中,我们会执行较长时间的 monkey 测试,直到单事件覆盖完成,或是在较长的时间段内事件的覆盖度提高极为有限.本文未对事件对覆盖测试进行 monkey 的对比测试,原因在于本文的事件对覆盖测试收益并不高,和 monkey 测试意义不大,其理由会在下文进行说明.

表 5 中第 5,6 列"用例数"和"覆盖率"分别表示了该方法下生成的测试用例数以及其统计下来的覆盖率.首先对于单事件覆盖,我们的测试方法在 6 个应用上的覆盖率基本都能达到 100%.第 17 号实验,即 Camera Fv5 Lite 应用上未能达到 100%的原因在于存在 2 个事件需要更长的 ME 参数设置才能达到,实验证明当 ME 设置为 9 时便可达到 100%覆盖.然而设置更大的 ME 会导致生成时间过长,反而不利于测试进行.

而等事件数的 monkey 测试覆盖率仅在 75%-33.6%之间,并且大体上随着应用的事件数增多而减少.其原因正如本文第 2 章所描述:monkey 测试无效事件多,容易跳转至其他应用,小型控件难以触发,事件容易集中在少数界面上导致覆盖不均,以及其并不支持部分安卓设备上的事件等等.并且随着应用的 GUI 复杂度的提高,这些现象则会更加显著.在第 6,10,14 号实验中,事件数最多的应用 GoodWeather 在 monkey 测试下反而达到最高的覆盖率,其原因在于另 2 个应用中的部分事件需要一些前置事件才能触发,例如 Remindly 需要建立提醒记录才能对该记录进行操作,而 World Time 则需要建立地点的实例.在随机测试下,这些前置事件难以达成,致使覆盖率低下,同时这也致使本文方法需要设置更高的 ME.在表 5 中, Remindly 和 World Time 均需要将 ME 设置为 10 才能达到单事件的全覆盖,这也导致了其生成时间达到 21.5s 和 34.8s.

在充分的 monkey 测试下,仅有 TipCalculor 应用的覆盖率达到了 100%.WorldTime 覆盖率只有 78.3%,原因在于该应用中的地点实例的创建步骤为:1.输入有意义的关键字,例如 London 的前 3 个字符 Lon;2.从下拉列表

中进行选择;3.点击右方小图标进行确定添加.这 3 个步骤在随机事件下难以按序达成,使得有效创建地点实例的可能性极低,部分基于该地点实例的事件无法达成.Remindly 应用中则需要使用到长按 LongPress 的事件,而 monkey 并不支持,导致该事件以及其后续事件无法覆盖,覆盖率只有 85.5%.GoodWeather 覆盖率为 94.1%,其原因在于在其 IFML 模型中,存在一个 Wifi 连接相关的系统事件以及主界面上向右滑动的视图元素事件,它们在 monkey 测试的机制下所无法覆盖的.而对于 Camera FV5 Lite 和 Flower Player 应用,其充分 monkey 测试的覆盖率均小于 80%,其原因除了上述的系统事件和特殊行为事件无法被覆盖以外,还在于 Camera FV5 Lite 应用的主界面边界上存在大量的小型按钮,并且在触摸它们后会在边界生成新的控件按钮或新的界面.因此为覆盖这些内容,要求测试时能够多次连续触摸边界上的元素,而这在 monkey 测试的随机性下发生概率极低:在长达 2390s 的 monkey 测试后,这些边界元素相关的事件仍有 15 个未被覆盖,并且在额外增加了足够的测试时间下其覆盖率依然没有明显提升.而 Flower Player 应用在测试时集中在音乐列表界面,播放界面以及专辑界面等部分容易进入的界面,并且也与 Camera FV5 Lite 类似,部分小型按钮展开的列表按钮难以覆盖完全,导致其覆盖率低下,仅有 74.7%.

上述 monkey 实验结果均为测试时人工进行事件覆盖确认得出,可能存在一定误差,但其较大的差距显示了其对于单事件的覆盖率,测试效率远低于本文方法,并且随着应用越发复杂,覆盖率差距越大,所耗的时间也越长.

而对于事件对覆盖而言,本文方法在这 6 个应用上的覆盖率在 100%-47.6%,仅有 TipCalculor 应用达到了 100%覆盖.在其他应用上未达到 100%的原因有 2 个:1.存在一定数量的事件对在逻辑上不可达的,如激活表达式的使用会使得在触发某事件后另一些事件无法直接被触发,从而导致这类事件边对无法被覆盖;2.剩余事件对的覆盖需要更高的遍历参数 ME 和 MDE.Camera FV5 Lite 和 Flower Player 应用的事件对覆盖率远低于前面 4 个应用,其具体原因在于这 2 个应用在逻辑上远复杂于前者.以 Good Weather 和 Camera FV5 Lite 为例,通过简单的统计可知 Good Weather 上逻辑不可达的事件对仅为 4 组,而 Camera FV5 Lite 则有近 100 组事件对不可达,其数量众多在于在该应用中激活表达式的应用繁多,参数值的改动导致视图组件/事件等不可见的情况出现较为频繁.并且由于其逻辑复杂,为达到特定边需要执行额外的前置事件,导致所需要的遍历参数 ME 更大,在将其设定为 8 时会不可达到.实验证明在将其设置为 9 和以上时会有更高的覆盖率.

表 5 第 7,8 列”生成时间”和”执行时间”表示了这些测试实验的生成/执行测试用例的时间.表中各应用上的生成时间均有较大差距,在于影响测试用例生成时间的因素非常多,其中主要有 2 个:一个是应用的 GUI 结构和复杂度,如实验 13,17 和 21,虽然采用了相同的遍历参数,但 Camera FV5 Lite 和 Flower Player 的 GUI 复杂度远高于 Good Weather,导致了其生成时间上 3.5s 和 193s 以及 278.6s 间的巨大差异;另一个则是参数 ME 的设定,通过在 Good Weather 应用上设置不同的 ME 值也可得到相应的结论:当 ME 为 7 时,生成时间为 1.1s;ME 为 8 时,则为 3.5s;当 ME 为 9 时,则为 10.2s.对于不同应用,ME 的增长会导致生成时间按照指数级增长.单事件/事件对覆盖时遍历主逻辑是一样的,只是其挑选生成测试序列的覆盖准则发生了变化,故单事件/事件对覆盖所花的时间也比较相似,实验 1 和 4,5 和 8,9 和 12,13 和 16,17 和 20,21 和 24 中的生成时间差别不大则正好说明了这一点.然而执行近似事件数的 monkey 测试和本文测试方法相差时间相差较大,主要在于本文所使用的自动化测试工具 Robotium,在执行测试用例时需要退出/重启应用,并且在执行事件时会有默认的等待时间以及响应等待时间,这些延迟时间是不可避免的.并且我们认为这种有适当等待时间的测试方法更符合应用使用者的实际使用情况.

表 5 最后一列”缺陷”表示了在这些测试实践中发现的缺陷的序号,其缺陷的具体内容可见下表 6.

首先对于应用 Good Weather,在进行测试时由于应用的特性,我们默认其初始情况下网络与 GPS 均打开,为此缺陷 1 由于较为明显,无论是 monkey 测试还是本文方法均可找到.然而缺陷 2 要求只有在在 Wifi 无连接情况下才会触发,monkey 本身并未设计有专门开/关 Wifi 的能力,故最终无法发现缺陷 2.然而在本文方法中,通过在对应 IFML 模型中建立一个 Wifi 系统事件用于表示 Wifi 的突然关闭,便可在测试序列中覆盖该事件,在测试脚本中则体现为 Robotium 关闭设备的 Wifi 连接.而在此情况下,本文方法则最终可以观测到该缺陷 2 的发生.

而在应用 Camera FV5 Lite 中, 缺陷 3 由于由于静止图像模式进入较为简单,缺陷 6 当其快速进行页面切换时极为容易引发,因此无论 monkey 还是本文方法均可以找到.而对于缺陷 4, 5 由于并未出现严重的异常事件如崩溃,并未有模型支持的 monkey 随机测试无法判断并确认该类情况是否真正属于应用缺陷.然而对于本文方法,由于对于测试用例的生成是来源于已经建立的 IFML 模型,故能够以 IFML 模型为标准来判断该类情况是否属于应用缺陷.在测试实践中表现为生成的测试用例无法执行下去时,我们便可认为 IFML 模型与对应的应用存在不一致,在以 IFML 模型为准的情况下,我们便可确认应用中存在缺陷.显然,本文方法能很好地与利用 IFML 模型驱动开发方法相结合,复用在设计/实现应用时所用到的 IFML 模型,并且该 IFML 模型更具权威,准确性更高.

在应用 Flower Player 中,缺陷 7 所在的页面较深,并且从观察中发现在进入应用较长时间后并不会触发该缺陷(可能是因为该应用需要一定时间来初始化或者加载广告).随机性较高的 monkey 测试无法在短时间内定向到该缺陷所在位置,因此几乎不可能观测到该缺陷的发生.然而具有较高导向性的本文方法则可以有效地发现这一问题.

Table 6 Details of defects found in the six apps

表 6 6 个应用中发现的缺陷信息

缺陷序号	对应应用	缺陷描述
1	Good Weather	在应用的搜索界面输入关键字时,出现未处理的 java.lang.NullPointerException 异常并闪退
2	Good Weather	当无网络(Wifi)连接时,定位会陷入无限的等待模式
3	Camera FV5 Lite	相机处于静止图像模式时,当进入设置面板或打开相册会大概率出现未处理的 java.lang.RuntimeException: startPreview failed 异常并闪退,并且在下次打开应用时也必定会闪退
4	Camera FV5 Lite	将相机曝光事件设置为/取消设置为 Short/Long/Long+时,相应的曝光按钮将会无法点击,直到用户进入并退出相册界面或重启应用
5	Camera FV5 Lite	当第一次打开设置存储位置的界面时,其会出现相应的提示并可以勾选是否不要再度显示该提示,若勾选并确定了,将会导致该存储位置界面无法再度打开
6	Camera FV5 Lite	当频繁进行界面切换时容易引起 java.lang.RuntimeException 异常而导致崩溃
7	Flower Player	在开启应用的最初一定时间内,打开背景设定界面选择图片时会陷入无限等待模式

最后通过两个应用的边/边对覆盖的缺陷发现比较,我们可以发现边对覆盖比之边覆盖并没有明显的效果提升,而观察所消耗的时间可以看到边对覆盖所花的时间要比边覆盖多 4-10 倍,为此我们认为边对覆盖的提升效果在这两个应用上有限,为此不再做对应的 monkey 测试进行比较.

6 相关工作

6.1 关于移动端模型驱动开发的研究

模型驱动开发的研究已有一段时间,但是将它应用于移动终端应用仍是一个较为新颖的主题.考虑到目前的移动系统繁多,特别是安卓系统版本/设备众多呈现碎片化的现象,这一应用是极为有价值的:建立抽象模型,通过其生成多个适用于不同移动端系统,或不同系统版本的移动端应用,可以避免对同一应用的重复从头实现,节省大量的重复开发成本.目前有部分文献针对这一主题进行了相应的研究.Parada 等人[4]直接基于 UML 类图和时序图提供移动应用的高层抽象模型并实现了自动化生成,但利用 UML 类图无法直观地展现应用的前端特征,因此并不适用于注重前端设计的移动应用.Balagtas-Fernandez 等人[12]设计了 Mobia 这一图像化的移动应用

建模套件进行移动应用的模型驱动开发,但 Mobia 支持的建模语言与 UML 等标准语言差异较大,同时抽象层次不高,对系统版本的演化支持较差.此外,也有专门针对数据驱动的应用基于 md² 这一 mvc 结构模型进行移动端跨平台模型驱动开发的研究[5],但 md² 本身以文本形式显示,并且采用 mvc 结构反而增加了建模的复杂度,也并不适合于移动应用.此外,也有一些在类似的模型上增加其抽象层次,从而使得应用开发更加灵活多变的方式[13].对于本文所用到的 IFML 标准,研究还处于起步阶段.Brambilla 等人[14]通过建立移动应用的 IFML 模型,来生成对应的 HTML5,CSS3 和 JavaScript 代码,并最终包装在移动端安卓与 IOS 平台的应用中.但他们没有面向安卓应用做细致的扩展,导致所使用的模型无法生成可自动执行的测试用例.

6.2 关于IFML建模相关的研究

现有研究工作中,IFML 除了用于移动端的模型驱动开发以外,也有在其他领域进行建模以及相应扩展的研究.例如 Ed-douibi 等人[15]利用 IFML 建立网页模型,生成符合 REST 原则的网络应用编程接口;Raneburger 等人在[16]中描述了使用 IFML 实现独立多设备 GUI 生成的机制,并将其与 Model Based Useware-Engineering(MBUE)和 Unified Communication Platform(UCP)进行了简单的比较;Frajt´ak 等人[17,18]利用 IFML 对一般应用建模并转化为对应的前端测试模型,并最终利用该模型进行了针对前端的自动化测试用例的生成;Brajnik 等人[19]通过对一般应用进行 IFML 的建模,提出了将应用模型中的数据流和控制流进行分离的可能性;LAAZ 等人[20]则尝试将 IFML 模型与 OWL(Web Ontology Language)实体整合起来,以增强网页应用的用户接口的展示.

总体而言,IFML 本身由 WebML(Web Modeling Language)发展而来,其针对网页应用的建模较为全面,并有较多的扩展和研究,但针对移动端的应用,虽有上述对移动端建模的实践尝试,但在特定平台应用,如安卓应用上基于 IFML 进行测试依然有较高难度.为此本文对 IFML 的移动端扩展限定在安卓端,为 IFML 面向安卓进行定制,细化或简化部分内容,从而将扩展的 IFML 成功应用于基于模型的自动化测试中.关于安卓终端应用自动化测试的研究

6.3 关于安卓终端应用自动化测试的研究

近年来安卓终端迅速发展,其上所使用的安卓应用无论是在规模上还是在数量上都呈爆炸式增长,作为其软件生命周期的一环——测试承担了越来越艰巨的任务.为此各式针对安卓终端的自动化测试工具应运而生.

目前有不少能够支持自动化执行的工具,如安卓端测试工具 TestQuest Pro[21],它能读取预定应的测试行为序列 Test verb[22],生成并执行对应的测试用例,最终进行强有力的图片,音频识别来进行测试验证; Jiang[9]等人设计了在智能安卓设备上采取了一种基于敏感事件来简化测试用例的设计以及增强其有效性和可重用性的方法来实现自动化的黑盒测试,并研发了相应的工具 Moible Test.工业上当前是用较为频繁的该类工具也很多,如 Robotium,Appium,MonkeyRunner 等.这一类自动化执行工具已经较为成熟,但它们仍然需要人工输入测试序列,测试脚本等,消耗大量人力,为此实现测试脚本或者用例的自动化生成逐渐成为提高测试自动化程度的重点.

近年来,随着面向对象软件开发技术的广泛应用和软件测试自动化的要求,特别是基于模型的软件开发技术的逐渐普及,基于模型的软件测试方法逐渐得到了软件开发人员和软件测试人员的认可和接受[23].大量的研究着眼于测试用例的自动化生成和执行,通过获取安卓应用内的事件触发信息,随后根据这些事件信息生成对应的事件模型,随后自动化生成测试用例并自动化执行,例如 Li 等人[24]在 UML 活动图基础上实现了一种在安卓智能终端应用上的基于模型的自动化测试方法,并生成相应的工具 ADAutomation,但正如 7.1 中所述,UML 不适用于注重前端设计移动安卓应用.目前针对安卓终端的自动化测试大体都是通过自动化遍历应用,触发界面上事件并进行反馈,最终在建立应用对应的 GUI 模型的同时完成测试.例如 Anand 等人[25]将 concolic 测试应用于安卓应用输入事件的自动化生成上,利用符号化执行实现对安卓应用的自动化测试, Machiry[26]推出了一种为安卓应用自动化生成测试输入的方法,大体为在自动化的执行中能够监控应用对事件的响应,并根据这些响应(或者人工给出的输入)指导下一步事件的执行,最终这一方法被具体设计成一个名为 Dynodroid 的工具.其他有许多在 GUIRipper[27]基础上针对移动端应用的扩展如 AndroidRipper[28], MobiGUITAR[29],以及 Amalfitano 等人

[30]设计的提供一个自动实施崩溃测试并生成相应测试报告的测试工具集等等.这一类关于自动化测试的研究能够实现较高程度的自动化,但是正由于其模型的自动化生成,其准确度存在一定问题,并且无法准确建立可能会影响 GUI 模型的应用逻辑.这使得下面的自动化测试效果大打折扣.同时它们依赖的模型往往为以 GUI 树为基础的控制流模型,它们往往仅能够展示事件触发信息,与 IFML 相比表现力不足.Su 等人[31]采用逆向工程静态分析和动态 UI 探测来抽取安卓应用的随机模型,然后基于模型生成测试用例进行执行.同时使用 Gibbs 抽样来分析执行结果来精化原随机模型,进一步更好地生成测试用例,形成一个回环的方法.该方法基本无需人工干预,并且通过系统化的执行反馈精化模型,随机模型的准确性和测试的效果都能够得到有效保障.与该方法重点关注通过测试反馈来精化模型不同,本文着重于扩展 IFML 模型以适用于安卓应用的建模和测试.扩展后的 IFML 模型能直观有效地描述丰富多样的安卓应用交互过程,并生成可执行的测试用例.值得注意的是,本文提出的测试方法依赖于较为完整的模型,因此如果设计阶段的模型不够完整正确,可以借鉴 Su 等人提出的方法,通过测试反馈来完善模型. Sadeghi 等人[32]提出的 PATDroid 方法着眼于安卓应用测试需要的权限组合,并以此来提高测试效率.Garcia 等人[33]提出的 LetterBomb 方法能自动化地检测安卓应用中可能存在的 ICC 漏洞,通过符号化执行生成相应的攻击 Intent 来检验这些漏洞是否能被利用.这两项工作面向特定目标研究了如何更有效地产生测试输入数据,而本文提出的是一种测试应用的通用方法,重点在于覆盖尽可能多的应用交互逻辑.因此,上述方法与本文方法具备一定的互补性,若能结合则可以更好地对安卓应用进行测试.国内也有类似的研究,陈军成等人针对现有的基于模型 GUI 测试用例自动生成过程中面临的测试用例规模庞大以及生成的测试用例无效问题,从分析事件处理函数的角度出发,提出了一种 GUI 测试模型 EHG,并针对该模型提出两个测试覆盖准则,来控制测试规模并提高事务处理函数的代码结构覆盖率[34].相比而言,本文扩展的 IFML,直接以前端控件及其相应事件出发,能更好地对不同类型的控件和事件建模,从而在测试过程中能有效识别控件并通过各类动作触发事件,有利于测试过程的自动化.

7 总结和展望

在移动平台尤其是安卓平台飞速发展的今天,为满足广大用户纷繁复杂对移动的需求,无数的安卓应用应需而生.在当前应用需求易变,开发/迭代周期短的情况下,作为应用开发者并没有多余时间为其应用进行手工测试.因此在当前趋势下,针对安卓应用的自动化的测试都是极为重要的.然而当前的自动化测试往往都集中于测试脚本的自动化执行,无法达到理想的测试效果.

安卓设备和系统存在多样化和碎片化的现象,使得 MDE 成为安卓应用开发的一种有效方式.而基于模型的测试,是 MDE 的重要一环.为此,本文对目前适合于移动/安卓应用建模的 IFML 标准进行了面向安卓端的扩展,并以此为基础,实现了基于扩展后的 IFML 模型的自动化测试方法.通过遍历待测应用的 IFML 模型,生成对应的测试序列,最终转化成测试引擎的脚本,可由其进行自动化测试,并以测试的执行信息定位待测应用中存在的缺陷,即与 IFML 模型不一致的地方.该方法能够与模型驱动的软件开发方法相结合,复用设计时所建立的模型,减少测试成本,是极具价值的自动化测试方法.

当前,该方法处于初步建立的阶段,还存在一定的局限性.首先,该方法中当其参数 ME 和 MDE 设置的值增大时,生成测试用例所花的时间将会较长,为此提高测试用例生成效率将会成为今后主目标之一;其次,本方法对于系统应用的处理还较为有限,为此还需要不断对未被覆盖的系统事件的处理进行设计.

References:

[1] Gather.2017 Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016: http://www.gartner.com/newsroom/id/3609817

[2] Android.wiki : https://en.wikipedia.org/wiki/Android_(operating_system)

[3] Chandra R, Karlsson B F, Lane N, et al. Towards Scalable Automated Mobile App Testing[R]. Technical Report MSR-TR-2014-44, 2014.

[4] Parada A G, de Brisolara L B. A model driven approach for android applications development[C]//Computing System Engineering (SBESC), 2012 Brazilian Symposium on. IEEE, 2012: 192-197.

[5] Heitkötter H, Majchrzak T A, Kuchen H. Cross-platform model-driven development of mobile applications with md 2[C]//Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, 2013: 526-533.

[6] W. Yang, M. R. Prasad, and T. Xie, “A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications,” in Hybrid Systems: Computation and Control. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 250–265.

[7] Developers A. Monkeyrunner[J]. 2015. https://developer.android.com/studio/test/monkeyrunner/index.html

[8] Appium automation for apps. http://appium.io/

[9] Bo J, Xiang L, Xiaopeng G. Mobiletest: A tool supporting automatic black box test for software on smart mobile devices[C]//Proceedings of the Second International Workshop on Automation of Software Test. IEEE Computer Society, 2007: 8.

[10] IFML: The Interaction Flow Modeling Language.http://www.ifml.org/..

[11] Robotium. User scenario testing for Android. http://code.google.com/p/robotium/.

[12] Balagtas-Fernandez F, Tafelmayer M, Hussmann H. Mobia Modeler: easing the creation process of mobile applications for non-technical users[C]//Proceedings of the 15th international conference on Intelligent user interfaces. ACM, 2010: 269-272.

[13] Vaupel S, Taentzer G, Harries J P, et al. Model-driven development of mobile applications allowing role-driven variants[C]//International Conference on Model Driven Engineering Languages and Systems. Springer, Cham, 2014: 1-17.

[14] Brambilla M, Mauri A, Umuhoza E. Extending the interaction flow modeling language (IFML) for model driven development of mobile applications front end[C]//International Conference on Mobile Web and Information Systems. Springer International Publishing, 2014: 176-191.

[15] Ed-Douibi H, Izquierdo J L C, Gómez A, et al. EMF-REST: generation of RESTful APIs from models[C]//Proceedings of the 31st Annual ACM Symposium on Applied Computing. ACM, 2016: 1446-1453.

[16] Raneburger D, Meixner G, Brambilla M. Platform-Independence in Model-Driven Development of Graphical User Interfaces for Multiple Devices[C]//International Conference on Software Technologies. Springer Berlin Heidelberg, 2013: 180-195.

[17] Frajták K, Bureš M, Jelínek I. Transformation of IFML schemas to automated tests[C]//Proceedings of the 2015 Conference on research in adaptive and convergent systems. ACM, 2015: 509-511.

[18] Frajták K, Bureš M, Jelınek I. Using the Interaction Flow Modelling Language for Generation of Automated Front–End Tests[J]. Annals of Computer Science and Information Systems, 2015, 6: 117-122.

[19] Brajnik G, Harper S. Detaching control from data models in model-based generation of user interfaces[C]//International Conference on Web Engineering. Springer International Publishing, 2015: 697-700.

[20] Laaz N, Mbarki S. Integrating IFML models and owl ontologies to derive UIs web-Apps[C]//2016 International Conference on Information Technology for Organizations Development (IT4OD). IEEE, 2016: 1-6.

[21] TestQuest. TestQuest Pro Guide. Technical Report.http://testquest.iexposure.com.

[22] TestQuest. Developing a Low Cost, Test Automation Solution with TestQuest TestVocabulary/TestVerb Technology.TestQuest White Paper. http://testquest.iexposure.com

[23] 颜炯, 王戟, 陈火旺. 基于模型的软件测试综述[J]. 计算机科学, 2004, 31(2):184-187.基于模型的软件测试综述

[24] Li A, Qin Z, Chen M, et al. ADAutomation: An Activity Diagram Based Automated GUI Testing Framework for Smartphone Applications[C]//Software Security and Reliability (SERE), 2014 Eighth International Conference on. IEEE, 2014: 68-77.

[25] 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: 59.

[26] Machiry A, Tahiliani R, Naik M. Dynodroid: An input generation system for android apps[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013: 224-234.

[27] Memon A M, Banerjee I, Nagarajan A. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing[C]//WCRE. 2003, 3: 260.

[28] Amalfitano D, Fasolino A R, Tramontana P, et al. Using GUI ripping for automated testing of Android applications[C]//Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ACM, 2012: 258-261.

[29] Amalfitano D, Fasolino A R, Tramontana P, et al. MobiGUITAR: Automated model-based testing of mobile apps[J]. IEEE Software, 2015, 32(5): 53-59.

[30] Amalfitano D, Fasolino A R, Tramontana P, et al. A toolset for GUI testing of Android applications[C]//Software Maintenance (ICSM), 2012 28th IEEE International Conference on. IEEE, 2012: 650-653.

[31] Su T, Meng G, Chen Y, et al. Guided, stochastic model-based GUI testing of Android apps[C]//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM, 2017: 245-256.

[32] Sadeghi A, Jabbarvand R, Malek S. PATDroid: permission-aware GUI testing of Android[C]//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM, 2017: 220-232.

[33] Garcia J, Hammad M, Ghorbani N, et al. Automatic generation of inter-component communication exploits for Android applications[C]//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM, 2017: 661-671.

[34] 陈军成, 薛云志, 赵琛. 一种基于事件处理函数的 GUI 测试方法[J]. 软件学报, 2013, 24(12): 2830-2842.