软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(10):3148−3167 [doi: 10.13328/j.cnki.jos.005793]

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

面向安卓应用建模的 IFML 扩展*

陆一飞, 潘敏学, 张 天, 王林章, 李宣东

(计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 潘敏学, E-mail: mxp@nju.edu.cn; 张天, E-mail: ztluck@nju.edu.cn

摘 要: 随着智能机以及平板电脑的普及,安卓应用逐渐成为日常生活中不可或缺的重要元素之一,其复杂度也呈几何倍数增长.安卓平台存在的多设备类型、多操作系统版本问题,使得应用的设计和开发更为复杂.在这一现状下,提倡在安卓应用开发中使用模型来描述其开发需求与设计,以帮助开发人员更好地将注意力集中于应用,加深对开发意图的理解,更好地进行后续的开发工作.然而,当前对安卓应用的建模都采用了传统模型,无法满足安卓应用事件驱动和注重图形用户界面的特点.为此,将注重前端展示以及事件交互的交互流建模语言(IFML)应用于安卓应用的建模,描述应用中的 GUI 结构以及其中工作流的传递,从而指导应用的开发工作.考虑到安卓平台的特点,对 IFML 进行了相应的面向安卓的扩展,提高了其可用性与对安卓应用的适用性,并对 IFML 模型进行了形式化定义,使得 IFML 模型能以丰富而又精确的语义来刻画开发者对于安卓应用的设计,并在应用的实现和演化中不断发挥指导作用.另外,进一步探索了 IFML 模型在应用测试这一场景中的作用.基于模型的测试方法能够检验设计和实现的一致性,还能在应用的演化过程中避免测试用例的重复编写.在案例研究中,针对 5 个安卓应用进行了 IFML 建模与测试.实验结果表明,扩展后的 IFML 在安卓应用的建模上可行、有效,所建立的 IFML 模型可直接用于测试工作,用于检测应用实现与设计是否保持一致,从而保证应用的开发质量.

关键词: 交互流建模语言;安卓应用;模型驱动工程;基于模型的测试

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.面向安卓应用建模的 IFML 扩展.软件学报,2019,30(10):3148−3167. http://www.jos.org.cn/1000-9825/5793.htm

英文引用格式: Lu YF, Pan MX, Zhang T, Wang LZ, Li XD. Extension to interaction flow modeling language (IFML) for Android application modeling. Ruan Jian Xue Bao/Journal of Software, 2019,30(10):3148−3167 (in Chinese). http://www.jos.org.cn/1000-9825/5793.htm

Extension to Interaction Flow Modeling Language (IFML) for Android Application Modeling

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 of smartphones and tablets, Android devices have gradually become one of the most important elements in our daily life. Along with it, Android applications are now flourishing and their complexity increases geometrically. Meanwhile, Android fragmentation is aggravating, which forces developers to design and develop the same Android apps for different Android versions and devices. In this case, employing models are proposed for requirements and designs in Android app development. With models, dividing and conquering these requirements and designs are possible which reduces the general complexity. At the same

* 基金项目: 国家重点研发计划(2017YFA0700604); 国家自然科学基金(61502228, 61632015); 中央高校基本科研业务费专项资金(020214380045)

Foundation item: National Key R&D Program of China (2017YFA0700604); National Natural Science Foundation of China (61502228, 61632015); Fundamental Research Funds for the Central Universities (020214380045)

本文由“面向 DevOps 的软件工程新技术”专题特约编辑荣国平、白晓颖、岳涛推荐.

收稿时间: 2018-08-31; 修改时间: 2018-10-31; 采用时间: 2018-12-14; jos 在线出版时间: 2019-05-22

CNKI 网络优先出版: 2019-05-22 15:25:50, http://kns.cnki.net/kcms/detail/11.2560.TP.20190522.1525.002.html

time, models of high expression help developers to better understand the purpose and finally guide the development work. However, the traditional models are no longer suitable, since Android apps are even-driven and GUI centric. Therefore, Interaction Flow Modeling Language (IFML) is adopted, the new OMG standard for front-end design and event interaction, in Android app modeling to describe apps' GUI structures and workflows and guide the development. Furthermore, an extension of IFML is proposed for Android to improve its usability and compatibility for Android apps. A formal definition of the IFML model is also given in this paper. The rich semantics of IFML models can elaborate the designs for Android apps, which will further systematically guide the development of these apps during their evolution. Moreover, these IFML models are used to check the consistency between design and implementation in the form of testing. In this way, the effort of writing test cases is reduced and productivity is enhanced as the apps evolve. A tool for modeling and testing for Android apps with IFML is presented, called ADAMANT. To verify the proposed approach's feasibility, ADAMANT is applied on five real-world apps. The results show that the use of the extended IFML in Android app development is effective, and the IFML models can directly be used for testing, ensuring the design is in consistent with the implementation. In this manner, it ensures the quality of development and benefits the sustainable development of apps.

Key words: interaction flow modeling language; Android application; model-driven engineering; model-based testing

随着智能机与平板电脑的日渐普及,安卓设备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 Gartner 公司的调研[1],截至 2018 年第一季度,安卓系统在智能手机市场的占有率已经达到 85%.此外,至 2018 年 6 月,单在 Google Play 应用商店中可获得的安卓应用已超过 330 万[2].毫无疑问,安卓设备以及安卓应用已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功能的同时,还需要应对各种各样的操作环境的变化[3],导致其应用复杂度也呈现指数级的增长.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和系统版本.而其多样化和碎片化,使得同一个应用经常需要开发多个版本,用于适配不同版本的安卓系统和设备.这些因素综合在一起,使得相应的开发与测试中的工作量大为增加.

针对上述问题,模型的使用是一种比较常见的方式.通过模型来描述并设计安卓应用,可将原本繁琐的应用主体分解为不同的模块逐个处理,降低了开发的整体难度.模型的使用可以将安卓应用中设备、系统版本相关和无关的部分相分离,使得应用的逻辑设计这一类不因设备和系统版本变化的部分得到充分复用,避免重复开发.目前,国际上已有学者将各类模型应用于安卓应用的设计和开发.Parada 等人采用 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];Heitkötter 等人针对数据驱动应用,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桌面应用设计,没有考虑安卓应用本身的特点.与传统的桌面应用不同,安卓应用由事件驱动,大量依赖于图形化用户界面(GUI)[6],这就需要有能够有效对 GUI 进行建模的语言.交互流建模语言(interaction flow modeling language,简称为 IFML)[7]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用户之间的交互.它能够与其他基于 UML 的模型兼容,有良好的可扩展性与兼容性.它以事件流为驱动,重视用户、系统事件以及人机交互,能够有效地应对安卓应用基于 GUI 的事件驱动特性.因此,本文将研究基于 IFML 对安卓应用建模的方法.

根据用户的需求,在设计阶段,面向安卓应用建立的 IFML 模型,能够有效地描述安卓应用中极为重要的 GUI 结构以及各 GUI 元素间的交互关系,即工作流的传递,并利用其强大的表达能力促进开发人员对应用设计的认识,指导开发工作系统、有效地进行,减少理解误区.此外,IFML 模型所包含的丰富语义使得依据该模型进行的测试效果极为有效.这些测试的执行能够保证应用实现与设计,即 IFML 模型保持一致,从而保证开发质量.同时,在应用的演化过程中,开发人员可以通过增量式修改 IFML 模型来驱动演进工作的进行.在已建立 IFML 模型的基础上,开发人员仅需要低额的工作量便可完成其上的修改工作,这使得 IFML 建模的初期成本被不断稀释.该 IFML 模型还拥有广泛的应用前景,通过提高自动化的方式,无论是代码的自动化生成还是测试的自动化生成与执行,都能有效地减少开发成本,提高开发效率.然而当前的 IFML 广泛应用于多平台应用的建模,在描述安卓应用时太过宽泛,对安卓应用端特有元素的描述能力较为欠缺,因此将 IFML 应用于安卓应用开发的效果将大打折扣.为此,本文研究了 IFML 面向安卓端的扩展,以此提高 IFML 对安卓应用的描述能力.基于该扩展

后的 IFML,使用者能够建立与安卓应用更加契合的模型,用于安卓应用的设计、开发和持续发展.

本文的工作与贡献主要包括 4 个方面.(1) 本文提出了面向安卓平台的 IFML 扩展,以更好地对安卓应用进行 IFML 建模.(2) 对于 IFML 标准中非形式化的部分及本文提出的扩展部分,我们都给出了形式化的定义,为基于 IFML 模型的各类开发和测试方法奠定了理论基础.(3) 探索了面向安卓端的 IFML 的应用前景,针对安卓应用的测试需求开发了基于 IFML 模型的安卓应用的建模与测试工具 ADAMANT.利用该工具,使用者可以以图形化的方式建立 IFML 模型,并利用模型生成测试用例并执行.(4) 基于 ADAMANT,本文为当前 5 个真实的安卓应用进行了 IFML 建模,并基于模型对应用进行了测试.其良好的测试结果表明了将 IFML 应用于安卓应用的建模以推进开发工作这一方法的可行性.

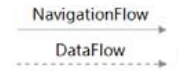
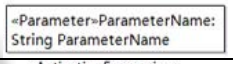
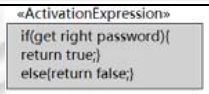
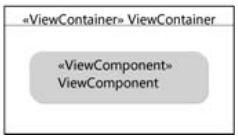
本文第 1 节从总体上介绍 IFML 及其特点,并简要介绍其主体组成结构.第 2 节以一个安卓应用为例描述当前 IFML 在描述安卓应用时的不足,说明扩展 IFML 以用于安卓应用开发工作的必要性.第 3 节有针对性地对 IFML 进行安卓平台的扩展,并对扩展后的 IFML 模型进行实例描述.第 4 节在第 3 节的基础上,提出扩展后的 IFML 模型,以及模型中执行语义的形式化定义.第 5 节介绍本文实现的建模与测试工具 ADAMANT,并将其应用于 5 个安卓应用之上,用于探索面向安卓应用的 IFML 扩展在指导应用开发过程中的可行性.第 6 节探讨并比较在移动应用中使用模型辅助开发、测试,以及对 IFML 扩展这 3 方面的已有工作.第 7 节总结全文,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 IFML 是以图形的形式描述软件系统前端设计中的展示内容、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[7].它在 2013 年被 OMG 协会纳为标准,并在 2015 年发表了 1.0 官方正式文档.

Table 1 The introduction to commonly used elements of IFML model

表 1 IFML 模型中常用元素简介

名称	描述	IFML 图形
交互流(InteractionFlow)	代表数据或视图焦点的转移,一般分为两种:导向流(NavigationFlow)和数据流(DataFlow),它们均可持有一个参数绑定组(ParameterBinding Group),包含多个参数绑定信息(ParameterBinding),表示该交互流转移中参数数据的传递	
参数(Parameter)	表示参数信息,一般不单独存在,而是由其他成员持有,用于参与成员中表达式(Expression)的计算等	
表达式(Expression)	它定义了一组表达式,这组表达式最终会给出一个结果,用于进行模型中的逻辑判断.表达式本身不实例化,而以其子类形式出现,如活动表达式(ActivationExpression)、交互流表达式(InteractionFlowExpression)等	
视图元素(ViewElement)	表示直接显示给用户的前端元素,又分为视图容器和视图组件: ■ 视图容器(ViewContainer).代表的是一个整体性的界面.一个容器内部可以嵌套包含多个视图元素 ■ 视图组件(ViewComponent).代表的是整体性界面上的一个控件.一个显示的部件,不单独显示,由视图容器所包含出现.可能会包含多个视图组件部件(ViewComponentPart),用以辅助显示	
行为(Action)	代表的是一块将被执行的事务逻辑,往往由事件(event)引起触发.同时,它本身会含有对应的行为事件(ActionEvent)在其内部逻辑执行完后触发,并产生新的交互流将视图焦点转到其他交互流元素之上	
事件(Event)	指的是可能会影响该应用系统状态的一个事件,往往会拥有多个可能产生的交互流,但每次只能执行其中一条交互流.选择哪条由事件所含有的交互流表达式(InteractionFlowExpression)来决定	

IFML 的主要目的在于提供软件工程师们用以描述图形化界面应用前端的 IFML 模型(IFMLModel),它可以支持诸如桌面应用、网页应用等等多种终端上的应用,具有较强的泛用性.IFML 模型以树状结构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.本文在表 1 中对 IFML 模型中的常用元素进行了介绍.

大体而言,IFML 模型中存在多个视图容器,代表应用或系统中的界面.每个视图容器可以拥有属于自己的

视图组件来表示该界面上的控件元素.在视图组件上又可包含视图组件部件来对该视图组件进一步补充说明.视图容器和视图组件可以拥有参数元素来辅助表示模型的状态,并且参数的数值可参与表达式的计算,其计算结果用于模型内的逻辑判断.此外,视图容器、视图组件或视图组件部件也可持有事件元素.当事件被触发时,交互流会从该事件导出,传递给其他元素,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将会进行参数的传递.特别地,当导向流指向行为元素时,行为所代表的业务逻辑将会被执行,完成后将会触发其上的行为事件,从而引出新的交互流指向其他元素.

2 案例分析

本节将介绍一个来自 Github 上的安卓开源应用 Good Weather,并使用标准的 IFML 为其建模.Good Weather 用于展示当地的天气信息,该应用可在 Github、Google Play 及 F-droid 上获取.

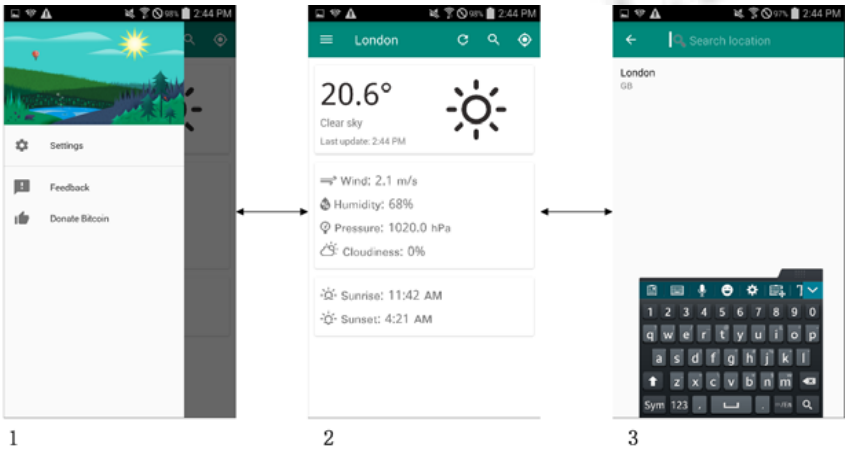


Fig.1 Three views of Good Weather and their relations

图 1 Good Weather 应用的 3 个视图及它们之间的关系

图 1 显示了 Good Weather 这一安卓应用的 GUI 视图,以及各视图之间的跳转关系.其中,图 1-2 为 Good Weather 应用的主界面.它显示了选定城市的天气与相关环境信息.通过点击右上角的 3 个图标按钮可以分别对天气进行刷新、对城市进行文本搜索(如图 1-3 所示)以及进行 GPS 位置搜索.在主界面中,也可通过向右滑动打开其侧边栏(如图 1-1 所示),点击其中的 Setting、Feedback 以及 Donate Bitcoin 文本,也可分别打开设定、反馈以及捐款页面.

图 2 显示的是本文为 Good Weather 应用建立的 IFML 模型中的主界面和侧边栏部分(如图 1-2 和图 1-1 所示).在该 IFML 模型中,本文以视图容器(白色方角框)来表示主界面、工具栏以及侧边栏这类整体的界面.在这些视图容器中,我们以视图组件(灰色圆角框)来表示文本、图片、图标以及按钮这些控件,它们可触发相应的事件(图中白色小圆圈),引起交互流的传递.如 HomeButton 这一视图组件代表了主界面左上角的按钮,其上的事件 touch the button,表示用户在点击这一按钮时将会将交互流导向 SideBar 这一容器,即应用的侧边栏.此外,该模型中也以参数 isLocated 来表示该应用当前是否进行了定位,使用激活表达式来判断是否根据 isLocated 的值来显示详细的天气信息,并在行为 locating 的执行中会动态地修改 isLocated 的值(其具体执行过程由 UML 动态模型表示).此过程中 IFML 原有的模型元素概念太过宽泛,不足以精确描述安卓应用中的独有元素,具体有以下 3 类.

1) 丰富的界面元素.安卓设备较小的设备屏幕,无法同时容纳大量的控件元素,因而需要多个界面来协同完成应用的功能.为此,诸如工具栏、侧边栏等仅占据屏幕一部分的轻量级界面容器被广泛应用于界面间的跳转或导航.它们与占据应用整个屏幕的、用于实现应用主功能的各个界面,在外观和使用方式上都存在较大的

差异,无法由 IFML 中原有的视图容器和视图组件进行有效区分.而对于安卓端的控件,诸如文本、按钮、图片等诸多元素,它们的外形以及其上可触发的事件各异(如文本和按钮上通常可以进行长按、点击操作,而图片则通常可以进行滑动、缩捏操作等等),以原 IFML 中的视图容器和视图组件统一地表示这些元素可能会引起设计和实现中的误解,为此,我们需要对 IFML 中的视图容器和视图组件进行扩展,进而有效地区分这些容器和组件.

2) 多样的用户手势.众所周知,安卓应用也支持多样的用户手势,如点击、长按、滑动等.同时,在同一个控件上,不同的手势操作通常会引起不同的事件行为.例如,在文件管理类应用中,对于以列表形式出现的文件列,普通的点击操作将会打开该文件的详细信息界面,而长按操作则会弹出文件的操作菜单,包含删除、移动、详情等等.原 IFML 仅以事件下的子类型、视图元素事件(ViewElementEvent)来进行统一的表示.因此,在上述的 IFML 模型中,为了区分各事件所代表的具体操作行为,仍需要使用自然语言来进行辅助说明.这一方式不但建模时存在不便,而且在设计和实现时也可能会引入歧义.为此,对于安卓用户手势的扩展也势在必行.

3) 繁多的系统事件.同时,由于安卓设备便携的特点,安卓应用往往还需处理多样的系统事件,如电量变化、网络连接、GPS 定位、传感器感知等类型的事件.在 IFML 中,它们都由事件下的子类型系统事件(SystemEvent)来统一表示.这样的表示方式也过于笼统,因此,本文也对安卓端的系统事件进行了扩展.

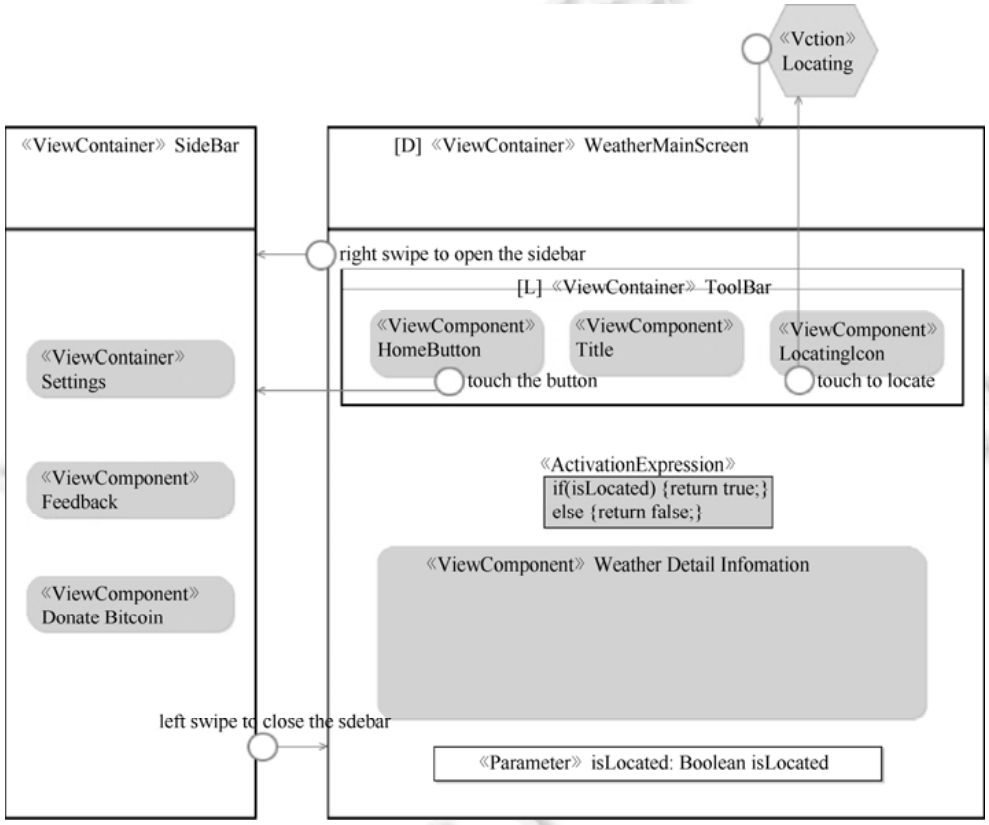


Fig.2 A part of IFML model for Good Weather

图 2 Good Weather 应用的部分 IFML 模型

综上所述,本文拟对 IFML 标准进行面向安卓端的扩展,从而将 IFML 模型应用于安卓应用的建模,在应用的演进过程中持续性地指导应用的设计与实现工作.

3 面向安卓平台的 IFML 扩展

由于原 IFML 在用于安卓应用建模时存在种种不足,本文认为需要对 IFML 进行面向安卓端的扩展.为此,

本文以安卓官方指导文档为基础,并以当前安卓应用中的主流常用元素作为补充,提出了以下 3 个方面的安卓端扩展:(1) 对视图容器与视图组件的扩展;(2) 对表达式和行为的扩展;(3) 对事件的扩展.

3.1 针对视图容器与视图组件的扩展

视图容器代表的是一个整体性的界面,本文为其添加了安卓应用视图容器(AndroidAppContainer)作为安卓端应用中使用的视图容器子类型,并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;抽屉是指可从左/右滑出的侧边栏;工具栏表示应用中具有导航或显示标题等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.通过对这些安卓常用视图容器的特例化,使得建模更加便捷的同时有效区分了各类常用元素.其具体关系结构可如图 3 所示.

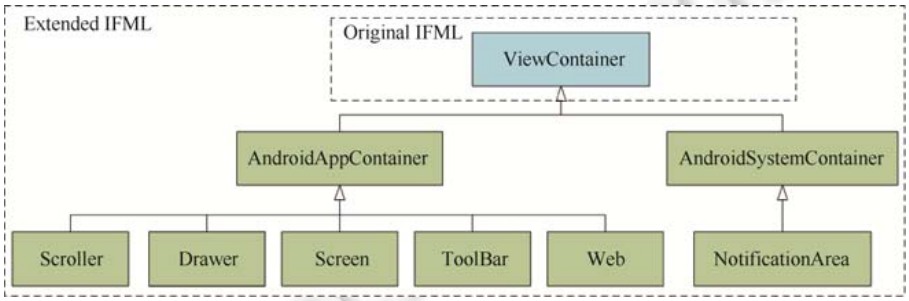


Fig.3 The Android extension to ViewContainer

图 3 视图容器的安卓端扩展

视图组件代表的是视图容器中用于展示信息或接受输入的、并可引发应用事件的组件.IFML 标准中所有的组件均以视图组件来表示,过于笼统.本文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,并给出了以下更加具体的视图组件实例: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)表示会引起状态切换的按钮;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;编辑框(EditText)表示为文本输入框;进度条(ProgressBar)表示为进度条组件.本文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如图 4 所示.

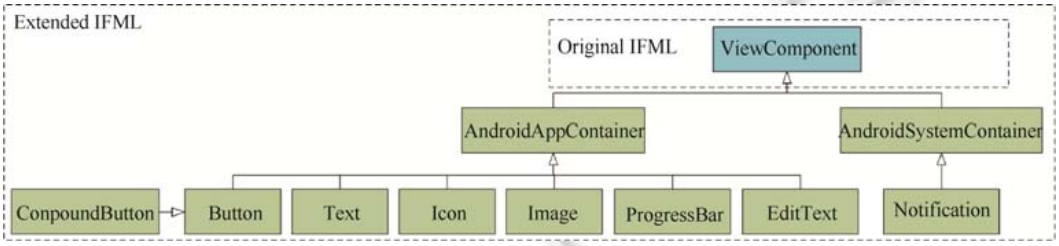


Fig.4 The Android extension to ViewComponent

图 4 视图组件的安卓端扩展

此外,针对安卓系统上一些独有的界面元素,相应地,本文使用安卓系统视图容器(AndroidSystemContainer)与安卓系统视图组件(AndroidSystemComponent)来表示.本文亦在其下创建了一组实例:通知区域(Notification Area)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.该组实例能够简化建模过程,并且由于安卓系统中通知区域的唯一性,该组实例能够与之对应.

3.2 针对表达式与行为的扩展

表达式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两类:交互流表达式,用于判断对应的事件在当前状况下最终触发了哪些交互流;激活表达式,用于判断该表达式所属元素是否可见/可触发.本文还额外定义了两种表达式,来辅助进行IFML模型中的逻辑判断,分别是:

(1) 定义表达式新的子类型,默认表达式(DefaultExpression).作为表达式逻辑中重要的参与元素,参数持有一个默认表达式类型的属性defaultValue,它将赋予该参数的最初值.然而,原IFML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,即计算时可以修改参数数值的新表达式子类型,执行表达式(ExcutionExpression).在原IFML中,行为用于表示由事件所引起的事务逻辑.其具体执行需要借用UML活动图等动态模型来表示.然而,对于通常的安卓应用开发来说,多个模型的使用以及模型间的频繁切换太过繁琐.为此,本文在行为下添加一个执行表达式类型的成员executionContent,来轻量级地描述某行为中的具体逻辑.通过执行executionContent可以修改部分参数的数值,表示行为所代表的业务逻辑对模型状态的影响.

此外,安卓终端也有许多诸如通话、摄像、照相等设备独有的行为,本文也为其进行了安卓终端的扩展:扩展了安卓行为(AndroidAction),用以表示安卓应用中所引发的事务逻辑,并添加了安卓行为事件(Android Action Event)与之对应,表示为安卓行为执行结束后所触发的行为事件.同时,本文添加了相机行为(Camera Action)和相机行为事件(CameraActionEvent)这一对实例,用于表示照相/录像的行为以及对应的行为事件;类似地,还有话筒行为(MicrophoneAction)与话筒行为事件(MicrophoneActionEvent),此处不再一一详述.这些安卓行为能够与安卓上的特有行为相对应,简化对该类复杂行为进行的繁杂建模过程,提高了模型的可用性.

3.3 针对事件的扩展

除了第3.2节中扩展的行为事件外,在第2节中,我们提及原IFML用视图元素事件与系统事件分别表示安卓应用中丰富的用户手势和安卓系统所引起的繁多的事件.为此,本文也对这两类事件进行了扩展.

本文首先在系统事件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了 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(Connection Event),表示系统连接产生的系统事件,最终的扩展结构可如图5所示.

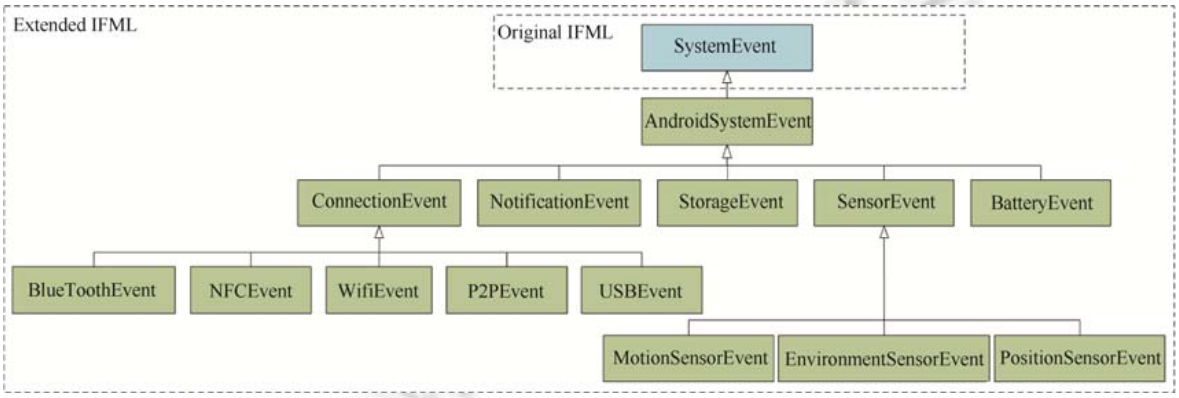


Fig.5 The Android extension to SystemEvent

图5 系统事件的安卓端扩展

同时,我们还对其中的连接事件和传感器事件作了进一步的划分: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(Environment SensorEvent)和位置传感器事件(PositionSensorEvent),其下可继续进行划分,这里不再详细描述.

安卓端提供了通过压力和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.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,在其下还增加了 8 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(Long PressEvent),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(Drap DropEvent),表示为用户拖拽行为触发的事件.同时,与第 3.1 节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.最终的扩展结构可如图 6 所示.上述事件的安卓扩展,不仅方便了建模过程,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 建模带来极大的方便的同时减少了用非形式化语言描述时可能引起的歧义.

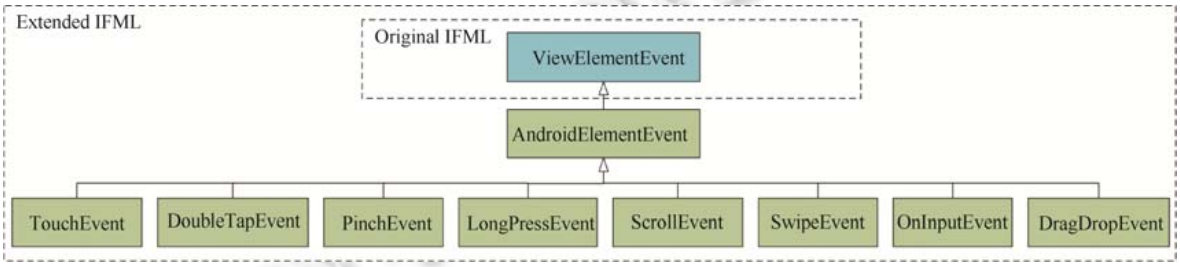


Fig.6 The Android extension to ViewElementEvent

图 6 视图元素事件的安卓端扩展

3.4 扩展后的IFML模型实例

仍以第 2 节中驱动案例 Good Weather 为例,使用该扩展后的 IFML 为其建模也会更加准确.图 7 展示了最终建立的扩展后的 IFML 模型,并在各元素附近进行了标号.

对于图 1-1 中 Good Weather 所拥有的主界面,我们以扩展后的屏幕元素来表示,即图中 WeatherMainScreen 元素(v_{c1}),在开发中它对应于一个 Activity(安卓中最基本的活动应用组件).随后可以看到主界面上存在大量的控件用于显示天气信息,在 IFML 模型中,我们将这些控件整合起来,使用特殊的类型 Detail 来简化表示,即元素 Weather Detail Information(v_{p4}).在主界面的上方,存在一条工具栏,它在实现中作为 Activity 的特殊附属控件 ToolBar 或 ActionBar 而存在,因此在扩展后的 IFML 中,我们将它以工具栏元素 ToolBar(v_{c3})嵌套在 WeatherMain Screen 元素内,来表示它的附属关系.而在该工具栏上,存在各式的按钮、文本以及图标,它们在开发中作为 Button、Text 以及 ImageButton 等类型的控件而出现.在 IFML 中则对应于本文扩展的按钮 Home Button(v_{p1})、文本 Title(v_{p2})和图标 LocatingIcon 元素(v_{p3})(这里,为了简化只显示了其中的定位图标).随后,图 1-2 中应用的侧边栏,在开发中以 DrawerLayout 这一布局来实现,并且通常为整个应用的所有 Activity 所共享,为此,我们在 IFML 模型中以抽屉元素 SideBar 来表示(v_{c2}),并独立于所有其他的视图容器类元素.其上也包含这一侧边栏中的各类文本组件(v_{p5},v_{p6},v_{p7}).同时,在主界面和侧边栏可通过左右滑动来相互切换,因此可使用扩展后的滑动事件,即图中的 right swipe 和 left swipe 元素以及它们所引出的交互流(e_2,i_2 和 e_5,i_5)来表示,在开发中,它们通常对应于滑动类型的事件监听器.在工具栏上的 Home 按钮以及定位图标上的点击事件也类似于此(e_1,i_1 和 e_3,i_3).

可以看到,文中对视图容器、视图组件以及事件的扩展可帮助开发者有效地区分各个不同类型的界面、控件以及事件.同时,这些 IFML 元素也能与各类开发组件相对应,因此能够有效地指导开发工作系统地进行.

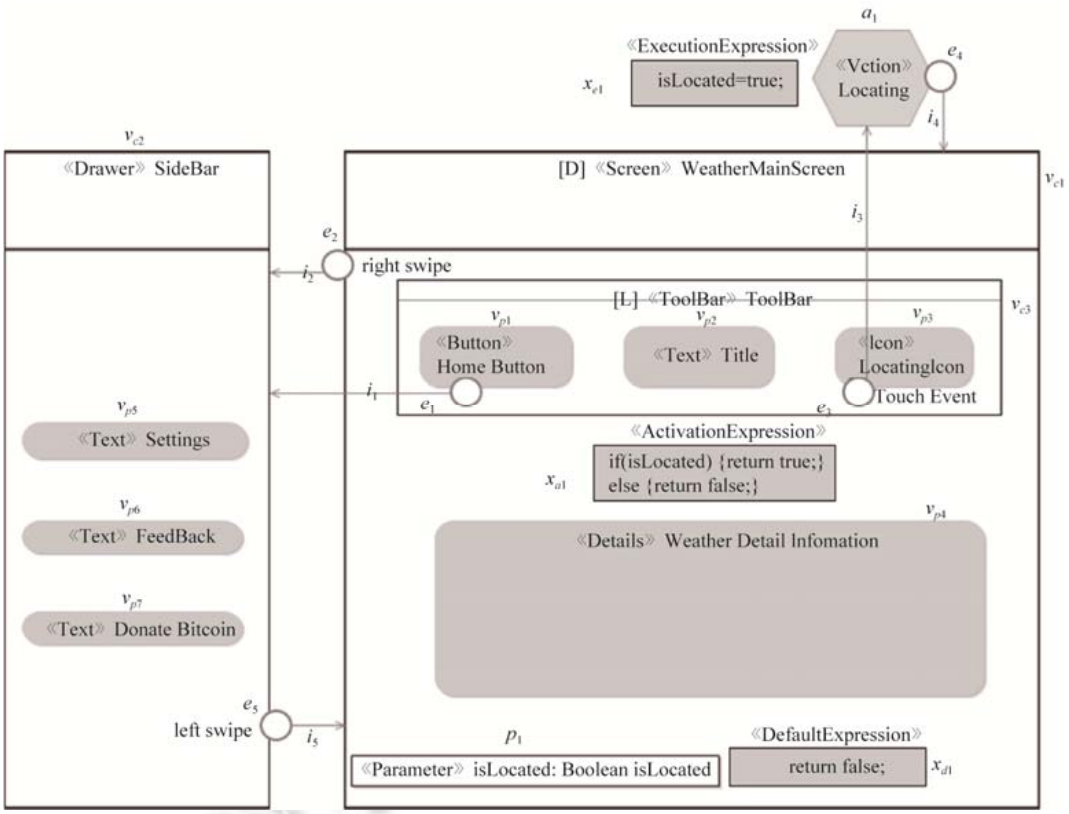


Fig.7 A part of IFML model for Good Weather after the Android extension

图 7 扩展后 Good Weather 应用的部分 IFML 模型

此外,该应用中与事件交互相关的逻辑我们也可使用参数、表达式以及行为来表示.在该应用中,用户需要先进行定位(元素 v_{p3} 上的触摸事件 e_3),随后应用根据地点位置请求天气信息数据(我们将这个请求过程用行为元素 Locating 来表示,即 a_1),并最终将这一数据在组件 Weather Detail Information 中显示出来.这一串逻辑的具体内容也需要参数和表达式的帮助:我们用参数 isLocated(p_1)来表示该应用是否进行了定位操作并获取了天气信息,同时,该参数使用图中默认表达式(x_{d1})将其初始化为 false,即最初并未进行定位.根据设计意图,仅当进行了定位,即参数 isLocated 为 true 时,才会显示详细的天气信息,即组件 Weather Detail Information 的显示.因此,我们将激活表达式(x_{a1})的内容设置为 if(isLocated){return true;}else{return false;},使其计算结果与参数 isLocated 保持一致,并由组件 Weather Detail Information 持有该激活表达式.显然,其显示状态最终与参数 isLocated 保持一致,从而达到了我们的设计要求.对应地,Locating 行为具体的逻辑执行可以由这一执行表达式(x_{e1})表示:我们以 isLocated=true 轻量地表示参数 isLocated 所代表的定位,发送请求,信息展示这一连串逻辑的执行,并且执行完成后 isLocated 的值发生了改变,因此也将最终决定组件 Weather Detail Information 显示与否.

通过执行表达式的使用,我们将应用的内部逻辑以表达式的形式进行轻量级的建模,一定程度上避免了使用 UML 动态模型进行详细的行为建模的繁琐过程,同时将 IFML 中的逻辑表示仅限于 IFML 本身,避免了多模型相互引用的情形,最终降低了建模的复杂度,简化了建模过程.

本文的安卓端扩展,实际上在 IFML 模型的泛用性(抽象能力)以及表现力上达到了一个均衡点.利用该扩展后的 IFML 为安卓应用进行建模时更加方便、快捷,在降低建模的复杂度的同时也使得模型所展现的设计意图更加精准、明确.同时,通过将安卓端常用元素加入扩展中,使得建立的 IFML 模型也更加贴近实际的安卓应用.在开发过程中,它能够作为设计标准指导实现工作的进行,避免实现工作偏离设计意图所导致的额外工程量,因

此,能够有效地提高开发效率.此外,基于该扩展后的 IFML 模型,也可实现部分开发工作的自动化.典型的方式,例如利用这一建立/修改后的IFML模型,自动化生成实现代码,或是生成相应的测试用例,驱动应用的开发/演进工作的进行,并最终利用这些用例检验应用开发是否正确,都能极大地提高安卓应用的开发效率.为此,本文也研究了对扩展后的 IFML 模型的形式化定义,使得通过 IFML 模型自动化生成测试用例或是实现代码,都成为可能.

4 IFML 模型的形式化定义

IFML 标准文档仅对模型的语法进行了形式化的定义,而其中的语义则采用了自然语言这一类非形式化的方式进行了描述,因此,标准中的许多语义不够准确.为了更好地支持 IFML 模型在安卓应用开发中的使用,我们对标准中模糊的部分进行了严格的定义,并连同前文提出的安卓端扩展,对扩展后的 IFML 模型进行了形式化的定义.更重要的,本文也对 IFML 模型的动态语义进行了形式化定义,从而使得 IFML 模型中事件触发、交互流传递以及模型状态迁移都有唯一的解释.因此,在使用该 IFML 模型中的定义来描述安卓应用中的界面构成,以及界面间的交互时都极为准确,避免了该 IFML 模型在作为设计意图体现时可能产生的歧义.

首先,我们给出 IFML 模型中各个重要元素的形式化定义.

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 用于标记该表达式所属的具体类型;l 表示解析当前表达式所使用的语言,如 Java、OCL 等;b 以文本形式存储了该表达式的表示.本文将 X 按照不同的类型划分为不同的集合,由第 3.2 节已知,共有 4 个子类型,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 IFML 模型中视图元素的集合.$VE$ 可进一步分为视图容器和视图组件,记为 $VE=VCT\cup VCP$,其中,VCT 表示视图容器的集合,VCP 表示视图组件的集合.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);$vc\in VCT$,表示该容器/组件的父容器.ax 与 SE 均可为空,当 ax 为空时,该视图容器/组件默认激活.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 IFML 模型中的视图组件部件的集合.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,pvp 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 IFML 模型中行为的集合.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;$ex\in X_E$,即本文扩展的 *execution Content*,用于表示其内部逻辑对应的执行表达式.

定义 5. $E=\{e_1,e_2,\ldots,e_n\}$为 IFML 模型中事件的集合.对于任意 $e\in E$,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;$ix\in X_I$ 为交互流表达式,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 为空时,表示该事件默认活跃,当 ix 为空时,会触发 SI 中所有交互流.本文按照扩展内容,将事件 E 分为 3 个类型,记为 $E=E_S\cup E_A\cup E_V$,其分别表示为系统事件、行为事件以及视图元素事件的集合.当某一事件属于视图元素事件时,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 $ae=(et,ax,ix,SI,pe)\in E_A$,则有 $pe\in A$.

定义 6. $I=\{i_1,i_2,\ldots,i_n\}$为 IFML 模型中交互流的集合.交互流可分为导向流与数据流,记为 $I=N\cup D$,对于任一 $i\in I$,均有 $i=(it,s,t,PBG)$.其中,it 表示该交互流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,$t\in VE\cup VP\cup A$,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 $i\in SI$,则 $t\notin A$.

$PBG=\{pb_1,pb_2,\ldots,pb_n\}$为参数绑定组,表示该交互流所携带的数据绑定信息的集合.对于任意 $pb\in PB$,其表示为一条参数绑定信息,有 $pb=(sp,tp)$,表示为在当前状态下将源参数 sp 的值传递给目标参数 tp.

定义 7. P 表示为 IFML 模型中被使用到的参数的集合,有 $P=\{p_1,p_2,\ldots,p_n\}$,对于任意 $p\in P$,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 为入参或是出参;te 表示该参数的具体类型;$dx\in X_D$,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 $M=(V,A,E,I,X,P,v_I)$,其中,

- $V=\{v_1,v_2,\ldots,v_n\}$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 $V=VE\cup VP$.
- $A=\{a_1,a_2,\ldots,a_n\}$为行为的集合.
- $E=\{e_1,e_2,\ldots,e_n\}$为事件的集合.
- $I=\{i_1,i_2,\ldots,i_n\}$为交互流的集合.
- $X=\{x_1,x_2,\ldots,x_n\}$为表达式的集合.
- $P=\{p_1,p_2,\ldots,p_n\}$为参数的集合.
- $v_I\in V$ 为该 IFML 模型最初情况下显示的视图容器.

在给出扩展后 IFML 模型的语法后,本文也给出该 IFML 模型的动态语义,即对 IFML 模型的模型状态和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,但是由于其复杂性,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从而导致了状态的转化,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交替序列来表示其执行路径.IFML 模型的运行时状态,可记为 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的集合,以及参数和它们当前值的映射集合.形式化地,状态 CS 可定义为一个二元组(CM,CA),CM 为上述的激活元素集合,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 $P\to U$,其中,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.

为了更好地描述状态 $CS=(CM,CA)$,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 true 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 $v_i\in V$ 时,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_i,有 $Lv(v_i,pv_i)$=true 成立.据此定义,该 Lv 函数可继续递归于 pv_i 的父节点.特别地,当可见元素 $v_i\in A$ 时,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),则当前所有的默认可见元素可用集合$\{x\in A\cup V|Lv(v_i,x)=\text{true}\}$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 中的执行表达式来改变 CA 中元素的数值,从而得到新的映射集合 CA'来代替 CA.
- 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 $e_i=(et,ax,ix,SI,pe)\in E$,其下的交互流表达式 $ix\in X_I$,存在函数 $eval(ix,CA)=SI'$,SI'表示为 ix 所对应的事件在当前状态下所触发的交互流的集合,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 $enable:CM \rightarrow E$,$enable(CM)$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 $e=(et,ax,ix,SI,pe) \in enable(CM)$,均需满足 $eval(ax,CA)=$true,$pe \in CM$.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 $a=(at_a,ax_a,SE_a,vct_a,ex_a)$,其中的执行表达式,即 $CA_i' \leftarrow exec(ex_a,CA_i)$.

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,$SNI \subset SI'$.对于交互流 $i=(it_i,s_i,t_i,PBG_i) \in SI'$,需要依次执行 $CA' \leftarrow exec(PBG_i,CA_i')$,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 $CM'=CM-\{x \in A \cup V | Lv(pe_e,x)=\text{true} \&\& x \in CM\}$.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 $CM'=CM' \cup \{x \in A \cup V | Lv(t_i,x)=\text{true} \&\& eval(x.ax,CA')=\text{true}\}$.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 $M=(V,A,E,I,X,P,v_I)$,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:

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

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.特别地,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 $CA=\varnothing$,对该 IFML 模型中所有的参数 $p=(d,te,dx) \in P$,逐次执行 $CA \leftarrow exec(dx,CA)$,最终即可得到其初始的映射集合 CA_0.而已知 v_I 表示为该 IFML 模型的初始视图容器,则有 $CM_0=\{x \in A \cup V | Lv(v_I,x)=\text{true} \&\& eval(x.ax,CA_0)=\text{true}\}$,可以获得最初的 CM_0.

不妨仍然以图 7 所示的 IFML 模型为例.由第 3.4 节中的建模过程可知,对于该 IFML 模型 M,有 $M=(V,A,E,I,X,P,v_I)$,其中有 $V=\{v_{c1} \sim v_{c3},v_{p1} \sim v_{p7}\}$,$A=\{a_1\}$,$E=\{e_1 \sim e_5\}$,$I=\{i_1 \sim i_5\}$,$X=\{x_{e1},x_{a1},x_{d1}\}$,$P=\{p_1\}$,且有初始视图容器 $v_I=v_{c1}$,即为图 1-1 所示的主界面(即 WeatherMainScreen 元素,其最左端标有 D 字样,含义为 Default,表示该最外层的视图容器即为初始容器).此时,初始状态 $CS_0=(CM_0,CA_0)$.最初 $CA_0=\varnothing$,对于模型中的唯一参数 $p_1=(d_1,te_1,x_{d1})$,对其初始化,有 $CA_0 \leftarrow exec(x_{d1},CA_0)$,$CA_0=\{p_1 \rightarrow \text{false}\}$为初始的 CA.并且,已知当且仅当 $v_x \in \{v_{c1},v_{c2},v_{p1},v_{p2},v_{p3},v_{p4}\}$,有 $Lv(v_{c1},v_x)=$true(表示当 v_{c1} 显示时,这些元素也默认显示).而其中 $v_{p4}=(v_{c1},x_{a1},\varnothing,\text{null})$,其显示状态受激活表达式 x_{a1} 影响,在初始状态 CA_0 下 $eval(x_{a1},CA_0)=$false,即不显示.故,$CM_0=\{x \in A \cup V | Lv(t_i,x)=\text{true} \&\& eval(x.ax,CA_0)=\text{true}\}$,即 $CM_0=\{v_{c1},v_{c2},v_{p1},v_{p2},v_{p3}\}$.至此,初始状态 $CS_0=(CM_0,CA_0)$的初始化完成.

随后,通过函数 $enable(CM_0)=\{e_1,e_2,e_3\}$可获得当前可触发的事件.不妨假设此时执行事件为 e_3,记为 $e_3=(et_3,\text{null},\text{null},SI_3,v_{p3})$.有 $SI_3=\{i_3\}$,有且仅有 1 条交互流,即为 e_3 唯一能触发的交互流,其中,$i_3=(it_3,e_3,a_1,\varnothing)$,可知视图焦点转移到行为 a_1 上,$a_1=(at_1,\text{null},SE_1,\text{null},x_{e1})$.此时,既无参数传递,也无执行表达式,故 CA 不变,有 $CA_1=CA_0=\{p_1 \rightarrow \text{false}\}$.同时,$CM$ 首先去除失去焦点的元素,即触发事件 e_3 的父元素 v_{p3},$CM_1'=CM_0-\{x \in A \cup V | Lv(v_{p3},x)=\text{true} \&\& x \in CM_0\}$,有 $CM_1'=\varnothing$.随后,CM 再加上重新获得视图焦点的元素,即 a_1.而 $a_1 \in A$,由之前的函数定义可知,有且仅有 $Lv(a_1,a_1)=$true,且 a_1 未持有激活表达式,故 $CM_1=CM_1' \cup \{x \in A \cup V | Lv(a_1,x)=\text{true} \&\& eval(x.ax,CA_1)=\text{true}\}$,$CM_1=\{a_1\}$.通过执行事件 e_3,我们实现了状态转移:

$$CS_0=(CM_0,CA_0) \xrightarrow{e_3} CS_1=(CM_1,CA_1).$$

而在状态 CS_1 下时,$enable(CM_1)=\{e_4\}$,$e_4=(et_4,\text{null},\text{null},SI_4,a_1)$.此时,需执行 a_1 中的执行表达式 x_{e1},因此有 $CA_2 \leftarrow exec(x_{e1},CA_1)$,最终 $CA_2=\{p_1 \rightarrow \text{true}\}$.同时,$SI_4=\{i_4\}$仅有 1 条交互流,$i_4=(it_4,e_4,v_{c1},\varnothing)$即为所需执行的交互流,其目标元素 v_{c1} 将获得视图焦点.在去除失去焦点的元素 a_1 后,$CM_2'=\varnothing$.已知当且仅当 $v_x \in \{v_{c1},v_{c2},v_{p1},v_{p2},v_{p3},v_{p4}\}$,有 $Lv(v_{c1},v_x)=$true,而且 $CA_2=\{p_1 \rightarrow \text{true}\}$,$eval(x_{a1},CA_2)=$true,故与初始状态 CS_0 不同,元素 v_{p4} 也将加以显示,即 $CM_2=CM_2' \cup \{x \in A \cup V | Lv(v_{c1},x)=\text{true} \&\& eval(x.ax,CA_2)=\text{true}\}$,最终,$CM_2=\{v_{c1},v_{c2},v_{p1},v_{p2},v_{p3},v_{p4}\}$,故我们实现了第 2

步状态转移 $CS_1=(CM_1,CA_1) \xrightarrow{e_4} CS_2=(CM_2,CA_2)$.

从 IFML 模型的初始状态 $CS_0=(CM_0,CA_0)$开始,按照上述示例不断地通过 enable 函数选择一个当前可触发事件执行并更新状态,我们便可获得一条可执行路径

$$CS_0 \xrightarrow{e_0} CS_1 \xrightarrow{e_1} \ldots \xrightarrow{e_{n-1}} CS_n.$$

通过对扩展后的IFML 模型中各元素的属性以及各元素间关系的形式化定义,在将这些元素转化为安卓应用中的组件元素,即代码的生成时,能够形成固定的转化规则,使得自动化代码生成成为可能;而对 IFML 模型中模型状态以及可执行路径的定义,则制定了 IFML 模型状态间通过触发事件来互相转化的转化规则,从而使得自动化地遍历 IFML 模型生成测试用例并执行成为可能,本文也对这一部分内容加以实现,可详见下一节.

5 案例研究与分析

经过本文上述章节对 IFML 进行的安卓端扩展,在安卓应用开发中使用该扩展后的 IFML 进行对应用需求、设计的建模,能够使得应用的设计以更精确、更加贴合实际安卓应用的方式来加以展现.并且,通过文本对该扩展后的IFML 模型的形式化定义,这一 IFML 模型除了在设计和实现上起到这一指导性的作用外,还能具有诸如自动化测试之类的具体用途.基于该 IFML 模型的测试能够检测应用的实现是否与应用的设计保持一致,从而保证应用在开发过程中的正确性,这是设计模型的一种常见使用场景.本节将探索扩展的 IFML 在安卓应用基于模型的测试这一场景下的可行性.为此,本文首先实现了基于扩展的 IFML 模型的建模与测试工具 ADAMANT,并选取了 5 个当前较为知名的开源安卓应用作为案例来进行分析研究.

5.1 基于扩展的IFML模型的建模与测试工具ADAMANT

工具 ADAMANT 源自于 IFML 官网上一个基于 Sirius 项目[8]的 IFML 建模项目.我们对该项目按本文面向安卓应用的 IFML 扩展进行了修改和完善.最终,ADAMANT 以 eclipse 插件的形式集成起来,使用它即可在 eclipse IDE 上以图形化的方式为安卓应用进行 IFML 建模.如图 8 所示,即为我们提供的 IFML 建模视角.

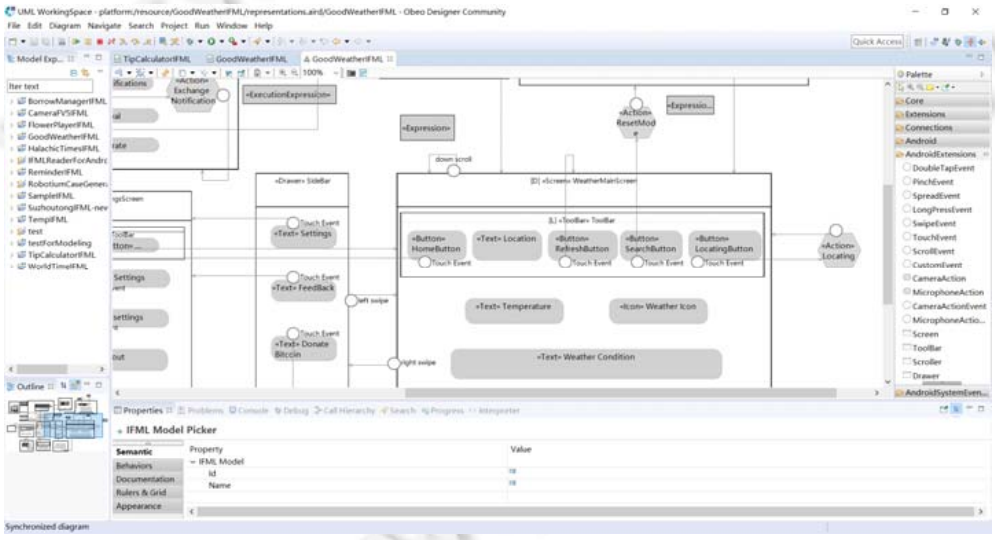


Fig.8 The modeling perspective of ADAMANT

图 8 工具 ADAMANT 的建模视角

在如图 8 所示的视角中,用户在建立相应的 IFML 模型项目后,可以从右侧工具栏内选择不同的包,从中选择不同类型的 IFML 元素,将其拖拽至主界面,这即表示在该 IFML 模型中建立了相应的元素实例;在选中 IFML 模型中的元素实例后,也可以在下方的属性标签栏中对该元素的属性进行设置.此外,右侧工具栏中,Android、

AndroidExtensions 等即为本文对该工具的安卓端扩展,利用这些包内元素,可以简化建模过程,使用者也可以创建具有安卓特点的元素实例,使得建立的 IFML 模型更加贴合安卓应用.

随后,我们也为 ADAMANT 添加了测试功能——当使用者根据该 IFML 模型中的设计实现了相应的安卓应用后,ADAMANT 可以根据这一 IFML 模型自动化地生成相应的测试用例,并在给定的设备上自动化地执行,从而检验最终实现的应用制品是否与原本的应用设计保持一致.具体而言,ADAMANT 读出了 IFML 模型数据后,根据第 4 节中对 IFML 模型以及其中可执行路径的定义,ADAMANT 通过模型遍历的方式,生成大量的可执行路径,并可按不同的事件覆盖准则选择路径集合将其转化为测试用例并执行.

这个过程中,最为重要的即为模型遍历,图 9 给出了该遍历算法的伪代码表示.算法主体,即 pathsTraversing 函数的入参 CS 表示为当前模型的状态,Paths 表示为生成测试用例的集合,SE 则表示为当前路径中事件的集合.模型的遍历以初始状态 CS_0 开始执行这一 pathsTraversing 函数,Paths 和 SE 均为∅.在该函数中,它通过 enable 函数获取当前状态下任何可能触发的事件,遍历选择事件加入当前的路径的事件集合 SE 中(第 2 行~第 4 行).若达到中止条件时(第 5 行,中止条件诸如当前路径中的事件数达到了预先设定的最大值),则检查当前事件是否未被覆盖并对该路径真正可执行进行检测(第 6 行).如果可行,便会根据当前的可执行路径生成对应测试用例并将其加入 Paths 中(第 7 行~第 9 行);而若未达到中止条件,则通过 possibleSIGroup 函数获取该事件可能触发的交互流(第 10 行~第 11 行),并根据选择的交互流更新模型状态为 *CS′*,递归上述 pathsTraversing 函数(第 12 行~第 13 行).此过程中获得的 Paths 集合即为根据该 IFML 模型生成的测试用例集合.最后,ADAMANT 在测试框架 Robotium[25]的支持下,将这些测试用例运行在待测应用之上,并记录测试执行时间、测试错误日志等测试执行信息.

```
Algorithm 1. Path Traversing Algorithm.
Input:1) CS;2) Paths;3) SE;
Output: Test sequences starting from CS.
1.  procedure pathsTraversing(CS,Paths,SE)
2.      candidateEvents⇐enable(CS.CM);
3.      for event∈candidateEvents do
4.          SE⇐SE+event;
5.          if reach end conditions then
6.              if uncovered(SE) && check(SE) then
7.                  path⇐generatePath(SE);
8.                  Paths⇐Paths+path;
9.              end if
10.         else
11.             for SI∈possibleSIGroup(event) do
12.                 CS′⇐generateCS(CS,event,SI);
13.                 pathsTraversing(CS′,Paths,SE);
14.             end for
15.         end if
16.         SE⇐SE-event;
17.     end for
18.     return Paths;
19.end
```

Fig.9 The algorithm of generating test cases via traversing the IFML model

图 9 遍历 IFML 模型生成测试用例算法

在实际应用中,ADAMANT 允许开发人员将应用设计意图以 IFML 模型的形式来展示,从而指导应用的开发工作,使开发人员达成统一的设计认知,避免误解.同时,ADAMANT 根据 IFML 模型生成的测试用例,可作为应用实现和设计是否保持一致的一个检验标准,当测试用例执行失败时,则说明当前的应用很可能与设计意图相悖,并且 ADAMANT 记录的错误日志也可帮助开发人员定位异常点.在应用的演进过程中,ADAMANT 的效果则会更加明显:开发人员根据新的需求使用 ADAMANT 对 IFML 模型进行修改,以指导应用的修改工作,而 ADAMANT 高度自动化的测试功能,则避免了在不断演进过程中测试用例的重复编写,提高了开发效率.

5.2 ADAMANT应用建模分析

由于应用的开发工作是一个较为漫长的过程,本文没有在设计时就开始利用 ADAMANT 进行建模,随后实现应用并使用 ADAMANT 的测试功能检验开发工作的正确完成.替代性地,本文直接选择了当前已开发好的 5 个知名的应用.这 5 个应用均来自 Wiki 和 Github 所提供的安卓知名开源应用的列表[9,10].我们选定了这些应用最新发布的版本,作为本文的实验版本.并且,根据对这些应用的使用体验、核心源代码理解等获取的信息,我们使用 ADAMANT 对这些应用逆向地进行了手工 IFML 建模.最终,应用 ADAMANT 的测试功能,基于这些 IFML 模型,对应用进行了相应的测试.表 2 展示了这些应用的基本信息以及 IFML 模型信息.

Table 2 The informations of these 5 five apps and their IFML models

表 2 5 个应用的基本信息以及 IFML 模型信息

应用	应用详细信息				IFML 模型信息					
	版本	类型	代码行数	Activity 数	视图容器	视图组件	事件	交互流	表达式	建模时间(h)
OwnCloud	2.0.1	办公	72 862	12	65	136	195	203	72	20
ConnectBot	1.9.2	通信	28 791	12	52	112	149	154	49	13
Ringdroid	2.7.4	音频	5 295	4	15	52	57	61	19	4
Kiwix	2.2	数据	11 270	8	38	121	108	115	63	8
Omni-notes	5.4.1	办公	20 305	9	58	121	167	172	70	12

表 2 中展示了这 5 个应用的版本、类型、代码行数以及应用代码中 Activity 的数量(Activity 为安卓应用中最重要的应用组件,一个 Activity 粗略地代表应用的一个主界面).受文章篇幅所限,文本在此无法给出这些应用的 IFML 模型的全部信息,在表中仅给出了这些 IFML 模型中各类重要 IFML 模型元素,即视图容器、视图组件、事件、交互流以及表达式的数量.此外,表中也给出了建立这些 IFML 模型所花费的时间.

总体而言,Activity 数量以及代码行数体现了这些应用的规模,而我们建立的 IFML 模型的规模也与应用基本保持一致.虽然建立这些 IFML 模型花费了 4~20 小时不等,但考虑到我们所选择的均是当前具有较长发展历史、具有一定规模的成熟应用,相较于它们数月的开发时间来说,花费 20 小时以内从头建立这些 IFML 模型的代价是可以接受的.并且,这些成本仅为初次成本,随着应用的不断演进,开发人员仅需要低额的工作量便可在已有 IFML 模型上进行修改,因此,整体的建模成本将会被不断稀释.然而,该 IFML 模型在开发中的指导作用可有效地分解开发复杂度,促使开发工作高效、系统地进行.并且,基于该 IFML 模型进行的测试能够有效地提高测试效果,并且避免了在应用开发迭代中编写测试用例的重复劳作.此外,该 IFML 模型也拥有广泛的应用前景,伴随着自动化程度的提高,基于该 IFML 模型的诸如自动化代码生成的研究也将被推进,因此,可进一步解放人工劳力,提高开发效率.据此,我们认为建立 IFML 模型所花费的时间是值得的.

5.3 ADAMANT基于模型测试实验

为了说明 IFML 模型在驱动安卓应用开发中,除了提供设计和实现上的指导性作用外的广泛用途,我们还探索了基于 IFML 模型对安卓应用进行自动化测试这一方法的可行性.通过上述实验为这 5 个应用进行了 IFML 建模之后,本文还通过 ADAMANT 依据建立的 IFML 模型对这些应用进行了测试.使用安卓模拟器作为我们的测试设备,在测试中将模拟器设备分别设置为 Nexus5X、Nexus S 以及 Pixel,将模拟器的安卓版本分别设置为 4.4、5.0 以及 5.1,进行了实验.最终,这些实验结果虽然存在一些轻微的差异,但均可正常进行测试,这表明,利用 ADAMANT,同一个 IFML 模型可广泛应用于不同的安卓设备以及安卓版本.

为了量化测试效果,本文还详细列出了设备设置为 Nexus5X,安卓版本设置为 5.1 的模拟器下 ADAMANT 的具体测试结果,见表 3.ADAMANT 运行在 8G 内存、i7 处理器的 Win 10 笔记本电脑之上,最终测试实验结果见表 3.同时,为了对比测试效果,我们将 ADAMANT 进行测试所花费的时间记录下来(表中“执行时间”栏),在相同的应用上使用安卓最主流、最广泛使用的随机事件流工具 Monkey 进行实验,并给定了相同的执行时间.

表 3 第 2 栏表明了该应用对应的 IFML 模型的事件数,由于在测试中是通过模型遍历,以事件覆盖为准则生成测试用例,因此,IFML 模型规模越大,事件数越多,ADAMANT 生成的测试用例也越多,执行时间也越长.随后,

其 3~5 栏表明使用的测试方法、该方法生成的测试用例数量以及这些用例中总共执行的测试操作或事件数量.第 6 栏~第 9 栏则表明了用例执行的结果——包含这些用例中执行失败的数量、这些用例执行所达到的代码行的覆盖率、执行的时间以及在该测试进行中发现的缺陷或异常.

Table 3 Testing result of test cases for these 5 apps

表 3 对 5 个应用的各项测试执行结果

应用	IFML 模型事件数	测试方法	执行用例数	测试事件	用例执行失败数	代码行覆盖率(%)	执行时间(s)	缺陷
OwnCloud	195	ADAMANT	81	648	10	56.48	2 500	1,2,3,4
		Monkey	–	16 000	–	43.85		–
ConnectBot	149	ADAMANT	71	483	6	54.39	2 000	–
		Monkey	–	13 000	–	30.03		–
Ringdroid	57	ADAMANT	28	210	0	81.50	600	–
		Monkey	–	4 000	–	63.60		–
Kiwix	108	ADAMANT	35	252	3	77.96	1 200	5
		Monkey	–	8 000	–	63.44		5
Omni-notes	167	ADAMANT	66	594	5	70.58	2 800	6
		Monkey	–	18 000	–	46.29		–

从静态代码覆盖率来看,ADAMANT 在这 5 个应用上均达到了 50%以上的代码覆盖率,且其均值达到了 68.18%,相较于 Monkey 的均值 49.44%,有将近 18.74%的代码覆盖率的提升.这一巨大的提升,主要体现在以下 3 个方面:(1) 通过 ADAMANT 所建立的 IFML 模型具有丰富的语义,且在经过扩展后可以以表达式的形式来表示各事件执行、元素显示、页面跳转之间的关系,因此,能够生成高度关联性的测试用例.而 Monkey 这类随机测试由于其随机性,将会出现大量的冗余以及无效事件,因此拉开了测试差距;(2) 由于该 IFML 模型是手工建立的,因此在该 IFML 模型中,使用者能为特定的输入控件提供候选输入内容,有效的文本输入大大提升了测试用例的效果;(3) 通过本文的扩展,IFML 模型可以建立种类丰富的事件,用户操作如长按、双击、滑动、缩捏等,系统事件如 Wifi 连接、数据连接等,而这一点在 Monkey 上有较大的欠缺,例如它只能支持简单的点击以及滑动这两种用户操作.

随后,在测试用例执行上,存在一定量的执行失败测试用例,我们将其展示在表 3 第 6 栏中.对这些失败的用例进行了人工检查,其中,大多数源自于应用本身的缺陷,或是应用本身的设计与建模者的认识相悖,其中便存在着应用实际的开发与初始的设计不一致的情形.此外,执行失败也与外部环境存在一定关联.例如,在 OwnCloud 中,由于网络波动或是服务器异常,部分操作可能会超时,最终导致测试用例执行失败.

最后,在缺陷发现上,ADAMANT 发现了 6 个缺陷,见表 4.而 Monkey 只发现了第 5 条.其中,第 1、2、4、5 条缺陷均引发了应用的崩溃;第 3 条和第 6 条缺陷未引发应用崩溃,但 ADAMANT 检测到了 OwnCloud 和 Omni-Notes 应用的 IFML 模型与实现的不一致性.我们通过对比 Github 上相关的缺陷 issue,发现这两个缺陷均由开发者确认为实现中的真实错误和疏漏.

Table 4 Details of defects found in these 5 apps

表 4 5 个应用中发现的缺陷信息

缺陷序号	对应应用	缺陷描述
1	OwnCloud	当进入 settings 的 Backup 界面时引起 NullPointerException 异常,触发崩溃
2	OwnCloud	在主界面或是上传文件页面进行重命名时引起 NullPointerException 异常,触发崩溃
3	OwnCloud	在进行 search 操作后执行返回操作时,文件列表依旧保持了 search 后的状态,与设计意图相悖
4	OwnCloud	在文件夹内层进行搜索时将会触发 ClassCastException 崩溃
5	Kiwix	应用中使用语音朗读功能后,关闭应用时 Pico TTS 服务(语音服务)会崩溃,错误信息为 Fatal signal 11 (SIGSEGV), code 1, fault addr 0x7f0dda291970 in tid 14926(com.svox.pico)
6	Omni-notes	在对某条笔记进行右画删除时,可以依次点击“Remove”“Password”,返回键来跳过密码验证,与开发者设计意图相悖

综合来看,ADAMANT 所实现的测试功能,可在较短时间内对应用进行有效的测试,达到较高的测试覆盖率,并发现应用中的缺陷,以及在应用中与原本设计存在偏差的部分.因此,鉴于 ADAMANT 良好的测试效果,当开发人员通过 ADAMANT 建立 IFML 模型作为应用的设计时,其测试功能可以帮助开发人员检验实现完成的

应用是否与原设计意图保持一致,以尽可能地减少实现中的偏差所带来的额外工作量.另外,ADAMANT 也可支持增量式的开发过程:在应用演进中,开发者可以逐步建立并不断丰富 IFML 模型,此时,ADAMANT 依然可以进行相应的测试,从而指导开发的进行,并且伴随着 IFML 模型的不断精化,其生成的测试用例也会更加精确、有效.这一过程中,出众的测试效果以及极低的测试成本能够抵消 IFML 的建模成本,最终极大地提高开发效率与质量.

6 相关工作

6.1 关于移动端应用设计开发建模的研究

近年来,考虑到目前的移动系统和设备的繁多,为了降低重复开发的成本,将模型应用于移动应用开发的研究大量涌现[11].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;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;也有专门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跨平台模型驱动开发的研究[5].近年来,也有不少将图形化领域特定语言(DSL)应用于移动应用建模的研究,Vaquero-Melchor 等人[13]对复数 DSL 进行扩展,用于对应用的各个方面的信息,如外部交互、移动性以及上下文环境等进行建模,并整合为一个 active DSL. Rieger 等人[14]提出了 Münster App Modeling Language(MAML),一种面向非技术人员的图形化 DSL,该语言以流程的方式描述了数据、视图、业务逻辑以及用户交互.但 MAML 仅以关键的字词代替了流程中视图的具体展示和事件交互,因此,在表现移动应用丰富 GUI 时表现能力不足.此外,Vaupel 等人[15,16]通过在模型上增加其抽象层次,从而提供了一种可详可略的建模方式,使得移动应用开发更加灵活多变.国内,杜一等人[17]提出了基于 E-UIDL 模型的移动应用开发方法,该模型着重于应用界面,并以领域模型、抽象用户界面模型等等诸多子模型来对界面进行描述,提高其可复用性及可扩展性,但很显然,这也引起了模型在可用性上的不足.对于本文所用到的 IFML 标准,也有类似的研究.Brambilla 等人[18]通过建立移动应用的 IFML 模型,来生成对应的 HTML5、CSS3 等网页代码,最终包装在移动端安卓与 iOS 平台的应用中,并在实际的开发案例中进行实验.然而,该研究仍然需要一定的人工参与,且将网页代码作为中转产物将会遗失移动平台独有的信息.而本文对 IFML 进行了安卓端扩展,使得 IFML 用于安卓应用建模时拥有较高的匹配度,虽未进行具体的代码自动化生成工作,但本文还对扩展后的 IFML 模型进行了形式化定义,这为代码自动化生成打下了基础.而在这一定义下,文本实现的基于该 IFML 模型的自动化测试方法,也能作为判断设计与实现一致性的手段,以驱动开发、演进工作的进行.

6.2 关于IFML建模相关的研究

现有研究工作中,IFML 被广泛应用于众多领域之中进行建模以及相应扩展的研究.例如,Ed-Douibi 等人[19]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 等人在文献[20]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 Communica- tion Platform(UCP)进行了简单的比较;Frajták 等人[21,22]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[23]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[24]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.这些基于 IFML 模型探索代码生成与测试执行自动化的研究,证明了将 IFML 应用于软件开发自动化的可行性和有效性.此外,Huang 等人[25]提供了从安卓应用中抽取 IFML 模型的方法,该方法也可与本文的工作相结合,以降低从已有开发项目中建立 IFML 模型的成本.

6.3 关于安卓终端应用自动化测试的研究

目前,针对安卓应用的自动化测试工具,在工业界,诸如 Robotium[26]、Appium[27]以及 MonkeyRunner[28]之类的自动化测试执行工具已经较为普及.近年来,学术界大量的研究着眼于利用模型来自动化生成测试用例,著名

的有,2012 年 Amalfitano 等人基于 GUIRipper[29]进行安卓端扩展的研究工作 AndroidRipper[30].它以动态执行的方式遍历安卓应用,并生成相应的 GUI 模型,最终实现测试的目的.从此之后,这类自动化构建模型生成测试输入的测试方法逐渐成为学术界安卓测试的主流之一:Azim 等人[31]通过综合使用静态分析和动态执行的方式,生成安卓应用对应的动态和静态活动转移图(activity transition graph),从而进行针对性的以及广度优先的应用系统化遍历测试;Choi 等人[32]结合机器学习技术来构建模型,并在学习中以页面跳转来实现状态变更,从而减少了应用的重启;2015 年,AndroidRipper 的后续版本 MobiGUITAR[33]诞生,它用一个状态机模型代替原来无状态的 GUI 模型,并且在模型遍历、更新算法上进行了优化;Su 等人[34]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加以执行,使用 Gibbs 抽样来分析执行结果,并以此提炼原随机模型,进一步生成更加有效的测试用例.近年来,不少基于程序分析、随机测试、组合测试等等的测试方法也引起了学术界的关注.Mirzaei 等人[35]提出了通过程序分析技术获取对输入的各种约束,从而削减测试输入组合的方法,Song 等人[36]提出了通过直接调用事件处理器的回调函数而非基于 GUI 事件的随机测试方法,Sadeghi 等人[37]则将安卓权限引入测试之中,并通过混合式的程序分析方法来减少不必要的权限组合测试.上述这些方法通常独立于应用的开发工作,而本文基于 IFML 模型进行测试,是作为将 IFML 应用于安卓应用设计建模,并指导应用开发工作的一环.本文的方法立足于 IFML 模型的这一前提,因为 IFML 模型的存在,其测试效果更佳,也能发现应用设计和实现不一致的地方,从而提高了将 IFML 模型引入安卓应用开发的功效.

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的对移动的需求,安卓应用的复杂度逐级攀升.而安卓设备和系统存在的多样化和碎片化,使得在安卓应用开发中增加了额外开发的工作量和难度.针对这一现象,本文根据安卓应用事件驱动的特性,将 IFML 模型引入安卓应用的开发过程,希望开发者能将应用的设计以 IFML 模型的形式进行具象化,降低开发难度,指导应用的开发工作.然而,目前宽泛的 IFML 并不适用于移动/安卓应用建模,为此,本文对 IFML 标准进行了面向安卓端的扩展,对扩展后的 IFML 模型进行了形式化定义,使得无论是基于 IFML 模型生成代码还是执行测试等等方法都成为可能.最后,本文实现了相应的建模与测试工具 ADAMANT,使用它为 5 个应用建立了 IFML 模型,并基于这些 IFML 模型进行了自动化测试.其良好的实验结果显示了本文对 IFML 进行的安卓端扩展、对 IFML 模型进行的形式化定义的有效性,以及将其应用于软件开发过程中的可行性.

当前,该方法处于初步建立的阶段,其应用于真实的安卓软件开发中的有效性还有待检测.因此,在未来的研究中,我们将在实际的应用开发案例中,将完全人工开发的方式以及基于该扩展后的 IFML 模型的开发方式,乃至于其他模型驱动开发方法进行对比实验,将应用初次开发所花费的成本以及随着软件演进时的持续成本、产品质量等作为衡量标准来评判这一方法在实际开发中的有效性.

在自动化方面,本文目前仅将 IFML 模型应用于测试的生成与执行.在未来工作中,我们将研究通过 IFML 模型自动化生成应用代码的方法.由于 IFML 模型能够精确地描述应用的 GUI 结构以及其中的工作流传递,我们初步计划为通过 IFML 模型,自动化地生成应用的主体骨架,包含:应用中主要的 Activity(包含应用的 GUI 界面以及界面上的控件元素)、控件元素上的各类事件监听器以及事件监听器中各 GUI 界面相互跳转的逻辑代码.这些代码可来自于安卓应用与对应扩展后的 IFML 模型的资源库,该资源库可通过相关工作中介绍的 IFML 模型抽取技术,对当前开源安卓应用进行抽取获得.随后,我们可基于该资源库,通过结合机器学习和模型驱动开发技术,来实现这些代码的自动化生成,或是通过寻找相似 IFML 模型片段直接获得实现代码.随后,开发人员可对未生成的应用业务逻辑进行补充,形成一个半自动化的开发方式,降低开发中的人工成本,提高效率.

References:

[1] Gartner Says Worldwide Sales of Smartphones Returned to Growth in First Quarter of 2018. 2018. https://www.gartner.com/newsroom/id/3876865

[2] Android.wiki. 2018. https://en.wikipedia.org/wiki/Android_(operating_system)

[3] Chandra R, Karlsson BF, Lane N, Liang C, Nath S, Padhye J, Ravindranath L, Zhao F. Towards scalable automated mobile app testing. Technical Report, MSR-TR-2014-44, Microsoft Research, 2014.

[4] Parada AG, de Brisolara LB. A model driven approach for Android applications development. In: Proc. of the Computing System Engineering (SBESC). IEEE, 2012. 192–197.

[5] Heitkötter H, Majchrzak TA, Kuchen H. Cross-platform model-driven development of mobile applications with md^2. In: Proc. of the 28th Annual ACM Symp. on Applied Computing. Coimbra: ACM, 2013. 526–533.

[6] Yang W, Prasad MR, Xie T. A grey-box approach for automated GUI-model generation of mobile applications. In: Hybrid Systems: Computation and Control. Berlin, Heidelberg: Springer-Verlag, 2013. 250–265.

[7] IFML: The interaction flow modeling language. 2013. http://www.ifml.org/

[8] Sirius: The easiest way to get your own modeling tool. 2018. https://www.eclipse.org/sirius/

[9] List of free and open-source Android apps. 2018. https://en.wikipedia.org/wiki/List_of_free_and_open-source_Android_applications

[10] Github.pcqpcq: Open-source-Android-apps. 2016. https://github.com/pcqpcq/open-source-android-apps

[11] Umuhoza E, Brambilla M. Model driven development approaches for mobile applications: A survey. In: Proc. of the Int'l Conf. on Mobile Web and Information Systems. Vienna: Springer-Verlag, 2016. 93–107.

[12] Balagtas-Fernandez F, Tafelmayer M, Hussmann H. Mobia modeler: Easing the creation process of mobile applications for non-technical users. In: Proc. of the 15th Int'l Conf. on Intelligent User Interfaces. Hong Kong: ACM, 2010. 269–272.

[13] Vaquero-Melchor D, Palomares J, Guerra E, de Lara J. Active domain-specific languages: Making every mobile user a modeller. In: Proc. of the 20th Int'l Conf. on Model Driven Engineering Languages and Systems (MODELS). Austin: ACM, 2017. 75–82.

[14] Rieger C. Business apps with MAML: A model-driven approach to process-oriented mobile app development. In: Proc. of the Symp. on Applied Computing. Marrakech: ACM, 2017. 1599–1606.

[15] Vaupel S, Taentzer G, Harries JP, *et al*. Model-driven development of mobile applications allowing role-driven variants. In: Proc. of the Int'l Conf. on Model Driven Engineering Languages and Systems. Valencia: Springer-Verlag, 2014. 1–17.

[16] Vaupel S, Taentzer G, Gerlach R, Guckert M. Model-driven development of mobile applications for Android and iOS supporting role-based app variability. Software & Systems Modeling, 2018,17(1):35–63.

[17] Du Y, Tian F, Dai GZ. Development approach based on extensible user interface description language. Ruan Jian Xue Bao/Journal of Software, 2015,26(7):1772−1784 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4584.htm [doi: 10.13328/j.cnki.jos.004584]

[18] Brambilla M, Mauri A, Umuhoza E. Extending the interaction flow modeling language (IFML) for model driven development of mobile applications front end. In: Proc. of the Int'l Conf. on Mobile Web and Information System. Springer-Verlag, 2014. 176–191.

[19] Ed-Douibi H, Izquierdo JLC, Gómez A, Tisi M, Cabot J. EMF-REST: Generation of RESTful APIs from models. In: Proc. of the 31st Annual ACM Symp. on Applied Computing. Pisa: ACM, 2016. 1446–1453.

[20] Raneburger D, Meixner G, Brambilla M. Platform-independence in model-driven development of graphical user interfaces for multiple devices. In: Proc. of the Int'l Conf. on Software Technologies. Reykjavic: Springer-Verlag, 2013. 180–195.

[21] Frajták K, Bureš M, Jelínek I. Transformation of IFML schemas to automated tests. In: Proc. of the 2015 Conf. on Research in Adaptive and Convergent Systems. Prague: ACM, 2015. 509–511.

[22] Frajták K, Bureš M, Jelınek I. Using the interaction flow modelling language for generation of automated front–end tests. In: Proc. of the Annals of Computer Science and Information Systems. 2015,6:117–122.

[23] Brajnik G, Harper S. Detaching control from data models in model-based generation of user interfaces. In: Proc. of the Int'l Conf. on Web Engineering. Rotterdam: Springer-Verlag, 2015. 697–700.

[24] Laaz N, Mbarki S. Integrating IFML models and owl ontologies to derive UIs Web-Apps. In: Proc. of the 2016 Int'l Conf. on Information Technology for Organizations Development (IT4OD). Fez: IEEE, 2016. 1–6.

[25] Huang A, Pan M, Zhang T, Li X. Static extraction of IFML models for Android apps. In: Proc. of the 21st ACM/IEEE Int'l Conf. on Model Driven Engineering Languages and Systems. Copenhagen: ACM, 2018. 53–54.

[26] Robotium. User scenario testing for Android. 2016. http://code.google.com/p/robotium/

[27] Appium automation for apps. 2018. http://appium.io/

[28] Developers A. Monkeyrunner. 2018. http://developer.android.com/studio/test/monkeyrunner

[29] Memon AM, Banerjee I, Nagarajan A. GUI ripping: Reverse engineering of graphical user interfaces for testing. In: Proc. of the WCRE. Victoria: IEEE, 2003,3:260.

[30] Amalfitano D, Fasolino AR, Tramontana P, *et al*. Using GUI ripping for automated testing of Android applications. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. Essen: ACM, 2012. 258–261.

[31] Azim T, Neamtiu I. Targeted and depth-first ex-ploration for systematic testing of Android apps. In: Proc. of the 2013 ACM SIGPLAN Int'l Conf. on Object Oriented Programming Systems Languages & Applications. Indianapolis: ACM, 2013. 641–660.

[32] Choi W, Necula G, Sen K. Guided GUI testing of android apps with minimal restart and approximate learning. ACM SIGPLAN Notices, 2013,48(10):623–640.

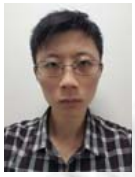
[33] Amalfitano D, Fasolino AR, Tramontana P, *et al*. MobiGUITAR: Automated model-based testing of mobile apps. IEEE Software, 2015,32(5):53–59.

[34] Su T, Meng G, Chen Y, *et al*. Guided, stochastic model-based GUI testing of Android apps. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 245–256.

[35] Mirzaei N, Garcia J, Bagheri H, Sadeghi A, Malek S. Reducing combinatorics in GUI testing of Android applications. In: Proc. of the 38th IEEE/ACM Int'l Conf. Software Engineering (ICSE). Austin: IEEE, 2016. 559–570.

[36] Song W, Qian X, Huang J. Ehbdroid: Beyond GUI testing for Android applications. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering. Urbana-Champaign: IEEE Press, 2017. 27–37.

[37] Sadeghi A, Jabbarvand R, Malek S. Patdroid: Permission-aware GUI testing of Android. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 220–232.



陆一飞(1994－),男,江苏苏州人,博士,CCF 学生会员,主要研究领域为软件测试,程序分析,交互式模型验证.

潘敏学(1983－),男,博士,助理研究员,CCF 专业会员,主要研究领域为高可信软件的验证与分析,形式化方法.

张天(1978－),男,博士,副教授,CCF 高级会员,主要研究领域为模型驱动工程.

王林章(1973－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为模型驱动的软件测试与验证,安全测试,软件测试自动化.

李宣东(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为软件建模与分析,软件测试与验证.