УДК 004.514

ОБЗОР СУЩЕСТВУЮЩИХ ПОДХОДОВ К МОДЕЛИРОВАНИЮ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ANDROID

Сарычева Ю.Ю., Белов Ю.С.

ФГБОУ ВО «Московский государственный технический университет имени Н.Э. Баумана», Калужский филиал, Калуга, e-mail: yulia.sarycheva99@mail.ru

Графический интерфейс пользователя является неотъемлемой частью программного обеспечения на большинстве основных платформ, включая ОС Android, и представляет широкий интерес для многих исследователей. Конечные пользователи взаимодействуют с приложением в зависимости от того, что они хотят делать с приложением и что они видят в его графическом интерфейсе. Поскольку разные приложения часто используют общие шаблоны проектирования пользовательского интерфейса, интуитивно понятно, что способ взаимодействия людей с графическим интерфейсом можно обобщить для разных приложений. Цель модели взаимодействия состоит в том, чтобы зафиксировать такие обобщенные модели взаимодействия. Приложения для Android состоят из четырех основных типов компонентов: действий, сервисов, приемников вещания и поставщиков контента. В данном исследовании будут рассмотрены основные сущности графического интерфейса пользователя в приложениях на базе ОС Android, а также существующие подходы к моделированию пользовательского интерфейса для приложений на базе ОС Android. К существующим подходам относятся: Activity Transition Graph, Window Transition Graph, Screen Transition Graph. Модель SVG состоит из трех основных компонентов: экстрактора топологии, построителя экранов и межкомпонентного анализатора.

Ключевые слова: графический интерфейс пользователя, тестирование, модель генерации, автоматизированное тестирование

REVIEW OF EXISTING APPROACHES TO MODELING THE ANDROID USER INTERFACE

Sarycheva Yu.Yu., Belov Yu.S.

Bauman Moscow State Technical University, Kaluga branch, Kaluga, e-mail: yulia.sarycheva99@mail.ru

The graphical user interface (GUI) is an integral part of software on most major platforms, including the Android OS, and is of great interest to many researchers. End users interact with the application based on what they want to do with the application and what they see in its GUI. Because different applications often use common user interface design patterns, it is intuitive that the way people interact with a GUI can be generalized across applications. The purpose of an interaction model is to capture such generalized interaction models. Android apps are made up of four main types of components: activities, services, broadcast receivers, and content providers. This study will consider the main entities of the graphical user interface in applications based on the Android OS, as well as existing approaches to modeling the user interface for applications based on the Android OS. Existing approaches include: Activity Transition Graph, Window Transition Graph, Screen Transition Graph. An SVG model consists of 3 main components: a topology extractor, a screen builder, and a cross-component analyzer.

Keywords: graphical user interface, testing, generation model, automated testing

Анализ графического интерфейса приложения представляет большой интерес для многих исследователей и практиков, на данный момент существуют два направления исследований в этой области. Один из них – понять поведение приложений с точки зрения разработки программного обеспечения, другой – с точки зрения взаимодействия человека с компьютером для анализа дизайна пользовательского интерфейса [1].

Цель – исследовать методы, применяющиеся к моделированию GUI для мобильных приложений на базе ОС Android.

Структура Android-приложения

Android – это мобильная операционная система с открытым исходным кодом, в первую очередь предназначенная для мобильных устройств с сенсорными экранами. Впервые она была показана в 2007 г. ее разработчиками Google и Open Handset Alliance [2]. Самой последней стабильной версией Android является Android 13.

Приложения для Android состоят из четырех основных типов компонентов [3]: действий, сервисов, приемников вещания и поставщиков контента.

Действия – это основные строительные блоки пользовательского интерфейса, которые облегчают взаимодействие между пользователем и приложением.

Активность – это точка входа для взаимодействия с пользователем [4]. Эти действия запускаются, когда приложение использует другое приложение, а также являются точкой взаимодействия с пользователем приложения. Приложения обычно состоят из нескольких таких действий, которые вместе образуют целое приложение, но не связаны друг с другом сильно.

Сервисы выполняют задачи, не требующие пользовательского интерфейса, например предварительную загрузку файлов для более быстрого доступа или воспроизведение музыки в фоновом режиме после того, как пользователь переключился на другие действия.

Широковещательные приемники – это компоненты, которые отвечают на уведомления. Широковещательная рассылка может исходить от:

1) системы, например, для объявления о низком заряде батареи;

2) другого приложения, например, для уведомления о завершении загрузки файла;

3) от других компонентов приложения, например, для уведомления об успешном входе в систему.

Контент-провайдеры управляют доступом к данным, например, для хранения и извлечения контактов.

Существующие подходы

На данный момент существуют следующие подходы к моделированию пользовательского интерфейса Android-приложений:

− ATG – Activity Transition Graph;

− WTG – Window Transition Graph;

− STG – Screen Transition Graph.

График перехода активности (ATG) фиксирует действия приложения и переходы между этими действиями. Он не моделирует другие компоненты приложения и не моделирует действия, которые запускают переходы, т.е. какой пользовательский ввод запускает переход между действиями.

Оконный график перехода (WTG) расширяет ATG, устраняя некоторые из этих ограничений: он моделирует обработчик событий для каждого перехода, а также включает меню и диалоги как отдельные узлы модели. Однако он не рассматривает фрагменты, ящики, службы и широковещательные приемники.

Рассмотрим пример авторизации через Facebook (запрещенная в РФ социальная сеть; принадлежит корпорации Meta, которая признана в РФ экстремистской и запрещена) в тестовом приложении. Для желаемого результата пользователю необходимо совершить действия, представленные на рис. 1.

Для данного приложения и ATG, и WTG будут включать только три узла, соответствующие действиям приложения: «Контроллер», который соответствует действию на рис. 1, a–c и f, «Аутентификатор» для действия на рис. 1, d, и Facebook (запрещенная в РФ социальная сеть; принадлежит корпорации Meta, которая признана в РФ экстремистской и запрещена) для действия на рис. 1, e, которое реализовано внутри Facebook SDK (запрещенная в РФ социальная сеть; принадлежит корпорации Meta, которая признана в РФ экстремистской и запрещена). Модели ATG и WTG не будут включать переходы, так как все переходы в этом приложении происходят из фрагментов. Таким образом, ни одна из этих моделей не будет содержать путь, ведущий к активности Facebook (запрещенная в РФ социальная сеть; принадлежит корпорации Meta, которая признана в РФ экстремистской и запрещена).

Кроме того, в моделях не учитываются широковещательные приемники, и, следовательно, никакие действия, следующие за входом в Facebook (запрещенная в РФ социальная сеть; принадлежит корпорации Meta, которая признана в РФ экстремистской и запрещена), например, на рис. 1, f, недоступны.

Еще более серьезным недостатком существующих моделей пользовательского интерфейса является то, что они рассматривают компоненты пользовательского интерфейса, такие как действия и меню, как отдельные узлы. Эти модели неточно представляют состав компонентов пользовательского интерфейса на экранах, что препятствует их применимости для сценария исследования, ориентированного на цель: поскольку представления не включают достаточно деталей для извлечения исполняемого пути.

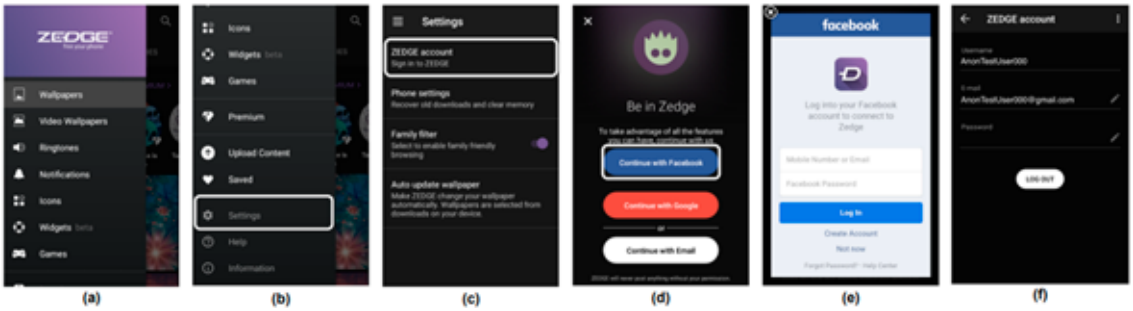

Рис. 1. Пример авторизации

Рис. 2. Экстрактор STG

График перехода экрана (SVG) моделирует экраны приложений, представляя состав активности контейнера, размещенных фрагментов, меню и навигационных диалогов на каждом экране.

В STG основные узлы $\mathbb{V}s$ представляют экраны приложений, а вспомогательные узлы $\mathbb{V}r$ и $\mathbb{V}b$ представляют службы и приемники вещания соответственно. Каждый экранный узел в $Vs \in \mathbb{V}s$ состоит из одного контейнера действий VsA, нуля или более фрагментов VsF, нуля или одного меню/ящика VsM и нуля или одного диалога VsD. Коллекция всех узлов экрана соответствует возможным экранам, разрешенным в приложении. Действие с n фрагментами и одним ящиком может соответствовать $2 \times 2n$ различных экранов (все комбинации фрагментов, с ящиком и без него).

Вспомогательные узлы необходимы, так как переходы между экранами могут проходить через эти элементы. Ребра E в STG соответствуют переходам между узлами $\mathbb{V} = \mathbb{V}s \cup \mathbb{V}r \cup \mathbb{V}b$.

Каждое ребро $e \in E$ имеет метку $e\tau$, которая представляет событие, запускающее переход: либо действие пользователя, например нажатие кнопки «Учетная запись» для перехода от элемента № 3 к элементу № 5, либо намерение, которое запускает широковещательный приемник. Переходы, которые запускаются автоматически, например, когда сервис открывает диалог пользовательского интерфейса без каких-либо уведомлений или кликов, не имеют в модели триггеров событий: $e\tau = \emptyset$.

Экстрактор STG состоит из трех основных компонентов, показанных на рис. 2: экстрактора топологии, построителя экранов и межкомпонентного анализатора.

Экстрактор топологии

Средство извлечения топологии идентифицирует основные компоненты приложения, т.е. действия, службы и приемники широковещательной рассылки, а также графики их вызовов. Он опирается на реализацию FlowDroid [5] для извлечения компонентов как из исходного кода приложения, так и из XML-файлов и связывания каждого компонента с его точками входа, то есть событиями жизненного цикла и методами обратного вызова для конкретного приложения. Он также использует FlowDroid для построения графиков вызовов всех точек входа компонента. Например, активность может регистрировать обратные вызовы, которые вызываются при нажатии кнопки или получении обновления местоположения; затем соответствующий обработчик обратного вызова будет связан с графом вызовов действия и проанализирован между событиями жизненного цикла onResume и onPause действия. Методы приложения, аннотированные «@JavascriptInterface», также могут запускаться веб-представлениями с поддержкой JavaScript [6]. Поскольку вызов этих методов зависит от содержимого WebView, доставляемого во время выполнения, экстрактор топологии консервативно расширяет графы вызовов, предполагая, что любой метод с поддержкой JavaScript в компоненте может быть вызван любым WebView с поддержкой JavaScript в том же компоненте. После извлечения компонентов и их графов вызовов узлы, соответствующие службам и широковещательным приемникам, добавляются непосредственно в STG в $\mathbb{V}r$ и $\mathbb{V}b$ соответственно. Далее строятся экраны приложений в $\mathbb{V}s$.

Конструктор экрана

Ключевым моментом построения модели является моделирование экранов пользовательского интерфейса и переходов между экранами. С этой целью конструктор экрана анализирует граф вызовов каждого действия, собирая размещенные фрагменты, меню, диалоговые окна и панели навигации. Фрагменты могут быть определены статически в файле макета активности или динамически в коде. Конструктор экрана начинает с извлечения файлов макета с помощью AXMLPrinter [7], а затем идентифицирует объявляемые ими фрагменты. Чтобы собрать объявления фрагментов в коде, конструктор экрана просматривает граф вызовов каждого действия, определяя методы транзакций фрагментов, которые добавляют, удаляют или заменяют фрагменты. Графики активности вызовов сканируются в несколько итераций. Во-первых, конструктор экрана анализирует

события жизненного цикла действия, инициированные при инициализации действия: от onCreate до onResume. Для каждого события он идентифицирует методы транзакций фрагментов m, которые добавляют, удаляют или заменяют фрагменты; затем он выполняет межпроцедурный, контекстно-зависимый и потокозависимый анализ пути вызова m, чтобы идентифицировать тип Java фрагмента (фрагментов), обрабатываемых в m. Как только все фрагменты событий жизненного цикла от onCreate до onResume действия A обработаны, Screen Builder создает «базовый» экранный узел Vs в $\mathbb{V}$s для действия A, причем VsF содержит (возможно, пустой) список идентифицированных фрагментов. Этот «базовый» экранный узел отображается при запуске действия. Методы жизненного цикла, выдаваемые при приостановке, остановке или возобновлении действия, могут дополнительно добавлять, удалять или заменять фрагменты на «базовом» экране. Таким образом, конструктор экрана анализирует возможные цепочки вызовов методов жизненного цикла: «onPause → onResume» и «onPause → onStop → onRestart → onResume», чтобы извлечь фрагменты из каждой из этих цепочек. Если фрагменты в цепочке отличаются от фрагментов в «базовом» экранном узле Vs, добавляется новый экранный узел Vs' для активности A, которая содержит объединение фрагментов в Vs и идентифицированных в цепи. Он также добавляет переход между Vs и Vs' с пустой меткой, так как этот переход запускается системой автоматически. Точно так же обрабатываются фрагменты в методах обратного вызова. Поскольку порядок обратных вызовов нельзя предсказать, конструктор экрана предполагает, что обратные вызовы могут выполняться в любом порядке; таким образом, он создает новый экран Vs'', если он еще не существует, для любого возможного порядка методов обратного вызова, которые изменяют фрагменты «базового» экрана. Конструктор экрана также создает границы перехода между этими экранами и устанавливает метку перехода e_τ в качестве действия, запускающего соответствующий обратный вызов. Наконец, конструктор экрана анализирует каждый экран Vs, чтобы идентифицировать его меню и диалоги. С этой целью он анализирует графы вызовов активности экрана VsA и всех его собранных фрагментов VsF, чтобы определить методы, инициализирующие меню, ящики и диалоги. Когда такой метод найден, конструктор экрана копирует экран Vs для действия A в V$\hat{s}$, добавляет найденное меню, ящик или диалоговое окно к V$\hat{s}$, добавляет V$\hat{s}$ к набору всех экранных узлов Vs и создает границу перехода между Vs и V$\hat{s}$. Метка перехода e_τ устанавливается как действие, запускающее метод обратного вызова.

Межкомпонентный анализатор

На предыдущем шаге собраны все узлы экрана и переходы между узлами экрана одной активности, но с разными фрагментами, меню, ящиками и диалогами. Межкомпонентный анализатор собирает межкомпонентные переходы между узлами, соответствующими различным компонентам Android, например экранам различных действий, сервисов и приемников вещания. Эти переходы выполняются с помощью методов межкомпонентной связи (ICC), и мы полагаемся на интеграцию FlowDroid с IccTA [7] для определения связей ICC между компонентами приложения. Для каждого канала, где источником и целью являются службы и/или широковещательные приемники, Межкомпонентный анализатор просто создает соответствующий фронт перехода e в STG. Если целью перехода является широковещательный приемник, метка перехода e_τ устанавливается равной широковещательной рассылке, которая инициирует событие, указанное в фильтре намерений приемника широковещательной рассылки. Если целью перехода является служба, $e_\tau = \emptyset$, так как этот переход запускается «автоматически», без какого-либо пользовательского или системного события. Действия представлены несколькими экранами и, таким образом, требуют более тонкого подхода. Если источником связи ICC является действие A, межкомпонентный анализатор идентифицирует точку входа действия p, из которой исходит сообщение. Затем он находит все узлы экрана в STG, соответствующие A, до которых можно добраться из «базового» узла экрана A через переходы, связанные с действием, запускающим p. Он добавляет переход от всех этих узлов к узлам, представляющим цели, помечая их действием, которое запускает p. Когда целью также является действие, межкомпонентный анализатор находит только «базовый» экранный узел этого действия и создает переход к этому узлу. Это связано с тем, что при запуске нового действия оно запускается на начальном экране; переходы между различными экранами целевого действия уже обрабатываются конструктором экрана.

Некоторые действия становятся доступными только после взаимодействия с экраном. Например, только после добавления товара в корзину будет активна кнопка «Оформить заказ». Поскольку STG

не моделирует правильный порядок пользовательских событий (например, щелчков, прокрутки и т.д.) на экране, средство выбора действий динамически взаимодействует с экраном, пытаясь изменить его состояние и активировать требуемое действие. Как и другие инструменты динамического исследования [8], он использует взвешенную стратегию исследования пользовательского интерфейса, которая выбирает следующее событие для конкретного виджета на основе типа события, прошлой частоты выполнения и количества новых виджетов, развертываемых событием.

Если Vs' по-прежнему не может быть достигнуто после определенного количества попыток (в настоящее время установлено 50 итераций), средство выбора действий возвращается и переходит к следующему пути. Если ни на одном из путей, ведущих к текущей цели, нет доступных действий или если еще нет набора «рабочих» целей, средство выбора действий выполняет поиск в ширину, чтобы найти следующую доступную цель, и повторяет поиск этой цели. Он возвращает «unreachable», если не найдено ни одного действия, ведущего к какой-либо из целей.

Заключение

Графический интерфейс является неотъемлемой частью программного обеспечения на большинстве основных платформ, включая Android, и представляет широкий интерес для многих исследователей. Были исследованы методы, применяющиеся к моделированию GUI для мобильных приложений на базе ОС Android: ATG, WTG, STG.

Серьезным недостатком ATG и WTG является то, что они рассматривают компоненты пользовательского интерфейса, такие как действия и меню, как отдельные узлы. Эти модели неточно представляют состав компонентов пользовательского интерфейса на экранах, что препятствует их применимости для сценария исследования, ориентированного на цель: поскольку представления не включают достаточно деталей для извлечения исполняемого пути. График перехода экрана (SVG) моделирует экраны приложений, представляя состав активности контейнера, размещенных фрагментов, меню и навигационных диалогов на каждом экране, что выгодно отличает его от других методов.

Список литературы

1. Винокуров А.В., Лавлинская О.Ю. Уровни организации автоматизированного тестирования мобильных приложений для операционной системы Android // Вестник Воронежского института высоких технологий. 2020. № 3 (34). С. 22–26.

2. Naja F., Mansur S., Wibawanto A. Automated Software Testing on Mobile Applications: A Review with Special Focus on Android Platform // 20th International Conference on Advances in ICT for Emerging Regions. 2020. P. 4–6.

3. Михалевская К.А., Сергачева М.А. Сравнение инструментов для автоматизации тестирования мобильных приложений на ОС Android // Chronos: естественные и технические науки. 2020. № 2 (30). С. 45–49.

4. Воробьев Н.А., Бурмин Л.Н., Степанов Ю.А. Сравнительный анализ средств тестирования мобильных приложений // Евразийский союз ученых. 2020. № 6–1 (75). С. 36–38.

5. King T.M., Arbon J., Santiago D. AI for Testing Today and Tomorrow: Industry Perspectives. IEEE International Conference On Artificial Intelligence Testing (AITest). 2019. P. 81–88.

6. Сарычева Ю.Ю., Белов Ю.С. Применение искусственного интеллекта в автоматизированном тестировании GUI // Научные исследования в современном мире. Теория и практика: сборник избранных статей Всероссийской (национальной) научно-практической конференции. 2022. С. 55–56.

7. Pan M., Xu To., Pei Yu. GUI-Guided Test Script Repair for Mobile Apps. IEEE Transactions on Software Engineering. 2022. Vol. 48. No. 3. P. 3–5.

8. Плодухин Д.М. Реализация модели автоматизированного тестирования // Огарёв-Online. 2020. № 13 (150).