СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

УДК 004.42

ИНТЕРПРЕТИРУЕМЫЕ МОБИЛЬНЫЕ ПРИЛОЖЕНИЯ

О. Ю. Рязанов, Ю. Д. Рязанов

Белгородский государственный технологический университет им. В. Г. Шухова

Поступила в редакцию 11.07.2018 г.

Аннотация. В статье проведен анализ существующих способов разработки мобильных приложений и предложен новый подход к их разработке. Подход заключается в создании на сервере описания компонентов мобильного приложения в формате XML, которое передается на мобильное устройство и интерпретируется в нем нативным интерпретатором. Это позволяет снизить стоимость разработки и сделать более удобными поддержку и использование мобильных приложений.

Ключевые слова: мобильное устройство, мобильное приложение, веб-приложение, интерпретатор.

Annotation. In this article analyzes existing methods of mobile application development and proposed new approach to their development. Approach is to create descriptions of mobile application components on the server with XML format that is sent to mobile device and interpreted by native interpreter. This can help reduce cost of development and increase support and usability of mobile applications.

Keywords: mobile device, mobile application, web-application, interpreter.

ВВЕДЕНИЕ

Мобильные приложения в настоящее время являются основным источником потребления контента. Форм-фактор устройств, на которых они запускаются, позволяют пользователям получать информацию, управлять услугами, оплачивать услуги и товары практически в любом месте.

Основными видами мобильных приложений являются нативные, веб-приложения и гибридные приложения. Эти виды приложений определяют подходы к разработке программного обеспечения для мобильных устройств.

Самым распространенным подходом к разработке мобильных приложений является разработка нативных приложений для мобильных операционных систем [1]. Помимо разработки нативных приложений, программисты занимаются адаптацией веб-приложений к использованию их на мобильных устройствах [2]. Новым, но динамично развивающимся подходом к разработке мобильного программного обеспечения является гибридный подход, представляющий собой разработку нативного приложения при помощи веб-технологий [3]. Все эти виды и подходы имеют свои плюсы и минусы.

В качестве компромиссного решения, которое может наиболее полно содержать в себе плюсы существующих видов и подходов к разработке мобильных приложений, предлагается альтернативный подход к разработке программного обеспечения для мобильных устройств. Подход заключается в создании на сервере описания компонентов мобильного приложения в формате XML, которое передается на мобильное устройство и интерпретируется в нем нативным интерпретатором. Такие приложения будем называть интерпретируемыми мобильными приложениями.

В статье выполнен анализ существующих видов мобильных приложений, описан подход к разработке интерпретируемых мобильных приложений, отмечены преимущества данного подхода перед существующими, описан способ реализации нативного интерпретатора описания компонентов мобильного приложения на примере операционной системы Android.

АНАЛИЗ ВИДОВ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Нативные мобильные приложения имеют свои достоинства и недостатки. Первым недостатком мобильных приложений является их установка. Мобильное приложение нужно найти, скачать и установить на устройство. При всех попытках разработчиков мобильных операционных систем упростить этот процесс, затраты времени от момента, когда пользователь осознает, что приложение ему нужно, до момента первого запуска остается велико. Можно привести ряд примеров, при котором не моментальный запуск приложения приведет к неудобствам у пользователя. Например, оплата времени за использование парковочного пространства. Человек, который впервые пользуется информационной системой для оплаты стоянки не знает, что терминал оплаты принимает только безналичные платежи. Однако он видит вывеску на терминале, что существует мобильное приложение. Пользователь ищет в магазине приложений приложение парковочного пространства, находит множество приложений с одинаковой иконкой и подписью, которое сокращено до «Парковочное пространство города…». Только поиск приложения для нужного города и скачивание занимает несколько минут. Но на этом затраты времени не кончаются, ведь место в памяти устройства может не хватить для установки приложения. Это является вторым недостатком мобильных приложений.

Мобильные приложения в сфере потребления услуг являются однотипными и выполняют одни и те же задачи. Они содержат в себе личный кабинет клиента, уведомления о новостях, списки товаров и услуг, их бронирование или оплату. Но каждый поставщик товаров или услуг имеет свое мобильное приложение. Когда пользователь пользуется услугами разных поставщиков, покупает товары в разных магазинах, он устанавливает на устройство именно однотипные приложения магазинов и сервисов.

Следует обратить внимание на стоимость разработки мобильных приложений. Трудоемкость разработки одного мобильного приложения составляет примерно 250 человеко-часов [4]. Но необходимо учитывать, что разработав одно приложение, не удастся удовлетворить всех пользователей. На текущий момент на рынке мобильных операционных систем доминируют две – Android от Google и iOS от Apple. Отсюда следует, что нужно разработать минимум два нативных приложения для охвата большей части пользователей. А если в приложении предполагается возможность взаимодействия лично знакомых людей, то разработки одного приложения точно будет недостаточно, так как пользоваться таким приложением смогут только те люди, в кругу знакомых которых распространены смартфоны на одной и той же операционной системе, что является крайне редкой ситуацией.

В какой-то мере описанные проблемы решаются при помощи веб-приложений [2]. Веб-приложения не требуют установки, следовательно, время с момента надобности использования приложения до первого запуска значительно уменьшается. Стоимость разработки мобильных веб-приложений ниже, чем у нативных. Это обусловлено тем, что веб-приложения кроссплатформенные и нет необходимости писать разный код для разных операционных систем. Но веб-приложения более требовательны к аппаратным характеристикам устройства. На устройствах среднего и бюджетного класса плавность анимации и скорость отклика низкие, а иногда и некомфортные для использования. Поддержка жестов, платежей, камеры, датчиков положения устройства не могут быть использованы в веб-приложениях.

Кроме разработки нативных и веб-приложений, существует гибридный подход к разработке мобильных приложений [3]. Суть этого подхода заключается в том, что разрабатывается нативное мобильное приложения при помощи веб-фреймворков. Гибридные приложения имеют возможность использовать все функции смартфона, но так же как и нативные, требуют установки. Однако большая часть кода приложения является одинаковой для разных платформ. Это является основным преимуществом гибридных приложений. Отсутствие необходимости писать специфический код для разных платформ существенно снижает затраты на разработку программного обеспечения. Гибридные приложения работают быстрее и более плавно, чем веб-приложения, но все равно уступают по этим показателям нативным мобильным приложениям.

ИНТЕРПРЕТИРУЕМЫЕ МОБИЛЬНЫЕ ПРИЛОЖЕНИЯ

На рис. 1 показана схема разработки и структура интерпретируемого мобильного приложения.

Разработчик интерпретируемого приложения создает описание компонентов приложения в виде XML-файлов и загружает их на сервер. На сервере осуществляется компоновка описания компонентов в один документ в формате JSON, который будет загружен приложением-интерпретатором. Загруженный приложением-интерпретатором документ обрабатывается парсером. На выходе парсера – объекты, модели компонентов приложения. Создаются только те объекты, которые необходимы для интерпретации текущего экрана приложения. Остальные данные сохраняются в памяти устройства для отложенного парсинга. На основе моделей компонентов отображается интерфейс интерпретируемого приложения. Компоненты этого приложения представляют собой иерархию, состоящую из экранов, блоков и модулей.

Проанализировав ряд мобильных приложений [5,6] сетей общественного питания, сетей магазинов и салонов красоты, был сделан вывод о том, что интерпретатор должен уметь интерпретировать небольшое количество шаблонов экранов, таких как, шаблон с контентом на весь экран, с вкладками, с панелью навигации. Контент на экране можно разбить на блоки, такие как кнопки, текст, кнопки с текстом, поля ввода, контейнеры с прокруткой и фиксированным числом блоков внутри, блоки выбора изображений, выбор даты, времени, расписание, блок выбора одного варианта из двух и другие.

Описать структуру приложения, состоящего из блоков и экранов, можно при помощи XML-разметки [7]. Разметка одного экрана приложения будет представлять собой тип

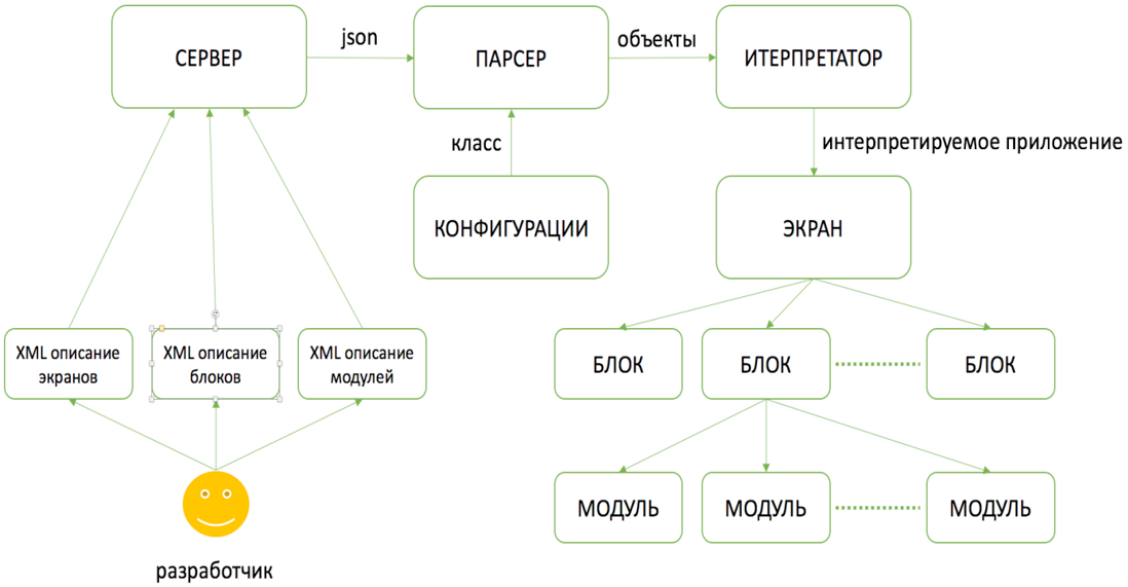

Рис. 1. Интерпретируемое мобильное приложение

шаблона экрана и его стиль: цвет фона или фоновое изображение. Если шаблон экрана содержит панель навигации, то в разметке будет указан стиль панели навигации и объекты внутри панели: цвет, текст заголовка и функциональные клавиши. Разметка должна содержать блоки, которые находятся на этом экране. У каждого блока есть стиль, стиль компонентов внутри блока (текста, изображений, кнопок и т.п.) и значения компонентов (строки, ссылки на изображения, определение клика кнопки и т.п.)

ПОДХОД К РАЗРАБОТКЕ ПРИЛОЖЕНИЯ-ИНТЕРПРЕТАТОРА

Процесс интерпретации приложения, описанного при помощи XML-разметки, можно описать следующим алгоритмом:

1. Загружается с сервера преобразованное в JSON-документ XML-описание начального экрана.
2. Выполняется чтение шаблона экрана и его стилей из JSON-разметки.
3. Открывается экран с нужным шаблоном и применяются к нему стили.
4. Создается обработчик кликов для экрана.
5. Создается фабрика блоков.
6. Выполняется чтение блоков и их стилей из JSON-дкумента.
7. Описание блоков и их стилей передается в фабрику и создаются блоки, которые вставляются в шаблон экрана.
8. Всем блокам устанавливается обработчик кликов.
9. Всем блокам передается список разметок блоков, содержащихся внутри них. Если этот список не пуст, то переход к пункту 6, иначе пункт 10.
10. Обрабатываются клики пользователя, запросы прав, отправка файлов, форм и другие события жизненного цикла. Если клик требует открытия нового экрана, то загружается JSON-разметка нового экрана и переход к пункту 2.

Рассмотрим пример реализации приложения-интерпретатора для мобильной операционной системы Android. Экраном в ОС Android является Activity. Для различных шаблонов экранов создаются разные Activity с различной разметкой и разной логикой их инициализации. Для некоторых – это установка панели навигации, для других – заполнение вкладок. Activity содержит экземпляр класса фабрики блоков. Фабрика блоков реализует один метод – создание android.View по id блока и типу блока. Id блока необходим для поиска стилей и данных, а в зависимости от типа блока фабрика инстанциирует экземпляр android.View. Созданные блоки необходимо добавить на экран. Для этого в разметке Activity содержится контейнер ViewGroup. В качестве ViewGroup может выступать LinearLayout. В большинстве случаев его будет достаточно, так как блоки представляют собой горизонтальные элементы, расположенные вертикально друг за другом.

Фабрика блоков инстанциирует экземпляр android.View. Для этого она выполняет сопоставление типа блока с FactoryTask. FactoryTask – интерфейс, позволяющий по id блока и необходимым зависимостям вернуть экземпляр android.View. Для каждого блока необходимо создать реализацию FactoryTask, в которой будет описан процесс создания блока. Стоит обратить внимание, что FactoryTask только создает блок, а не стилизует и заполняет его другими блоками и данными. Создание android.View выполняется при помощи LayoutInflater. FactoryTask обеспечивает создаваемый экземпляр необходимыми зависимостями, такими как обработчик кликов, менеджер прав и другие.

После того, как блок создан и добавлен во ViewGroup на экране, в блок приходит callback жизненного цикла. Этот callback запускает процесс стилизации и заполнения блоков. FactoryTask предоставляет часть JSON-разметки блока, которая содержит стили и данные для этого блока. Если реализовывать интерпретатор на языке Kotlin, то удобно создать extension функции для TextVeiw, ImageView, Button и других используемых компонентов, которые будут принимать на вход стиль текста, картинки, кнопки или другие стили и применять к компоненту. В этот же момент устанавливаются слушатели кликов и данных, которые предоставляет

FactoryTask. Слушатели закрыты интерфейсом, а его реализацией является Activity.

В некоторых случаях необходимо взаимодействие блоков на одном экране между собой, например, если на экране есть три блока одинакового типа для выбора даты. Такое разделение часто используется в приложениях для разбиения рабочего дня на утро, день, вечер. Экран с тремя блоками необходим для того, что бы пользователь забронировал время какой-либо услуги. Выбранная дата выделяется каким-либо селектором. Взаимодействие между этими блоками необходимо для того, чтобы в каждом из блоков одновременно не было выбрано одно и тоже время. Такое взаимодействие может быть реализовано при помощи сервиса, который предоставляется с экрана блокам, либо блокам от блоков-контейнеров. Сервис выполняет роль хранилища и наблюдателя. Во время создания блока необходимо сообщить сервису о том, что создаваемый блок должен получать информацию о том, что выбрана какая-либо дата. Когда осуществляется выбор даты на каком-либо блоке, то из блока сообщается сервису о выбранной дате, а сервис уведомляет все блоки о том, что выбранная дата изменена. В блоке должна быть обработка сообщения о смене блока. В ней выполняется проверка, есть ли выбранная дата в блоке. Если выбранной даты нет, можно понять, что необходимо снять выделение с даты в этом блоке. Такой случай может быть не только для выбора даты. Это может быть бронирование столика, покупка билета и другие действия. Главное, что бы выбираемая сущность имела уникальный идентификатор для возможности определения, в каком именно блоке необходимо выполнять выделение элемента.

В мобильных операционных системах существуют операции, на выполнение которых необходим запрос пользовательских разрешений [8]. К ним относятся доступ к местоположению, чтение контактов, доступ к внешней памяти и другие. При выполнении такой операции на экране появляется диалоговое окно, в котором описано разрешение и возможность разрешить или отклонить запрос. Может возникнуть такая ситуация, что двум разным блокам на экране потребуются одинаковые права. Если каждый блок будет запрашивать права, которые ему необходимы, то пользователь увидит два диалоговых окна, которые будут отображены один на другом. Если принять разрешение на одном из окон, то второе по-прежнему останется активным. Такое поведение приложения не является нормальным. Для решения этой проблемы может быть создан менеджер запроса прав. В операционной системе Android его экземпляр будет создавать Activity, передавая себя в качестве объекта, у которого можно запрашивать права и сообщать о данных, которые будут возвращены в метод жизненного цикла onRequestPermissionsResult. На менеджер будет возлагаться обязанность по упорядочиванию запросов прав и уведомлению блоков, о получении тех или иных прав. Из блока можно передать в менеджер функцию-замыкание, входным параметром которой будет переменная, показывающая, получены ли права, и константу, определяющую запрашиваемые права. Для работы менеджера необходима структура данных, которая является очередью с ключами элементов, которые являются списками функций. Тогда алгоритм работы менеджера будет следующим:

1. Добавить в очередь массивов под ключом, переданным менеджеру, функцию замыкание.

2. Если в очереди один элемент, то перейти к пункту 3, иначе 4.

3. Выполнить запрос прав с ключом первого элемента очереди, определяющим тип прав.

4. Обработать результат запроса. Выполнить функции из первого элемента очереди с результатом запроса прав.

5. Удалить первый элемент в очереди.

6. Если очередь не пуста, перейти к пункту 3, иначе конец.

Использование такого менеджера прав позволит избежать появления дублирующихся окон запроса прав и позволит предоставлять результаты запроса не от Activity к блоку, а от Activity к менеджеру, от менеджера к блоку, что позволит выполнять модульное тестирование компонентов [9].

ЗАКЛЮЧЕНИЕ

В результате реализации описанного подхода к разработке мобильных приложений и приложения-интерпретатора могут быть решены проблемы, которые возникают в результате использования нативных мобильных приложений и веб-приложений. Благодаря интерпретации в режиме реального времени пользователям не понадобится устанавливать приложения, тратить на это время и память устройства. Запуск с помощью одного приложения-интерпретатора разных приложений позволит избежать установок множества однотипных приложений на смартфоне. Результат интерпретации разметки при помощи компонентов операционной системы позволит приложениям работать быстро и плавно даже на слабых устройствах.

Работа поддержана грантом РФФИ № 16-07-00487.

СПИСОК ЛИТЕРАТУРЫ

1. What is native app? – Режим доступа: https://searchsoftwarequality.techtarget.com/definition/native-application-native-app. – (Дата обращения: 11.07.2018).

2. Веб-приложение – Режим доступа: https://ru.wikipedia.org/wiki. – (Дата обращения: 11.07.2018).

3. *Ерошенко, М. Н.* Гибридные мобильные приложения: основные характеристики и особенности / М. Н. Ерошенко // Вестник магистратуры, 2015, № 5 (44), Том 1, с. 19–20.

4. How Much Should it Cost to Hire an App Developer? – Режим доступа: https://www.upwork.com/blog/2014/01/much-cost-hire-app-developer/ – (Дата обращения: 11.07.2018).

5. Приложения в Google Play – KFC Клуб. – Режим доступа: https://play.google.com/store/apps/details?id=com.loyaltyplant.partner.kfc. – (Дата обращения: 11.07.2018).

6. Приложения в Google Play – Black Star Burger. – Режим доступа: https://play.google.com/store/apps/details?id=com.loyaltyplant.partner.blackstarburger. – (Дата обращения: 11.07.2018).

7. *Murata M., Kohn D. and C. Lilley* (2009-09-24). «Internet Drafts: XML Media Types». – Режим доступа: http://tools.ietf.org/html/draft-murata-kohn-lilley-xml-03. – (Дата обращения: 11.07.2018).

8. *Дейтел, П.* Android для разработчиков. 3-е издание. / П. Дейтел, Х. Дейтел, А. Уолд. – СПб. : Питер, 2016. – 512 с.

9. *Milano, D. T.* Android Application Testing Guide / D. T. Milano – Packt Publishing Ltd., 2011. – 314 p.

Рязанов О. Ю. – магистрант кафедры программного обеспечения вычислительной техники и автоматизированных систем, Белгородский государственный технологический университет им. В. Г. Шухова.
Тел.: +7 (915) 571 – 75 – 85
E-mail: weery@live.ru

Рязанов Ю. Д. – доцент кафедры программного обеспечения вычислительной техники и автоматизированных систем, Белгородский государственный технологический университет им. В. Г. Шухова.
Тел.: +7 (910) 325 – 73 – 75
E-mail: razanov.yd@bstu.ru

Ryazanov O. Yu. – Graduate student of the Department of Software Computer and Automated Systems, BSTU after V. G. Shukhov.
Tel.: +7 (915) 571 – 75 – 85
E-mail: weery@live.ru

Ryazanov Yu. D. – Associate Professor of the Department of Software Computer and Automated Systems, BSTU after V. G. Shukhov.
Tel.: +7 (910) 325 – 73 – 75
E-mail: razanov.yd@bstu.ru