Санкт-Петербургский Государственный Университет
Математико-механический факультет
Кафедра системного программирования

Генерация Android приложений в проекте QReal:Web

Курсовая работа студента 344 группы
Захарова Владимира Александровича

Научный руководитель: ст. преп. Брыксин Тимофей Александрович

Санкт-Петербург
2014

Оглавление

Введение

Не возникает сомнений, что мобильные технологии становятся неотъемлемой частью нашей жизни. При этом все большую популярность приобретают смартфоны и планшеты. Аналитическая компания Gartner[1] в феврале 2014 года заявила, что в конце 2013 года продажи смартфонов составили 57,6% от всех продаж мобильных телефонов. Поэтому возникает необходимость в создании большого количества мобильных приложений.

Высокая сложность и стоимость разработки накладывают серьезные ограничение на создание приложений. В связи с этим появляется потребность в создании инструмента, позволяющего быстро разрабатывать мобильные приложения. Кроме того, хочется, чтобы была возможность задавать относительно сложную логику поведения, и чтобы инструментом могли бы пользоваться люди весьма отдаленные от программирования.

В связи с развитием сети Интернет и браузерных технологий разумно реализовать именно браузерный инструмент, т.к. это позволит пользователям получать к нему доступ из любого места, где есть Интернет, и не будет требовать установки каких-либо дополнительных компонент. Кроме этого, это даст возможность контролировать доступ к инструменту.

На данный момент существует довольно много средств для создания приложений, как простых, основанных на наборе шаблонов, так и более сложных, позволяющих задавать некоторую нешаблонную логику. Но в каждом из этих средств есть некоторые недостатки, которые и планируется устранить в проекте QReal:Web.

Проект QReal:Web

В 2013 году на кафедре системного программирования СПбГУ были написаны прототипы дизайнеров интерфейса и логики, эмулятора и генератора под платформу Android. В данных решениях остался ряд недостатков:

[1] http://www.gartner.com/newsroom/id/2665715

- Использование неудобного для редактора формата текущего состояния дизайнеров. Для описания интерфейса использовался формат очень близкий к формату описания интерфейса в Android приложениях, такой формат позволял практически без изменений получать файлы, описывающие интерфейс, но он был неудобен для дизайнеров.
- Неудачный подход к заданию логики. Даже задание достаточно простой логики было довольно громоздким.
- Плохая расширяемость генератора. Хотя возможностей у редактора было немного, уже были серьезные проблемы с расширением функциональности генератора.

В связи с этим было принято решение разработать новый редактор и генератор к нему на основе полученного опыта.

Постановка задачи

Целью моей курсовой работы в проекте QReal:Web было создание генератора Android приложений, который по полученным данным от редактора создает готовое приложение.

Для достижения этой цели мною были сформулированы следующие задачи:

1. Исследовать существующие подходы к генерации кода.
2. Выбрать необходимые для решения задачи инструменты.
3. Реализовать генерацию внешнего вида приложения по данным, полученным от дизайнера интерфейса.
4. Реализовать генерацию логики поведения по данным, полученным от дизайнера логики.
5. Реализовать автоматическую сборку целевого приложения по сгенерированным файлам.
6. Провести апробацию генератора на наборе примеров.

Существующие подходы к генерации

В книге “Domain Specific Languages” Мартина Фаулера [1] описаны несколько подходов к генерации кода.

Преобразовательная генерация (Transormer generation)

В данном подходе для генерации кода пишется преобразователь (transformer), который проходит по входящей семантической модели и выдает необходимый код. Данный подход применяется, когда итоговый текст просто связан с входящей моделью, и большая часть кода генерируется.

Шаблонная генерация (Template generation)

В этом подходе заранее вручную пишутся выходные файлы с местами для вставки для тех мест, которые могут изменяться. Места для вставки заполняются кодом, зависящем от текущего контекста, и получается итоговый выходной файл. Применяется, когда в выходных файлах много статического содержимого.

Осведомленная о модели генерация (Model-Aware generation)

В данном подходе код генерируется с явным подобием семантической модели, чтобы сохранить разделение общего и специального кода внутри сгенерированного кода.

Независимая от модели генерация (Model Ignorant generation)

В этом подходе вся логика жестко задается в сгенерированном коде, поэтому нет явного представления семантической модели.

Проанализировав подходы, в моей работе было принято решения использовать комбинацию преобразовательной и шаблонной генераций, т.к. основную структуру файлов исходного кода можно представить в виде шаблонов с местами для вставки, а по полученным данным сгенерировать необходимый контекст.

Взаимодействие компонент сервиса

В проекте QReal:Web взаимодействие компонент сервиса происходит следующим образом. Пользователь в браузере задает внешний вид и логику поведения приложения. Далее текущее состояние дизайнеров интерфейса и логики сериализуются в JSON[2] (JavaScript Object Notation) файлы, которые отправляются на сервер. Сервер передает эти файлы генератору и просит сгенерировать приложение. На рис. 1 показано данное взаимодействие. Прототип дизайнера логики разработан в рамках курсовой работы студента 344 группы Агеева Дениса. Сервер и прототип дизайнера интерфейса разработан в рамках дипломной работы студента 445 группы Бумакова Никиты.

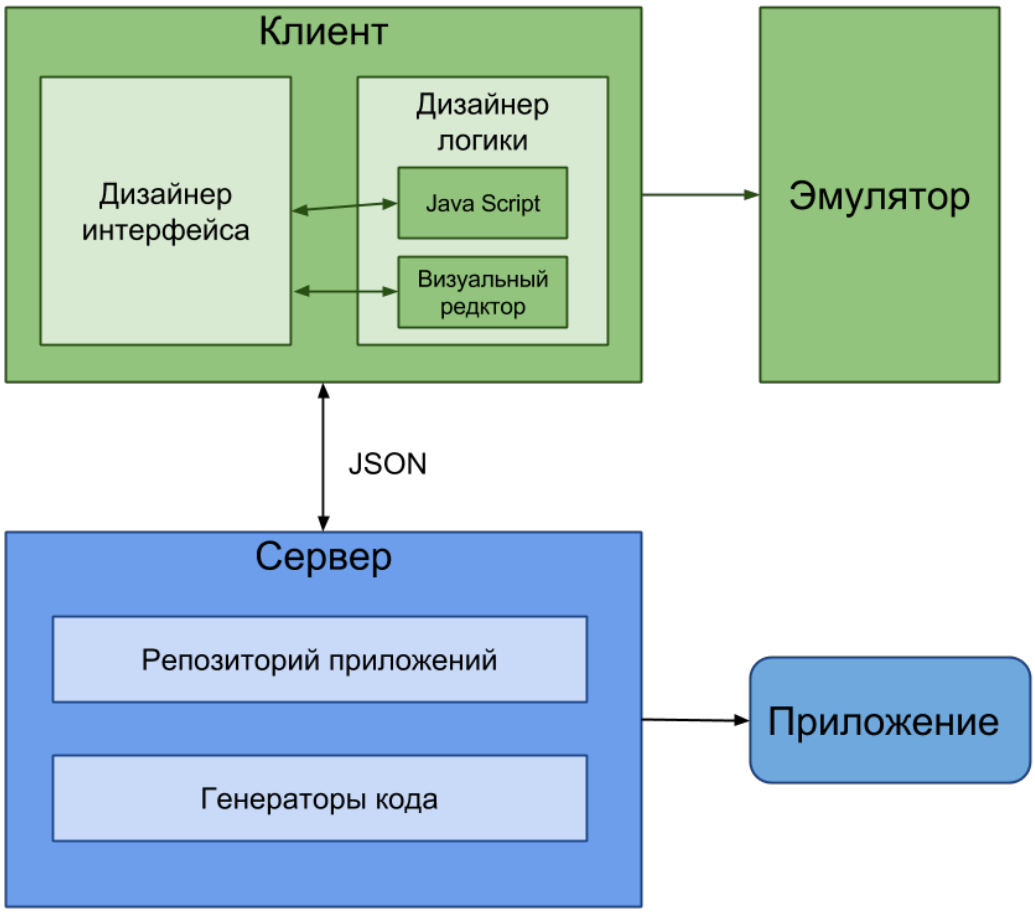

Рис. 1. Взаимодействие компонент сервиса

[2] http://json.org/

Реализация

Архитектура

Был разработан парсер JSON файлов и набор классов для сборки файлов, необходимых для получения приложения. Классы для сборки файлов используют заранее написанные шаблоны с проиндексированными местами для вставки. При их реализации был применен паттерн Строитель (Builder) [4], благодаря этому есть возможность подменить их для генерации под другие мобильные платформы. При проходе по JSON файлу в экземпляры классов для сборки заносятся строки, которые и будут вставлены в шаблоны, при вызове метода build.

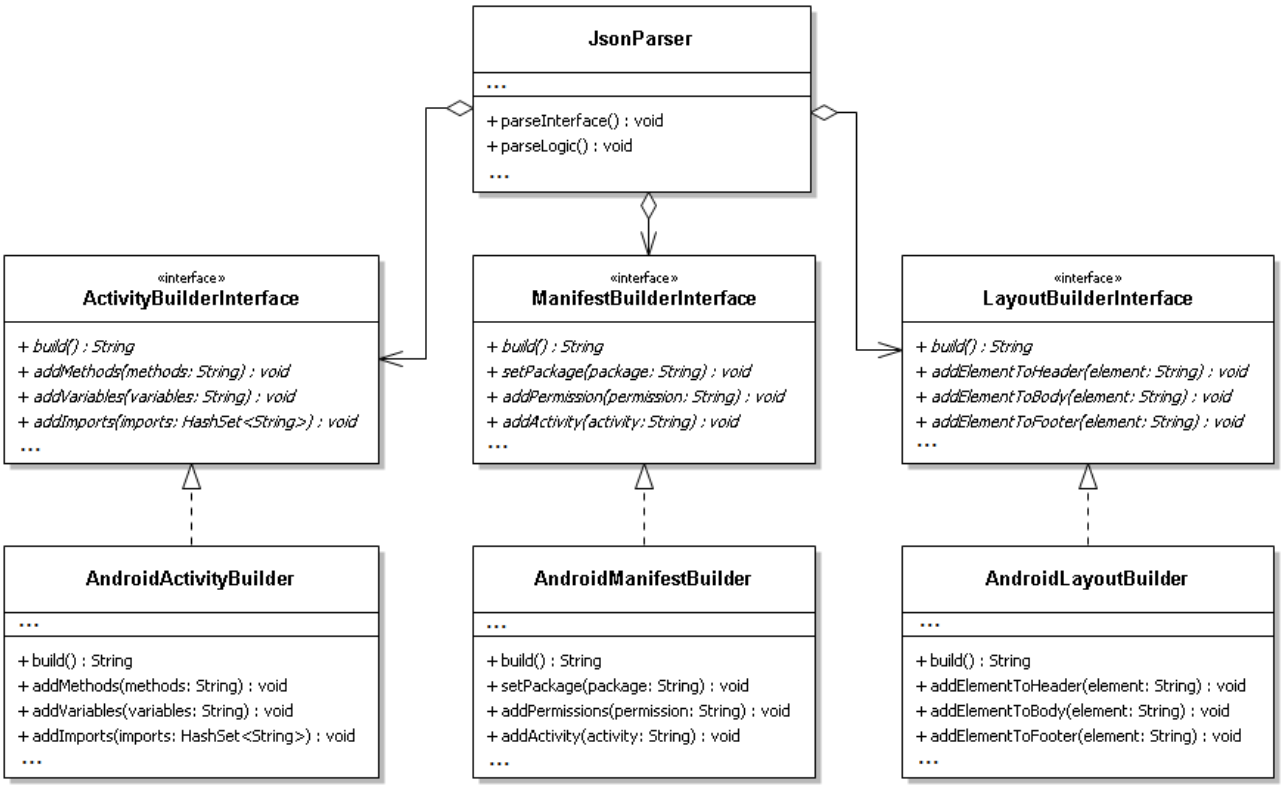

Рис. 2. Архитектура.

Используемые инструменты

В качестве языка программирования был взят C#, т.к. под него написано большое количество различных библиотек, он неплохо документирован, объектно-ориентированный язык подходит для создания архитектуры с возможностью малозатратного расширения функциональности генератора. Кроме того, были выбраны дополнительные инструменты: JSON.NET[3], OSMdroid[4], Apache Ant[5]. Далее подробно будет описан каждый выбор.

Парсер JSON

Для реализации парсера JSON файлов была выбрана библиотека JSON.NET. Она зарекомендовала себя как хорошо документированная и удобная в использовании библиотека. Кроме того, мне было необходимо иметь возможность потокового чтения файла, и иметь способ получать объекты и массивы JSON файла по заданному имени. Эта библиотека удовлетворяет данным требованиям.

Логика

Для описания состояния дизайнера логики был разработан следующий формат:

```
{
   "nodes": [ … ],
   "services": [ … ],
   "links": [
      {
         "source": " … ",
         "target": " … "
      }
```

[3] http://json.codeplex.com/
[4] https://code.google.com/p/osmdroid/
[5] http://ant.apache.org/

```
  ]
}
```

Здесь “nodes” - элементы, участвующие в задании логики (кнопки, поля для текста ввода, карты и т.д.), “services” - сервисы, описывающие некоторое действие (переход между экранными формами, получение местоположения и т.д.), “links” - связи, соединяющие “nodes” и “services”, тем самым задавая действие при активации элемента.

Парсер с файлом для логики работает следующим образом: в циклах проходит по массивам “links” и “services”, при этом информация заносится в хэш-таблицы. Для одной хэш-таблицы данные достаются из массива “links”. В качестве ключа берется значение поля “source”, а значения - значение поля “target”. В другую хэш-таблицу заносится пара (название сервиса ; объект, соответствующий ему). Информация из хэш-таблиц затем используется при работе с файлом, отвечающим за описание состояния дизайнера интерфейса.

Визуальная часть

Для описания состояния дизайнера интерфейса был разработан следующий формат:

Project

```
{
   "type": "App",
   "name": "Project name",
   "projectPackage": "Project package",
   "Pages": [
         ...
   ]
}
```

"Pages" - массив из элементов "Page"

Page

```
{
   "type": "Page",
   "id": "Page name",
   "header": {
            "type": "Header",
            "id": "headerId",
            "title": "Title",
            "theme": "theme name"
   },
   "padding": "padding value",
   "Controls": [
         ...
   ]
}
```

"Controls” - массив из элементов управления

Парсер последовательно проходит по файлу, при этом заносит нужные строки в экземпляры классов для сборки. При проходе по массиву элементов управления, если в хэш-таблицах с информацией о логике есть соответствующий элемент, то по объекту-значению добавляются нужные строки исходного кода.

Компоновщик для размещения элементов внутри строки

Дизайнер интерфейса поддерживает возможность размещать элементы на одной линии, пока они вмещаются в ширину экрана, иначе элемент переносится на следующую линию. В Android SDK нет стандартной поддержки такой возможности, поэтому мной был написан свой компоновщик, размещающий элементы именно таким образом.

Был написан класс InlineLayout, который наследуется от стандартного класса ViewGroup, в нем переопределены методы onMeasure, onLayout. Первый метод рассчитывает размер компоновщика на основе размеров его сыновей. Второй размещает сыновей внутри компоновщика необходимым образом.

Написанный компоновщик подключается в виде библиотеки и используется в XML файле как стандартный компоновщик.

На рис. 3 изображен пример работы компоновщика. В данном пример кнопки 2 - 5 размещаются внутри линии. Разное количество кнопок на линии в дизайнере интерфейса и сгенерированном приложении, т.к разная ширина экрана у дизайнера и реального устройства.

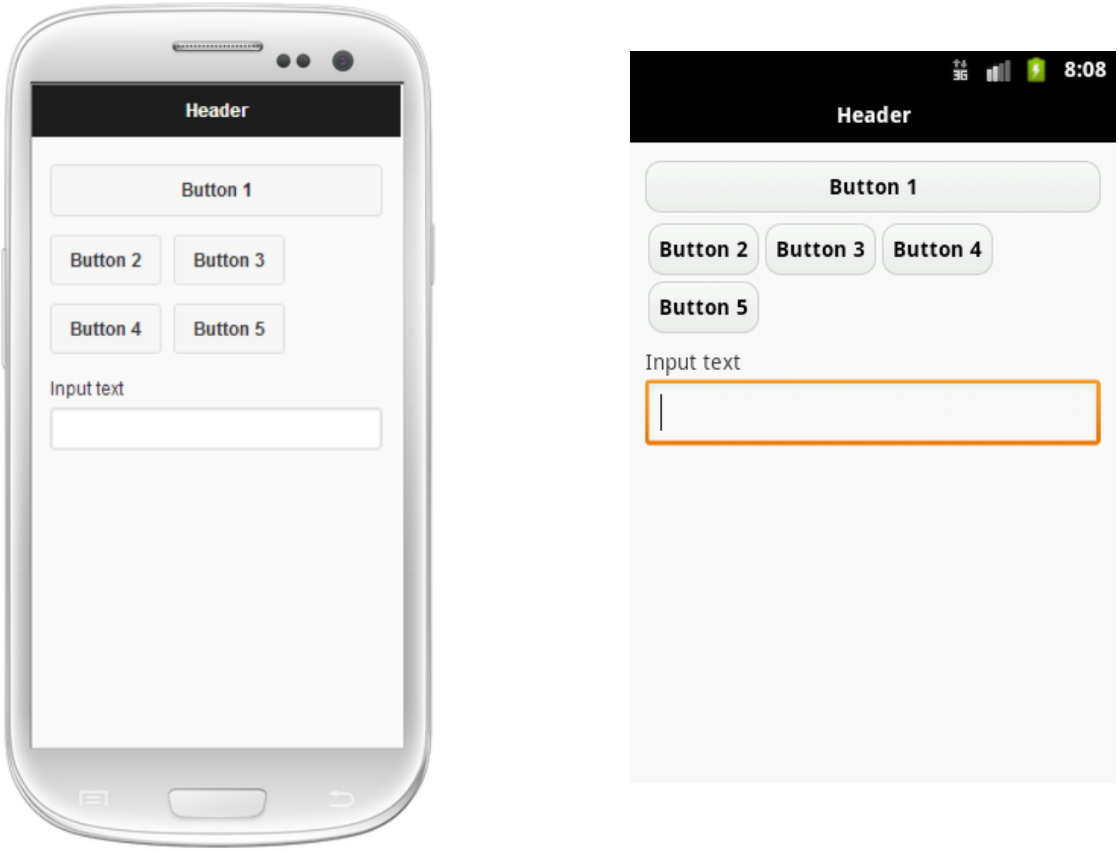

Рис. 3. Пример работы компоновщика (слева - экран дизайнера интерфейса, справа - экран реального устройства)

Поддержка карт

Для поддержки карт сначала планировалось использовать стандартные для платформы Android Google Maps[6], но они требуют создания ключа для каждого приложения по сертификату приложения SHA-1 и имени пакета, при этом ключ вручную создается в консоли разработчика Google. Поэтому мой выбор пал на OSMdroid - это OpenStreetMap[7] для платформы Android. Для использования этих карт не требуется создания какого-либо ключа, поэтому их удобно использовать при генерации приложений.

[6] https://developers.google.com/maps/documentation/android/start
[7] http://www.openstreetmap.org/about

Сборка

Для автоматической сборки приложения использовались Android SDK[8] и утилита Apache Ant.

При помощи стандартных средств Android SDK по сгенерированным файлам создается build.xml, который использует Apache Ant для сборки приложения. На выходе создается apk файл, который можно установить на телефон. Был написан скрипт, который вызывает соответствующие команды с нужными параметрами.

Существуют другие средства для сборки, например, Apache Maven[9], но они не так удобны в данном случае, т.к. файл структуры проекта пришлось бы создавать вручную.

[8] http://developer.android.com/sdk/index.html
[9] http://maven.apache.org/

Апробация

Была проведена апробация, целью которой является проверка возможности генерации приложения по предоставленным от дизайнеров интерфейса и логики данным. Работа генератора была опробована на 2 примерах: приложение “визитка” и приложение с использованием сенсоров устройства.

Приложение “Визитка”

Приложение “визитка” - приложение, показывающее общую информацию об организации. Оно состоит из фиксированного содержимого и набора кнопок, при нажатии на которые пользователь переходит на другой экран. На рис. 4 изображена схема задания в дизайнере логики перехода между формами при нажатии на кнопку. Кнопка соединяется с сервисом навигации, у которого есть свойство, в котором указано на какую форму необходимо перейти. На рис. 5 изображен основной экран приложения в редакторе интерфейса и на реальном устройстве. Процесс генерации подробно будет описан в следующем, более показательном, примере.

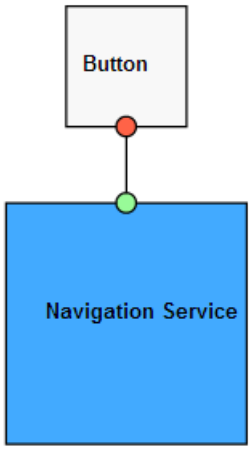

Рис. 4. Схема перехода между формами

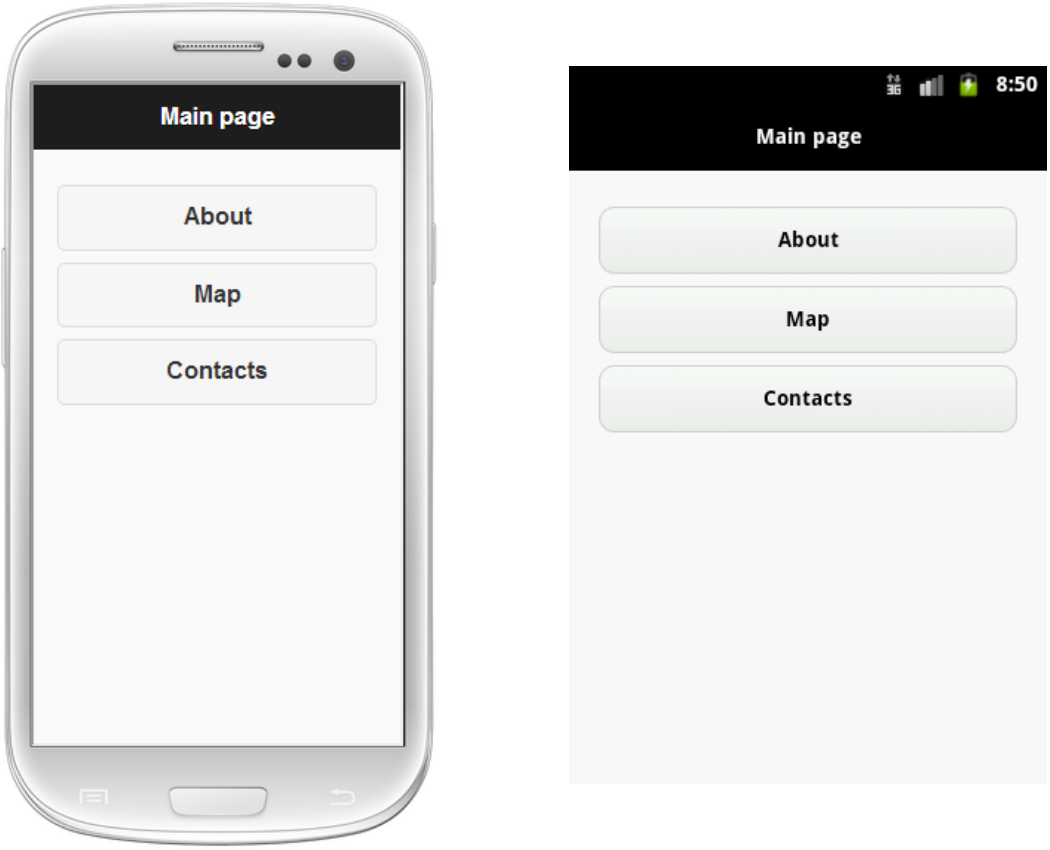

Рис. 5. Основной экран приложения “визитка” (слева - экран дизайнера интерфейса, справа - экран реального устройства)

Пример взаимодействия с сенсорами телефона

В качестве примера взаимодействия с сенсорами телефона было создано приложение, показывающее местоположение пользователя на карте. Запрос местоположения происходит по нажатию на кнопку. На рис. 6 показан способ задания в дизайнере логики такого поведения приложения. Кнопка соединяется с сервисом геолокации, который, в свою очередь, с картой, на которой необходимо указать местоположение. На рис. 7 слева изображено как в дизайнере интерфейса задавался внешний вид приложения, справа - сгенерированное приложение. Далее опишем процесс генерации.

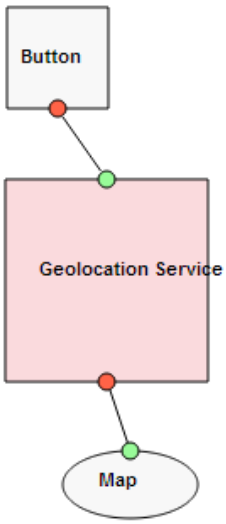

Рис. 6. Схема обновления положения пользователя

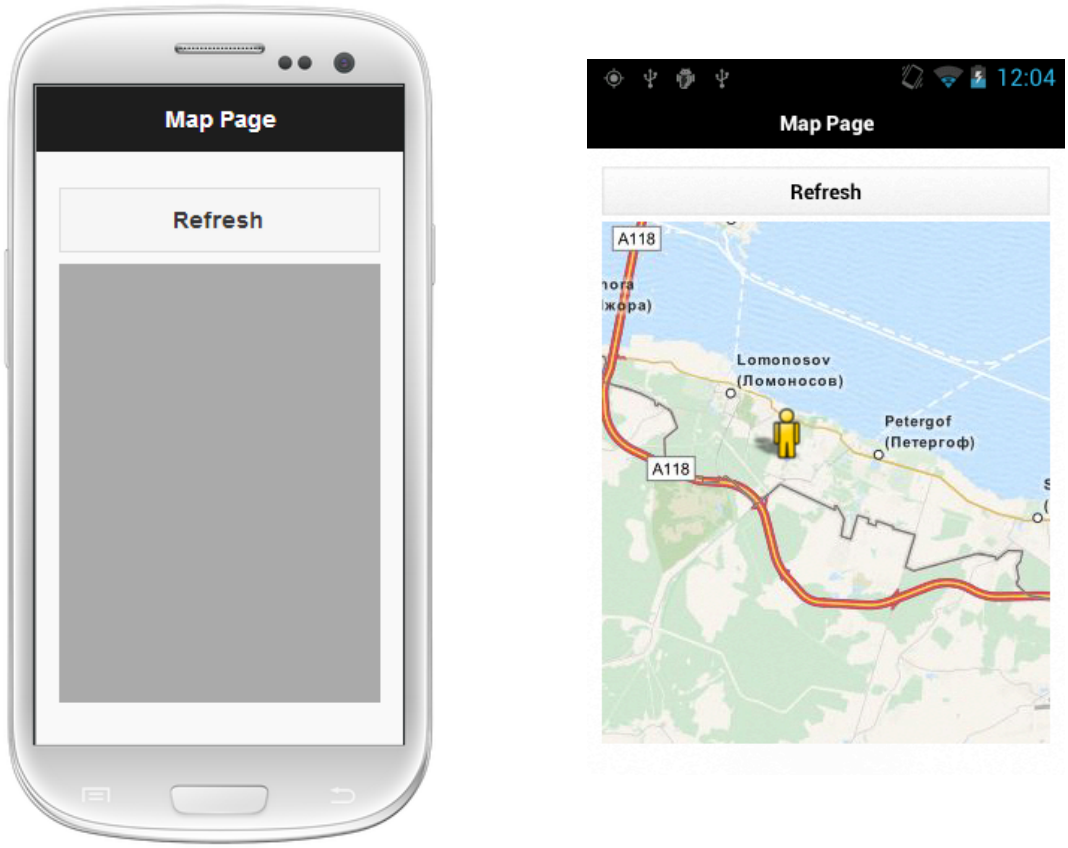

Рис. 7. Приложение, использующее геолокацию (слева - экран дизайнера интерфейса, справа - экран реального устройства)

Генератору на вход приходят JSON файлы следующего вида:

Задание интрефейса

```
{
   "type": "App",
   ...
   "Pages": [
      {
         "type": "Page",
         ...
         "Controls": [
            {
               "type": "Button",
               "id": "button2",
               "text": "Refresh",
               ...
            },
            {
               "type": "Map",
               "id": "map1"
            }
         ]
      }
   ]
}
```

Задание логики

```
{
   "nodes": [
      ...
   ],
   "services": [
      {
         "id": "geoloc1",
         "type": "GeolocationService"
      }
   ],
   "links": [
      {
         "source": "button2",
         "target": "geoloc1"
      },
      {
         "source": "geoloc1",
         "target": "map1"
      }
   ]
}
```

Сначала обрабатываем файл с логикой. В цикле проходим по массиву “links”, в хэш-таблицу заносим две пары ключ-занчение: (button2, geoloc1), (geoloc1, map1). Аналогично в цикле проходим по массиву "services" и в другую хэш-таблицу заносим пару (geoloc1, {"id": "geoloc1", "type": "GeolocationService"}), т.е. в качестве значения выступает объект, соответствующий названию.

При проходе по файлу задания интерфейса генерируется текст в XML[10](Extensible Markup Language) файлах для задания интерфейса, а также для кнопки по ключу “button2” в хэш-таблице получаем имя сервиса геолокации “geoloc1”, по нему получаем объект в другой хэш-таблице, в нем по значению поля “type” определяем тип сервиса, по ключу “geoloc1” получаем карту “map1”, на которой и нужно показать местоположение, генерируем соответствующий исходный код.

[10] http://www.w3.org/TR/REC-xml/

Результаты

Таким образом в рамках данной курсовой работы мною были сделано:

1. Изучены существующие походы к генерации кода, выбран подходящий подход.
2. Выбраны подходящие инструменты для реализации генератора.
3. Реализована генерация кода внешнего вида приложения по присланному JSON файлу.
4. Реализована генерация кода логики поведения приложения по JSON файлу.
5. Реализована автоматическая сборка приложения.
6. Проведена апробация работы генератора на конкретных примерах.

Список литературы

[1] Domain Specific Languages — Martin Fowler, 2010

[2] Язык программирования C# 5.0 и платформа .NET 4.5 — Эндрю Троелсен [Andrew Troelsen], 2013

[3] C# 4.0. Полное руководство — Герберт Шилдт [Herbert Schildt], 2011

[4] Приемы объектно-ориентированного проектирования. Паттерны проектирования — Эрих Гамма [Erich Gamma], Ричард Хелм [Richard Helm], Ральф Джонсон [Ralph Johnson], Джон Влиссидес [John Vlissides], 2013

[5] Командная строка Microsoft Windows — Уильям Р. Станек [William R. Stanek], 2004

[6] Professional Android 4 Application Development — Reto Meier, 2012

[7] Android 4 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов — Сатия Коматинени [Satya Komatineni], Дэйв Маклин [Dave MacLean], 2012

[8] JSON.NET, URL: http://json.codeplex.com/

[9] Android SDK, URL: http://developer.android.com/guide/index.html

[10] Apache Ant, URL: http://ant.apache.org/

[11] OSMdroid, URL: https://code.google.com/p/osmdroid/

[12] JSON, URL: http://json.org/