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

Создание инфаструктуры редактора диаграмм в браузере

Курсовая работа студента 344 группы
Агеева Дениса Витальевича

Научный руководитель: ст. преп. Брыксин Тимофей Александрович

Санкт-Петербург

2014

Оглавление

Оглавление
1. Введение
2. Постановка задачи
3. Существующие решения редакторов диаграмм
    3.1 Diagramo
    3.2 Rappid
    3.3 jsUML2
    3.4 Вывод
4. Реализация
    4.1. Инструменты
        4.1.1 TypeScript
        4.1.2 jQuery
        4.1.3 AngularJS
        4.1.4 JointJS
5. Архитектура редактора
6. Апробация
    6.1 Приложений “Визитка
    6.2 Взаимодействие с сенсорами телефона
7. Результаты
В результате проделанной работы были получены следующие результаты:
8. Литература

1. Введение

Ни для кого не секрет, что за последние пару лет мобильные технологии набрали огромную популярность. Это связано с доступностью смартфонов и планшетов, развитием информационных технологий. Так же по статистике аналитической компании Gartner[1], в 2013 году объем продаж смартфонов превысил отметку в 50% среди всех продаж мобильных телефонов. По этим причинам возникает потребность в создании большого числа мобильных приложений.

Но чтобы создавать мобильные приложения, даже самые простые, человек должен обладать навыками программирования, устанавливать нужный софт на компьютер, что очень усложняет процесс создания мобильного приложения. В связи с этим появляется потребность в создании конструктора мобильных приложений, не требующего от пользователя обладать навыками программирования.

Так же сейчас наблюдается тенденция перехода от десктопных приложений, устанавливаемых конечному пользователю к онлайн-сервисам. Они позволяют пользователю работать без предварительной установки нужных приложений, достаточно только интернета и браузера.

На данный момент существует множество решений для создания мобильных приложений, но нет решения, которое позволяло бы создавать мобильное приложение, которое можно было бы наделять сложной бизнес-логикой, не владея навыками программирования. Поэтому в рамках проекта QReal:Web было решено реализовать редактор диаграмм, который мог бы задавать логику мобильного приложения визуально.

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

2. Постановка задачи

Целью моей курсовой работы в данном проекте было создание движка редактора диаграмм. Для достижения этой цели мною были сформулированы следующие задачи:

1) Проанализировать существующие решения редакторов диаграмм.
2) Выбрать нужные инструменты для создания редактора.
3) Разработать инфраструктуру редактора диаграмм обладающего следующими требованиями:
    a) Возможность описания логики приложения с помощью диаграмм.
    b) Легкая расширяемость редактора, то есть простое добавление нужных элементов в редактор диаграмм
    c) Импорт и экспорт диаграмм в формате JSON
    d) Наличие таких компонентов как палитра интструментов, рабочая область и редактор свойств элементов.

3. Существующие решения редакторов диаграмм

На данный момент существует множество приложений, позволяющих создавать и редактировать диаграммы в браузере. Большинство из них представляют из себя простую возможность создания диаграмм и сохранением как картинку, без наличия импорта, экспорта в каком-то текстовом формате, задания собственных объектов или свойств.

3.1 Diagramo

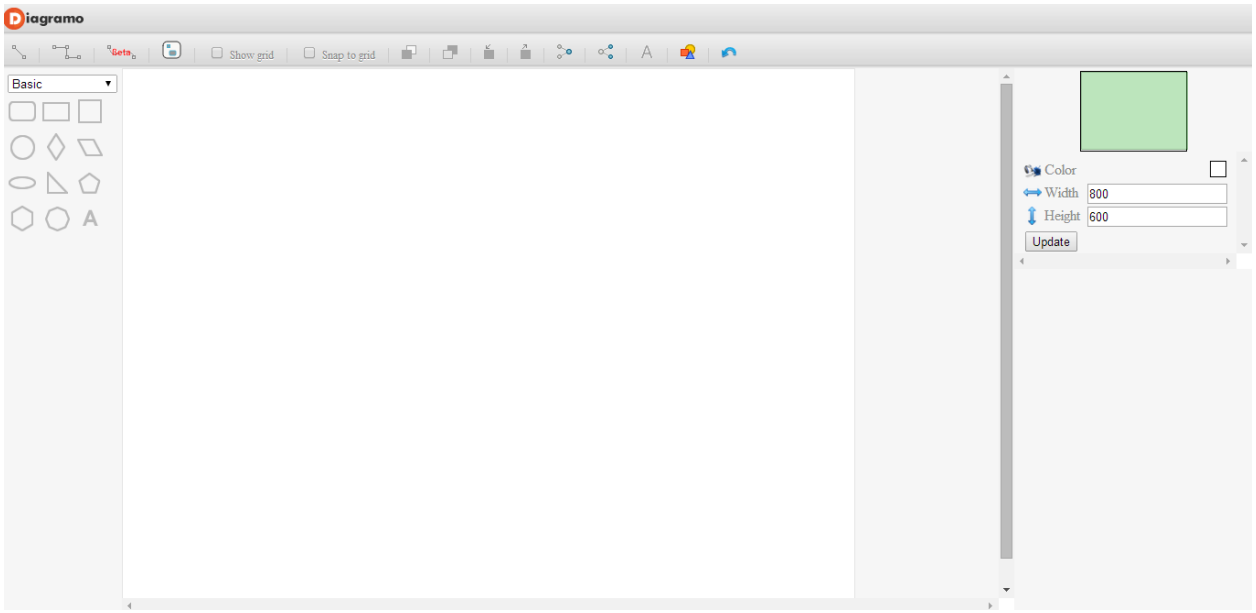

Рис. 1 Вид пользовательского интерфеса редактора Diagramo

Diagramo[2] является неплохим и браузерным редактором диаграмм с открытым исходным кодом. Он позволяет рисовать простые элементы, задавать их свойства. Но у него маленькая фунцкиональность: он умеет только рисовать диаграмм и экспортировать в картинку. Так же он тяжело расширяемый и сложно интегрируемый фреймворк, который не подходит для задания логики приложения.

[2] http://diagramo.com

3.2 Rappid

Rappid[3] - очень хороший продвинутый редактор диаграмм, имеет множество возможностей. Такие как: отрисовки диаграмм, создание собственных объектов, задание определенных свойств. Экспорт и импорт из JSON. Так же у него есть удобное API, которое можно было бы использовать, но к сожалению этот редактор является платным решением.

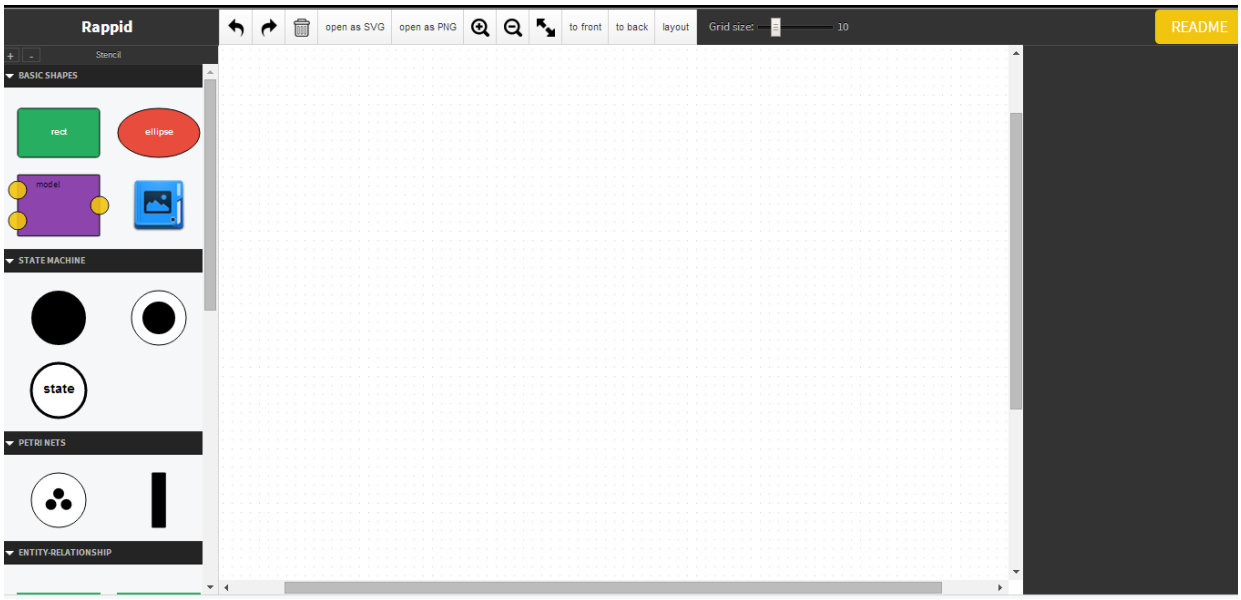

Рис. 3 Пользовательский интерфейс редактора диаграмм Rappid

3.3 jsUML2

jsUML2 - легкая библиотека HTML5/javascript с открытым исходным кодом для UML диаграмм. Она позволяет разработчику легко вставлять UML диаграммы в веб-приложения, просто вызывая ранее созданые методы в этом редакторе. В ней есть импорт и экспорт в xml. Это стандартный UML редактор диаграмм. Но, чтобы добавлять собственные элементы, нужно дописывать их код. К тому же нет возможности задать свой формат импорта и экспорта данных, валидации.

[3] http://jointjs.com/rappid

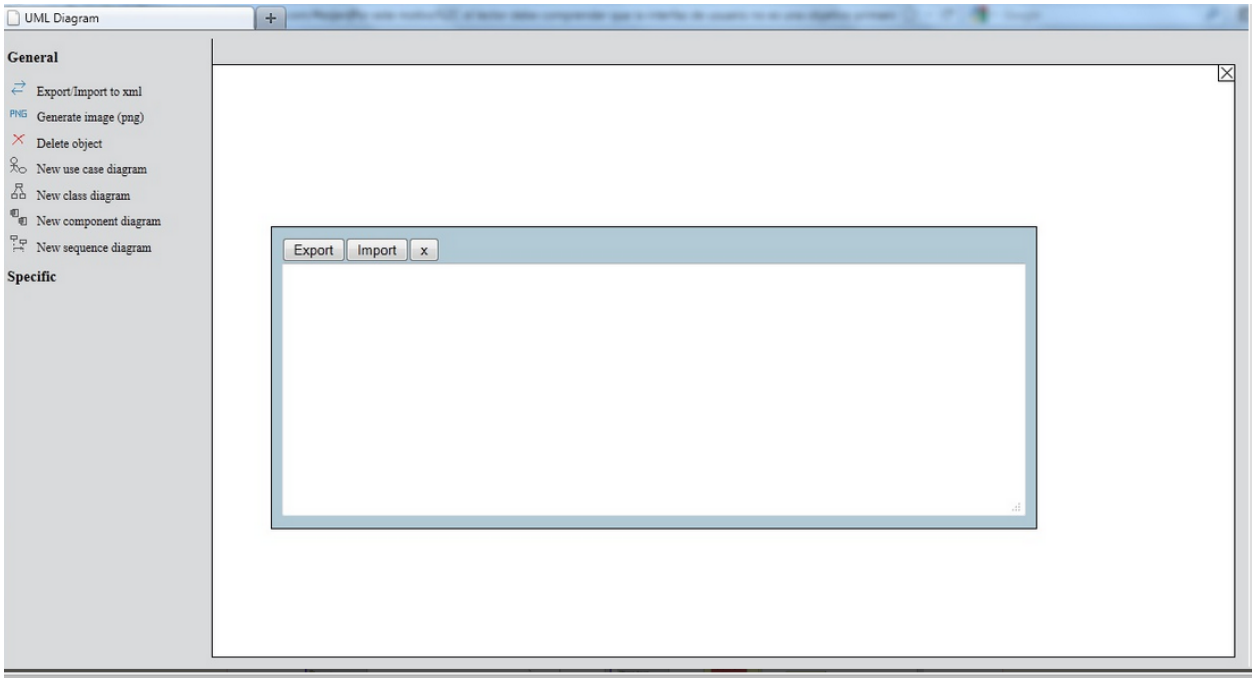

Рис. 4 Пользовательский интерфейс jsUml2

3.4 Вывод

Исходя из анализа существующих решений, видно, что пока нет гибкого решения, которое можно было бы переиспользовать для задания логики мобильного приложения, ввиду отсутствия простого инструмента валидации, создания собственных элементов с определенными свойствами, настройка импорта и экспорта диаграмм в определенном формате. Основные решения либо очень плохо расширяемы, мало функционы, либо не являеются проектами с открытым исходным кодом.

Таким образом было решено создать собственную, расширяемую инфаструктуру для задания логики мобильного приложения.

4. Реализация

4.1. Инструменты

Так как редактор диаграмм позиционируется как браузерное приложение, очевидным фактом является то, что он должен быть кросс-браузерным и не требовать от пользователя устанавливать стороннее программное обеспечение, например Adobe Flash Player, поэтому было решено для отображения диаграмм использовать технологию HTML5, которую поддерживает любой современный браузер. Основным языком разработки браузерных приложений является JavaScript. К сожалению, данный язык обладает рядом минусов. Во-первых, динамическая типизация, из-за нее могут появляться регрессионные ошибки, что увеличивает процесс разработки и отладки приложения. Во-вторых, отсутствие модульности, ООП - это делает процесс разработки приложения громоздским и сложным, тем более это затрудняет писать легко расширяемые приложения. Поэтому для решения этих проблем были выбраны следующие технологии : TypeScript[4], AngularJS[5], JQuery[6].

4.1.1 TypeScript

TypeScript - это язык, предложенный компанией Microsoft в 2012 году для создания веб-приложений. Язык транслируется в JavaScript код, что делает его совместимым с JavaScript приложениями.

Кроме того, любой JavaScript код будет валидным TypeScript кодом, это позволяет использовать JavaScript библиотеки в своем проекте. Так же TypeScript позволяет работать со статической типизацией, что упрощает процесс создания приложения. И в TypeScript есть поддержка ООП, модульности, что позволяет создавать гибкие, масштабируемые и расширяемые приложения. Таким образом этот язык решил наши проблемы с динамической типизацией, и расширяемостью.

4.1.2 jQuery

jQuery — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML.

Библиотека jQuery помогает легко :

- Получать доступ к любому элементу DOM.
- Обращаться к атрибутам и содержимому элементов DOM.

[4] http://www.typescriptlang.org/
[5] http://angularjs.org
[6] http://jquery.com

- Манипулировать элементами DOM.

4.1.3 AngularJS

При разработке приложения хотелось бы использовать некоторые шаблоны проектирования, в том числе и MVC, но поддержки этого шаблона изначально в языке TypeScript нет, поэтому возникает необходимость использования сторонней JavaScript библиотеки для создания MVC приложения. Такой подход позволит нам разделить наше приложения на независимые компоненты, позволит легко внедрить его в конструктор приложений, расширять логику, не изменяя часть, которая отвечает за отображение. Выбор пал на AngularJS, потому что эта библиотека обладает всем необходимым для поддержки MVC, хорошо документирована и проста в использовании.

4.1.4 JointJS

Наконец, чтобы значительно упростить процесс разработки редактора диаграмм, хотелось воспользоваться библиотекой, которая позволяла бы легко создавать элементы диаграмм, чтобы можно было ими манипулировать. С этой целью было проведено исследования существующих библиотек для рисования элементов диаграмм и получен следующий результат:

	Jgraph	Javascript InfoVis ToolKit	Draw2D	JointJS
Бесплатна	-	+	-	+
Хорошо документирована	+	+	+	+
HTML5 SVG	+	-	+	+
Настройка	+	+	+	+
Проста в использовании	+	-	-	+

Рис. 5 Сравнительная таблица библиотек для рисования диаграмм

Из этой таблицы видно, что JointJS[7] обладает всем нужными нам свойствами и было решено использовать ее для создания движка редактора диаграмм.

[7] http://jointjs.com

5. Архитектура редактора

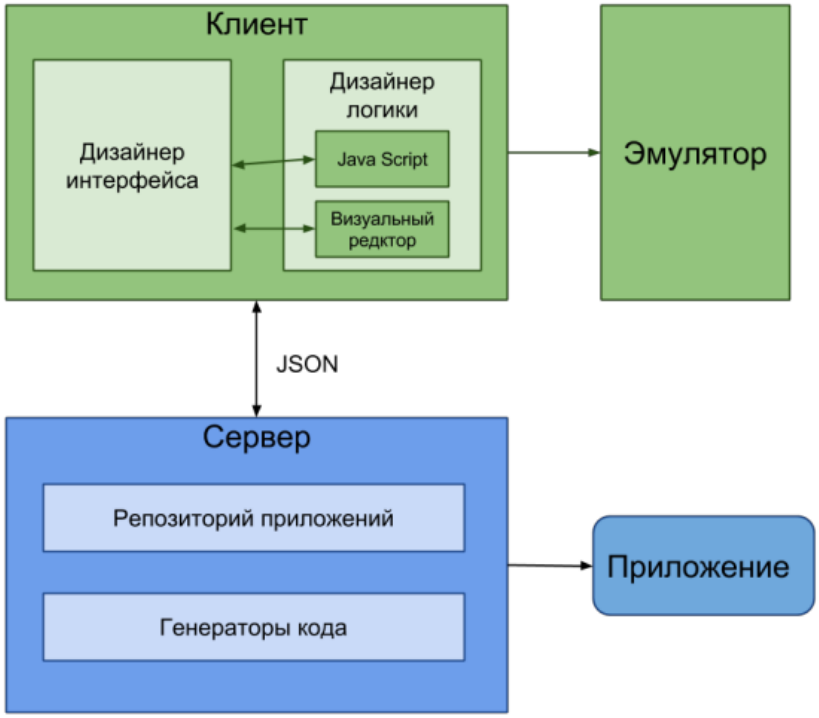

Рис. 6 Взаимодействие компонент сервиса

Пользователь задает элементы своего приложения в конструкторе мобильных приложений, после чего нажимает на кнопку “задать логику”, и конструктор генерирует метамодель, в которой описано, какие элементы присутствуют в редакторе и их некоторые свойства. Затем эта информация в формате JSON отправляется в редактор диаграмм, который в свою очередь преобразует это в диаграмму, определяются свойства элементов. Далее пользователь начинает задавать логику приложения. После этого нарисованная диаграмма транформируется в модель в формате JSON и в свою очерень отправляется обратно в конструктор, где генерируется приложение. Прототип конструктора приложений описан в рамках дипломной работы Бумакова Никиты 445 группы, генератор приложения описан в рамках курсовой работы Захарова Владимира 344 группы.

Метамодель описывает элементы, добавленные пользователем в конструкторе приложений. Модель задает уже логику между описанными элементами с помощью сервисов.

```
"nodes": [
    {
        "id": "elementId",
        "type": "elementType",
        "name": "elementName",
    }
  ],
  “activities” [
    “id” : “activityId”
  ]
```

Метамодель

```
 “links” : [
    {
        “source” : “sourceId”,
        “target” : “targetId”
    }
   ],
  “services” : [
     {
       “id” : “serviceId”,
       “type” : “serviceType”,
       “activity” : “activityId”
     }
]
```

Модель

На следующем рисунке показана архитектура разработанного в данной работе редактора диаграмм.

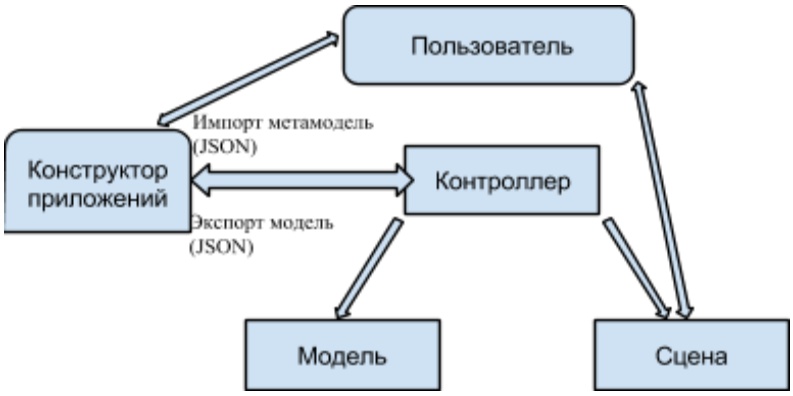

Рис. 7 Архитектура редактора диаграмм

Логически редактор состоит из трех компонент.

Первое, это контроллер. Он принимает от конструктора метамодель, преобразует его в графический вид, задания свойств объектов. Так же он отвечает за экспорт модели конструктору.

Модель состоит из всех возможных элементов, описывает их свойства, порты и как валидировать нашу диаграмму.

Сцена же представляет из себя рабочую область, где пользователь задает логику приложения ввиде диаграмм, так же там есть редактор свойств, в котором можно указывать нужные свойства нашим элементам диаграммы.

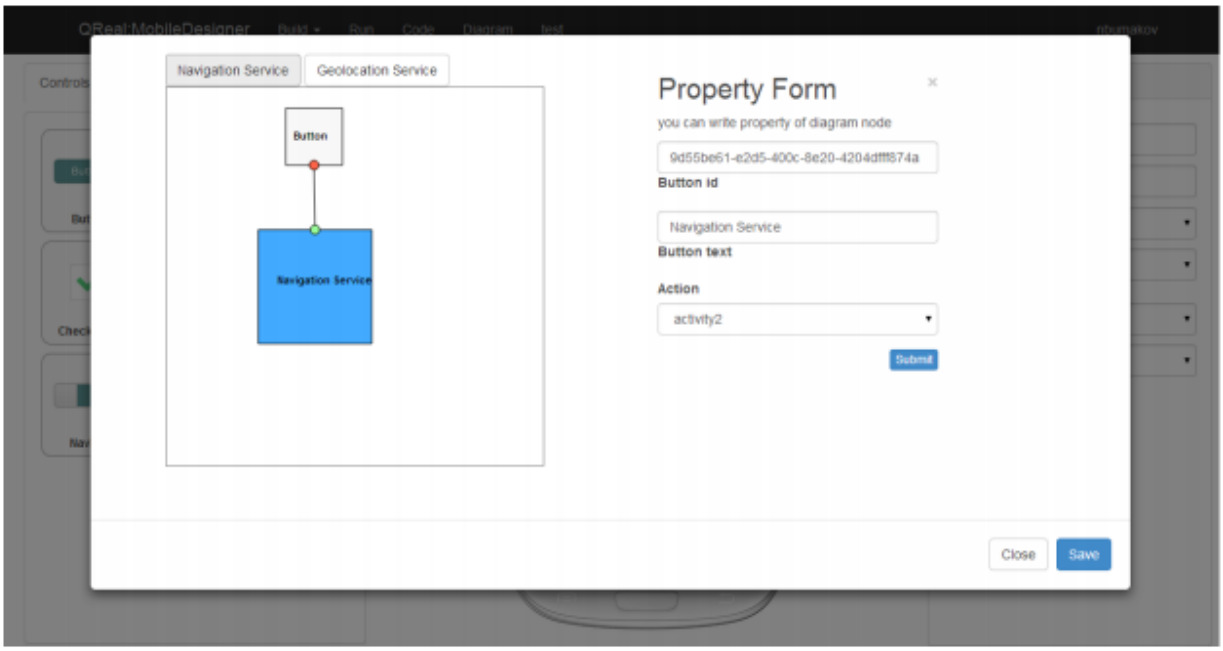

Рис. 8 Пользовательский интерфейс прототипа дизайнера

6. Апробация

Была проведена апробация, целью которой была проверка возможности задания логики мобильного приложения с помощью диаграмм. Для этого была проведена интеграция с констуктором и генератором мобильных приложений.

6.1 Приложение “Визитка”

Простейшим примером приложения является приложение “визитка”. Оно состоит из фиксированного контента и набора кнопок, по нажатию на которую открывается новый экран. Для реализации этой задачи в редакторе диаграмм есть специальный сервис под названием “NavigationService”, представляющий возможность работы с переходами между экранами. Для этого пользователю надо создать этот элемент в редакторе диаграмм и соединить его с кнопкой, а в свойствах сервиса указать экран, на который хотим перейти.

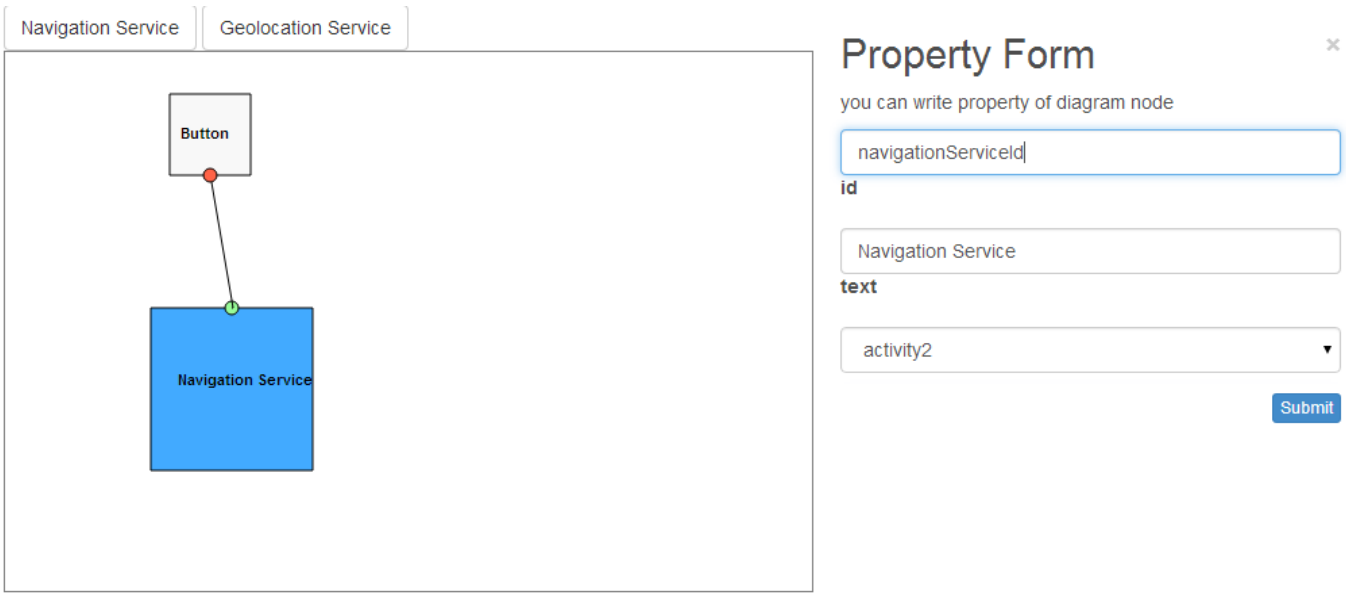

Рис. 9 Схема задания логики перехода между экранами

Получается, что после сохранения диаграммы, в конструктор вернется JSON, изображенный на рисунке 10.

```
“nodes” : [
        {  “id” : “button2”, “type” : “button” , “action” : “click”  }
]
"services": [
        { "id": "navig1”, “type": "Navigation Service", “activity” : “activity3”
        }
    ],
    "links": [
        {
            "source": "button2",
            "target": "navig1"
        }
     ]
```

Рис. 10 JSON, который описывает логику переключения между экранами

Из него видно, что есть кнопка, сервис и стрелочка, которая соединяет кнопку с сервисом. В сервисе же указано, на какой экран надо будет перейти по нажатию на кнопку. Вся эта информация отправляется уже в генератор, который создает нужное нам приложение.

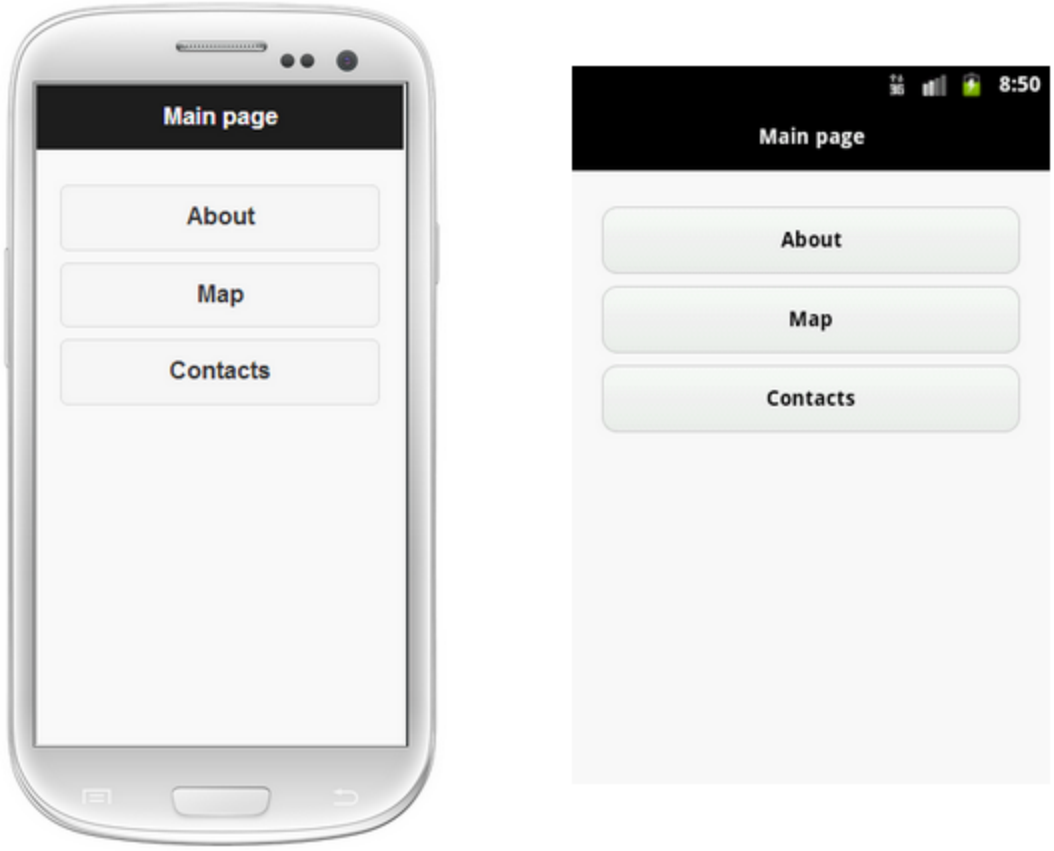

Рис. 11 Основной экран приложения “визитка” (слева - экран дизайнера интерфейса, справа - экран реального устройства)

6.2 Взаимодействие с сенсорами телефона

В качестве примера взаимедействия с сенсорами мобильного приложения было решено создать приложение, показывающее место положение человека на карте по нажатию на кнопку. Для реализации такой логики пользователю в редакторе диаграмм нужно создать “GeolocationService” и на вход ему соединить кнопку, на выход соединить его с картой.

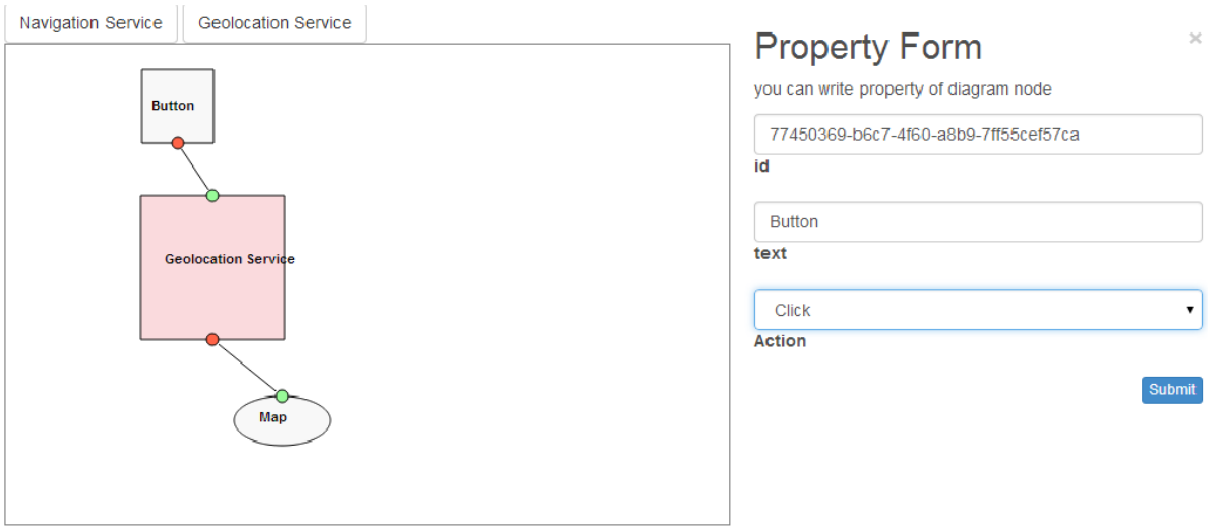

Рис. 12 Схема задания логики приложения для работы с картами

Из этой диграммы получает JSON, изображенный на рисунке 13.

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

Рис. 13 JSON, который описывает логику для работы с картами

В данном случае, нам нужно создать геолокационный сервис, к которой нужно на вход дать кнопку, а на выход карту. Карта и кнопка задаются в конструкторе интерфейса приложения. После этого эта JSON отправляется на генератор приложения.

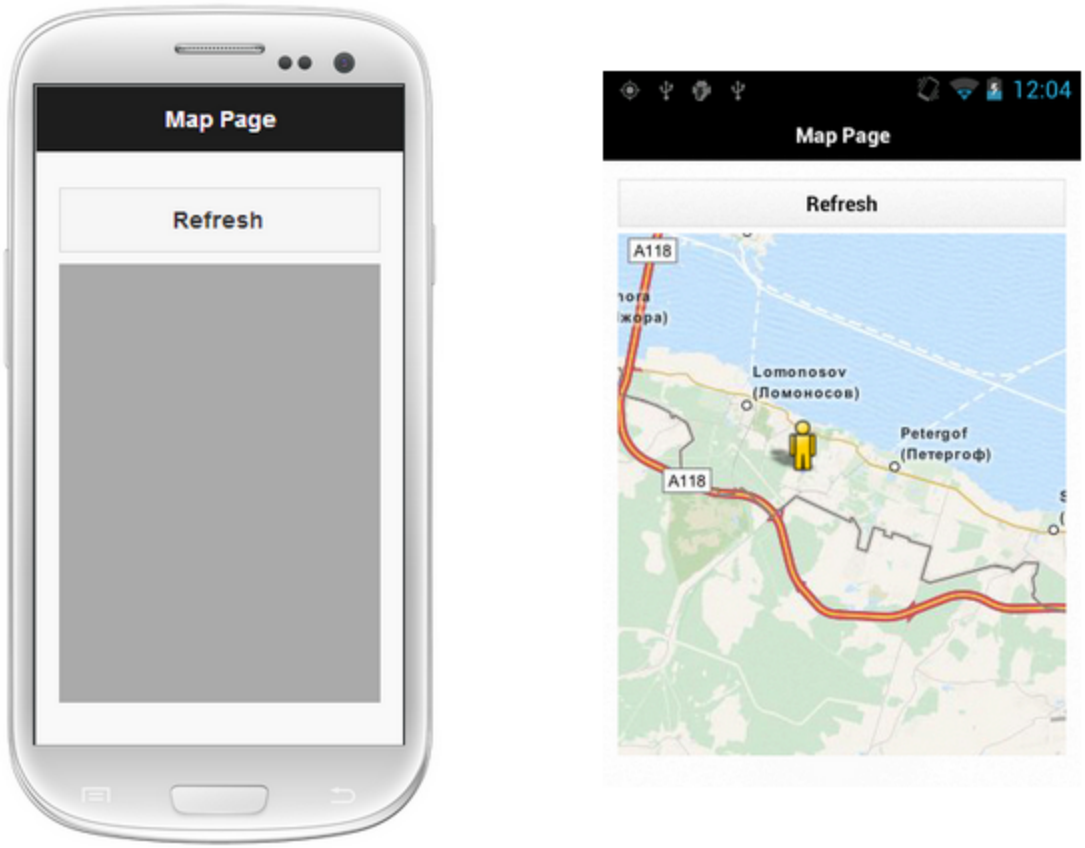

Рис. 14 Приложение, использующее геолокацию (слева - экран дизайнера интерфейса, справа - экран реального устройства)

7. Результаты

В результате проделанной работы были получены следующие результаты:

1. Проанализированы существующие решения редакторов диаграмм.
2. Выбраны подходящие инструменты для создания редактора.
3. Разработана инфраструктура редактора диаграмм со следующей функциональностью:
    a. Поддержка описания логики приложения с помощью диаграмм.
    b. Возможность легкого расширения редактора, то есть простое добавление нужных элементов в наш метаязык.
    c. Возможность импорт и экспорта диаграмм в формате JSON.
    d. Валидация созданой диаграммы “на лету”
4. Были созданы такие компоненты как палитра интструментов, рабочая область и редактор свойств элементов.
5. Проведена апробация редактора на конкретных примерах.

8. Литература

[1] TypeScript Revealed — Dan Maharry, 2013

[2] Приемы объектно-ориентированного программирования. Паттерны проектирования — Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 2010

[3] AngularJS — Brad Green, Shyam Sheshadri, 2013

[4] jQuery для начинающих — Антон Шевчук, 2013

[5] JointJS, URL: http://jointjs.com