Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Малиновский Илья Константинович

Разработка клиента в системе Ubiq Mobile для платформы Firefox OS

Курсовая работа

Научный руководитель:
Ведущий разработчик ООО ”НМТ” Невоструев К. Н.

Санкт-Петербург
2016

Оглавление

Введение

В 2011-2015 годах в Mozilla Foundations была разработана операционная система для мобильных устройств на базе используемого в браузере Mozilla Firefox движка Gecko. Ее основными преимуществами являются:

- низкие требования к аппаратному обеспечению;
- оптимизация работы с ресурсами сети Интернет [1].

Последнее достигается благодаря нетрадиционной для мобильных платформ архитектуре. В отличие от большинства мобильных операционных систем, в которых соединительным звеном между ОС и веб-страницей служит браузер, в разработке Mozilla Foundations браузер с добавленным к нему специфическим для смартфонов API имеет прямой доступ к аппаратному обеспечению устройства. Таким образом, между веб-страницей и "железом" становится на один уровень абстракции меньше.

Прямым следствием первого преимущества является ориентированность Firefox OS на низкобюджетные смартфоны, что делает систему доступной для малообеспеченных слоев населения.

В настоящее время большой популярностью пользуются мобильные интернет-сервисы. Основным критерием их качества являются надежность и эффективность передачи данных. Один из способов доступа к интернет-сервисам – использование браузера – влечет за собой снижение скорости передачи данных. Другой – создание нативного приложения для сервиса – приводит к необходимости заниматься реализацией отдельной версии для каждой из широко распространенных мобильных операционных систем.

Разрабатываемая в Санкт-Петербургском государственном университете система Ubiq Mobile предназначена для решения обеих проблем:

- система ориентирована на использование мобильного интернета, то есть, в частности, медленного и нестабильного соединения;

- приложения создаются в единой среде на базе .NET и функционируют на всех поддерживаемых мобильных ОС.[2]

Практически вся бизнес-логика приложений в системе Ubiq Mobile сосредоточена на стороне сервера. За отображение графического интерфейса приложений на мобильном устройстве и передачу информации о действиях пользователя на сервер отвечает тонкий клиент, взаимодействующий с сервером по оригинальному древовидному байтовому протоколу, надстроенному над стеком TCP/IP. На сегодняшний день в системе Ubiq Mobile реализованы клиенты для следующих платформ:

- iOS;
- Android;
- Windows Phone;
- BlackBerry OS;
- Java ME.

Среди этих платформ нет ни одной операционной системы на HTML5. В свою очередь, такие системы набирают в последние годы все большую популярность: например, можно упомянуть разрабатываемую с 2012 года ОС Tizen или H5OS, первый релиз которой состоялся всего три месяца назад.

Целью данной курсовой работы является разработка нового компонента системы Ubiq Mobile – тонкого клиента для платформы Firefox OS.

Значимость поставленной задачи не только в поддержке новой платформы, но и в упрощении будущей разработки клиентов для других операционных систем на базе HTML5 и для браузеров. Несмотря на различия, связанные с механизмами клиент-серверного взаимодействия и с кроссбраузерной верстkой, основная часть – преобразование дерева управляющих элементов Ubiq Mobile к дереву HTML-тегов – во всех этих случаях ничем не будет отличаться от аналогичной составляющей клиента для Firefox OS.

1. Анализ существующего решения

Несмотря на то что тонких клиентов в системе Ubiq Mobile для операционных систем на базе HTML5 до этой работы не существовало, все клиенты работают с одним и тем же протоколом и с одними и теми же входными данными. В связи с этим целесообразно привести анализ архитектуры клиента для платформы iOS, описанной в дипломной работе К.Н. Невоструева [4].

Главным классом рассматриваемого клиента является *Ubiq Mobile App Delegate*. За передачу данных между ним и сервером отвечают классы *Data Receiver* и *Data Sender*. Отдельный класс, называющийся *Menu Table Delegate*, отвечает за отображение меню. Каждый управляющий элемент (другими словами, секция в протоколе Ubiq Mobile) представлен экземпляром класса-наследника *Ubiq Command*. Входные данные в дерево объектов преобразует класс *Ubiq Parser*. Класс *App Controller* отвечает за вывод дерева управляющих элементов на экран.

Выбор такого решения тесно связан со спецификой используемого инструмента – объектно-ориентированного языка программирования Objective-C. При разработке клиента для ОС на базе HTML5 основным языком становится прототип-ориентированный JavaScript, требующий своего подхода к разработке архитектуры.

2. Реализация

2.1. Интерфейс приложения

По интерфейсу описываемый в этой работе клиент для Firefox OS и упомянутый в первой части клиент для iOS имеют практически полное сходство. Центральным элементом интерфейса клиента для Firefox OS является форма, в верхней панели которой расположены название текущего приложения (либо строка "Ubiq", если ни одно приложение не запущено), кнопка вызова меню настроек и кнопка, предназначенная для открытия экрана со списком доступных приложений. Часть экрана под этой строкой полностью используется для отображения графических объектов. Рисунок 1 демонстрирует это на примере одного из тестовых приложений Ubiq Mobile.

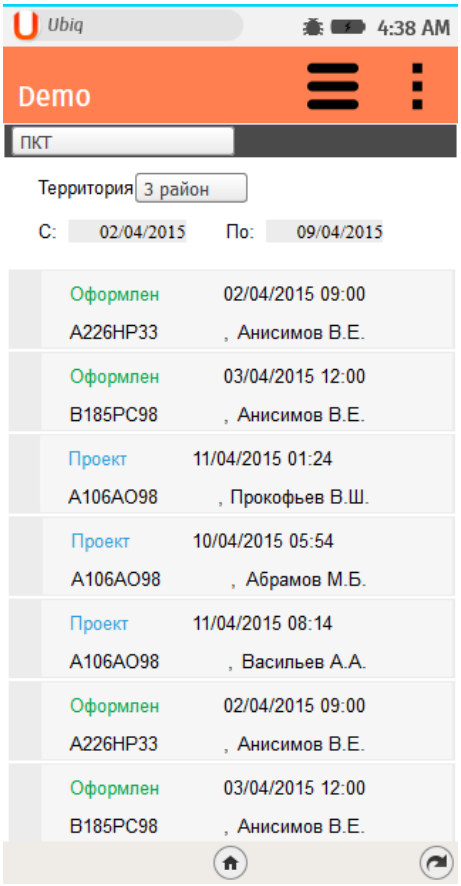

Рис. 1

Меню позволяет установить настройки соединения и отображается поверх основной формы, как изображено на рисунке 2. Наконец, экран

со списком приложений становится доступен, когда соединение с сервером, на котором размещены эти приложения, установлено. Он также находится слоем выше основной формы, но не закрывает ее полностью (см. рисунок 3).

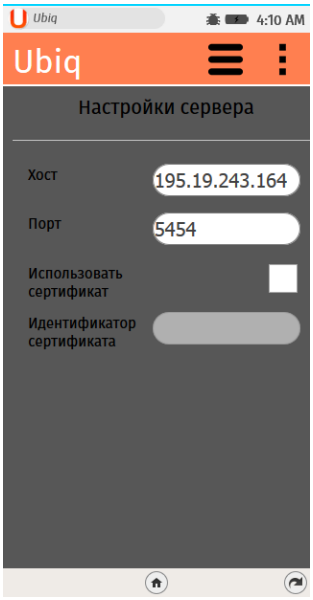

Рис. 2

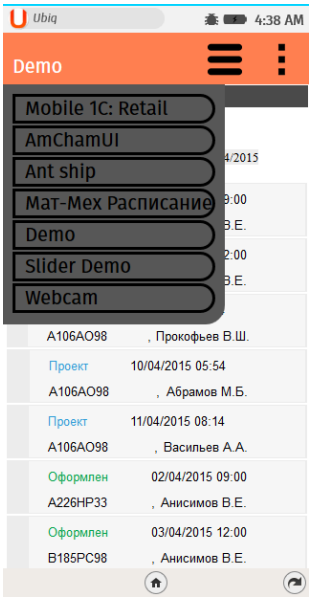

Рис. 3

2.2. Архитектура приложения

2.2.1. Общая схема

Как уже упоминалось в конце первой части данной работы, при проектировании архитектуры этого клиента требуется принимать во внимание специфику используемого средства разработки – языка JavaScript. Кроме того, нельзя не учитывать удобство хранения древовидных структур непосредственно в DOM-модели HTML-документа и их изменения при помощи jQuery. Значит, можно ”на лету” формировать HTML-дерево управляющих элементов и не перегружать приложение набором классов, соответствующих секциям команды передачи графической информации, и хранением избыточной древовидной структуры.

Подключение к серверу по TCP осуществляется при помощи объекта *navigator.mozTCPSocket*. Получаемые от сервера данные приходят по частям, поэтому они накапливаются до тех пор, пока их раз-

мер не станет равным длине передаваемой команды, указанной в ее первых байтах. Как только данные получены, они передаются в функцию *parseServerResponse*. Если пришедшая от сервера команда соответствует описанию элементов на экране (секция *Screen* или *UpdateScreen*), то управление передается в функцию *parseSection*. Результатом вызова этой функции является представление заданного переданной командой дерева управляющих элементов в виде строки в формате HTML. Эта строка затем записывается в свойство $innerHTML$ контейнера, отвечающего за область экрана для отображения графических объектов.

Все секции можно условно подразделить на *терминальные* (соответствующие листовым вершинам дерева управляющих элементов) и *промежуточные* (все остальные). Примерами терминальных секций являются $TextLabel$, $Button$, $TextField$, $CheckBox$, и т.д. В свою очередь, среди нетерминальных секций можно выделить контейнеры, позволяющие расположить внутри себя сразу несколько элементов ($LinearLayout$, $AbsoluteLayout$ и $Grid$), и контейнеры для одиночных дочерних узлов дерева ($Border$, $Selectable$ и $Cell$).

Функция *parseSection* принимает на вход команду или ее часть, отвечающую за передачу графической информации о поддереве дерева управляющих элементов. На первом этапе, общем для всех случаев, определяется, какая секция является корневой для этого поддерева, и извлекается информация о стандартных (т.е. присущих всем секциям) полях в представлении этой секции. Далее управление переходит в функцию, соответствующую корневой секции. Каждая такая функция занимается обработкой специфических для данного управляющего элемента полей. Если этот элемент соответствует терминальной секции, то в вызванной функции формируется его представление в формате HTML, которое затем возвращается в функцию *parseSection*. Если же рассматривается промежуточная секция, то оставшаяся необработанной часть команды разбивается на части, соответствующие дочерним подсекциям текущей, после чего для обработки каждой из дочерних подсекций вызывается функция *parseSection*. Наконец, их представление в формате HTML "вкладывается" (по-разному, в зависимости от ис-

пользуемой промежуточной секции) в описание родительской секции, после чего завершенное описание родительской секции возвращается из обрабатывающей родительскую секцию функции.

Рассматриваемый подход схематично изображен на рисунке 4.

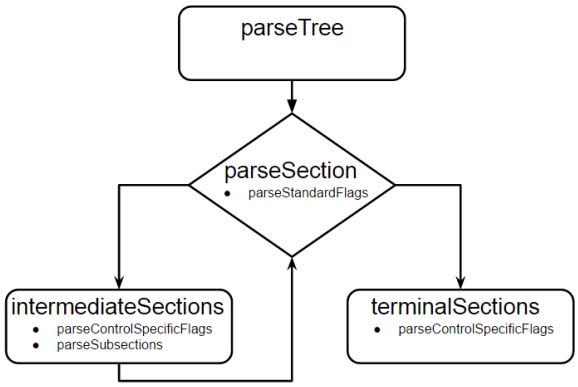

Рис. 4

Таким образом, рекурсивный спуск по дереву, описываемому переданной командой, в предлагаемом решении заменяется несколькими проходами по циклу, образованному вершинами *Parse Section* и *Intermediate Sections*, и завершается однократным переходом в *Terminal Sections* из *Parse Section*.

Тем не менее, эта рекурсивная процедура не позволяет осуществить преобразование отдельных частей переданной команды. О таких случаях речь пойдет в следующих подразделах.

2.2.2. Поддержка интерактивных элементов

Встроить обработчики событий прямо в HTML-представление, генерирующееся при помощи описанной в прошлом подразделе рекурсивной процедуры, невозможно: это нарушает Content Security Policy. По этой причине потребовалось найти обходное решение. В процессе генерации HTML-дерева в отдельном списке сохраняются ID всех элементов, обработчик нажатия на которые требуется добавить. После того, как HTML-представление переданного экрана приложения сформировано и вставлено в соответствующую часть кода клиента, элементы, соответствующие включенным в вышеупомянутый список ID, по очереди извлекаются при помощи функции *document.getElementById*().

Для каждого такого элемента обработчик устанавливается при помощи метода $addEventListener()$ (альтернативы – например, установка свойства $onClick$ вручную – по-прежнему нарушают Content Security Policy). Обработчиком является функция с единственным параметром $event$. Важно отметить, что чтобы извлечь информацию об ID элемента из этого параметра, следует использовать $event.currentTarget.id$, а не $event.target.id$. Первое решение работает всегда, тогда как второе – лишь в тех случаях, когда рассматриваемый элемент не содержит в своем поддереве другие элементы. Если же нажатие производится, например, по секции *Selectable*, то $event.target.id$ будет соответствовать максимально вложенной подсекции, а не этой секции.

2.2.3. Поддержка простых форм

Протокол Ubiq Mobile включает в себя четыре подсекции, соответствующие геометрическим примитивам: линиям, прямоугольникам, эллипсам и многоугольникам. Если прямоугольники и эллипсы можно было бы реализовать при помощи HTML-контейнеров $<div>$, а для имитации линий можно было бы воспользоваться CSS-атрибутом $-webkit-transform$, то для рисования произвольного многоугольника альтернативу уже не подобрать. В качестве универсального решения для рисования примитивов всех четырех типов был выбран $<canvas>$.

Использование $<canvas>$ приводит к необходимости вызывать код на JavaScript для элементов генерирующегося HTML-дерева. По причинам, аналогичным описанным в предыдущей части, сделать это непосредственно в рамках построения дерева нельзя. Таким образом, на этой стадии в соответствующие части дерева помещаются элементы $<canvas>$, а в отдельном списке сохраняются тройки, состоящие из ID элемента $<canvas>$, типа примитива, который требуется нарисовать на этом элементе, и списка дополнительных параметров, определяющих внешний вид примитива. После завершения генерации HTML-дерева для каждого из запомненных элементов при помощи стандартного для работы с $<canvas>$ набора функций изображается соответствующий примитив.

2.2.4. Поддержка изображений

Расположение изображения на экране определяется секцией Image Block, при этом информация о данных изображения может передаваться до дерева управляющих элементов, одновременно с ним или после. Кроме того, единожды переданные данные изображения могут быть использованы для нескольких элементов типа Image Block.

Таким образом, в функции, отвечающей за включение секции Image Block в дерево HTML-элементов, необходимо проверить, было ли уже получено изображение с требуемым ID, и если да, то преобразовать его содержимое в кодировку base64 и записать результат в значение атрибута *src* с соответствующим (зависящим от формата изображения) префиксом. В то же время в функции обработки подсекции Images следует узнать, не требовалось ли какому-то элементу, соответствующему секции ImageBlock, изображение, ID которого совпадало бы с переданным. Для первой проверки требуется поддерживать ассоциативный массив, отображающий ID изображения в его содержимое, а для второй – множество ID всех Image Block, в которые при создании не были установлены изображения.

2.3. Соответствие классов протокола и их полей HTML-тегам и CSS-атрибутам

Часть классов протокола Ubiq Mobile имеет близкие аналоги в HTML:

- UbiqNBImageBlock – $<img>$;
- UbiqNBTextLabel – $<font>$;
- UbiqNBDropBox – $<select>$;
- UbiqNBButton – $<input\ type{=}button>$;
- UbiqNBTextField – $<input\ type{=}text|number|password>$;
- UbiqNBCheckBox – $<input\ type{=}checkbox>$.

Аналоги оставшихся классов требовалось получить на основе комбинаций тегов и их атрибутов. Так, секции UbiqNBBorder, UbiqNBCell и UbiqNBSelectable соответствуют одному и тому же HTML-элементу – $<div>$, но для Selectable требуется добавить обработчик нажатия, для Border – рамку заданной толщины, а для Cell – ничего из вышеуказанного.

Секции UbiqNBList тоже соответствует $<div>$, но на этот раз с CSS-атрибутами $display{:}flex; flex\text{-}direction{:}row|column$. Аналогичным образом реализована поддержка секции UbiqNBLinearLayout.

Секция UbiqNBAbsoluteLayout снова не что иное, как обычный $<div>$, но со значением *relative* атрибута *position*. Для всех дочерних элементов тот же атрибут должно иметь значение *absolute*.

Наиболее простой способ реализовать секцию UbiqNBGrid – воспользоваться стандартными HTML-таблицами, т.е. тегами $<table>$, $<tr>$ и $<td>$.

Наконец, UbiqNBImageButton получается как композиция двух тегов: внутрь $<button>$ следует поместить $<img>$, обработчик нажатия назначить только на кнопку, но при этом менять при необходимости внутри обработчика значение свойства *src* у вложенного внутрь $<button>$ изображения.

Прямые аналоги существуют и не для всех стандартных флагов. Поля *horizontalAlignment* и *verticalAlignment* в соответствии с протоколом могут принимать по три значения, соответствующих выравниванию по центру и по краям для каждой из осей. Тем не менее, в CSS горизонтальное и вертикальное выравнивание устроены по-разному. Более того, способ выравнивания вдоль одной и той же оси зависит от типа элемента. Для горизонтального выравнивания "inline" элементов [3], достаточно использования атрибута *text-align*, тогда как для размещения "block-level" элементов (например, многократно используемых элементов типа $<div>$) вдоль той же оси требуется задавать значение атрибутов *margin-left* и *margin-right* (0 и *auto*, соответственно, для выравнимания по правому краю, *auto* и *auto* для размещения по центру, *auto* и 0 для выравнивания по правому краю). В то же время

значения атрибутов $margin$ согласно протоколу могут задаваться отдельно. Таким образом, теоретчески возможна ситуация, при которой можно будет отобразить HTML-элемент с верными значениями атрибутов только одного из типов: $margin$ и $align$ (в существующих на сегодняшний день приложениях не возникает ситуации, где атрибуты обоих типов одновременно имеют значения, отличающиеся от используемых по умолчанию, но протокол это не запрещает).

Заключение

В рамках данной курсовой работы был разработан тонкий клиент в системе Ubiq Mobile для платформы Firefox OS, поддерживающий терминальный протокол для NativeBasic-модели обмена данными.

Продолжением этой работы может стать оформление функций, отвечающих за построение дерева HTML-элементов, в виде отдельной библиотеки и реализация на основе этой библиотеки клиентов для других операционных систем на HTML5, а также эмулятора клиента системы Ubiq Mobile в веб-среде.

Список литературы

[1] Jongboom Jan. Firefox OS in Action. — 2015. — 121 p. — URL: https://leanpub.com/firefoxosinaction (online; accessed: 20.05.2016).

[2] Ubiq Mobile. — 2016. — URL: http://ubiqmobile.com/ru/разработчикам/ (дата обращения: 20.05.2016).

[3] developer.mozilla.org. — 2016. — URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Inline_elements (online; accessed: 20.05.2016).

[4] К.Н. Невоструев. Разработка и реализация терминального протокола и клиента в системе Ubiq Mobile для платформы iOS // Дипломная работа, Санкт-Петербургский государственный университет. — 2012.