Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Разработка общей схемы обработки разрывов мобильных соединений в системе Ubiq Mobile

Курсовая работа студента 444 группы

Ефимова Глеба Дмитриевича

Научный руководитель В. В. Оносовский

Санкт-Петербург

2014

Оглавление

1. Введение

В настоящее время сфера мобильных технологий находится в стадии активного роста: постоянно появляются устройства с уникальными особенностями, новые платформы и средства разработки под них. Все это многообразие технологий покрывается весьма трудоемко, разработчики вынуждены реализовывать одну и ту же логику под каждую из мобильных платформ, учитывая их особенности, что требует большого количества специалистов с высоким уровнем квалификации в соответствующей платформе.

Для решения проблемы фрагментации существует два подхода кроссплатформенной разработки. Первый заключается в использовании фреймворков, которые могут генерировать платформозависимый код. Примерами такого подхода являются Xamarin, PhoneGap Второй подход заключается в использовании клиент-серверной системы, где разработаны тонкие клиенты под все мобильные платформы и единый сервер, координирующий работу пользователей. Примером такого подхода являет сервис Amazon Appstream, который позволяет транслировать приложения на мобильные устройства, используя мощности облачных вычислительных ресурсов.

На математико-механическом факультете в течение последних лет разрабатывается платформа Ubiq Mobile, использующая клиент-серверный подход. При передаче информации используется оригинальный протокол, расположенный поверх транспортного уровня в модели OSI. Это обстоятельство в совокупности с компактность. передаваемых данных предоставляет обширные возможности оптимизации сетевого трафика, эффективное использование ресурсов и быстроту работы в медленных сетях, в сравнении с другими подходами.

Система Ubiq Mobile позволяет реализовывать приложения, плюсами которых являются скорость и простота разработки, при этом максимально используя возможности конкретных мобильных платформ.

2. Обзор системы Ubiq Mobile

2.1 Протокол

В основе соединения сервера и клиентов лежит протокол типа «атрибуты и структуры». Терминальный сервер и клиент, могут принимать и отправлять команды. Каждая команда протокола имеет древовидную структуру, элементами которой являются секции с некоторым набором полей или атрибутов. Секции представляют собой непрерывные

Main section length	Main section code	Client ID	Connection ID	Subscetions
3 bytes	1 byte	4 bytes	4 bytes	variable

Section length	Section code	Section data	Subsections
3 bytes	1 byte	variable	variable

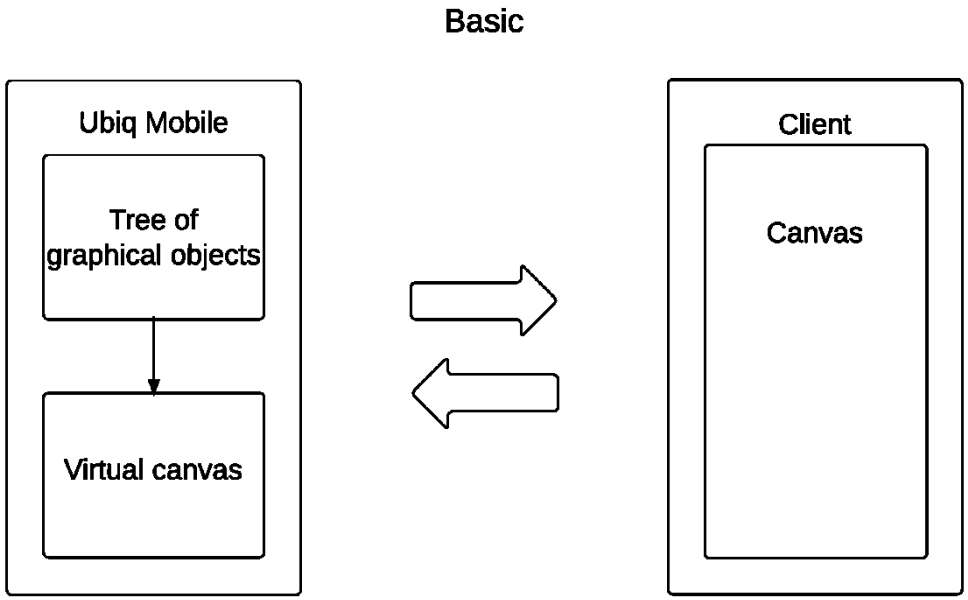
типа. К полям фиксированной длины относятся числа, символы, дата, время. Примером полей переменной длины служат массивы, изображения, строки. Все числа представляются в little-endian формате – т.е. в начале идут младшие байты

2.2 Модель интерфейса

В платформе принято различать два вида клиентов. К первому типу относятся клиенты, на платформах которых отсутствуют обширные графические возможности. Примерами таких платформ являются телефоны с поддержкой Java ME, в частности Nokia Series 40, Nokia Asha. Второй тип клиентских устройств обладает развитыми графическими возможностями и достаточным количеством вычислительной мощности. К таким платформам относят iOS, Android, Windows Phone, Tizen, Ubuntu Mobile, Sailfish OS и другие.

2.2.1 Модель интерфейса Basic

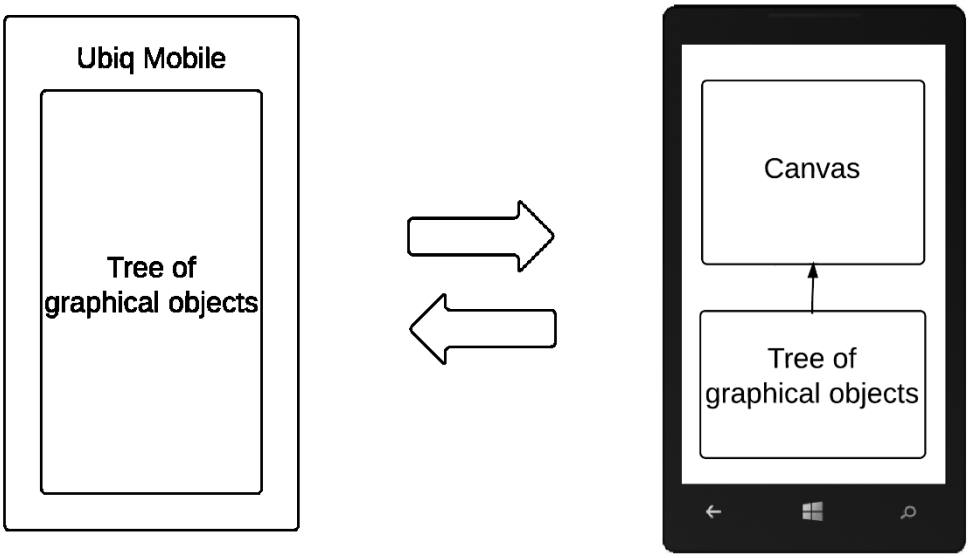
Для первого типа клиентов используется модель Basic. Она предполагает использование так называемого «виртуального холста», он располагается на сервере и используется для предварительной прорисовки дерева графических объектов отправляемой команды, затем полученная структура сжимается и отправляется клиентскому приложению, в котором достаточно быстро отображается. Такая модель прекрасно подходит для устройств со слабыми графическими возможности. Использование данной модели на более технологичных устройствах влечет за собой потерю качества, которое может быть



2.2.2 Модель интерфейса NativeBasic

Для второго типа клиентов используется модель NativeBasic. В данном случае дерево графических объектов строится на сервере и отправляется командой на клиент, где производится обработка и отображение дерева непосредственно на экран. Клиент сам решает, как должен выглядеть элементы пользовательского интерфейса, используя для этого мощности устройства и возможности графической подсистемы. Приложения, работающие в модели NativeBasic удобны в использовании на любых разрешениях экрана и близки по своему визуальному представлению к нативным приложениям, так как используются управляющие элементы, предоставленные целевой платформой.

NativeBasic



3. Постановка задачи

Целью работы является разработка общей схемы работы с разрывами сети при передаче данных между клиентами и сервером. Реализация и тестирование сценариев должны производиться на клиенте под управлением платформы Windows Phone. Таким образом, работа включает в себя реализацию клиента под данную операционную систему для терминальной платформы Ubiq Mobile и использовать модель NativeBasic.

Основные требования к программе-клиенту:

- Взаимодействие с терминальным сервером по протоколу Ubiq Mobile
- Использование визуальных элементов интерфейса, предлагаемых платформой
- Достаточная производительность
- Стабильная работа при низкокачественном соединении

Используя реализованный клиент, требуется провести анализ ситуаций, ведущих к разрывам соединения, и классифицировать их. На основе полученной классификации разработать общую схему обработки разрывов для всех клиентских платформ и реализовать полученную стратегию в Windows Phone клиенте.

4. Архитектура клиента

В силу некоторых различий между ранними и поздними версиями операционной системы Windows Phone, структура проекта разделена на отдельные сборки для каждой версии платформы. Основная разработка ведется для старшей версии, исходные файлы остальных модулей ссылаются на файлы в ней. При помощи условной компиляции происходит разделение платформозависимого кода.

```
14  #if WP8
15  using Client.WP80.Services;
16  #endif
17
18  #if WP7
19  using Client.WP71.Services;
20  #endif
```

Таким образом платформенные зависимости разрешаются на этапе сборки приложения. В структуре проекта присутствует ресурсная сборка, в ней расположены строки локализации, файлы изображений и прочие ресурсы клиента.

Архитектура клиента строится по приятому платформой паттерну Model-View-ViewModel (модель – представление – представление модели). Это некоторая модификация шаблона проектирования Presentation Model, предназначенная для технологий Windows Presentation Foundation и Silverlight от компании Microsoft. Компонент Model аналогичен Model в паттерне Model-View-Control, он представляет собой структуры данных и их взаимодействие, сущности и связи между ними. Часть View отвечает за представление информации перед пользователем. Представление является подписчиком на источник информации, которую она предоставляет, в случае изменения исходных данных изменяется и представление. Компонент ViewModel является прослойкой между Model и View, агрегируя в себе все необходимые для пользователя сущности модели. Представление модели является издателем событий изменения модели и слушателем изменений представления. В первом случае она должна уведомлять об изменении пользовательский интерфейс, а во втором дает модели соответствующие команды, основываясь на действиях пользователя. Таким образом компоненты View и ViewModel относительно просты, а основная логика сосредоточена в Model.

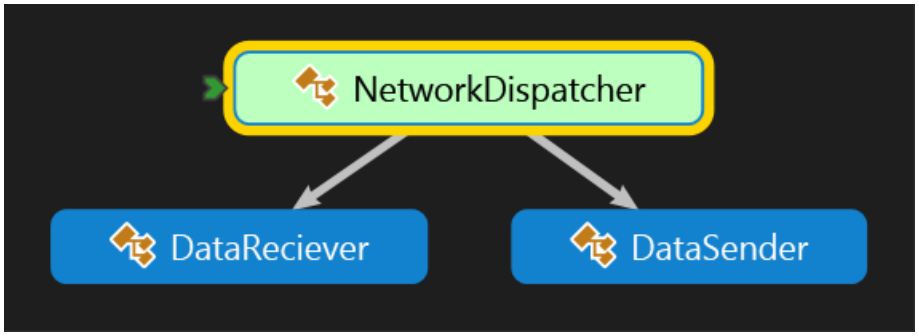
В архитектуре компоненты Model выделены три основных составляющих:

- Работа с сетью
- Реализация протокола
- Центр управления

Каждая составляющая всегда выполняет свою основную задачу в отдельном потоке, таким образом тяжеловесные задачи отправки или получения данных, обработки входящих команд выполняются параллельно, держа основной поток исполнения свободным для действий пользователя.

4.1 Работа с сетью

Компонента работы с сетью состоит из сетевого диспетчера (NetworkDispatcher), приемника (DataReceiver) и передатчика (DataSender), все вместе они решают задачи создания и поддержки соединения, выполнения политики работы с разрывами, приема и передачи сообщений. Сетевой диспетчер следит за соединением, получает уведомления приемника и ставит сообщения в очередь на отправку передатчика, уведомляет ядро клиента о наличии новых сообщений. DataSender и DataReceiver выполняют задачи сетевого ввода и вывода, которые требуют много времени и являются критическим звеном для производительности и отзывчивости приложения, поэтому выполняются в отдельных потоках.

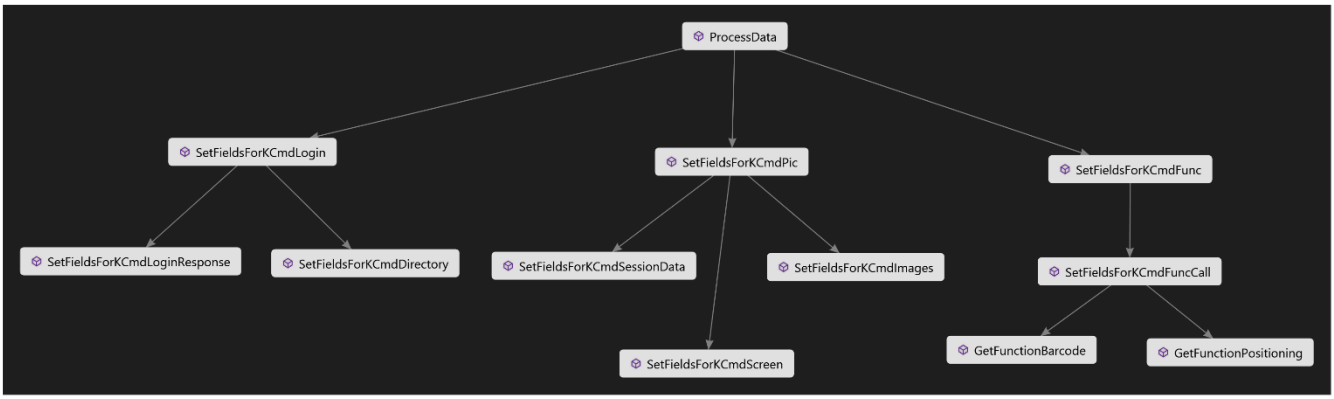


4.2 Клиентская реализация протокола

Компонента протокола решает задачи чтения и инициации исполнения команд, а также их создания. Она инкапсулирует в себе протокол Ubiq Mobile, предоставляя пользователю (в данном случае ядру клиента) интерфейс работы с командами.

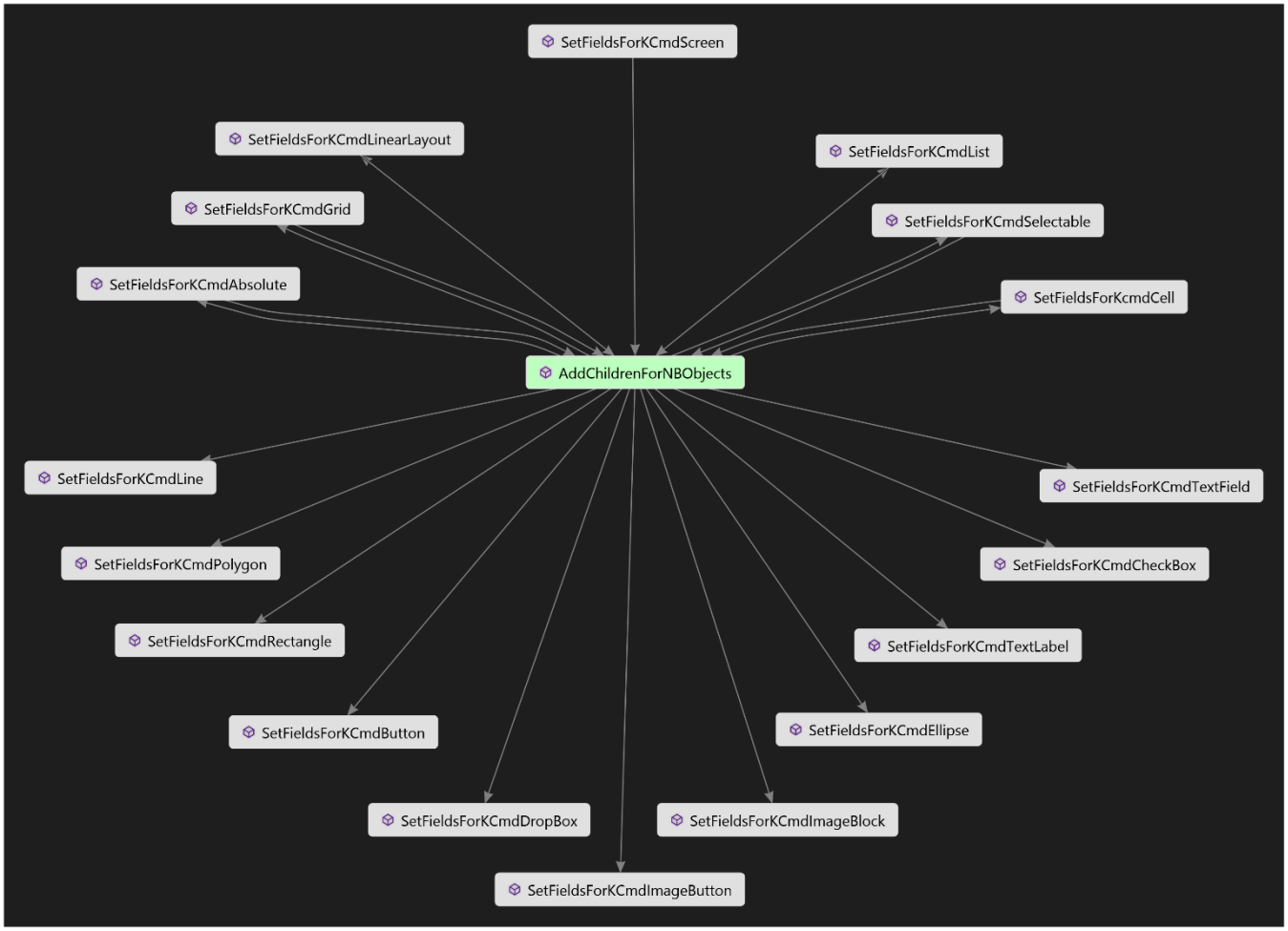
4.2.1 Обработка входящих соединений

При получении нового сообщения необходимо перевести сообщение из бинарной формы в команду протокола. Разбор сообщения осуществляется последовательным посекционным чтением данных из сообщения, что является обходом дерева в глубину. В процессе чтения строится дерево управляющих блоков-команд.



Когда разбор сообщения произведен, полученные команды в древовидном виде готовы к исполнению, это могут быть команды авторизации управления состоянием клиента, передача графической информации или вызов специальных функций на клиенте.

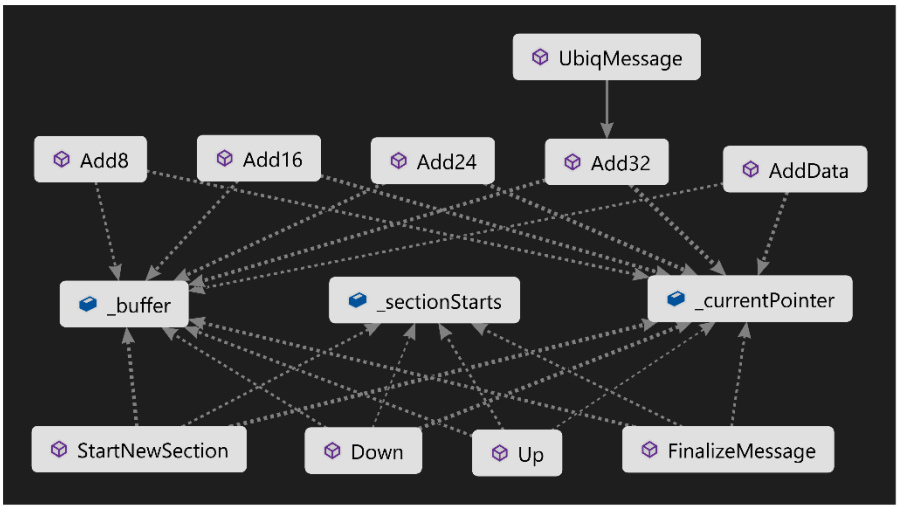
Подсекции в команде KCmdScreen представляют собой дерево управляющих элементов в модели Native Basic, а сама команда - корень. При разборе каждому управляющему элементу модели сопоставляется управляющий элемент платформы. Все визуальные элементы в протоколе делятся на две категории, контейнеры и управляющие элементы. На схеме представлено ключевое различие между ними: элементы первого типа, представленные методами в верхней половине, могут быть как «родителями», так и «детьми» в дереве, а второго типа, представленные методами, расположенными в нижней половине схемы, могут выступать только в роли «детей».



Разобранная команда KCmdScreen является прообразом визуального дерева экрана, который требуется отобразить.

4.2.2 Обработка исходящих сообщений

Отправляемые сообщения строятся по тем же принципам, что и входящие. Последовательно формируется древовидная структура команд, в полях которых содержатся данные. Класс UbiqMessage предоставляет интерфейс для создания любых типов сообщений.



4.2.3 Отображение дерева графических объектов.

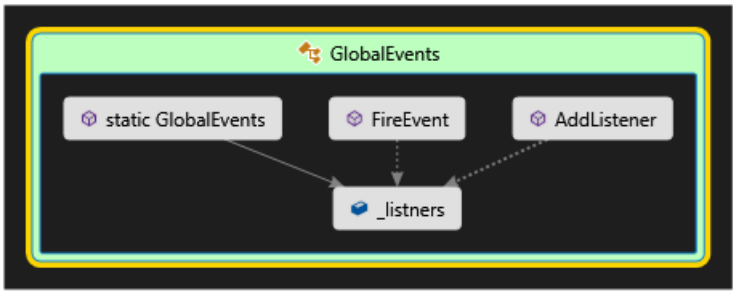
В клиенте реализована иерархия типов NativeBasic, все классы имеют одного общего предка – UbiqNBObject, он содержит в себе поле флагов полей, которое показывает наличие или отсутствие атрибутов, и сами атрибуты. Остальные классы являются наследниками главного предка и в дополнение к общим флагам имеют специфичные для типа флаги наличия или отсутствия полей. В процессе создания дерева нативных графических элементов используется конвейер:

1. Инициализация контрола, заполнение нужных полей, затем инициализация его детей
2. Измерение элементов, уточнение размеров детей и расположение в родителях
3. Прорисовка дочерних управляющих элементов, создание самого контрола и его передача родительскому элементу

Процесс перевода дерева из модели NativeBasic в интерфейс плафтормы реализован с помощью паттерна FactoryMethod. Благодаря такому подходу, NB элементы можно легко и быстро модифицировать без риска оказаться в ситуации, требующей рефакторинга или реинжиниринга.

4.3 Центр управления

Составляющая центра управления координирует работу клиента и его составляющих, отвечает за операционную деятельность: сохранение и синхронизация внутренних настроек, кэширование графических файлов, использование необходимых сервисов и обработка данных, полученных от компонент протокола и сети. Так же она инициирует создание исходящих сообщений. Когда пользователь совершает действие, которое требует новой информации, центр генерирует событие, на которое подписаны заинтересованные слушатели – обработчики, инициирующие создание и отправку сообщения.



Сервисы в клиенте построены по принципу шаблона проектирования Singleton: в клиенте присутствует один экземпляр каждого сервиса, появляясь по мере необходимости, так называемая «ленивая» инициализация.

```
11          private static readonly Lazy<LocationService> service =
12              new Lazy<LocationService>(
13                  () => new LocationService(),
14                  LazyThreadSafetyMode.ExecutionAndPublication);
15
16          public static LocationService Instance
17          {
18              get
19              {
20                  return service.Value;
21              }
22          }
```

5. Анализ разрывов соединения

Перед проведением тестирования работы в неустойчивых соединениях необходимо выделить внешние факторы событий, при которых происходит обрыв. Было выделено четыре теста, которые покрывали большинство типов ситуаций:

- Выход из зоны Wi-Fi и переход в зону Cellular
- Выход из зоны Wi-Fi в зону без связи
- Переход из зоны со связью в зону с отсутствием связи или очень плохим качеством (например, лифт)
- Пропадание сетевого соединения (проблема провайдера)

Тестирование проводится при подключенном клиенте на двух сервисах:

- Ant Ship (Морской бой)
- Webcam (Веб-камера)

Выбор сервисов обоснован различиями в типах трафика, который они генерируют. В морском бое идет равноценный обмен сообщениями с сервером (прием - отправка), во втором случае трафик идет в основном одну сторону (условный видеопоток от сервера). Таким образом в тестировании первого сервиса участвует три стороны: Windows Phone клиент, сервер Ubiq Mobile и любой другой клиент, симулирующий противника. В приложении веб-камеры участвует также клиент Windows Phone, сервер Ubiq Mobile и источник видеопотока.

По результатам анализа были выделены две группы обрывов:

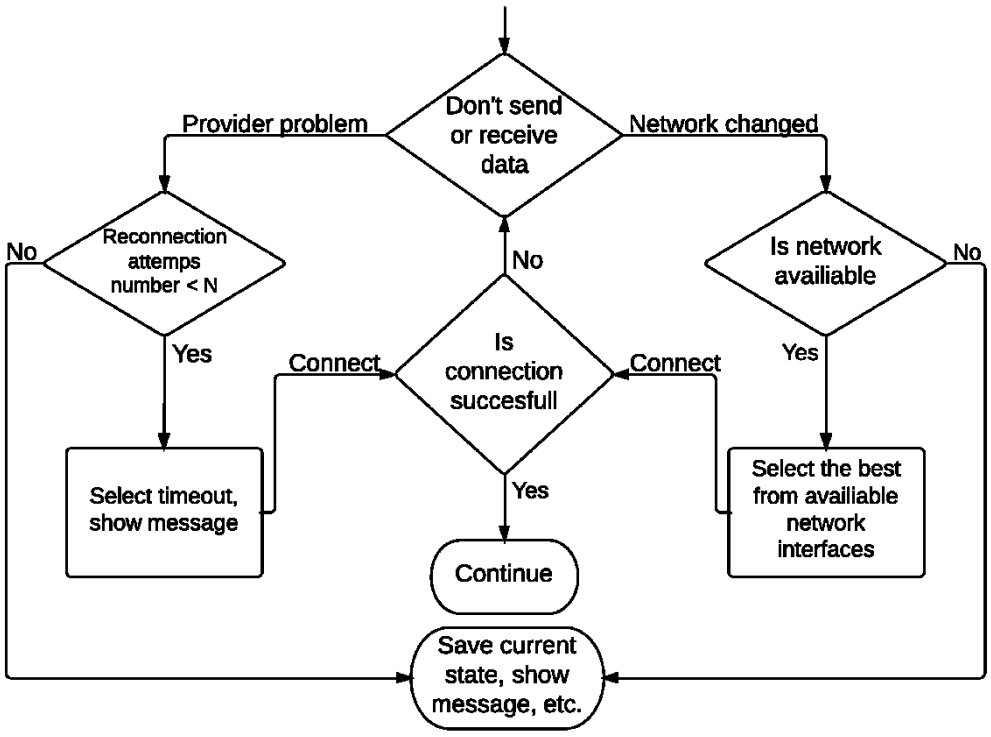
- Связанные с изменением сетевых интерфейса (отключение старого/подключение нового, более качественного)
- Связанные с проблемами на стороне провайдера или падением сервера

Первая категория определяется с помощью встроенного в платформу API работы с сетью, достаточно добавить функцию обработчик на событие NetworkAvailabilityChanged в классе DeviceNetworkInformation. Таким образом в момент изменения какого-либо сетевого интерфейса становится понятно, что произошло и какие возможности для продолжения есть. Учитывая специфику поставленной задачи, было проведено исследование других платформ на наличие аналогичных возможностей. В системах iOS и Android похожие API были обнаружены, следовательно, определение изменений интерфейса возможно на большинстве клиентских платформ в модели NativeBasic.

Определение обрывов второй категории происходит менее очевидным образом. Возможны два варианта развития событий: при отправке сообщения или при его получении. В первом случае возникает timeout операции, по которому можно определить, что сервер недоступен из-за каких-либо причин. А при получении сокет находится в «спящем» прослушивающем состоянии, ожидая получения данных, но если произошел обрыв, никаких данных до клиента не дойдет, следовательно, сокет будет бесконечно долго ожидать. Обрыв при получении данных обнаруживается двумя способами. Первый заключается в том, что удаленный сокет может послать сообщение о том, что он закрывается и тогда статус нашего сокета изменится, разрыв будет определен. Второй способ предполагает вызывать по таймеру некоторую функцию, которая проверяет есть ли возможность что-нибудь записать в этот сокет, в случае отсутствия возможности определяется разрыв.

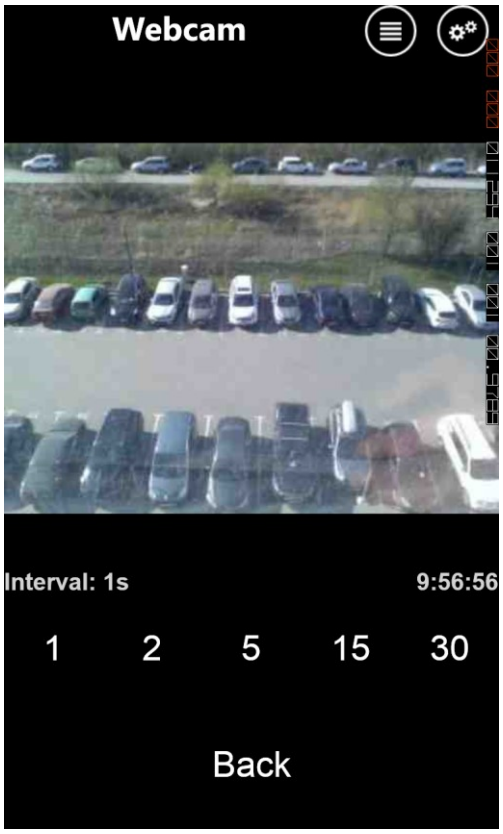
6. Разработка схемы

Используя проведенный анализ, позволяющий определить причины и следствия разрывов разных типов, можно построить общую блок-схему процесса восстановления соединения после обрыва. Сначала определяется причина, по которой данные не передаются. Затем, в случае изменения сетевого интерфейса происходит проверка доступных соединений, через которые можно соединиться с сервером. Если такие есть, последовательно перебираются каналы в порядке уменьшения качественных характеристик. При удачном подключении продолжается работа, иначе происходит возврат в начало, к определению причины отсутствия соединения. Если после перебора всех доступных соединений подключиться не удалось, то сохраняется текущее состояние экрана, очереди сообщений на отправку и выводится сообщение об отсутствии подключения. В случае если есть соединение, но сервер недоступен, происходят попытки переподключения, число которых ограничено и выбирается эмпирическим путем, в зависимости от платформы. Перед подключением делается пауза на время, увеличивающееся с каждой неудачно попыткой, и отображение сообщения о происходящем. В случае успеха число попыток и время тайм-аута обнуляется, затем продолжается работа. Если соединение снова отсутствует, происходит возврат в начало, к определению причины обрыва. По исчерпанию числа попыток схема завершается аналогично первому случаю: сохраняется экран, очередь сообщений и выводится



7. Апробация

Проверка схемы проводилась также на Windows Phone платформе в разнообразных ситуациях. Схема дала нужный результат – при наличии рабочего соединения, переподключение клиента происходило успешно, иначе, после нескольких попыток, клиент уведомлял о невозможности подключения, что говорит о правильности выбора обобщенных ситуаций. Апробация проводилась так же, как и анализ обрывов, в приложениях AntShip и Webcam.



Использование клиента в «полевых» условиях подтвердило непрерывность работы в следующих ситуациях:

- Wi-Fi ⇔ Cellular
- Cellular ⇔ Cellular (смена соты)
- Восстановление из труднодоступных зон (лифт, подвал и т.д.)

8. Заключение

8.1 Результаты

В данной курсовой работе была разработана схема переподключения при разрывах сетевого соединения, универсальная для всех клиентов модели NativeBasic. Её клиентская реализация повышает работоспособность приложения и повышает удобство использования. В ходе выполнения работы был реализован и протестирован Windows Phone клиент для платформы Ubiq Mobile, который отвечает поставленным требованиям производительности и стабильности работы.

8.2 Планы

Первостепенной задачей станет реализация и обкатка полученной схемы на платформах iOS и Android: необходимо добавить в текущие реализации клиентов логику работы с обрывами. Затем будет необходимо поддержать разрывы соединения со стороны сервера, что позволит более качественно в любой момент времени проводить восстановление последней работы. Тут также появляются вопросы безопасности повторных подключений: необходимо убедиться в подлинности клиента, которому планируется отправлять данные. Чтобы внести изменения подобного рода, необходимо модифицировать этот протокол: добавить некоторые поля и реализовать их обработку на клиентах и сервере.

Список литературы

1. Спецификация формата и структуры терминального протокола Ubiq Mobile
2. Implementing the Model-View-ViewModel Pattern
   http://msdn.microsoft.com/en-us/library/ff798384.aspx
3. Design Patterns: Elements of Reusable Object-Oriented Software
   Ralph Johnson, John Vlissides, Richard Helm, Erich Gamma
   ISBN 9780321700698
4. Windows Phone Develop
   http://dev.windowsphone.com/en-us/develop