Слои для Браузера Дополненной Реальности.

Баулин И.Н.

***Аннотация*— В данной статье изложены результаты квалификационной работы, выполненной автором в Лаборатории ОИТ во время обучения в магистратуре на факультете ВМК МГУ им. М.В. Ломоносова. Рассматривается практическая задача представления стороннего контента в браузере дополненной реальности. Основная ценность работы состоит не только в создании конкретного полезного сервиса, но и в предоставлении инструментов для решения подобного рода задач.**

***Ключевые слова*—дополненная реальность, фото-хостинг, мэшап.**

I. Введение

Понятие дополненной реальности представляет собой одну из разновидностей виртуальной реальности. В дополненной реальности реальные объекты в реальном окружении тем или иным образом дополняются виртуальными элементами, образуя, таким образом, комбинированную картинку. Рональд Азума в своей классической статье «A Survey of Augmented Reality» выделяет основные признаки систем дополненной реальности, в частности, подчеркивая, что эти системы должны совмещать в себе три основных признака – совмещение виртуальных и реальных объектов, формирование комбинированной картинки в реальном времени, формирование картинки в 3D [1].

Если раньше для прикладных реализаций понятия дополненной реальности использовались достаточно сложные аппаратные средства, то сейчас, с развитием технологий, начала получать широкое распространение идея создание приложений с дополненной реальностью для мобильных устройств.

Существуют разные подходы для реализации подхода дополненной реальности в мобильных устройствах. Некоторые приложения пытаются анализировать каждый кадр видеопотока, поступающего с камеры смартфона и дополнять его виртуальными объектами. Другой подход – получать информацию о местоположении пользователя и направлении камеры его мобильного устройства посредством встроенного в мобильное устройство GPS приемника и G-сенсора. Именно такой подход реализован в одном из наиболее популярных приложений для мобильных устройств – браузере дополненной реальности Layar [2].

Layar – представляет собой разработанное в Голландии приложение под платформы Android и iOS (с недавнего времени добавлена поддержка Symbian). Layar представляет собой браузер дополненной реальности, позволяющий дополнять картинку с камеры мобильного телефона виртуальным содержимым. Сам клиент Layar – это всего лишь оболочка, инкапсулирующая в себе работу с аппаратурой устройства (GPS, G – сенсор и т.д.) Архитектура всей системы построена таким образом, что контент для браузера извлекается из внешних источников и предоставляется сторонними разработчиками. При этом у пользователя есть богатый выбор возможного контента. Различный контент от различных разработчиков предоставлен в виде, так называемых, слоев (layer).

Приложение может работать в нескольких режимах, самый интересный из которых – режим дополненной реальности (изображен на рисунке 1).

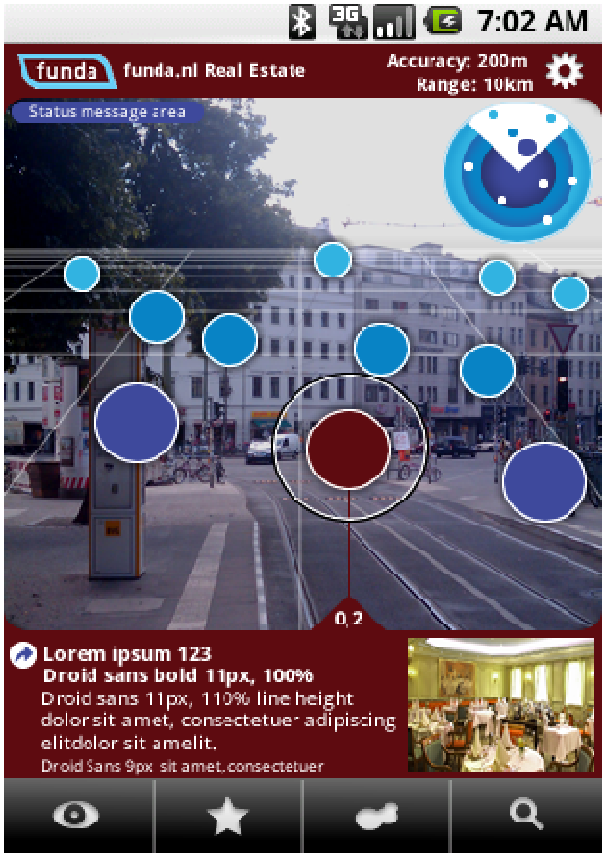

Рис. 1 AR браузер

На этом рисунке картинка, получаемая с камеры мобильного устройства, дополняется виртуальными элементами, связанные с конкретным слоем. Каждый слой содержит свою собственную информацию. В

Статья получена 20 сентября 2013.

Баулин И.Н. – выпускник магистратуры факультета Вычислительной Математики и Кибернетики Московского государственного университета им. Ломоносова.

самом простом случае это может быть слой, содержащий, к примеру, информацию обо всех ресторанах итальянской кухни в окрестности пользователя. Вся информация привязана к так называемым точкам интереса, которые изображены на рисунке в виде кругов разного размера и связаны с конкретным географическим положением. Массив точек интереса, доступных в данный момент пользователю, определяется географическими координатами, получаемыми с GPS устройства, направлением и углом наклона камеры. Размер круга определяется расстоянием от пользователя до точки интереса.

Каждую точку интереса и список действий с ней связанных определяет и настраивает разработчик слоя, к которому привязана данная точка. Таким образом, слой дополненной реальности можно определить как совокупность всех точек интереса данного слоя и некоторой статической метаинформации, связанной с данным слоем. При этом статическая метаинформация постоянно хранится на серверах самого сервиса Layar, а динамическая информация о массиве точек интереса получается в реальном времени от разработчика слоя (подробнее об этом будет рассказано ниже при описании архитектуры системы).

Целью данной работы было создание слоя для браузера Layar, дополняющего получаемую с камеры мобильного устройства картинку фотографиями старой Москвы. То есть пользователь, выбравший соответствующий слой в каталоге слоев, при наведении камеры своего смартфона на какой-то географический или архитектурный объект (здание, площадь, парк и т.п.) должен получить возможность увидеть фотографии этого объекта (или того, что было на его месте) в прошлом. При этом одной из целей было создание такой системы, которая позволила бы отказаться от централизованного опубликования фотографий одним человеком (некоторым контент провайдером), а вместо этого дать возможность конечным пользователям системы расширять базу архивных фотографий, активно используя свои собственные источники. Таким образом, на выходе хотелось получить создаваемый пользователями контент (user generated content – UGC) для данного слоя дополненной реальности. При этом все дополнения, вводимые множеством пользователей в базу архивных фотографий, должны быть сразу доступны для просмотра всем остальным пользователям браузера в реальном времени.

II. Обзор архитектуры

На рисунке 2 схематично изображена общая архитектура системы.

Как уже говорилось раньше, клиент Layar, получив информацию от GPS приемника и G-сенсора, должен подготовить запрос информации о точках интереса, для того чтобы потом вывести их пользователю поверх видеопотока с камеры. Вся необходимая информация запрашивается у сервера Layar, который является основным связующим звеном между остальными компонентами системы. Вызов сервера Layar осуществляется посредством интерфейса (Layar Client API), детали реализации которого закрыты от конечных пользователей.

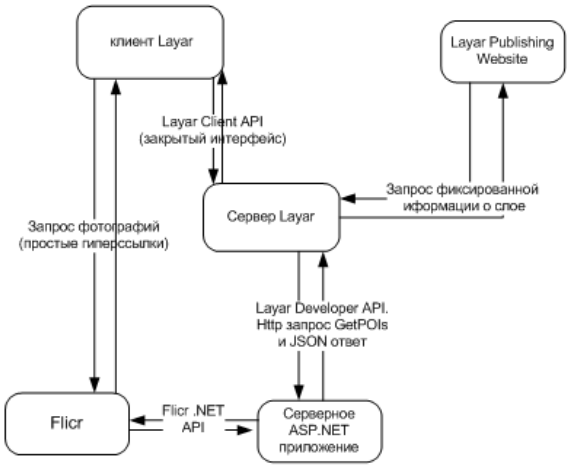

Рис. 2 Архитектура Layar

На сервере Layar хранится статическая информация о слое, загруженная туда разработчиком слоя посредством сайта Layar publishing. Layar publishing – это веб-интерфейс, позволяющий разработчикам управлять существующими слоями, публиковать и тестировать новые. При регистрации слоя на этом веб сайте, на сервер Layar загружается вся необходимая фиксированная информация о слое. Сюда входят:

- URL обработчика Layar Developer API запросов для предоставления динамической информации о точках интереса
- данные о внешнем виде слоя (набор используемых цветов, возможность загрузки собственных иконок для отображения точек интереса)
- текстовое описание слоя и тэги, по которым этот слой можно будет найти в каталоге
- версия Layar API, которая этим слоем поддерживается
- географические области, для которых доступен этот слой
- задание набора фильтров, которые будут доступны пользователю
- прочие настройки (например, признак того, является ли этот слой бесплатным или нет)

Главная из всех этих настроек – это URL адрес обработчика http запросов на стороне разработчика слоя, который будет обрабатывать веб-запросы от сервера Layar. Для запроса ресурсов от моего сервера сервер Layar использует один HTTP GET запрос определенного формата. В этом запросе веб-серверу разработчика передается вся необходимая информация от пользователя – географическое положение (широта, долгота), название запрашиваемого слоя, а также фильтры, установленные на стороне пользователя (в частности максимальная удаленность запрашиваемых точек интереса).

Получив всю необходимую информацию, обработчик

http запросов должен предоставить в ответ все необходимые данные. Эти данные в общем случае могут храниться локально, но в моем случае забираются с сервера - поставщика контента, Flickr.

Flickr – это онлайн веб 2.0 сервис, предоставляющий пользователям возможность загружать, хранить и управлять своими фотографиями. Он обладает собственным API, позволяющим программным образом обращаться к базе фотографий. В предлагаемой системе фотографии старой Москвы заводятся и геокодируются пользователями Flickr. Фото тэгируется одним из специальных тегов – «oldmoscow», «oldmos», «старая москва» и т.п.

Серверное приложение, получив запрос о предоставлении точек интереса, формирует Flickr API запрос к серверам Flickr, запрашивая все фотографии, удовлетворяющие заданному диапазону GPS координат и имеющим соответствующий тэг. Поиск осуществляется исключительно по тегам и геок-оординатам, никакой привязки к конкретному пользователю нет. В ответ Flickr предоставляет все фотографии, удовлетворяющие поисковым критериям и имеющим настройку приватности public (закрытые фотографии, естественно, в результирующую выборку не попадают). Такая система позволяет всем пользователям Flickr учувствовать в наполнении базы архивных фотографий. Таким образом, мы достигаем своей цели – получаем систему с user generated content.

Получив от Flickr необходимый массив информации (несколько ссылок на одну и ту же фотографию различных размеров, текстовое описание фото), задача серверного приложения – сформировать корректный ответ Layar серверу. В качестве формата ответа используется JSON. JSON ответ от сервера включает в себя всю необходимую информацию о точках интереса – их географические координаты, текстовое описание, удаленность от текущего географического местоположения пользователя и собственно ссылки на фотографии (представлены ссылки на фотографии в разных размерах).

Клиент Layar, получив информацию о массиве точек интереса, сам загружает из Flickr необходимые фотографии (размер загружаемой фотографии зависит от удаленности выбранной точки интереса от позиции пользователя). Фотографии рисуются как двумерные изображения поверх картинки с камеры. Все изображения масштабируются в соответствие с расстоянием до точек интереса, так чтобы эти изображения максимально естественно накладывались на реальные объекты. Само масштабирование настраивается разработчиком слоя и передается отдельно для каждой точки интереса при формировании JSON ответа от сервера.

III. LAYAR API

Для обработки http запроса, пришедшего от сервера Layar, было разработано специальное серверное приложение. В качестве платформы для создания этого серверного приложения использовалась связка IIS7+ASP.NET 2.0. Данный выбор продиктован исключительно удобством и возможностью использовать при разработки богатую библиотеку классов .NET. Кроме того, для платформы .NET существует удобная сборка, облегчающая разработчику работу с Flickr API, что и будет использоваться при обработке веб запросов.

Веб сервер IIS версии 7 стал гораздо теснее интегрирован с ASP.NET. [3] Так, в более ранних версиях этого веб-сервера от Microsoft ASP.NET был реализован как ISAPI - расширение к веб-серверу. Соответственно, при обработке веб-запроса, который предназначался обработке в ASP.NET, IIS перенаправлял этот запрос специальной aspnet_isapi.dll, которая и управляла приложениями ASP.NET. Запросы к не ASP.NET содержимому обрабатывались самим IIS или другими ISAPI фильтрами и не были видны ASP.NET. В версии же 7 очередь обработки веб-запросов ASP.NET напрямую накладывается на очередь обработки веб-запросов IIS. То есть ASP.NET представляет собой некую оболочку над стандартной схемой обработки запросов IIS, вместо того чтобы быть некой подключаемой частью общей схемы, как в более ранних версиях IIS.

Для создания обработчика http запросов в .NET необходимо создать класс, реализующий интерфейс IHttpHandler. Этот интерфейс помимо всего прочего включает в себя метод ProcessRequest, на вход которому подается объект класса HttpContext. Этот метод собственно и является входной точкой для обработки http запроса. В классе же HttpContext инкапсулируется вся необходимая информация о контексте пришедшего веб-запроса, в частности все переданные в веб-запросе параметры. Созданный класс заворачивается в .net сборку, которая и будет вызвана IIS при обработки пришедшего веб-запроса.

Для настройки отображения пришедших веб-запросов на разные обработчики, на веб-сервере используется секция *handlers* в конфигурационном файле веб-сервера *web.config*. В данном случае эта секция выглядит так:

```
<handlers>
  <add name="getPOIsHandler" verb="*"
    path="*"
    type="LayarHandler.getPOIsHandler, LayarHandler"
    resourceType="Unspecified" />
</handlers>
```

Обработчик с именем *getPOIsHandler* обрабатывает веб-запросы с маской * (то есть вобще все веб-запросы). При этом обработчик нужно искать в классе *getPOIsHandler* в сборке *LayarHandler*, которая обычно лежит в каталоге bin на веб-сервере.

Для получение информации о точках интереса от серверного приложения на стороне разработчика Layar использует простой http метод GET, то есть все параметры передаются в открытом в виде в самом URL запроса. Пример реального запроса от сервера Layar

приведен ниже:

http://193.169.33.71/getPOIs?lang=en&countryCode=RU&lon=37.6072311401&userId=6f85d06929d160a7c8a3cc1ab4b54b87db99f74b&developerId=0&developerHash=cba099d173c1d8ec668685d23b9290c4f216f716&version=4.0&radius=495&timestamp=1301151639463&lat=55.7636337005&layerName=oldmoscow&accuracy=100

В качестве основных параметров серверному приложению передается (часть из этих параметров опциональны)

- lon и lat - информация о текущих геокоординатах пользователя (его широта и долгота)
- layerName - название вызываемого слоя (на случай если серверное приложение обрабатываем запросы к нескольким слоям)
- version - версия API на стороне клиента (платформа активно развивается и более поздние версии Layar обладают существенно большей функциональностью; сервер должен знать, что из этого поддерживает клиент)
- userId - уникальный хэш код конечного пользователя слоя
- accuracy - точность геокоординат, которые предоставил клиент
- developerHash - используется только при разработке и отладке новых слоев разработчиком
- requestedPoiId - уникальное ID точки интереса, которая, по возможности, должна быть включена в ответ серверного приложения независимо от установленных фильтров (правила создания уникальных ID для точек интереса зависят исключительно от разработчика слоя, об этом будет сказано ниже при рассказе о подготовке ответа веб-серверу)
- checkboxlist, radiolist, radius, etc - фильтры выбранные пользователем данного слоя (сами фильтры зависят от слоя).

Один из наиболее интересных передаваемых параметров *radius* - указывает максимальное расстояние от текущего местоположения пользователя до интересующего его точки интереса. В идеале серверное приложение должно вернуть пользователю только те точки интереса, которые находятся в пределах круга с центром в точке (lan, lat) и с радиусом=radius, где *lan*,*lat* и *radius* - переданные в запросе параметры. Вообще говоря, честно высчитывать набор точек, лежащих внутри данного круга слишком сложно и не особенно нужно. Большинство систем, где реализован поиск геокодированной информации (например, Flickr и Picasa Web Albums), используют вместо окружности, так называемый *boundary box* - прямоугольник, задающий область поиска.

В данном случае эта область поиска будет задаваться квадратом с координатами сторон (lon - delta, lat - delta, lon + delta, lat + delta), где lan и lat - переданные параметры, а delta - параметр *radius*, представленный не в метрах, а в градусах. Переход от метров к градусам проведен из следующих соображений: общеизвестная длина окружности Земли равна 40009.88 км [4], соответственно изменение широты на один градус (при неизменности долготы) влечет за собой смещение на 40009.88*1000/360=111138.56 м и наоборот. В данных рассуждениях мы пренебрегаем тем фактом, что Земля не является идеальным шаром и немного приплюснута на полюсах. Соответственно, экваториальная длина окружности не будет совпадать с длиной окружности, проходящей через географические полюса. Однако такая потеря точности в контексте данной работы не принципиальна, и мы можем спокойно считать, что искомое смещение геокоординат в градусах delta=radius/111138.56. городов Российской Федерации,

IV. FLICKR API

Как уже говорилось при обзоре архитектуры системы, хранение фотографий старой Москвы обеспечивается сторонним сервисом хранения фотографий Flickr. Использование Flickr дает нам возможность реализовать в системе user generated content, что автоматически позволит сервису гораздо более активно расширяться (очевидно, что собрать большой банк фотографий одному человеку несоизмеримо сложнее, чем всему сообществу пользователей сервиса сообща). Кроме того, использование контента, который наполняют пользователи, во многом освобождает владельца сервиса от ответственности за выкладываемый контент перед правообладателями фотографий.

Одним из главных преимуществ использования Flickr в качестве поставщика фотографий было очень развитый программный интерфейс доступа к базе фотографий - Flickr API. Этот интерфейс дает сторонним разработчикам богатые возможности работы с Flickr и при этом хорошо задокументирован.[5] Кроме того, этот API хорошо поддерживается разработчиками, в отличие от Picasa Web Albums Data API [6] (разработка компании Google, схожая по функциональности с Flickr).

Flickr API представляет собой набор методов, к которым можно обратиться посредством нескольких конечных точек API. Для того чтобы произвести вызов метода Flickr API необходимо сформировать запрос к одной из конечных точек (например, *https://secure.flickr.com/services* или *http://api.flickr.com/services*), передав в качестве параметра имя вызываемого метода и аргументы для него. В ответ от Flickr должен прийти форматированный ответ. Данный API поддерживает большое количество разнообразных соглашений о вызове методов. Так в качестве формата запроса может быть использован простой REST (поддерживаются и метод GET, и метод PUT), XML-RPC и SOAP. В качестве ответа от сервера может прийти все те же REST, XML-RPC, SOAP, а тажке JSON и формат сериализации в PHP. При обращении к конечной точке в качестве обязательных параметров требуется уникальный API ключ разработчика (его легко можно заказать на сайте Flickr), а также имя вызываемого метода. В качестве опционального используется параметр *format*, применяемый для того, чтобы указать в каком формате необходимо прислать ответ на запрос. По умолчанию ответ придет в том же формате, что и формат исходного запроса.

Ключевой метод, вызов которого используется в моей системе, называется *flickr.photos.search*. Метод возвращает набор фотографий, удовлетворяющий определенным критериям. В нашем случае нас будут интересовать все гео-кодированные фотографии, удовлетворяющие заданному диапазону гео-координат и помеченные необходимым тэгом. В общем случае в ответ Flickr возвращает только общедоступные фото, для получения доступа к закрытым фотографиям, требуется аутентификация. Нам доступ к закрытым фотографиям не потребуется, так как основная идея системы состоит в том, чтобы пытаться найти необходимые фотографии среди всего массива фотографий Flickr, не ограничиваясь конкретными пользователями.

Метод *flickr.photos.search* имеет большое количество передаваемых параметров, все они, за исключением API ключа разработчика, опциональны. Ниже дано описание тех параметров, которые используются в данном приложении:

• bbox - параметр, отвечающий за фильтрацию результатов запроса по гео-координатам. Представляет собой 4 значения, разделенные запятой и определяющие ограничивающую область для гео-поиска. Первые два значения задают левую нижнюю точку прямоугольника, а последние два - правую верхнюю. То, каким образом мы получаем ограничивающую область в нашем случае, было описано в выше. Интересный момент состоит в том, что для предотвращения падения производительности базы данных Flickr от большого числа гео-запросов, на гео-запросы стоит дополнительное ограничение - в случае если в гео-запросах отсутствует дополнительная фильтрация (например, минимальная дата загрузки фотографии или минимальная дата снимка), то Flickr возвращает только фотографии, добавленные за последние 12 часов. В нашем случае дополнительным ограничивающим параметром являются тэги, поэтому наш запрос не подпадает под это ограничение.

• accuracy - интересный параметр, отвечающий за точность гео-координат. Представляет из себя целое число от 1 до 16 (где точность 1 - это самое размытое значение (World level), а 16 - наоборот самое точное - Street level). По умолчанию стоит самый жесткий уровень - 16. Фактически данная информация соответствует масштабу карты Flickr в тот момент, когда пользователь гео-кодировал данную фотографию. Очень часто пользователи не используют максимально возможное увеличение карты при гео-кодировании фотографии, поэтому большая часть фотографий на Flickr имеет параметр accuracy < 16. Для соблюдения разумного баланса между точностью и возможной потерей фотографий в результирующей выборке я использую точность=15.

• extras - параметр, отвечающий за то, какая именно информация о фотографиях будет в результирующей выборке. Понятно, что поиск фотографий осуществляется разными приложениями для разных целей, поэтому разным приложениям нужны совершенно разные данные о фотографии. В целях оптимизации запрос по умолчанию возвращает только самую необходимую информацию о фото. Для предоставления большей информации нужно задействовать данный параметр. В нашем случае мы указывает Flickr помимо стандартной информации дополнительно возвращать информацию о географических координатах фотографии.

• tags - набор тэгов, разделенных запятой, одним из которых должна быть помечена искомая фотография. В данный момент приложение пытается искать во Flickr фотографии старой Москвы по одному из следующих тэгов: oldmoscow, old_moscow, old moscow, oldmos, старая москва, old_moscow, старая_москва.

Для всех основных платформ доступны программные наборы, облегчающие работу с Flickr API. Эти наборы представляют из себя надстройку над интерфейсом Flickr, скрывающую от разработчика такие подробности как формирование веб-запроса в корректном формате и разбор ответа от сервера Flickr. Эти наборы не поддерживаются самим Flickr, корректность их работы зависит только от разработчиков данных наборов, но, тем не менее, они достаточно удобны и ссылки на них приведены на официальной странице документации по Flickr API.

В данной работе используется один из такиз наборов - Flickr .NET API Library [7]. Набор представляет собой .NET сборку, инкапсулирующую в себе вызов методов Flickr и обработку пришедших от сервера результатов. Все методы сборки соответствуют методам Flickr API, но написаны таким образом, что работа с ними осуществляется в стиле других классов стандартной библиотеки .NET. Например, используемый в данном случае метод Flickr.*PhotosSearch* соответсвует методу API *flickr.photos.search* и возвращает коллекцию объектов *PhotoCollection*, которую можно просмотреть стандартным циклом *foreach*. Кроме того, помимо инкапсуляции вызовов методов Flickr данная сборка реализует и другую дополнительную функциональность, в частности кэширование запросов.

V. LAYAR API - ОТКЛИКИ

После получения необходимых данных от Flickr в задачу разработанного серверного приложения входит построение корректного ответа для сервера Layar. В качестве формата ответного сообщения в Layar API используется формат JSON. JSON расшифровывается как Java Script Object Notation и представляет собой удобный легковесный текстовый формат представления данных. Пример реального JSON ответа, возвращенного серверным приложением, приведен ниже.


```
{
   "hotspots": [
      {
         "id": "60159991@N08-5501783931",
         "distance": 354.064037022007,
         "title": "Церковь Рождества Пресвятой Богородицы в Путинках на Малой Дмитровке",
         "type": 0,
         "lat": 55766814.000000,
         "lon": 37606952.000000,
         "attribution": " Старая Москва ",
         "line2": "снимок датирован:",
```

```
        "line3": "20 век",
        "line4": "",
        "imageURL":
"http://farm6.static.flickr.com/5140/5501783931_c6e77434
58_s.jpg",
        "dimension": 2,
        "object": {
           "baseURL": "http://farm6.static.flickr.com/5140/",
           "full": "5501783931_c6e7743458_m.jpg",
           "reduced": "5501783931_c6e7743458_s.jpg",
           "size": 1
        },
        "transform": {
           "rel": true,
           "angle": 0,
           "scale": 10.0
        },
        "actions": [

        ]
     }
  ],
  "layer": "oldmoscow",
  "errorString": "ok",
  "morePages": false,
  "errorCode": 0,
  "nextPageKey": null
}
```


Ответ от серверного приложения можно разделить на две части – собственно массив точек интереса (hotspots), а также ряд некоторой общей информации, связанной с конкретным запросом.
Большинство из параметров запроса, определенные в Layar API, опциональны и могут не использоваться при формировании ответа. Ниже описаны лишь те, которые используются в данном приложении:

• errorCode и errorString – поля, ответственные за возвращение результата выполнения запроса. 0 – нормальное завершение запроса. В случае возникновения какой-то ошибки при обработке запроса разработчик волен вернуть код этой ошибки и текст ошибки, который будет выведен на экран пользователю
• layer – просто текстовое имя слоя, для которого возвращаются точки интереса
• morepages, nextpagekey – параметры, используемые при разделении результатов запроса на несколько страниц в целях оптимизации. Такая оптимизация используется, если необходимо как можно быстрее отразить результаты выполнения запросу пользователю. В этом случае установленный флаг *morepages* в *true* указывает, что ответ разбит на несколько страниц, и в случае необходимости следующая страница будет дополнительно запрошена по ключу со значением *nextpagekey*. В данной работе этот механизм не используется, поэтому *morepages* установлен в *false*, а nextpagekey – в *null*

Вся информация о точках интереса хранится в специальном массиве *hotspots*, каждый из элементов которого представляет достаточно сложную структуру. Ниже приведено описание наиболее интересных из элементов, используемых в ответе серверу:

• id – уникальный идентификатор точки интереса. Способ создания этого идентификатора лежит на разработчике. Id обязательно должен быть уникальным, так как на основании этого идентификатора клиент Layar отслеживает массив доступных пользователю точек интереса при обновлении этого списка. Я для формирования уникального id комбинирую id фотографии и id пользователя Flickr. Необходимость использовать идентификатор пользователя, которому принадлежит фотография, обусловлена тем, что идентификатор самой фотографии уникален только в рамках данного пользователя, но не в рамках всей базы фотографий Flickr
• distance – расстояние от точки интереса до местоположения пользователя в метрах. Вообще говоря, задача определения расстояния между двумя географическими координатами сама по себе достаточно интересная. Здесь опять используется упрощение, что Земля представляет из себя форму сферы, а не эллипсоида, и поэтому в рамках данного упрощения могу применять сферическую геометрию и, в частности, формулы для вычисления расстояний на большом круге. Существует несколько основных способов рассчитать сферическое расстояние на большом круге (то есть, фактически, найти кратчайшее расстояние между двумя географическими координатами) – сферическая теорема косинусов и формула гаверсинусов. [8]. Из-за возможных ошибок округления сферическая теорема косинусов мало применима для расчета маленьких расстояний. С учетом того, что в своей работе я оперирую как раз небольшими расстояниями, расчет параметра *distance* производится по формуле гаверсинусов.

$$\Delta\sigma = 2\arcsin\left\{\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right)+\cos\phi_1\cos\phi_2\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right\}.$$

где
$\phi_1, \lambda_1;\ \phi_2, \lambda_2$ - широта и долгота двух географических точек в радианах, а
$\Delta\sigma$ - угловая разница
Искомое расстояние определяется как угловая разница, помноженная на радиус Земли.

• title, attribution, line2, line3, line4 – поля, ответственные за текстовое описание предоставленной точки интереса. Значение в поле *title* в данном случае соответствует названию фотографии во Flickr.

Атрибут dimension имеет особое значение и указывает, в каком режиме будут отображаться возвращенные точки интереса. Layar API, начиная с версии 3, поддерживает отображение двумерных и трехмерных объектов поверх видеопотока с камеры. До этого поддерживалось только отображение простых иконок, дополненных текстовой информацией (dimension=1). В нашем случае нам требуется выводить на экран пользователя двумерные фотографии старой Москвы, поэтому используемое

значение параметра dimension=2. Для поддержки таких двумерных изображений в JSON ответе присутствуют два специальных объекта– object (то, как этот объект будет выглядеть) и transform (то, как этот объект будет преобразовываться в зависимости от удаленности и направления взгляда пользователя).
Важный момент состоит в том, что сами фотографии хранятся на серверах Flickr, в JSON ответе присутствует только URL фотографии, которую самостоятельно использует клиент Layar. В целях оптимизации в атрибуте object указывается сразу два разных URL, которые ведут на одну и ту же фотографию, но в разном качестве. Выбор качества фотографии будет осуществляться клиентом на основании удаленности точки интереса от наблюдателя. В нашем случае хранение фотографии в разном качестве обеспечивается функциональностью самого Flickr. Качество фотографии кодируется в URL этой фотографии (а именно в последнем символе). В реализации используются суффиксы *s* (small square размер фотографии 75x75) и *m* (small – 240 пикселей на длинной стороне).

Очевидно, что в зависимости от расстояния пользователя до точки интереса размер фотографий должен быть разный. В идеале фотографии точек интереса должны масштабироваться в соответствие с реальными объектами. Для того чтобы предсказать необходимый размер фотографии, используются параметры size объекта object и scale объекта transform. Size – это указание разработчиком слоя реального размера объекта (в нашем случае изображения). Scale – коэффициент масштабирования, который применяется к параметру size. Масштабирование зависит от удаленности объекта (параметр *distance*). После умножения размера изображения на коэффициент масштабирования получившийся размер объекта в пикселях используется для определения того, какое качество фотографии будет использоваться для рендеринга.

Сочетание параметров *rel=true* и *angle=0* в объекте *transform* говорит о том, что изображение всегда будет повернуто лицом к пользователю независимо от реального направления камеры мобильного устройства.

VI. Примечание редактора

Это еще одна ранее не опубликованная работе в рамках научных тем, предлагаемых для квалификационных работ в лаборатории ОИТ [9]. Научный руководитель – к.ф.-м.н. Д.Е. Намиот. У автора получилась хорошая техническая работа и, весьма важно, полностью доведенная до практической реализации. Результаты работы и само направление ни в коей мере не хотелось бы забрасывать. В качестве весьма интересного направления можно предложить интеграцию дополненной реальности и контекстно-зависимых вычислений. Например, SpotEx [10,11] позволяет связать контент для мобильных пользователей с элементами сетевой инфраструктуры. Так вот этим контентом вполне может быть AR уровень. Например, как показано в [11], гео-координаты могут быть заменены информацией о сетевой близости. Вот это и интересно было бы использовать для программирования дополненной реальности.

Библиография

[1] Azuma, Ronald T. "A survey of augmented reality." Presence 6.4 (1997): 355-385.

[2] Madden L. Professional augmented reality browsers for smartphones: programming for junaio, layar and wikitude. – Wiley. com, 2011.

[3] Evjen, Bill, and Farhan Muha. Professional ASP. Net 2.0. Wiley. com, 2008.

[4] Петросова Р. А. и др. Естествознание и основы экологии Москва: Академия. – 1998.

[5] Flickr API. http://www.flickr.com/services/api/

[6] Bromberg, Yérom-David, et al. "Bridging the interoperability gap: overcoming combined application and middleware heterogeneity." Middleware 2011. Springer Berlin Heidelberg, 2011. 390-409.

[7] FlickrNet http://www.codeplex.com/wikipage?ProjectName=FlickrNet

[8] Привалов В. Н., Обабков И. Н. МЕХАНИЗМЫ РЕШЕНИЯ ПРОБЛЕМ ИСПОЛЬЗОВАНИЯ ДАННЫХ ГЕОИНФОРМАЦИОННЫХ СЕРВИСОВ //М 75 «Молодежный научный форум: Технические и математические. – 2013. – С. 21.

[9] Намиот, Д., & Сухомлин, В. (2013). О проектах лаборатории ОИТ. International Journal of Open Information Technologies, 1(5), 18-21

[10] Namiot, D., & Sneps-Sneppe, M. Local messages for smartphones. Future Internet Communications (CFIC), 2013 Conference on, pp.1-6, IEEE, DOI: 10.1109/CFIC.2013.6566322

[11] Dmitry Namiot and Manfred Sneps-Sneppe. “Geofence and Network Proximity”. Internet of Things, Smart Spaces, and Next Generation Networking, Lecture Notes in Computer Science Volume 8121, 2013, pp. 117-127, DOI: 10.1007/978-3-642-40316-3_11

Layers for Augmented Reality Browser.

Baulin I.N.

Abstract — This article presents the results of the qualifying work done by the author at the Laboratory of OIT during the master degree study at the Faculty of Computational Mathematics and Cybernetics Lomonosov Moscow State University. It describes the practical problem of representation of third-party content in an augmented reality browser. Also this paper presents the tools for creating augmented reality mashups.

Keywords — augmented reality, photos,mashup.