МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ПРОГРАММА «РАЗРАБОТЧИК ПРОФЕССИОНАЛЬНО-ОРИЕНТИРОВАННЫХ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА НА ТЕМУ

Контекстно-зависимый сканер QR-кодов

работу выполнил:

Абрамов Алексей Александрович

научный руководитель:

Старший научный сотрудник
Намиот Дмитрий Евгеньевич

МОСКВА 2013

Оглавление

Аннотация

Цель данной работы — реализация модификации сканера – кодов, позволяющего учитывать окружающий контекст на базе реализованого Open Source проекта QR-code reader Zxing.

В результате работы данная модификация была реализована и на ее базе создан опытный образец.

Работа содержит: 36 Страниц, 7 Иллюстраций, 3 Приложения.

Ключевые слова: Zxing, QR, сканер, учет контекста, Context-aware QR-code reader

Введение

QR-код (англ. *quick response* — быстрый отклик) — матричный код (двумерный штрих-код), разработанный и представленный японской компанией «Denso-Wave»в 1994 году.

Огромная популярность обычных штрихкодов в Японии привела к тому, что объём информации, зашифрованной в нём, вскоре перестал их устраивать. Они начали экспериментировать с другими способами кодирования небольших объёмов информации в графической картинке.

Основное отличие от обычного старого штрих-кода, который сканируют тонким лучом, QR-код определяется сенсором как двумерное изображение. Три квадрата в углах изображения синхронизирующие квадратики по всему коду позволяют нормализовать размер изображения и его ориентацию, а также как и угол, под которым сенсор относится к поверхности изображения. Точки переводятся в двоичные числа с проверкой по контрольной сумме.

Основное достоинство QR-кода — это лёгкое распознавание сканирующим оборудованием (в том числе и фотокамерой мобильного телефона), что дает возможность использования в торговле, производстве, логистике.

Максимальное количество символов, которые помещаются в один QR-код:

- цифры — 7089;
- цифры и буквы (включая кириллицу) — 4296;
- двоичный код — 2953 байт;
- иероглифы — 1817.

Аббревиатура «QR code» является зарегистрированной торговой маркой компании «DENSO Corporation»,но использование кодов не облагается никакими лицензионными отчислениями, а сами они опубликованы и описаны в качестве стандартов ISO.

Спецификация QR-кода не описывает никаких форматов данных. Наиболее популярные программы просмотра QR-кодов поддерживают такие форматы ,

получившие наибольшее распространение, данных: URL, Закладка в браузер, Email (с темой письма), SMS на номер (с темой), vCard, географические координаты.

Также есть возможность распознавать GIF, JPG, PNG или MID файлы меньше 4 КБ и зашифрованный текст, но эти форматы не получили популярности.

Применение

QR-коды больше всего распространены в Японии, стране, где штрих-коды пользовались такой большой популярностью, что объём информации, зашифрованной в коде, вскоре перестал устраивать индустрию. Уже в начале 2000 года QR-коды получили такое широкое распространение в Японии, что их можно было встретить практически везде. На большом количестве плакатов, упаковок и товаров. Там подобные коды наносятся практически на все товары, продающиеся в магазинах, их размещают в рекламных буклетах и справочниках. С помощью даже организовывают различные конкурсы и ролевые игры.

Практически все ведущие японские операторы мобильной связи совместно выпускают мобильные телефоны со встроенной поддержкой распознавания QR-кода под своим брендом.

В настоящее время QR-код очень широко распространён в странах Азии (особенно в Японии), постепенно развивается в Европе и Северной Америке. Наибольшее признание он получил среди пользователей мобильной связи — установив программу-распознаватель, абонент может моментально заносить в свой телефон текстовую информацию, добавлять контакты в адресную книгу, переходить по web-ссылкам, отправлять SMS-сообщения и т. д.

QR-коды активно используются музеями, а также и в туризме. Например, во Львове (Украина), объединение бизнесменов «Туристическое движение Львова» разместило QR-коды более чем на 80 туристических объектах. Это позволяет индивидуальному туристу легко ориентироваться в городе, даже не зная

украинского языка, так как QR-коды установлены на нескольких языках.

Постановка задачи

Цель данной работы — создание наиболее удобной модификации сканера QR-кодов, позволяющего учитывать окружающую обстановку основываясь на показаниях:

а) Данных технологии спутниковой связи GPS

б) Данных базовых станций сотовой связи (тех. A-GPS)

в) Имени и мощности окружающих место сканирования WI-FI сетей

Данная разработка поможет сильно упростить и удешевить полезные возможности применения технологии QR- кодов. Больше не нужно будет создавать уникальные QR-коды для каждого места их применения. Данная модификация позволит, основываясь на данных GPS получать координаты места сканирования и передавать эти данные на специальный веб-сервис. Он в свою очередь будет их анализировать и выдавать результат уже «привязывая» их к конкретному местоположению.

Теперь будет достаточно распечатать QR-код на листовках с закодированной веб-ссылкой и раздавать их где угодно совершенно не беспокоясь о привязке, так как при ее считывании и переходе все необходимые данные автоматически включатся в запрос и останется их только обработать.

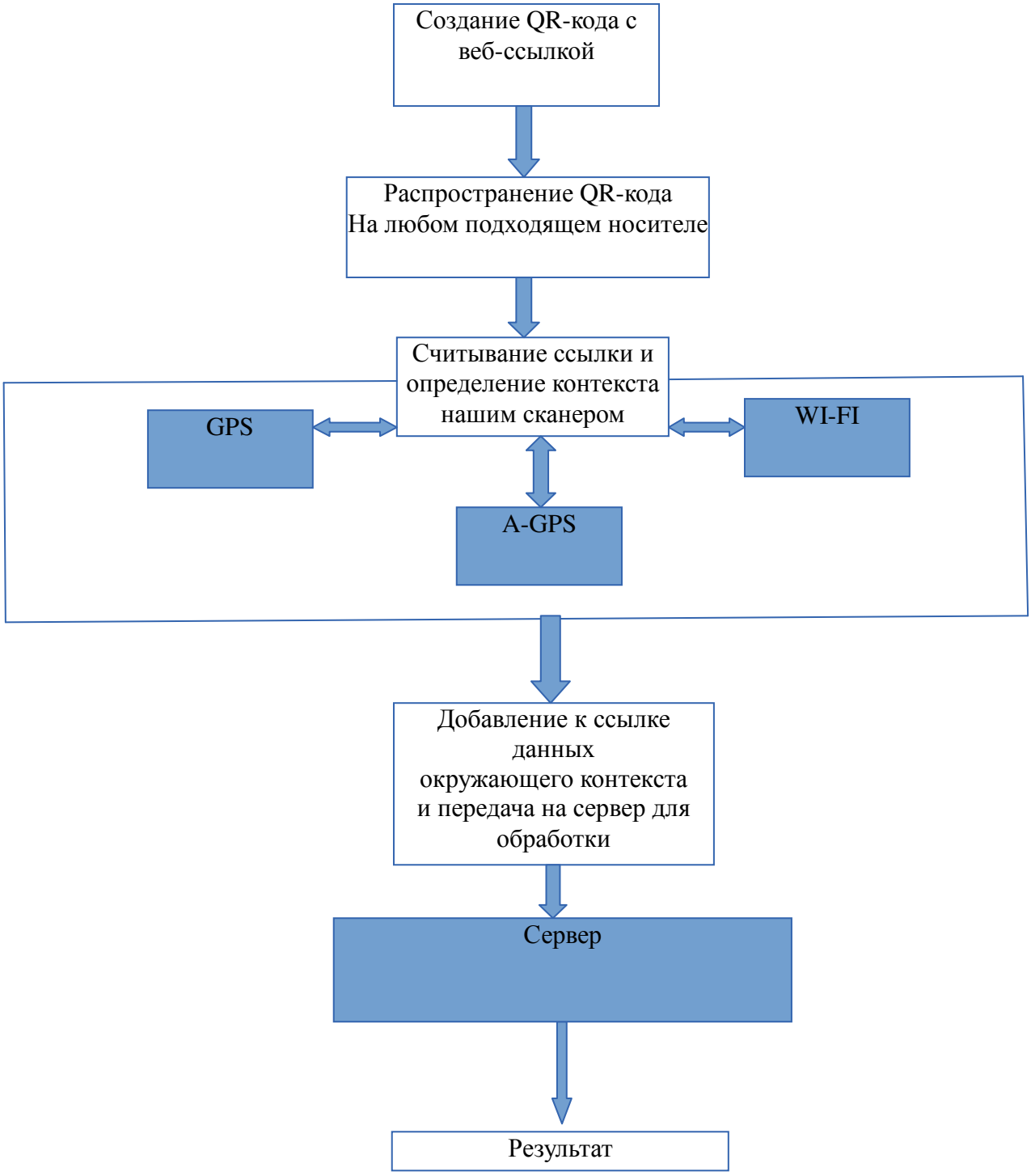

Рис. 1. Общая схема работы системы

Обзор существующих решений.

Proximity marketing- это маркетинговая технология, основанная на распределении рекламного контента в определенном месте и доступная с помощь беспроводных технологий. При этом она является наиболее востребованной информационной технологией в торговле. В связи с ростом конкуренции в торговле постоянно приходится искать новые способы привлечения клиентов. А рост и развитие информационных технологий не мог остаться незамеченным. Сейчас практически у каждого экономически активного человека есть мобильный телефон , который давно перестал быть лишь средством сотовой связи. Широкие возможности предоставляемые мобильными технологиями нашли свое отражение и в сфере торговли и развлечения. Наличие беспроводных технологий , будь то WI-FI,GPS, *Bluetooth,* открывает возможности недоступные всего лишь несколько лет назад.
Теперь человека можно информировать о той или иной услуге или возможности дистанционно и локально.

Этим и объясняется такая широкая его распространенность в развитых странах и городах.

На сегодняшний день представлено множество решений, в которых для этого используются самые разные технологии.
Их можно условно разделить на следующие категории:

- Интерактивные экраны и табло
- Гео-позиционные системы на основе GPS
- Системы на основе *Bluetooth*
- Системы на основе *WI-FI*
- Смешанные системы

Остановимся на них поподробнее:

Интерактивные экраны и табло – являются наиболее простым способом информирования и посетителя о предоставляемых услугах и возможностях. Как правила они содержат карты помещений с указанием возможных маршрутов достижения того или иного отдела. Часто их совмещают с банкоматами, платежными терминалами и т.п. Это служит для экономии места и снижает общие издержки;

Минусы таких решений очевидны:

- Дороговизна решения
- Отсутствие стандартов процедур обновлений

В тоже время это одно из самых удобных для пользователя способов поиска нужной ему информации. Ведь чтобы им воспользоваться не нужно иметь с собой абсолютно никаких приспособлений, а интерфейс подобных экранов довольно удобно продуман и интуитивен, что немаловажно для лиц пожилого возраста и детей.

Рис(2). Интерактивный экран

Другое возможное решение - использование гео-позиционных систем на основе

GPS.

Применение очевидно:

Человек имея телефон, смартфон или навигатор может получить координаты своего местоположения, отображаемые прямо на карте с заранее нанесенными на нее точками возможностей. Но работа в помещение не относится к сильным сторонам этой технологии, так как недоступность спутников позиционирование сильно сказывается на точности определения местоположения, что во многих случаях неприемлемо.

Одной из возможностей решения этой проблемы является использование ретрансляторов сигнала GPS, которые устанавливаются уже внутри помещений. В результате сигнал усиливается и на основе уже распланированных GPS-решений происходит позиционирование внутри здания. Этим достигается весьма неплохая точность определения местоположения. Но , естественно , это требует довольно существенных затрат по установке и обслуживанию ретрансляторов, что довольно дорого. Кроме того, этой возможностью могут воспользоваться лишь владельцы устройств с GPS.

Итак , плюсы такой системы:

- Хорошая точность позиционирование даже внутри зданий
- Относительная простота в использовании для человека

К минусам можно отнести:

- Дороговизна аппаратуры и монтирование с настройкой
- Относительная сложность и высокая стоимость в обслуживании

Что во многих случаях малого бизнеса делает данное решение нерентабельным.

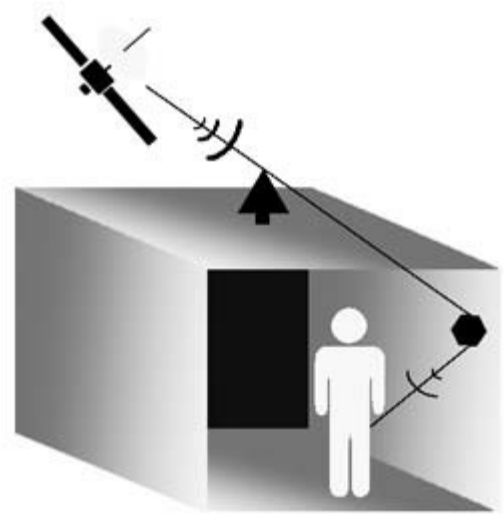

Рис.(3) Позиционирование с помощью GPS ретрансляторов

Следующее решение — использование технологии *Bluetooth*.
Оно построено на использовании возможностей *Bluetooth,* так как оно изначально локальная технология. Т.е. может использовать только находящиеся поблизости устройства.
Одно из возможных решений на данной технологии предложила Nokia. Они продемонстрировали прототип устройства использующее расширение стандарта *Bluetooth* 4.0 протоколом *Location Extension,* разработку которого же сами и ведут.
Суть сводится к следующему:
Используя принцип триангуляции(определения расстояния по 3-м точкам) массив антенн отслеживает местоположения устройств с “метками” *Bluetooth* внутри помещения , что дает хорошую точность. Но есть и минусы. Скорость определения *Bluetooth* устройств в несколько раз медленнее, чем если сравнивать их, например с WI-FI. Кроме того , есть и организационные вопросы, так как чаще всего у человека включен модуль WI-FI нежели *Bluetooth.* Хотя с выходом *Bluetooth* 4.0 энергопотребления этой технологии существенно уменьшилось.
Итого к плюсам можно отнести:

- Относительная дешевизна

- Хорошая степень точности позиционирования
- Bluetooth — изначально локальная технология
- Низкое энергопотребление клиентского устройства
- Наличие практически в каждом мобильном устройстве

К минусам же можно отнести:

- “Долгое” время определения устройств по сравнению с WI-FI
- Стадия прототипа
- На клиенте скорее включен WI-FI нежели *Bluetooth*
- Следующее решение — использование технологии *Wi-Fi*.

Сети Wi-Fi сейчас распространении повсеместно, что делает ее весьма привлекательным инструментом для позиционирования в помещениях .
Есть несколько компаний предлагающих свои наработки в этом направлении.
Как правило, такие решения предъявляют очень строгие требования к качеству сигнала. К тому же, требуют установки дополнительного программного обеспечения , а зачастую и покупки устройств, подчас весьма дорогостоящих.
Основная проблемная сложность систем на базе на базе *Wi-Fi*- это необходимость предварительной разметки всех помещений, где планируется использование этой системы. Так называемая калибровка.
В общем виде система позиционирования работает следующим образом: создается радио карта помещения. Для каждой точки определяются адреса «видимых» (с точки зрения *Wi-Fi*) точек доступа и соответствующая сила сигнала. Таких точек может быть очень много.
Термины *Wi-Fi*:
SSID — идентификация сети, RSSI — относительная сила сигнала.
Для одной эталонной точки — точки (места) выбранной для калибровки системы — получаем вектор таких пар значений. Это так называемый отпечаток (*Wi-Fi fingerprint*).
Подготовка помещения, собственно говоря, состоит именно в создании подобного рода базы данных, содержащей отпечатки и их гео-координаты.
Далее, на этапе практической эксплуатации, определяется *Wi-Fi*-отпечаток проверяемого мобильного устройства. По имеющейся

калибровочной базе данных ищутся наиболее «похожие» отпечатки. Если база отпечатков будет содержать и координаты, то это и есть способ, которым можно вычислить приблизительное расположение устройства. Таким образом, задача позиционирования в помещениях не будет, в принципе, отличаться от позиционирования при наличии *GPS*.

Существуют комбинационные решения (например,*AGPS — assisted GPS*), но в любом случае основой является база *Wi-Fi*-отпечатков.

Естественно, что точность позиционирования зависит от максимальной полноты базы. Ее наполнение и поддержка в актуальном состоянии — самый дорогостоящий этап систем позиционирования подобного рода. Возможны краудсординговые решения, когда информация о точках *Wi-Fi* собирается с помощью мобильных абонентов (как, например, собирается информация о дорожных пробках). В любом случае остается открытым вопрос о поддержке, так как сетевое окружение может меняться (состав *Wi-Fi*-узлов, например), точка доступа *Wi-Fi* перемещаться в другое место и т. д.

Еще одним большим минусом этой системы является отсутствие поддержки динамического гео-позиционирования. Большинство существующих систем *Wi-Fi*-позиционирования ориентированы именно на офисные сети, где расположение сетевых узлов может оставаться постоянным в течение длительного времени. Но что если, если точка доступа *Wi-Fi* открыта на мобильном устройстве? Большинство современных смартфонов поддерживают подобного рода возможности. В этих системах контент доступен в какой-то ограниченной области (то, что называется *location aware data,* — данные, зависящие от местоположения). Их доступность зависит от видимости некоторой точки доступа *Wi-Fi*. Триггер видимости может не только менять свое состояние (включен - выключен), но и перемещаться в пространстве .

Общий обзор методов позиционирования в помещениях можно найти, например, в [2].

Исследование и построение решения задачи

Как видно из обзора существующих решений основной трудностью для реализации доступности информационного контента для конечного пользователя являются:

а) Дороговизна таких решений

б) Сложность в поддержке этих систем в актуальном состоянии

В связи с этом было выбрано решение объединяющее возможности существующих , но при этом, лишенное их основных недостатков.

То, есть перед собой мы поставили задачу создать универсальный инструмент, позволяющий без лишних затрат и использую уже существующую инфраструктуру получать доступ к специально размещенной и потенциально интересной конечному пользователю информации.

В качестве основной концепции было выбран учет контекста(Context awareness).

Это концепция характеризуется следующей сутью:

Учет контекста является особенностью мобильных устройств, который определяется в дополнение к местоположению. В то время как местоположение можно определить, как определенные процессы в устройстве, контекст может быть применен более гибко с мобильными пользователями, особенно с пользователями смартфонов. Учет контекста возник из стремления иметь дело со связью между компьютерными системами и изменениями в окружающей среде, которые в противном случае принимаются за статические. Более подробно об этой концепции можно узнать например здесь[4].

В качестве основной носителя был выбран QR- код (двумерный штрих-код). Основной его плюс состоит в том, что он легко распознается сканирующим оборудованием (в том числе и фотокамерой мобильного телефона), что дает возможность использования в торговле, производстве, логистике. И так как практически у каждого сегодня имеется телефон с камерой, мы получаем потенциально 100% аудиторию имеющую доступ к нашей технологии без затраты дополнительных материальных ресурсов. Установка приложения на устройство не учитывается ввиду отсутствия материальных затрат.

Кроме того размещение данного кода не требует никаких дополнительных затрат , так как должно выполняться лишь одно требование – читабельность изображения для устройств. Вплоть до размещения на листах бумаги, стенах, асфальте и т.п. Также эта технология отличается довольно сильной коррекцией ошибок, что позволяет считаться информации даже при повреждении читабельности кода.

Рис 6. QR- код с изображение, частично снижающем его читабельность.

Количества информации, которое можно представить, таким образом, вполне достаточно, чтобы закодировать URL-ссылку, небольшую картинку или любую другую краткую информацию.

Само кодирование так же не представляет никакой сложности ввиду открытости алгоритмов и наличия большого количества готовых и бесплатных решений.

В качестве приложения –донора был выбран open source проект ZXING за его широкий функционал, стабильную работу и продуманный интерфейс, что сильно повысило скорость разработки нашего прототипа.

В качестве параметров для учета контекста были выбраны:

1) Координаты местоположения
2) Идентификационные и силовые параметры окружающих WI-FI сетей

Координаты планировалось определять на основе наиболее качественных на момент определения провайдеров.

1) GPS (основываясь на показаниях спутника)
2) A-GPS (основываясь на данных базовых станций сотовой связи)
3) Passive (при отсутствии других на основе последнего удачно определения)

Именно в приведенной последовательности и определяются данные местоположения от наиболее до наименее точных.

Определение окружающих сетей также не должно представлять сложности ввиду того что разработка прототипа планируется ввести для платформы Android, стандартная библиотека которого позволяет получить такой список довольно простым способом.

В итоге мы должны получить приложение доступное для большинства потенциальных пользователей имеющих смартфон с фотообъективом, которое позволит не просто считать закодированную информацию из QR- кода, но и при условии что закодирована URL-ссылка, добавить к ней в качестве параметров окружающий контекст. В нашем случае это географические координаты и список доступных WI-FI сетей с их уникальными характеристиками. Все это позволит при переходе по ссылке передавать эти данные на сервер, который в свою очередь потенциально сможет

проанализировать их и выдать соответствующий результат с учетом переданных параметров. Что и соответствует выбранной концепции, а так же решает поставленную задачу.

В качестве дальнейшего развития данного направления можно предложить создание серверной компоненты , которая и будет анализировать приходящие параметры и выдавать результат с учетом контекста.
Другим возможным решение может стать создание “облачной” базы данных , где будут храниться соответствия между приходящими запросами и их идентификаторами , что в последствии позволит выдавать результат по идентификатору без дополнительного анализа .

Описание практической части

Так как было принято решение использовать для практической реализации Open Source проект QR-code reader Zxing, это позволило сильно сократить время разработки, так как проект достаточно зрелый и имеет наиболее широкий функционал из доступных для модификации проектов.

Разработка осуществлялась для платформы Android как наиболее распространенной.

Основная реализация заключалась в создании 2-х основных классов:

1) ContexWiFiManager.java
2) LocatioManager.java

Далее остается лишь внести модификаций в класс QRCodeReader.java

Необходимо было решить следующие задачи:

1) Определение текущего место положения(координаты GPS)
2) Определение сетевого окружения (SSID,RSSI,MAC окружающих WI-FI сетей)
3) Объединение полученных результатов и слияние их с адресом из QR кода.

Начнем определение координат местоположения. Для начала определим необходимые требования для работы с GPS в системе.

1. **Давать возможность разработчику не следить за подписанием на уведомления об изменение GPS координат**. Часто бывает нужно, чтобы подписание действовало только в то время, когда приложение открыто – при деактивации формы (stop, pause) нужно отписываться, а при активации – создавать нового слушателя и подписывать его на уведомления об изменение. Недостатков в этом подходе довольно много. От тривиальных – можно забыть отписаться при закрытии формы, до нетривиальных – если слушателей много, то они могут быть не синхронизированы и иметь разное значение координат, а также

множество одинаковых объектов – это дополнительная нагрузка на память и процессор (и, как следствие, на аккумулятор). Поэтому логически вытекает второй пункт требований;

2. **Использовать всего один объект LocationListener и при этом он не должен быть singleton**. В Android OS экземпляры объектов уничтожаются вместе с Activity, на которой они были созданы поэтому создавать объекты с глобальным доступом следует не в Activity, а в Application. Создавать новый экземпляр LocationListener при каждом обращении к GPS не стоит, сборщик мусора сам убирает отписанных слушателей, но лишние объекты занимают память, процессорное, и увеличивают энергопотребление,- пусть и кратковременно. При объявлении LocationListener в Application, вся работа будет происходить с одним объектом, что упрощает отладку приложения. При работе с GPS на устройстве часто необходимо выставить ложные координаты. Стандартный способ требует изменения манифеста, добавления новых разрешений, написание класса для выставления координат и поэтому является слишком сложным и трудоемким. Гораздо проще выставить фиксированное значение координат для одного объекта LocationListener;
3. **Быстро получать приблизительные координаты**. Часто при старте приложения нужно определить примерное положение клиента (например, для загрузки конкретного контента или локализации приложения) – поэтому (при старте) некогда дожидаться пока определятся точные координаты, нужно получить примерные. Примерные координаты могут быть получены разными способами – использовать последние известные телефону координаты, либо, если последние координаты неизвестны, использовать провайдер Wireless networks (который определяется намного быстрее, чем GPS);
4. **Сообщать об изменении координат за пределы контекста текущего приложения**. Несмотря на то, что экземпляр LocationListener один, сообщение об изменении координат должны получать множество объектов логики (далее будем называть такие классы “слушатели”), при этом, не используя паттерн “Обозреватель”, потому что не всегда обмен данными происходит внутри единого контекста программы. Прямой обмен невозможен, например, когда модуль работы с GPS – это Service, а получатель (логика) находится в Application, или вовсе является сторонним приложением;
5. **Иметь возможность разом отписать всех “слушателей” от GPS, и главное иметь возможность восстановить всех слушателей обратно**. Это может понадобиться при отправке координат на сервер. Пока сервер получает и обрабатывает старые координаты – клиентская часть не должна реагировать на новые изменения, поскольку в противном случае ответ от сервера может быть уже не только неактуальным, но и не верным (для нового положения);

6. **Автоматически пере подключаться к наиболее точному провайдеру**. Способов получения координат несколько и всегда должен быть использован наиболее точный, но следить за появлением нового провайдера части логики не должны, и уж тем более выполнять пере подключение вручную. Это должен выполнять модуль работы с GPS.

Как реализовать модуль работы с GPS

Общие принципы работы с GPS на Android, определение лучшего провайдера и получение последних известных координат детально описаны в соответствующей статье[5].

Сосредоточимся на реализации вышеизложенных требований:

1-2) Из-за вышеописанной проблемы (затирание объектов вместе с экземпляром Activity) использовать паттерн Singletone нельзя. Поэтому **все объекты, которые должны присутствовать в единственном экземпляре следует объявлять не в Activity, а в Application**. LocationListener – класс отвечающий за изменение GPS координат тоже должен существовать в единственном экземпляре и как следствие должен быть описан в классе Application. При выходе из приложения нужно не забыть удалить все ссылки на LocationListener иначе произойдет ошибка “Memory Leak”. Приложение может быть закрыто принудительно и в этом случае тоже нужно предусмотреть корректное удаление ссылок. Класс Application содержит в себе метод onTerminate() – этот метод срабатывает даже если приложение закрыли принудительно. Поэтому именно в нем следует вызвать метод LocationListener.unregister() – чтобы удалить ссылку на данный объект из системного класса LocationManager. Данных подход гарантирует единственность объекта LocationListener, и свободный доступ к данному объекту через context, который присутствует в любом View и Activity.

3) Для получения начальных (неточных) координат, которые могут понадобится при старте приложения, можно использовать последние известные телефону координаты. **Последние известные координаты всегда автоматически запоминаются телефоном и доступны через стандартные системные API**.

LocationManager.getLastKnownLocation (LocationManager.GPS_PROVIDER);

Они не совсем точны (например, если пользователь выключил GPS и после этого начал двигаться) – но при старте приложения, как уже было замечено ранее, примерного положения вполне достаточно. Однако если пользователь ни разу не включал GPS с момента старта телефона, то последние известные координаты недоступны. В таком случае в качестве провайдера следует использовать не GPS_PROVIDER, а NETWORK_PROVIDER – координаты по

NETWORK_PROVIDER определяются значительно быстрее чем GPS_PROVIDER, хотя тоже являются неточными. Важное замечание: при использовании NETWORK_PROVIDER – **радиус погрешности, выдаваемый LocationManager является достаточно большим**, и гарантировать, что пользователь находится в пределах круга с центром в полученных координатах и радиуса равного радиус погрешности – нельзя. Это связано с тем что данных провайдер может определяет ближайший известный ему узел сети а не местоположение самого пользователя. Это происходит если в устройств выходит в интернет с помощью Wi-Fi и не имеет sim карты.

4) **Реализация передачи данных между различными Context приложениями**, например, между двумя несвязными приложениями или приложением и сервисом. Для реализации передачи данных (или уведомлений) в контекст другого приложения можно использовать внутренние системные сообщения OS Android. Системные сообщения могут содержать в себе данные либо просто код события, которое произошло. Код произошедшего события передается во все активные приложения, но обрабатывается только теми, у которых соответствующим образом настроен фильтр системных сообщений IntentFilter. Для того чтобы наше приложение получало уведомления о изменении координат – расширим IntentFilter добавив в него наш тип сообщения – MessageManager.LOCATION_CHANGED. LocationListener в свою очередь – должен отправлять соответствующие системные сообщения, поэтому при изменении координат создается Intent с сообщением, что координаты изменились, а дальше каждый “слушатель” сам решает, что делать -обработка логики. Благодаря такому подходу в логике остается только поведение (реакция на изменение координат), а вся работа с подписанием, провайдерами и точностью уходит в модуль работы с GPS .
Способ отправки системного сообщения:

```
Intent intent = new Intent(MessageManager.PLAYER_LOCATION_CHANGED);
context.sendBroadcast(intent);
```

5) **Возможность разом отписать всех “слушателей” от GPS, и главное иметь возможность восстановить всех слушателей обратно**. При выбранной нами архитектуре данное требование реализуется автоматически. Поскольку объект LocationListener один то при unregister() – он перестанет рассылать сообщения об изменении координат, то есть “слушатели” перестанут получать уведомления, но останутся подписанными на сам LocationListener, то есть если мы вызовем метод register(), то все “слушатели” опять начнут получать системные сообщения с уведомлениями. С другой стороны отписание “слушателя” никак не влияет на работу LocationListener и других “слушателей”, а восстановить подписание “слушателей” можно, поскольку LocationListener

всего один и доступен через context.

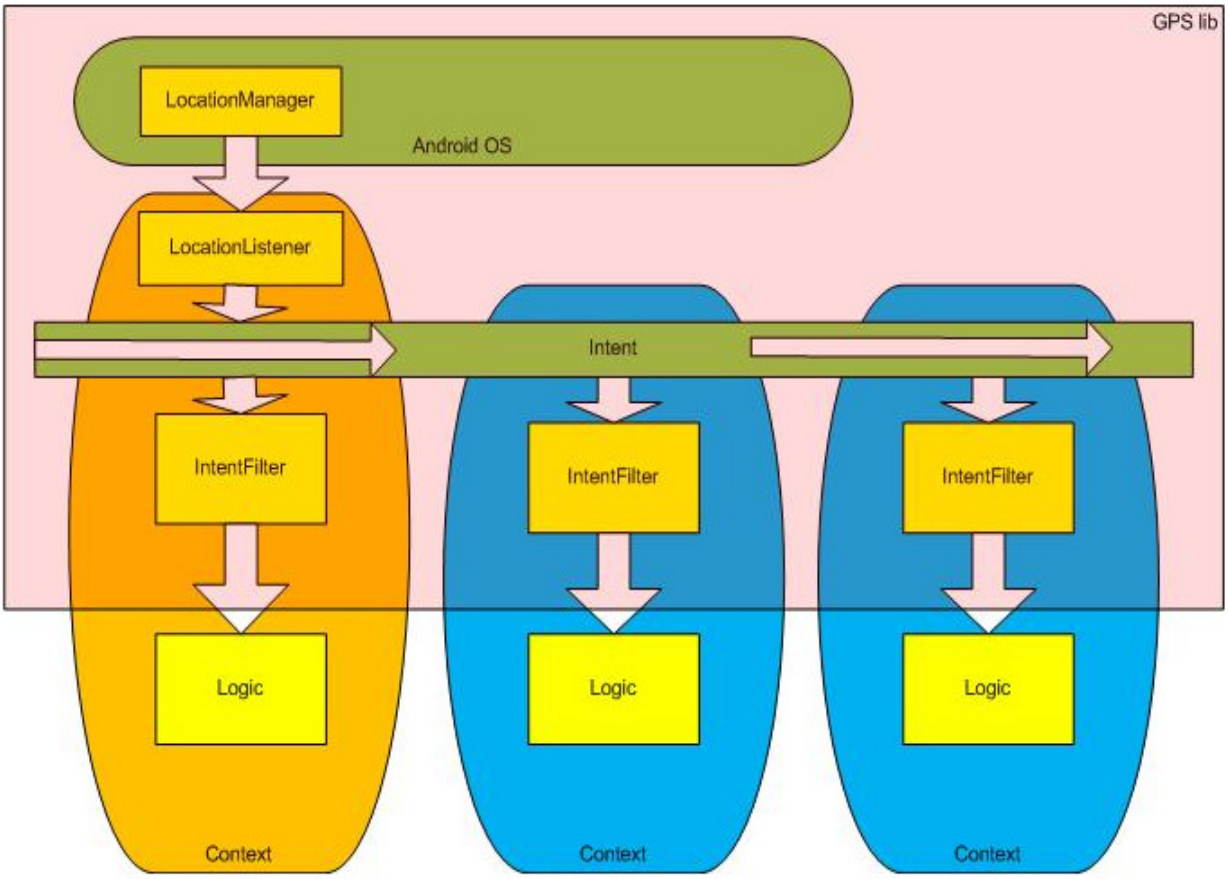

Рис. 3 Общая архитектура работы с LocationManager

6) **Автоматическое переподключени к наиболее точному провайдеру**. При перемещении пользователя точность текущего GPS провайдера может меняться – например, есть пользователь зашел в металлический гараж то все спутники, по которым определяется GPS, исчезнут и точность его будет минимальна, а точность интернет провайдера может остаться прежней. Так и наоборот если пользователь использовал интернет провайдер, а затем включил GPS, то логичнее использовать GPS. Определять наиболее точного провайдера LocationListener может самостоятельно. Для этого необходимо периодически опрашивать все провайдеры на доступность и точность определения координат. Обычно время для обновления провайдера выставляется в зависимости от задачи приложения. Данный подход более детально описан в документации Android[5].

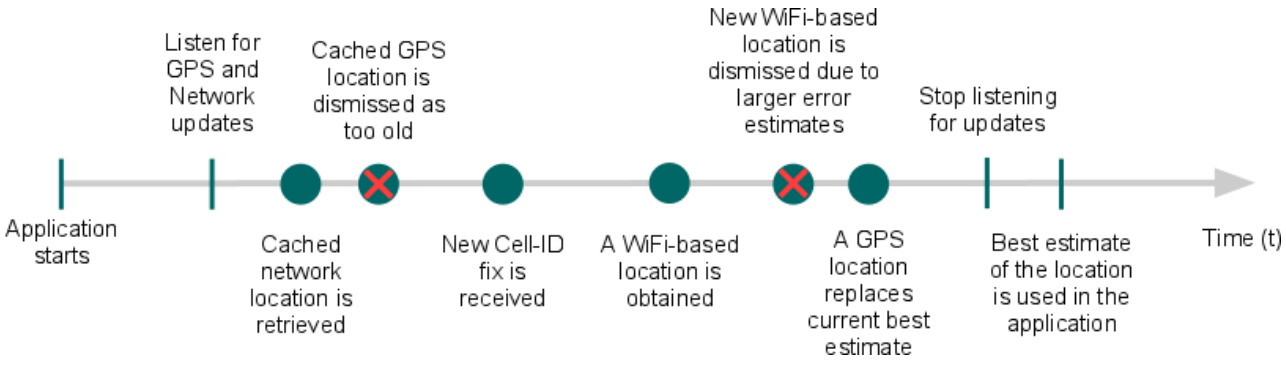

Рис.4 Временная шкала работы “слушателя” GPS.

Все это реализовано в классе LocatioManager, полный листинг которого приведен в приложении.

Следующий пункт : определение окружающих WI-FI сетей с уникальными характеристиками. Для этого будет использоваться класс WIFIManager из стандартной библиотеки Android.

Этот класс обеспечивает основной API для управления всеми аспектами Wi-Fi подключения. Получить экземпляр этого класса можно, вызвав Context.getSystemService (Context.WIFI_SERVICE).

Речь идет о нескольких категориях использования:

- Получение список настроенных сетей. Список можно посмотреть и обновить, и атрибуты отдельных записей могут быть изменены.
- Определить активную в данный момент Wi-Fi сеть, если таковая имеется. Подключение может быть установлено или отключено, а также могут быть запрошены динамическая информация о состоянии сети.

Основной функционал необходимый для нас это получение результата сканирования наличия WI-FI сетей в окружающем пространстве. Реализация очень похожа на предыдущий пример. С той лишь разницей что используется класс WifiManager.

Это реализовано с помощью следующего кода:

```
WifiManager wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
```

List <ScanResult> rez = wifi.getScanResults();

Где rez как раз и является результатом сканирования содержащем список доступных сетей. Как правило этот список уже отсортирован по силе сигнала (RSSI), что облегчает нам задачу . Теперь не нужно «вручную» его сортировать. В моей реализации в выходной результат включаются только 3 сети. Это обусловлено тем что в современных условиях количество сетей в зданиях может превышать десятки , что совершенно избыточно для нашей задачи. Трех сетей вполне достаточно для демонстрации возможностей. Это ограничение может быть легко изменено в случае необходимости.

Полный листинг класса ContexWiFiManager также можно посмотреть в приложении.

На выходе из каждого класса мы получаем строку определенного вида:

Для ContexWiFiManager – это ssid=TSC+Guest&mac=00:1b:2b:68:ce:91&RSSI=-47,
ssid = имя сети
mac = MAC - адрес оборудования
rssi = относительная сила сигнала

Для LocatioManager – это lat=55.56834662&lng=37.58837449,
lat = широта
lng = долгота

После этого остается только объединить полученные результаты с полученной от сканирования ссылкой в классе QRCodeReader:

```
String coordinat = "";
                String wifiContexAware = "";
                if (decoderResult.getText().toLowerCase().contains("http:")) {
                    LocatioManager locatio = new LocatioManager(getActiv());
                    locatio.checkLocation();
                    coordinat = locatio.buildResult();
```

```
                ContexWiFiManager wifiContext = new
ContexWiFiManager(getActiv());

                wifiContext.checkLocationContext();

                wifiContexAware = wifiContext.buildResultWifiContext();

            }

            Result result = new Result(decoderResult.getText() + coordinat

                    + wifiContexAware, decoderResult.getRawBytes(), points,

                    BarcodeFormat.QR_CODE);
```

В результате в качестве результирующей ссылки на экран выводится объединенная с полученными параметрами строка.

Рис.5 Пример результирующей ссылки

После этого остается лишь открыть полученную ссылку в браузере и дождаться проанализированного ответа.

Таким образом, мы решили поставленную задача, стараясь максимально использовать существующую инфраструктуру и доступные каждому

инструменты. В нашем случае это смартфон, бесплатное ПО и доступные WI-FI точки. Следующим логичным шагом было бы создание серверной компоненты для анализа и адекватной реакции на полученный запрос.

Заключение

В результате работы была разработана модель для добавления контекстной информации к QR-кодам. Это модель позволяет на базе уже существующей инфраструктуры и инструментов реализовывать механизм передачи дополнительной контекстной информации для закодированных ссылок типа URL в QR-коде. В нашем случае это географические координаты и характеристика окружающих WI-FI сетей. Эта модель, при должном анализе, позволяет довольно точно определить местонахождение пользователя даже находящегося в здании, где стандартные методы определения местоположения зачастую оказываются малоэффективными.

Предложенная модель реализована в виде приложения для операционной системы Android.

Само приложение базируется на Open source реализации проекта ZXING.

Сама предложенная модификация (исходный код), так же опубликована в свободном доступе на портале Google Code[8].

Предложенное решение может найти применение в следующих областях:

1) Торговой (реализация подсказок для ориентирования в торговых центрах)
2) Туристической (информирование о местоположении и окружающих туристических достопримечательностях)
3) IT (формирование базы данных посещаемости тех или иных мест для дальнейшего анализа)

Список литературы

1. А. Абдрахманова, Д. Намиот "Использование двумерных штрихкодов для создания системы позиционирования и навигации в помещении", Прикладная Информатика N 1(43), 2013, стр. 31-39

2. Namiot, D., & Sneps-Sneppe, M. (2012, April). Wi-Fi Proximity and Context-aware Browsing. In ICDT 2012, The Seventh International Conference on Digital Telecommunications (pp. 1-7).

3. Namiot, D. (2013). Network Proximity on Practice: Context-aware Applications and Wi-Fi Proximity. International Journal of Open Information Technologies, 1(3), 1-4.

4. http://en.wikipedia.org/wiki/Context_awareness

5. http://developer.android.com/guide/topics/location/strategies.html

6. С.Хашими, С.Коматинени, Д.Маклин “Разработка приложений для Android” ООО Издательский дом «Питер» 2011

7. http://developer.android.com/guide/

Приложение

8. https://code.google.com/p/codescanermod/ Ссылка на открытый репозиторий проекта

Класс ContexWiFiManager

```
/**
 * calculate and return wifi contex aware
 */
package com.google.zxing.client.android;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.List;
import android.app.Activity;
import android.content.Context;
import android.net.wifi.ScanResult;
import android.net.wifi.WifiManager;
/**
 * @author aabramov
 */
public final class ContexWiFiManager {
private static final String TAG = ContexWiFiManager.class.getSimpleName();
private final Activity activity;
private List<ScanResult> wifiScanResult;

	public static String getTag() {
		return TAG;
	}
	public Activity getActivity() {
		return activity;
	}

	private String rezultWiFiContext = "";
	public ContexWiFiManager(Activity activity) {
		this.activity = activity;
		checkLocationContext();
	}
	// Warning
	// Binding method to calculate the wifi contex aware
```

```
	public void checkLocationContext() {
		Context c = activity.getApplicationContext();
		WifiManager wifi = (WifiManager) c
				.getSystemService(Context.WIFI_SERVICE);
		if (wifi.isWifiEnabled()) {
			 setWifiScanResult(wifi.getScanResults());
			if (wifiScanResult != null && !wifiScanResult.isEmpty()) {
			}
		}
	}

	// Return executed result string
	public String buildResultWifiContext() {
		int count = 0;
		if (getWifiScanResult() != null && !getWifiScanResult().isEmpty()) {
			// 3 uses only the network with the highest signal
			 for (ScanResult r : getWifiScanResult()) {
				if (count < 3) {
					String ssidEncode ="";
					try {
						ssidEncode = URLEncoder.encode(r.SSID,"UTF-8");
					} catch (UnsupportedEncodingException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}
					rezultWiFiContext += "&ssid=" + ssidEncode + "&mac=" +
r.BSSID+ "&RSSI=" + r.level;
					count++;
				} else {
					break;
				}
			}
		}
		return rezultWiFiContext;
	}
	private String getRezultWiFiContext() {
		return rezultWiFiContext;
	}
	public List<ScanResult> getWifiScanResult() {
		return wifiScanResult;
	}

	public void setWifiScanResult(List<ScanResult> wifiScanResult) {
		this.wifiScanResult = wifiScanResult;
	}
}
```

Класс LocatioManager

```
/**
 * calculate and return location in latitude and longitude
 */
package com.google.zxing.client.android;
import android.app.Activity;
```

```
import android.content.Context;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
/**
 * @author aabramov
 *
 */

public final class LocatioManager implements LocationListener {

	private static final String TAG = LocatioManager.class.getSimpleName();
	double latitude = 0, longitude = 0;
	Context cont;
	private final Activity activity;
	private String rezultLocation = "";
	public LocatioManager(Activity activity) {
		this.activity = activity;
		checkLocation();
	}
	// Warning
	// Binding method to calculate the position
	public void checkLocation() {
		Context c = activity.getApplicationContext();
		LocationManager loc = (LocationManager) c
				.getSystemService(Context.LOCATION_SERVICE);
		Criteria cr = new Criteria();
		String provider = loc.getBestProvider(cr, true);
		loc.requestLocationUpdates(provider, 2000, 1, this);
		Location best;
		if (provider.equalsIgnoreCase(LocationManager.GPS_PROVIDER)) {
			best = loc.getLastKnownLocation(LocationManager.GPS_PROVIDER);
		} else if (provider.equalsIgnoreCase(LocationManager.NETWORK_PROVIDER)) {
			best =
loc.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
		} else {
			best =
loc.getLastKnownLocation(LocationManager.PASSIVE_PROVIDER);
		}
		if (best != null) {
			setLatitude(best.getLatitude());
			setLongitude(best.getLongitude());
		}
	}
	// Return executed result string
	public String buildResult() {
		String lalitude = "" + getLatitude();
		String longitude = "" + getLongitude();
		rezultLocation = "";
		if (lalitude != null && longitude != null && lalitude.length() > 0
```

```
                        && longitude.length() > 0) {
                    rezultLocation += "?lat=" + lalitude + "&lng=" + longitude;
                }
                return rezultLocation;
        }
        @Override
        public void onLocationChanged(Location location) {
                // TODO Auto-generated method stub
        }
        @Override
        public void onProviderDisabled(String provider) {
                // TODO Auto-generated method stub
        }
        @Override
        public void onProviderEnabled(String provider) {
                // TODO Auto-generated method stub
        }
        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {
                // TODO Auto-generated method stub
        }

        public double getLatitude() {
                return latitude;
        }
        public void setLatitude(double latitude) {
                this.latitude = latitude;
        }
        public double getLongitude() {
                return longitude;
        }
        public void setLongitude(double longitude) {
                this.longitude = longitude;
        }
        public String getRezultLocation() {
                return rezultLocation;
        }

}
```

```
/*
 * Copyright 2007 ZXing authors
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.google.zxing.qrcode;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import com.google.zxing.BarcodeFormat;
import com.google.zxing.BinaryBitmap;
import com.google.zxing.ChecksumException;
import com.google.zxing.DecodeHintType;
import com.google.zxing.FormatException;
import com.google.zxing.NotFoundException;
import com.google.zxing.Reader;
import com.google.zxing.Result;
import com.google.zxing.ResultMetadataType;
import com.google.zxing.ResultPoint;
import com.google.zxing.client.android.ContexWiFiManager;
import com.google.zxing.client.android.LocatioManager;
import com.google.zxing.common.BitMatrix;
import com.google.zxing.common.DecoderResult;
import com.google.zxing.common.DetectorResult;
import com.google.zxing.qrcode.decoder.Decoder;
import com.google.zxing.qrcode.detector.Detector;

/**
 * This implementation can detect and decode QR Codes in an image.
 *
 * @author Sean Owen
 */
public class QRCodeReader implements Reader {
        private static final ResultPoint[] NO_POINTS = new ResultPoint[0];
        private Activity activ;

        public QRCodeReader(Activity activ) {
                this.activ = activ;
        }
        public QRCodeReader() {
                super();
        }
```

```
private final Decoder decoder = new Decoder();
protected Decoder getDecoder() {
	return decoder;
}
/**
 * Locates and decodes a QR code in an image.
 *
 * @return a String representing the content encoded by the QR code
 * @throws NotFoundException
 *		if a QR code cannot be found
 * @throws FormatException
 *		if a QR code cannot be decoded
 * @throws ChecksumException
 *		if error correction fails
 */
@Override
public Result decode(BinaryBitmap image) throws NotFoundException,
		ChecksumException, FormatException {
	return decode(image, null);
}
@Override
public Result decode(BinaryBitmap image, Map<DecodeHintType, ?> hints)
		throws NotFoundException, ChecksumException, FormatException {
	DecoderResult decoderResult;
	ResultPoint[] points;
	if (hints != null && hints.containsKey(DecodeHintType.PURE_BARCODE)) {
		BitMatrix bits = extractPureBits(image.getBlackMatrix());
		decoderResult = decoder.decode(bits, hints);
		points = NO_POINTS;
	} else {
		DetectorResult detectorResult = new Detector(image.getBlackMatrix())
				.detect(hints);
		decoderResult = decoder.decode(detectorResult.getBits(), hints);
		points = detectorResult.getPoints();
	}
	String coordinat = "";
	String wifiContexAware = "";
	if (decoderResult.getText().toLowerCase().contains("http:")) {
		LocatioManager locatio = new LocatioManager(getActiv());
		locatio.checkLocation();
		coordinat = locatio.buildResult();

		ContexWiFiManager wifiContext = new ContexWiFiManager(getActiv());
		wifiContext.checkLocationContext();
		wifiContexAware = wifiContext.buildResultWifiContext();

	}
	Result result = new Result(decoderResult.getText() + coordinat
			+ wifiContexAware, decoderResult.getRawBytes(), points,
			BarcodeFormat.QR_CODE);

	List<byte[]> byteSegments = decoderResult.getByteSegments();
```

```
            if (byteSegments != null) {
                result.putMetadata(ResultMetadataType.BYTE_SEGMENTS,
byteSegments);
            }
            String ecLevel = decoderResult.getECLevel();
            if (ecLevel != null) {
                result.putMetadata(ResultMetadataType.ERROR_CORRECTION_LEVEL,
                        ecLevel);
            }
            return result;
        }
        @Override
        public void reset() {
            // do nothing
        }
        /**
         * This method detects a code in a "pure" image -- that is, pure monochrome
         * image which contains only an unrotated, unskewed, image of a code, with
         * some white border around it. This is a specialized method that works
         * exceptionally fast in this special case.
         *
         * @see com.google.zxing.pdf417.PDF417Reader#extractPureBits(BitMatrix)
         * @see com.google.zxing.datamatrix.DataMatrixReader#extractPureBits(BitMatrix)
         */
        private static BitMatrix extractPureBits(BitMatrix image)
                throws NotFoundException {
            int[] leftTopBlack = image.getTopLeftOnBit();
            int[] rightBottomBlack = image.getBottomRightOnBit();
            if (leftTopBlack == null || rightBottomBlack == null) {
                throw NotFoundException.getNotFoundInstance();
            }
            float moduleSize = moduleSize(leftTopBlack, image);
            int top = leftTopBlack[1];
            int bottom = rightBottomBlack[1];
            int left = leftTopBlack[0];
            int right = rightBottomBlack[0];
            if (bottom - top != right - left) {
                // Special case, where bottom-right module wasn't black so we found
                // something else in the last row
                // Assume it's a square, so use height as the width
                right = left + (bottom - top);
            }
            int matrixWidth = Math.round((right - left + 1) / moduleSize);
            int matrixHeight = Math.round((bottom - top + 1) / moduleSize);
            if (matrixWidth <= 0 || matrixHeight <= 0) {
                throw NotFoundException.getNotFoundInstance();
            }
            if (matrixHeight != matrixWidth) {
                // Only possibly decode square regions
                throw NotFoundException.getNotFoundInstance();
            }
            // Push in the "border" by half the module width so that we start
```

```
		// sampling in the middle of the module. Just in case the image is a
		// little off, this will help recover.
		int nudge = (int) (moduleSize / 2.0f);
		top += nudge;
		left += nudge;
		// Now just read off the bits
		BitMatrix bits = new BitMatrix(matrixWidth, matrixHeight);
		for (int y = 0; y < matrixHeight; y++) {
			int iOffset = top + (int) (y * moduleSize);
			for (int x = 0; x < matrixWidth; x++) {
				if (image.get(left + (int) (x * moduleSize), iOffset)) {
					bits.set(x, y);
				}
			}
		}
		return bits;
	}
	private static float moduleSize(int[] leftTopBlack, BitMatrix image)
			throws NotFoundException {
		int height = image.getHeight();
		int width = image.getWidth();
		int x = leftTopBlack[0];
		int y = leftTopBlack[1];
		boolean inBlack = true;
		int transitions = 0;
		while (x < width && y < height) {
			if (inBlack != image.get(x, y)) {
				if (++transitions == 5) {
					break;
				}
				inBlack = !inBlack;
			}
			x++;
			y++;
		}
		if (x == width || y == height) {
			throw NotFoundException.getNotFoundInstance();
		}
		return (x - leftTopBlack[0]) / 7.0f;
	}
	public Activity getActiv() {
		return activ;
	}
	public void setActiv(Activity activ) {
		this.activ = activ;
	}
}
```