Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Магистерская программа
«Программное обеспечение вычислительных сетей»

Магистерская диссертация

СЕРВИСЫ СЕТЕВОЙ БЛИЗОСТИ НА БАЗЕ БЕСПРОВОДНЫХ СЕТЕЙ

Работу выполнил
студент Горлов Андрей Олегович

Научный руководитель:
с.н.с., к.ф.-м.н. **Намиот Дмитрий Евгеньевич**

подпись научного руководителя

Москва
2015

Содержание

Аннотация **3**

Введение **4**

Постановка задачи **6**

1. Обзор существующих систем **7**

1.1. Сетевые протоколы 7

1.2. Решаемые задачи 10

1.3. Примеры систем 10

2. Исследование и построение архитектуры сервиса **18**

2.1. Элементы систем 18

2.2. Сетевая близость и ее свойства 21

2.3. Описание архитектуры сервиса 26

3. Описание практической части **38**

3.1. Использованные технологии 38

3.2. Серверная часть 40

3.3. Клиентская часть 45

Заключение **52**

Список литературы **53**

Аннотация

В работе рассматриваются вопросы разработки сервисов, использующих сетевую близость в средах беспроводной передачи данных для решения актуальных задач. Основной проблемой разработки таких сервисов можно считать выбор подходящей архитектуры и технологий, соответствующих поставленной перед сервисом задаче, а также преодоление ограничений, обусловленных таким выбором. Для выработки решения указанной задачи в работе рассматриваются существующие подходы к моделированию систем, использующих свойства сетевой близости, а также предлагается архитектура для таких сервисов. В заключительной части работы приводится описание разработанной архитектуры и прототипа, воплощающего ее основные идеи.

Введение

С каждым годом количество мобильных устройств стремительно увеличивается, некоторые из них уже стали занимать ведущее положение в нишах, в которых ранее долгое время использовались только персональные компьютеры. Круг задач, решаемый сегодня мобильными устройствами, постоянно растет, при этом подавляющее количество персональных мобильных устройств имеют встроенные модули беспроводной связи, работающие по ставшим уже привычными протоколам, к которым относятся, например, Bluetooth и Wi-Fi.

Одной из развивающихся является область построения решений на основе сетевой близости в беспроводных сетях. Сетевая близость — это свойство беспроводных сетей, которое определяет условие видимости устройств, находящихся в одной радио-среде. С использованием данного свойства решается ряд задач, которые относятся к классу контекстно-ориентированных. Для этого создаются сервисы, связывающие мобильные устройства тем или иным способом с некоторой информацией, окружающей их. Сетевая близость в решении такого рода задач играет немаловажную роль, предоставляя таким сервисам дополнительные возможности, которые ранее были труднодостижимы или отсутствовали.

В рассматриваемых в работе вопросах освещаются технологии, которые стали в последнее время обретать популярность в применении к описываемым задачам. Это такие технологии беспроводных сетей, как Bluetooth Low Energy, а также основанный на ней iBeacon, в которых устройства играют роль своеобразных меток, или тегов, позволяющих сторонним устройствам, находящимся в зоне действия беспроводной сети,

получать о них некоторую заранее определенную информацию. В работе рассматриваются решения, использующие данные технологии, а также описывается собственный альтернативный им подход.

Постановка задачи

В области построения сервисов, использующих свойство сетевой близости, есть ряд задач, решение которых требует особых подходов к проектированию программного обеспечения. Промышленные технологии, предлагаемые для решения этих задач, накладывают ограничения на разработку этого программного обеспечения. К этой проблеме также добавляется необходимость разрабатывать отдельные приложения для каждой задачи, стоящей перед сервисами такого рода.

Решение указанных проблем позволяет облегчить разработку сервисов, основанных на сетевой близости, делает их более доступными для внедрения, модификации, использования в сторонних проектах.

Предлагаемое в работе решение должно удовлетворять требованиям доступности, простоты в реализации, эффективности.

Целью данной работы является разработка архитектуры сервисов сетевой близости на базе беспроводных сетей. Для решения поставленной цели необходимо рассмотреть существующие на данный момент системы, использующие свойство сетевой близости, исследовать их особенности, проанализировать недостатки, описать архитектуру сервисов такого рода, а также разработать прототип, служащий подтверждением работоспособности предлагаемой архитектуры.

1. ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ

1.1. Сетевые протоколы

Для реализации сервисов сетевой близости подходит любой протокол беспроводной передачи данных малого радиуса действия. К таким протоколам относятся:

1. Wi-Fi — технология построения локальных вычислительных сетей, позволяющая обмениваться данными между устройствами посредством беспроводного сигнала, специфицированная в стандарте IEEE 802.11[15];

2. Bluetooth — стандарт беспроводной технологии передачи данных, который позволяет осуществлять коммуникацию беспроводных устройств на коротких дистанциях и строить так называемые персональные сети (Personal Area Networks, PAN);

3. Bluetooth Low Energy (Bluetooth LE, BLE), также известный как Bluetooth 4.0[16], или Bluetooth Smart, - относительно новая спецификация технологии Bluetooth, разработанная Bluetooth Special Interest Group (SIG), определяющая недорогое решение Bluetooth с низким энергопотреблением;

4. ZigBee — спецификация протоколов высокого уровня, основанных на стандарте IEEE 802.15[14] и используемых для построения персональных сетей, состоящих из маленьких цифровых радиоточек с низким энергопотреблением. Протокол ZigBee часто используют для построения сенсорных сетей[13], т. к. решение на основе этого протокола получается недорогим, узлы такой сети, требующие небольшого количества электроэнергии, хорошо подходят для длительной автономной работы.

Приведем сравнение описанных выше технологий беспроводных

сетей [4]:

	ZigBee	Wi-Fi	Bluetooth	Bluetooth LE
Поддерживаемая топология	Звезда, ячеистая	Звезда	Звезда	Звезда, шина
Частота	868 MHz (Европа) 915 MHz (Северная Америка) 2.4 GHz	2.4/5 GHz	2.4 GHz	2.4 GHz
Пропускная способность	250 Kbps	11/54 Mbps	1/3 Mbps	1 Mbps
Диапазон	от 10 до 100 м	до 100 м	до 10 м	до 40 м
Потребление энергии	Очень низкое	Высокое	Низкое	Очень низкое
Время жизни батареи	Несколько лет	Несколько часов	Несколько недель	Несколько месяцев
Стоимость	Низкая	Высокая	Средняя	Низкая
Поддержка смартфонами	Нет	Да	Да	Да
Разработан для позиционирования	Нет	Нет	Нет	Да
Точность	3-5 м	5-10 м	2-5 м	1-2 м
Применения	Промышленный контроль и мониторинг, сенсорные сети	Беспроводные локальные сети	Обмен данными между устройствами	Сенсоры, позиционирование, периферия

Таблица 1. Сравнение беспроводных технологий.

В последнее время набирают популярность сетевые технологии, которые используют свойство локальности, или так называемой сетевой близости, в беспроводных сетях. К таким технологиям относится, прежде всего, технология iBeacon[17], выпущенная не так давно компанией Apple, а также Google Nearby[11] и Samsung Placedge[10]. Все они используют в своей основе рассмотренную выше спецификацию протокола передачи данных Bluetooth Low Energy, наиболее существенным достоинством которой отмечают ее сверхнизкие требования к энергопотреблению. Многие устройства, выполненные по спецификации Bluetooth Low Energy (Bluetooth LE), могут работать на одной батарейке в течение года без необходимости подзарядки [9]. Низкое энергопотребление не единственное достоинство, которое обеспечивает растущую популярность устройствам, созданным с использованием Bluetooth LE, т.к. технологии со схожими требованиями к питанию существовали и до появления Bluetooth LE, например, указанный выше ZigBee [7]. Важным преимуществом, служащим сегодня распространению технологий, основанных на Bluetooth LE, является повсеместное использование сетевого стека Bluetooth: модулями Bluetooth оснащены сегодня мобильные устройства, компьютеры, различные устройства ввода, манипуляторы, автомобили и т.д. Вместе с тем стоит отметить, что традиционный Bluetooth всё еще является более популярным в сравнении с Bluetooth LE. И как будет описано далее, предлагаемая в работе архитектура сервисов, использующих свойства сетевой близости, также в качестве демонстрационного примера применяет привычный стек протоколов Bluetooth, что, впрочем, никоим образом ее не ограничивает.

1.2. Решаемые задачи

С помощью технологий, использующих свойство сетевой близости, решают сегодня различные классы задач:

1. Определение местоположения внутри помещений;
2. Предоставление дополнительной, зависящей от местоположения информации клиентам магазинов, посетителям музеев, выставок и т.д.;
3. Использование в розничной торговле для размещения рекламы, акций и купонов[5];
4. Информирование в системах общественного транспорта[8];
5. Аутентификация клиентов, предоставление доступа к объектам, либо к управлению ими[4];
6. Различные задачи для решений "умного города";
7. Системы платежей в местах оплаты, не требующих платежных карт и наличных средств, и др. [4]

1.3. Примеры систем

Сравним несколько существующих систем, использующих сетевую близость. Целью такого сравнения является выявление схожих черт существующих систем, решений и технологий, лежащих в их основе, а также основных недостатков, которым далее будет уделено особое внимание в архитектуре, предлагаемой в данной работе. В качестве критериев сравнения выберем естественные свойства программных решений и их моделей, присущие любым системам, в том числе и

описанным ниже. К таким критериям относятся: выбор технологий, составные части моделей, их функциональное назначение, роли, ограничения представленных моделей, применимость для решения указанного выше ряда задач.

В качестве первого примера рассмотрим SITA Common Use Beacon Registry (Схема 1)[8]. SITA Common Use Beacon Registry — это реестр iBeacon-ов общего пользования в авиатранспортной отрасли, основан на технологии iBeacon компании Apple. В технологии iBeacon beacon — это пакет специально разработанного компанией Apple протокола, работающего поверх Bluetooth LE. В этом пакете (beacon-е), который транслирует специальная радио-точка, называемая часто беспроводной меткой (здесь также названная beacon-ом), основными данными являются три числа, одно из которых — UUID (уникальный идентификатор) компании-владельца beacon-ов. Таким образом beacon-ы отделяют друг от друга по указанным в них UUID-ам компаний, эти UUID-ы не должны пересекаться. Технология iBeacon будет описана далее более подробно. Компания SITA предоставляет ряд API для доступа к своим сервисам, один из которых — реестр beacon-ов, предлагающий следующие возможности:

- владельцам beacon-ов (компаниям: т. к. сервис предназначен для компаний, осуществляющих авиаперевозки, то компаниями могут быть авиакомпании, аэропорты или третьи лица) предоставляется возможность управлять их beacon-инфраструктурой и отслеживать, где расположены beacon-ы в аэропорту;
- позволяет аэропортам производить мониторинг beacon-ов для предотвращения помех из-за размещенных в аэропорту точек доступа Wi-Fi;
- предоставляет владельцам beacon-ов доступ к простому механизму

добавления метаданных к beacon-ам;

- предоставляет API разработчикам, которые хотят создавать приложения, нацеленные на оказание авиатранспортных услуг, услуг в области перевозок, а также услуг в связанных с ними отраслях.

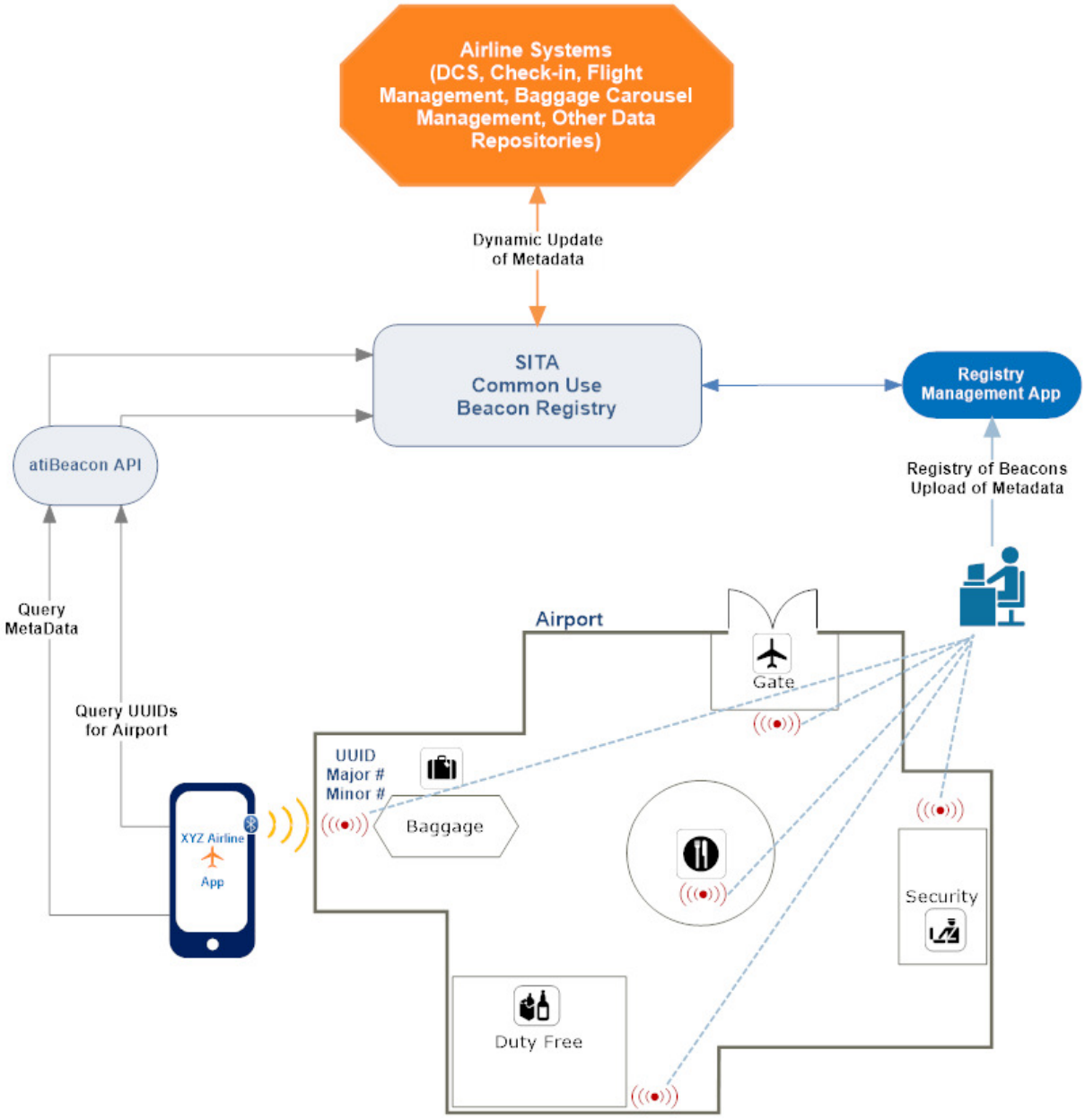

Схема 1. Схема взаимодействия с реестром

На схеме изображена модель взаимодействия с реестром через предоставленный компанией API.

Следующим примером рассмотрим систему Navizon Proximity Engine[18]. Navizon Proximity Engine это система, спроектированная для решения задач с использованием геозон[19], техники, которая позволяет очертить виртуальную границу вокруг интересующей области и выполнять некоторые заранее определенные действия, когда известное мобильное устройство входит в эту область или покидает ее. В качестве примеров таких действий можно отметить отправку SMS сообщений на смартфоны, доставку медиа-контента на пользовательское устройство на выставке, открывание дверей, активированных Wi-Fi тегом, отправку сообщения электронной почты, или отображение персонализированных сообщений на электронной доске объявлений.

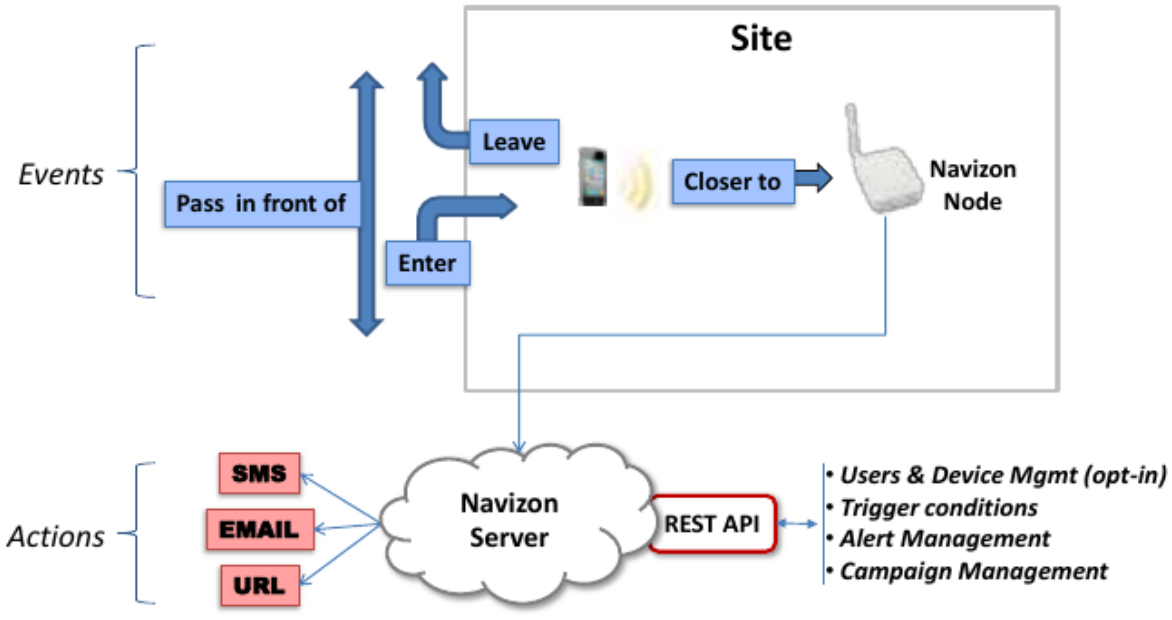

Схема 2. Схема функционирования Navizon Proximity Engine

В качестве беспроводной технологии в Navizon Proximity Engine используется Wi-Fi. Мобильные приложения не требуют установки программного обеспечения, однако Navizon Proximity Engine требует размещения в интересных для отслеживания местах специальных узлов аппаратного обеспечения (Navizon Node), подключенных к сети Интернет, а также облачного сервиса, который запускает на выполнение действия при наступлении заранее заданных условий. Узлы определяют активные Wi-Fi устройства, расположенные неподалеку, и периодически загружают на облачный сервис списки MAC-адресов обнаруженных устройств и уровень их сигнала. Сервис определяет известные устройства, запускает действия, основываясь на предопределенных условиях, определенных как Alert-ы (событие для одного устройства) или Campaign-ы (событие для группы устройств).

В качестве еще одного примера системы, использующей сетевую близость в беспроводных сетях, можно привести Guardly Platform[20]. Guardly Platform — решение для провайдеров аппаратного и программного обеспечения, желающих интегрировать в свои решения систему аварийного реагирования. Принцип работы системы, основанной на Guardly Platform, похож на принцип функционирования Navizon Proximity Engine. Среди потенциальных сфер применения своей системы разработчики отмечают производство устройств для медицинских нужд, системы обнаружения движений, решение для экстренного реагирования в чрезвычайных ситуациях и другие. Также как и в Navizon Proximity Engine, Guardly API использует в своей основе архитектуру REST.

В качестве последнего примера системы, использующей сетевую близость, рассмотрим решение Bluetooth Data Points (BDP)[1], предложенное в лаборатории Открытых Информационных Технологий

ВМК МГУ имени М.В. Ломоносова[21]. Система BDP является развитием идей, заложенных в проекте SpotEx [22], но использует в качестве среды передачи данных не Wi-Fi, как в SpotEx, а Bluetooth. Идея BDP состоит в связывании некоторого пользовательского контента с беспроводными метками (в данном случае с Bluetooth-метками, в отличие от SpotEx), причем контент может быть любой. В BDP рассматривается упрощенный вариант, демонстрирующий возможности решения: в качестве контента выступает URL, в качестве приложения, принимающего этот контент, - обычный браузер, или WebView, встроенная в приложение. Метки в BDP могут быть как существующими публичными Bluetooth-точками, так и специально созданными и запущенными на устройствах, в частности, на мобильных телефонах[23]. Привязка контента к меткам осуществляется в BDP за счет задания правил. Правила представляют собой предикаты, отвечающие за различные свойства меток, заданные относительно сканирующего устройства. Например, одним из основных свойств, отмеченных в описании BDP, является свойство видимости, т. е. если сканирующее устройство обнаружило метку с заданным SSID (Service Set Identifier, идентификатор метки), и существует правило, в котором указано, что в случае видимости метки с данным идентификатором необходимо показать соответствующий контент, то правило срабатывает, и указанный контент является искомым результатом. К одной метке в BDP можно привязывать несколько различных вариантов контента. Важным свойством решения в BDP отмечается отсутствие необходимости в соединении с Bluetooth-точками, а также возможность создания меток из обычных Bluetooth-устройств.

Сравним описанные выше решения, учитывая указанные в начале раздела критерии. В качестве технологий во всех перечисленных системах

используются распространенные беспроводные технологии, такие как Wi-Fi, Bluetooth или Bluetooth LE. При этом каждое из решений ориентировано только на один тип беспроводных сетей. Составные элементы систем схожи, имеют аналогичное назначение, однако некоторые из решений, например, такие как Navizon Proximity Engine, используют в качестве сканирующих устройств собственное аппаратное обеспечение, даже, несмотря на то, что в основе этого аппаратного обеспечения лежит технология Wi-Fi. Важным отличием в системах является «направление» сканирования меток, или то, какие именно устройства выступают в роли беспроводных меток, что позволяет определить две различные модели сканирования. Так, в Navizon Proximity Engine сканирующими устройствами являются стационарные точки, а пользовательские мобильные устройства являются метками, в то же время в остальных системах стационарные мобильные устройства выступают в качестве меток, что демонстрирует также и то, что такой подход является наиболее распространенным. В BDP нет ограничений относительно того, какие устройства выступают в роли сканирующих, а какие в роли меток. Важно отметить, что метками в данном случае являются пассивные устройства, которые сканируются другими устройствами независимо от их размещения, стационарно оно или нет. Все из выше перечисленных систем ограничены технологиями беспроводных сетей, которые они используют. Также к ограничениям относятся и узконаправленные решения в их реализациях, например, использование специализированных аппаратных средств в качестве меток. Важным является и то, что представленные системы не дают общего решения для нескольких из выше описанных задач, решаемых сервисами, использующими сетевую близость, а сосредоточены на каких-то одних, конкретных предметных областях, хотя попытка предложить более

общее решение содержится в описании дальнейшего направления исследований в рамках проекта BDP.

В данной работе в качестве основы для собственных разработок используются идеи, описанные в BDP. Выбор BDP обусловлен тем, что именно этот проект представляет собой наиболее гибкую модель системы, использующей сетевую близость. Так, в BDP в качестве меток может выступать любое Bluetooth-устройство, к любой метке можно привязать любой контент, не связанный с конкретной предметной областью, с каждой меткой может быть связано несколько вариантов контента.

2. ИССЛЕДОВАНИЕ И ПОСТРОЕНИЕ АРХИТЕКТУРЫ СЕРВИСА

2.1. Элементы систем

В существующих на данный момент системах, использующих свойство сетевой близости, устоялись некоторые решения, за элементами которых закреплены определенные роли. Так устройства, которые опрашивают другие устройства в беспроводной среде, осуществляя поиск, называют считывающими устройствами, или сканерами, те устройства, которые вещают, - транспондерами, передатчиками. Передатчики, которые выступают в роли меток в беспроводной среде, например, которые передают свой MAC-адрес и два числа Major и Minor в технологии iBeacon, также часто именуются тегами, или метками. Процесс поиска устройств сканером называют обнаружением устройств.

Считывающие устройства

Такими устройствами могут быть любые устройства, оснащенные модулями беспроводной связи, например, модулями Bluetooth. Ими могут быть компьютеры, планшеты, но чаще всего это коммуникаторы. На коммуникатор предварительно требуется установить специальную программу, осуществляющую обнаружение меток.

Метки

Например, в технологии iBeacon, имеющей сегодня наибольшую популярность из тех, которые используют Bluetooth LE, метки вещают с определенной частотой пакет, содержащий:

1. iBeacon префикс с информацией о том, что beacon принадлежит протоколу iBeacon;
2. UUID идентификатор - идентификатор, который позволяет отделить beacon-ы одной компании от beacon-ов другой;
3. Major - идентификатор, который может идентифицировать группу beacon-ов, например, принадлежащую некоторому магазину компании, - другой магазин тогда будет иметь другой идентификатор Major;
4. Minor - идентификатор, который может идентифицировать beacon в одной группе;
5. Measured Power - уровень сигнала метки, измеренный на расстоянии одного метра от нее.

При этом частота широковещательных сообщений может быть изменена, она часто составляет от 50 мс до 2 сек.

Процесс обнаружения

Этот процесс запускается сканирующим устройством, которое собирает обнаруженные метки.

Сделаем ряд важных замечаний на примере технологии iBeacon. Подход технологии Apple практически целиком перенимается реализациями других производителей, так что отдельных пояснений они не

требуют.

1. Стоит отметить, что метки iBeacon требуют установленного регламента назначения уникальных идентификаторов компаний, но что особенно важно, так это то, что метки нуждаются в поддержке специального протокола;
2. Метки реализуются в качестве специальных устройств, которым требуется монтаж;
3. Наиболее важным, пожалуй, является необходимость устанавливать специальное программное обеспечение на считывающее устройство, причем специальное в данном случае означает, что это программное обеспечение должно быть своё для каждой компании, которая использует технологию iBeacon.

Все эти пункты можно считать недостатками технологий iBeacon и аналогичных ей, поэтому далее будет рассматриваться подход, предложенный в проекте Bluetooth Data Points (BDP), изложенный в [1].

Отметим также важные отличия идей, изложенных в BDP, от технологии iBeacon:

1. Отсутствие необходимости в установке специального приложения для каждого вендора: для работы требуется только одно приложение, которое умеет читать все метки и получать по ним необходимую информацию;
2. В BDP используются метки, не требующие специального протокола: в качестве меток используются обычные пакеты идентификации устройства в стандартном протоколе Bluetooth;
3. В качестве транспондеров в BDP можно использовать любое устройство, оснащенное модулем Bluetooth, причем создать метку и запустить ее в работу можно программным путем, что позволяют

сделать современные мобильные операционные системы, достаточно включить Bluetooth устройство в режиме discoverable.

4. Сама метка в BDP данных не передает, данные о распространяемом с помощью меток содержимом хранятся на удаленном сервере, к которому сканирующее устройство самостоятельно получает доступ.

Далее предлагаемая в работе архитектура сервиса использует в качестве своей основы идеи, заложенные в BDP, так что дальнейшее описание модели сервисов, основанных на сетевой близости, можно считать развитием модели BDP. Основной использованной идеей проекта BDP, которая перенимается в данной работе, является использование в качестве беспроводных меток обычных устройств, оснащенных соответствующими модулями беспроводной связи. В отличие от BDP, где в качестве протокола беспроводной передачи данных предлагается использовать Bluetooth, в предлагаемой далее архитектуре ограничения на используемый протокол не налагаются.

2.2. Сетевая близость и ее свойства

Сетевой близостью можно назвать свойство беспроводных сетей, позволяющее некоторым образом определить отношение расстояния между узлами сети. За такое отношение часто принимают бинарную функцию, которая позволяет определить "видимость" одного устройства другим, т.е. устройство беспроводной сети способно определить, находится ли другое устройство в некоторой его окрестности или нет. В данном случае под окрестностью, или диапазоном видимости, строго говоря, понимают не

геометрическую окрестность, а окрестность устройства, в которой устройство способно демодулировать и декодировать пакет, посланный некоторым трансмиттером (передатчиком). Таким образом, предикат, определяющий отношение сетевой близости, возвращает 1, если устройство может получить пакет, отправленный другим устройством, и 0 в ином случае[3].

Важно отметить, что свойство сетевой близости доступно независимо от конкретной технологии построения беспроводных сетей. Это справедливо и для протокола Bluetooth, который используется далее в работе при реализации прототипа. Также при определении "видимости" одного устройства другим часто интересуются другим свойством, уровнем сигнала (RSS)[3]. Это свойство позволяет помимо определения близости устройств узнать степень этой близости. На точность определения степени близости устройств влияет множество факторов, в т.ч. и случайных, которые стараются учитывать в системах позиционирования внутри помещений (IPS)[3], [2].

Степень близости, использующую RSS источника сообщения, можно рассматривать как другой способ определения отношения расстояния между узлами беспроводной сети. Функция степени близости уже не является бинарной, и в качестве значения может возвращать, например, величины в децибелах, относительные величины и т.д. Чаще всего используются децибелы, т.к. именно в этих величинах значение уровня сигнала чаще всего доступно принимающему устройству.

В IPS такое определение близости устройств применяют для позиционирования устройств, используя методы триангуляции, трилатерации и т.п.[2] При решении многих задач толкование свойства сетевой близости, как расстояния между устройствами, не требуется,

например, такими задачами являются все из ранее перечисленных кроме первой.

Далее рассматриваются оба варианта определения близости, и как бинарной функции, и как функции расстояния, однако стоит уточнить, что не всегда возможно определить уровень сигнала, или это возможно только при установлении соединения между устройствами[6]. Не теряя общности можно использовать понятие сетевой близости, уточняя по необходимости, когда речь идет о функции расстояния.

Сетевая близость характеризуется рядом свойств:

1. Работоспособность в помещениях;
2. Точность;
3. Мобильность.

Работоспособность в помещениях

Данное свойство хорошо применимо в IPS. В то время как системы глобального позиционирования, такие как GPS, в зданиях не работают или работают очень плохо, сетевая близость удачно используется внутри помещений для позиционирования, при этом она не требует длительного поиска спутников связи или стационарных вышек, сложного оборудования и т.д. Для работы IPS, используя только сетевую близость, необходимо иметь достаточно плотную сеть устройств, поддерживающих выбранные технологии беспроводных сетей. Плотность сети, т.е. насколько часто встречаются устройства по пути движения, или насколько много их находится в окрестности устройства, зависит от характеристик выбранной технологии беспроводных сетей и объясняется алгоритмами, вычисляющими положение устройства относительно устройств в его

окрестности. Такие алгоритмы требуют, по меньшей мере, трех устройств в границах видимости. Чем больше видимых устройств, тем выше, обычно, точность позиционирования. Для подобных сервисов требуются также карты помещений, которые необходимо предварительно загрузить на устройство, а также метаданные о местоположении на карте известных устройств.

Точность

Сетевая близость позволяет локализовать объекты, которые находятся в непосредственной близости, при этом можно быть уверенным в точности данных при использовании функции расстояния, т.к. локальность сетевой близости обеспечивает известные возможности, зависящие от характеристик выбранной беспроводной технологии передачи данных. Важным качеством является и то, что сетевая близость позволяет локализовать объекты логически, в данном случае точность информации о видимости устройств скорее несет семантическую нагрузку, а не географическую (пространственную). С помощью сетевой близости можно говорить об указании местоположения с точностью до объекта локализации, это может быть этаж, офис, комната и т.д. В частности, сетевая близость может указывать и на видимость некоторого заранее известного объекта, оснащенного передатчиком, даже если он не находится внутри какого-либо помещения. Например, в качестве такого объекта может выступать автомобиль (современные автомобили оснащаются модуля Bluetooth).

Мобильность

Данное свойство позволяет с помощью сетевой близости устанавливать отношения между нестационарными устройствами. В отличие от систем глобального позиционирования, в котором для удовлетворения указанного требования необходимо обладать знаниями о географических координатах всех устройств, отношение видимости между которыми потребовалось бы установить, сетевая близость позволяет указать, какие устройства находятся поблизости друг от друга, не зная их абсолютного размещения, не имея сведений об их географических координатах. Таким образом, даже, если оба устройства фактически не обладают знаниями о своих географических координатах, они могут получить сведения о ближайших устройствах даже в тех случаях, когда они двигаются, но расстояние между ними не превышает возможностей выбранной технологии беспроводной связи. Примером, когда это свойство актуально, может являться подвижной состав электропоезда, оснащенный Bluetooth модулем, в котором находится пассажир с Bluetooth-устройством. В данном случае можно также отметить и возможность указать расположение устройства с точностью до вагона. Подобные возможности хорошо пригодны в информировании в системах общественного транспорта.

Таким образом, можно отметить, что сетевая близость позволяет локализовать объекты с точностью выбранной беспроводной технологии передачи данных, т.е. выбирая конкретную технологию можно контролировать, варьировать свойство локальности сетевой близости.

2.3. Описание архитектуры сервиса

В архитектуре сервиса, использующего сетевую близость на базе беспроводных сетей, выделим несколько крупных элементов:

1. Сканирующее устройство;
2. Метки;
3. Удаленный сервис;
4. Хранилище.

Проведем описание элементов архитектуры, используя, где это потребуется, в качестве выбранного протокола стандартный Bluetooth. Общности архитектура не потеряет, и при небольших изменениях, всё описанное справедливо и для реализаций, использующих Bluetooth LE, WiFi, ZigBee или любую другую подходящую беспроводную технологию. Выбор протокола Bluetooth в качестве демонстрационного позволяет описывать архитектуру более детализировано, предоставляя одновременно пример ее конкретной модификации. Также прототип, описанный в практической части работы, использует в качестве архитектуры именно такую модификацию, что обеспечивает для демонстрации ее работоспособности сквозной пример. Отметим еще раз, что в предлагаемой архитектуре протокол беспроводной передачи данных не фиксируется, и в качестве конкретного протокола может быть выбран любой доступный в условиях решаемой задачи.

Сканирующее устройство

На сканирующее устройство устанавливается приложение,

основными задачами которого являются:

1. Сканирование беспроводной среды на предмет обнаружения меток;
2. Формирование из меток и контекстной информации слепков;
3. Передача слепков удаленному сервису;
4. Подписка на push-уведомления;
5. Отображение уведомлений;
6. Совершение некоторых действий в ответ на уведомления;
7. Активация/деактивация обнаружения и обработки уведомлений;
8. Добавление правил отображения информации на удаленный сервер, если само сканирующее устройство выступает в роли метки.

Приложение сканирования состоит из двух частей. Одна часть - визуальная, отображение настроек приложения (настроек контроля сканирования), отображение объявлений, просмотр обнаруженных слепков, выполнения действий, переданных в данных push-уведомления и т.п. Другая часть - собственно сканирующий сервис, запущенный в фоновом режиме, не имеет визуальной составляющей, однако предоставляет доступ к своему состоянию и настройкам через область уведомлений. Сканирующий фоновый сервис запускается при старте приложения и работает, даже если приложение завершить, при этом он может запустить визуальную часть приложения для контроля состояния сканирования. При старте сервиса проверяется, включен ли Bluetooth-адаптер, и если не включен, включается, далее запускается процесс обнаружения меток. По завершении процесса обнаружения меток, они собираются вместе с некоторой контекстной информацией в слепок и отправляются удаленному сервису. Режим работы сервиса настраивается, но общая логика его заключается в том, что он запускает сканирование раз в какой-то

установленный промежуток времени, по завершении которого уходит в ожидание на определенное время. Процесс сканирования зависит от выбранной технологии беспроводной передачи данных, и, например, в стандартном протоколе Bluetooth может занимать около 12 секунд, поэтому можно произвести настройку сервиса для принудительного завершения сканирования до истечения указанного интервала времени. Это может осуществляться, например, с целью экономии заряда батареи, т.к. после завершения сканирования сервис может также выключать Bluetooth-устройство. Еще одной причиной принудительного завершения сканирования можно отметить желание сократить время формирования слепка для более быстрой отработки общего цикла доставки объявлений.

Метки

Метками, как и в BDP, в предлагаемой архитектуре являются обычные Bluetooth-устройства, на которых может запускаться приложение, переводящее устройство в режим discoverable, т.е. в режим «доступно для обнаружения». Устройство может быть переведено в указанный режим и иным путем, если есть такая возможность, без установки на него соответствующего приложения. Больше от устройства ничего не требуется, никаких данных специального назначения оно не передает. Основная его задача - быть доступным.

Удаленный сервис

Сервис реализуется на основе архитектуры REST, используя в качестве тела HTTP-сообщений JSON-объекты, и выполняет несколько

задач, среди которых (в скобках указаны endpoint-ы сервиса, соответствующие рассматриваемым запросам):

1. Регистрация в сервисе доставки push-уведомлений (выполняется заранее, без необходимости совершения запроса извне);
2. Регистрация слепков (запрос registerFingerprint);
3. Поиск правил по слепку (внутренняя работа сервиса, выполняемая на запрос регистрации слепка);
4. Формирование содержимого, используя найденные правила (поиск данных в хранилище по правилам);
5. Отправка push-уведомлений с содержимым (запрос на сторонний сервер доставки push-уведомлений);
6. Ведение статистики (сохранение в хранилище данных по запросам);
7. Добавление правил (запрос addRule).
8. Регистрация точек (запрос registerPoint).

В качестве платформы для разворачивания сервиса удобно использовать PaaS.

Хранилище

Хранилище использует реляционную систему управления базами данных и хранит:

1. Присланные слепки со временем их прихода;
2. Правила;
3. Метки;
4. Данные запросов;
5. Объявления.

Опишем, как выглядит полный жизненный цикл предлагаемого сервиса. Сначала заводятся метки, они регистрируются в сервисе с помощью отправки по endpoint-у registerPoint с передачей в качестве параметра данных метки. Регистрацию метки может производить клиентское приложение, которое выступает в роли самой метки, другое клиентское приложение, если пользователем предоставлены данные о метке, а также с помощью веб-интерфейса доступа к сервису. На сканирующее устройство устанавливается приложение для сканирования меток. После установки запускается процесс сканирования описанным выше образом, по завершении сканирования, проверяется, получены ли необходимые данные для формирования слепка. Если данные не получены, сканирование откладывается на установленный таймаут и повторяется снова по его прошествии. Если данные получены, формируется слепок, далее он отправляется сервису по endpoint-у registerFingerprint с передачей данных слепка. Сервис обрабатывает слепок и производит поиск правил, которым он удовлетворяет. Если такие правила найдены не были, на этом работа сервиса по обработке текущего запроса завершается сохранением данных по запросу. Если соответствующие правила обнаружились, по ним выполняется формирование содержимого ответа, поиск объявления и отправка данных по нему push-уведомлением на сервис доставки с последующим сохранением данных по запросу. Далее сервис доставки push-уведомлений доставляет уведомления на сканирующее устройство, обработчик, подписанный на уведомления, вызывается мобильной операционной системой, и управление передается сканирующей программе, которая добавляет локальные уведомления в область уведомлений с содержимым, пришедшим в push-уведомлении. Если по правилам сервис сформировал несколько уведомлений, то несколько

уведомлений будет доставлено и сканирующему устройству. И задачей сканирующей программы уже является возможная группировка полученных уведомлений.

Более подробно параметры запросов, отправляемые объекты и логика описаны далее.

Объявления

Объявления - это некоторый текст, который сервис должен отправить сканирующему устройству в ответ на присланный слепок в случае, если нашлись правила, ему удовлетворяющие. В качестве объявления может быть и обычный текст, текст с форматированием, HTML-страница, URL и т.п. В простейшем случае, как это рассмотрено в BDP, сканирующее приложение является обычным веб-браузером, т.е. WebView встроенным в приложение, в которое загружается URL, присланный сервисом. Но наиболее универсальным объявление может быть, если в нем будет указан тип данных, содержащихся в его теле. В таком случае не нужно выбирать некоторый общий тип данных, а можно добавлять поддержку новых типов данных в сканирующее приложение при условии, что формат данных, присланных в объявлении, не меняется. Таким образом, можно задавать не только текст, графику, ресурс на удаленном сервере и т.д., но и более сложные данные, а также действия, которые может поддерживать сканирующее приложение. Одним из примеров таких типов могут быть географические координаты с текстом, которые необходимо отобразить на карте. Современные мобильные операционные системы позволяют одному приложению запускать другие приложения, причем часто не только стандартные, но сторонние, установленные и не являющиеся частью

системы стандартной поставки. URL в таком случае разумно было бы открывать в стандартном браузере, точно также как и графические файлы, или звуковые файлы.

Для определения такого поведения, необходим объект объявления, который сможет в себе инкапсулировать такую логику.

Таким образом, объект объявления должен в себе содержать:

1. Тип данных;

2. Сами данные;

3. Действие, которое необходимо над ними совершить.

В формате JSON такое объявление, присланное сервисом, может выглядеть, например, следующим образом:

```
{
  "type": "geographic-coordinates",
  "data": { "latitude": 55.4506, "longitude": 37.3704 },
  "action": "show-on-map"
}
```

При приеме нескольких уведомлений сканирующее приложение должно быть способно группировать их в области уведомлений. Для группировки можно использовать категорию объявления. Таким образом, к представленному выше формату можно добавить категорию. В таком случае представленное выше объявление может выглядеть так:

```
{
  "type": "geographic-coordinates",
  "data": { "latitude": 55.4506, "longitude": 37.3704 },
  "action": "show-on-map",
  "category": "trade"
}
```

Семантика за категориями может закрепляться любая, но разумно предположить, что это будут категории доступных услуг.

В хранилище такое же объявление может выглядеть так:

Announces (primary key id, creation_time, activation_start, activation_period, data, type, action, category), где:

1. id - идентификатор объявления;
2. creation_time - время создания объявления;
3. activation_start - время, с которого объявление считается активным;
4. activation_period - период, в течение которого с начала активации объявление считается активным;
5. data - данные;
6. type - тип данных;
7. action - действие над данными;
8. category - категория объявления.

Следует отметить, что активация объявлений может контролироваться не только сервисом, но и сканирующим приложением, когда сканирующее приложение (посредством настройки) может выбирать, например, насколько старые объявления оно будет отображать, независимо от того, активно оно до сих пор или нет. В таком случае по образцу с категорией в отправляемом JSON можно указывать и другие данные, представленные в хранилище, например, дату создания объявления или дату, с которой объявление считается активированным, и период его активности.

Слепки и метки

В отличие от BDP в предлагаемой архитектуре также вместо просто

меток используются слепки. Слепки - это объекты, которые собираются по завершении процесса сканирования меток, состоящие из следующего набора данных:

1. Список меток;
2. Время создания слепка;
3. Тип сети (для описания мы приняли, что тип сети Bluetooth);
4. MAC-адрес Bluetooth-адаптера сканирующего устройства;
5. RSSI (опционально, в случае, если используется Bluetooth LE);
6. Комментарий (опционально, простой текст, описывающий место, в котором слепок был сформирован, задается пользователем по требованию сканирующего приложения);
7. Географические координаты (опционально, в случае, если сервис гео-позиционирования включен, работает и может предоставить координаты);
8. Идентификатор сканирующего устройства для отправки push-уведомления.

Таким, например, может быть JSON, который отправляется удаленному сервису в качестве тела HTTP-пакета на endpoint registerFingerprint:


```
{
 "macAddress": "34:FC:2F:42:59:4F",
 "networkType": "Bluetooth",
"points": [ { "deviceAddress": "7C:31:93:C9:37:CA", "deviceClass": "5a020c", "deviceName": "HTC Device" } ],
 "time": 1430017732769,
 "registrationId": "APA91bHun4MxP5e..."
}
```

Здесь points - это список меток. Метки содержат следующие данные:

1. MAC-адрес;

2. Класс устройства;

3. Имя устройства.

Точки могут быть соответственно зарегистрированы с помощью отправки на endpoint сервиса registerPoint JSON-сообщения следующего содержания:

```
{
  "deviceAddress": "7C:31:93:C9:37:CA",
  "deviceClass": "5a020c",
  "deviceName": "HTC Device"
}
```

Преимуществом использования слепков вместо отдельных меток можно считать возможность установить, что они обнаружены совместно. Кроме того сам слепок может нести дополнительную информацию, которая не описывается собственно метками, например, контекстную информацию. В данном случае опциональными указаны комментарий, координаты и т.д. Потенциально же, как развитие идеи, можно ожидать добавление в слепок информации, например, об интенсивности света, окружающих звуках и т.д.

В хранилище данные сущности могут выглядеть, например, так:

Fingerprints(primary key id, mac_address, network_type, time), где

1. id - идентификатор слепка;

2. mac_address - MAC-адрес сканирующего устройства;

3. network_type - тип сети;

4. time - время создания слепка.

Как видно, не хранится registrationId, идентификатор для отправки push-уведомлений, хранить его не имеет смысла, т.к. это временные

данные, нужны только для отправки уведомлений, да и устройство всё равно должно его отправлять каждый раз при отправке слепка. Связь слепка и меток также не задается в отношении слепка, т.к. для этого создается специальное отношение:

Fingerprints_Points(fingerprint_id, point_id), где

1. fingerprint_id - идентификатор слепка;

2. point_id - идентификатор метки.

Для меток создается следующее отношение:

Points(primary key id, mac_address, device_class, device_name), где

1. id - идентификатор метки;

2. mac_address - MAC-адрес метки;

3. device_class - класс устройства;

4. device_name - имя устройства.

Атрибуты mac_address, device_class и device_name должны быть ограничены на уникальность, чтобы обеспечить свойство уникальности добавляемых меток.

Правила

В простейшем случае правило - это связка метки и объявления, и условие его выполнения - наличие метки в слепке. Если метка в слепке присутствует, соответствующее объявление отправляется сканирующему устройству, идентифицированному registrationId. Сколько метод в слепке обнаружено, для которых заданы правила, столько уведомлений сервис отправляет.

Отношение, представляющее правило может выглядеть, как:

Rules(primary key id, point_id, announce_id) где

1. id - идентификатор правила;

2. point_id - идентификатор метки;

3. announce_id - идентификатор объявления.

Пара point_id и announce_id должна быть с ограничением на уникальность для того, чтобы правила не дублировались по существу.

Надо заметить, что важным свойством является возможность привязать к одной точке несколько объявлений.

Для такой простой модели правил лучше подходят нереляционные базы данных, например, Key-Value базы данных или RDF[12]. Однако если предположить, что правила могут задаваться несколько более сложным способом, реляционная модель может оказаться более эффективной. Так, мы можем добавить некоторые операции над правилами, такие как AND и OR, а сами правила формулировать в виде простых предикатов, например POINT_IS_VISIBLE, тогда объединяя несколько правил с помощью указанных операций и сформулированных предикатов, мы можем создавать сложные правила из более простых. В данном случае подходящая модель базы данных является предметом исследования, т.к. такие правила фактически образуют выражение, которое хорошо представимо в виде дерева, внутренними вершинами которого являются операции, а листьями - данные из полученного слепка. Для представления подобных структур часто используют графовые базы данных[26].

3. ОПИСАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ

3.1. Использованные технологии

Для реализации предложенной архитектуры в работе был разработан программный прототип, приложение с основными функциональными решениями, описанными в архитектуре. В качестве средств реализации предпочтительными считались те, которые удовлетворяют условиям открытости, высокой доступности, а также являются достаточно простыми и удобными в использовании и развертывании. Указанные свойства являются важными при разработке прототипов и исследовании в области информационных технологий, т. к. позволяют сконцентрироваться именно на решении поставленной задачи, а не на технических аспектах конкретных технологий разработки программного обеспечения. Важно также отметить, что предложенная в работе архитектура не зависит от конкретных программных технологий, но ей необходима определенная свобода в их выборе, которая может обеспечить предъявляемые в архитектуре требования.

Исходя из изложенной мотивации в выборе программных средств и инструментария для реализации прототипа, в работе была сделана ставка на решения с открытым исходным кодом, являющиеся де-факто стандартом в области разработки программного обеспечения. Выбор данных средств был обусловлен также и практической целесообразностью.

В качестве основного языка программирования и среды исполнения была выбрана технология Java. Она использовалась для реализации как серверной, так и клиентской частей прототипа, что обеспечило более

высокую интероперабельность между двумя основными элементами архитектуры.

Для реализации серверной части прототипа удобным средством выступил фреймворк Spring, в него входит целый ряд готовых решений для наиболее распространенных задач. Так, в Spring существуют стандартные решения для создание сервисов на основе REST-архитектуры, ORM Hibernate (одну из наиболее распространенных версий JPA), а также реализация популярного шаблона проектирования MVC для разработки приложений, ориентированных на взаимодействие с пользователем. Кроме того Spring позволяет развертывать и запускать приложения несколькими способами. Одним из наиболее удобных, в т.ч. для прототипирования, является способ, реализованный в Spring Boot. Т.к. для серверной части использовалась платформа Java EE, в которую входит сервер приложений, механизм сервлетов и т. д., то Spring Boot оказался наиболее удобен тем, что объединяет в себе популярные реализации указанных элементов платформы, обеспечивая простой подход к управлению серверным решением. В Spring Boot по умолчанию используется веб-сервер и контейнер сервлетов Apache Tomcat. Другим удобным свойством Spring Boot является возможность запуска сервера как стандартного Java-приложения, при этом веб-сервер и контейнер сервлетов размещаются внутри одного JAR-файла. Такое решение упрощает развертывание веб-приложения и особенно удобно для создания прототипа.

В качестве системы сборки и развертывания используется Maven, в файле pom.xml которого указан способ сборки, зависимости и плагины. Среди зависимостей проекта помимо указанного контейнера сервлетов и встроенного в приложение веб-сервера Apache Tomcat также используются драйвер баз данных PostgreSQL, JSTL и JPA модуль Spring Boot-а.

На стороне сервера используется база данных PostgreSQL и JPA Repositories из фреймворка Spring для хранения данных и доступа к ним соответственно. JSTL используется вместе с JSP для формирования веб-страниц пользовательского интерфейса. Несмотря на то, что используются стандартные технологии создания веб-приложения, которые позволяют развернуть сервер на любой машине, в т.ч. и локальной, в прототипе для доступа к серверу через Интернет использовалась платформа Heroku в качестве решения PaaS. Heroku позволяет быстро развернуть приложение, предоставляя удобную панель администрирования и домен для доступа к приложению. Heroku в качестве системы управления базами данных по умолчанию предлагает PostgreSQL.

В качестве клиентской части для реализации в прототипе выбрана мобильная платформа Android. Она удовлетворяет условиям доступности и простоты в использовании в качестве средства разработки. Важным свойством Android, как программного решения, можно считать достаточно полную документацию, открытый исходный код, API с широким спектром возможностей.

3.2. Серверная часть

Серверное приложение состоит из следующих частей:

1. Модель, или объекты предметной области;
2. Слой доступа к данным;
3. Сервис для мобильных клиентов;
4. Веб-интерфейс для создания правил, просмотра данных и т.д.;
5. Стартовый модуль запуска;

6. Конфигурация приложения.

Модель

Модель состоит из объектов предметной области, содержит сущности всех указанных в архитектуре понятий, таких как уведомление (Announce), точка (Point), слепок (Fingerprint), правило (Rule) и т. д. Сущности модели располагаются в отдельном пакете domain.

Слой доступа к данным

Слой доступа к данным использует абстракции репозиториев данных, специализированных типами хранящихся объектов, например, для хранения слепков и доступа к ним используется FingerprintRepository, для точек — PointRepository, для правил RuleRepository и т. д. Каждый из них создан на основе CrudRepository из Spring JPA. Так, например, выглядит PointRepository:

```
public interface PointRepository extends CrudRepository<Point, Long> {
    Point findByDeviceAddress(String pointMacAddress);
...
}.
```

CrudRepository реализует основные операции работы с данными и зависит от типа этих данных, т. к. реализован как generic-интерфейс Java. Репозитории из Spring JPA (расположенные в пакете org.springframework.data.repository) позволяют создавать запросы к базе данных на основе специальной нотации имен методов, по которым фреймворк с помощью reflection генерирует нужные запросы. Так, в

описанном выше примере метод findByDeviceAddress позволяет найти точку по ее MAC-адресу. При этом реализация метода, да и класса для данного интерфейса от разработчика не требуется. Таким образом, слой доступа к данным представляет собой набор репозиториев для соответствующих типов данных модели.

Сервис для мобильных клиентов

Сервис для мобильных клиентов использует другой удобный механизм Spring-a, позволяющий создавать REST-сервисы простым образом. Этот механизм основан на использовании RestController-a. RestController-ом может быть любой класс, обозначенный специальной аннотацией @RestController языка Java. В самом классе создаются методы и отображение адреса в HTTP пакете для каждого из них. Так, например, выглядит сигнатура метода регистрации слепков в сервисе:

```
@RequestMapping(value = "/registerFingerprint", consumes =
"application/json")
public void registerFingerprint(@RequestBody Fingerprint fingerprint) {
…
}
```

В данном случае точкой входа, методом сервиса, или endpoint-ом, является /registerFingerprint, что указано в аннотации. При отправке запроса на этот endpoint, управление передается в указанный выше метод. В качестве второго параметра выступает описание ожидаемого формата данных для этого метода. В данном случае, в соответствии с предложенной архитектурой, данными является JSON-сообщение. Аннотация @RequestBody указывает на то, откуда брать данные для параметра метода,

в указанном выше методе это тело HTTP-пакета. Т.к. Spring использует по умолчанию библиотеку Jackson для сериализации объектов Java в JSON и обратно, он автоматически разберет тело запроса и сконструирует объект предметной области Fingerprint. Если это ему сделать не удастся, например, в теле запроса будет некорректный JSON-объект или его не будет вовсе, или будет указан неверный Content-Type в HTTP-заголовках, сервис вернет ошибку. В случае, если формирование объекта прошло успешно, сервис использует репозиторий для манипуляции с данными, например, в указанном выше методе слепок будет сохранен для статистики следующим образом:

```
…
fingerprintRepository.save(fingerprint);
…
```

После сохранения осуществляется поиск с помощью RuleRepository правил для точек, входящих в слепок, по их MAC-адресам методом, аналогичным описанному для PointRepository. В прототипе реализованы простые правила, отображающие MAC-адрес точки на текст.

Для доступа к репозиториям RestController использует DI (Dependency Injection), реализованный в Spring. Поиск репозиториев осуществляется также Spring-ом благодаря конфигурации приложения.

Веб-интерфейс

Веб-интерфейс реализован с помощью JSP и JSTL, поиск представлений пользовательского интерфейса осуществляется Spring-ом, сконфигурированным файлом application.properties. Представления располагаются по пути webapp/WEB-INF/jsp.

Стартовый модуль запуска

Стартовый модуль запуска реализуется на основе SpringBootServletInitializer:

```
@ComponentScan
@EnableAutoConfiguration
@ImportResource("applicationContext.xml")
public class Application extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
            return application.sources(Application.class);
    }
…
```

В аннотациях указывается файл конфигурации, это стандартный подход Spring. Запускается приложение простым указанием Spring-у класса приложения:

```
SpringApplication.run(Application.class, args);
```

Конфигурация приложения

Конфигурирование приложения осуществляется в файле applicationContext.xml, помимо прочих стандартных описаний Bean-ов Spring-а в этом файле также указывается URL к базе данных. Т.к. база данных конфигурируется сервисом Heroku, в конфигурации необходимо производить разбор переменных окружения, задаваемых Heroku при старте

приложения:

```
  <bean class="java.net.URI" id="dbUrl">
      <constructor-arg value="#{systemEnvironment['DATABASE_URL']}"/>
    </bean>
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="url" value="#{ 'jdbc:postgresql://' + @dbUrl.getHost()
+ ':' + @dbUrl.getPort() + @dbUrl.getPath() }"/>
      <property name="username"
          value="#{ @dbUrl.getUserInfo().split(':')[0] }"/>
      <property name="password"
          value="#{ @dbUrl.getUserInfo().split(':')[1] }"/>
    </bean>
```

Аналогично, в качестве источника данных в конфигурации указывается провайдер баз данных. Исходный код сервера доступен на GitHub [27].

В качестве клиентской платформы в прототипе выбрана система Android. Она наиболее удобна для разработки мобильных приложений в исследовательских целях: достаточно много доступно написанной документации, примеров приложений, открытый исходный код системы, API с широким функционалом, позволяет разрабатывать приложения под любой операционной системой: Windows, MacOS или Linux.

3.3. Клиентская часть

Клиентское приложение имеет схожее разбиение в своей базовой

части, например, в нем также представлены объекты предметной области, визуальные компоненты, созданные уже на основе стандартных элементов Android. Специфическими для клиента компонентами являются:

1. Сервис фонового сканирования;
2. Подписчики на события;
3. Запросы к сервису;
4. Уведомления.

Сервис фонового сканирования

Фоновый сервис Android — это сервис, который после своего запуска может работать независимо от того, запущено основное приложение или нет. В клиентском приложении сервис фонового сканирования — это сервис Android, который осуществляет сканирование в беспроводной среде, фиксирует найденные точки, собирает их в слепки и отправляет удаленному сервису. В прототипе используется только беспроводная сеть Bluetooth для сканирования точек. Интервал сканирования устанавливается по умолчанию в 20 секунд, но может настраиваться в приложении. Т.к. Bluetooth сканирование в общей сложности может работать около 12 секунд, то все точки, которые были собраны за это время, попадают в один слепок. Для осуществления своей задачи сервису требуются остальные компоненты приложения. Так, он активирует запрос сервису, использует подписчиков для получения точек, обнаруженных во время сканирования.

Сервис запускается в главной задаче приложения вызовом:

DiscoveryService.startOn(this);

- параметром которого является контекст выполнения.

Запуск осуществляется простым образом:

```
public static void startOn(Context context) {
        if (!isRunning(context)) {
                Intent intent = new Intent(context, DiscoveryService.class);
                context.startService(intent);
        }
}
```

Сервис на приложение может быть только один, в системе Android он регистрируется по своему полному квалифицированному имени, что гарантирует его уникальность. Сервис реализует полный жизненный цикл сервиса Android, при первоначальном запуске создает локальное уведомление в области уведомлений, через которое можно перейти к его настройкам в основном приложении. При старте сервис запускает подписчиков на события обнаружения точек и Bluetooth-адаптер:

```
IntentFilter filter = new IntentFilter();
filter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
registerReceiver(bluetoothStateObserver, filter);
...
if (!bluetoothAdapter().isEnabled()) {
        bluetoothAdapter().enable();
}
```

После чего стартует обнаружение доступных точек:

```
bluetoothAdapter().startDiscovery();
```

При завершении работы сервиса, в случае, если пользователь его останавливает явно, сервис отписывает подписчиков.

Подписчики на события

Подписчики реализуются на основе класса BroadcastReceiver системы Android. Подписчик, отвечающий за получение найденных точек и формирование слепка, регистрируется следующим образом:

```
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(bluetoothPointsCollector, filter);
```

Каждый из подписчиков должен перегрузить метод public void onReceive(Context context, Intent intent) класса BroadcastReceiver. Так получаются обнаруженные точки в подписчике bluetoothPointsCollector:

```
@Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        BluetoothDevice device = intent.getParcelableExtra
                (BluetoothDevice.EXTRA_DEVICE);
        Point point = new Point(device);
        fingerprint.addPoint(point);
    }
```

- где fingerprint — слепок, в который добавляется найденная точка. Когда обнаружение закончено, слепок проверяется на корректность и отправляется сервису.

Запросы к сервису

Запрос к сервису реализован как асинхронная задача, выполняющаяся в фоновом рабочем потоке. Специально для таких задач в Android есть класс AsyncTask, который и был использован. Так задача

запускается:

```
if (Network.isAvailable(context)) {
    new FingerprintRegistration(fingerprint).execute();
}
else {
…
}
```

В случае отсутствия доступа к Интернет слепки складываются в очередь. Сама задача должна реализовать метод protected Void doInBackground(Void... params), например, для FingerprintRegistration он выглядит так:

```
@Override
protected Void doInBackground(Void... params) {
        try {
                URL url = new
                URL("http://bdpservice.herokuapp.com/registerFingerprint");
                HttpURLConnection connection =
                        (HttpURLConnection)url.openConnection();
                ...
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Content-type",
                        "application/json");
                ...
                OutputStream outputStream = connection.getOutputStream();
                BufferedWriter writer = new BufferedWriter(new
                        OutputStreamWriter(outputStream, "UTF-8"));
                writer.write(fingerprint.toJson());
```

```
                ...
            }
…
```

Далее идет обработка ошибок в случае неудачной попытки зарегистрировать слепок. Как и на стороне сервера, клиентское приложение использует библиотеку сериализации Jackson.

Уведомления

Уведомления на клиенте создаются локально при получении notification-а от центра уведомлений стандартным для Android образом, не требующим дополнительных усилий. Так, в прототипе используется соответствующий NotificationCompat.Builder.

Важным объединяющим узлом в предложенной архитектуре является центр уведомлений. Регистрация в нем осуществляется и сервером, и клиентом, она описана в соответствующей документации Google Cloud Messaging[24],[25]. Клиент с каждым слепком отправляет свой registration_id, который сервер использует для отправки уведомления через центр уведомлений. В прототипе registration_id отправляется как часть Fingerprint-а для простоты реализации. Исходный код клиента доступен на GitHub [28].

Описанная реализация прототипа предложенной архитектуры может развиваться дальше согласно развитию идей, заложенных в самой архитектуре. Одним из вариантов возможного развития прототипа может быть добавление стандартных действий в мобильное приложение и выполнение их над данными, приходящими от центра уведомлений. Другим возможным вариантом может быть развитие представления

информации в зависимости от категории, которой она принадлежит. В серверной части логичным продолжением работы было бы расширение возможностей правил и операций над ними, а также использование статистики, построенной по слепкам, для вычисления интересных в рамках прикладных задач метрик.

Заключение

В работе было рассмотрено понятие сетевой близости, ее свойства, описаны преимущества, доступные сервисам, использующим их. Далее были описаны технологии и сервисы, использующие Bluetooth LE, рассмотрены их ограничения, указаны недостатки, для решения которых была предложена архитектура и подход к построению сервисов, основанных на сетевой близости. Предложенная в работе архитектура базируется на идеях использования любых сетевых устройств в беспроводных средах в качестве меток, фонового сканирования устройств, использования центра уведомления для доставки контента на устройства, а также правил, позволяющих задавать соответствия между слепками и контентом и управлять его отправкой на конечное устройство. Для реализации предложенной архитектуры достаточно обычных мобильных телефонов или устройств с модулями беспроводной связи небольшого радиуса действия, а также применения распространенных серверных технологий, что было проиллюстрировано на примере прототипа, реализованного в соответствии с предложенной в работе архитектурой. При этом описанная архитектура не ограничена примененными для реализации прототипа технологиями и имеет несколько точек роста, дающих возможность продолжать исследования в этой области и развивать имеющуюся модель.

Список литературы

[1] Д.Е. Намиот, Мобильные Bluetooth теги // International Journal of Open Information Technologies. - 2014. - vol. 2, no. 5 – С. 17-23.

[2] Dmitry Namiot, On Indoor Positioning // International Journal of Open Information Technologies. - 2015. - vol. 3, no. 3 – С. 23-26.

[3] Neal Patwari and Alfred O. Hero, Using Proximity and Quantized RSS for Sensor Localization in Wireless Networks, Univercity of Michigan, Dept. of Electrical Engineering and Computer Science, CA, USA, Sep. 2003.

[4] Agusti Corbaco Salas, Indoor Positioning System based on Bluetooth Low Energy, Universitat Politechnica De Catalunya, Barcelonatech, Barcelona, Jun. 2014.

[5] Bianca Deordica and Marian Alexandru, Advertisement using Bluetooth Low Energy, Transilvania Univercity, Brasov, Romania, Review of the Air Force Academy, No 2 (26) 2014.

[6] Android Bluetooth Gatt. URL: https://developer.android.com/reference/android/bluetooth/BluetoothGatt.html

[7] ZigBee. URL: https://ru.wikipedia.org/wiki/ZigBee

[8] SITA Common Use Beacon Registry. URL: https://www.developer.aero/Beacon-Registry-API/API-Overview

[9] Bluetooth Low Energy. URL: https://en.wikipedia.org/wiki/Bluetooth_low_energy

[10] Samsung Placedge. URL: https://placedge.samsung.com/

[11] Google Nearby. URL: https://developers.google.com/games/services/android/nearby

[12] Resource Description Framework. URL:

https://ru.wikipedia.org/wiki/Resource_Description_Framework

[13] F. Akyildiz and I.H. Kasimoglu, "Wireless Sensor and Actor Networks: Research Challenges", Ad Hoc Networks, vol. 2, no. 4, p. 351-367, Oct. 2004.

[14] IEEE 802.15 Working Group for WPAN URL: http://www.ieee802.org/15/

[15] IEEE 802.11TM WIRELESS LOCAL AREA NETWORKS URL: http://www.ieee802.org/11/

[16] Bluetooth Core Specification 4.2 URL: https://www.bluetooth.org/en-us/specification/adopted-specifications

[17] iOS: Understanding iBeacon URL: https://support.apple.com/en-gb/HT202880

[18] Navizon Proximity Engine URL: https://www.navizon.com/product-navizon-hub

[19] Dmitry Namiot, GeoFence Services // International Journal of Open Information Technologies. Nov. 2013.

[20] Guardly Platform API URL: https://www.guardly.com/technology/api

[21] Гурьев Д. Е., Намиот Д. Е., Шнепс М. А. О телекоммуникационных сервисах //International Journal of Open Information Technologies. – 2014. – Т. 2. – No. 4. – С. 13-17.

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

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

[24] GCM HTTP Connection Server URL:

http://developer.android.com/google/gcm/http.html

[25] Implementing GCM Client on Android URL: http://developer.android.com/google/gcm/client.html

[26] DB-Engines Ranking of Graph DBMS URL: http://db-engines.com/en/ranking/graph+dbms

[27] BDPServer URL: https://github.com/SrVisor/BDPServer

[28] BDPClient URL: https://github.com/SrVisor/BDPClient