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

Кафедра системного программирования

Рагимов Руслан Вугарович

Разработка аппаратно-программного комплекса для организации сети датчиков

Бакалаврская работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А. Н.

Научный руководитель:
ст. преп. Сартасов С. Ю.

Рецензент:
ООО «Оракл Девелопмент СПБ», Старший руководитель группы Косенчук А. М.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Software Engineering Chair

Ruslan Ragimov

Development of hardware-software complex for sensor net organisation

Bachelor's Thesis

Admitted for defence.
Head of the chair:
professor Andrey Terekhov

Scientific supervisor:
assistant Stanislav Sartasov

Reviewer:
Senior Engineering Manager, Oracle Development SPB, LLC Alexey Kosenchuk

Saint-Petersburg
2016

Оглавление

Введение

Сегодня названия «умный браслет», «умные часы», «умный дом» встречаются очень часто. Как правило, любая «умная» вещь имеет возможность подключения к сети Интернет[3]. Отсюда и появился термин «Интернет вещей». Этот термин скрывает за собой весьма популярную на сегодняшний день концепцию, заключающуюся в том, что в современном мире использование сети Интернет для обмена информацией стало возможным и даже необходимым не только для полноценных компьютеров, а для любых устройств, будь то автомобиль или бытовая техника. Эти «умные» устройства служат как для повышения комфорта жизни людей (интеллектуальное управление климатом в доме, освещением в доме и на садовом участке), так и для выполнения сложной интеллектуальной работы (например, анализ состояния экологии).

Одним из важных процессов в концепции Интернета вещей является сбор данных о каких-либо событиях или окружающей среде. Например, сбор и правильный анализ данных о влажности и температуре с полей может помочь в планировании и организации посадки овощей или других растений.

Рассмотрим пример более подробно: допустим, есть необходимость собирать данные об интенсивности солнечного света внутри большого парника с овощами. Тогда придётся установить некоторое, возможно, большое количество датчиков и организовать сбор информации с них (а точнее, сбор с дальнейшей отправкой на сервер для анализа). В данном случае нет необходимости получать данные в режиме реального времени - достаточно получать данные с некоторой периодичностью или по расписанию. Именно этот случай будет рассматриваться в данной работе. Также есть несколько принципиально разных подходов к сбору данных[8], т.к парник может, например, находиться в зоне Wi-Fi или наоборот может находиться далеко от цивилизации даже вне зоны покрытия[12] сети GSM. В работе будут более подробно рассмотрены эти подходы и возможность их использования для периодического сбора данных.

1. Обзор предметной области

Для начала необходимо рассмотреть и сравнить возможные подходы к сбору данных с датчиков.

1.1. Возможные подходы к сбору данных

Введём несколько понятий. Поскольку некоторые подходы предполагают наличие двух видов устройств в системе сбора данных, выделим устройства-шлюзы (агрегаторы) и устройства-датчики. Датчиком будем называть устройство с подключенными к нему сенсорами[1]. Таким устройством может быть как некая аппаратная платформа[2], позволяющая подключать к ней сенсоры, так и полноценный компьютер. Зачастую устройства-датчики обладают весьма ограниченными вычислительными ресурсами. Шлюзом (агрегатором) обычно является более мощное устройство, которое имеет непосредственный доступ к сети Интернет. В данном случае шлюз играет роль отправителя данных на сервер.
Возможные подходы к сбору данных:

- «Датчик-шлюз»
- «Несколько мощных шлюзов-сборщиков»
- «Mesh-сеть»
- «Нестационарный агрегатор»

1.1.1. «Датчик-шлюз»

Каждое устройство снабжается собственным выходом в сеть для отправки данных с сенсоров. Для этого устанавливается некий модем или сетевая карта на каждое устройство-датчик. Получается, что датчик в

[1]Сенсор - чувствительная часть датчика, получающая информацию из внешнего мира
[2]Примерами таких платформ могут служить Raspberry Pi или Freescale Kinetis K64

то же время является шлюзом. Таким образом, видно, что данный подход требует вычислительной мощности от датчика, что ведёт в увеличению его размеров, стоимости, энергопотребления и тепловыделения.

Рис. 1: Подход «Датчик-шлюз»

Стоит отметить, что этот вариант больше подходит для сбора данных в режиме реального времени и является довольно надёжным. Ещё один нюанс состоит в том, что не всюду есть возможность получить доступ в Интернет (например, в лесу).

1.1.2. «Несколько мощных шлюзов-сборщиков»

Этот вариант наиболее распространённый и простой. По территории расставляется некоторое количество шлюзов так, чтобы каждый датчик имел возможность подключения к одному из шлюзов. Шлюзы снабжаются выходом в Интернет (это может быть как 3G модем, спутниковая связь, так и проводное подключение). Таким образом, шлюзы связываются с датчиками, собирают с них данные и отправляют на сервер.

Такой подход вполне прост и надёжен, но также есть свои недостатки. Во-первых, каждый шлюз требует гораздо более мощного источника питания, чем датчик. Поэтому могут возникнуть проблемы с питанием шлюзов, если рядом нет стационарного источника. Во-вторых, стоимость шлюзов может сильно превышать стоимость датчиков. Поэтому в случаях с большим количеством шлюзов общая стоимость системы может оказаться велика.

Рис. 2: Подход «Несколько мощных шлюзов-сборщиков»

1.1.3. «Mesh-сеть»

Гораздо более интересный и необычный подход. Предположим, что есть всего один или несколько датчиков, имеющих доступ в сеть (в данный момент не важно каким образом: самостоятельно или с помощью шлюза), а остальные датчики могут передавать информацию только между собой. Тогда если каждый датчик имеет связь хотя бы с одним соседним, то любой датчик может передать свои данные на сервер посредством своих соседей.

Рис. 3: Подход «Mesh-сеть»

Из достоинств следует отметить гибкость и масштабируемость такой сети. С одной стороны, стоимость такой системы не должна быть высокой, т.к в принципе может хватить даже одного шлюза или одного датчика с модемом, а остальные датчики могут оставаться дешёвыми.

Но здесь есть свой нюанс: в такой сети каждый датчик должен производить маршрутизацию, что накладывает свои требования на вычислительную мощность, а как следствие и на энергопотребление и стоимость. Поэтому этот подход может оказаться не таким выгодным, как с первого взгляда. К тому же, как показывают исследования, такая архитектура обладает низкой надёжностью ввиду некоторых физических причин.

1.1.4. «Нестационарный агрегатор»

Также нестандартный подход[2] к сбору данных. Он подразумевает, что сбор ведётся с некоторой периодичностью или по расписанию и не пригоден для сбора в реальном времени. Каждый датчик оснащается модулем беспроводной передачи данных (например, Bluetooth Low Energy), а в роли агрегатора выступает некоторое мобильное устройство (например, смартфон), также оснащенное передатчиком. Датчики обладают собственным хранилищем, что позволяет сохранять полученные с сенсоров данные до появления агрегатора в радиусе видимости.

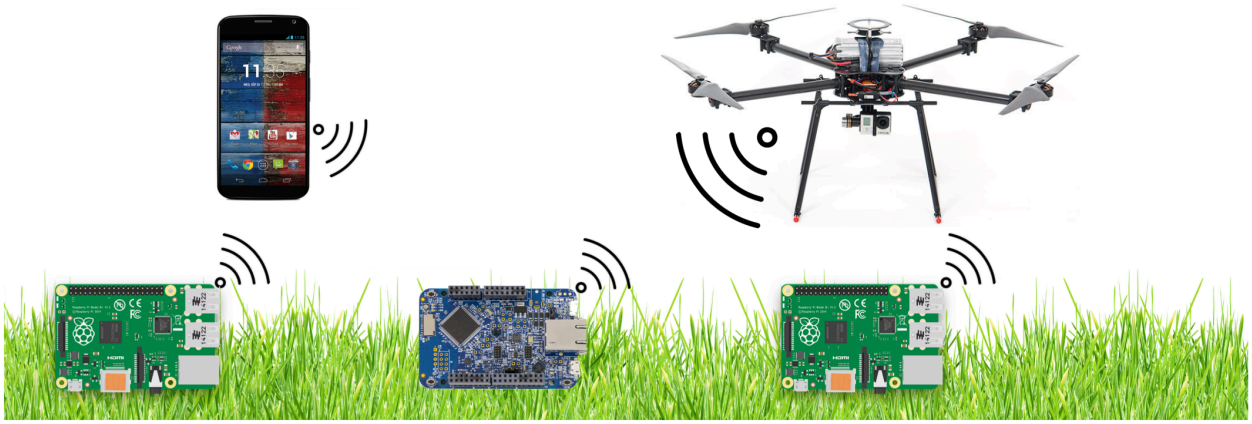

Рис. 4: Подход «Нестационарный агрегатор»

В соответствии с расписанием датчики активируют свои модули передачи данных, предоставляя возможность мобильному агрегатору обнаруживать их. Как только агрегатор обнаруживает датчик, между ними устанавливается соединение и начинается передача всех накоплен-

ных датчиком данных. Далее мобильный агрегатор имеет возможность отправить собранные данные на сервер для дальнейшей обработки.

Данный подход отличается от предыдущих тем, что на территории, где располагаются датчики, не требуется иметь доступ к сети Интернет. Это значительно повышает автономность системы и делает экономически менее затратным её построение. Также стоит отметить то, что датчики работают независимо друг от друга и их местоположение определяется только физической доступностью с точки зрения агрегатора. Это делает систему легко масштабируемой и гибкой.

Вывод: поскольку подход, описанный в пункте 1.1.4, при определенных условиях имеет ряд преимуществ перед другими подходами, имеет смысл рассмотреть существующие решения для построения таких систем. В данной работе рассмотрение будет вестись с точки зрения устройств-датчиков.

1.2. Решения для построения систем сбора данных

На сегодняшний день подход «Нестационарный агрегатор» не является широко распространенным. В связи с этим был найден всего один фреймворк[9], позволяющий строить системы подобного рода. Поэтому кроме этого фреймворка рассмотрим некоторые решения для сбора данных, которые изначально не ориентированы, но возможно могут быть использованы для сбора нестационарным агрегатором. Таких решений[7] огромное количество, поэтому было выбрано несколько из них для примера.

С точки зрения устройств-датчиков, среди ключевых свойств решений для построения рассматриваемых систем выделим следующие:

1. Отсутствие требования подключения устройств-датчиков к Интернет
2. Независимость от протокола передачи и формата данных
3. Мультиплатформенность

4. Низкие ограничения на задание пользовательской логики получения данных и их предобработки

5. Предоставление дополнительной функциональности для самостоятельной работы устройств-датчиков (хранение собранных данных до появления агрегатора в зоне видимости, включение/выключение модулей передачи данных по расписанию и т.п.)

6. Открытость программного кода

1.2.1. Open Data Kit Sensors

- Не требует подключения к Интернет
- Не фиксирует протокол и формат передачи данных
- Взаимодействие с низкоуровневыми сенсорами (I2C, SPI,..) только через Arduino и только по USB
- Агрегаторская часть фреймворка доступна только для Android
- Ограничивает свободу задания логики взаимодействия с датчиками (получение данных с сенсоров только с постоянной частотой)
- Не предоставляет хранилище данных на стороне датчиков
- Открытый программный код[9]

1.2.2. Carriots

- Разработчики указывают: «подключайте любое устройство к Carriots[5]; всё, что Вам нужно - доступ к Интернет»
- Использует REST API
- Не имеет дополнительной поддержки для самостоятельной работы устройств-датчиков
- Проприетарное ПО

1.2.3. ThingSpeak

- Предлагает сервер на Ruby On Rails в качестве агрегаторской части, что может избавить от необходимости подключения датчиков к Интернет
- Использует REST API
- Не имеет дополнительной поддержки для самостоятельной работы устройств-датчиков
- Открытый программный код[11]

Вывод: Существующие инструменты не ориентированы на сбор данных с помощью нестационарного агрегатора, что означает необходимость адаптации этих инструментов для такого подхода. В связи с этим было бы удобно иметь некий фреймворк, который обладал бы перечисленными свойствами и позволял адаптировать существующие решения/платформы Интернета Вещей для сбора данных нестационарным агрегатором.

2. Постановка задачи

Целью данной работы является разработка части, отвечающей за работу устройств-датчиков, фреймворка, предназначенного для организации систем сбора данных нестационарным агрегатором и соответствующего следующим требованиям:

- Не должен требовать подключения к Интернет
- Независимость от протокола и формата передачи данных
- Расширяемость и низкая связность компонентов
- Мультиплатформенность: запуск возможен не только на специфических платформах, но и на обычном компьютере
- Минимальные ограничения на задание пользовательской логики получения данных и их предобработке
- Предоставление дополнительной функциональности для самостоятельной работы устройств-датчиков

Для достижения цели были поставлены следующие задачи:

- Проработать архитектуру фреймворка
- Выбрать программную платформу в качестве основы для фреймворка
- Реализовать фреймворк на выбранной программной платформе
- Построить аппаратный прототип, используя разработанный фреймворк

Примечание: в данной работе рассматривается разработка первой из указанных далее частей фреймворка. Фреймворк состоит из двух частей: часть для устройств-датчиков и часть для нестационарного агрегатора[3]. Далее под словом фреймворк будем подразумевать разрабатываемую часть фреймворка, если не указано иное.

[3]Разработка части нестационарного агрегатора рассмотрена в бакалаврской работе Павлова В.А.

3. Описание инструментария

3.1. Программная платформа для фреймворка

В качестве основы для фреймворка была выбрана Java Micro Edition Embedded 8.[10]. Это бесплатная программная платформа, предложенная компанией Oracle. Характеристики:

- Основана на всем известной Java ME, к которой была добавлена поддержка специфических функций для встроенных систем
- Оптимизирована для запуска на устройствах с ограниченными ресурсами (требует 128КБ RAM) и энергопотреблением
- Предоставляет API для доступа к портам GPIO, I2C, SPI, UART
- Поддерживает механизмы безопасной передачи данных (TLS 1.2)
- Java ME SDK предоставляет эмуляторы для Windows и Linux
- Поддерживает такие платформы, как ARM Cortex-M3/-M4, ARM 9/BREW MP, ARM 11/Linux OS
- Возможен запуск на Cortex-M4 под управлением ОС mbed[13]
- Для прототипирования есть поддержка платформ Raspberry Pi и Freescale FRDM-K64F

Также важным моментом является то, что язык Java ME Embedded совместим с языком Java SE/EE, что позволит, при соблюдении некоторых правил во время разработки фреймворка, использовать этот фреймворк как на платформе Java ME, так и на Java SE/EE, что обеспечит мультиплатформенность.

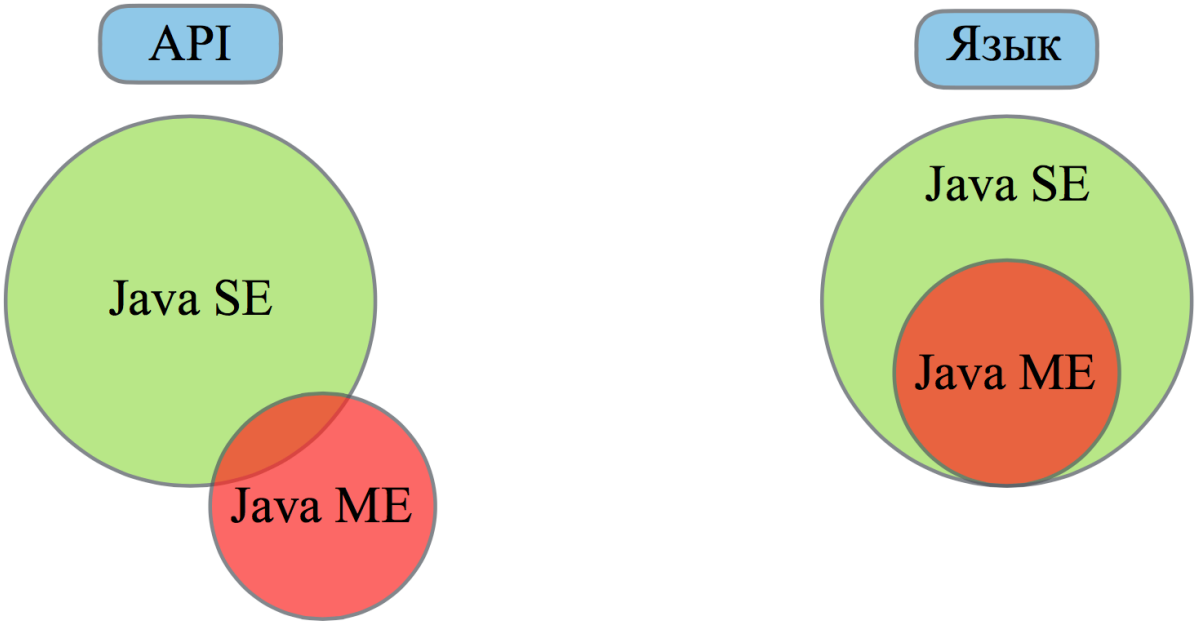

Рис. 5: Совместимость Java ME Embedded и Java SE

4. Проектирование фреймворка

4.1. Функциональность и требования

В соответствии с описанием, предложенным в пункте 1.1.4, подхода к сбору данных, выделим основную функциональность, которую будет предоставлять фреймворк на устройстве-датчике:

- Получение информации с сенсоров
- Предобработка этой информации (в случае, если это позволяют вычислительные способности устройства)
- Сохранение информации
- Управление модулем передачи данных
- Обнаружение агрегатора
- Извлечение данных из хранилища
- Отправка данных агрегатору
- Удаление данных, которые были успешно отправлены
- Планирование вышеперечисленных действий

При этом необходима максимальная гибкость задания пользовательской логики. Например, планирование действий может быть как по расписанию (по таймеру), так и в зависимости от других событий; получение информации с сенсоров может производиться как путем периодичного опроса сенсора, так и путем подписки на события сенсора (некоторые сенсоры имеют только 2 состояния: 0 и 1; в таком случае нет смысла постоянно опрашивать сенсор, достаточно лишь «слушать» его события). В связи с этим логика компонентов фреймворка должна быть легко переопределяема, а сама архитектура не должна накладывать ограничения на возможные сценарии получения информации с сенсоров.

4.2. Компоненты

На основе предъявленных требований к функциональности и гибкости, была выработана примерная схема, обозначающая основные компоненты фреймворка. Объединенные между собой, эти компоненты будут составлять некоторое монолитное приложение.

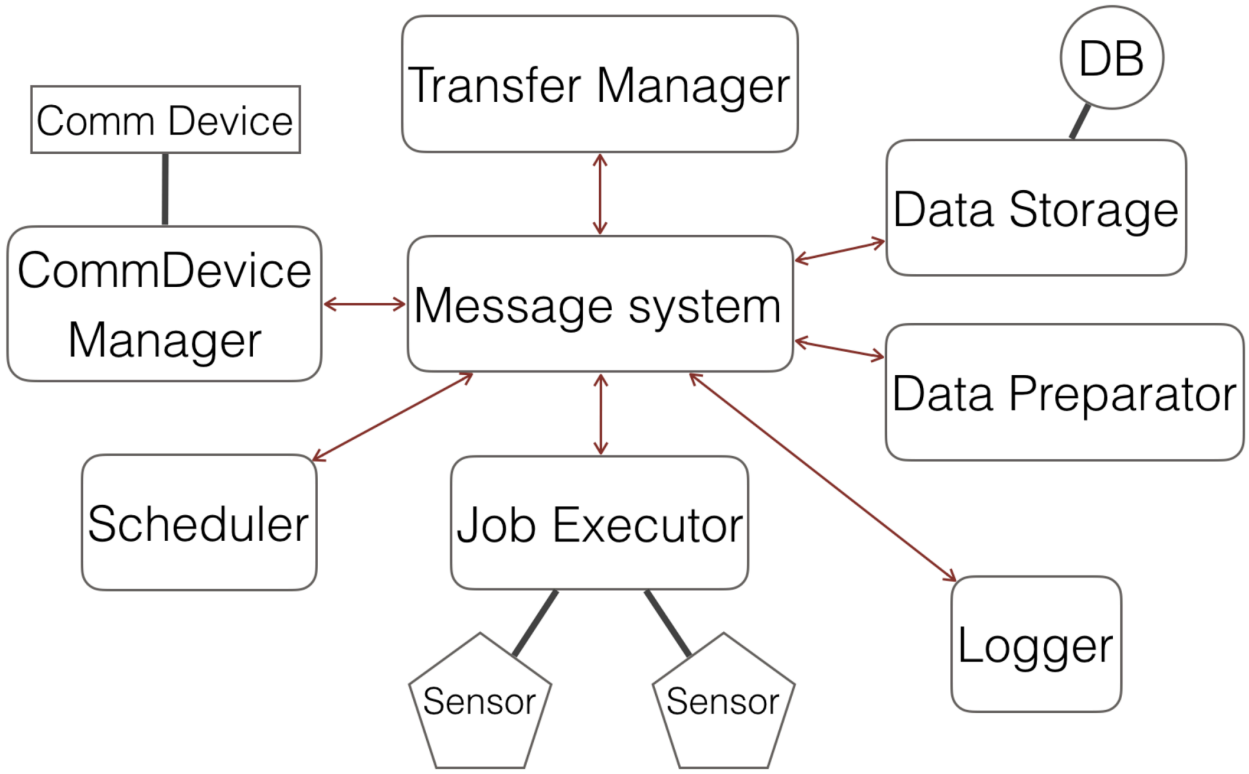

Рис. 6: Компоненты фреймворка (прямоугольники с закруглёнными краями)

- *Job Executor.* Для наиболее гибкого и удобного получения данных с датчиков была предложена следующая идея: пусть есть набор некоторых «работ» (Job), каждая из которых инкапсулирует в себе логику работы с некоторым множеством сенсоров; тогда независимые друг от друга датчики можно поместить в разные «работы», а датчики, которые работают сообща, в одну «работу». Каждая «работа» производит единицы данных (JobData) и отправляет в систему для дальнейшей обработки. Job Executor - компонент, отвечающий за выполнение этих самых Job. Он берет на себя управление потоками, запуск и остановку «работ».

- *Data Preparator.* Занимается предобработкой данных. Обрабатывает данные, опубликованные Job'ами и отправляет готовые данные на сохранение.

- *Data Storage.* Отвечает за сохранение данных в БД/файле/другой структуре и за извлечение данных оттуда по запросу.

- *Transfer Manager.* Занимается отправкой данных агрегатору. Получая оповещение об обнаружении нового агрегатора, он создает сессию передачи данных (Transfer Session), запрашивает данные из хранилища (Data Storage) и передает эти данные в сессию. Transfer Session реализует протокол передачи данных (более высокого уровня, чем физический).

- *CommDevice Manager.* Управляет модулями передачи данных. Он запускает/останавливает модули передачи данных и обрабатывает потоки, в которых выполняются их драйверы.

- *Scheduler.* Это планировщик. С помощью него можно задавать расписание включения/выключения аппаратных компонентов (сенсоров и модулей передачи данных) и расписание выполнения предобработки и сохранения данных.

4.3. Система сообщений

В целях понижения связности компонентов приложения было принято решение использовать шаблон проектирования Издатель/Подписчик. Была предпринята попытка использовать существующие библиотеки, предлагающие системы сообщений. Однако по различным причинам, вытекающим из требований к фреймворку, было решено разработать собственную простую систему, имеющую только необходимый для данного фреймворка функционал и не требующую дополнительной поддержки со стороны ОС (например, поддержку сокетов). Разработанная система сообщений обладает следующими характеристиками:

- Каждое сообщение имеет тему

- Любой объект может подписаться как на тему, так и на публикации конкретного объекта
- Существует иерархия тем
- Подписчик темы получает также все сообщения по унаследованным темам

Такое архитектурное решение позволяет легко добавлять новые компоненты в систему. Также позволяет автоматически направлять в лог все сообщения без вызова метода log, т.к логгер подписан на тему Any, от которой наследуются все остальные темы. Все компоненты системы подписаны на тему Shutdown. Сообщение с такой темой публикуется при завершении приложения. Получая такое сообщение, все компоненты готовятся к завершению работы.

5. Реализация

В соответствии с решениями, принятыми при проектировании (раздел 4), был реализован фреймворк на выбранной в разделе 3 программной платформе. Следующая схема отображает процесс работы приложения, построенного с помощью этого фреймворка.

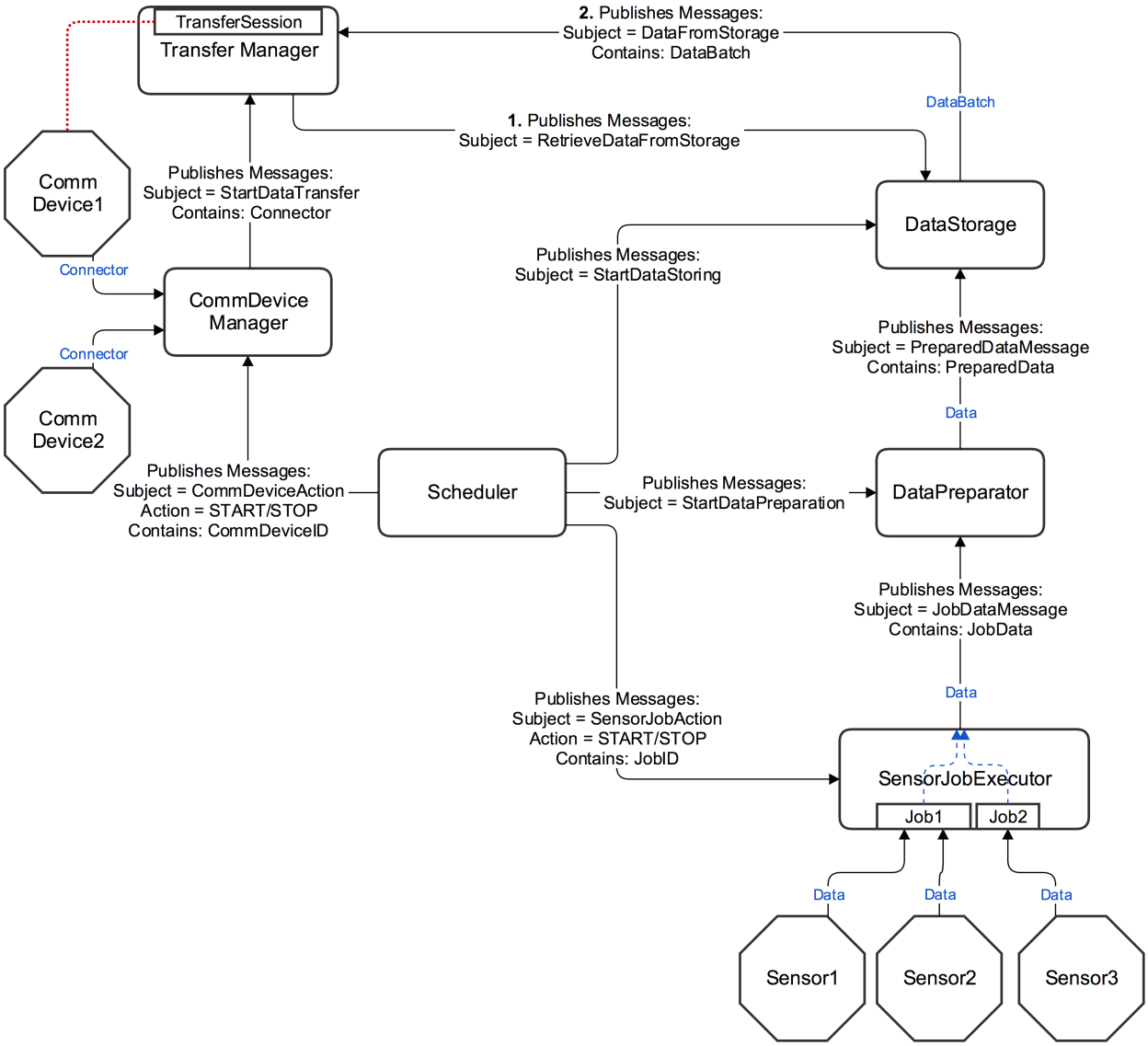

Рис. 7: Схема работы фреймворка

Голубым цветом выделены типы данных, передающихся между компонентами. Стрелочками обозначены публикации сообщений (от издателя к подписчику).

1. Планировщик (Scheduler) даёт команды на запуск ”работ” (Job), указывая их идентификаторы.

2. Сенсоры публикуют полученную из внешнего мира информацию. Тип данных наследуется от Data. Каждый сенсор может использовать свой тип данных.

3. Каждая Job подписана на публикации своих сенсоров. Агрегируя информацию со всех своих сенсоров, Job публикует данные, тип которых также наследуется от Data, но может отличаться от типа, который производят сенсоры.

4. Данные, полученные от всех ”работ”, собираются в очередь внутри Data Preparator.

5. В соответствии с расписанием планировщик даёт команду ”начать подготовку данных”.

6. Data Preparator обрабатывает данные, накопившиеся у него во входной очереди (алгоритм обработки задан пользователем). После этого он публикует эти данные. Как и раньше, выходной тип данных может отличаться от входного.

7. Data Storage собирает данные от Data Preparator к себе во входную очередь.

8. В соответствии с расписанием планировщик даёт команду ”начать сохранение данных”.

9. Data Storage сохраняет данные, накопившиеся у него во входной очереди, в долговременную память (БД/файл/иная структура данных).

10. В соответствии с расписанием планировщик даёт команду ”включить модуль передачи данных”, указывая идентификатор модуля.

11. При обнаружении агрегатора модуль передачи данных публикует некий Connector, который сопоставлен подключению к агрегатору. Через этот коннектор можно совершать отправку и получение байтов.

12. Transfer Manager получает Connector и создает сессию Transfer Session. В эту сессию он передает коннектор. Также он публикует запрос на извлечение данных из хранилища.

13. Data Storage получает запрос на извлечение данных и публикует данные "пачками" (DataBatch). Размер DataBatch задаётся пользователем.

14. Данные от Data Storage попадают во входную очередь Transfer Manager для дальнейшей отправки

15. Сессия стартует, и начинается передача данных, находящихся во входной очереди Transfer Manager. Красная линия показывает, что сессия для отправки данных использует коннектор, который в свою очередь работает через модуль передачи данных (CommDevice).

16. По завершении передачи в системе публикуется сообщение об отключении агрегатора.

6. Апробация

В качестве апробации фреймворка было решено построить прототип тестовой системы сбора данных. Датчик должен будет: получать с определенной частотой информацию с сенсора и сохранять её в БД Java RMS, при подключении агрегатора извлекать накопленные данные из БД и отправлять агрегатору. Данные передаются в формате JSON.

6.1. Реализация прототипа на основе разработанного фреймворка

Система сбора данных в нашем случае должна состоять как минимум из двух аппаратных устройств: агрегатора и датчика. Далее будет рассмотрено построение датчика. Для построения датчика требовалось выполнить следующие задачи:

Аппаратная часть

1. Выбрать аппаратную платформу для датчика
2. Выбрать хотя бы один сенсор, совместимый с аппаратной платформой
3. Выбрать модуль передачи данных, совместимый с аппаратной платформой (если он не встроен в аппаратную платформу)
4. Выбрать дополнительные электронные компоненты, необходимые для подключения сенсора и модуля передачи данных
5. Собрать схему из перечисленных ранее электронных компонентов

Программная часть

1. Написать драйвер модуля передачи данных
2. Написать драйвер(ы) для сенсора(ов) и описать логику получения данных
3. Описать тип(ы) и формат(ы) данных и методы сериализации этих данных

6.1.1. Аппаратная часть

В качестве аппаратной платформы была выбрана Raspberry Pi 2 Model B. Эта платформа поддерживается Java ME Embedded. Также она предоставляет порты GPIO, I2C, SPI, UART для подключения цифровых периферийных устройств, в том числе сенсоров и модулей передачи данных.

Также был выбран сенсор Ultrasonic HC-SR04[6], подключающийся через GPIO. Данный сенсор предназначен для измерения расстояния до объектов с помощью ультразвуковых импульсов. Требуется источник питания 5В. Работает сенсор следующим образом: на контакт Trig подаётся логическая единица (напряжение 2.5-5В) длительностью 10мкс, сенсор генерирует звуковой импульс на частоте 40кГц и слушает эхо. Получая эхо, выдаётся 5В на контакт Echo. По времени между подачей Trig и получением Echo определяется расстояние до объекта.

Рис. 8: Ultrasonic HC-SR04

Передачу данных было решено осуществлять по протоколу Bluetooth Low Energy, ввиду его распространенности и поддержки современными смартфонами. Для прототипа использовался модуль Bolutek CC41-a[4], подключающийся посредством UART. Управление модулем осуществляется с помощью AT-команд[1]. Рассчитан на источник питания 3.3В.

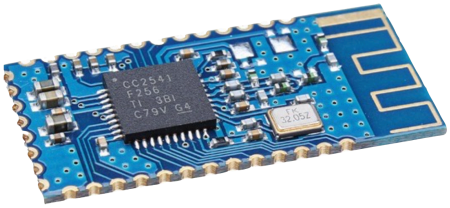

Рис. 9: Bolutek CC41-a

При подключении сенсора к платформе возникла проблема: сенсор выдаёт на контакт Echo 5В, а GPIO контакты Raspberry Pi рассчитаны на напряжение до 3.3В, в связи с чем при прямом подключении возможно повреждение Raspberry Pi. Поэтому было решено сделать резистивный делитель напряжения для понижения напряжения на контакте Echo. Для этого были выбраны 2 резистора по 470 Ом. Таким образом, используя два одинаковых резистора в делителе, получаем напряжение 5В / 2 = 2.5В.

На основе выбранных компонентов была собрана и протестирована следующая схема. Схема была собрана на беспаечной макетной плате.

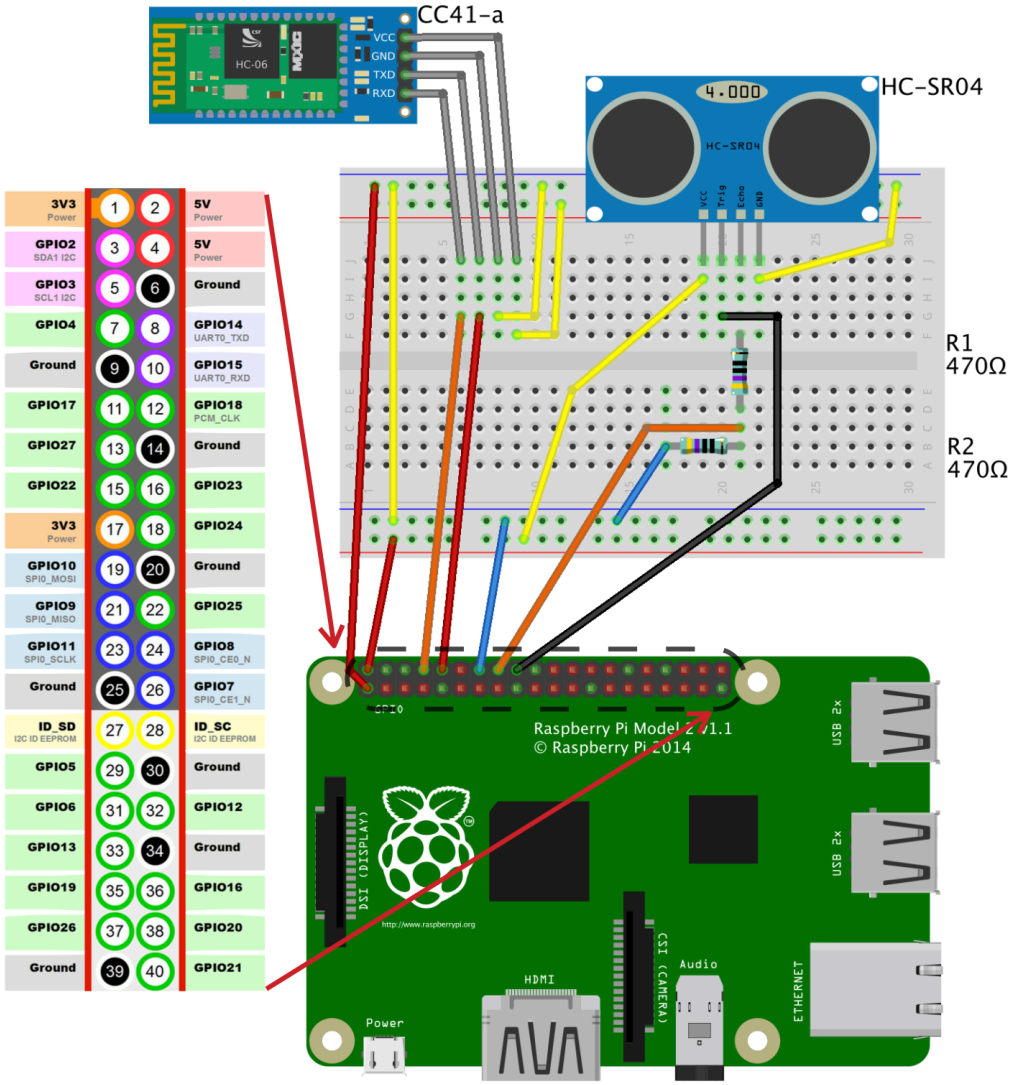

Рис. 10: Схема прототипа

6.1.2. Программная часть

При написании драйвера модуля передачи данных был использован API JavaME Embedded для работы с портом UART. Для управления модулем было необходимо отправлять ему команды вида «AT+<слово>». При старте драйвер посылает команду «AT+RESET» для перезагрузки модуля. Когда внешнее устройство подключается к модулю, драйвер получает сообщение «OK+CONN», что означает, что можно начинать передачу. Сообщение «OK+LOST» означает, что связь разорвана. Драйвер реализует интерфейс UARTEventListener, чтобы получать уведомления о появлении новых данных в порте UART.

Драйвер сенсора пользуется Java MEE API для работы с GPIO. Реализованный драйвер рассчитан только на синхронное считывание информации с сенсора (предоставляет метод read). После реализации самого драйвера, необходимо описать логику получения информации с сенсора. Это делается путём наследования от абстрактного класса SensorJob и реализации его абстрактных методов. Ниже приведён пример простейшей реализации этого класса.

```
public HCSR04Job(JobID id) {
        super(id);
        hcsr04Sensor = (Sensor<HCSR04Data>) Core.getFactory().createSensor(new SensorID(4));
}

@Override
public void execute() throws InterruptedException, PLSOSException {
        hcsr04Sensor.start();
        while (true) {
                publishJobData(new HCSR04Data(hcsr04Sensor.read().distance));
                Thread.sleep(2000);
        }
}

@Override
protected void finish() {
        try {
                hcsr04Sensor.stop();
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
}
```

Рис. 11: Реализация класса HCSR04Job

Также необходимо описать тип пользовательских данных и метод его сериализации. В данном прототипе использовался сенсор-дальномер, выдающий число типа double. Поэтом пусть пользовательский тип данных будет содержать одно поле - число типа double. Для целей логирования все пользовательские типы данных должны переопределять метод toString. В качестве формата передачи данных в данном прототипе использовался JSON, поэтому также добавим метод toJSON для приведения данных в этот формат.

```
public class HCSR04Data extends Data {

    public final double distance;

    public HCSR04Data(double distance) {
        this.distance = distance;
    }

    @Override
    public String toString() {
        return "HCSR04Data " + distance;
    }

    public JSONObject toJSON() {
        JSONObject jsonObject0 = new JSONObject();
        JSONObject jsonObjectI = new JSONObject();
        try {
            jsonObject0.put("HCSR04Data", jsonObjectI);
            jsonObjectI.put("Distance", distance);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return jsonObject0;
    }

}
```

Рис. 12: Реализация класса HCSR04Data

В данном прототипе использовался простой метод сериализации данных. Поскольку JSON - текстовый формат данных, то для сериализации было достаточно перевести данные в формат JSON и применить к полученному тексту метод getBytes() стандартного Java класса String. Десериализация происходит в обратном порядке.

Построенный прототип был протестирован совместно с агрегаторской частью, выполненной на смартфоне с ОС Android.

Заключение

В рамках данной бакалаврской работы были достигнуты следующие результаты:

- Рассмотрены возможные подходы к сбору данных с датчиков
- Проведен обзор инструментов для построения систем сбора данных нестационарным агрегатором и выявлены недостатки этих инструментов
- Сформированы требования к функциональности и архитектуре разрабатываемого фреймворка
- Проработана архитектура фреймворка
- Выбрана программная платформа в качестве основы для фреймворка
- Реализован фреймворк на выбранной платформе, удовлетворяющий сформированным требованиям
- Построен аппаратный прототип, использующий разработанный фреймворк
- Проведены тесты

Разработанный фреймворк исправляет обнаруженные недостатки (с точки зрения сбора нестационарным агрегатором) у существующих решений и может служить как для создания систем сбора данных с нуля, так и для адаптации существующих решений к сбору нестационарным агрегатором.

Список литературы

[1] AT COMMANDS. — URL: https://halckemy.s3.amazonaws.com/uploads/document/file/94325/FE85NGOIH90OKGT.pdf (online; accessed: 02.02.2016).

[2] Abd Elwahab Boualouache Omar Nouali Samira Moussaoui, Derder Abdessamed. A BLE-based data collection system for IoT // New Technologies of Information and Communication (NTIC), 2015 First International Conference on New Technologies of Information and Communication. — 2015.

[3] Balani Naveen. Enterprise IoT - A Definite Handbook / Ed. by Rajeev Hathi. — 2015.

[4] Bolutek BLE-CC41-a SPECIFICATION. — URL: http://img.banggood.com/file/products/20150104013145BLE-CC41-A%20Spefication.pdf (online; accessed: 02.02.2016).

[5] Carriots - Internet of Things Platform. — URL: https://www.carriots.com/ (online; accessed: 08.03.2016).

[6] HC-SR04 User Guide. — URL: http://elecfreaks.com/estore/download/EF03085-HC-SR04_Ultrasonic_Module_User_Guide.pdf (online; accessed: 01.02.2016).

[7] Internet of Things Platforms - Postscapes. — URL: http://postscapes.com/internet-of-things-platforms/ (online; accessed: 04.02.2016).

[8] J. Cecílio P. Furtado. Wireless Sensors in Heterogeneous Networked Systems. — 2014.

[9] Open Data Kit Sensors. — URL: https://opendatakit.org/use/sensors/ (online; accessed: 16.04.2016).

[10] Oracle Java ME Embedded Overview. — URL: http://www.oracle.com/technetwork/java/embedded/javame/embed-me/overview/index.html (online; accessed: 02.04.2016).

[11] ThingSpeak: Internet Of Things. — URL: https://thingspeak.com/ (online; accessed: 08.03.2016).

[12] Thomas Zachariah Noah Klugman Bradford Campbell Joshua Adkins Neal Jackson Prabal Dutta. The Internet of Things Has a Gateway Problem // HotMobile '15 Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications. — 2015.

[13] mbed OS. — URL: https://www.mbed.com/en/development/software/mbed-os/ (online; accessed: 04.04.2016).