Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Прикладних інформаційних технологій та електроінженерії
(повна назва факультету)
Комп'ютерно-інтегрованих технологій
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка автоматизованої системи організації збору даних в промисловій мережі давачів

Виконав(ла): студент(ка) 4 курсу, групи КТз-41
спеціальності 151 – Автоматизація та комп'ютерно-інтегровані технології
(шифр і назва спеціальності)

(підпис) Лисак О.Л. (прізвище та ініціали)

Керівник (підпис) Стухляк П.Д. (прізвище та ініціали)

Нормоконтроль (підпис) Чихіра І.В. (прізвище та ініціали)

Завідувач кафедри (підпис) Микитишин А.Г. (прізвище та ініціали)

Рецензент (підпис) Шовкун О.П. (прізвище та ініціали)

Тернопіль
2021

АНОТАЦІЯ

Кваліфікаційна робота бакалавра складається з пояснювальної записки та графічної частини (ілюстративний матеріал – слайди).

Об’єм графічної частини роботи становить _16__ слайдів.

Об’єм пояснювальної записки складає 56 друкованих сторінок формату А4 (210×297), об’єм додатків – - друкованих сторінок формату А4.

Робота складається з чотирьох розділів, в яких нараховується 17 рисунків та 4 таблиць з даними.

В роботі використано 16 літературних джерел.

Метою даної роботи є розробка частини, що відповідає за роботу пристроїв-датчиків, фреймворка, призначеного для організації систем збору даних нестаціонарним агрегатором і відповідає наступним вимогам:

- не повинен вимагати підключення до Інтернет;
- незалежність від протоколу і формату передачі даних;
- можливість розширення та низька зв'язність компонентів;
- мультиплатформеність: запуск можливий не тільки на специфічних платформах, але і на звичайному комп'ютері;
- мінімальні обмеження на завдання користувальницької логіки отримання даних і їх передобробці;

Розроблений фреймворк виправляє виявлені недоліки (з точки зору збору нестаціонарним агрегатором) у існуючих рішень і може служити як для створення систем збору даних з нуля, так і для адаптації існуючих рішень до збору нестаціонарним агрегатором.

Ключові слова: ФРЕЙМВОРК, ДАТЧИК, ПЛАТФОРМА, КОМПОНЕНТИ.

ЗМІСТ

ВСТУП

Сьогодні назви «розумний браслет», «розумні годинник», «розумний дім» зустрічаються дуже часто. Як правило, будь-яка «розумна» річ має можливість підключення до мережі Інтернет [1]. Звідси і з'явився термін «Інтернет речей» (IP). Цей термін приховує за собою вельми популярну на сьогоднішній день концепцію, яка полягає в тому, що в сучасному світі використання мережі Інтернет для обміну інформацією стало можливим і навіть необхідним не тільки для повноцінних комп'ютерів, а для будь-яких пристроїв, будь то автомобіль або побутова техніка . Ці «розумні» пристрої служать як для підвищення комфорту життя людей (інтелектуальне управління кліматом в будинку, освітленням в будинку і на садовій ділянці), так і для виконання складної інтелектуальної роботи (наприклад, аналіз стану екології).

Одним з важливих процесів в концепції Інтернету речей є збір даних про будь-які події або навколишньому середовищу. Наприклад, збір і правильний аналіз даних про вологість і температурі з полів може допомогти в плануванні і організації посадки овочів або інших рослин.

Сьогодні у багатьох областях IP стала актуальна проблема регулярного збору даних з певною періодичністю з датчиків, розподілених по великій території. Зокрема, отримав популярність особливий вид збору даних [2], при якому пристрій, що збирає дані (далі агрегатор) переміщається в просторі, а пристрої, які отримують дані із зовнішнього середовища і накоплені їх (далі датчики) нерухомі . При такому підході до збору даних датчики можуть розташовуватися в місцях, де відсутня вихід у глобальну мережу. З цього можна зробити висновок, що даний вид збору невибагливмй до навколишнього середовища, ніж, наприклад, підхід, при якому є ряд стаціонарних агрегаторів-шлюзів, що мають вихід в глобальну мережу і періодично збирають дані з датчиків у своїй зоні видимості. Також використання даного виду збору дозволяє знизити потенційну вартість рішення, так як при такому вигляді збору знижуються вимоги на апаратні складові системи. Адже тепер як нестаціонарного збірника даних може

виступати недорогий смартфон, нетбук і будь-яке інше щодо малопотужності пристрій.

Безліч завдань, що вирішуються за допомогою використання виду збору нестаціонарним агрегатором, широке. Наприклад, збір нестаціонареним агрегатором можливий в наступних сценаріях:

- Пристрій на автомобілі збирає інформацію про потенціальну дорожню небезпеку з датчиків, встановлених уздовж автомагістралі
- Пристрій сільського працівника (наприклад, квадрокоптер) при попаданні в зону видимості мініатюрних датчиків, що розкидані по великій зоні, отримує сповіщення про різні властивості грунту

Достаток сфер застосування обговорюваного виду збору ставить розробників систем збору даних перед однією з таких проблем:

- Проблема пошуку програмних засобів для швидкої побудови систем періодичного збору даних нестаціонарним агрегатів
- Проблема трудомісткою адаптації існуючих інструментів до підходу з нестаціонарним агрегатором.

Тому створення деякого фреймворка для організації мереж з нестастаціонарного збирання даних змогло б значно полегшити роботу розробників і адміністраторів таких мереж при вирішенні вказаних проблем.

1. АНАЛІТИЧНА ЧАСТИНА

1.1 Постановка задачі

Основною метою даної роботи є розробка агрегаторської частини фреймворка, призначеного для організації систем періодичного збору даних в мережах датчиків за допомогою нестаціонарного агрегатора. Агрегаторскій компонент фреймворка повинен мати наступні властивості:

- Гнучка архітектура: агрегаторский додаток будується по частинах, окремі частини програми можуть бути швидко замінені на нові, агрегаційні процес конфігуруємо
- Мультиплатформеність і невисокі апаратні вимоги: аг-регаторский компонент повинен бути спроектований з урахуванням можливості розробки систем під великий сегмент пристроїв, який включає в себе смартфони і пристрої порівнянні з ними по потужності
- Незалежність від конкретних протоколів спілкування між компо-нентами системи збору
- Незалежність від формату даних, що передаються між компо-нентами системи збору
- Відносна простота використання агрегаторского компонента фреймворку для розробників

Для досягнення мети були поставлені такі завдання:

- Огляд феймворків аналогічних цільовому
- Вибір програмної платформи в якості основи для розробки агрегаторской частини фреймворка
- Розробка архітектури агрегаторской частини фреймворка, дозволяє розробникам легко здійснювати створення різному параметризованих нестаціонарних агрегаторів

- Створення агрегаторского прототипу на основі розробленого фреймворку

1.2 Огляд існуючих аналогів

В сфері Інтернету Речей існує величезна кількість платформ для організації різних розумних систем. У цій роботі не проводився огляд всіх можливих аналогів. Розглядалися лише ті існуючі системи, які вирішували проблему організації збору даних і дозволяли будувати по-різному параметризрвані рішення.

ODK Sensors

ODK Sensors [3] є найближчим аналогом розроблюваного фреймворка. Даний фреймворк був створений для полегшення роботи розробників при написанні додатків під платформу Android для збору даних з внутрішніх або зовнішніх датчиків. Фреймворк не має поділу на компоненти для агрегатора і датчиків, і вся настройка системи ведеться на стороні збирача. Як стверджують розробники фреймворка, деякі типові датчики підтримують тільки такі низькорівневі I / O інтерфейси, як I^2C, *SP I* та інші, що ускладнює зовнішнє конфігурація таких датчиків. Фреймворк дозволяє будувати додатки для збору даних, які можуть варіюватися в наступних напрямках: протокол спілкування з датчиками, формат зібраних даних, конфігурація датчиків. Фреймворк надає окремі інтерфейси для розробників додатків, розробників драйверів датчиків і розробників самого фреймворка. Серед основних переваг фреймворка при вирішенні завдання організації збору даних нестаціонарним агрегатором можна виділити:

- Фреймворк допускає відсутність виходу в Інтернет у датчиків
- Гнучка архітектура фреймворка

Серед недоліків фреймворка при вирішенні поставленого завдання можно зазначити:

- Фреймворк не надає вбудованої підтримки вивантаження дан-них на віддалений сервер

- Централізована конфігурація на стороні збирача увеличи-кість трудомісткість налаштування деяких типових датчиків
- Фреймворк придатний для розробки тільки під платформу Android

ThingSpeak

ThingSpeak [5] є безкоштовним онлайн сервісом, який дозволяє організовувати збір даних з датчиків з подальшим збереженням в хмарі. Даний сервіс надає деякий API для швидкої побудови власного додатка. Даний сервіс має функціонал, що дозволяє проводити аналіз зібраних даних, будувати їх різні візуальні подання за допомогою MATLAB, створювати подієві тригери і інше. ThingSpeak використовує концепції каналів для організації збору інформації з датчиків. Канали надають інтерфейс для читання і запису і зберігають всю інформацію, яку в них записали пристрою. Для того, щоб почати роботу в системі ThingSpeak досить зареєструватися в системі, створити канал, налаштувати датчик на відсилання даних в створений канал і при необхідності налаштувати візуальне відображення отриманих даних. Як тільки канал зареєстрований в ThingSpeak, він здатний приймати, обробляти дані і надавати до них доступ зовнішнім додатками. Канали підтримують маніпуляції з даними в форматах JSON, XML і CSV. ThingSpeak використовує HTTP протокол для прийому даних з датчиків. Для настройки власного сервера збору даних потрібно Ruby on Rails.

Таким чином, серед переваг даної системи для вирішення постав-лених завдань можна відзначити:

- Простота використання
- Надання безлічі налаштувань збору і подальшої обробки даних

Серед недоліків системи при її використанні для вирішення поставлених завдання можна відзначити:

- Спілкування в даній системі засновано на фіксованому протоколі і не забезпечує необхідної протоколонезалежності
- Система націлена на збір даних в реальному часі
- Трудомісткий запуск сервера на малопотужному смартфоні через ситемні вимоги Ruby On Rails

Axibase Collector

Axibase Collector [5] є окрема Java програма, яка дозволяє організовувати збір інформації різної природи з різних зовнішніх джерел даних. Після збору даних дані відправляються в Axibase Time Series Database [6]. Різні агрегаторскі процеси можуть бути задані за допомогою створення спеціальної Job з певним типом, що характеризує формат зыбраних даних і протокол спілкування з джерелами даних. Можливо завдання розкладу Job за допомогою cron виразів. Axibase Collector вимагає Java версії 1.7 або вище. Продукт запускається на пристроях з як мінімум 1 процесором з тактовою частотою від 2 GHz, оперативно-ної пам'яттю від 1 GB і з такими операційними системами як Ubuntu 14+, RedHat Enterprise Linux 6 +, Suse Linux Enterprise 11, Debian 6 +, CentOS 6 +. Таким чином, серед переваг даного продукту для вирішення поставленого завдання варто відзначити:

- Надання можливості завдання розкладу збору даних
- Взаємозамінністю формату даних, що збираються і протоколу спілкування з джерелами

Серед недоліків продукту при вирішенні поставленого завдання можна виділити:

- Високі системні вимоги: Axibase Collector не зможе запуститися на малопотужному смартфоні
- відсутність можливості завдання обробки даних на самому агрегаторі
- Фіксована кінцеве сховище даних

SDCF

SDCF [7] - фреймворк для збору даних з датчиків Android пристроїв з відкритим вихідним кодом. Даний фреймворк надає можливість гнучкої конфігурірації процесу збору даних з датчиків різного типу на Android пристроях. Зібрані дані зберігаються в локальній базі даних на пристрої. Фреймворк дозволяє організовувати періодичну відправку даних на віддалений сервер в форматі XML. Також даний фреймворк може розсилати зібрані дані інших додатків на Android пристрої. Проект має свій портал з доступною Java_doc документацією. Таким чином, серед переваг даного продукту для вирішення поставленого завдання варто відзначити:

- Надання можливості організації періодичного збору даних за допомогою Android пристрої
- Гнучка конфігурованість процесу збору даних
- Можливість роботи з датчиками різного типу

Серед недоліків продукту при вирішенні поставленого завдання можна виділити:

- Фіксована платформа, під яку рішення можуть бути збудовані за допомогою фреймворка
- Фіксований формат відсилаються даних
- В архітектурі SDCF пристрої датчиків і пристрій агрегатів збігаються

Висновки

Існуючі інструменти для організації збору даних недостатньо адаптовані для організації збору даних за допомогою нестаціонарного збирача. Наведені приклади систем ілюструють деякі з найбільш часто зустрічаються розробниками складнощів при спробі організувати систему збору даних нестаціонарним агрегатором під свій конкретний випадок за допомогою існуючих продуктів: залежність від конкретних протоколів спілкування агрегатора з датчиками, невідповідна архітектура, залежність від формату передачі даних, недостатня мультиплатформенність, неможливість запустити систему на «слабких» пристроях і інші. З огляду на вище перелічене було вирішено розробити власну архітектуру агрегаторской частини фреймворка,

яка змогла б поєднувати в собі плюси і уникати мінуси перерахованих вище систем збору. Цільовий фрейм-ворк повинен надавати інструментарій, орієнтований на сторенення гнучкої конфігурації мультиплатформових систем періодичного збору даних з датчиків нестаціонарним агрегатором. Розроблюваний фреймворк також міг би бути корисним при адаптації підходу з використанням нестаціонарного агрегатора до існуючих системам збору.

2 ПРОЕКТНА ЧАСТИНА

2.1. Розробка агрегаторского компонента і прототипування

Розробка архітектури

При створенні агрегаторского компонента фреймворка для організації систем збору даних в мережах датчиків нестаціонарним агрегатором передбачалося, що:

- У цільовій системі збору допускається використання декількох незалежних складальників
- У цільовій системі збору допускається використання віддаленого сервера для надійного збереження інформації, зібраної агрегаторами
- У цільовій системі збору агрегатори можуть заздалегідь не знати про існування датчиків, готових до передачі даних
- Компонент не вирішує питання безпеки передачі даних

Перед розробкою архітектури агрегаторской частини фреймворка потрібно було задати вимоги на архітектуру. Ці вимоги до когось компонентами були сформульовані в термінах основних можливостей, які він повинен надавати розробнику:

- Можливість конфігурації стратегії збору: можливий одно- тимчасовий запуск декількох по-різному параметрезованих про-процесів збору (збори по різних протоколах, збори різної перио-дічності з різних датчиків тощо)
- Можливість конфігурації пропускної здатності збору: кількість одночасно оброблюваних датчиків може бути задано розробником будь-яким чином сигналізувати при виявленні нового датчика)
- Можливість організації та налаштування локального довгострокового збереження даних на пристрої збирача
- Можливість швидко підміняти тип даних, що збирають з датчиків: система не повинна бути залежна від даних, які вона збирає з датчиків
- Можливість налаштування відправки даних на віддалений сервер: відправка даних на сервер може проводитися за різними пропротоколами, в різному форматі, мати різний розклад

- Можливість настройки логістики: логістика є важливою частиною будь-якої системи, тому адміністратор системи повинен мати контроль над політикою логістики

В роботі агрегаторского компонента цільового фреймворка було виділено кілька ключових процесів: конфігурація, збір даних з датчиків, логістика і обробка подій, локальне збереження даних, вивантаження даних на віддалений сервер. Архітектура компонента була побудована таким чином, що кожен процес проходив максимально незалежно від інших. Для кожного процесу в ар-хітектури компонента можна виділити підкомпонент, що відповідає за забезпечення протікання цього процесу.

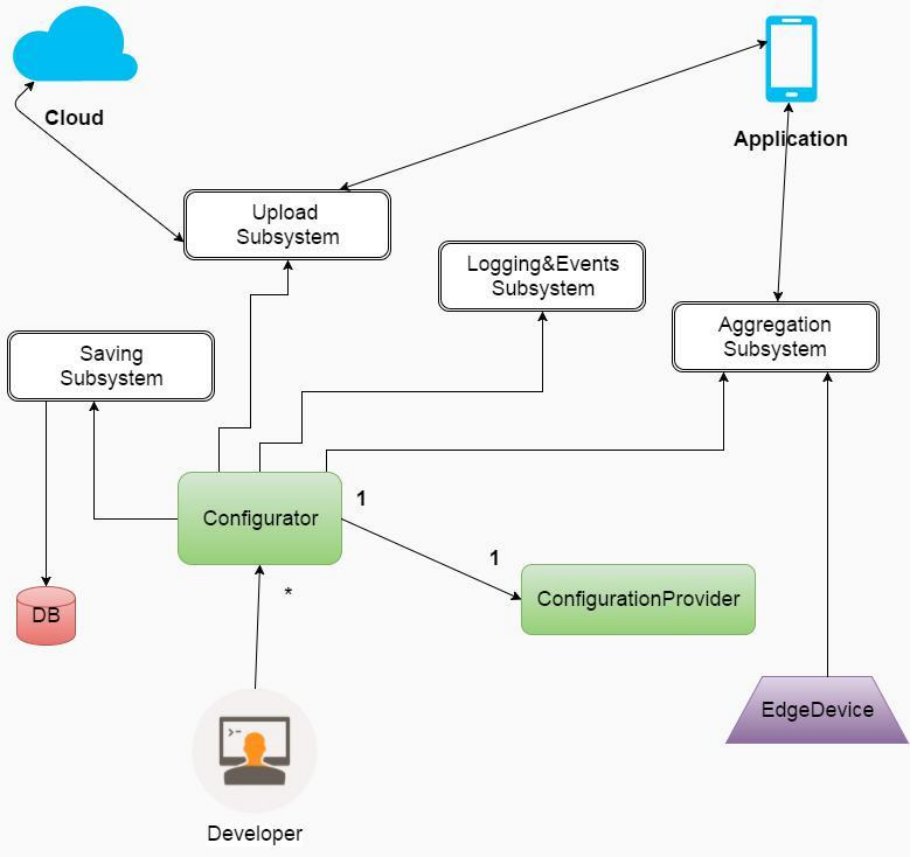
Далі буде коротко описано кожен підкомпонент. Опис кажного підкомпонента буде забезпечуватися графічним представленням його архітектури.

Конфігурація.

Зважаючи на той факт, що агрегаторскій компонент націлений на організацію різних параметрезованих систем збору даних, підкомоп-нент конфігурації системи грає одну з найбільш важливих ролей і потребує опису в першу чергу. Саме цей підкомпонент надає деякий інтерфейс для розробника системи і дозволяє останньому створювати абсолютно несхожі один на одного системи, швидко підміняючи потрібні елементи в ній.

Зауваження: Далі на всіх схемах елементи, які можуть бути легко замінені розробником на інші за рахунок підкомпонента конфігурації, будуть позначені зеленим кольором.

Загальна структура підкомпоненте конфігурації представлена на малюнку 1.



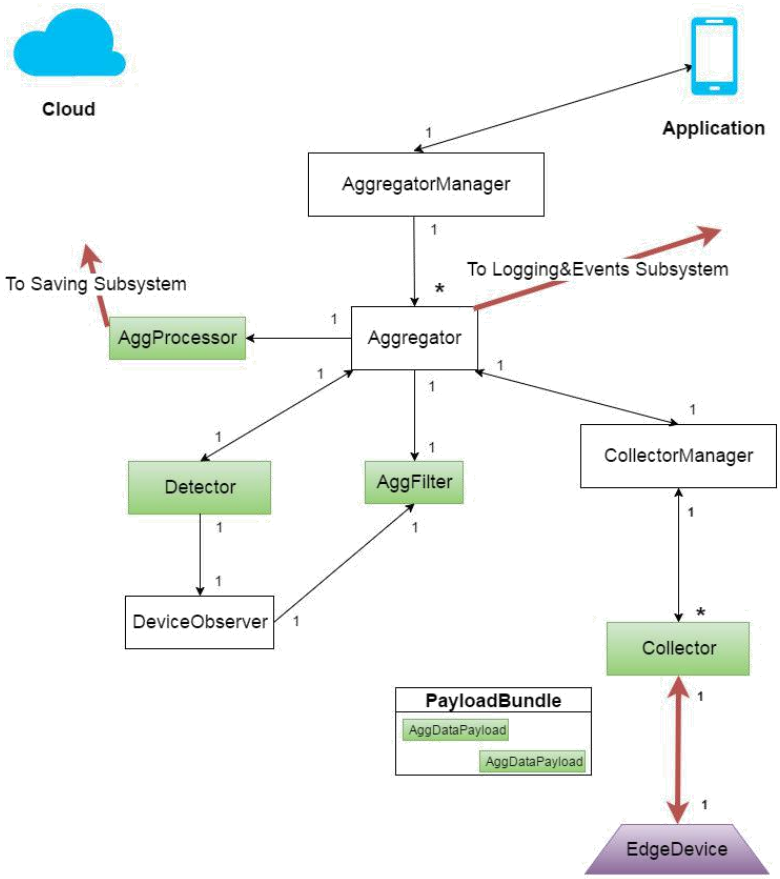
Мал. 1: Архітектура підкомпонента, що відповідає за конфігурування всієї системи

Процедура конфігурації майбутньої системи була максимально полегшена для розробника. Підкомпонент був спроектований таким чином, що створення екземплярів більшості класів, які параметризуют систему збору, контролюється самою системою, а не розробником. Розробник всього лише створює певний об'єкт, реалізующий інтерфейс *Configurator* (Далі конфігуратор), який буде будувати елементи майбутньої системи. Після створення почпткового класу свого конфігуратора розробник може швидко підміняти побудова необхідних елементів, перевизначаючи методи, що їх створює. Якщо розробник не хоче створювати свого конфігуратора, визначаючи всі методи інтерфейсу *Configurator*, він може скористатися стандартної реалізацією *Def aultP lsos Configurator*, Наслідуючи від цього класу і перевизначаючи потрібні методи. Про реалізаціях стандартної більш детально мова піде при описі інших підкомпонентів. Конфігуратор буде також надавати системі об'єкт, реалізующий інтерфейс *Configuration P rovider*(Далі надавач) і що містить всі

необхідні елементарні настройки для цих побудов. Далі елементи можуть будуватися за допомогою конфігуратора з використанням загальних параметрів надавача. Якщо в якийсь момент часу слід поміняти глобальні настройки об'єкт представника може бути підмінений в конфігураторі. Якщо такі зміни занадто глобальні для системи, розробник може створити спеціальний пакет налаштувань для потрібного елемента і побудувати його. Варто також замітити, що архітектура підкомпонента досить гнучка, і розробник може створювати кількох конфігураторів зі своїми представниками, якщо того вимагає система. В архітектурі описуваного підкомпонента використовувався шаблон проектування AbstractFactory [8].

2.2. Збір даних з датчиків

Підкомпонент збору даних є одним з найбільш важливих підсистем агрегаторского компонента. Зважаючи на це факту даний підкомпонент буде описаний детально. Розроблена архітектура підкомпонента представлена на малюнку_2.



Мал. 2: Архітектура підкомпонента, що відповідає за збір даних з датчиків

Цей підкомпонент відповідальний за виявлення датчиків, забезпе-чення збору даних, тимчасове зберігання зібраних даних, а також їх фільтрацію і обробку. Підкомпонент управляється безпосереднього самим додатком, надаючи останньому деякий інтерфейс через Singleton [8] клас *Aggregator Manager*. Даний клас є певний контейнер об'єктів класу *Aggregator* і управляє їх створенням, запуском, зупинкою, модифікацією. Таким чином, один агрегаційний процес асоціюється з об'єктом класу *Aggregator*, який може бути по-різному параметризрваний. Одними з найважливіших параметрів є конфігурації об'єктів *Collector*, *Detector*, *AggF ilter*. Більш детально ці класи і їх реалізації по замовчуванням описані нижче.

- *Collector*

Абстрактний клас, який відповідає за взаємодію з датчиком. Інкапсулює всю логіку відправки і прийняття інформації з датчиків. Як об'єкт класу *Collector* (Далі колектор) за замовчуванням будь-який конфігуратор буде повертати об'єкт класу *Def aultDebugCollector*, "Збирає" довільне число штучно генеруємих пакетів. Один колектор веде спілкування з одним датчиком. Кілька колекторів можуть бути запущені одночасно для паралельної обробки декількох датчиків (число регулюється налаштуванням *threadCount*). Життєвий цикл декількох колекторів контролюється об'єктом класу *CollectorM anager*. реалізуючи клас *Collector*, розробник вказує найбільшу частину протоколозалежної поведінки його системи. Реалізація цього класу зводиться до визначення декількох методів: установка з'єднання, ідентифікація сторін, прийняття деякої "порції" даних, відсилання підтвердження. Саме така послідовність дій формує все спілкування збірника з датчиком. Формат даних легко замінюємо. Всі дані, отримані з зовнішнього світу датчиком, обгорнуті на стороні агре-Гатор в об'єкти класу *AggDataP aylod* (Далі об'єкти-дані). Реалізація класу *AggDataP aylod* вводить новий тип даних, збирає системою. Самі об'єкти-дані передаються "порціюями" (по кілька штук об'єктом класу *PayloadBundle*) Для зменшення обсягу переданої службової інформації.

- *Detector*

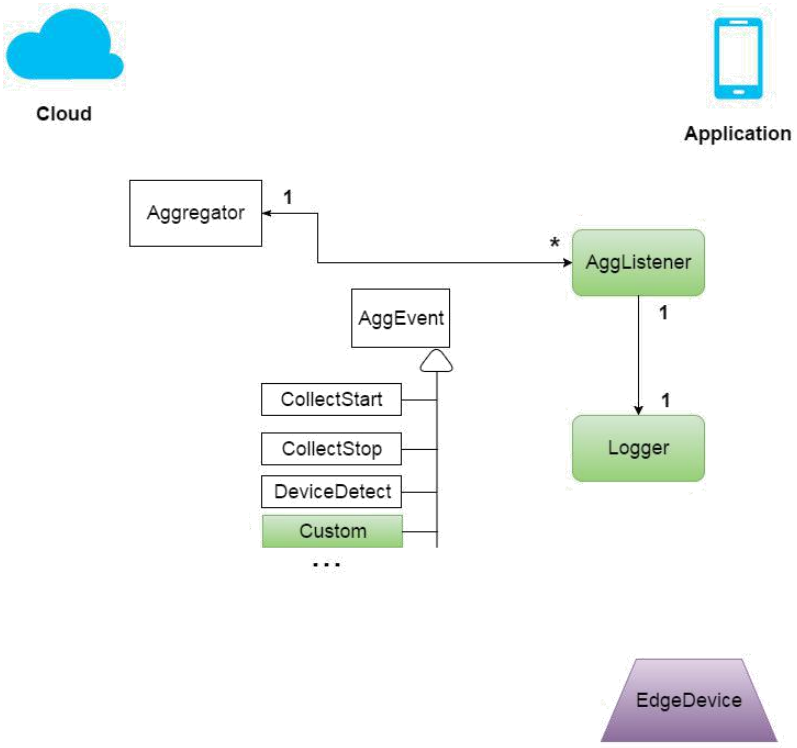
Абстрактний клас, відповідальний за виявлення датчиків. Підпідтримує циклічні опитування (налаштування *detectorRef resh*) мережі на наявність пристроїв, а також асинхронне виявлення. Як об'єкт класу *Detector* (Далі детектор) за замовчуванням будь-який конфігуратор буде повертати об'єкт класу *DefaultDebugDetector*, Довільним чином штучно "виявляє" одне з 50000 налагоджувальних пристроїв. Реалізація *Detector* є останньою протоколозалежною реалізацією елемента системи після визначення класів колекторів.

- *AggF ilter*

Інтерфейс, який використовується для фільтрації виявляючих пристроїв і що входять з ними даних. За замовчуванням будь-який конфигуратор буде повертати реалізацію *AggF ilter*, яка пропускає всі пристрої і дані.

2.3 Логістика і обробка подій

Розглянутий підкомпонент відповідальний за логістику повідомлень агрегаторского компонента, а також за обробку подій, які генеруються об'єктами класу *Aggregator* (Далі об'єкти-агрегатори). Розроблена архітектура підкомпонента схематично представлена на малюнку 3.



Мал. 3: Архітектура підкомпонента, що відповідає за логістику і опрацювання подій

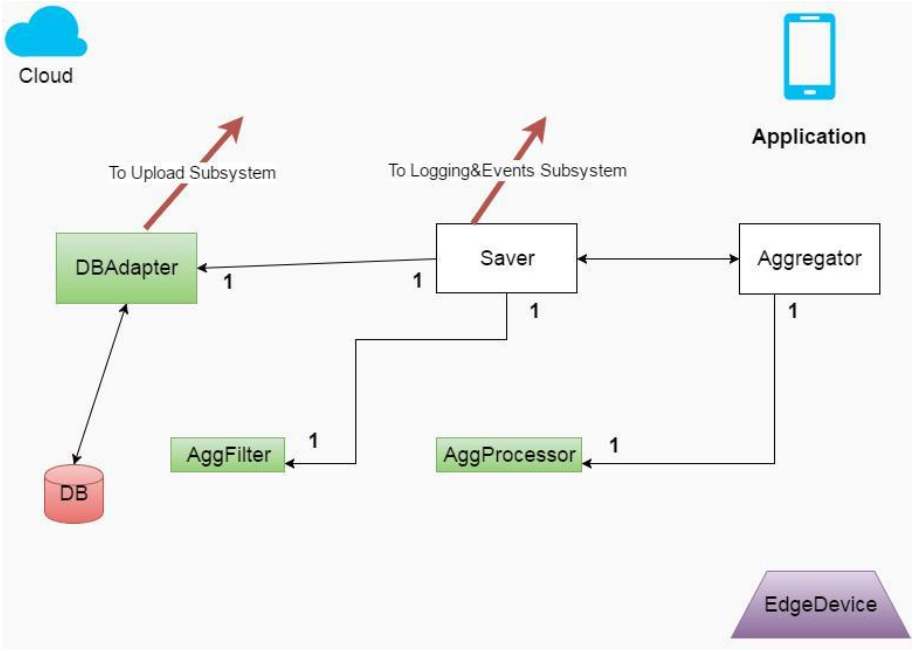
Для кожного об'єкта-агрегатора конфігуратором створюється об'єкт класу *AggListener* (Далі слухач). За замовчуванням будь-який конфі-гуратор буде повертати об'єкт класу *DefaultLoggingAggListener*, Який буде писати в логи компонента хроніку подій. Слухач може реагувати на різні події агрегації, які від самого початку підтримуються компонентом: початок збору з датчика, кінець збору з датчика, агрегатор запущений, агрегатор зупинений, виникла помилка збору, виявлений новий датчик. Реакція на кожну таку подыю може бути модифікована розробником. При побудові архітектури даного компонента за основу був узятий шаблон проектування Observer [8]. Розробник також може додавати свої подію, наслідуючи від класу *AggEvent*, і задавати реакцію своїх слухачів на них.

Також важливим елементом архітектури є інтерфейс *Logger*

(Об'єкт класу, що реалізує цей інтерфейс, далі буде називатися логером). Будь-який Логер надає функціонал запису прийнятих повідомлень в файли і їх читання. За замовчуванням будь-який конфігуратор повертає Логер, який працює з системними потоками введення і виведення.

Локальне збереження даних

Розглянутий підкомпонент відповідальний за довгострокове зберігання даних прийнятих з датчиків підкомпонентою збору. Розроблена архітектура підкомпонента схематично представлена на малюнку 4.



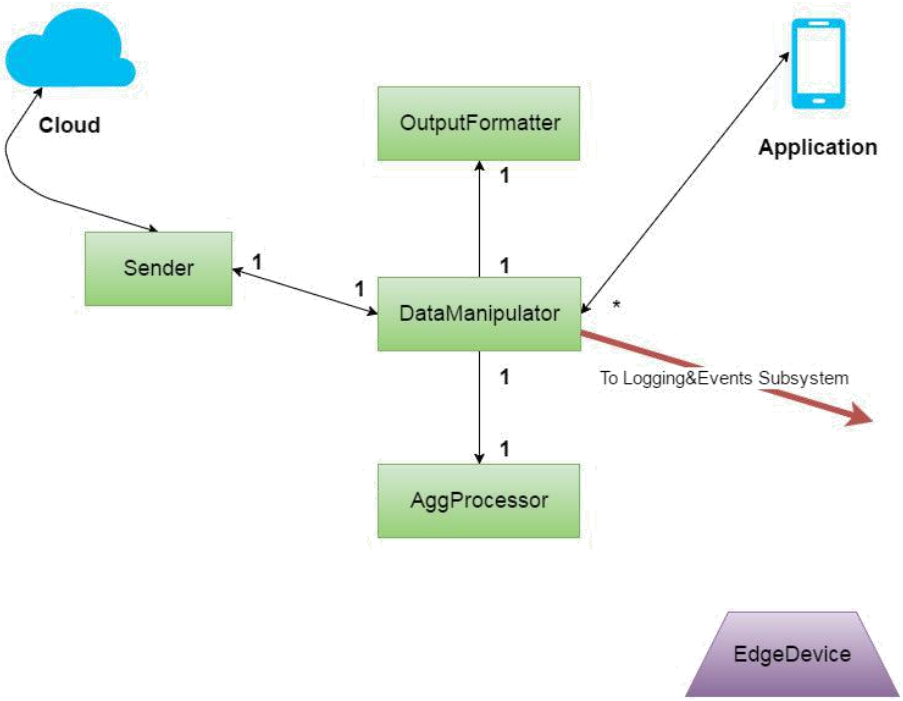
Мал. 4: Архітектура підкомпонента, що відповідає за локальне збереження даних.

Ключовими класами цього подкомпоненте є *DBAdapter* , *AggF ilter*, *AggP rocessor*.

- *AggP rocessor*
  Інтерфейс, який використовується для обробки накопичених даних перед їх збереженням. Надає метод, який приймає колекцію даних і повертає її модифікацію за замовчуванням будь-який конфигуратор буде повертати реалізацію *AggProcessor,* яка не змінює вхідні колекцію.
- *DBAdapter*
  Інтерфейс, що надає методи для збереження і вилучення даних з довгострокового і надійного сховища. Без втручання розробника за умовчанням будь-який конфігуратор повертає об'єкт класу *Def aultRAM Adapter*, Який буде зберігати дані в оперативній пам'яті пристрою замість більш надійного місця і надавати до них зручний доступ. Об'єкт класу, що реалізовує інтерфейс *DBAdapter* (Далі адаптер) використовується об'єктом класу *Saver* (Далі збережувач). Ця залежність класів пояснюється використанням шаблону проектування Strategy [8]. Для кожного об'єкта-агрегатора розробник може задавати свою поведінку збережувача: розклад збережувача, обсяг даних, при накопиченні якого слід зробити збереження, і так далі.
- *AggFilter*
  Той же інтерфейс, який був описаний в 2.2 Даний елемент корисний для фільтрації колекції вже модифікованих з допомогою об'єкта класу *AggP rocessor* даних.

2.4. Вивантаження даних

Призначення даного підкомпонента - різні маніпуляції з локально збереженими даними, які в першу чергу включають в себе вилучення даних з довгострокового сховища, відправку на віддалений сервер. Розроблена архітектура підкомпонента схематично представлена на малюнку 5.



Мал. 5: Архітектура підкомпонента, що відповідає за вивантаження даних

Ключовими класами цього підкомпоненте є *DataM anipulator*, *OutputF ormatter*, *Sender*, *AggP rocessor*.

- *AggP rocessor*

Той же інтерфейс, який був описаний в розділі “Локальне збереження даних”. Даний елемент може бути корисний для модифікації давно збереженої інформації перед відправкою даних на віддалений сервер.

- *OutputF ormatter*

Абстрактний клас, який надає методи для форматування підготовлених для вивантаження на віддалений сервер даних. Об'єкти класу, що реалізовує інтерфейс *OutputFormatter* далі будемо називати видоперетворювач. Кінцевий формат даних видоперетворювача можна підмінити до і після побудови останнього. Будь-який конфігуратор за замовчуванням при запиті на побудову видоперетворювача будуватиме об'єкт класу *Def aultJSON OutputF ormatter*, який буде виробляти сериалізації даних в формат JSON за допомогою програмної бібліотеки jackson [7].

- *Sender*

Інтерфейс, що надає функціонал для роботи з конкретним віддаленим сервером. Об'єкт класу, що реалізує інтер-фейс *Sender* (Далі відправник) виробляє аутентифікацію на віддаленому сервері (використовуючи налаштування надавача), організовує відправку форматованих видоперетворювачем даних на сервер і її переривання. Будь-який конфігуратор по замовчуванню при запиті на побудову відправника будуватиме об'єкт класу *Def aultDropboxSender*, який буде проходити аутентіфікацію на сервері Dropbox [9] за допомогою налаштувань представника і завантажувати дані в хмару з допомогою його API.

- *DataM anipulator*

Клас, який надає інтерфейс для вилучення довготерміново збережених даних і організацію їх відправки на відвіддалений сервер з подальшим очищенням пам'яті при необхідності. Об'єкт класу *DataManipulator*, за замовчуванням що повертає будь-яким конфігуратором, далі буде називатися маніпулятором. Будь-який маніпулятор має в розпорядженні деякий адаптер до довгострокового сховища і відправника, що дозволяє йому надавати наступний функціонал: витяг зі сховища даних, що задовольняють будь-яким умовам, контроль транзакційності відправки даних, видалення відправлених даних. Відправка даних маніпулятором була спроектована з використанням шаблону проектування Strategy.

2.5. Вибір програмної платформи для реаліції агрегаторского компонента фреймворка

Основними характеристиками програмної платформи при її ви-борі для розробки агрегаторского компонента були:

1. Кросплатформеність, можливість створення програмного забезпечення під мобільні пристрої.
2. Наявність можливості побудови додатків з гнучкою архітектурою.
3. Поширеність серед розробників і простота розгортання.

Перша характеристика платформи важлива для створення мультиплатформенної системи. Нестаціонарний агрегатор може представляти із

себе додаток, запущений на смартфоні - тому від платформи потрібна сумісність з мобільними пристроями. Друга ха-рактеристика відповідає за модульність цільового фреймворка, яка потрібна для надання можливості швидкого побудови цільових систем збору даних. Третя характеристика показує, наскільки легко буде потенційному розробнику розібратися в архітектурі І почати безпосереднє використання підсумкового фреймворка для простої побудови своєї системи збору. В якості платформи, чиї перераховані вище характеристики були прийнятними для побудови агрегаторского компонента, була обрана Java платформа [10].

По-перше, додатки під цю платформу пишуться на мові про-грамірованія Java, який відомий як одні з кращих мов для написання високопродуктивних кроссплатформенних додатків. Java програма зможе запуститися на будь-якому пристрої, на якому працює віртуальна машина Java, тобто пропадає необ-ність потенційно трудомісткою перекомпіляції написаного додатка під різні кінцеві платформи.

По-друге, мова програмування Java використовує парадигму об'єктно-орієнтованого програмування, що дозволяє будувати додаток з гнучкою архітектурою при правильному проектуванні (наприклад, при використанні шаблонів проектування [8]).

По-третє, мова Java займає перше місце в рейтингу TIOBE [11] і має рейтинг рівний 20.8% на квітень 2019 року, що ілюструє популярність Java платформи серед розробників. До додаткових переваг мови програмування Java можна віднести і те, що вона є поширеним і багатим по наданих можливостям інструментом для створення додатків під мобільну платформу Android. Операційна система Android на даний момент не перестає бути домінантною серед платформ сучасних смартфонів [12]. Даний факт призводить до того, що вибір Java платформи сприяє створенню фреймворка, який дозволяв би організовувати системи збору даних, де в якості агрегаторів можуть виступати більшість існуючих мобільних Android пристроїв.

Таким чином, представлена в 2.1 архітектура була створена на вищеописаній платформі. Апробування отриманого компонента описано нижче.

Створення прототипу

З метою апробації створеної агрегаторской частини фреймворка було вирішено реалізувати тестове додаток на основі розробленого компонента під платформу Android з огляду на її популярності (див. 2.5).

Тестове додаток збирача було реалізовано під Android платформу версії 4.0 і вище. Запуск програми проводився на пристроях, що мають різні версії встановленої Android OS і характерні потужності. Цими пристроями були: смартфон Explay N1 (Android 4.2.2), смартфон Asus Laser 2 Zenfone ZE 500KL (Android 5.0.2). Додаток було протестовано спільно з додатком, побудованим на основі компонента фреймворка для датчиків.

Додаток збирача надавали мінімальний користувацький інтерфейс для налаштування системи під час її роботи. Додаток проводив автоматичний збір даних з датчиків на Raspberry Pi по Bluetooth в форматі JSON. Для користувача було доступно завдання розкладу збереження даних в локальну базу даних SQLite. Користувач мав можливість переглядати логи, які генеруються системой. Система дозволяла користувачу ініціювати вивантаження збережених даних в хмару Dropbox в форматі JSON.

3 СПЕЦІАЛЬНА ЧАСТИНА

3.1 Можливі підходи до збору даних

Введемо декілька понять. Оскільки деякі підходи передбачають наявність двох видів пристроїв в системі збору даних, виділим пристрої - шлюзи (агрегатори) і пристрої-датчики. Датчиком будемо називати пристрій з підключеними до нього сенсорами. Таким пристроєм може бути як якась апаратна платформа, що дозволяє підключати до неї сенсори, так і повноцінний комп'ютер. Найчастіше пристрої - датчики мають дуже обмежені обчислюальні ресурси. Шлюзом (агрегатором) зазвичай є більш потужний пристрій, який має безпосередній доступ до мережі Інтернет. В даному випадку шлюз відіграє роль відправника даних на сервер.

Можливі підходи до збору даних:

- «Датчик-шлюз»
- «Кілька потужних шлюзів-збирачів »
- «Mesh-мережа»
- «Нестаціонарний агрегатор»

«Датчик-шлюз»

Кожен пристрій забезпечується власним виходом в мережу для відправки даних з сенсорів. Для цього встановлюється якийсь модем або мережева карта на кожен пристрій - датчик. Виходить, що датчик в

одночас є шлюзом. Таким чином, видно, що даний підхід вимагає обчислювальної потужності від датчика, що веде до збільшення його розмірів, вартості, енергоспоживання і тепловиділення.

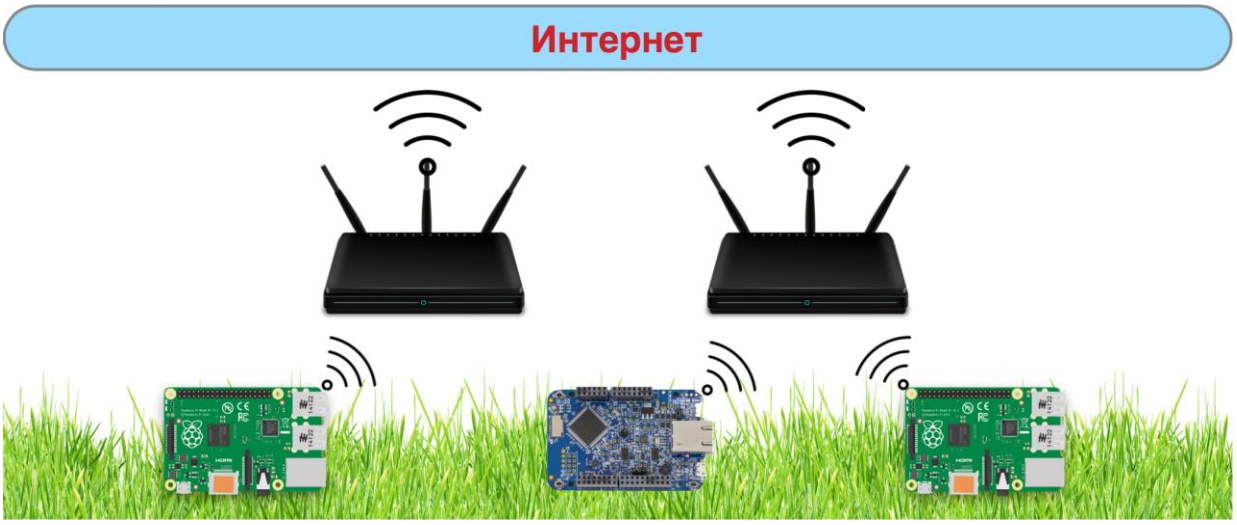
Мал. 6: Підхід «Датчик-шлюз»

Варто зазначити, що цей варіант більше підходить для збору даних в режимі реального часу і є досить надійним. Ще один нюанс полягає в тому, що не скрізь є можливість отримати доступ в Інтернет (наприклад, в лісі).

«Кілька потужних шлюзів-збирачів »

Цей варіант найбільш поширений і простий. За території розставляються кілька шлюзів так, щоб кожен датчик мав можливість підключення до одного з шлюзів. Шлюзи забезпечуються виходом в Інтернет (це може бути як 3G модем, супутниковий зв'язок, так і дротове підключення). Таким чином, шлюзи зв'язуються з датчиками, збирають з них дані і відправляють на сервер.

Такий підхід цілком простий і надійний, але також має свої недоліки. По-перше, кожен шлюз вимагає набагато більш потужного джерелаживлення, ніж датчик. Тому можуть виникнути проблеми з живленням шлюзів, якщо поруч немає стаціонарного джерела. По-друге, вартість шлюзів може сильно перевищувати вартість датчиків. Тому у випадках з великою кількістю шлюзів загальна вартість системи може виявитися велика.



Мал. 7: Підхід «Кілька потужних шлюзів-збирачів»

«Mesh-мережа»

Набагато більш цікавий і незвичайний підхід. Припустимо, що є всього один або кілька датчиків, що мають доступ в мережу (в даний момент не має значення яким чином: самостійно або за допомогою шлюзу), а інші датчики можуть передавати інформацію тільки між собою. Тоді якщо кожен датчик має зв'язок хоча б з одним сусіднім, то будь-який датчик може передати свої дані на сервер за допомогою своїх сусідів.

Мал. 8: Підхід «Mesh-мережа»

З переваг слід відзначити гнучкість і масштабованість такої мережі. З одного

боку, вартість такої системи не повинна бути високою, так як в принципі може вистачити навіть одного шлюзу або одного датчика з модемом, а решта датчики можуть залишатися дешевими.Але тут є свій нюанс: в такій мережі кожен датчик повинен виводити маршрутизацію, що накладає свої вимоги на обчислювальну потужність, а як наслідок і на енергоспоживання та вартість. Тому цей підхід може виявитися не таким вигідним, як з першого погляду. До того ж, як показують дослідження, така архітектура має низьку надійність зважаючи на деякі фізичних причин.

«Нестаціонарний агрегатор »

Також нестандартний підхід [13] до збору даних. Він передбачає, що збір ведеться з певною періодичністю або за розкладом і не придатний для збору в реальному часі. Кожен датчик оснащується модулем бездротової передачі даних (наприклад, Bluetooth Low Energy), а в ролі агрегатора виступає деякий мобільне пристрій (наприклад, смартфон), також оснащений передавачем. Датчики мають власное сховище, що дозволяє зберігати одержані з сенсорів дані до появи агрегатора в радіусі видимості.

Мал. 9: Підхід «Нестаціонарний агрегатор»

Відповідно до розкладу датчики активують свої модулі передачі даних, надаючи можливість мобільному агрегатору виявляти їх. Як тільки агрегатор виявляє датчик, між ними встановлюється з'єднання і починається передача всіх накоплених датчиком даних. Далі мобільний агрегатор має можливість відправити зібрані дані на сервер для подальшої обробки.

Даний підхід відрізняється від попередніх тим, що на території, де розташовуються датчики, не потрібно мати доступ до мережі Інтернет. Це значно підвищує автономність системи і робить економічно менш витратним її побудову. Також варто наголосити на тому, що датчики працюють незалежно один від одного і їх місце розташування ви- виділяється тільки фізичної доступністю з точки зору агрегатора. Це робить систему легко масштабується і гнучкою.

висновок: оскільки підхід, описаний в пункті “Нестаціонарний агрегатор”, при визначених умовах має ряд переваг перед іншими підходами, має сенс розглянути існуючі рішення для побудови таких систем. У даній роботі розгляд буде вестися з точки зору пристроїв-датчиків.

3.2 Рішення для побудови систем збору даних

На сьогоднішній день підхід «Нестаціонарний агрегатор не є широко поширеним. У зв'язку з цим було знайдено всього один фреймворк [3], що дозволяє будувати системи подібного роду. Тому крім цього фреймворка розглянемо деякі рішення для збору даних, які спочатку не орієнтовані, але можливо можуть бути використані для збору нестаціонарним агрегатором. Таких рішень [14] величезна кількість, тому було вибрано декілька з них для прикладу.

З точки зору пристроїв-датчиків, серед ключових властивостей решеннння для побудови розглянутих систем виділимо наступні:

1. Відсутність вимоги підключення пристроїв-датчиків до Ін- тернету
2. Незалежність від протоколу передачі і формату даних
3. Мультиплатформеність

4. Низькі обмеження на завдання користувальницької логіки отриман- ня даних і їх перед обробка
5. Надання додаткової функціональності для самостійності роботи пристроїв-датчиків (зберігання зібраних даних до появи агрегатора в зоні видимості, включення / вимикання модулів передачі даних за розкладом і т.п.)
6. Відкритість програмного коду

Open Data Kit Sensors

- Не вимагає підключення до Інтернет
- Не фіксує протокол і формат передачі даних
- Взаємодія з низькорівневими сенсорами (I2C, SPI, ..) тільки через Arduino і тільки по USB
- Агрегаторская частина фреймворка доступна тільки для Android
- Обмежує свободу завдання логіки взаємодії з датчика ми (отримання даних з сенсорів тільки з постійною частотою)
- Не надає сховище даних на стороні датчиків
- Відкритий програмний код [3]

Carriots

- Розробники вказують: «підключайте будь-який пристрій до Carriots [9]; все,
- що вам потрібно - доступ до Інтернет »
- Використовує REST API
- Не має додаткової підтримки для самостійної роботи пристроїв-датчиків
- Власницьке ПЗ

ThingSpeak

- Пропонує сервер на Ruby On Rails в якості агрегаторской частини, що може позбавити від необхідності підключення датчиків до Інтернет
- Використовує REST API
- Не має додаткової підтримки для самостійної роботи пристроїв-датчиків
- Відкритий програмний код [4]

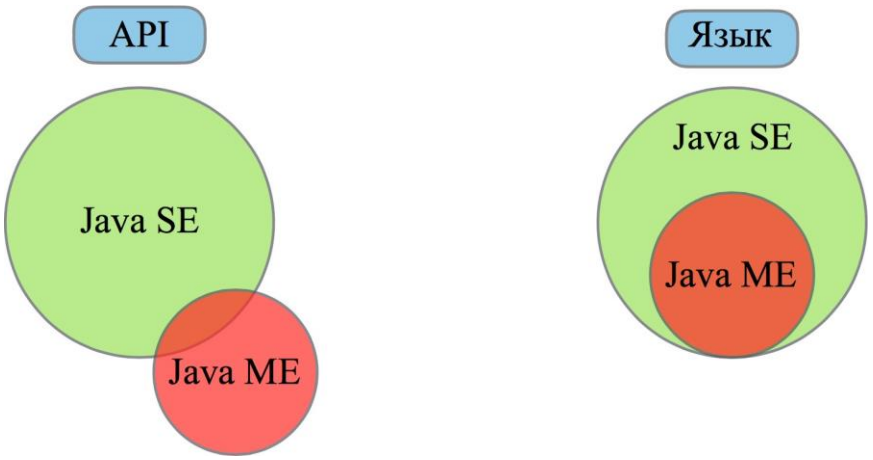
висновок: Існуючі інструменти не орієнтовані на збір даних за допомогою нестаціонарного агрегатора, що означає необхідність адаптації цих інструментів для такого підходу. У зв'язку з цим було б зручно мати якийсь фреймворк, який володів би перечисельними властивостями і дозволяв адаптувати існуючі решення платформи Інтернету Речей для збору даних нестаціонарним агрегатором.

3.3 Програмна платформа для фреймворка

В якості основи для фреймворка була обрана Java Micro Edition Embedded 8. [10]. Це безкоштовна програмна платформа, предложен- ная компанією Oracle. Характеристики:

- Заснована на всім відомої Java ME, до якої була додана підтримка специфічних функцій для вбудованих систем
- Оптимізована для запуску на пристроях з обмеженими ре-ресурсами (вимагає 128кб RAM) і енергоспоживанням
- Надає API для доступу до портів GPIO, I2C, SPI, UART
- Підтримує механізми безпечної передачі даних (TLS 1.2)
- Java ME SDK надає емулятори для Windows і Linux
- Підтримує такі платформи, як ARM Cortex-M3 / -M4, ARM 9 / BREW MP, ARM 11 / Linux OS
- Можливий запуск на Cortex-M4 під управлінням ОС mbed [15]
- Для прототипирования є підтримка платформ Raspberry Pi і Freescale FRDM-K64F

Також важливим моментом є те, що мова Java ME Embedded сумусний з мовою Java SE / EE, що дозволить, при дотриманні деяких правил під час розробки фреймворка, використовувати цей фреймворк як на платформі Java ME, так і на Java SE / EE, що забезпечить мультиплатформеність.



Мал. 10: Сумісність Java ME Embedded і Java SE

3.4 Проектування фреймворка

Функціональність і вимоги

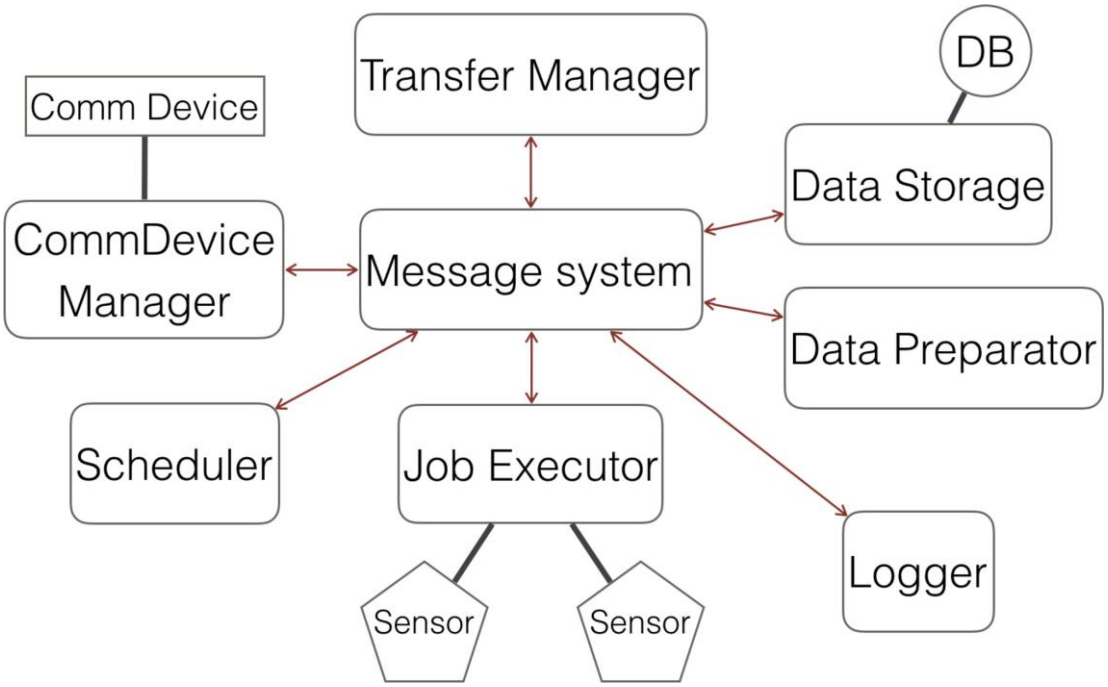
Відповідно до опису, запропонованим в пункті “Нестаціонарний агрегатор” , підходи до збору даних, виділимо основну функціональність, яку надаватиме фреймворк на пристрої-датчику:

- Отримання інформації з сенсорів
- Передобробка цієї інформації (в разі, якщо це дозволяють обчислювальні здатності пристрою)
- збереження інформації
- Управління модулем передачі даних
- виявлення агрегатора
- Витяг даних зі сховища
- Відправка даних агрегатору
- Видалення даних, які були успішно відправлені
- планування перерахованих вище дій

При цьому необхідна максимальна гнучкість завдання користувацької логіки. Наприклад, планування дій може бути як за розкладом (по таймеру), так і в залежності від інших подій; отримання інформації з сенсорів може проводитися як шляхом періодічного опитування сенсора, так і шляхом підписки на події сенсора (деякі сенсори мають тільки 2 стани: 0 і 1; в такому випадку немає сенсу постійно опитувати сенсор, достатньо лише «слухати» його події). У зв'язку з цим логіка компонентів фреймворка повинна бути легко передбачена, а сама архітектура не повинна накладати обмеження на можливі сценарії отримання інформації з сенсорів.

Компоненти

На основі пред'явлених вимог до функціональності і гнучкості, була вироблена схема, що позначає основні компоненти фреймворка. Об'єднані між собою, ці компоненти будуть складати деякий монолітний додаток.



Мал. 11: Компоненти фреймворка (прямокутники з закругленими краями)

- *Job Executor*. Для найбільш гнучкого і зручного одержання даних з датчиків була запропонована наступна ідея: нехай є набір деяких «робіт» (Job), кожна з яких інкапсулює в собі логіку роботи з деяким безліччю сенсорів; тоді незалежні один від одного датчики можна помістити в різні «Роботи», а датчики, які працюють спільно, в одну «роботу». Кожна «робота» виробляє одиниці даних (JobData) і відправляє в систему для подальшої обробки. Job Executor - компонент, який відповідає за виконання цих самих Job. Він бере на себе управління потоками, запуск і зупинку «робіт».

- *Data Preparator*. Займається передобробкою даних. Обробляти дані, опубліковані Job'ами і відправляє готові дані на збереження.
- *Data Storage*. Відповідає за збереження даних у БД / файлі / іншій структурі і за вилучення даних звідти за запитом.
- *Transfer Manager*. Займається відправленням даних агрегатору. Отримаючи сповіщення про виявлення нового агрегатора, він створює сесію передачі даних (Transfer Session) запит на отримання даних зі сховища (Data Storage) і передає ці дані в сесію. Transfer Session реалізує протокол передачі даних (більш високого рівня, ніж фізичний).
- *CommDevice Manager*. Управляє модулями передачі даних. Він запускає / зупиняє модулі передачі даних і обробляє потоки, в яких виконуються їх драйвери.
- *Scheduler*. Це планувальник. За допомогою нього можна задавати розклад включення / вимикання апаратних компонентів (сен- соров і модулів передачі даних) і розклад виконання пре- добработки і збереження даних.

Система повідомлень

З метою зниження зв'язності компонентів додатка було прийнято рішення використовувати шаблон проектування Видавець / Підписчик. Була зроблена спроба використовувати існуючі бібліотеки, що пропонують системи повідомлень. Однак з різних причин, що випливають з вимог до фреймворку, було вирішено розробити власну просту систему, що має тільки необхідний для даного фреймворка функціонал і не вимагає додаткової підтримки з боку ОС (наприклад, підтримку сокетів). Розроблена система повідомлень володіє наступними характеристиками:

- Кожне повідомлення має тему

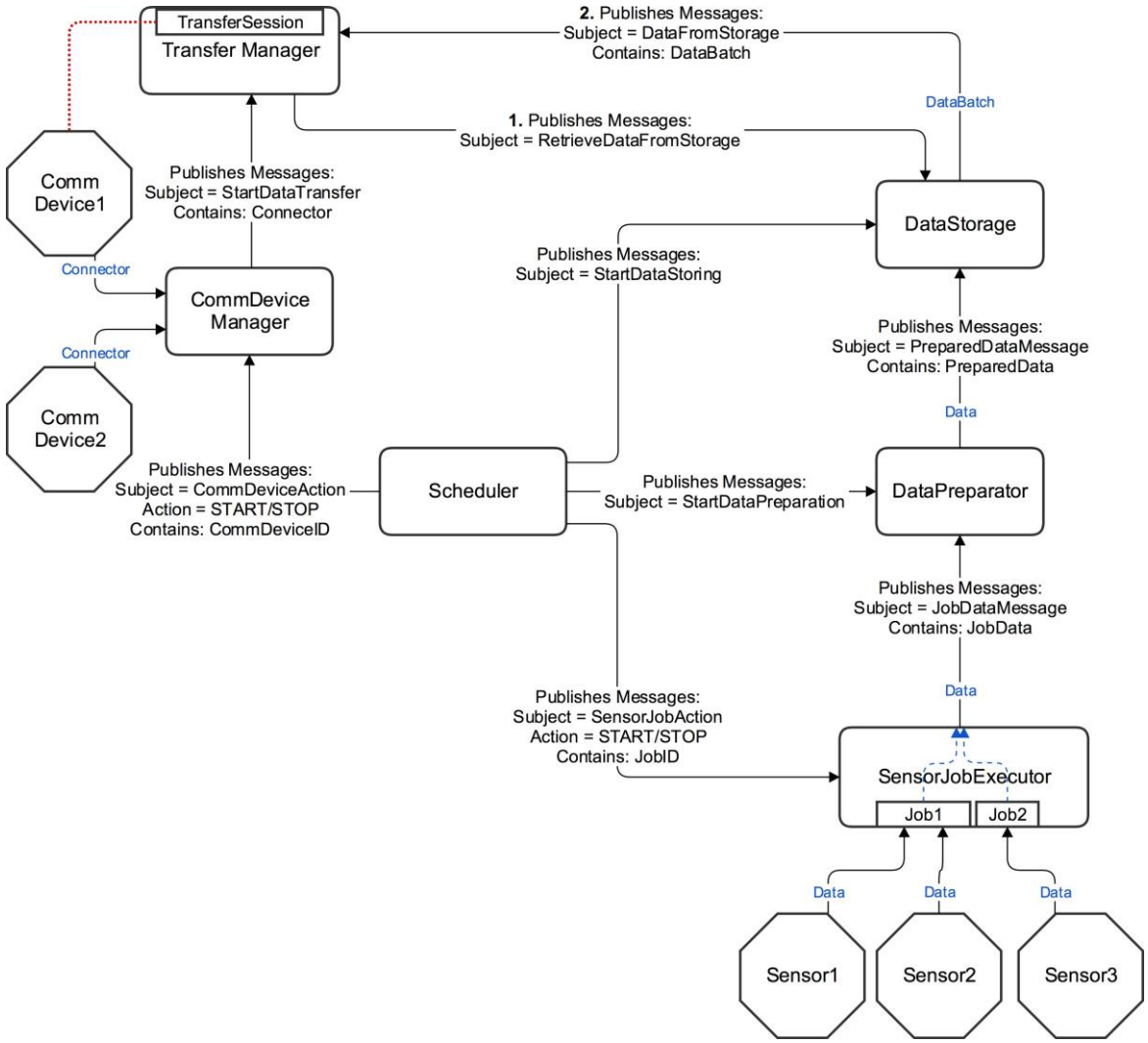
- Будь-який об'єкт може підписатися як на тему, так і на публікація цієї конкретного об'єкта
- Існує ієрархія тем
- Передплатник теми отримує також всі повідомлення по успадкованим темам

Таке архітектурне рішення дозволяє легко додавати нові компоненти в систему. Також дозволяє автоматично направляти в лог всі повідомлення без виклику методу log, так як логер підписаний на тему Any, від якої успадковуються всі інші теми. Всі компоненти системи підписані на тему Shutdown. Повідомлення з такою темою публікується при завершенні програми. Отримуючи таке повідомлення, все компоненти готуються до завершення роботи

Реалізація

Відповідно до рішень, прийнятих під час проектування (разділ 3.4), був реалізований фреймворк на обраної в розділі 3 програмної платформи. Наступна схема відображає процес роботи додатка, побудованого за допомогою цього фреймворка.

Блакитним кольором виділені типи даних, що передаються між ком-компонентами. Стрілочками позначені публікації повідомлень (від видавця до передплатника).



Мал. 12: Схема роботи фреймворка

1. Планувальник (Scheduler) дає команди на запуск "робіт" (Job), вказуючи їх ідентифікатори.
2. Сенсори публікують отриману із зовнішнього світу інформацію. Тип даних успадковується від Data. Кожен сенсор може використати свій тип даних.
3. Кожна Job підписана на публікації своїх сенсорів. Агрегируя інформацію з усіх своїх сенсорів, Job публікує дані, тип яких також успадковується від Data, але може відрізнятися від типу, який виробляють сенсори.
4. Дані, отримані від всіх "робіт", збираються в чергу всередині Data Preparator.

5. Відповідно до розкладу планувальник дає команду "почати підготовку даних".

6. Data Preparator обробляє дані, що накопичилися у нього у вхідній черзі (алгоритм обробки заданий користувачем). Після цього він публікує ці дані. Як і раніше, вихідний тип даних може відрізнятися від вхідного.

7. Data Storage збирає дані від Data Preparator до себе до вхідної черги.

8. Відповідно до розкладу планувальник дає команду "почати збереження даних".

9. Data Storage зберігає дані, накопичені у нього від вхідної черги в довготривалу пам'ять (БД / файл / інша структура даних).

10. Відповідно до розкладу планувальник дає команду "включити модуль передачі даних", вказуючи ідентифікатор модуля.

11. При виявленні агрегатора модуль передачі даних публікує якийсь Connector, який зіставлений підключенню до агрегато- ра. Через цей коннектор можна здійснювати відправку і отримання байтів.

12. Transfer Manager отримує Connector і створює сесію Transfer Session. У цю сесію він передає коннектор. Також він публікує запит на вилучення даних зі сховища.

13. Data Storage отримує запит на вилучення даних і публікує дані "пачками" (DataBatch). Розмір DataBatch задається користувачем.

14. Дані від Data Storage потрапляють до вхідної черги Transfer Manager для подальшої відправки

15. Сесія стартує, і починається передача даних, що знаходяться у вхідній черзі Transfer Manager. Червона лінія показує, що сесія для відправки даних використовує коннектор, який в свою чергу працює через модуль передачі даних (CommDevice).

16. По завершенні передачі в системі публікується повідомлення про від ключении агрегатора.

Апробація

Як апробації фреймворка було вирішено побудувати прототип тестової системи збору даних. Датчик повинен буде: отримувати з певною частотою інформацію з сенсора і зберігати її в БД Java RMS, при підключенні агрегатора витягувати накопичені дані з БД і відправляти агрегатору. Дані передаються в форматі JSON.

Реалізація прототипу на основі разработаного фреймворка

Система збору даних в нашому випадку повинна складатися як мінімум з двох апаратних пристроїв: агрегатора і датчика. Далі буде розглянуто побудову датчика. Для побудови датчика потрібно виконати наступні завдання:

Апаратна частина

1. Вибрати апаратну платформу для датчика
2. Вибрати хоча б один сенсор, сумісний з апаратної платформою
3. Вибрати модуль передачі даних, сумісний з апаратної платформою (якщо він не вбудований в апаратну платформу)
4. Вибрати додаткові електронні компоненти, необхідні для підключення сенсора і модуля передачі даних
5. Зібрати схему з перерахованих раніше електронних компонентів

Програмна частина

1. Написати драйвер модуля передачі даних
2. Написати драйвер (и) для сенсора (ів) та описати логіку отримання даних
3. Описати тип (и) і формат (и) даних і методи серіалізації цих даних

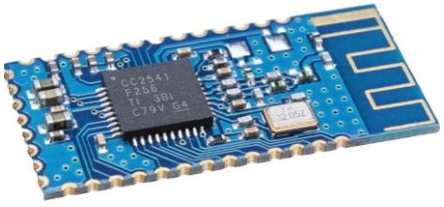
Апаратна частина

В якості апаратної платформи була обрана Raspberry Pi 2 Model B. Ця платформа підтримується Java ME Embedded. Також вона надає порти GPIO, I2C, SPI, UART для підключення цифрових периферійних пристроїв, в тому числі сенсорів і модулів передачі даних.

Також був обраний сенсор Ultrasonic HC-SR04 [16], Що підключається через GPIO. Даний сенсор призначений для вимірювання відстані до об'єктів за допомогою ультразвукових імпульсів. Потрібно джерело живлення 5В. Працює сенсор наступним чином: на контакт Trig подається логічна одиниця (напруга 2.5-5В) тривалістю 10мкс, сенсор генерує звуковий імпульс на частоті 40кГц і слухає відлуння. Отримуючи відлуння, видається 5В на контакт Echo. За часом між подачею Trig і отриманням Echo визначається відстань до об'єкта.

Мал. 13: Ultrasonic HC-SR04

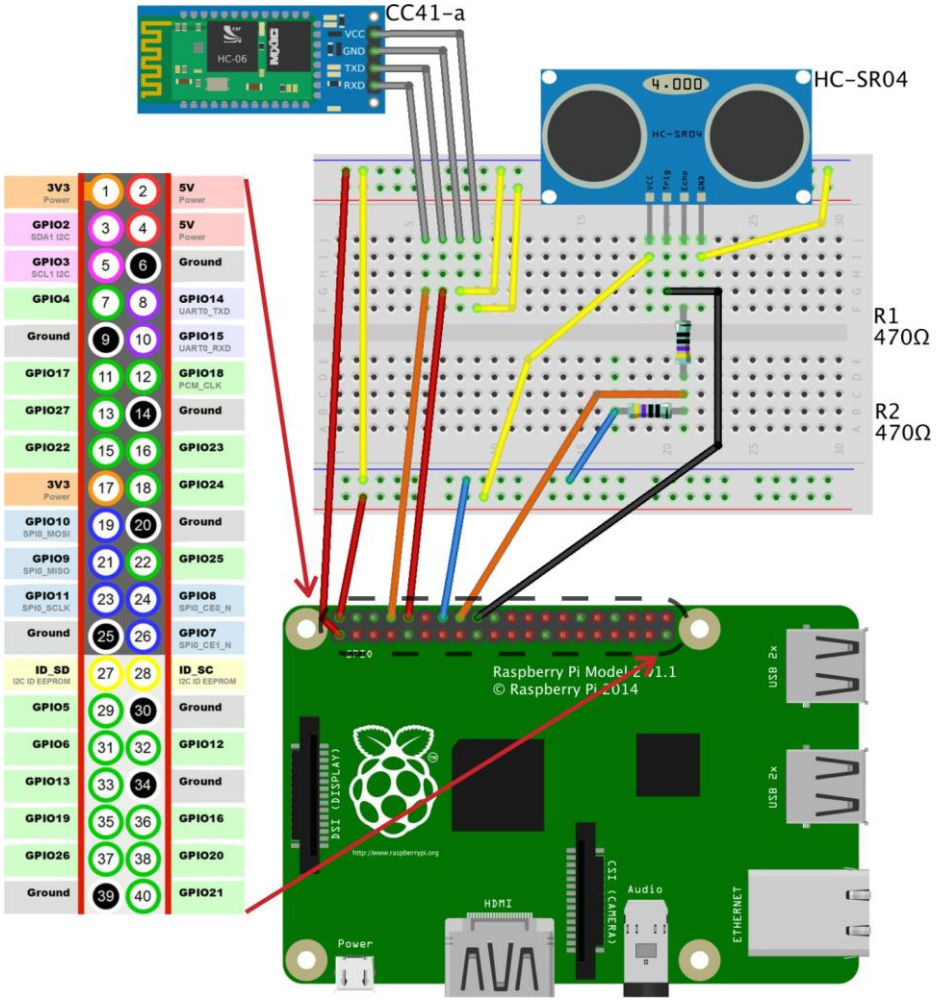
Передачу даних було вирішено здійснювати по протоколу Bluetooth Low Energy, з огляду на його поширеності та підтримки сучасними смартфонами. Для прототипу використовувався модуль Bolutek CC41-a [8], Що підключається за допомогою UART. Управління модулем здійснюється за допомогою AT-команд [2]. Розрахований на джерело живлення 3.3В.



Мал. 14: Bolutek CC41-a

При підключенні сенсора до платформи виникла проблема: сенсор видає на контакт Echo 5В, а GPIO контакти Raspberry Pi розраховані на напругу до 3.3В, в зв'язку з чим при прямому підключенні можливе пошкодження Raspberry Pi. Тому було вирішено зробити резисторний дільник напруги для зниження напруги на контакті Echo. Для цього були обрані 2 резистора по 470 Ом. Таким чином, використовуючи два однакових резистора в дільнику, отримуємо напругу 5В / 2 = 2.5В.

На основі обраних компонентів була зібрана і протестована наступна схема. Схема була зібрана на безпаєчній макетній платі.



Мал. 15: Схема прототипу

Програмна частина

При написанні драйвера модуля передачі даних був використаний API JavaME Embedded для роботи з портом UART. Для управління модулем було необхідно відправляти йому команди виду «AT + <слово>». При старті драйвер посилає команду «AT + RESET» для перезагрузки модуля. Коли зовнішній пристрій підключається до модуля, драйвер отримує повідомлення «OK + CONN», що означає, що можна почати передачу. Повідомлення «OK + LOST» означає, що зв'язок розірвано. Драйвер реалізує інтерфейс UARTEventListener, щоб отримувати повідомлення про появу нових даних в порте UART.

Драйвер сенсора користується Java MEE API для роботи з GPIO. Реалізований драйвер розрахований тільки на синхронне зчитування інформації з сенсора (надає метод read). Після реалізації самого драйвера, необхідно описати логіку отримання інформації з сенсора. Це робиться шляхом успадкування від абстрактного класу SensorJob і реалізації його абстрактних методів. Нижче наведено приклад найпростішої реалізації цього класу.

```
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();
        }
}
```

Мал. 16: Реалізація класу HCSR04Job

Також необхідно описати тип призначених для користувача даних і метод його сериалізації. В даному прототипі використовувався сенсор-далекомір, який видає число типу double. Тому нехай призначений для користувача тип даних буде містити одне поле - число типу double. Для цілей легування всі призначені для користувача типи даних повинні перевизначати метод toString. Як формат передачі даних в даному прототипі використовувався JSON, тому також додамо метод toJSON для приведення даних в цей 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;
    }

}
```

Мал. 17: Реалізація класу HCSR04Data

В даному прототипі використовувався простий метод серіалізації даних. Оскільки JSON - текстовий формат даних, то для серіалізації було досить перевести дані в формат JSON і застосувати до отриманого тексту метод getBytes () стандартного Java класу String. Десеріалізація відбувається в зворотному порядку.

Побудований прототип був протестований спільно з агрегаторської частиною, виконаною на смартфоні з ОС Android.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОНОРОНИ ПРАЦІ

4.1 Вплив виробничого середовища на працездатність та здоров’я користувачів відео дисплейними терміналами (ВДТ)

Широке впровадження комп'ютерної техніки, що дає змогу автоматизувати багато рутинних операцій, одержати доступ до численних джерел інформації, швидко проводити потрібні розрахунки, істотно підвищує продуктивність праці користувачів ВДТ ПЕОМ. Проте активне впровадження у практику персональних комп'ютерів має двоякий характер. З одного боку, підвищується результативність праці, а з другого — з'являються фактори, які несприятливо впливають на здоров'я працюючої людини. У зв'язку з цим набуває актуальності вивчення фізіологічних, психологічних, соціальних та виробничих наслідків використання ВДТ ПЕОМ, розробка та активне застосування заходів, що нормалізують працю та зберігають здоров'я користувачів.

Вагомий вплив на працездатність та здоров’я користувачів комп’ютерів здійснює виробниче середовище. Це середовище у виробничих приміщення в основному, визначається мікрокліматом, освітленням, наявністю шкідливих речовин у повітрі, рівнем шуму, випромінюванням.

Під виробничим мікрокліматом розуміють стан повітряного середовища виробничого приміщення, який визначається температурою, відносною вологістю, рухом повітря та тепловим випромінюванням нагрітих поверхонь, що в сукупності впливають на тепловий стан організму людини. В процесі трудової діяльності людина перебуває у постійній тепловій взаємодії з виробничим середовищем.

За нормальних мікрокліматичних умов в організмі працівника, завдяки терморегуляції, підтримується температура тіла (36,6˚С).

Кількість тепла, що утворюється в організмі, залежить від фізичного навантаження працівника, а рівень тепловіддачі — від мікрокліматичних умов виробничого приміщення. Оскільки робота за комп'ютером характеризується малими фізичними навантаженнями, то цей вид діяльності належить до категорії легких робіт за критерієм енерговитрат організму (ГОСТ 12.1.005-88). Для того, щоб фізіологічні процеси в організмі людини відбувалися нормально, теплова енергія, що виділяється під час роботи організмом, повинна повністю відводитись у навколишнє середовище. Порушення теплового балансу може призвести до перегрівання або ж переохолодження організму людини і, зрештою, до захворювання.

Віддача тепла організмом людини здійснюється, в основному, за рахунок випромінювання і випаровування вологи з поверхні шкіри. Чим нижча температура повітря і швидкість його руху, тим більше тепла віддається випромінюванням. При високій температурі значна частина тепла втрачається випаровуванням поту.

Вологість повітря істотно впливає на віддачу тепла випаровуванням. Через високу вологість випаровування погіршується і віддача тепла зменшується. Зниження вологості покращує пронос тепловіддачі випаровуванням. Однак, надто низька вологість викликає висихання слизових оболонок, їх пересихання та розтріскування, забруднення хвороботворними мікробами.

Рухомість повітря визначає рівень тепловіддачі з поверхні шкіри конвекцією і випаровуванням. Різкі коливання температури в приміщенні, яке продувається холодним повітрям значно порушують терморегуляцію організму і можуть викликати простудні захворювання.

Таким чином, для нормального теплового самопочуття людини важливо забезпечити певне співвідношення температури, відносної вологості

та швидкості руху повітря, тобто певні мікрокліматичні умови. Такі умови визначаються, в основному, категорією роботи, що виконується, та періодом року і можуть бути оптимальними та допустимими.

Відповідно до ДСанПіН 3.3.2-007-98 у виробничих приміщеннях та робочих місцях з ВДТ та ПК мають забезпечуватись оптимальні значення параметрів мікроклімату (див.табл. 4.1).

Таблиця 4.1 - Нормовані параметри мікроклімату для приміщень з ВДТ та ПК

Пора року	Категорія робіт	Температура повітря, °С, не більше	Відносна вологість повітря,	Швидкість руху повітря, м/с
Холодна	легка- 1а	22-24	40-60	**0,1**
	легка- 1б	21-23	40-60	**0,1**
Тепла	легка- 1а	23-25	40-60	**0,1**
	легка- 1б	22-24	40-60	0,2

До категорії 1а належать роботи, що виконуються сидячи і не потребують фізичного напруження, при яких витрати енергії складають до 139 Вт, а до категорії 1б — роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним напруженням, при яких витрати енергії становлять від 140 до 174Вт.

В холодну пору року значення відносної вологості повітря часто є нижчими за встановлені норми і становлять в середньому 30—40%. Це призводить не лише до надмірного висихання слизових оболонок очей, носа, горла, а й до нагромадження зарядів статичної електрики, що утворюються в процесі роботи комп'ютера.

Температура повітря в приміщеннях у теплий період року іноді перевищувала нормовані значення, особливо в приміщеннях, розташованих з південної сторони будівлі. Швидкість руху повітря, як правило, була в межах норми. Встановлено, що нагріті поверхні комп'ютера та лазерного принтера помітно не впливають на підвищення температури повітря на робочому місці, однак літом такий вплив може бути значно більшим.

Для забезпечення оптимальних мікрокліматичних умов в будь-який період року приміщення, в яких розташовані комп'ютеризовані робочі місця повинні бути обладнані системами опалення. Однак найкраще вирішення цього питання — це встановлення кондиціонерів, які автоматично підтримують задані параметри мікроклімату.

4.2 Забруднення повітря на робочих місцях з ВДТ

В повітрі зовнішнього природного середовища, як і в повітряному середовищі приміщень завжди є наявною певна кількість заряджених частинок, які називаються іонами. Так в 1 $см^3$ чистого зовнішнього повітря міститься близько 1000 негативних іонів і понад 1200 позитивних. Іонний склад повітря може значно змінюватись під-впливом цілої низки факторів, до яких також належить специфіка виробничої діяльності.

Встановлено, що вже через 5 хвилин роботи ВДТ концентрація легких негативних іонів знизилась приблизно у 8 разів, а через 3 години роботи — була вже на рівні, близькому до нуля. Істотно знизилась концентрація середніх та важких негативно заряджених частинок Разом з тим концентрація позитивних іонів зростала, і через 3 години роботи з ВДТ у повітрі робочої зони переважали позитивно заряджені частинки усіх розмірів. Така зміна балансу іонного складу повітря призводить до несприятливого впливу на здоров'я користувачів ВДТ.

ДНАОП 0.03-3.06-80 "Санітарно-гігієнічні норми допустимих рівнів іонізації повітря виробничих та громадських приміщень" регламентує рівні іонізації повітря приміщень при роботі за ВДТ та ПК (див.табл. 4.2).

Таблиця 4.2 - Рівні іонізації повітря приміщень при роботі за ВДТ ПК

Рівні	Число іонів в 1 см повітря	
	n+	n -
Мінімально необхідні	400	600
Оптимальні	1500-3000	3000-5000
Максимально допустимі	50000	50000

Необхідні концентрації позитивних та негативних іонів в повітрі робочих зон можна забезпечити застосуванням:

- генераторів негативних іонів;
- установок штучного зволоження.
- кондиціонерів;
- примусової вентиляції (провітрювання, системи загальнообмінної припливно-витяжної вентиляції, пристрої місцевої вентиляції);
- захисних екранів, що заземлені.

Багатьма дослідниками було відмічено, що до кінця робочого .дня в повітря робочої зони різко зростала концентрація CO_2, яка сягала від 0,12 — 0,13 до 0,19% (в атмосферному повітрі CO_2 міститься 0,03%).

Також серед речовин, у яких було виявлено перевищення ГДК у повітрі біля робочих місць з ВДТ найчастіше знаходились озон, оксиди азоту, пил.

Особливу небезпеку щодо виливу на здоров'я представляє підвищена концентрація озону — високотоксичного подразнюючого газу. Озон належить до так званих радіомиметичних речовин — хімічних сполук, що викликають в живих організмах зміни, схожі з тими, які виникають після дії

іонізуючого випромінювання. Тому озон вважається не лише подразнюючою, а й канцерогенною речовиною.

Основними джерелами озону на комп'ютеризованих місцях є ВДТ та лазерні принтери. З огляду на це, необхідно виключати ВДТ у випадках коли він не використовується, а лазерний принтер бажано розташувати подалі від робочого місця оператора.

Основним заходом щодо запобігання несприятливого впливу озону та інших шкідливих речовин на здоров'я операторів є забезпечення функціонування припливно-витяжної вентиляції. Для того, щоб шкідливі речовини не проникали із сусідніх приміщень, в приміщеннях з ВДТ необхідно створити деякий надлишковий тиск.

Відповідно до ГОСТ 12.1.005-88 вміст озону в повітрі робочої зони не повинен перевищувати 0,1 мг/м^3; вміст оксидів азоту — 5 мг/м^3; вміст пилу — 4 мг/м^3.

4.3 Виробничий шум та вібрація

Відомо, що шум несприятливо діє на слуховий аналізатор та інші органи та системи організму людини. Визначальне значення щодо такої дії має інтенсивність шуму, його частотний склад, тривалість щоденного впливу, індивідуальні особливості людини, а також специфіка виробничої діяльності. Ті види діяльності, у яких поєднується напружена розумова робота та інтенсивне використання комп'ютера (редагування тексту, верстка оригіналу, "запуск" та відлагодження програм) характеризується відчутним впливом навіть незначних рівнів шуму. Цей вплив виражається у зниженні розумової працездатності, швидкій втомлюваності, послабленні уваги, появі головного болю.

Рівні звукового тиску в октавних смугах частот, рівні звуку та еквівалентні рівні звуку на робочих місцях, обладнаних ВДТ і ПК визначені ДСанПіН 3.3.2-007-98 (див.табл. 4.3).

Таблиця 4.3 - Допустимі рівні звуку, еквівалентні рівні звуку і рівні звукового тиску в октавних смугах частот

Вид трудової діяльності	Рівні звукового тиску в дБ в октавних смугах із середньогеометричними частотами, Гц									
	31,5	63	125	250	500	1000	2000	4000	8000	Рівні звуку, еквівалентні рівні
Програмісти ЕОМ	86	71	61	54	49	45	42	40	38	50
Оператори в залах обробки інформації на ЕОМ та оператори	96	83	74	68	63	60	57	55	54	65
В приміщеннях для розташування шумних	103	91	83	77	73	70	68	66	64	75

Основними заходами та засобами боротьби з шумом є:

— зниження рівнів шуму в джерелі його утворення (застосовується, як правило, в процесі проектування);

— використання звукопоглинаючих та звукоізолюючих засобів;

— раціональне планування виробничих приміщень та робочих місць.

На комп'ютеризованих робочих місцях основними джерелами шуму є вентилятори системного блоку, накопичувані, принтери ударної дії Для зниження рівнів шуму на робочих місцях рекомендується розмістити друкувальні пристрої ударної дії (матричні, шрифтові принтери тощо) в іншому приміщенні, або огородити їх звукоізолюючими екранами.

Зовнішні шуми (вулиця, суміжні приміщення) також негативно впливати на функціональний стан операторів ВДТ, тому стіни приміщень, в яких розташовані комп'ютеризовані робочі місця бажано облицювати звукопоглинаючими матеріалами, які мають максимальний коефіцієнт звукопоглинання в межах частот 31.5 – 800 Гн і дозволені для оздоблення приміщень органами державного санітарного-епідеміологічного нагляду.

Під час виконання робіт з ВДТ значення характеристик вібрації не повинні перевищувати допустимих значень, визначених СН 3044-84 та ГОСТ 12.1.012-90 (табл. 4.4).

Таблиця 4.4 - Санітарні норми вібрації категорії 3 технологічної типу "В"

Середньо-геометричні частоти смуг Гц	Допустимі значення по осях X_0, Y_0, Z_0							
	віброприскорення				віброшвидкості			
	м /с^2		дБ		м/с10^{-2}		дБ	
	1/3 окт	1/1окт	1/3окт	1/1окт	1/3 окт	1/1окт	1/3 окт	1/1 окт
1.6	0.0125	0.02	32	36	0.13	0.18	88	91
2.0	0.0112		31		0.089		85	
2.5	0.01		30		0.063		82	
3.15	0.009	0.014	29	33	0.0445	0.063	79	82
4.0	0.008		28		0.032		76	
5.0	0.008		28		0.025		74	
6.3	0.008	0.014	28	33	0.02	0.032	72	76
8.0	0.008		28		0.016		70	
10.0	0.01		30		0.016		70	
12.5	0.0125	0.028	32	39	0.016	0.028	70	75
16.0	0.016		34		0.016		70	
20.0	0.0196		36		0.016		70	

25.0 31.5 40.0	0.025 0.0315 0.04	0.056	38 40 42	45	0.016 0.016 0.016	0.028	70 70 70	75
50.0 63.0 80.0	0.05 0.063 0.08	0.112	44 46 48	51	0.016 0.016 0.016	0.028	70 70 70	75
Кориговані і еквівалентні значення та їх рівні	0.014		33		0.028		75	

Для зниження вібрації обладнання, пристрої, пристосування необхідно встановлювати на спеціальні амортизуючі прокладки, передбачені нормативними документами.

ВИСНОВКИ

У роботі було розроблено автоматизованої системи організації збору даних в промисловій мережі здавачів.

В рамках даної роботи були досягнуті наступні результати:

- Розглянуто можливі підходи до збору даних з датчиків
- Проведено огляд інструментів для побудови систем збору даних нестаціонарних агрегатором та виявлено недоліки цих інструментів
- Сформовано вимоги до функціональності та архітектури розробляємого фреймворка
- Опрацьована архітектура фреймворка
- Обрана програмна платформа в якості основи для фреймворку
- Реалізовано фреймворк на обраній платформі, що задовольняє сформованим вимогам
- Побудований апаратний прототип, який використовує розроблений фреймворк
- проведено тести

Розроблений фреймворк виправляє виявлені недоліки (з точки зору збору нестаціонарним агрегатором) у існуючих рішень і може служити як для створення систем збору даних з нуля, так і для адаптації існуючих рішень до збору нестаціонарним агрегатором.

ПЕРЕЛІК ПОСИЛАНЬ

1. Balani Naveen. Enterprise IoT - A Definite Handbook / Ed. by Rajeev Hathi. – 2015.
2. AT COMMANDS. - URL:https: //halckemy.s3.amazonaws. com / uploads / document / file / 94325 / FE85NGOIH90OKGT.pdf .
3. Open Data Kit Sensors. - URL:https://opendatakit.org/use/ sensors /
4. ThingSpeak. ThingSpeak. - 2016. - URL:https://thingspeak.com
5. Axibase Collector. - URL:http://axibase.com/products/ axibase-time-series-database / writing-data / collector /
6. Axibase Time Series Database. - URL:http://axibase.com/ products / axibase-time-series-database /
7. Martin Atzmueller Katy Hilgenberg. Towards capturing social interactions with SDCF: an extensible framework for mobile sensing and ubiquitous data collection // MSM '13 Proceedings of the 4th International Workshop on Modeling Social Media. – 2013
8. Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides. – 1994
9. Dropbox. - URL:https://www.dropbox.com
10. Oracle. Java Software. - URL:https://www.oracle.com/java/
11. TIOBE Index. - URL:http://www.tiobe.com/tiobe_index
12. Global market share held by smartphone operating systems from 2009 to 2015. - URL: http://www.statista.com/statistics/263453/ global-market-share-held-by-smartphone-operating-systems /
13. 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

14. InternetofThingsPlatforms-Postscapes. - URL:http://postscapes.com/internet-of-things-platforms/

15. MbedOS. - URL: https://www.mbed.com/en/development/software / mbed-os /

16. HC-SR04 User Guide. - URL: http://elecfreaks.com/estore/ download / EF03085-HC-SR04_Ultrasonic_Module_User_Guide.pdf