ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут захисту інформації

Кафедра інформаційної та кібернетичної безпеки
Освітньо-кваліфікаційних рівень - бакалавр
Напрям підготовки 6.170101 Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідуючий кафедрою ІКБ
доктор технічних наук, с.н.с
В.Л. Бурячок

«____»_____2015 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
ГРЕБЕНЮКУ ОЛЕКСАНДРУ ВОЛОДИМИРОВИЧУ

1. Тема роботи: **Програмний комплекс для порівняльного аналізу цілісності передавання даних у безпроводових каналах зв'язку 2,4-2,5 ГГц**

   керівник: Соколов Володимир Юрійович, старший викладач; завдання затверджені наказом вищого навчального закладу від «27» лютого 2015 року № 87
2. Строк подання студентом роботи 8 червня 2015 р.
3. Вихідні дані до роботи: універсальний інструмент дослідження спектру для системних адміністраторів і науковців, опис принципів побудови програми, огляд характеристик технічних засобів, які застосовуються для аналізу безпроводових мереж зв’язку, огляд та вибір бібліотек та програмного забезпечення, реалізація протоколів і інтерфейсів, програмна реалізація.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
   4.1. Огляд бібліотек та програмного забезпечення.
   4.2. Середовище розробки.
   4.3. Конструювання графічного інтерфейсу.
   4.4. Використані бібліотеки.
   4.5. Реалізація протоколів і інтерфейсу.
   4.6. Загальна схема роботи програми.
   4.7. Шаблони проектування.

4.8. Робота з пристроями.
4.9. Аналіз даних.
4.10. Реєстрація повідомлень.
4.11. Використання вбудованого Wi-Fi адаптера для сканування каналів.
4.12. Графічний інтерфейс.
4.13. Допоміжні класи.
4.14. Реалізація функції повторного програвання (Replay).
4.15. Робота з аналізаторами спектру.

5. Перелік графічного матеріалу:
5.1. Рисунки, таблиці.
5.2. Презентація доповіді, виконана в Microsoft PowerPoint.

6. Дата видачі завдання 3 лютого 2015 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів	Примітка
1	2	3	4
1.	Розробка загальної схеми роботи програми	20.03.15 р.	
2.	Вибір бібліотек та програмного забезпечення	25.03.15 р.	
3.	Розробка архітектури програми	28.03.15 р.	
4.	Огляд бібліотек та програмного забезпечення	30.03.15 р.	
5.	Реалізація протоколів і інтерфейсу	08.04.15 р.	
6.	Робота з аналізаторами спектру	15.04.15 р.	
7.	Адаптація комплексу для роботи з аналізаторами спектру та вбудованою мережевою картою	20.04.15 р.	
8.	Адаптація для роботи під Windows та Linux	05.05.15 р.	
9.	Оформлення презентацій	10.05.15 р.	
10.	Отримання рецензій	15.05.15 р.	
11.	Захист в ДЕК	20.05.15 р.	

Студент Гребенюк О.В.

Керівник роботи Соколов В.Ю.

РЕФЕРАТ

Пояснювальна записка дипломного проекту: 92 сторінки, 28 рисунків, 8 джерел, 9 додатків.

Об’єкт проектування – програмне забезпечення, в якому можна порівнювати аналізатори спектрів різних версій від різних виробників.

Мета роботи – створити універсальний інструмент дослідження спектру для системних адміністраторів і науковців.

Метод проектування – розробка схеми роботи, адаптація пристроїв для роботи, вивчення принципів роботи з пристроями, реалізація коду програми.

Робота містить опис принципів побудови програми, огляд характеристик технічних засобів, які застосовуються для аналізу безпроводових мереж зв’язку. В першій частині роботи описані бібліотеки та програмне забезпечення. В другій – представлена реалізація протоколів і інтерфейсів, а третя частина присвячена програмній реалізації.

Галузь використання – адміністрування та аналіз безпроводових мереж зв’язку стандарту IEEE 802.11 2,4 ГГц.

АНАЛІЗАТОР СПЕКТРУ, РІВЕНЬ СИГНАЛУ, MDRV

ЗМІСТ

ВСТУП

А к т у а л ь н і с т ь т е м и. Швидкий розвиток безпроводових технологій призвів до заповнення робочих частот і збільшення взаємного впливу безпроводових мереж одна на одну. Такий негативний влив призводить до погіршення умов передавання даних, тобто погіршення цілісності обміну інформацією в таких мережах. Виявлення вільних частот в діапазоні дозволяє динамічно переналаштовувати мережу, покращуючи цілісність даних (як одного з елементів забезпечення безпеки).

Аналізатори спектра дозволяють проводити сканування частотного діапазону в режимі реального часу, таким чином, відповідають вимогам для інструментів для забезпечення цілісності даних. Для вимірювання інтенсивності електромагнітного поля в діапазоні 2,4–2,5 ГГц використовуються промислові аналізатори спектру на мікросхемах Chipcon CC2500 і CC2511-F32, Cypress CYRF6934, CYRF6935 і CYRF6936, які дозволяють отримати більш-менш адекватне уявлення про стан ефіру. Але різні пристрої працюють на різних операційних системах і в різних програмних середовищах з різною частотою оновлення отриманої і графічної інформації, тому порівняння таких пристроїв становить значну проблему.

Таким чином, в даній роботі зроблено спробу створити одне програмне забезпечення, в якому можна порівнювати аналізатори спектрів різних версій від різних виробників. Хоча для розробки вибрана мова програмування Java, проте через особливості драйверів деякі пристрої не мають сумісності з усіма операційними системами. Найбільш сумісною виявилася ОС Linux (дистрибутиви Ubunto, Mint і Gentoo), в якій підтримуються всі доступні автору аналізатори спектру.

В роботі описані методи проектування, що застосовувались в даному проекті; детально описаний процес роботи з кожним пристроєм і виявлені недоліки в їхніх конструкціях; вирішене питання в універсалізації інтерфейсів роботи з різними типами пристроїв (HID і віртуальний COM); вибрана методологія управління проектами Scrum; для легкості доопрацювання програмного коду, супроводу і повторного використання застосований модульних підхід. Вирішена задача накопичування даних, які використовуються для отримання поточних графіків для серед-

нього, модального і медіанного значень. При роботі з мережевими картами розроблений алгоритм оцінки загального рівня завантаження каналів. Реалізована одночасна робота кількох пристроїв від різних виробників і однакових пристроїв. Вирішена задача формату збереження та повторного використання збережених даних.

В першій частині роботи описані бібліотеки та програмне забезпечення. В другій – представлена реалізація протоколів і інтерфейсів, а третя частина присвячена програмній реалізації (робота за пристроями, обробка отриманих даних, представлення в графічній формі результатів обробки).

Метою даної дипломної роботи є створити універсальний інструмент дослідження спектру для системних адміністраторів і науковців.

М е т а і з а в д а н н я д о с л і д ж е н н я. **Метою роботи** створити універсальний інструмент дослідження спектру для системних адміністраторів і науковців. Для досягнення поставленої мети у роботі вирішуються такі **завдання**:

– адаптація аналізаторів спектру для роботи зпрограмним комплексом;

– порівняти іх між собою;

– зробити спектрограми в реальному часі;

– зібрати дані для майбутнього аналізу.

Виходячи з цього **об’єктом проектування** в роботі є програмне забезпечення, в якому можна порівнювати аналізатори спектрів різних версій від різних виробників. **Предметом дослідження** – оцінка спектру сигналу.

М е т о д и п р о е к т у в а н н я. розробка схеми роботи, адаптація пристроїв для роботи, вивчення принципів роботи з пристроями, реалізація коду програми.

Н а у к о в а н о в и з н а о д е р ж а н и х р е з у л ь т а т і в. У майбутньому можна додати підтримку іншим аналізаторам спектру, реалізувати графік типу waterfall, вдосконалити вже існуючі алгоритми, застосувати вмонтовану мережеву карту у якості аналізатора спектру.

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface – прикладний програмний інтерфейс

CSS – Cascading Style Sheets – каскадні таблиці стилів

COM – Communication Port, Serial Port – послідовний порт, двонаправлений послідовний інтерфейс, призначений для обміну байтовою інформацією

CVS – Concurrent Versioning System – система контролю версій

GNU – GNU’s Not UNIX – вільна UNIX-подібна операційна система, що розробляється Проектом GNU

GNU GPL – GNU General Public License – загальна публічна ліцензія GNU або Загальна громадська ліцензія GNU

HID – Human Interface Device – тип комп'ютерного пристрою для прямої взаємодії з людиною

IRP – I/O Request Packet – структура даних ядра ОС Windows, що забезпечує обмін даними між додатками і драйвером, а також між драйвером і драйвером

JDK – Java Development Kit – безкоштовний розповсюджуваний Oracle (раніше Sun) комплект розробника застосунків на мові Java

JNI – Java Native Interface – стандартний механізм для запуску коду, під керуванням віртуальної машини Java який написаний на мовах C / C ++, або Ассемблера

JSON – JavaScript Object Notation – текстовий формат обміну даними між комп'ютерами

JVM – Java Virtual Machine – набір комп'ютерних програм та структур даних, що використовують модель віртуальної машини для виконання інших комп'ютерних програм чи скриптів

NIO – Non-blocking I/O – колекція прикладних програмних інтерфейсів для мови Java, призначених для реалізації високопродуктивних операцій введення вивода

PID – Product Identifier – ідентифікатор продукту

RSSI – Received Signal Strength Indication – в телекомунікації, пристрій для вимірювання рівня потужності сигналу

UI – User Interface – інтерфейс користувача

USB – Universal Serial Bus – універсальна послідовна шина, призначена для з'єднання периферійних пристроїв обчислювальної техніки

VID – Vendor Identifier – ідетнифікатор постачальника

XML – Extensible Markup Language –стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками

ОС – Операційна система

1 ОГЛЯД БІБЛІОТЕК ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Середовище розробки

Для розробки проекту було використано IDE IntelliJ IDEA Community Edition (див. рисунок 1.1). IntelliJ IDEA – комерційне інтегроване середовище розробки для Java від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії Community Edition і повнофункціональної комерційної версії Ultimate Edition, для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні складання підготовлені для Linux, Mac OS X і Windows.

Community версія середовища IntelliJ IDEA підтримує інструменти для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial, Git і GitHub, засоби складання Maven і Ant, мови програмування Java, Java ME, Scala, Clojure, Groovy і Dart. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань [1].

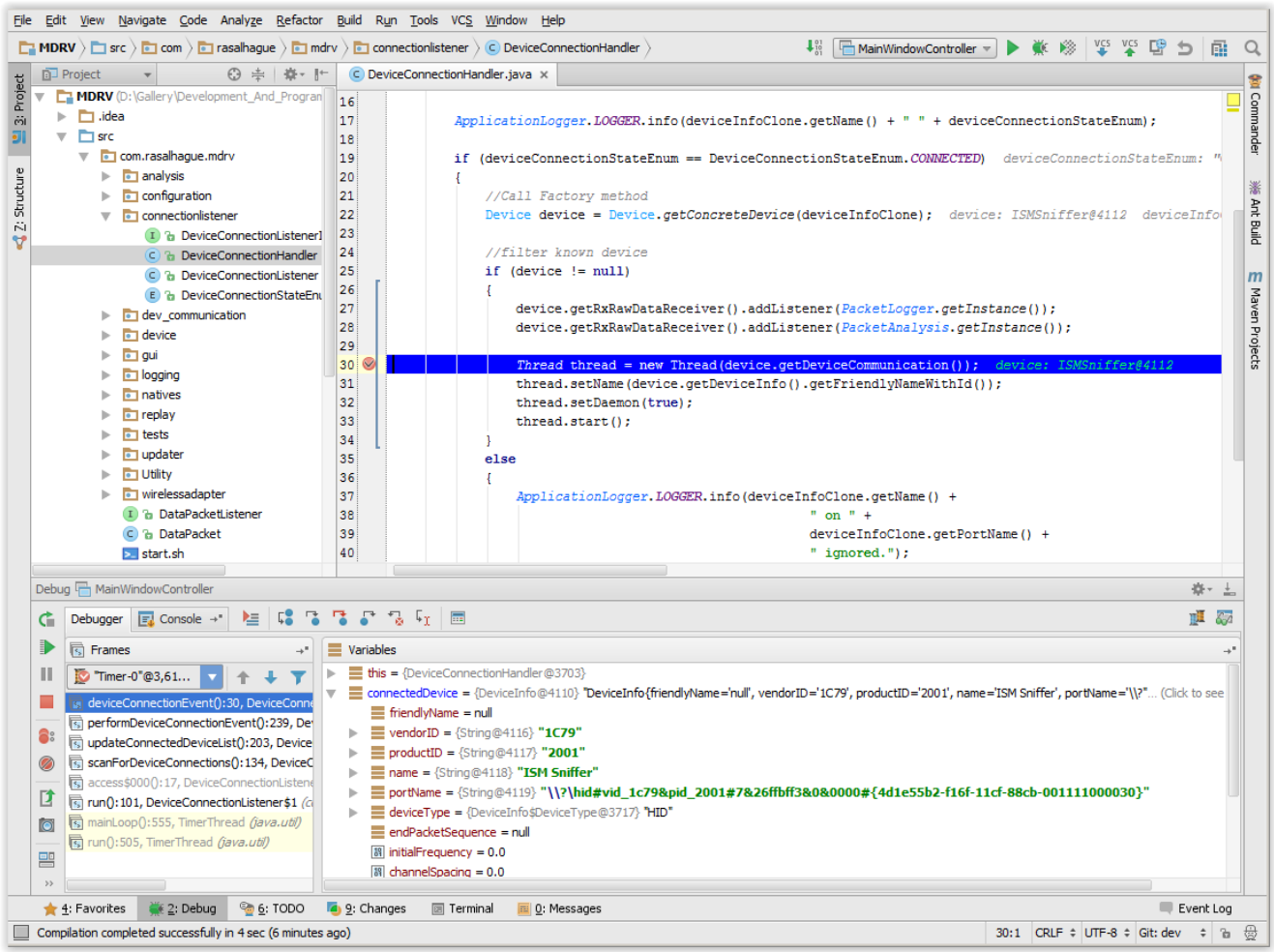


Рисунок 1.1. Процес налагодження програми в IntelliJ IDEA

1.2 Конструювання графічного інтерфейсу

Для конструювання графічного інтерфейсу технологія JavaFX використовує формат розмітки FXML – це декларативна мова на основі XML, яка створена корпорацією Oracle для визначення інтерфейсу користувача JavaFX 2.0. Oracle надає спеціальну програму для роботи з FXML – JavaFX Scene Builder 2.0. Приклад роботи програми JavaFX Scene Builder можна побачити на рисунок 1.2.

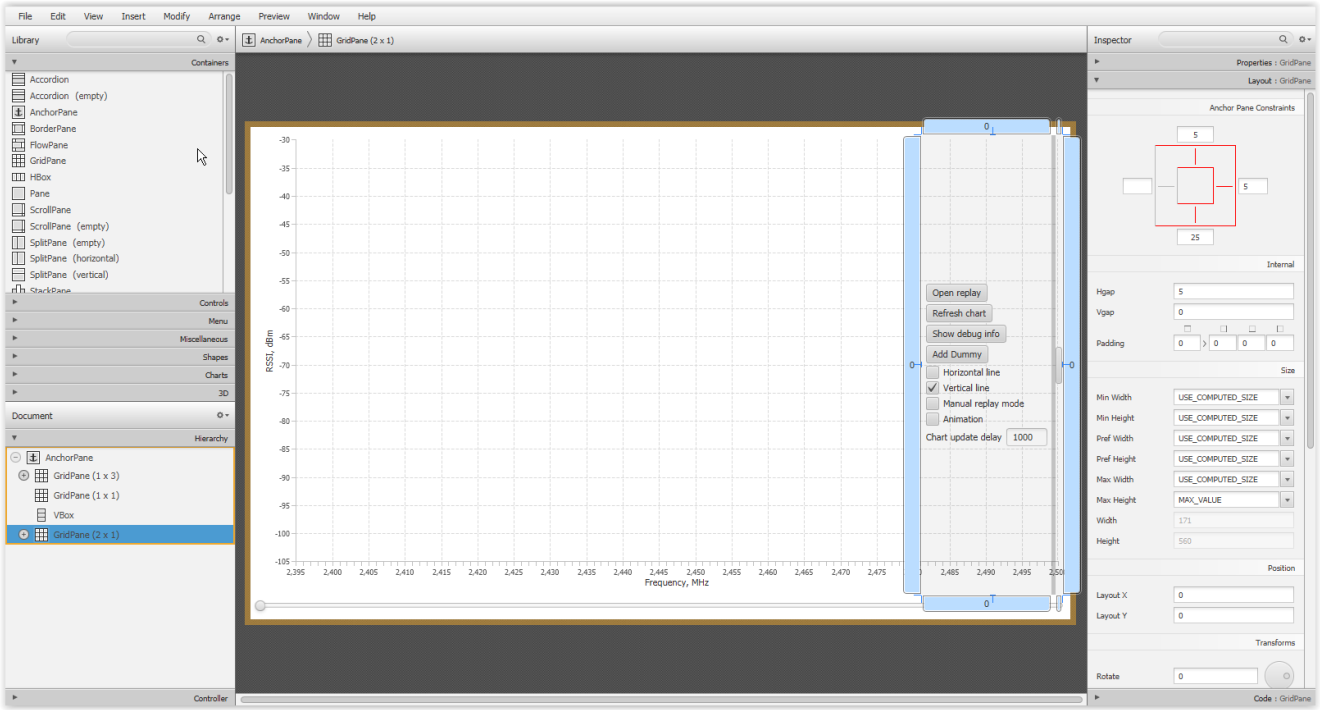


Рисунок 1.2. Конструювання файлу MainWindow.fxml у JavaFX Scene Builder

1.3 Використані бібліотеки

В даному проекті використовуються бібліотеки сторонніх розробників. Для завантаження бібліотек було використано Maven Central Repository.

Бібліотеки для роботи з пристроями:

– JavaHIDAPI;

– jSSC;

– usb4java.

Бібліотека для роботи з інтерфейсом користувача:

– ControlsFX.

Бібліотека для роботи с форматом JSON:

– Google Gson.

Бібліотеки, які розширюють стандартні можливості Java та підвищують якість коду в цілому:

– Apache Commons Collections;

– Apache Commons Lang;

– Reflections.

1.3.1 Apache Commons Collections

Java Collections Framework був важливим доповненням в JDK 1.2. Він додав, багато потужних структур даних, які прискорюють розробку найбільш значущих Java-додатків. Commons-Collections розвинули класи JDK шляхом надання нових інтерфейсів, реалізацій і утиліт. Надається під Apache License.

В даному проекті використовується клас бібліотеки BidiMap, який представляє собою мапу, ключ якої може буди використаний як значення, а значення як ключ.

1.3.2 Apache Commons Lang

Стандартні Java-бібліотеки не в змозі забезпечити достатньо методів для маніпулювання основними класами. Apache Commons Lang надає ці додаткові методи.

Lang надає безліч допоміжних утиліт для java.lang API, зокрема методів маніпуляції String, основні чисельні методи, reflection, concurrency, створення і серіалізация властивостей системи. Крім того, він містить основні вдосконалення java.util.Date. Надається під Apache License.

В даному проекті ця бібліотека використовується для більш зручного визначення ОС, на якій запущена програма (клас org.apache.commons.lang3. SystemUtils).

1.3.3 ControlsFX

ControlsFX є проект з відкритим кодом для JavaFX, яка покликана забезпечити дійсно високоякісний UI та інші інструменти, які доповнюють JavaFX. Бібліотека має JavaDoc документацію високої якості. Надається під BSD 3-Clause License.

В даному проекті бібліотека використовується як зручний засіб відображення діалогових вікон.

1.3.4 Google Gson

Gson це бібліотека для перетворення об’єктів Java у формат JSON. Вона також може бути використана для перетворення рядка JSON до еквівалентного об’єкта Java. Gson може працювати з довільними Java об’єктами, включаючи вже існуючі об’єкти, на які не мають сирцевого коду. Надається під Apache License 2.0.

Переваги Gson:

– забезпечує прості методи toJson і fromJson для перетворення Java об’єктів в JSON і навпаки;
– вже існуючі об’єкти можуть бути перетворені в та з JSON;
– розширена підтримка Java Generics;
– користувацькі подання для об’єктів.

В даному проекті використовується для підтримки функціонування функцій налаштувань програми та зберігання та програвання збережених треків.

1.3.5 JavaHIDAPI

Java HID API є JNI, що дозволяє використовувати бібліотеку HIDAPI з Java коду. HIDAPI є мультиплатформною бібліотекою, яка дозволяє додатку взаємодіяти з USB та Bluetooth пристроями HID-класу під Windows, Linux і Mac OS X.

Після створення екземпляра HIDManager можна використовувати деякі з його методів. Метод listDevices повертає список активних в даний момент часу HID. Кожен пристрій представлено екземпляром класу HIDDeviceInfo, який містить інформацію про пристрої. Щоб відкрити пристрій потрібно викликати метод Open.

HIDManager також надає кілька зручних методів для швидкого пошуку і відкриття пристрою або шляху (openByPath) або через vendor id / product id / serial number (openById).

Кожний відкритий пристрій представлено класом HIDDevice. Якщо пристрій відкрито кілька разів, HIDDevice буде однаковий, але безпека потоків не гарантується.

HIDAPI може бути використаний в рамках однієї з трьох ліцензій:

– GNU Public License, версія 3.0;

– BSD-стиль ліцензії;

– оригінал ліцензії HIDAPI.

1.3.6 jSSC

jSSC (Java Simple Serial Connector) – бібліотека для роботи з COM портами з Java. jSSC підтримує Win32 (Win98–Win8), Win64, Linux (x86, x86–64, ARM), Solaris (x86, x86–64), Mac OS X 10.5 і вище (x86, x86–64, PPC, PPC64). Надається під GNU Lesser GPL.

В даному проекті ця бібліотека використовується як основний засіб для взаємодії з COM-пристроями (Pololu Wixel, TI ez430-RF2500, Ubiquiti AirView2).

1.3.7 Reflections

В інформатиці відбиття або означає процес, під час якого програма може відстежувати і модифікувати власну структуру і поведінку під час виконання. Парадигма програмування, покладена в основу відображення, називається рефлексивним програмуванням. Це один з видів метапрограмування.

У більшості сучасних комп'ютерних архітектур програмні інструкції (код) зберігаються як дані. Різниця між кодом і даними в тому, що виконуючи код, комп'ютери обробляють дані. Тобто інструкції виконуються, а дані обробляються так, як написано цими інструкціями. Однак програми, написані за допомогою деяких мов, здатні обробляти власні інструкції як дані і виконувати, таким чином, рефлексивні модифікації. Такі самомодифікуючі програми в основному створюються за допомогою високорівневих мов програмування, що використовують віртуальні машини (наприклад, Smalltalk, скриптові мови).

Reflections сканує директорію класів, індексує метадані та дозволяє получати доступ до них під час виконання програми. Надається під Other Open Source License.

Використовуючи Reflections ви можете запросити такі метадані:

– підтипи певного типу;

– типи / constructos / методи / поля з анотацією;

– отримати всі погодження ресурсів відповідні регулярному виразу;

– отримати всі методи з конкретною сигнатурою, параметрами та типом повернення.

В даному проекті ця бібліотека використовується для знаходження всіх підкласів класу Device, тим самим генеруючи список всіх підтримуваних профілів пристроїв.

1.3.8 usb4java

usb4java це бібліотека Java для доступу до USB-пристроїв. Вона заснована на native libusb 1.0 і використовує Java NIO буфери для обміну даними між libusb і Java. usb4java також підтримує стандарт javax-USB (JSR-80) через розширення usb4java-javax.

Підтримує платформи Linux (x86 32/64 біт, ARM 32 біт), OS X (x86 32/64 біт) і Windows (тільки x86 32/64 біт). Але інші платформи можуть працювати так само добре (якщо є Java 6 і підтримуються libusb) шляхом компіляції бібліотеки JNI вручну. Надається під LGPL.

В даному проекті бібліотека використовується тільки у якості допоміжної.

1.4 Висновки до першого розділу

У першому розділі було розглянуто бібліотеки та програмне забезпечення. Зроблено огляд IDE IntelliJ IDEA та програми для роботи з FXML JavaFX Scene Builder. Вибрані бібліотеки для роботи з пристроями, інтерфейсом користувача, форматом JSON та допоміжні.

2 РЕАЛІЗАЦІЯ ПРОТОКОЛІВ І ІНТЕРФЕЙСУ

2.1 Загальна схема роботи програми

Загальна схема роботи програми (див. рисунок 2.1) зображена у вигляді діаграми послідовності.

При підключені користувачем пристрою до системи потік класу DeviceConnectionListener знаходить та ідентифікує його. Далі створюється новий потік для роботи з підключеним пристроєм.

Потік для роботи з пристроєм працює у циклі, в якому відбувається спроба зчитати данні. При успішному зчитуванні генерується подія, яка розповсюджує прийняті пристроєм значення RSSI у виді пакету.

PacketAnalysis аналізує отримані дані та генерує подію, яка перехоплюється системою графічного відображення даних для їх візуалізації.

PacketLogger зберігає пакети для можливості повторного використання.

Користувач також може використати вмонтовану мережеву карту для визначення навантаження конкретного каналу. При цьому активується клас WirelessAdapterCommunication, який зчитує данні з мережевої карти та генерує подію, яка перехоплюється системою графічного відображення даних для їх візуалізації.

Програма розроблялася модульною, так як з самого початку не було відомо чітких вимог до проекту. Модульність дозволяє зробити прозорішими тексти програми, прискорити написання і тестування, а також дає можливість легко супроводжувати її, так як для внесення змін потрібно працювати з одним модулем не чіпаючи інші [2–4].

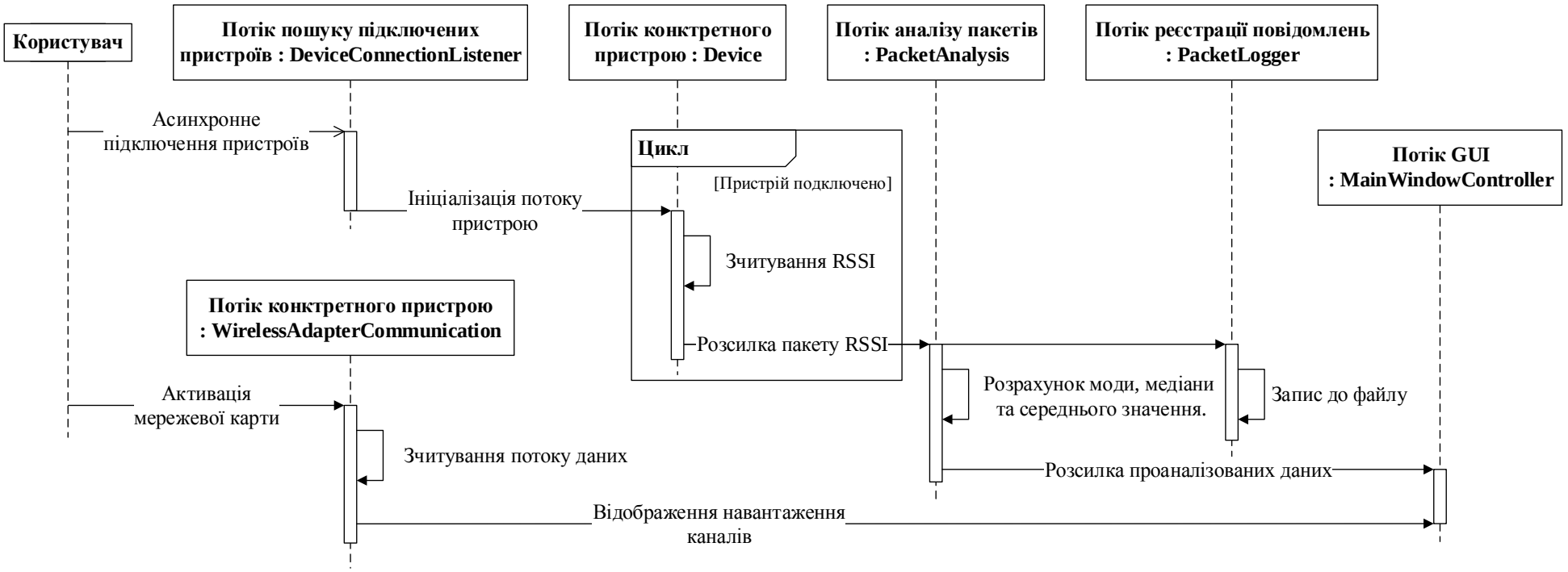


Рисунок 2.1. Загальна схема роботи програми

2.2 Шаблони проектування

2.2.1 Шаблон Observer

Спостерігач, Observer (див. рисунок 2.2) – поведінковий шаблон проектування. Також відомий як «підлеглі» (Dependents), «видавець-передплатник» (Publisher-Subscriber).

Призначення – визначає залежність типу «один до багатьох» між об’єктами таким чином, що при зміні стану одного об’єкта всіх залежних від нього сповіщають про цю подію.

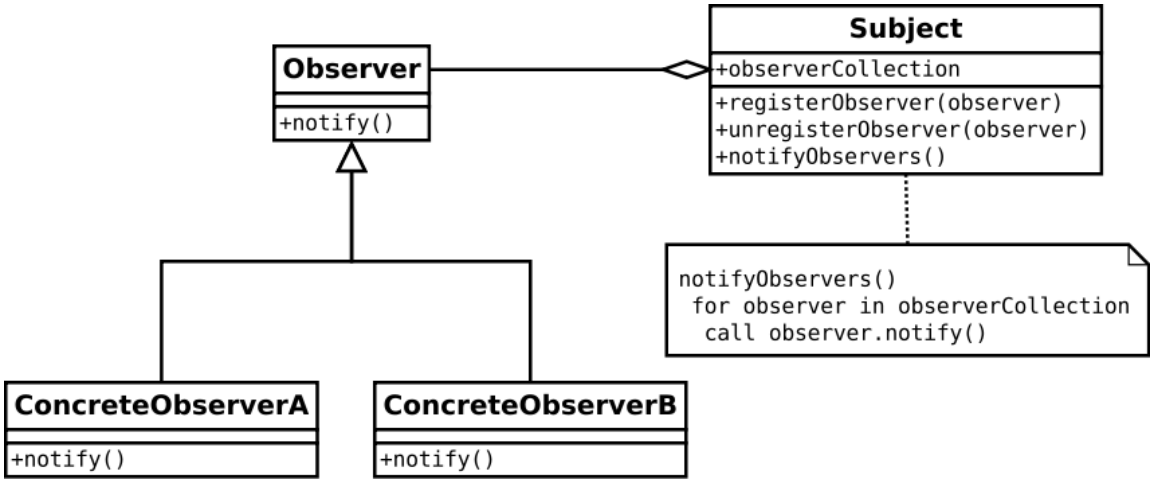


Рисунок 2.2. Схема шаблону проектування Observer

При реалізації шаблону Observer зазвичай використовуються наступні класи:

– Observable – інтерфейс, що визначає методи для додавання, видалення та оповіщення спостерігачів;

– Observer – інтерфейс, за допомогою якого спостерігач отримує сповіщення;

– ConcreteObservable – конкретний клас, який реалізує інтерфейс Observable;

– ConcreteObserver – конкретний клас, який реалізує інтерфейс Observer.

Шаблон Observer застосовується в тих випадках, коли система володіє такими властивостями:

– існує, як мінімум, один об’єкт, що розсилає повідомлення;

– є не менше одного одержувача повідомлень, причому їхня кількість і склад можуть змінюватися під час роботи програми.

Цей шаблон часто застосовують в ситуаціях, в яких відправника повідомлень не цікавить, що роблять одержувачі з наданою їм інформацією.

В даній програмі шаблон Observer застосовується для комунікації між модулями програми. Це дозволяє явно відокремити їх один від одного, роблячи їх незалежними.

2.2.2 Шаблон Singleton

Одинак, Singleton – шаблон проектування, відноситься до класу твірних шаблонів. Гарантує, що клас матиме тільки один екземпляр, і забезпечує глобальну точку доступу до цього екземпляра.

Для деяких класів важливо, щоб існував тільки один екземпляр. Рішення полягає в тому, щоб сам клас контролював свою «унікальність», забороняючи створення нових екземплярів, та сам забезпечував єдину точку доступу. Це є призначенням шаблону Одинак.

В даному проекті шаблон Singleton використовується для наступних класів: допоміжного, реєстрації повідомлень, аналізу пакетів, меню налаштувань, пошуку підключених пристроїв.

2.2.3 Шаблон Strategy

Стратегія, Strategy (див. рисунок 2.3) – шаблон проектування, належить до класу шаблонів поведінки. Відомий ще під іншою назвою – "Policy". Його суть полягає у тому, щоб створити декілька схем поведінки для одного об’єкту та винести в окремий клас. Шаблон Strategy дозволяє міняти вибраний алгоритм незалежно від об’єктів-клієнтів, які його використовують.

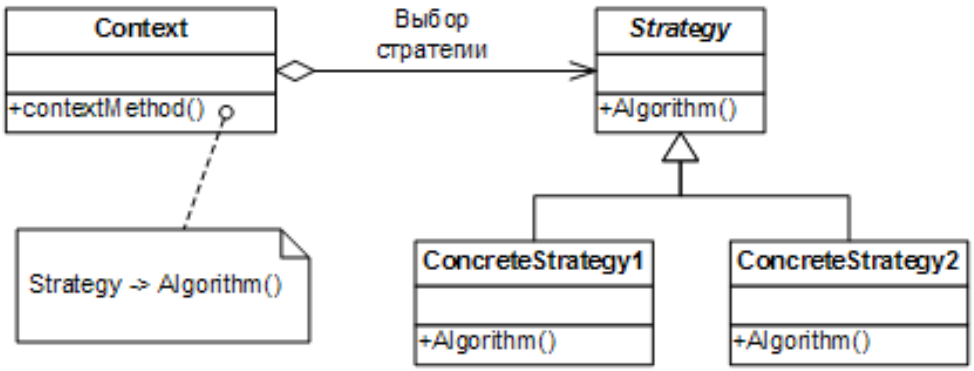


Рисунок 2.3. Схема шаблону проектування Strategy

Завдання шаблону – за типом клієнта (або за типом оброблюваних даних) вибрати відповідний алгоритм, який слід застосувати.

Учасники:

– Strategy – визначає, як будуть використовуватися різні алгоритми;

– ConcreteStrategy – реалізують ці різні алгоритми;

– Context – використовує конкретні класи ConcreteStrategy за допомогою посилання на конкретний тип абстрактного класу Strategy.

В даному проекті шаблон Strategy використовується для вибору конкретного пристрою зі стеку доступних, виходячи з його параметрів [5].

2.3 Абстрактний клас Device та його реалізації

Device – клас, який абстрагує загальну поведінку для всіх пристроїв (див. додаток В).

2.3.1 Статичний фабричний метод

Звичайний спосіб отримання екземпляру класу – відкритий конструктор. Існує ще один метод: клас може забезпечити статичний фабричний метод, який повертає екземпляр класу.

Однією з переваг статичних фабричних методів є те, що на відміну від конструкторів, вони мають імена. Друга перевага статичних фабричних методів: вони не зобов’язані створювати новий об’єкт при виклику. Третя перевага статичних фабричних методів є те, що на відміну від конструкторів, вони можуть повернути об’єкт будь-якого підтипу.

У класі Device використовується статичний фабричний метод з наступною сигнатурою:

```
public static Device getConcreteDevice(DeviceInfo deviceInfo)
```

Клас DeviceConnectionHandler викликає фабричний метод getConcreteDevice з параметром DeviceInfo в якому знаходиться інформація о пристрої.

Задача методу getConcreteDevice: на основі даних з DeviceInfo повернути ініціалізований конкретний екземпляр класу пристрою. Це досягається за допомогою спеціального механізму Reflection. Використовуючи Reflection реалізовано пошук конкретного класу по всім можливим нащадкам абстрактного класу Device. Такий підхід дає можливість додавати підтримку реалізацій нових пристроїв не чіпаючи при цьому інші класи.

2.3.2 Конкретні реалізації

Для реалізації конкретного пристрою треба заповнити шаблон DeviceTemplate (див. додаток Г). Далі наведено скорочений лістинг шаблону DeviceTemplate:

```
public class DeviceTemplate extends Device
{
    public final static String FRIENDLY_NAME = "";
    public final static String VENDOR_ID = "";
    public final static String PRODUCT_ID = "";
    public final static float INITIAL_FREQUENCY = 2400f;
    public final static float CHANNEL_SPACING = 0f;
    public final static byte[] END_PACKET_SEQUENCE = new byte[]{};
    public final static boolean MANUAL_DEVICE_CONTROL = false;

    @Override
    public void initializeDevice()
    {    }

    @Override
    public ArrayList<Byte> parse(ArrayList<Byte> dataToParse)
    {
        ArrayList<Byte> finalArray = new ArrayList<>(dataToParse);

        return finalArray;
    }
```

```
    @Override
    public byte[] customReadMethod()
    {
        return new byte[0];
    }
}
```

Поле FRIENDLY_NAME використовується для ідентифікації пристрою у графічному інтерфейсі.

Поле VENDOR_ID використовується для зберігання ідентифікатору виробника пристрою. Повинен бути у шістнадцятковій системі числення. На приклад "1FFB".

Поле PRODUCT_ID використовується для зберігання ідентифікатору пристрою. Повинен бути у шістнадцятковій системі числення.

Поле INITIAL_FREQUENCY використовується для зберігання мінімальної частоти, яку пристрій може бачити (Base Frequency). Значення береться з документації к пристрою. На приклад 2400f.

Поле CHANNEL_SPACING використовується для визначення між каналами. Значення береться з документації к пристрою. На приклад 327.450980f.

Поле END_PACKET_SEQUENCE використовується для зберігання символів кінця пакету. Символи кінця пакету використовуються класом RxRawDataReceiver для генерування пакетів зі значеннями RSSI.

Поле MANUAL_DEVICE_CONTROL встановлюється, коли треба отримати прямий контроль над USBHID. Це потрібно у ситуаціях, коли бібліотека за замовчуванням не спрацьовує за якихось причин. У цьому режимі програма не відкриватиме та не робитиме спроб зчитати з пристрою. Метод customReadMethod активується. Приклад: клас MetaGeekWiSpyGen1.

Метод initializeDevice з наступною сигнатурою:

```
public void initializeDevice()
```

використовується для ініціалізації пристрою, якщо це потрібно. Інакше тіло методу можна залишити пустим.

Метод parse з наступною сигнатурою:

```
public ArrayList<Byte> parse(ArrayList<Byte> dataToParse)
```

використовується для розбору даних, які були сформовані пристроєм. Формат повернення – масив байтів, кожен елемент якого – значення RSSI.

Метод customReadMethod з наступною сигнатурою:

```
public byte[] customReadMethod()
{
	return new byte[0];
}
```

використовується для перевизначення стандартної поведінки USBHID. Для використання цього методу треба встановити прапорець MANUAL_DEVICE_CONTROL.

У шаблоні DeviceTemplate реалізована JavaDoc документація за допомогою якої кожен може додати в програму підтримку свого пристрою.

2.4 Робота з пристроями

2.4.1 Пошук підключених пристроїв

DeviceConnectionListener (див. додаток А) – клас, задача якого сканувати систему на предмет підключень пристроїв через задані проміжки часу (за замовчуванням – 1 с).

Клас реалізує патерн програмування Singleton, тому що немає сенсу запускати в одній програмі декілька екземплярів цього класу. Також реалізує патерн програмування Observer, за допомогою якого відбувається нотифікація підписчиків на подію підключення пристрою. Клас DeviceConnectionHandler (див. додаток Б) підписується на подію підключення пристрою, викликає статичний метод для пошуку конкретного пристрою, яке підтримує програма та, у випадку успіху, запускає потік для роботи з пристроєм.

2.4.2 Взаємодія з пристроєм

Основним класом для взаємодії програми з пристроєм є абстрактний клас DeviceCommunication (див. додаток Д). Він реалізує шаблон програмування

Singleton та має статичний фабричний метод getInstance, який повертає конкретну реалізацію, залежно від переданого параметра DeviceInfo. Сигнатура функції getInstance:

```
public static DeviceCommunication getInstance(DeviceInfo deviceInfo)
```

Всього є три конкретні реалізації абстрактного класу DeviceCommunication:

– COMDeviceCommunication – для взаємодії з COM-пристроями;
– HIDDeviceCommunication – для взаємодії з USBHID;
– DummyDeviceCommunication – для реалізації тестового програмного пристрою.

DeviceCommunication спроектований для роботи в окремому потоці за допомогою реалізації інтерфейсу Runnable. Код старту потоку наведено нижче:

```
Thread thread = new Thread(device.getDeviceCommunication());
thread.setName(device.getDeviceInfo().getFriendlyNameWithId());
thread.setDaemon(true);
thread.start();
```

У першому рядку створюється об’єкт потоку з параметром ініціалізованого конкретного класу DeviceCommunication.

У другому рядку задається ім’я потоку для його ідентифікації у разі виникнення виключення.

Методом setDaemon потік помічається як потік користувача. JVM завершує свою роботу тільки тоді, коли всі потоки, що залишилися, помічені як потоки користувача. Це запобігає ситуаціям, коли програма не може бути завершена оскільки один з потоків взаємодії з пристроєм не відповідає.

2.4.3 Спадкування класу DeviceCommunication

Дана програма може працювати в режимі реального часу с декількома пристроями різних типів. Підтримуються 2 типи пристроїв: перший визначається системою як Human Interface Device; інший – як Serial Port Device. Для кожного з типів потрібен свій варіант взаємодії. Тому було вирішено скористатися спадкуванням (див. рисунок 2.4). У даній ситуації структура коду схожа на структуру шаблона

програмування Strategy, тільки замість інтерфейсу використовується абстрактний клас. Також можна використати інтерфейс, оголосивши стандартну реалізацію за допомогою спеціальних default методів (замість абстрактних), які реалізовані в інтерфейсах Java версії 8.

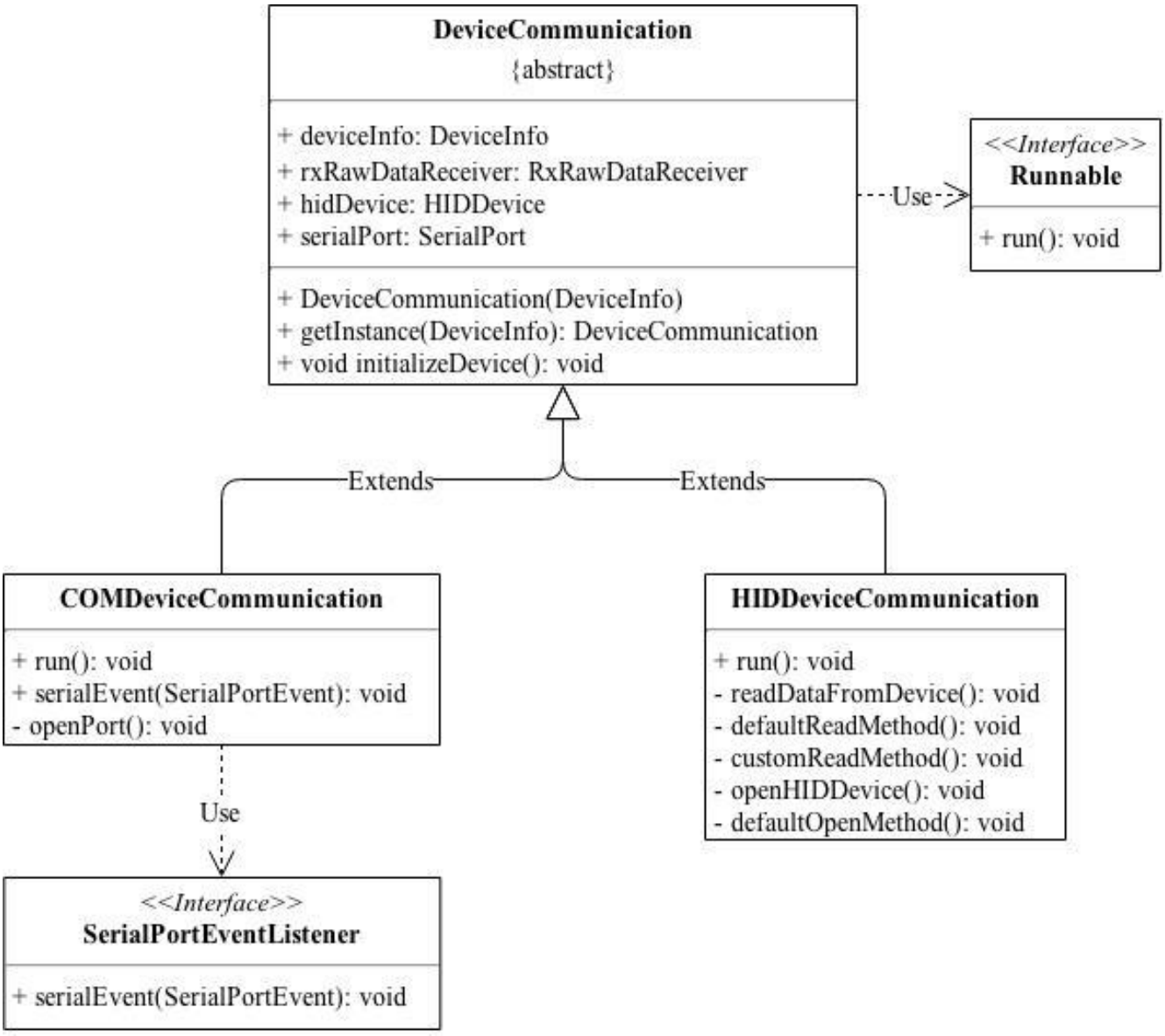


Рисунок 2.4. Спадкування класу DeviceCommunication

Це рішення розділяє роботу з кожним типом пристрою на свій конкретний клас, що задовольняє вимогам об’єктно-орієнтованого програмування і дозволяє додавати нові типи пристроїв не змінюючи при цьому код інших.

2.4.4 Прийняття даних з пристрою та генерування пакету

Кожен об’єкт класу DeviceCommunication має вкладений клас RxRawDataReceiver,основна задача якого приймати сирі дані від пристрою, складати їх в пакет (масив значень RSSI) та сповіщати про складання нового пакету всіх підписчиків – класу реєстрації пакетів PacketLogger та класу аналізу пакетів PacketAnalysis.

Основний принцип роботи полягає у виклику методу receiveRawData з наступною реалізацією:

```
public void receiveRawData(byte[] rawData)
{
    for (byte byteProcess : rawData)
        processReceivedByte(byteProcess);
}
```

Цей метод працює у якості буфера для метода processReceivedByte, що приймає один байт. processReceivedByte накопичує прийнятий байт у буфері із якого у подальшому генерується пакет з RSSI. Для визначення кінця пакету використовується поле END_PACKET_SEQUENCE. При знаходженні потрібної послідовності послідовність переноситься у початок пакету (для ситуацій, коли послідовність кінця пакету знаходиться у початку наступного фізичного пакету з пристрою) та викликається методи, які зберігають зібранні дані та сповіщають підписчиків.

2.5 Аналіз даних

За аналіз даних відповідає клас PacketAnalysis (див. додаток Е). PacketAnalysis генерує наступні масиви даних: мода, медіана, середнє арифметичне і максимальне.

Для генерування масиву максимальних значень використовується спеціальна функція, яка приймає як параметри поточний та попередній масив значень, порівнює значення цих двох масивів та заносить більше значення у масив, який повертається.

Для генерування масиву з середніми арифметичними значеннями використовується формула

$$\frac{1}{n}\sum_{i=1}^{n} x_i \,,$$

де x – значення RSSI на заданій частоті; n – загальна кількість пакетів зі значеннями RSSI для конкретного пристрою.

Основна проблема була в лінійному збільшенні часу на розрахунок, оскільки кількість пакетів зі значеннями RSSI постійно збільшується, що лінійно збільшує кількість ітерацій. Рішення наступне: для запобігання повторного розрахунку усіх значень при кожному новому пакеті введено структуру даних, яка зберігає суму всіх попередніх значень RSSI ($A_1 - A_3, B_1 - B_3, C_1 - C_3$) для кожної частоти для конкретного пристрою та загальну кількість прийнятих пакетів L_Σ (див. рисунок 2.5). Це дозволяє, при надходженні нового масиву RSSI, просто додати нове значення до вже наявної суми та розділити на загальну кількість.

Для підрахунку моди треба визначити значення, що трапляється найчастіше. Якщо сканувати кожен раз всі надіслані пакети, то, як і у випадку з середнім значенням, складність розрахунку буде лінійно зростати. Тому введено допоміжну структуру наступного виду:

```
HashMap < [Пристрій] -> ArrayList < HashMap < [RSSI] -> [Кількість] > > >
```

Кожному конкретному пристрою відповідає масив мап з RSSI у якості ключа та кількістю значень RSSI у якості значення. На приклад маємо наступні значення (див. рисунок 2.5): L_1 ($A_1 = -90$; $B_1 = -80$; $C_1 = -70$); L_2 ($A_2 = -80$; $B_2 = -80$; $C_2 = -70$); L_3 ($A_3 = -80$; $B_3 = -75$; $C_3 = -60$), які відповідають наступному матричному представленню, у якому кожному пристрою (Device) відповідає масив (ArrayList, рядки матриці), в якому знаходиться мапа з відношенням значень RSSI до їх кількості:

$$[Device] \rightarrow \begin{matrix} -90 \rightarrow 1 & -80 \rightarrow 2 \\ -80 \rightarrow 2 & -75 \rightarrow 1 \\ -70 \rightarrow 2 & -60 \rightarrow 1 \end{matrix}$$

При кожному надходженні пакету до поточної кількості значень RSSI додається 1 у випадку колізії. Така структура дає можливість використовувати накопичені значення для підрахунку моди з постійною швидкістю – потрібно лише взяти ключ з найбільшою кількістю значень. Ця ж структура використовується і для пошуку медіани – масив сортується та береться значення, яке знаходиться посередині масиву.

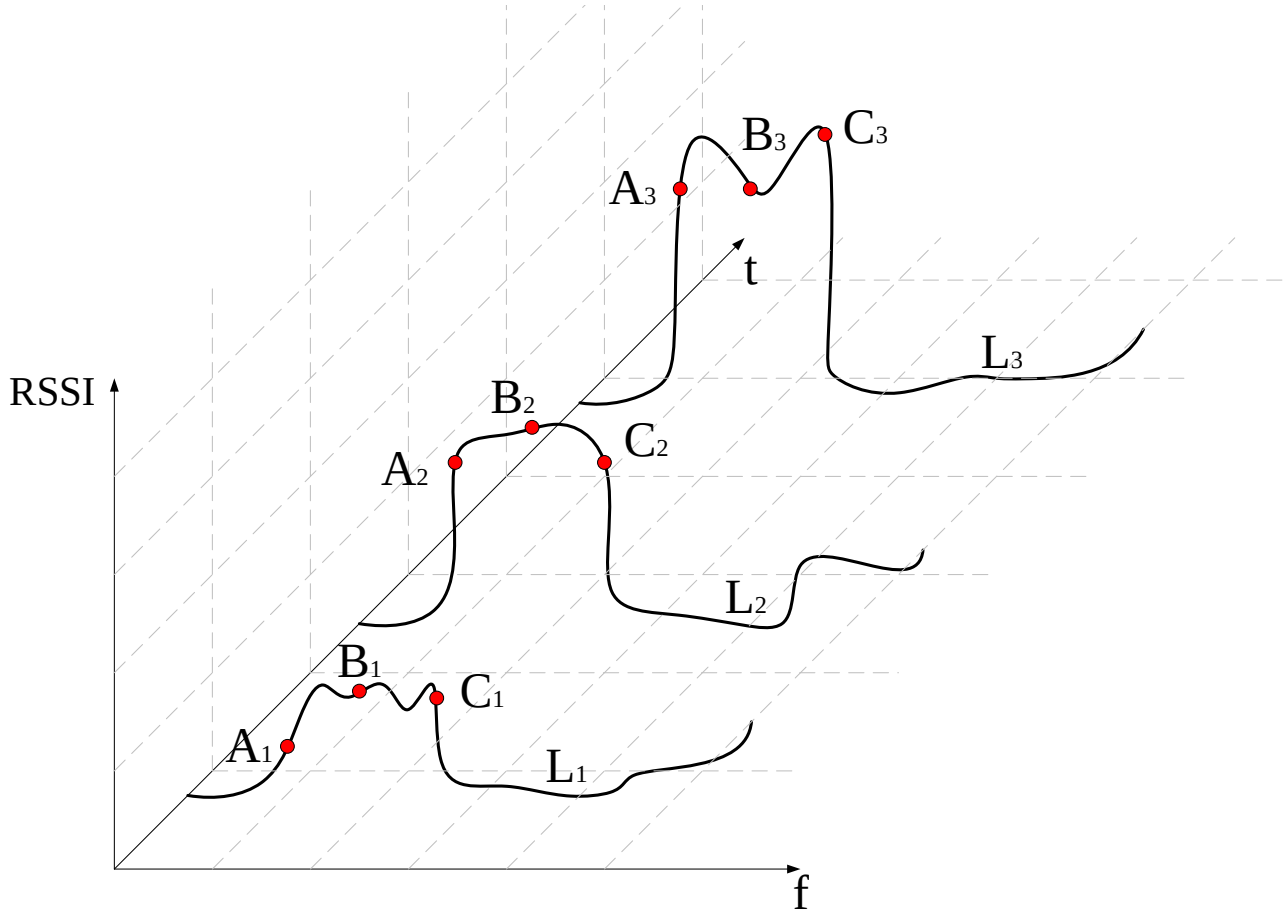


Рисунок 2.5. Схематичне відображення масивів RSSI у часі

Проаналізовані данні зберігаються у наступній структурі даних:

```
LinkedHashMap<Long, HashMap<DeviceInfo, HashMap<AnalysisKey,
ArrayList<Byte>>>>
```

Ідея структури наступна: проаналізовані дані (HashMap<AnalysisKey, ArrayList<Byte>>) прив’язується до конкретного пристрою (DeviceInfo), який прив’язується до мітки у момент його створення. LinkedHashMap використовується

для гарантії того, що дані в структурі будуть у тій послідовності в якій вони були внесені.

PacketAnalysis використовує шаблон проектування Observer для сповіщення класів, які зацікавленні у проаналізованих даних.

2.6 Реєстрація повідомлень

Для реєстрації повідомлень використовується вбудований клас java.util.logging.Logger пакету java.util.logging. Об'єкт Logger використовується для запису повідомлень для конкретної системи або компонента програми. Logger, як правило, використовує ієрархічну структуру імен. Імена можуть бути довільними, як правило, встановлюються на основі імені пакета чи імені класу. Крім того, можна створювати «анонімний» Logger, що не зберігаються в просторі імен реєстратора.

Об'єкти Logger може бути отримана шляхом виклику getLogger factory methods, який створить новий Logger або поверне підходящий існуючий. Важливо відзначити, що Logger повернений одним з методів getLogger може бути знищений при збірці сміття в будь-який момент, якщо на нього немає посилання.

Повідомлення будуть пересилатися на зареєстровані оброблювачі, який може пересилати повідомлення в різні напрямки, в тому числі консолі, файли, журнали ОС і т. п.

Реєстратор повідомлень підтримує декілька рівнів повідомлень: OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST.

В даному проекті за реєстрацію повідомлень відповідає клас ApplicationLogger (див. додаток Ж). Приклад використання:

```
ApplicationLogger.setup();
ApplicationLogger.LOGGER.info("MetaGeek_WiSpy24x2 has been initialized");
```

Метод ApplicationLogger.setup викликається один раз при старті програми для ініціалізації об'єкту реєстратора повідомлень та створення log-файлу. У класі MyLogFormatter, який розширює клас Formatter, вказано формат виводу за допомогою пере визначення наступної функції, яка викликається для кожної реєстрації повідомлення:

```
@Override
public String format(LogRecord record)
```

Можливий варіант зареєстрованих повідомлень виглядає так:

```
16.12.14 12:55:48.523 [INFO] [logging.ApplicationLogger.setup] Logger has
initialized
16.12.14 12:55:48.572 [INFO] [updater.UpdateChecker.initCurrentVersion]
Current version: Version{major=0, minor=3}
16.12.14 12:55:49.385 [INFO]
[connectionlistener.DeviceConnectionListener.runSchedule] Listening
schedule has started. Waiting for devices...
16.12.14 12:55:49.397 [WARNING]
[wirelessadapter.WirelessAdapterCommunication.searchWirelessAdapters] phy#0
16.12.14 12:55:49.418 [INFO]
[connectionlistener.DeviceConnectionListener.deviceConnectionEvent] USB
Device CONNECTED
16.12.14 12:55:49.448 [INFO]
[wirelessadapter.WirelessAdapter.setUpChannelToFrequencyMap]
channelToFrequencyMap: {1=2412, 2=2417, 3=2422, 4=2427, 5=2432, 6=2437,
7=2442, 8=2447, 9=2452, 10=2457, 11=2462, 12=2467, 13=2472, 14=2484}
```

Формат виводу наступний:

```
[Date] [Time] [Logging Level] [The Class from which logger has been called]
[Message]
```

2.7 Використання вбудованого Wi-Fi адаптера для сканування каналів

Якщо при завантажені програма встановлює, що в системі є бездротовий модуль з підтримкою режиму моніторингу, то програма автоматично розпочинає процедуру сканування, використовуючи заданий в меню налаштувань діапазон.

WirelessAdapterCommunication – клас, задача якого розпочати процедуру сканування. Для цього спочатку виконуються пошук адаптерів:

```
ArrayList<WirelessAdapter> wirelessAdapters = searchWirelessAdapters();
```

Якщо хоч один адаптер був знайдений, то викликається функція вибору адаптеру, яка виводить користувачу список із доступних адаптерів. Варто відмітити, що функція вибору адаптеру автоматично повертає адаптер, якщо він один:

```
WirelessAdapter wirelessAdapter = chooseWirelessAdapter(wirelessAdapters);
```

Після вибору адаптеру програма переводить його у режим моніторингу, використовуючи вмонтовані в дистрибутив програми ifconfig та iw:

```
switchToMonitorMode(wirelessAdapter);
```

Далі розпочинається переключення каналів у заданому діапазоні та активація програми tcpdump, яка дозволяє захоплювати і аналізувати мережний трафік, що проходить через комп'ютер, на якому запущена ця програма:

```
startChannelSwitching(wirelessAdapter);
startListening(tcpDumpCommand, wirelessAdapter);
```

Приклад роботи наведено на наступному рисунку (див. рисунок 2.6). Поточне значення RSSI підтверджує завантаженість діапазону частот, що підсвічується.

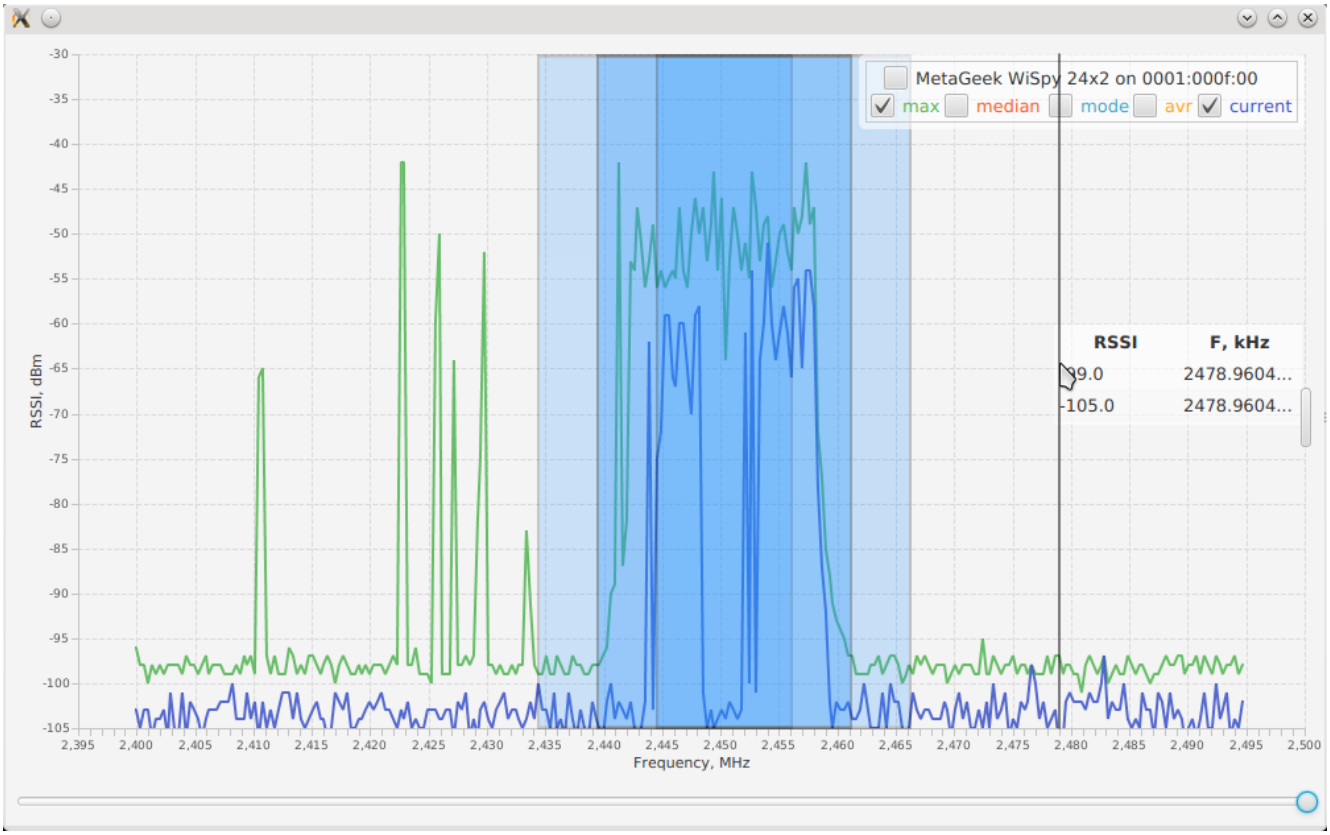


Рисунок 2.6. Приклад роботи мережевої карти у режимі моніторингу.
Завантаженість в діапазоні 2435–2465 МГц

Діапазон каналів для сканування розпочинається від 1 та закінчується 14. Для зміни діапазону треба заповнити відповідне поле у меню налаштувань. Програма також розпізнає потрібний напрямок сканування. Регулярне вираження для розпізнання діапазону каналів наведено нижче:

```
(?<channelStart>\d{1,2}[^15-99]*?)(-(?<channelEnd>\d{1,2}[^15-99]*?))?
```

На приклад, строчка «9-7» при проходженні регулярного вираження буде мати дві групи: перша, channelStart, буде містити початкове значення діапазону – 9; друга, channelEnd, буде містити кінцеве значення діапазону – 7.

2.8 Графічний інтерфейс

Графічний інтерфейс (див. рисунок 2.7) реалізовано за допомогою технології JavaFX. JavaFX – платформа та набір інструментів для створення насичених інтернет застосунків (англ. Rich Internet Applications, RIA) з можливістю підвантаження медіа та змісту.

Рисунок 2.7. Графічний інтерфейс програми: 1 – LineChart;
2 – список підключених пристроїв; 3 – плаваюча підказка;
4 – меню налаштувань; 5 – шкала часу

Елемент GUI LineChart (1) відповідає за відображення та оновлення графіків. Вісь абсцис відповідає частоті та вимірюється в мегагерцах. Вісь ординат відповідає RSSI рівню прийнятого сигналу.

Список підключених пристроїв (2) включає в себе кнопки управління відображенням графіків максимального, моди, медіани, середнього та поточного значень. Також користувач може включити чи виключити одразу всі графіки за допомогою CheckBox навпроти назви пристрою.

Плаваюча підказка (3) з поточним значенням RSSI та відповідною частотою з’являється при переміщені курсора у вертикальній площині у межах графіку.

Доступ до меню налаштувань можна отримати за допомогою спеціальної області (4), на яку треба навести курсор для активації.

При створенні пакету зі значеннями RSSI також створюється мітка часу, до якої прив’язаний створений пакет. Для переміщення по міткам часу використовується слайдер (5).

Графік має 2 осі: вісь абсцис позначена як Frequency та вимірюється в мегагерцах; вісь ординат позначена як RSSI та вимірюється в децибелах в розрахунку на 1 міліват.

2.8.1 Меню налаштувань

Меню налаштувань (див. рисунок 2.8) розроблену у виді вертикального списку.

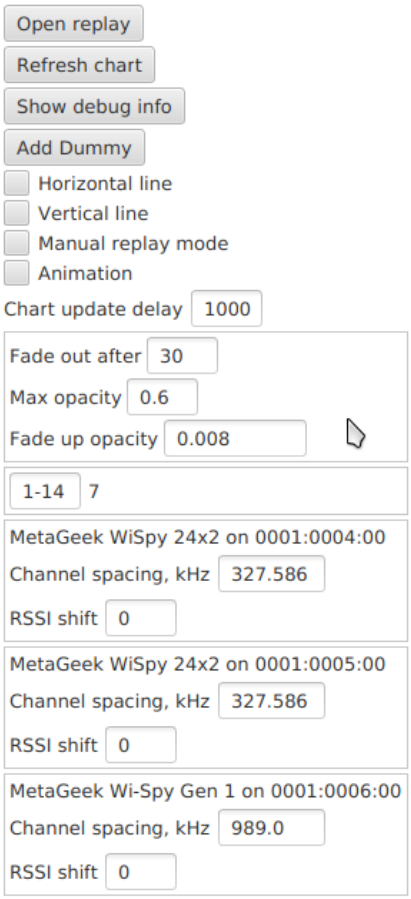


Рисунок 2.8. Меню налаштувань

Кнопка **Open replay** потрібна для відкриття файлу з попередньо записаними даними.

Кнопка **Refresh chart** потрібна для очищення графіку від набраної інформації.

Кнопка **Show debug info** включає режим показу додаткової інформації, яку можна використовувати для налагодження та тестування програми (див. рисунок 2.9). Додаткова інформація включає в себе сітку для калібрування каналів та поле для показу файла історії програми.

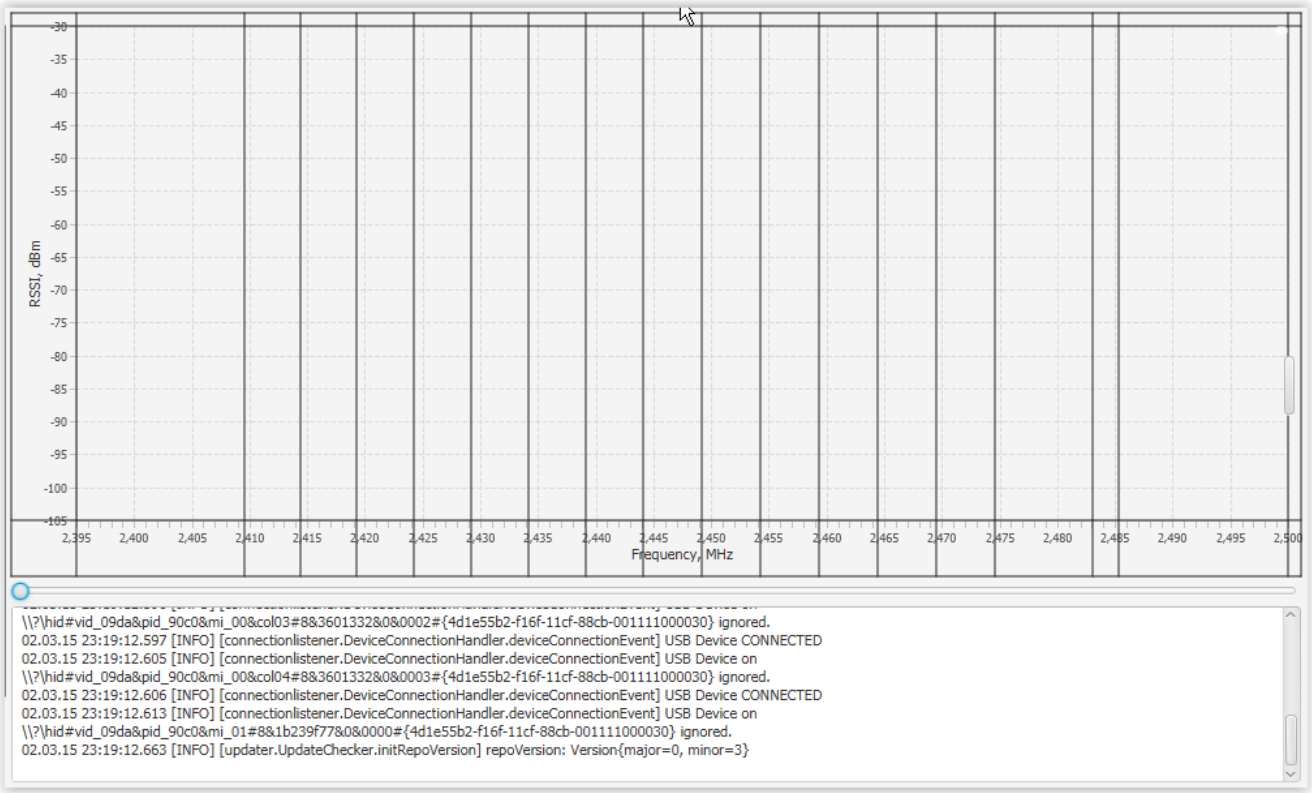


Рисунок 2.9. Додаткова інформація для налагодження

Кнопка **Add Dummy** потрібна для додавання до програми тестового програмного пристрою.

CheckBox **Horizontal line** та **Vertical line** відповідають за відображення допоміжних ліній на графіку.

За замовчуванням програма відображає значення пакетів RSSI в реальному часі, тобто індикатор шкали часу завжди знаходиться вкінці шкали. CheckBox **Manual replay mode** дозволяє включити режим ручного управління часом.

CheckBox **Animation** дозволяє включити або виключити анімацію. На деяких слабких системах це є актуально.

У полі **Chart update delay** можна встановити інтервал оновлення графіку. Ця функція корисна для слабких систем та у випадках, коли треба аналізувати поточні дані з пристрою в реальному часі.

Наступний блок відповідає за налаштування відображення завантаженості каналу, що сканується бездротовим адаптером. У полі **Fade out after** вказується кількість секунд, до зникнення ефекту завантаженості каналу. У полі **Max opacity** вказується ступінь непрозорості ефекту завантаженості каналу. У полі **Fade up opacity** вказується коефіцієнт, що визначає як швидко буде ефект завантаженості каналу буде набирати непрозорості.

У наступному полі вказується канал чи діапазон каналів, що скануються. Відповідає наступному регулярному виразу:

```
"(?<channelStart>\\d{1,2}[^15-99]*?)(-(?<channelEnd>\\d{1,2}[^15-99]*?))?"
```

Тобто доступні значення для вводу від 1 до 14 включно. Щоб вказати діапазон значень треба скористатися символом тире. Канали будуть переключатися у тій послідовності, що вказана у полі.

При підключені пристрою в меню налаштувань автоматично генерується налаштування для щойно підключеного пристрою та зазвичай потрібні для калібрування пристрою. Поле **Channel spacing** відповідає за відстань між значеннями RSSI на графіку. Поле **RSSI shift** відповідає за зсув значень RSSI на графіку по шкалі абсцис.

2.8.2 Шкала часу

Шкала часу дозволяє користувачу відображати вже зчитані дані, не зупиняючи процес зчитування поточних. Основною проблемою при розробці шкали часу була повільна робота графічної бібліотеки JavaFX при великій кількості даних для обробки. Тобто, на приклад, маємо масив з 1000 значень і при цьому поточне значення, що відображається є останнє в масиві, а потрібно повернутися до першого. Якщо у циклі посилати кожне значення від 999 до 1 графічна система буде довго обробляти дані. Для вирішення проблеми використано акумулювання даних за відповідний період (від 999 до 1), що виключає попередні значення того ж самого пристрою, а на вихід графічної системи потрапляє тільки фінальний результат.

2.8.3 Файли ресурсів

Окрім файлів FXML, які потрібні для розмітки об’єктів інтерфейсу, було використано каскадні таблиці стилів (CSS) для налаштування стилю графічного об’єкту. На приклад, для налаштування стилю списку підключених пристроїв використано наступний код:

```
#chartLegendVbox{
    -fx-background-color: rgba(255, 255, 255, 0.6);
    -fx-background-radius: 5;
}
```

#chartLegendVbox – звернення до графічного об’єкту за його ідентифікатором;

-fx-background-color – задання кольору фону у форматі rgba;

-fx-background-radius – задання округлення кутів фону.

2.9 Допоміжні класи

В програмі використовуються спеціальні класи, в яких сформовані статичні методи. Ці методи використовуються у програмі у різних місцях. Класи можна знайти у пакеті com.rasalhague.mdrv.Utility. Наприклад, наступна функція повертає псевдовипадкове число типу int з заданого діапазону:

```
public static int randInt(int min, int max)
```

Для приведення PID чи VID пристрою до чотирьох символів використовується функція

```
public static String normalizePidVidToLength(String str)
```

Щоб виклику Runnable в JFX Thread і чекати доки він не завершиться можна використати функцію

```
public static void runAndWait(final Runnable run) throws
InterruptedException, ExecutionException
```

Для завантаження Native бібліотек використано функцію, яка розпаковує обрану бібліотеку до тимчасової директорії

```
public static void loadLibraryFromJar(String path) throws IOException
```

2.10 Реалізація функції повторного програвання (Replay)

Усі данні з пристрою записуються до файлу, таким чином, щоб їх можна було відтворити. Данні записуються в форматі JSON – це текстовий формат обміну даними. JSON базується на тексті. Формат дозволяє описувати об’єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). В даній програмі пакети серіалізуються у файл.

Бібліотека Gson дозволяє як серіалізувати данні так і десеріалізувати. Приклад роботи бібліотеки:

```
private final Gson gson = new GsonBuilder()
.setPrettyPrinting()
.setExclusionStrategies(new CustomExclusionStrategies()).create();
writer.write(gson.toJson(dataPacket));
```

Додаток АУ першій строчці викликається будівельник об’єкту та методом setPrettyPrinting встановлюється режим форматування виводу; у робочій версії програми цей режим вимкнено для економії місця у файлі. Функцією setExclusionStrategies вказується користувацький клас, який вказує які поля треба пропустити при обробці об’єкту. Далі викликається функція gson.toJson(dataPacket), яка повертає об’єкт String, який записують в файл за допомогою writer.write(). Приклад серіалізованих даних можна побачити у

Приклад серіалізованих даних. Для десеріалізації використовується наступних код:

```
JsonReader jsonReader = new JsonReader(fileReader);
Type type = new TypeToken<ArrayList<DataPacket>>() {}.getType();
gson.fromJson(jsonReader, type);
```

Створюємо об’єкт JsonReader, оголошуємо тип, яким було записано у файл дані та десеріалізуємо функцією gson.fromJson().

Основною проблемою функції Replay є пропорційне зростання часу на відкриття файлу відповідно до його розміру. Ця проблема не вирішена у поточній версії програми. Можливий шлях вирішення проблеми – відкриття (зчитування та десеріалізація) не цілого файлу, а його частини за допомогою потокового (Streaming) зчитування.

2.11 Висновки до другого розділу

У другому розділі було розроблено загальну схему програми. Також було розглянуто:

– використані шаблони проектування (Observer, Singleton, Strategy);
– абстрактний клас Device та його реалізації, а також шаблон для нових пристроїв;
– роботу з пристроями (пошук, взаємодія);

– аналіз даних та алгоритми оптимізації розрахунку середнього значення, моді та медіани;
– реєстрація системних повідомлень;
– використання вбудованого Wi-Fi адаптера для сканування каналів та відображення поточного статусу завантаженості;
– можливості графічного інтерфейсу;
– функцію повторного програвання.

3 РОБОТА З АНАЛІЗАТОРАМИ СПЕКТРУ

3.1 MetaGeek Wi-Spy 2.4i (Gen 1)

MetaGeek Wi-Spy Gen 1 (див. рисунок 3.1), як і вся лінійка пристроїв MetaGeek Wi-Spy підключається до операційної системи як HID.

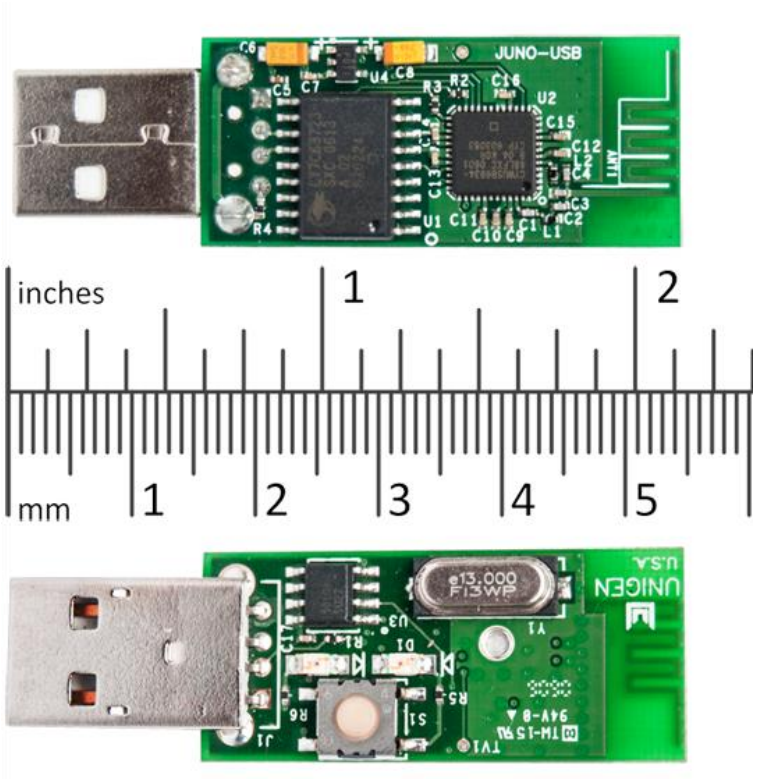


Рисунок 3.1. MetaGeek Wi-Spy Gen 1

Підключенні даного пристрою до програмного комплексу MDRV за допомогою стандартної бібліотеки JavaHIDAPI не вдалося ні під Linux ні під Windows. Ось так виглядає виключення:

```
java.io.IOException: The supplied user buffer is not valid for the
requested operation.
   at com.codeminders.hidapi.HIDDevice.read(Native Method)
```

З коду помилки видно, що функція читання, яка являє собою Native Method, вважає переданий їй буфер, в який треба зчитати данні, не дійснім.

Далі наведено шляхи вирішення проблеми.

3.1.1 Використання високорівневих функцій бібліотеки usb4java

У різних бібліотеках реалізації можуть відрізнятися, тому було вирішено спробувати бібліотеку usb4java. Але при спробі виконати читання даних:

```
int received = pipe.syncSubmit(data);
```

програма зависає.

При спробі відіслати на пристрій IRP – структуру даних ядра Windows, яка забезпечує обмін даними між програмою та драйвером, а також між драйвером та драйвером:

```
 irp = device.createUsbControlIrp((byte) (UsbConst.REQUESTTYPE_DIRECTION_IN
 |
 UsbConst.REQUESTTYPE_TYPE_STANDARD |
 UsbConst.REQUESTTYPE_RECIPIENT_DEVICE),
 UsbConst.REQUEST_CLEAR_FEATURE, (short) 8, (short) 8);
 irp.setData(data);
device.syncSubmit(irp);
```

виникає виключення:

```
javax.usb.UsbPlatformException: USB error 2: Unable to submit control
message: Invalid parameter
```

Ситуація схожа з кореневою проблемою з буфером обміну.

3.1.2 Використання низькорівневих функцій бібліотеки usb4java

При використанні низькорівневих функцій бібліотеки usb4java ситуація не змінилась: проблема з буфером обміну.

3.1.3 Використання JNI

Java Native Interface (див. рисунок 3.2) – стандартний механізм для запуску коду, під керуванням віртуальної машини Java (JVM), який написаний на мовах C / C ++ чи Ассемблера, і скомпонований у вигляді динамічних бібліотек, дозволяє

не використовувати статичне зв’язування. Це дає можливість виклику функції C / C ++ з програми на Java, і навпаки.

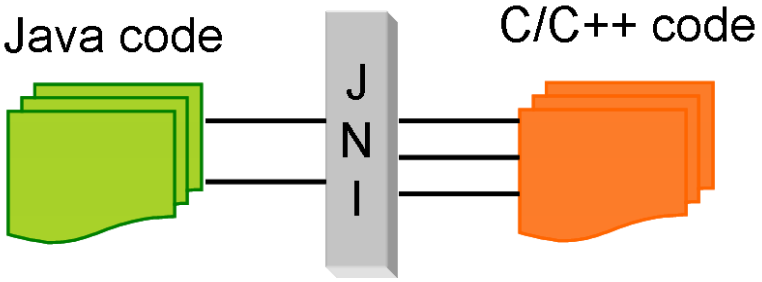


Рисунок 3.2. Схематичне зображення механізму роботи JNI

Основуючись на програмі Kismet Spectools, яка працювала з даним пристроєм, для вирішення проблеми було вирішено зробити JNI до C++. Ключовим фактором для роботи пристрою став виклик функції бібліотеки libusb

```
int libusb_detach_kernel_driver (libusb_device_handle *dev, int
interface_number)
```

яка працює тільки для ОС Linux. Вона виконує від’єднання драйверу ядра внаслідок чого можна зробити захват інтерфейсу:

```
libusb_detach_kernel_driver(dev_handle, 0)
libusb_claim_interface(dev_handle, 0);
```

Після цього можна зчитати данні з пристрою:

```
libusb_control_transfer(dev_handle, USB_ENDPOINT_IN + USB_TYPE_CLASS +
USB_RECIP_INTERFACE, HID_GET_REPORT, (HID_RT_FEATURE << 8), 0, buf,
buf_size, TIMEOUT);
```

3.1.4 Розбір даних з пристрою

Пристрій повертає пакети довжиною в 8 символів:

```
[0, 4, 2, 2, 1, 2, 1, 1]
[7, 2, 3, 7, 0, 0, 2, 0]
[14, 1, 2, 0, 1, 2, 2, 3]
[21, 1, 2, 1, 3, 2, 2, 1]
[28, 1, 3, 3, 2, 0, 0, 2]
```

```
[35, 1, 0, 1, 1, 1, 3, 1]
[42, 2, 1, 2, 2, 1, 1, 3]
[49, 2, 2, 1, 16, 31, 31, 16]
[56, 2, 1, 1, 3, 1, 2, 1]
[63, 3, 1, 3, 2, 1, 2, 3]
[70, 3, 1, 1, 2, 2, 1, 2]
[77, 1, 2, 0, 2, 2, 1, 0]
```

Перший елемент масиву можна інтерпретувати як кількість вже переданих значень RSSI, чи як порядковий номер наступного значення RSSI. Від другого до восьмого включно – значення RSSI.

Після того, як пристрій пробігає по всьому спектру, данні приводяться до чисельного типу та заносяться у масив, до якого, у кінець, підмішується символ кінця пакету, заданий константою END_PACKET_SEQUENCE:

```
END_PACKET_SEQUENCE = new byte[]{-1};
[4, 2, 2, 1, 2, 1, 1, 2, 3, 7, 0, 0, 2, 0, 1, 2, 0, 1, 2, 2, 3, 1, 2, 1, 3,
2, 2, 1, 1, 3, 3, 2, 0, 0, 2, 1, 0, 1, 1, 1, 3, 1, 2, 1, 2, 2, 1, 1, 3, 2,
2, 1, 16, 31, 31, 16, 2, 1, 1, 3, 1, 2, 1, 3, 1, 3, 2, 1, 2, 3, 3, 1, 1, 2,
2, 1, 2, 1, 2, 0, 2, 2, 1, 0, -1]
```

3.2 MetaGeek Wi-Spy 2.4x2

На відміну від MetaGeek Wi-Spy Gen 1, Wi-Spy 2.4x (див. рисунок 3.3) оснащений знімною зовнішньої антеною, яка може бути замінена на спрямовану або більш потужну.

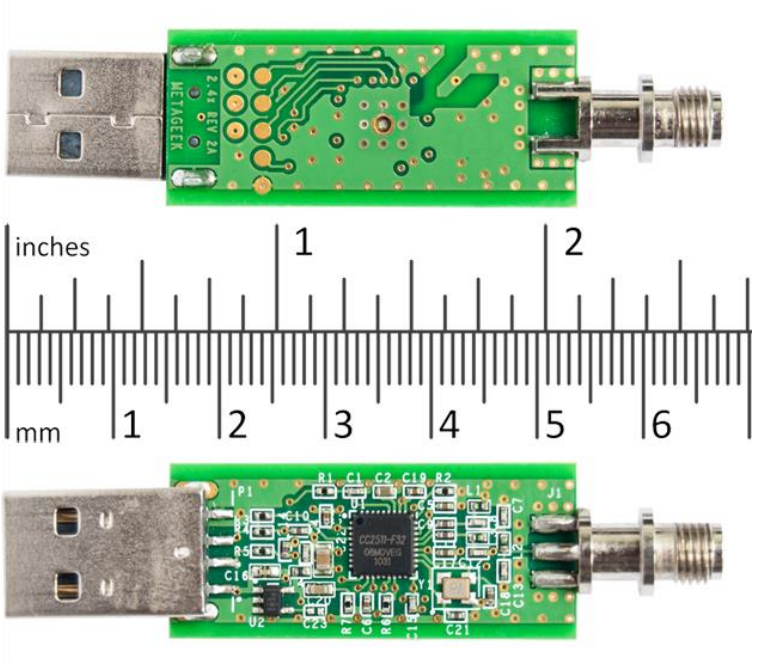
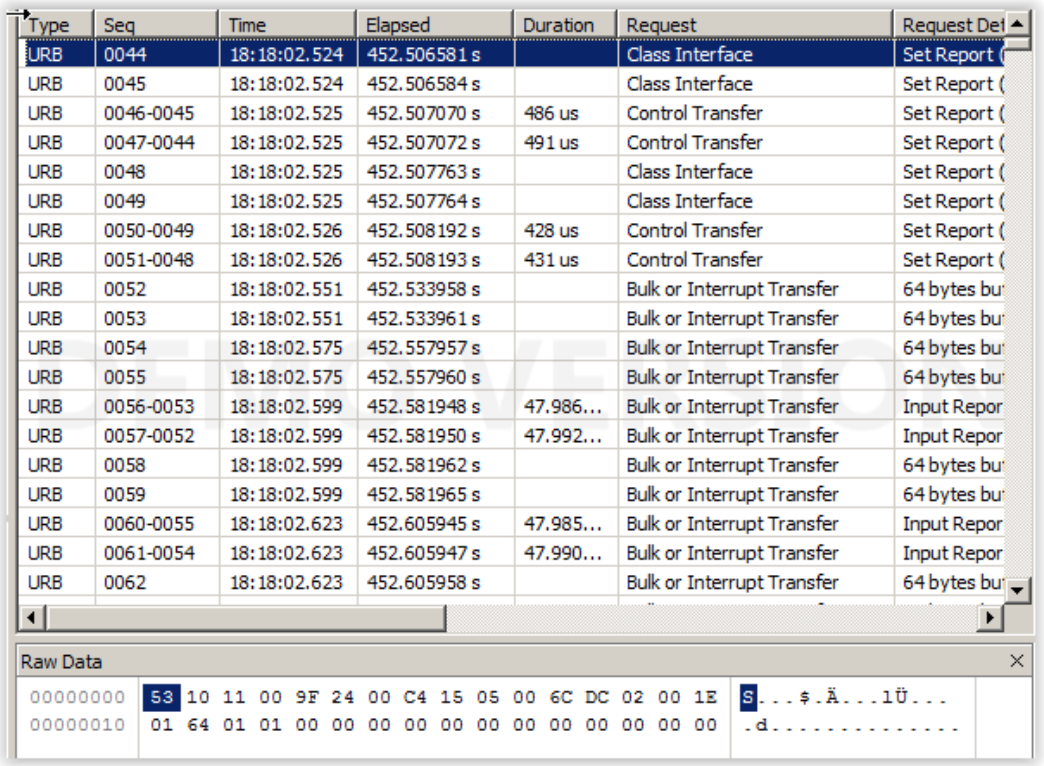


Рисунок 3.3. MetaGeek Wi-Spy 2.4x

3.2.1 Ініціалізація

Для того, щоб ініціалізувати пристрій потрібно знати спеціальну послідовність ініціалізації, яку потрібно передати пристрою. Послідовність ініціалізації представляє собою набір команд, які пристрій здатний сприймати. У даному випадку офіційної документації немає, тому використаємо сніффер даних USB порту USBlyzer у поєднанні з програмою MetaGeek Chanalyzer, у якій вже реалізовано ініціалізацію пристрою (див. рисунок 3.4).



Type	Seq	Time	Elapsed	Duration	Request	Request Det
URB	0044	18:18:02.524	452.506581 s		Class Interface	Set Report (
URB	0045	18:18:02.524	452.506584 s		Class Interface	Set Report (
URB	0046-0045	18:18:02.525	452.507070 s	486 us	Control Transfer	Set Report (
URB	0047-0044	18:18:02.525	452.507072 s	491 us	Control Transfer	Set Report (
URB	0048	18:18:02.525	452.507763 s		Class Interface	Set Report (
URB	0049	18:18:02.525	452.507764 s		Class Interface	Set Report (
URB	0050-0049	18:18:02.526	452.508192 s	428 us	Control Transfer	Set Report (
URB	0051-0048	18:18:02.526	452.508193 s	431 us	Control Transfer	Set Report (
URB	0052	18:18:02.551	452.533958 s		Bulk or Interrupt Transfer	64 bytes bu
URB	0053	18:18:02.551	452.533961 s		Bulk or Interrupt Transfer	64 bytes bu
URB	0054	18:18:02.575	452.557957 s		Bulk or Interrupt Transfer	64 bytes bu
URB	0055	18:18:02.575	452.557960 s		Bulk or Interrupt Transfer	64 bytes bu
URB	0056-0053	18:18:02.599	452.581948 s	47.986...	Bulk or Interrupt Transfer	Input Repor
URB	0057-0052	18:18:02.599	452.581950 s	47.992...	Bulk or Interrupt Transfer	Input Repor
URB	0058	18:18:02.599	452.581962 s		Bulk or Interrupt Transfer	64 bytes bu
URB	0059	18:18:02.599	452.581965 s		Bulk or Interrupt Transfer	64 bytes bu
URB	0060-0055	18:18:02.623	452.605945 s	47.985...	Bulk or Interrupt Transfer	Input Repor
URB	0061-0054	18:18:02.623	452.605947 s	47.990...	Bulk or Interrupt Transfer	Input Repor
URB	0062	18:18:02.623	452.605958 s		Bulk or Interrupt Transfer	64 bytes bu

Raw Data

```
00000000  53 10 11 00 9F 24 00 C4 15 05 00 6C DC 02 00 1E  S...$.Ä...lÜ...
00000010  01 64 01 01 00 00 00 00 00 00 00 00 00 00 00 00  .d..............
```

Рисунок 3.4. Сніффінг даних USB порту для визначення послідовності ініціалізації

При спробі відправити команду ініціалізації до пристрою під ОС Windows виникає наступне виключення:

```
java.io.IOException: The parameter is incorrect.
   at com.codeminders.hidapi.HIDDevice.write(Native Method)
```

яке говорить про те, що в Native Method був переданий не коректний параметр. Треба відмітити, що той самий програмний код коректно працює під Linux та не працює під Windows. У подальшому можлива спроба виправити це за допомогою JNI, а поки що цей пристрій підтримується тільки на Linux.

3.2.2 Розбір даних з пристрою

За одну передачу пристрій видає наступну послідовність:

```
[74, 0, 0, 0, 0, 33, 67, 67, 69, 66, 68, 65, 67, 66, 66, 64, 67, 67, 67,
67, 67, 67, 70, 62, 68, 68, 64, 66, 68, 68, 67, 68, 67, 68, 64, 67, 68, 68,
65, 67, 66, 68, 69, 65, 66, 66, 68, 68, 65, 69, 67, 66, 67, 66, 67, 68, 66,
66, 66, 68, 67, 67, 68, 65, 74, 59]
```

Структура пакету схожа на MetaGeek Wi-Spy 2.4i Gen 1. Перший елемент масиву можна інтерпретувати як кількість вже переданих значень RSSI, чи як порядковий номер наступного пакету значень RSSI. Від другого до п’ятого включно – сервісна інформація. Всі інші – значення RSSI зі здвигом на 170 одиниць.

4 TEXAS INSTRUMENTS EZ430-RF2500

Texas Instruments ez430-RF2500 (див. рисунок 4.1) перед застосуванням треба було попередньо налаштувати. Проблема була в тому, що пристрій не бачив частину спектру (див. рисунок 4.2–рисунок 4.4).

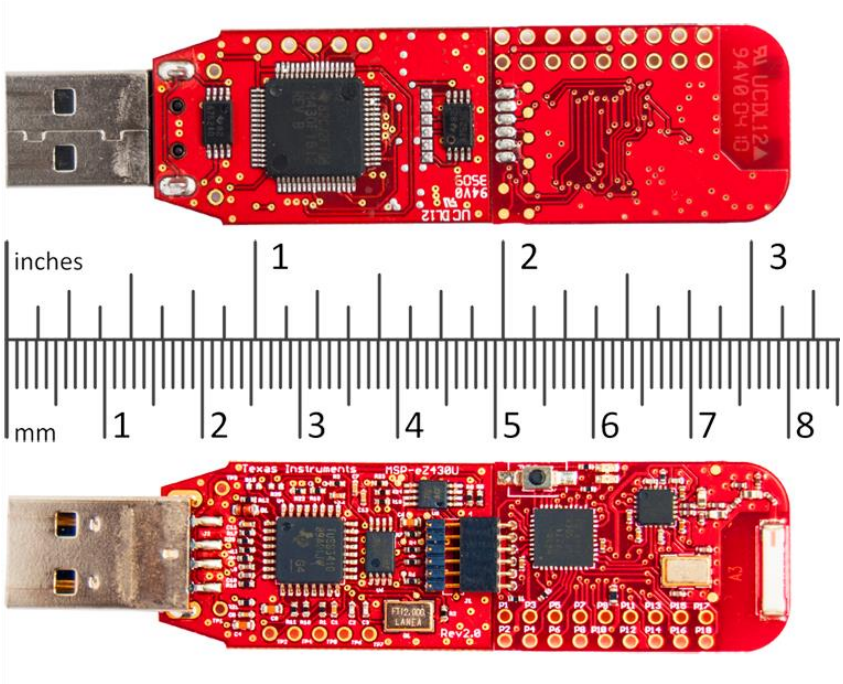


Рисунок 4.1. Texas Instruments ez430-RF2500

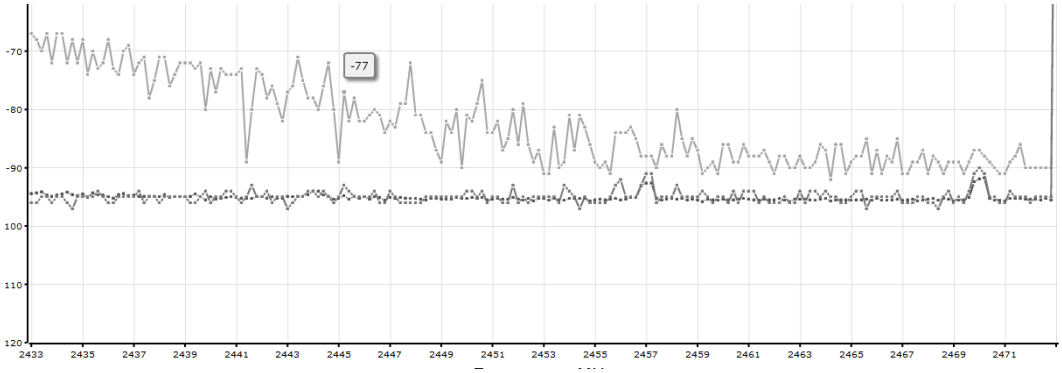


Рисунок 4.2. 1-й канал під загрузкою

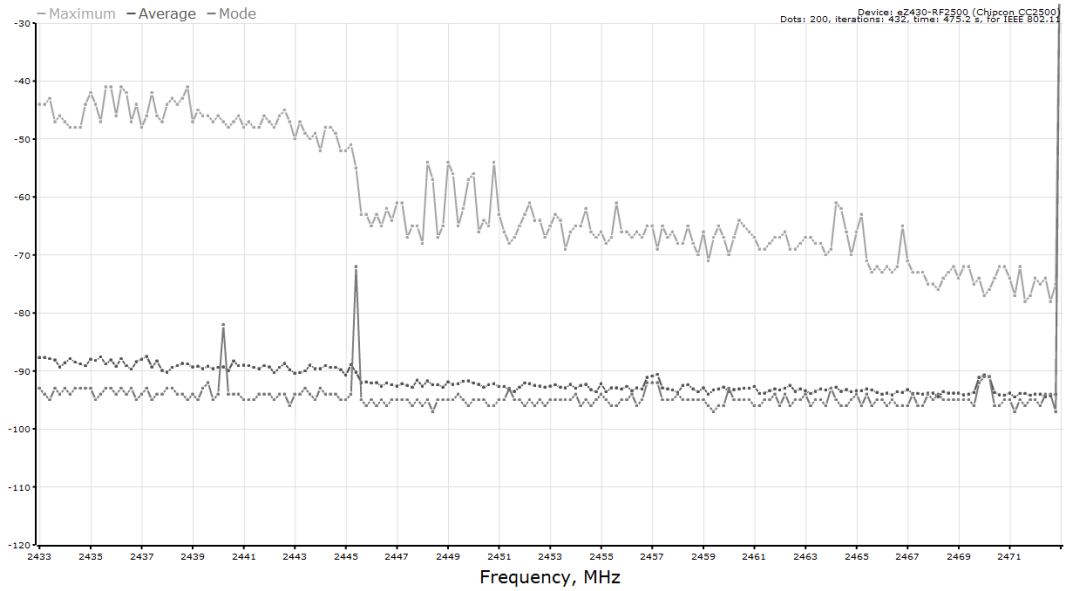


Рисунок 4.3. 6-й канал під загрузкою

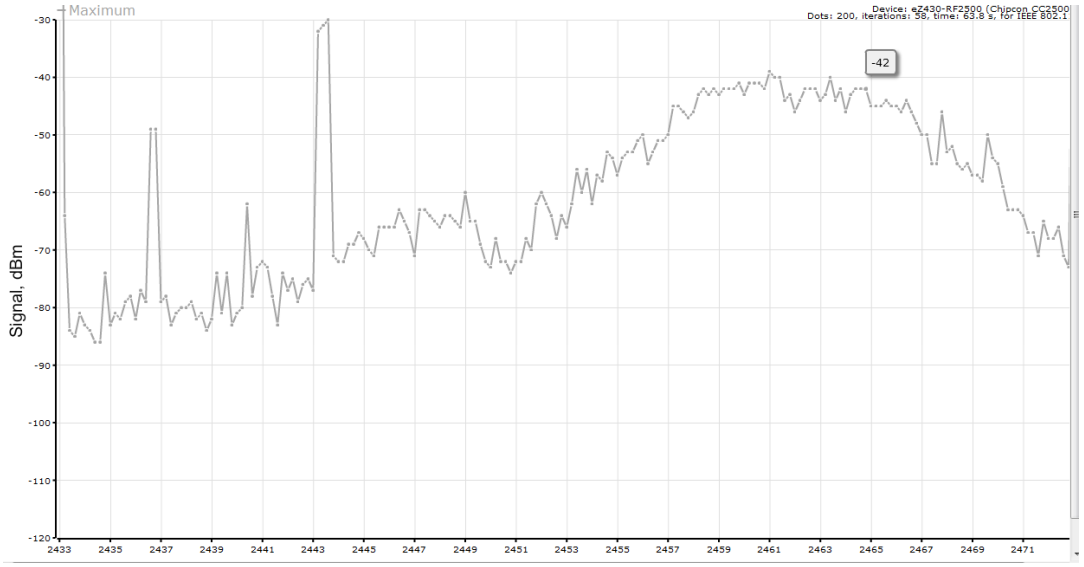


Рисунок 4.4. 11-й канал під загрузкою

З графіків вище видно, що діапазон частот до шостого каналу, тобто частоти 2401–2426 МГц, а також після 11-го каналу, тобто частоти 2473–2483 МГц, випадають із зони видимості пристрою.

Для вирішення даної проблеми треба:

1. Визначити, які регістри відповідають за потрібні налаштування та встановити коректні параметри.
2. Прошити пристрій виправленим кодом.
3. Провести тестування.

У ході роботи над даним пристроєм було використано програмне забезпечення SmartRF Studio, та програмний код із статті Creating a Spectrum Analyzer to Measure Noise. Для компіляції програмного коду та відправки його до пам’яті пристрою було використано IDE IAR Embedded Workbench.

4.1.1 Пошук потрібних регістрів та встановлення коректних параметрів

У заготовочному файлі source_code\drivers\mrfi\smartrf\CC2500\smartrf _CC2500.h було знайдено об’явлення регістрів. Нижче приведені регістри, які відповідають налаштуванню Base Frequency:

```
#define SMARTRF_SETTING_FREQ2 0x5D
#define SMARTRF_SETTING_FREQ1 0x93
#define SMARTRF_SETTING_FREQ0 0xB1
```

Для перевірки введемо ці значення до SmartRF Studio.

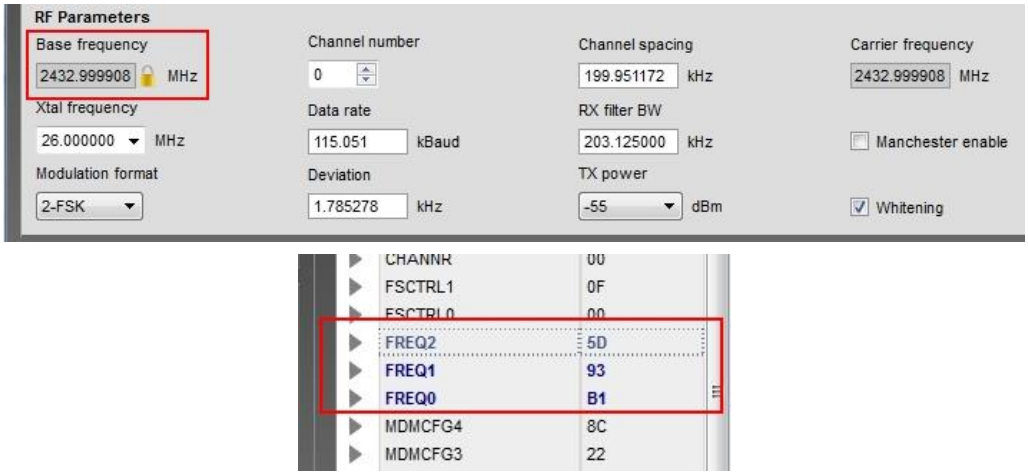


Рисунок 4.5. Перевірка значень регістрів у SmartRF Studio

Рисунок 4.5 вказує, що при таких значеннях регістрів Base Frequency = 2433, що підтверджує проблему. Підставимо коректні значення Base Frequency (див рисунок 4.6).

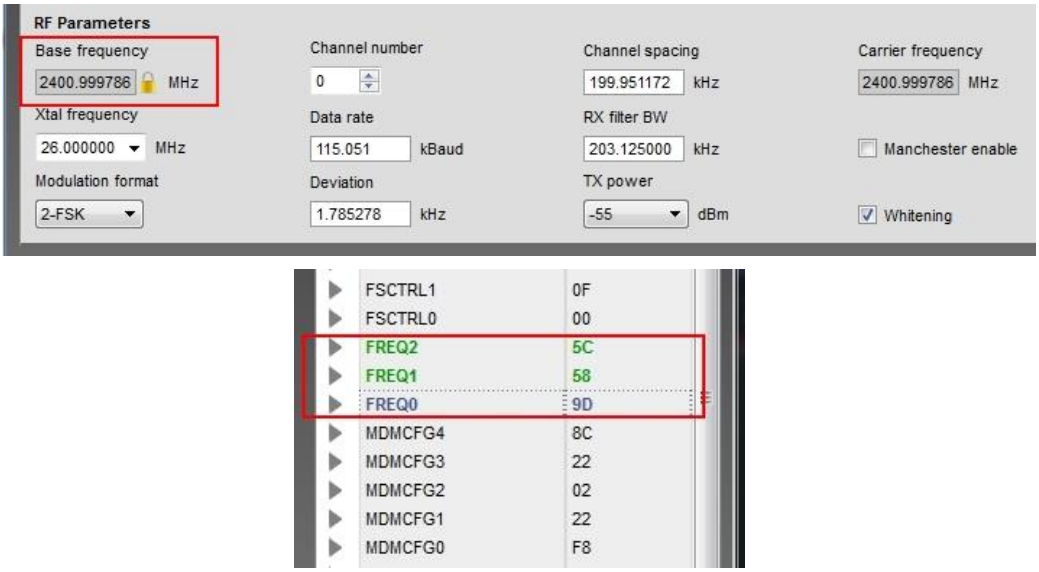


Рисунок 4.6. Коректні значення регістрів для Base Frequency

У коді це буде виглядати наступним чином:

```
#define SMARTRF_SETTING_FREQ2 0x5C //old val 0x5D
#define SMARTRF_SETTING_FREQ1 0x58 //old val 0x93
#define SMARTRF_SETTING_FREQ0 0x9B //old val 0xB1
```

4.1.2 Тестування змін

Для тестування змін використовується наступний код:

```
void print_rssi(int8_t rssi)
{
  char output[] = {" 000 "};
  if (rssi<0) {output[0]='-';rssi=-rssi;}
  output[1] = '0'+((rssi/100)%10);
  output[2] = '0'+((rssi/10)%10);
  output[3] = '0'+ (rssi%10);
  TXString(output, (sizeof output)-1);
}
int main(void)
{
  int8_t rssi;
  uint8_t channel;
  BSP_Init();
  MRFI_Init();
  P3SEL    |= 0x30;
  UCA0CTL1  = UCSSEL_2;
  UCA0BR0   = 0x41;
```

```
  UCA0BR1   = 0x3;
  UCA0MCTL  = UCBRS_2;
  UCA0CTL1 &= ~UCSWRST;
  MRFI_WakeUp();
  __bis_SR_register(GIE);
  while(1) {
    for (channel=0;channel<200;channel++) {
      MRFI_RxIdle();
      mrfiSpiWriteReg(CHANNR,channel);
      MRFI_RxOn();
      rssi=MRFI_Rssi();
      print_rssi(rssi);
    }
    TXString("\n",1);
  }
}
```

Функція print_rssi генерує масив зі зчитаними даними та за допомогою функції TXString відправляє їх на порт, де їх приймає дане програмне забезпечення. У точці входу, функції main, знаходиться код ініціалізації пристрою (функції BSP_Init, MRFI_Init та MRFI_WakeUp), та цикл читання RSSI:

– MRFI_RxIdle() – переключення пристрою в режим простою;
– mrfiSpiWriteReg(CHANNR,channel) – запис до регістру CHANNR номер каналу для переключення;
– MRFI_RxOn() – переключення пристрою в режим прийняття даних;
– MRFI_Rssi() – повертає поточне значення RSSI;
– TXString("\n",1) – відсилання символ кінця строки, який вказує на завершення циклу [6].

Використовуючи нові значення регістрів отримуємо результат – рисунок 4.7 засвідчує коректність нових значень Base Frequency.

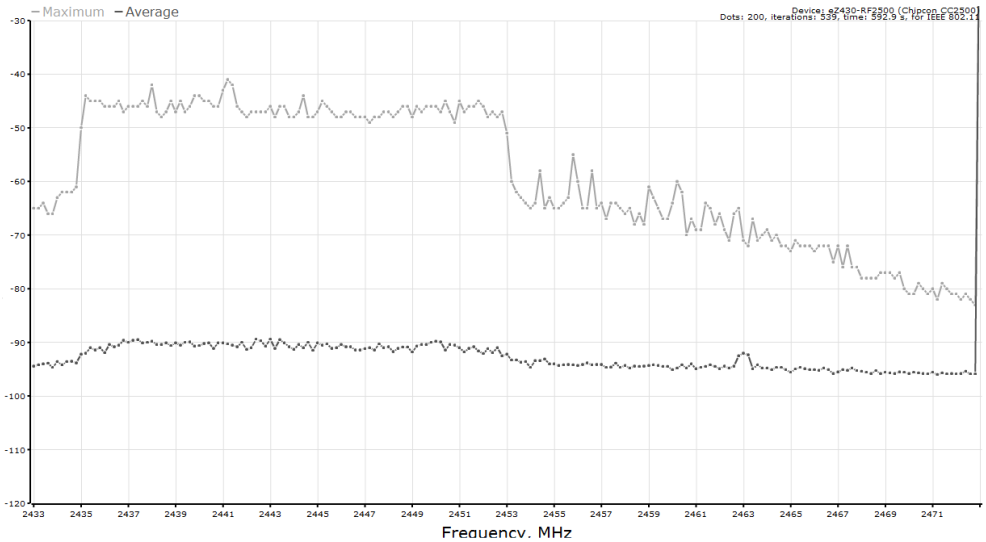


Рисунок 4.7. Тестування скорегованого Base Frequency

Далі скорегуємо значення регістрів Channel Spacing, який відповідає за відстань між дискетами так, щоб Carrier frequency був близьким до максимального для даного пристрою (2484 МГц). Для цього використаємо поле Channel number та Carrier frequency (див. рисунок 4.8).

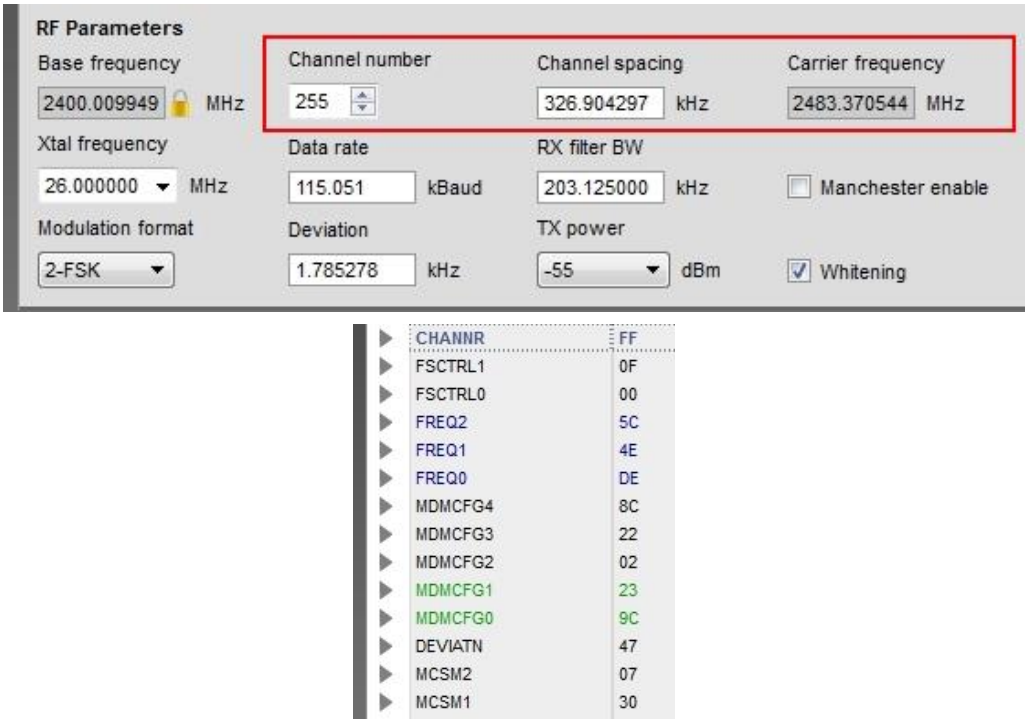


Рисунок 4.8. Корегування значень регістру Channel Spacing

Рисунок 4.9 засвідчує коректність змін Channel Spacing – пристрій працює у повному доступному діапазону частот з мінімальним доступним Channel Spacing та з максимальними доступним обсягом каналів (255).

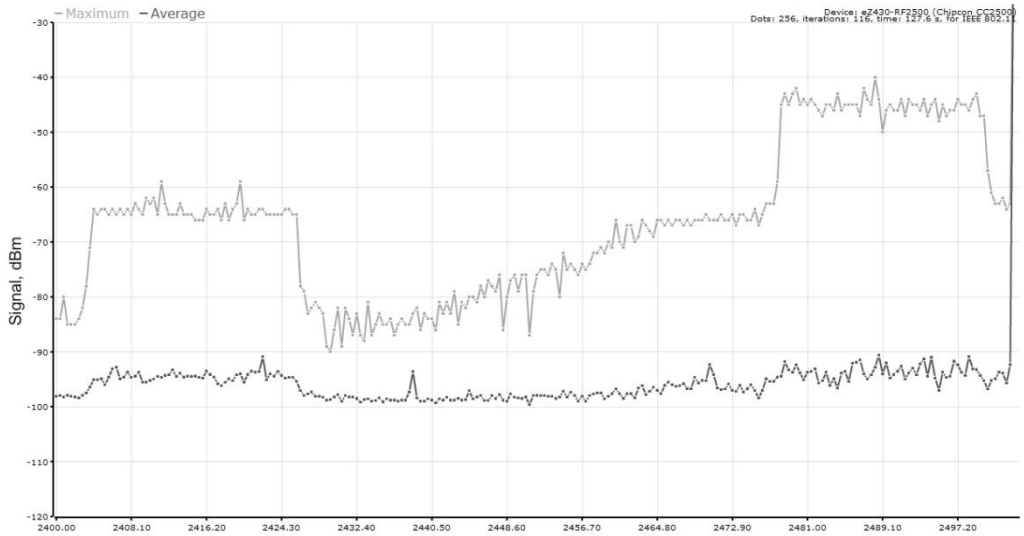


Рисунок 4.9. Тестування змін Channel Spacing

4.1.3 Overclocking

Оскільки досягнуто максимум дискрет при мінімальній відстані між ними, то було вирішено, що можна зчитувати дані по блоках частоти: виставити мінімальне значення Channel Spacing і, на льоту змінюючи частоту, прослухати по черзі кожен з діапазонів частот (на приклад 2400–2410, 2411–2421, 2422–2433 і так далі) і відсилати ці значення на порт одним великим рядком, ніби це була одна ітерація зчитування.

З документації по CC2500 маємо: при зміні регістрів, які відповідають за частоту, при запущеному частотному генераторі, можуть статися непередбачувані речі. Таким чином, програмування частоти повинно виконуватись тільки коли генератор знаходиться в режимі очікування [7].

Спробуємо розділити діапазон частот на 2 частини – 2401–2440 та 2440–2478 Основний цикл зчитування змінено відповідно:

```
while(1) {

    //change to 1th range of base frq
    MRFI_RxIdle();
```

```
    mrfiSpiWriteReg(FREQ0,0x9D);
    mrfiSpiWriteReg(FREQ1,0x58);
    mrfiSpiWriteReg(FREQ2,0x5C);

    for (channel=0;channel<255;channel++) {
      MRFI_RxIdle();
      mrfiSpiWriteReg(CHANNR,channel);
      MRFI_RxOn();
      rssi=MRFI_Rssi();
      print_rssi(rssi);
    }

    //change to 2th range of base frq
    MRFI_RxIdle();
    mrfiSpiWriteReg(FREQ0,0x9D);
    mrfiSpiWriteReg(FREQ1,0xD8);
    mrfiSpiWriteReg(FREQ2,0x5D);

    for (channel=0;channel<255;channel++) {
      MRFI_RxIdle();
      mrfiSpiWriteReg(CHANNR,channel);
      MRFI_RxOn();
      rssi=MRFI_Rssi();
      print_rssi(rssi);
    }

    TXString("\n",1);
  }
```

Отримаємо (див. рисунок 4.10) 510 значень RSSI за одне проходження повного діапазону частот.

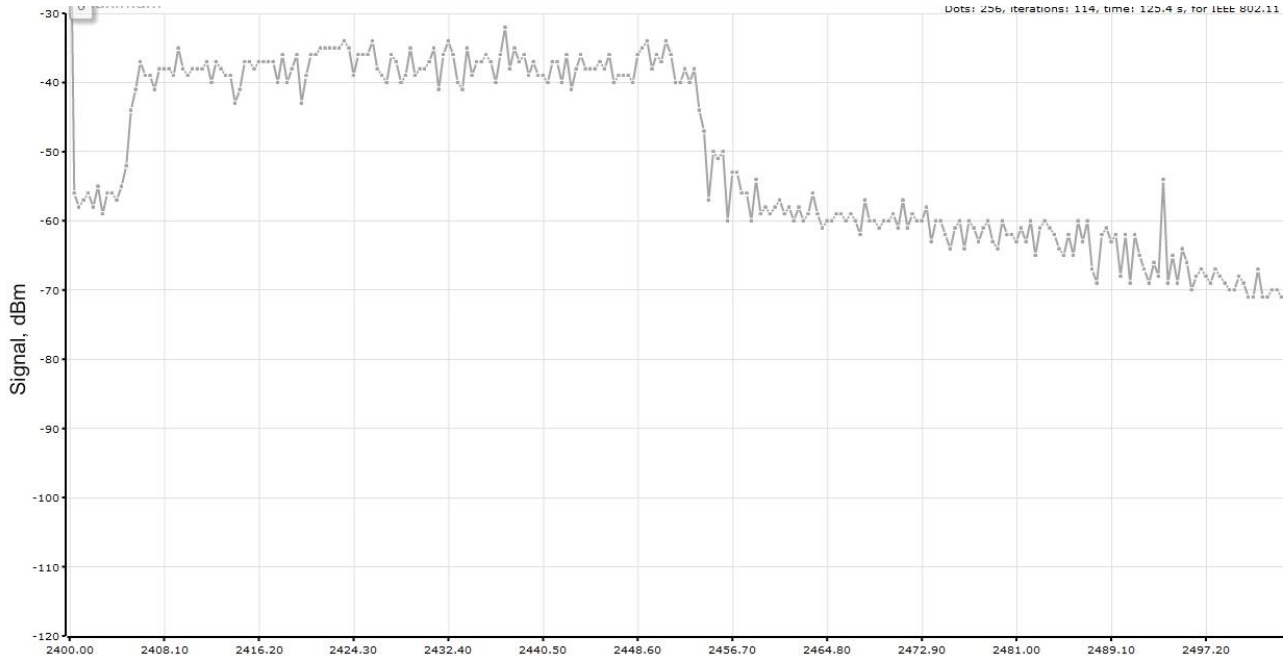


Рисунок 4.10. 510 значень RSSI

Отже, тепер можна задати мінімальне значення Channel Spacing (25.390625) і розбити діапазон 2,4–2,5 на ще більше під-діапазонів (8) і отримати максимальну частоту дискрет (2040), але при цьому у скільки разів більше піддіапазонів в стільки ж разів потрібно часу на зчитування (див. рисунок 4.11).

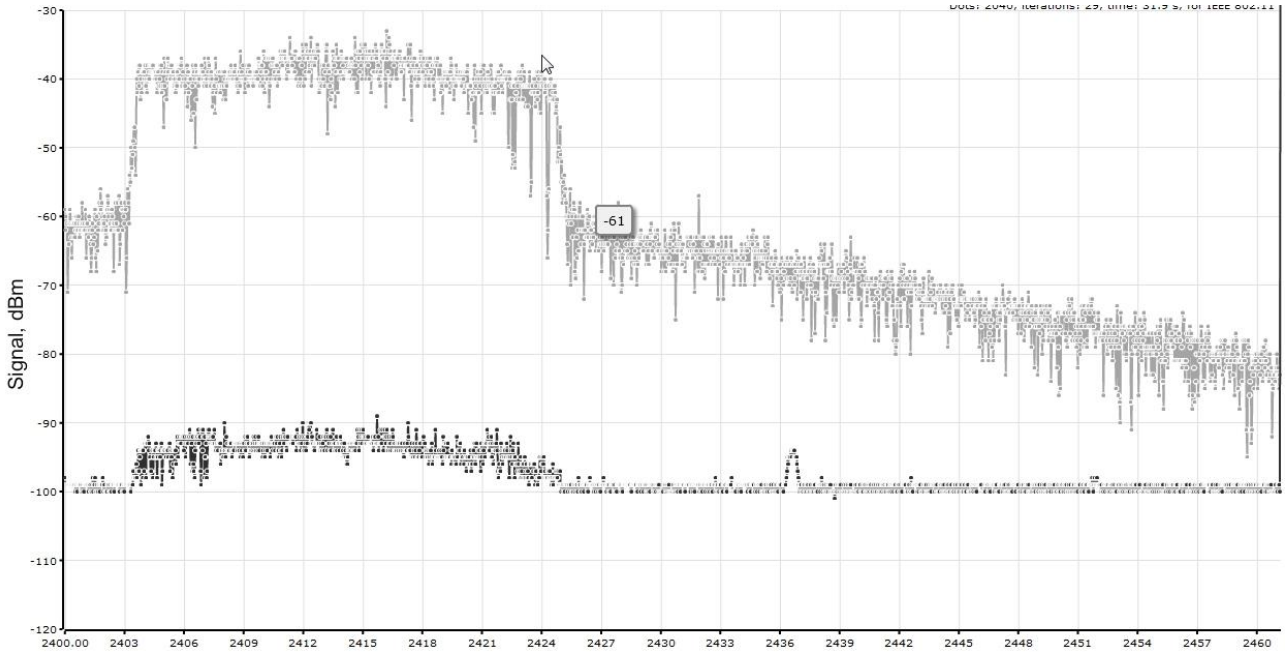


Рисунок 4.11. 2040 значень RSSI

4.1.4 Підключення до MDRV

Пристрій визначається системою як COM. Формат пакету дуже простий: спочатку йдуть значення RSSI, а у кінці – символ кінця строки (‘\n’). Пристрій ініціалізується самостійно при підключенні до комп’ютера.

4.2 Ubiquiti AirView2

Ubiquiti AirView2 (див. рисунок 4.12) має частотний діапазон 2399–2485 МГц.

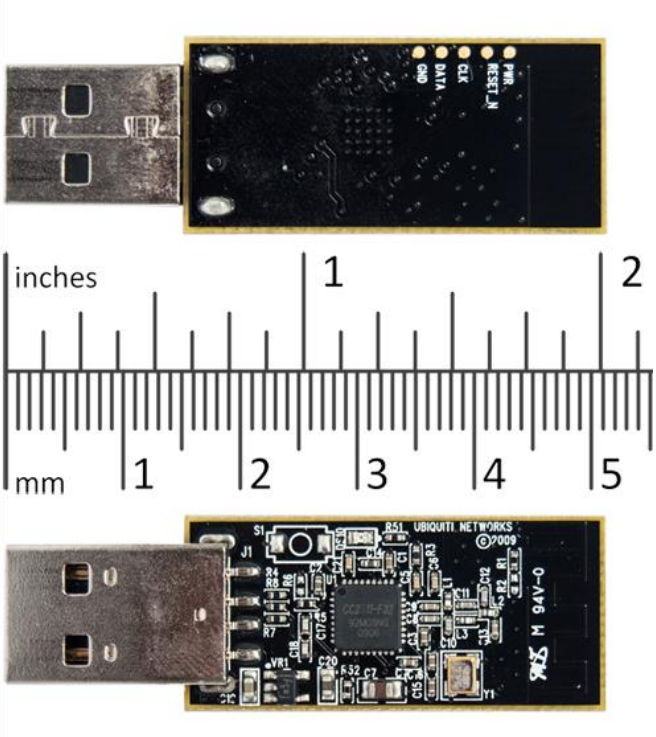


Рисунок 4.12. Ubiquiti AirView2

4.2.1 Ініціалізація

Визначається як COM пристрій і ініціалізується за допомогою передачі на нього спеціальних послідовностей, які приведені нижче:

```
byte[] intByte = new byte[]{0x69, 0x6E, 0x74}; //int
byte[] bsByte = new byte[]{0x0A, 0x62, 0x73, 0x0A}; //.bs.
```

Відповідно до ASCII перша послідовність означає “int”, тобто “initialize”. Друга послідовність “.bs.” (begin scan).

4.2.2 Розбір даних з пристрою

Пристрій визначається системою як COM. Формат пакету дуже простий: спочатку йдуть значення RSSI, а у кінці – символ кінця строки (‘\n’).

4.3 Unigen ISM Sniffer (Wi-detector)

Визначається як USBHID. Пристрій (див. рисунок 4.13) ініціалізується та починає передавати значення RSSI самостійно, при підключенні до комп’ютера. Версія пристрою – 2.0.

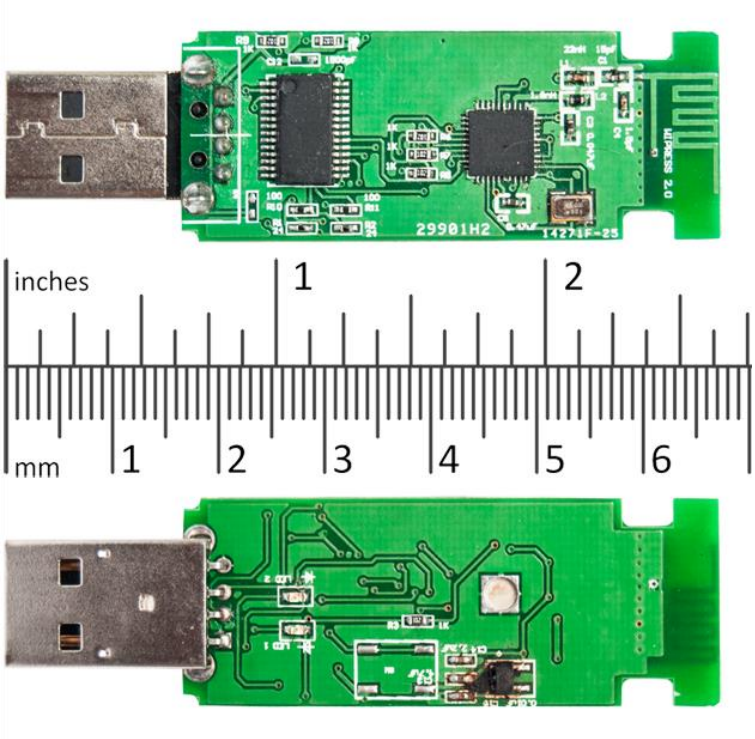


Рисунок 4.13. Unigen ISM Sniffer (Wi-detector)

Значення RSSI потрібно корегувати за наступною формулою:

```
(((aByte - 135) + 100) * 1.428) – 100 ,
```

де aByte – передане пристроєм значення.

4.4 Pololu Wixel

Визначається як COM. Пристрій (див. рисунок 4.14) ініціалізується та починає передавати значення RSSI самостійно, при підключенні до комп'ютера. Пакет, як і у Ubiquiti AirView2 та TI ez430-RF2500 спочатку значення RSSI, а у кінці – символ кінця строки ('\n').

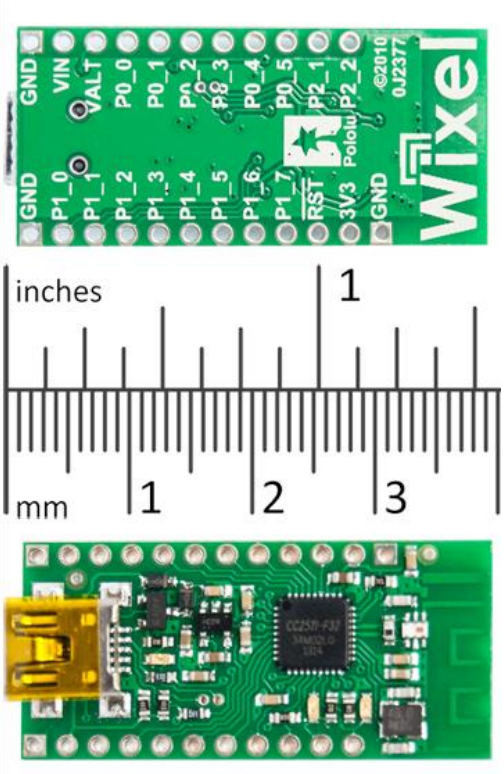


Рисунок 4.14. Pololu Wixel

4.4.1 Прошивка

Можна розробляти програми під Wixel за допомогою будь-якого текстового редактора. В даному разі пристрій прошивався через Eclipse (див. рисунок 4.15), використовуючи Wixel SDK.

З коду прошивки (див. додаток И) можна виділи наступні функції:

– putchar(char c) – відсилає символ на порт;

– reportResults() – виводить значення RSSI;

– systemInit(), usbInit(), analyzerInit() – ініціалізація системи;

– checkRadioChannels() – проходить по частотному діапазону заповнюючи масив з RSSI;

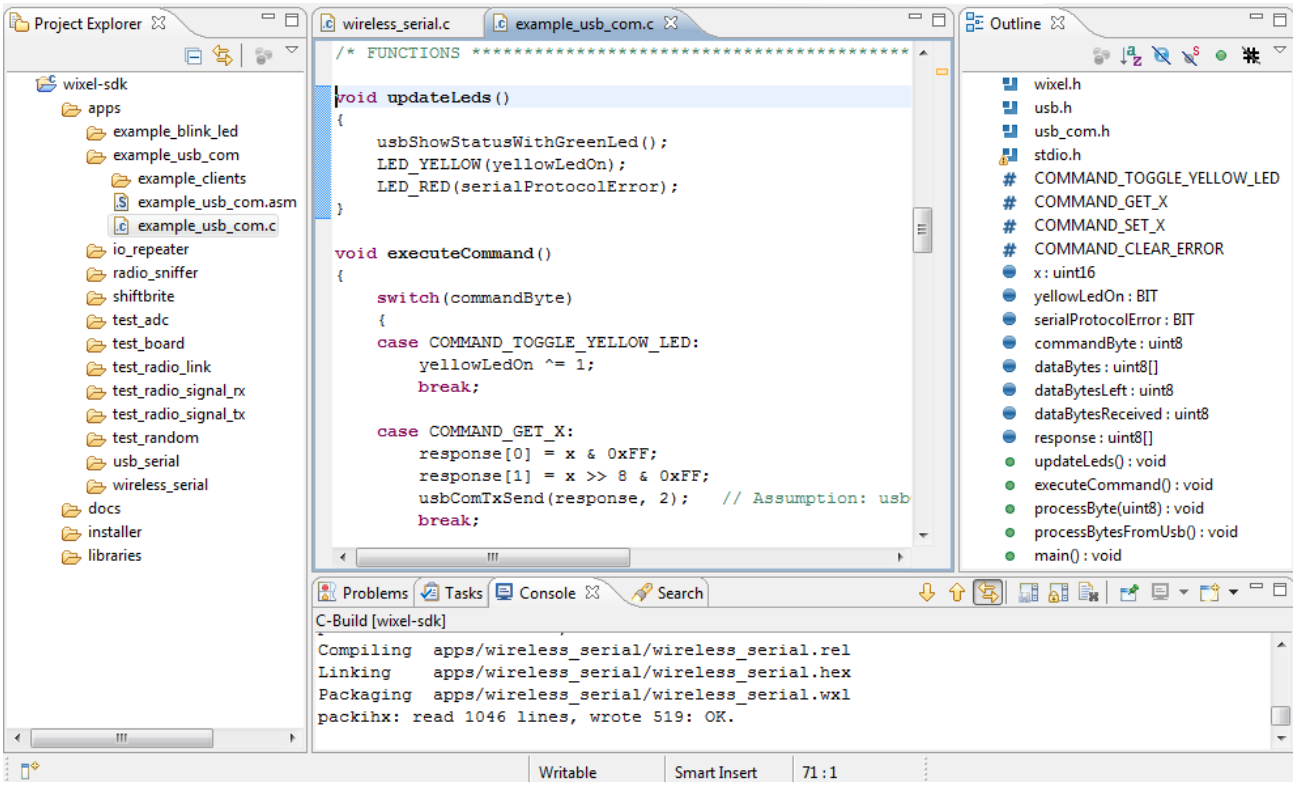


Рисунок 4.15. Eclipse IDE

4.1 Висновки до третього розділу

У другому розділі було описано процес роботи з кожним аналізатором спектру та його процес підключення до програмного комплексу. Були вирішені такі проблеми з Texas Instruments ez430-RF2500, який працював у скороченому діапазоні. Також був розроблений Java Native Interface для підключення MetaGeek Wi-Spy 2.4i (Gen 1).

ВИСНОВКИ

В роботі представлені результати розробки універсального засобу для роботи з аналізаторами спектру бюджетного сегменту від різних виробників. Проаналізовані протоколи обміну між пристроями і програмним забезпеченням, яке з ними поставляється, вибрані найкращі ідеї для реалізації відображення результатів сканування. Вбудована можливість «гарячого» приєднання/від'єднання пристроїв, підключення кількох однакових пристроїв, робота з мережевою карткою.

Дані від різних пристроїв зберігаються в універсальній структурі даних, для яких існує можливість повторного «програвання» і аналізу результатів.

З отриманих даних в режимі реального часу розраховуються середнє значення, мода і медіана, а також отримується максимальне значення. За результатами цих даних існує можливість відстеження стаціонарних і нестаціонарних завад, видів передавальних пристроїв, завантаженість ефіру тощо.

В результаті роботи вирішені проблеми уніфікації роботи з різними пристроями (різні значення частотних відліків), проблема із складністю роботи з великими масивами даних, оптимізація архітектури програми для додавання нових пристроїв, проблема з масштабуванням лінії часу при збереженні даних, проблема вибору чутливості відображення даних з мережної карти, проблеми оптимального відношення «ширина діапазону – кількість точок вимірювання – швидкість сканування» при прошивці деяких аналізаторів спектру.

В якості середовища розробки вибране Intellij IDEA і JavaFX Scene Builder, для кроспратформеності додатка використовується мова програмування Java, але в результаті не вийшло на ОС Windows задіяти деякі пристрої (через закритий сирцевий код драйверів) і мережеву картку.

Процес розробки програмного забезпечення відповідає основним принципам шаблону проектування Scrum.

Результатом впровадження результатів роботи стало програмне забезпечення MDRV, яке розповсюджується за GNU GPL. Вихідні коди програмного забезпечення MDRV представлені в системі контролю версій GitHub [8].

СПИСОК ЛІТЕРАТУРИ

1. Макконнелл, С. Совершенный код. Мастер-класс // Стив Макконнелл / Пер. с англ. – М.: Русская редакция, 2010. – 896 с.
2. Bloch, J. Effective Java // Joshua Bloch / 2nd edition. – Sun Microsystems, 2008. – 369 p.
3. Мартин, Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста // Роберт Мартин / Пер. с англ. – СПб: Питер, 2010. – 464 с.
4. Сьерра, К. Изучаем Java // Кэти Сьерра, Берт Бейтс / Изд. 2-е / Пер. с англ. – Эксмо, 2005. – 708 р.
5. Гамма, Э. Приемы объектно-ориентированного проектирования: Паттерны проектирования // Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес. – СПб.: Питер, 2010. – 368 с.
6. Watteyne, T. Creating a Spectrum Analyzer to Measure Noise // Thomas Watteyne. – 2009. – 4 p. See more: http://cnx.org/content/m21599/1.3/
7. CC2500 Low-Cost Low-Power 2.4 GHz RF Transceiver. – Dallas: Texas Instruments, 2011. – 96 p.
8. Програмне забезпечення і вихідні коди для аналізу спектру MDRV https://github.com/RasAlhague/MDRV

Додаток А

ЛІСТИНГ КЛАСУ DEVICECONNECTIONLISTENER

Клас розглянуто у пункті 2.4.1.

```
package com.rasalhague.mdrv.connectionlistener;

import com.codeminders.hidapi.HIDDeviceInfo;
import com.codeminders.hidapi.HIDManager;
import com.rasalhague.mdrv.device.core.DeviceInfo;
import com.rasalhague.mdrv.logging.ApplicationLogger;
import jssc.SerialPortList;

import java.io.IOException;
import java.util.*;

/**
 Одинак. Прислухається до нових підключень.
 */
public class DeviceConnectionListener {
  private static final DeviceConnectionListener INSTANCE = new
DeviceConnectionListener();
  static {
    com.codeminders.hidapi.ClassPathLibraryLoader.loadNativeHIDLibrary();
  }
  private final ArrayList<DeviceInfo> dummyDeviceList = new ArrayList<>();
  private final ArrayList<DeviceInfo> connectedDeviceList = new
ArrayList<>();
  private final List<DeviceConnectionListenerI> listeners = new
ArrayList<>();
  private final long scanTimerPeriodMs = 1000;
  private boolean isListening = false;
  private Timer timer;

  private DeviceConnectionListener() { }

  /**
   Повертає INSTANCE.

   @return the INSTANCE
   */
  public static DeviceConnectionListener getInstance() {
    return INSTANCE;
  }

  /**
```

```
   Почати слухання.
   */
  public void startListening() {
    runSchedule();
  }

  /**
   Зупинити слухання.
   */
  public void stopListening() {
    cancelSchedule();
  }

  /**
   Додати тестовий пристрій.
   DummyDevice повинен бути "DummyDevice [SeqNumb]"
   */
  public void addDummyDevice() {
    Random random = new Random();

    dummyDeviceList.add(new DeviceInfo("1111", "1111", "DummyDevice " +
(dummyDeviceList.size() + 1), "DummyPort " + (dummyDeviceList.size() + 1),
DeviceInfo.DeviceType.DUMMY, new byte[]{10}, 2399, 500));
  }

  /**
   Провірка на прослуховування.

   @return the boolean
   */
  public boolean isListening() {
    return isListening;
  }

  /**
   Додати слухача.

   @param toAdd
   the to add
   */
  public void addListener(DeviceConnectionListenerI toAdd) {
    listeners.add(toAdd);
  }

  private void runSchedule() {
    if (!isListening) {
      timer = new Timer();
      TimerTask timerTask = new TimerTask() {
        @Override public void run() {
```

```
          scanForDeviceConnections();
        }
      };

      long timerDelayMs = 0;
      timer.schedule(timerTask, timerDelayMs, scanTimerPeriodMs);
      isListening = true;

      ApplicationLogger.LOGGER.info("Listening schedule has started.
Waiting for devices...");
    }
    else {
      cancelSchedule();
      runSchedule();
    }
  }

  private void cancelSchedule() {
    timer.cancel();
    isListening = false;

    ApplicationLogger.LOGGER.info("Listening schedule has canceled.");
  }

  private void scanForDeviceConnections() {
    ArrayList<DeviceInfo> combinedList = new ArrayList<>();

    combinedList.addAll(getCOMPortsList());
    combinedList.addAll(getHIDDevicesList());
    combinedList.addAll(getDummyDeviceList());

    updateConnectedDeviceList(combinedList);
  }

  /**
   Повертає поточні підключені порти як ArrayList<DeviceInfo>

   @return поточні підключені порти
   */
  private ArrayList<DeviceInfo> getCOMPortsList() {
    // Повертає їмя підключених портів
    String[] portNames = SerialPortList.getPortNames();

    //Генерує масив з portNames
    ArrayList<DeviceInfo> deviceInfoList = new ArrayList<>();

    for (String portName : portNames) {
      deviceInfoList.add(new DeviceInfo(portName));
    }
```

```
    return deviceInfoList;
  }

  private ArrayList<DeviceInfo> getHIDDevicesList() {
    try {
      HIDDeviceInfo[] hidDeviceInfos =
HIDManager.getInstance().listDevices();

      // Генерує масив з portNames
      ArrayList<DeviceInfo> deviceInfoList = new ArrayList<>();

      if (hidDeviceInfos != null) {
        for (HIDDeviceInfo hidDeviceInfo : hidDeviceInfos) {
          deviceInfoList.add(new DeviceInfo(hidDeviceInfo));
        }
      }

      return deviceInfoList;
    }
    catch (IOException e) {
      e.printStackTrace();
    }

    return null;
  }

  private ArrayList<DeviceInfo> getDummyDeviceList() {
    return this.dummyDeviceList;
  }

  //region Реалізація Наглядача

  /**
   Порівнює поточний скан з попереднім

   @param scannedDevicesList
   */
  private void updateConnectedDeviceList(ArrayList<DeviceInfo>
scannedDevicesList) {
    //додавання
    for (DeviceInfo deviceInfo : scannedDevicesList) {
      if (!connectedDeviceList.contains(deviceInfo)) {
        connectedDeviceList.add(deviceInfo);
        performDeviceConnectionEvent(deviceInfo,
DeviceConnectionStateEnum.CONNECTED);
      }
    }
```

```
    //видалення
    //створює масив для ConcurrentModificationException
    ArrayList<DeviceInfo> clone = new ArrayList<>();
    clone = clone.getClass().cast(connectedDeviceList.clone());
    for (DeviceInfo deviceInfo : clone) {
      if (!scannedDevicesList.contains(deviceInfo)) {
        performDeviceConnectionEvent(deviceInfo,
DeviceConnectionStateEnum.DISCONNECTED);
        connectedDeviceList.remove(deviceInfo);
      }
    }
  }

  private void performDeviceConnectionEvent(DeviceInfo deviceName,
DeviceConnectionStateEnum connectionStateEnum) {
    // Сповістити всіх.
    for (DeviceConnectionListenerI listenerI : listeners) {
      listenerI.deviceConnectionEvent(deviceName, connectionStateEnum);
    }
  }

  //endregion
} }
```

Додаток Б

ЛІСТИНГ КЛАСУ DEVICECONNECTIONHANDLER

Клас розглянуто у пункті 2.2.

```
package com.rasalhague.mdrv.connectionlistener;

import com.rasalhague.mdrv.analysis.PacketAnalysis;
import com.rasalhague.mdrv.device.core.Device;
import com.rasalhague.mdrv.device.core.DeviceInfo;
import com.rasalhague.mdrv.logging.ApplicationLogger;
import com.rasalhague.mdrv.logging.PacketLogger;
import com.rits.cloning.Cloner;

public class DeviceConnectionHandler implements DeviceConnectionListenerI {
  @Override public void deviceConnectionEvent(DeviceInfo connectedDevice,
DeviceConnectionStateEnum deviceConnectionStateEnum) {
    DeviceInfo deviceInfoClone = new Cloner().deepClone(connectedDevice);

    ApplicationLogger.LOGGER.info(deviceInfoClone.getName() + " " +
deviceConnectionStateEnum);

    if (deviceConnectionStateEnum == DeviceConnectionStateEnum.CONNECTED) {
      //виклик Factory method
      Device device = Device.getConcreteDevice(deviceInfoClone);

      //фільтр знайомих пристроїв
      if (device != null) {

device.getRxRawDataReceiver().addListener(PacketLogger.getInstance());

device.getRxRawDataReceiver().addListener(PacketAnalysis.getInstance());

        Thread thread = new Thread(device.getDeviceCommunication());
        thread.setName(device.getDeviceInfo().getFriendlyNameWithId());
        thread.setDaemon(true);
        thread.start();
      }
      else {
        ApplicationLogger.LOGGER.info(deviceInfoClone.getName() +
                                      " on " +
                                      deviceInfoClone.getPortName()
+
                                      " ignored.");
      }
    }
```

```
    }
}
```

Додаток В

ЛІСТИНГ КЛАСУ DEVICE

Клас розглянуто у пункті 2.3.

```
package com.rasalhague.mdrv.device.core;

import com.rasalhague.mdrv.dev_communication.DeviceCommunication;
import com.rasalhague.mdrv.dev_communication.RxRawDataReceiver;
import com.rasalhague.mdrv.device.DeviceHistory;
import com.rasalhague.mdrv.logging.ApplicationLogger;
import org.reflections.Reflections;

import java.util.Set;

public abstract class Device implements DeviceTemplateI {
  private static final String FRIENDLY_NAME_FIELD_NAME = "FRIENDLY_NAME";
  private static final String VENDOR_ID_FIELD_NAME = "VENDOR_ID";
  private static final String PRODUCT_ID_FIELD_NAME = "PRODUCT_ID";
  private static final String CHANNEL_SPACING_FIELD_NAME =
"CHANNEL_SPACING";
  private static final String END_PACKET_SEQUENCE_FIELD_NAME =
"END_PACKET_SEQUENCE";
  private static final String INITIAL_FREQUENCY_FIELD_NAME =
"INITIAL_FREQUENCY";
  private static final String MANUAL_DEVICE_CONTROL_FIELD_NAME =
"MANUAL_DEVICE_CONTROL";

  private static final String REFLECTION_INIT_PATH =
"com.rasalhague.mdrv.device";
  private static DeviceHistory deviceHistory = new DeviceHistory();
  protected DeviceCommunication deviceCommunication;
  protected DeviceInfo deviceInfo;

  public static Device getConcreteDevice(DeviceInfo deviceInfo) {
    Reflections reflections = new Reflections(REFLECTION_INIT_PATH);
    Set<Class<? extends Device>> devicesClassSet =
reflections.getSubTypesOf(Device.class);

    for (Class<? extends Device> concreteDeviceClass : devicesClassSet) {
      try {
        String vendorId = (String)
concreteDeviceClass.getField(VENDOR_ID_FIELD_NAME).get(null);
        String productId = (String)
concreteDeviceClass.getField(PRODUCT_ID_FIELD_NAME).get(null);
```

```
        String friendlyName = (String)
concreteDeviceClass.getField(FRIENDLY_NAME_FIELD_NAME).get(null);
        float channelSpacing =
concreteDeviceClass.getField(CHANNEL_SPACING_FIELD_NAME).getFloat(null);
        float initialFrequency =
concreteDeviceClass.getField(INITIAL_FREQUENCY_FIELD_NAME).getFloat(null);
        byte[] endPacketSequence = (byte[])
concreteDeviceClass.getField(END_PACKET_SEQUENCE_FIELD_NAME).get(null);
        boolean useCustomReadMethod = (boolean)
concreteDeviceClass.getField(MANUAL_DEVICE_CONTROL_FIELD_NAME).get(null);

        if (deviceInfo.getProductID().equals(productId) &&
deviceInfo.getVendorID().equals(vendorId)) {
          Device device = concreteDeviceClass.newInstance();
          deviceInfo.setSomeFields(friendlyName, endPacketSequence,
initialFrequency, channelSpacing, device);
          deviceInfo.setManualDeviceControl(useCustomReadMethod);
          device.initializeObject(deviceInfo);

          return device;
        }
      }
      catch (IllegalAccessException | NoSuchFieldException |
InstantiationException e) {
        ApplicationLogger.LOGGER.severe(e.getMessage());
        e.printStackTrace();
      }
    }

    return null;
  }

  public void initializeObject(DeviceInfo deviceInfo) {
    this.deviceInfo = deviceInfo;
    this.deviceCommunication = DeviceCommunication.getInstance(deviceInfo);

    deviceHistory.checkForCollision(this.deviceInfo);
  }

  @Override public int hashCode() {
    return deviceInfo.hashCode();
  }

  @Override public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Device device = (Device) o;
```

```
    if (!deviceInfo.equals(device.deviceInfo)) return false;

    return true;
  }

  public DeviceCommunication getDeviceCommunication() {
    return deviceCommunication;
  }

  public RxRawDataReceiver getRxRawDataReceiver() {
    return deviceCommunication.getRxRawDataReceiver();
  }

  public DeviceInfo getDeviceInfo() {
    return deviceInfo;
  }
}
```

Додаток Г

ЛІСТИНГ КЛАСУ DEVICETEMPLATE

Клас розглянуто у пункті 2.3.2.

```
package com.rasalhague.mdrv.device.devices;

import com.rasalhague.mdrv.device.core.Device;

import java.util.ArrayList;

/**
 * Шаблонний клас для заповнення.
 */
public class DeviceTemplate extends Device
{
    /**
     * Використовується в GUI для ідентифікації.
     */
    public final static String FRIENDLY_NAME = "";

    /**
     * Повинен бути в 16 системі числення. Н.п. "1FFB".
     */
    public final static String VENDOR_ID = "";

    /**
     * Повинен бути в 16 системі числення.
     */
    public final static String PRODUCT_ID = "";

    /**
     * Мінімальна частота пристрою. Н.п. 2400f.
     */
    public final static float INITIAL_FREQUENCY = 2400f;

    /**
     * Відстань між каналами. Н.п. 327.450980f.
     */
    public final static float CHANNEL_SPACING = 0f;

    /**
     * Байт чи послідовність байтів, що ідентифікую кінець пакету. "Пакет"
означає RSSI пакет починаючи з INITIAL_FREQUENCY и до кінця
     * зони видимості пристрою.
     */
```

```
    public final static byte[] END_PACKET_SEQUENCE = new byte[]{};

    /**
     * Укажіть TRUE тільки якщо ви хочете управляти пристроєм вручну
     * В даному випадку програма не буде відкривати та читати пристрій
     * customReadMethod() активується
     * Приклад MetaGeekWiSpyGen1.java
     * Працює тільки для HIDUSB
     */
    public final static boolean MANUAL_DEVICE_CONTROL = false;

    /**
     * Використовуйте цей метод для ініціалізації пристрою. Якщо пристрою
не потрібна ініціалізація – лишіть пустим.
     */
    @Override
    public void initializeDevice()
    {

    }

    /**
     * Використовуйте цей метод для парсинга данних з пристрою. Формат по-
вернення – масив Byte. Кожен елемент - RSSI в форматі "-100".
     * Не передавайте @dataToParse як параметр повернення
     */
    @Override
    public ArrayList<Byte> parse(ArrayList<Byte> dataToParse)
    {
        ArrayList<Byte> finalArray = new ArrayList<>(dataToParse);

        return finalArray;
    }

    /**
     * Використовуйте цей метод для перевизначення HIDUSB поведінки. Кори-
сно, коли com.codeminders.hidapi метод не працює.
     * <p>
     * !!! ВАЖЛИВО !!! Якщо бажаєте використати цей метод встановіть
MANUAL_DEVICE_CONTROL до TRUE
     */
    @Override
    public byte[] customReadMethod()
    {
        return new byte[0];
    }
}
```

Додаток Д

ЛІСТИНГ КЛАСУ DEVICECOMMUNICATION

Клас розглянуто у пункті 2.4.2.

```
package com.rasalhague.mdrv.dev_communication;

import com.codeminders.hidapi.HIDDevice;
import com.rasalhague.mdrv.device.core.DeviceInfo;
import com.rasalhague.mdrv.logging.ApplicationLogger;
import jssc.SerialPort;

public abstract class DeviceCommunication implements Runnable {
  final DeviceInfo deviceInfo;
  final RxRawDataReceiver rxRawDataReceiver;
  public HIDDevice hidDevice;
  public SerialPort serialPort;

  protected DeviceCommunication(DeviceInfo devInfo) {
    deviceInfo = devInfo;
    rxRawDataReceiver = new RxRawDataReceiver(deviceInfo);
  }

  public static DeviceCommunication getInstance(DeviceInfo deviceInfo) {
    if (deviceInfo.getDeviceType() == DeviceInfo.DeviceType.COM) {
      return new COMDeviceCommunication(deviceInfo);
    }
    if (deviceInfo.getDeviceType() == DeviceInfo.DeviceType.HID) {
      return new HIDDeviceCommunication(deviceInfo);
    }
    if (deviceInfo.getDeviceType() == DeviceInfo.DeviceType.DUMMY) {
      return new DummyDeviceCommunication(deviceInfo);
    }

    return null;
  }

  public RxRawDataReceiver getRxRawDataReceiver() {
    return rxRawDataReceiver;
  }

  void initializeDevice() {
    deviceInfo.getDevice().initializeDevice();

    ApplicationLogger.LOGGER.info(deviceInfo.getName() + " has
initialized.");
```

```
    }
}
```

Додаток Е

ЛІСТИНГ КЛАСУ PACKETANALYSIS

Клас розглянуто у пункті 2.5.

```
public class PacketAnalysis implements DataPacketListener {
  private final HelperAnalysisMaps helperAnalysisMaps = new
HelperAnalysisMaps();
  private final List<AnalysisPerformedListener> analysisPerformedListeners
= new ArrayList<>();
  private volatile LinkedHashMap<Long, HashMap<DeviceInfo,
HashMap<AnalysisKey, ArrayList<Byte>>>> timedAnalysisResults = new
LinkedHashMap<>();

  public static PacketAnalysis getInstance() {
    return PacketAnalysisHolder.INSTANCE;
  }

  public LinkedHashMap<Long, HashMap<DeviceInfo, HashMap<AnalysisKey,
ArrayList<Byte>>>> getTimedAnalysisResults() {
    return timedAnalysisResults;
  }

  @Override public synchronized void dataPacketEvent(DataPacket dataPacket)
{
    if (dataPacket.isAnalyzable()) {
      final DeviceInfo deviceInfo = dataPacket.getDeviceInfo();
      final long packetCreationTimeMs =
dataPacket.getPacketCreationTimeMs();

      //search for last <AnalysisKey> for specific dev
      HashMap<AnalysisKey, ArrayList<Byte>> prevResultsMap = null;
      ArrayList<HashMap<DeviceInfo, HashMap<AnalysisKey, ArrayList<Byte>>>>
list = new ArrayList<>(timedAnalysisResults.values());
      for (int i = list.size() - 1; i >= 0; i--) {
        HashMap<DeviceInfo, HashMap<AnalysisKey, ArrayList<Byte>>> value =
list.get(i);
        if (value.containsKey(deviceInfo) &&
                value.get(deviceInfo).containsKey(AnalysisKey.MAX) &&
                value.get(deviceInfo).containsKey(AnalysisKey.AVR)) {
          prevResultsMap = value.get(deviceInfo);
          break;
        }
      }

      /**
```

```
       * generate scheme and put <AnalysisKey> into
       */
      HashMap<AnalysisKey, ArrayList<Byte>> hashMapToAdd = new HashMap<>();
      if (prevResultsMap != null) {
        //MAX
        ArrayList<Byte> joinMax = joinMax(dataPacket.getDataPacketValues(),
prevResultsMap.get(AnalysisKey.MAX));

        //AVR
        ArrayList<Byte> joinAvr = joinAvr(dataPacket.getDataPacketValues(),
helperAnalysisMaps.getPacketCountForDevice(deviceInfo),
helperAnalysisMaps.getRssiSumForDevice(deviceInfo));

        hashMapToAdd.put(AnalysisKey.MAX, joinMax);
        hashMapToAdd.put(AnalysisKey.AVR, joinAvr);
      }
      else {
        hashMapToAdd.put(AnalysisKey.MAX,
dataPacket.getDataPacketValues());
        hashMapToAdd.put(AnalysisKey.AVR,
dataPacket.getDataPacketValues());
        hashMapToAdd.put(AnalysisKey.NEW_SERIES, null);
      }

      /**
       * CURRENT
       */
      hashMapToAdd.put(AnalysisKey.CURRENT,
dataPacket.getDataPacketValues());

      /**
       * Perform Analysis for MODE and MEDIAN and AVR
       */
      helperAnalysisMaps.updateHelperMaps(dataPacket);
      hashMapToAdd.put(AnalysisKey.MODE, calculateMode(helperAnalysisMaps,
deviceInfo));
      hashMapToAdd.put(AnalysisKey.MEDIAN,
calculateMedian(helperAnalysisMaps, deviceInfo));

      /**
       * When we have 2 or > DataPacket in one moment
       * if packetCreationTimeMs does not exist, timedAnalysisResults will
created
       * else - it will be just updated
       */
      if (!timedAnalysisResults.containsKey(packetCreationTimeMs)) {
        HashMap<DeviceInfo, HashMap<AnalysisKey, ArrayList<Byte>>> map =
new HashMap<>();
        map.put(deviceInfo, hashMapToAdd);
```

```
        timedAnalysisResults.put(packetCreationTimeMs, map);
      }
      else {
        HashMap<DeviceInfo, HashMap<AnalysisKey, ArrayList<Byte>>> map1 =
timedAnalysisResults.get(packetCreationTimeMs);

        map1.put(deviceInfo, hashMapToAdd);
      }

      //            System.out.println(timedAnalysisResults);
      notifyAnalysisPerformedListeners(getTimedAnalysisResults());
    }
  }

  public void addListener(AnalysisPerformedListener toAdd) {
    analysisPerformedListeners.add(toAdd);
  }

  /**
   This method join new data to second parameter prevData
   */
  private synchronized ArrayList<Byte> joinMax(ArrayList<Byte> newData,
ArrayList<Byte> prevData) {
    if (newData.size() == prevData.size()) {
      ArrayList<Byte> joinedData = new ArrayList<>(prevData);
      byte prevNumber;
      byte newDataNumber;

      for (int i = 0, prevDataSize = prevData.size(); i < prevDataSize;
i++) {
        prevNumber = prevData.get(i);
        newDataNumber = newData.get(i);

        if (newDataNumber > prevNumber) {
          joinedData.set(i, newDataNumber);
        }
      }

      return joinedData;
    }
    else {
      ApplicationLogger.LOGGER.severe("newData.size() != prevData.size();
can not process. Returning prevData");
      ApplicationLogger.LOGGER.info("prevData" + prevData.size());
      ApplicationLogger.LOGGER.info("newData" + newData.size());

      return prevData;
    }
```

```
  }

  private synchronized ArrayList<Byte> joinAvr(ArrayList<Byte> newData,
Integer packetsAmount, ArrayList<Integer> rssiSumForDevice) {
    ArrayList<Byte> avrArray = new ArrayList<>();
    if (newData.size() == rssiSumForDevice.size()) {
      int newRssi;
      int sumRssi;
      for (int i = 0, newDataSize = newData.size(); i < newDataSize; i++) {
        newRssi = newData.get(i);
        sumRssi = rssiSumForDevice.get(i);

        avrArray.add((byte) ((sumRssi + newRssi) / (packetsAmount + 1)));
      }
    }
    else {
      ApplicationLogger.LOGGER.severe("newData.size() != prevData.size();
can not process. Returning prevData");
      ApplicationLogger.LOGGER.info("rssiSumForDevice " +
rssiSumForDevice.size());
      ApplicationLogger.LOGGER.info("newData " + newData.size());
    }

    return avrArray;
  }

  private void notifyAnalysisPerformedListeners(LinkedHashMap<Long,
HashMap<DeviceInfo, HashMap<AnalysisKey, ArrayList<Byte>>>>
analysisResultsMap) {
    for (AnalysisPerformedListener listener : analysisPerformedListeners) {
      listener.analysisPerformedEvent(analysisResultsMap);
    }
  }

  private static class PacketAnalysisHolder {
    /**
     The constant INSTANCE.
     */
    public static final PacketAnalysis INSTANCE = new PacketAnalysis();
  }

  synchronized ArrayList<Byte> calculateMode(HelperAnalysisMaps
helperAnalysisMaps, DeviceInfo deviceInfo) {
    /**
     * MODE postprocessing
     */
    ArrayList<Byte> mode = new ArrayList<>();
    HashMap<DeviceInfo, ArrayList<HashMap<Byte, Integer>>> helperMap =
helperAnalysisMaps.getModeMedianHelperMap();
```

```
    int maxRSSICountValue;
    byte maxRSSI;

    Set<DeviceInfo> helperMapDeviceKeys = helperMap.keySet();
    for (DeviceInfo helperMapDeviceKey : helperMapDeviceKeys) {
      if (helperMapDeviceKey.equals(deviceInfo)) {
        ArrayList<HashMap<Byte, Integer>> pointsHelperArray =
helperMap.get(helperMapDeviceKey);
        for (HashMap<Byte, Integer> helperDataPointMap : pointsHelperArray)
{
          maxRSSICountValue = 0;
          maxRSSI = 0;

          Set<Byte> rssiKeys = helperDataPointMap.keySet();
          for (Byte helperDataPointMapKey : rssiKeys) {
            if (helperDataPointMap.get(helperDataPointMapKey) >
maxRSSICountValue || maxRSSICountValue == 0) {
              maxRSSICountValue =
helperDataPointMap.get(helperDataPointMapKey);
              maxRSSI = helperDataPointMapKey;
            }
          }

          mode.add(maxRSSI);
        }
      }
    }

    return mode;
  }

  synchronized ArrayList<Byte> calculateMedian(HelperAnalysisMaps
helperAnalysisMaps, DeviceInfo deviceInfo) {
    /**
     * MEDIAN postprocessing
     */
    ArrayList<Byte> median = new ArrayList<>();
    HashMap<DeviceInfo, ArrayList<HashMap<Byte, Integer>>> helperMap =
helperAnalysisMaps.getModeMedianHelperMap();
    int rssiSortedArraySize;

    Set<DeviceInfo> helperMapDeviceKeys = helperMap.keySet();
    for (DeviceInfo helperMapDeviceKey : helperMapDeviceKeys) {
      if (helperMapDeviceKey.equals(deviceInfo)) {
        ArrayList<HashMap<Byte, Integer>> pointsHelperArray =
helperMap.get(helperMapDeviceKey);
        for (HashMap<Byte, Integer> helperDataPointMap : pointsHelperArray)
{
```

```
          ArrayList<Byte> rssiSortedArray = new ArrayList<>(new
TreeMap<>(helperDataPointMap).keySet());
          rssiSortedArraySize = rssiSortedArray.size();
          median.add(rssiSortedArray.get(rssiSortedArraySize / 2));
        }
      }
    }

    return median;
  }
}
```

Додаток Ж

ЛІСТИНГ КЛАСУ APPLICATIONLOGGER

Клас розглянуто у пункті 2.6.

```
package com.rasalhague.mdrv.logging;

import com.rasalhague.mdrv.Utility.Utils;

import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.MissingResourceException;
import java.util.logging.*;

public class ApplicationLogger extends Logger {
  //    public final static Logger GLOBAL_LOGGER =
Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);
  public final static Logger LOGGER = new ApplicationLogger();
  private final static String LOGGER_NAME = "ApplicationLogger";

  private ApplicationLogger() {
    super(ApplicationLogger.LOGGER_NAME, null);
  }

  public static Logger getLogger() {
    return LOGGER;
  }

  static public void setup() {
    LOGGER.setLevel(Level.ALL);

    try {
      String fileName = "logs" + File.separator +
Utils.addTimeStampToFileName("Application");
      Utils.createFile(fileName);

      //choose file header to add
      FileHandler fileTxt = new FileHandler(fileName);
      LOGGER.addHandler(fileTxt);

      ConsoleHandler consoleHandler = new ConsoleHandler();
      LOGGER.addHandler(consoleHandler);

      setFormatterToLoggerHandlers(LOGGER, new MyLogFormatter());
```

```
      LOGGER.info("Logger has initialized");
    }
    catch (SecurityException e) {
      LOGGER.log(Level.SEVERE, "Cannot create file due to Security
reason.", e);
    }
    catch (IOException e) {
      LOGGER.log(Level.SEVERE, "Cannot create file due to IO error.", e);
    }

  }

  public synchronized static void addCustomHandler(Handler handler) {
    LOGGER.addHandler(handler);

    setFormatterToLoggerHandlers(LOGGER, new MyLogFormatter());
  }

  public synchronized static void closeHandlers() {
    Handler[] handlers = LOGGER.getHandlers();
    for (Handler handler : handlers) {
      handler.close();
    }
  }

  private static void setFormatterToLoggerHandlers(Logger logger, Formatter
formatter) {
    Handler[] handlers = logger.getHandlers();
    for (Handler handler : handlers) {
      handler.setFormatter(formatter);
    }

    Logger loggerParent = logger.getParent();
    if (loggerParent != null) {
      Handler[] loggerParentHandlers = loggerParent.getHandlers();
      for (Handler handler : loggerParentHandlers) {
        handler.setFormatter(formatter);
      }
    }
  }
}

class MyLogFormatter extends Formatter {
  //    private final static String LOGGER_NAME = "ApplicationLogger";
  private static final SimpleDateFormat DATE_FORMAT = new
SimpleDateFormat("dd.MM.yy HH:mm:ss.SSS");

  @Override public String format(LogRecord record) {
```

```
    //        LocationInfo locationInfo = new LocationInfo(new Throwable(),
LOGGER_NAME);

    StringBuilder builder = new StringBuilder();
    //        String bracerOpen = "[";
    //        String bracerClose = "]";
    //        String dot = ".";
    String separator = " ";

    //        String packageName = this.getClass().getPackage().getName();
    String packageName = "com.rasalhague.mdrv";

    builder.append(DATE_FORMAT.format(new Date(record.getMillis())));

    builder.append(separator);

    builder.append("[").append(record.getLevel()).append("]");

    builder.append(separator);


builder.append("[").append(record.getSourceClassName().replace(packageName
+ ".", "")).append(".").append(record.getSourceMethodName()).append("]");

    builder.append(separator);

    builder.append(formatMessage(record));

    builder.append("\n");

    return builder.toString();
  }
}
```

Додаток З

ПРИКЛАД СЕРІАЛІЗОВАНИХ ДАНИХ

Серіалізація даних розглянута у пункті 2.10.


```
[{
      "dataPacketValues" : [-100, -104, -103, -104, -103, -103, -100, -106,
-103, -105, -101, -102, -100, -101, -103, -104, -105, -104, -104, -104,
-103, -102, -103, -101, -102, -103, -104, -103, -102, -104, -102, -103,
-103, -105, -103, -103, -105, -101, -104, -104, -102, -105, -103, -103,
-104, -101, -101, -101, -103, -105, -103, -103, -104, -102, -103, -103,
-105, -101, -106, -104, -101, -104, -105, -105, -104, -102, -103, -101,
-102, -103, -105, -103, -100, -104, -101, -103, -106, -102, -103, -104,
-105, -104, -105, -102, -102, -104, -105, -100, -105, -106, -104, -102,
-100, -102, -102, -102, -105, -102, -101, -104, -103, -104, -103, -100,
-104, -102, -103, -103, -105, -102, -104, -105, -104, -102, -103, -104,
-104, -102, -103, -105, -103, -106, -105, -105, -106, -102, -105, -100,
-106, -103, -102, -102, -103, -105, -105, -104, -102, -104, -103, -102,
-103, -96, -89, -103, -105, -102, -104, -103, -105, -104, -104, -102, -105,
-99, -106, -104, -104, -101, -101, -105, -103, -104, -107, -106, -103,
-104, -102, -105, -103, -102, -104, -102, -104, -103, -100, -103, -103,
-104, -104, -102, -104, -104, -102, -104, -105, -101, -103, -103, -101,
-101, -103, -104, -102, -102, -105, -102, -103, -103, -104, -102, -103,
-103, -99, -103, -104, -102, -106, -107, -102, -102, -106, -101, -102,
-104, -102, -104, -101, -105, -101, -99, -102, -102, -104, -104, -105,
-102, -104, -105, -103, -103, -104, -102, -100, -104, -103, -106, -105,
-106, -99, -105, -99, -105, -103, -101, -104, -104, -104, -104, -105, -105,
-103, -103, -103, -105, -100, -101, -103, -105, -104, -105, -106, -99,
-104, -104, -106, -105, -101, -103, -102, -101, -103, -104, -105, -105,
-104, -104, -103, -100, -102, -105, -103, -104, -105, -102, -104, -105,
-103, -103, -104, -102],
      "packetCreationTimeMs" : 1419014693138,
      "pointsAmount" : 290,
      "isAnalyzable" : true,
      "deviceInfo" : {
            "friendlyName" : "MetaGeek WiSpy 24x2",
            "vendorID" : "1DD5",
            "productID" : "2410",
            "name" : "Wi-Spy 2.4x2",
            "portName" : "0001:0004:00",
            "deviceType" : "HID",
            "endPacketSequence" : [74, 0, 0, 0],
            "initialFrequency" : 2400.0,
            "channelSpacing" : 327.586,
            "id" : 0,
            "manualDeviceControl" : false
```

```
      }
   }, {
      "dataPacketValues" : [-104, -100, -101, -104, -105, -103, -103, -103,
-104, -103, -104, -100, -104, -105, -103, -104, -100, -107, -106, -105,
-102, -102, -106, -103, -102, -104, -101, -102, -104, -104, -103, -100,
-100, -105, -102, -104, -100, -105, -104, -104, -104, -101, -104, -105,
-104, -102, -106, -102, -106, -100, -105, -105, -101, -105, -104, -98,
-100, -103, -103, -102, -103, -101, -103, -105, -104, -104, -97, -103,
-104, -105, -105, -102, -102, -103, -100, -103, -104, -102, -104, -100,
-103, -105, -103, -105, -105, -104, -103, -103, -102, -102, -103, -103,
-103, -102, -104, -108, -106, -103, -103, -103, -101, -105, -104, -103,
-103, -103, -104, -103, -104, -105, -104, -106, -103, -104, -102, -104,
-104, -102, -106, -101, -104, -104, -104, -101, -104, -99, -104, -102, -99,
-102, -103, -102, -99, -103, -103, -98, -106, -102, -104, -103, -101, -100,
-104, -103, -102, -106, -104, -101, -102, -103, -103, -103, -103, -100,
-95, -88, -87, -89, -89, -100, -103, -103, -102, -97, -105, -106, -105,
-104, -106, -104, -101, -102, -103, -105, -99, -99, -102, -100, -104, -103,
-103, -104, -106, -103, -102, -104, -103, -104, -105, -103, -105, -103,
-105, -104, -106, -105, -105, -104, -103, -100, -105, -105, -105, -103,
-103, -105, -103, -104, -102, -105, -102, -104, -102, -105, -104, -103,
-103, -101, -103, -101, -104, -104, -105, -102, -102, -105, -102, -103,
-101, -103, -105, -106, -104, -103, -103, -103, -103, -101, -102, -101,
-101, -105, -99, -100, -104, -106, -106, -103, -102, -104, -105, -103,
-100, -104, -102, -102, -104, -104, -103, -104, -106, -106, -104, -105,
-101, -102, -105, -106, -106, -103, -102, -105, -102, -102, -107, -102,
-104, -103, -102, -104, -103, -102, -102, -105, -102, -103, -101, -103,
-105, -106],
      "packetCreationTimeMs" : 1419014693685,
      "pointsAmount" : 290,
      "isAnalyzable" : true,
      "deviceInfo" : {
            "friendlyName" : "MetaGeek WiSpy 24x2",
            "vendorID" : "1DD5",
            "productID" : "2410",
            "name" : "Wi-Spy 2.4x2",
            "portName" : "0001:0004:00",
            "deviceType" : "HID",
            "endPacketSequence" : [74, 0, 0, 0],
            "initialFrequency" : 2400.0,
            "channelSpacing" : 327.586,
            "id" : 0,
            "manualDeviceControl" : false
      }
   }, {
      "dataPacketValues" : [-97, -98, -95, -97, -97, -97, -98, -97, -98,
-98, -97, -98, -95, -97, -95, -98, -100, -97, -98, -94, -97, -98, -98, -97,
-98, -100, -100, -98, -98, -97, -98, -97, -95, -97, -98, -97, -98, -97,
-97, -98, -97, -98, -95, -97, -98, -97, -97, -97, -98, -97, -100, -97,
-100, -97, -95, -95, -100, -95, -97, -98, -98, -97, -97, -97, -98, -100,
```

```
-100, -97, -98, -97, -98, -98, -98, -98, -98, -97, -97, -100, -97, -98,
-100, -97, -97,
          -100],
     "packetCreationTimeMs" : 1419015634406,
     "pointsAmount" : 84,
     "isAnalyzable" : true,
     "deviceInfo" : {
          "friendlyName" : "MetaGeek Wi-Spy Gen 1",
          "vendorID" : "1781",
          "productID" : "083E",
          "name" : "Wi-Spy",
          "portName" : "0001:0006:00",
          "deviceType" : "HID",
          "endPacketSequence" : [-1],
          "initialFrequency" : 2399.0,
          "channelSpacing" : 989.0,
          "id" : 0,
          "manualDeviceControl" : true
     }
  }
]
```

Додаток И

ПРОШИВКА WIXEL POLOLU

Робота з Wixel Pololu описана у пункті 4.4.

```
#include <wixel.h>
#include <radio_registers.h>
#include <stdio.h>
#include <usb.h>
#include <usb_com.h>

static int16 XDATA rssiValue[256];

void updateLeds()
{
    usbShowStatusWithGreenLed();
    // Yellow LED is controlled by checkRadioChannels
    LED_RED(0);
}

void analyzerInit()
{
    radioRegistersInit();

    MCSM0 = 0x14;    // Auto-calibrate  when going from idle to RX or TX.
    MCSM1 = 0x00;    // Disable CCA.  After RX, go to IDLE.  After TX, go
to IDLE.
    // We leave MCSM2 at its default value = 0x07
    MDMCFG2 = 0x70;   //disable sync word detection
    RFST = 4; //idle radio
}

void checkRadioChannels()
{
    uint16 i;
    uint16 channel;

    LED_YELLOW(1);

    for(channel=0; channel<256; channel++)
    {
        int32 rssiSum;
        rssiValue[channel] = -115;
        while(MARCSTATE != 1);  //radio should already be idle, but check
anyway
        CHANNR = channel;
```

```
        RFST = 2;  // radio in RX mode and autocal
        while(MARCSTATE != 13);  //wait for RX mode
        rssiSum = 0;
        for (i=0; i<100; i++)
        {
            if (TCON & 2) //radio byte available?
            {
                uint8 rfdata = RFD; // read byte
                TCON &= ~2; //clear ready flag
            }
            rssiSum += radioRssi();
        }
        RFST = 4; //idle radio
        rssiValue[channel] = (int16) (rssiSum/100);

        frequentTasks();
    }  // the above loop takes about 414 ms on average, so about 1.6
ms/channel

    LED_YELLOW(0);
}

void putchar(char c)
{
    while(!usbComTxAvailable()){ frequentTasks(); }
    usbComTxSendByte(c);
}

void reportResults()
{
    uint16 i;
    for (i=0; i<256; i++) { printf("%4d ", rssiValue[i]); }
    printf("\n");
}

void frequentTasks()
{
    boardService();
    usbComService();
    updateLeds();
}

void main()
{
    systemInit();
    usbInit();
    analyzerInit();

    while(1)
```

```
    {
        frequentTasks();
        checkRadioChannels();
        reportResults();
    }
}
```