

**WYŻSZA SZKOŁA INFORMATYKI I ZARZĄDZANIA
„Copernicus” we Wrocławiu**

WYDZIAŁ INFORMATYKI

Kierunek studiów: **Informatyka**

Poziom studiów: **Studia pierwszego stopnia-inżynierskie**

Specjalność: **Systemy i sieci komputerowe**

PRACA DYPLOMOWA INŻYNIERSKA

Paweł Koryciński

**Symulator sterownika do RET-a implementujący
protokół AISG 2.0**

Ret driver simulator for AISG 2.0 Device

Ocena pracy:
(ocena pracy dyplomowej, data, podpis promotora)

.....
(pieczętka uczelni)

Promotor:

dr Grzegorz Debita

WROCŁAW 2020

Spis treści

Wykaz skrótów i symboli	5
1 Wstęp	8
1.1 Wprowadzenie	8
1.2 Cel pracy	8
1.3 Motywacja	8
1.4 Zakres	9
1.4.1 Baza danych	9
1.4.2 Back-end	10
1.4.3 Front-end	10
1.4.4 Testowanie	10
Część Przeglądowa	12
2 RET	13
2.1 Prezentacja urządzenia	13
2.2 Zmiana szerokości głównej wiązki fali elektromagnetycznej	14
3 Protokół AISG 2.0	16
3.1 Warstwy	16
3.1.1 1-sza - Fizyczna	16
3.1.2 2-ga - Łącza danych	16
3.1.3 7-ma - Aplikacji	16
3.2 HDLC	17
3.2.1 Struktura ramki	17
3.2.2 Typy ramek	17
3.3 Typy ramek HDLC	18
3.3.1 Ramka XID	18
3.3.2 Ramka U	19
3.3.3 Ramka I	19
4 Dobre praktyki programowania obiektowego	21
4.1 Zasada pojedynczej odpowiedzialności	21
4.2 Zasada otwarte-zamknięte	21
4.3 Zasada podstawiania Liskov	21

4.4	Zasada segregacji interfejsów	21
4.5	Zasada odwrócenia zależności	22
	Część Praktyczna	23
5	Wzorce projektowe	24
5.1	Behawioralne	24
5.1.1	Komenda	24
5.1.2	Obiekt pusty	26
5.1.3	Metoda szablonowa	28
5.1.4	Strategia	28
5.1.5	Wstrzykiwanie zależności	31
5.1.6	Inicjowanie przy pozyskaniu zasobu	31
5.2	Kreacyjne	32
5.2.1	Budowniczy	32
5.2.2	Fabryka	33
6	Zewnętrzne biblioteki użyte w projekcie	34
6.0.1	ZeroMQ	34
6.0.2	GTest	34
6.0.3	CMake	35
6.0.4	Boost	35
7	Wymagania funkcjonalne wraz z komendami je realizującymi	36
7.1	Warstwa fizyczna	36
7.1.1	Ustanowienie prędkości połączenia - SetLinkSpeed	36
7.2	Warstwa łącza danych	36
7.2.1	Negocjacja roli - AddressAssignment	36
7.2.2	Negocjacja parametrów HDLC - HDLCParameters	36
7.2.3	Ustanowienie normalnego trybu odpowiedzi - LinkEstablishment	36
7.2.4	Negocjacja parametrów HDLC - 3GPPReleaseID	36
7.2.5	Negocjacja parametrów HDLC - AISGProtocolVersion	36
7.3	Warstwa aplikacyjna	36
7.3.1	Kalibracja - Calibrate	36
8	Uruchomienie programu	37
8.1	Konfiguracja środowiska	37
8.2	Kompilacja kodu źródłowego oraz wystartowanie programu	37
8.3	Efekt końcowy	38

9 Analiza nawiązanej komunikacji	39
9.1 Ustanowienie prędkości połączenia	40
9.2 Skanowanie urządzeń	40
9.3 Żądanie adresacji	42
9.4 Ponowne skanowanie urządzeń	44
9.5 Negocjacje parametrów HDLC	45
9.6 Przejście na normalny tryb odpowiedzi	46
9.7 Wersja standardu 3GPP	46
9.8 Wersja protokołu AISG	47
9.9 Kalibracja	48
10 Testowanie oprogramowania	50
10.1 Testy jednostkowe	50
10.1.1 Testowanie bazy danych	51
10.1.2 Testowanie budowania pełnej ramki AISG	52
10.1.3 Testowanie parsera danych wejściowych użytkownika	52
10.1.4 Testowanie fabryki tworzącej ciało ramki AISG	52
10.1.5 Testowanie interpretera ramki HDLC	52
10.2 Testy integracyjne	52
10.2.1 Integracja bazy danych z interfejsem użytkownika	53
10.2.2 Integracja interfejsu użytkownika z kontrolerem komend AISG	54
10.2.3 Integracja interfejsu użytkownika, kontrolera komend AISG oraz fabryki ramek HDLC	55
10.3 Manualne testy systemowe	55
10.4 Automatyczne testy systemowe	56
11 Dodatkowe informacje	57
11.1 Środowisko uruchomieniowe	57
11.2 Licencje	57
12 Podsumowanie	58
12.1 Co zostało zrealizowane	58
12.2 Co nie zostało zrealizowane	59
12.2.1 Walidacja sumy CRC	59
12.2.2 Podmiana bajtów o wartości flagi startu/stopu	59
12.2.3 Interwały czasowe	59
12.2.4 Ustalanie maski	60
12.3 Możliwości rozwoju	60

Spis treści

Bibliografia	61
Spis rysunków	62
Spis listingów	64
Zawartość płyty CD	65
Załączniki	66
Oświadczenie autorskie	66
Oświadczenie o udostępnieniu pracy	67

Wykaz skrótów i symboli

3GPP — organizacja normalizująca rozwój telefonii komórkowej;

5G — standard sieci komórkowej, następca LTE;

ADDR — Address - adres docelowy urządzenia budujący ramkę;

AISG 2.0 — protokół bazujący na komunikacji half duplex oraz protokoły HDLC;

Back-end — warstwa oprogramowania obsługująca niskopoziomową logikę biznesową;

bajt — najmniejsza adresowalna jednostka informacji pamięci komputerowej;

bit — najmniejsza jednostka informacji w odniesieniu do sprzętu komputerowej;

boost — zewnętrzna biblioteka dla języka C++;

broadcast — rozgłoszeniowy tryb transmisji danych;

C++ — język programowania;

ciało ramki — ramka AISG z pominięciem bajtów startu, stopu, sumy CRC;

CRC-16 — 16-bitowy cykliczny kod nadmiarowy;

CRC — Cyclic Redundancy Check - system sum kontrolnych;

CTRL — Control - bajt kontrolny ramki;

DB — database - baza danych;

debugowanie — proces analizy programu pod kątem zaistniałych błędów;

Debug — priorytet logowania wskazujący informacje potrzebne podczas debugowania;

delete — słowo kluczowe języka C++, którego użycie wywołuje destruktor obiektu;

driver — sterownik - program obsługujący urządzenie podłączone do komputera;

enkapsulacja — opakowanie danych z wyższej warstwy w warstwie niższej;

Error — priorytet logowania informujący o błędny wykonaniu programu;

execute — metoda interfejsu klasy ICommand służąca do uruchomienia komendy;

Front-end — warstwa oprogramowania obsługująca odbiór danych od użytkownika;

GI — Group Identifier - identyfikator grupy ramki;

GL — Group Length - długość grupy czyli liczba bajtów która pojawi się od kolejnej pozycji włącznie aż sumy CRC;

GSM — Global System for Mobile Communications - standard telefonii komórkowej;

half duplex — połączenie w którym naprzemienne jest przesyłanie i odbieranie informacji;

HDLC body — ramka HDLC bez bajtów startu, stopu oraz sumy CRC;

HDLC — High-Data Link Control - protokół warstwy łączności danych;

h — hour - godzina;

Info — priorytet logowania informujący o ważnych etapach w wykonaniu programu;

IPC — Inter-process communication - sposób komunikacji pomiędzy procesami OS;

klasa finalna — klasa po której nie można zdefiniować dziedziczenia;

klucz — unikalny identyfikator obiektu w bazie danych;

kontroler — klasa służąca do kolejkowania oraz uruchamiania komend;

lekser — dokonuje podziału wartości podanych przez użytkownika na tokeny;

little endian — cienkokońcowość;

LTE — Long Term Evolution - standard bezprzewodowego przesyłu danych;

metoda — funkcja należąca do klasy;

min — minute - minuta;

model OSI — model odniesienia łączenia systemów otwartych;

ms — milisecond - milisekunda;

N(R) — receive sequence number, numer porządkowy odebranej ramki;

N(S) — send sequence number, numer porządkowy wysłanej ramki;

null — wartość przypisywana do wskaźnika równa 0;

numer portu — jeden z parametrów socketa, identyfikujący proces nim zarządzający;

OFDMA — metoda zwielokrotnienia w dziedzinie częstotliwości;

OS — Operating system - system operacyjny, np: Linux, Windows;

P/F bit — Pool/Final bit - obliczany podczas kalkulacji bajtu kontrolnego;

payload — fragment wiadomości zawierający jedynie istotne informacje;

PI — Parameter Identifier - identyfikator parametru ramki;

PL — Parameter Length - długość wartości parametru ramki;

PL — Parameter Length - długość wartości parametru ramki;

POSIX Regexp — standard zapisu wyrażeń regularnych;

PV — Parameter Value - wartość parametru ramki;

RAII — Resource acquisition is initialization - inicjowanie przy pozyskaniu zasobu;

ramka — pakiet danych;

RET — Remote Electrical Tilt - urządzenie zmieniające elektryczny kąt wiązki anteny;

RNR — Receiver not ready - wartość ramki nadzorującej oznaczająca, że odbiornik nie jest gotowy;

RS-485 — standard transmisji szeregowej;

SNRM — Set Normal Response Mode - przejdź na normalny system odpowiedzi;

socket — dwukierunkowy punkt końcowy połączenia;

std::any — klasa reprezentująca dowolny typ danych;

std::string — klasa reprezentująca łańcuch znaków;

std::vector — klasa reprezentująca dynamiczną tablicę;

STL — Standard Template Library - biblioteka C++ w przestrzeni nazw std:::

s — second - sekunda;

TCP — Transmission Control Protocol - protkół sterowania transmisją;

TDD — Test Driven Development - programowanie sterowane testami;

token — najmniejsza analizowana jednostka tekstowa, danych wejściowych użytkownika;

Trace — priorytet logowania najwyższej rangi;

Spis treści

UA — Unnumbered Acknowledged - zaakceptowana ramka nienumerowana;

UDP — User Datagram Procotol - protokół pakietów użytkownika;

UI — User Interface - interfejs użytkownika;

UMTS — Universal Mobile Telecommunications System - standard telefonii komórkowej;

UniqueId — złożenie numeru seryjnego wraz z kodem producenta;

USB — Universal Serial Bus - uniwersalna magistrala szeregową;

Warning — priorytet logowania informujący o wykonaniu mogącym powodować błędy;

WCDMA — technika związana z dostępem do sieci radiowej;

1. Wstęp

1.1 Wprowadzenie

Stacja nadawcza to zintegrowany system składający się z modułu systemowego, modułu rozszerzeniowego, radia oraz fizycznej anteny. Ma ona na celu dostarczenie jak największej liczbie ludzi sygnału telekomunikacyjnego w celu nawiązania połączenia głosowego, wysłania wiadomości tekstowej czy skorzystania ze skrzynki mailowej. Głównym problemem jest to, że stacja nadawcza może być umieszczona w jednym miejscu, natomiast odbiorcy, mogą przemieszczać się, co niesie ze sobą problem efektywnego pokrycia obszaru zasięgiem sieci operatora. W celu modyfikacji charakterystyki sygnału, anteny dipolowe usprawniane są o dodatkowe urządzenie o nazwie *RET*, które jest szeroko stosowane w technologiach *GSM*, *WCDMA* czy *LTE*.

1.2 Cel pracy

Jako cel obrałem zaznajomienie się z protokołem komunikacyjnym AISG 2.0, którego użycie możemy zaobserwować na drodze pomiędzy radio modułem a wzmacniaczem antenowym czy RET-em. Jest to krok obowiązkowy przed rozpoczęciem zapoznawania się z AISG 3.0. Implementacja protokołu będzie wymagała wysokich umiejętności programowania w języku C++, tworzenia testów, wdrażania wzorców projektowych oraz wiedzy z zakresu systemu kontroli wersji czy inżynierii oprogramowania.

1.3 Motywacja

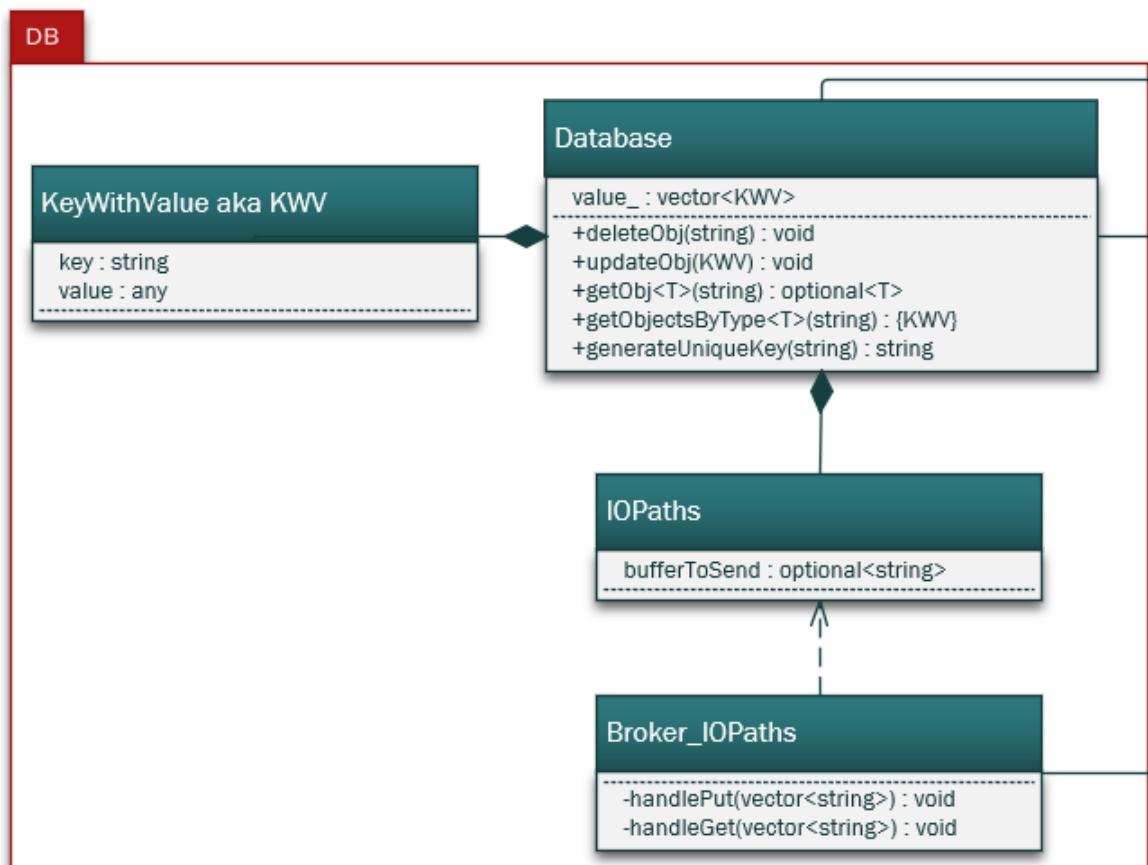
W przyszłości podłączenie RET-a do komputera (pomijając radio moduł) przy pomocy adaptera *RS-485 -> USB*, w celu skrócenia czasu testowania urządzenia.

1.4 Zakres

Praca obejmuje implementację sterownika, którego analizę możemy podzielić na:

1.4.1 Baza danych

- Implementacja:
 - `std::vector<std::string, std::any>`
 - Programowanie optymalne dla cache-u procesora
- Klucz:
 - Generowanie unikalnego;
 - Walidacja - Extended *POSIX Regexp*: `,/[a-z]+_[1-9]+”;`
- Operacje:
 - Dodaj;
 - Usuń;
 - Aktualizuj;
 - Pobierz wartość na podstawie : konkretnego/typu klucza;



Rysunek 1.1. Diagram klas przedstawiający relacje klas tworzących bazę danych.
(Opracowanie własne)

1.4.2 Back-end

- Realizacja wzorców projektowych:
 - Komenda;
 - Metoda szablonowa;
 - Obiekt pusty;
 - Budowniczy;
 - *RAII*;
 - Wstrzykiwanie zależności;
- Implementacja:
 - L1 - Warstwy fizycznej;
 - L2 - Warstwy łącza danych;
 - L7 - Warstwy aplikacyjnej;
- Logowanie ruchu aplikacji:
 - <h:min::s::ms> priorytet [nazwaPliku::nazwaFunkcji:numerLinii] komunikat;
 - Obsługiwane priorytety: Trace < Debug < Info < Warning < Error
 - Filtrowanie w zależności od wybranego minimalnego priorytetu
 - Rezultat przekazywany na wyjście standardowe oraz do pliku tekstowego

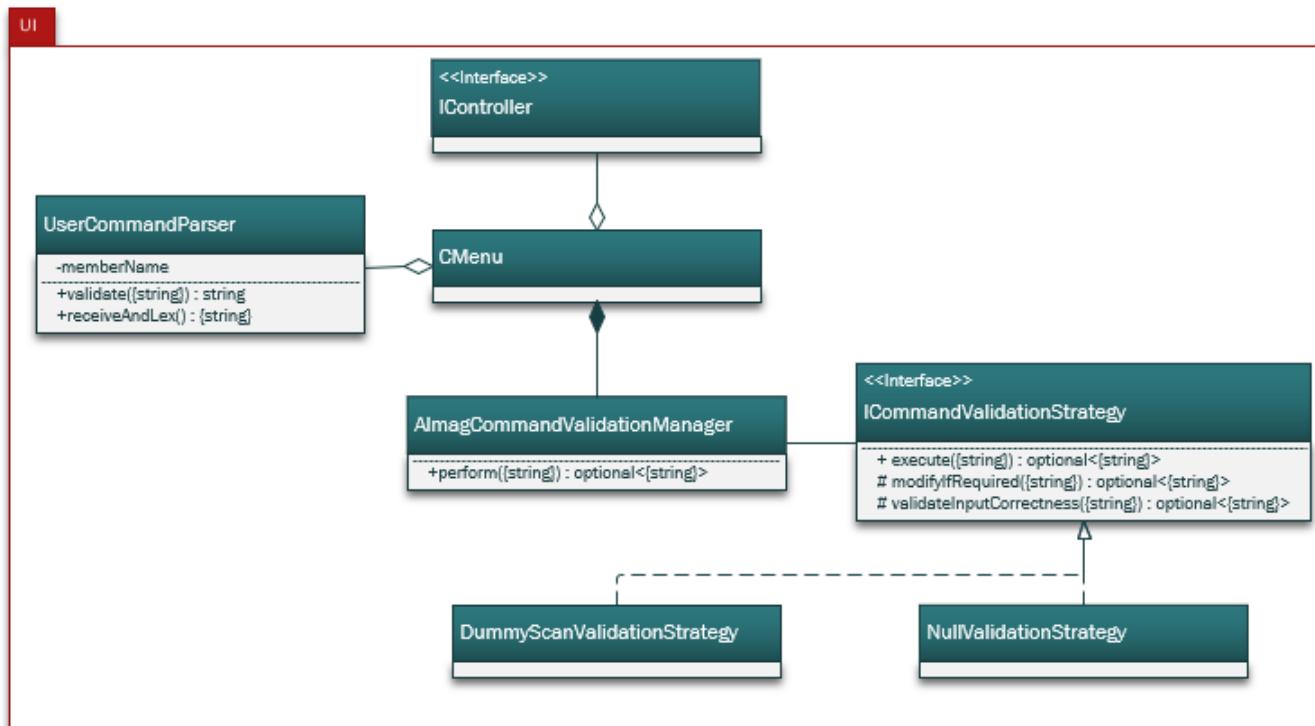
1.4.3 Front-end

- Konsolowy interfejs użytkownika umożliwiający wydawanie komend do:
 - Kontrolera sterownika;
 - Bazy danych;
- Realizacja wzorców projektowych:
 - Komenda;
 - Metoda szablonowa;
- Przekazywanie logów aplikacji z Back-endu;
- Walidacja komend wpisanych przez użytkownika:
 - Odrzucanie nieznanych komend;
 - Podpowiedź odnośnie wartości argumentów komend już znanych;

1.4.4 Testowanie

- Front-end
 - Jednostkowe
 - Modułowe
- Back-end
 - Jednostkowe
 - * HDLC
 - * HDLC Body

1. Wstęp



Rysunek 1.2. Diagram klas przedstawiający klasy wchodzące w skład interfejsu użytkownika.
(Opracowanie własne)

- * DB
- Modułowe
 - * Happy Path
 - * Sad Path
- Integracja
 - * UI + DB
 - * UI + DB + Back-end Controller
- Komponentowe

W celu weryfikacji działania symulatora sterownika od początku do końca zalecane jest uruchomienie całego komponentu jako czarną czarną skrzynkę oraz operowanie na nim przy pomocy zaimplementowanego interfejsu. W tym celu zaimplementowany został również symulator urządzenia, który odbiera wiadomości oraz odpowiada na nie.

Część Przeglądowa

2. RET

2.1 Prezentacja urządzenia

Na poniższym rysunku przedstawiono RET-a czyli urządzenie dla którego zaimplementowano symulator sterownika.[5]



Rysunek 2.1. RET - Miejsce na antenę.
(Zdjęcie własne)



Rysunek 2.2. RET - podłączonym kablem RS-485 oraz włożoną atrapą anteny.
(Zdjęcie własne)

2.2 Zmiana szerokości głównej wiązki fali elektromagnetycznej



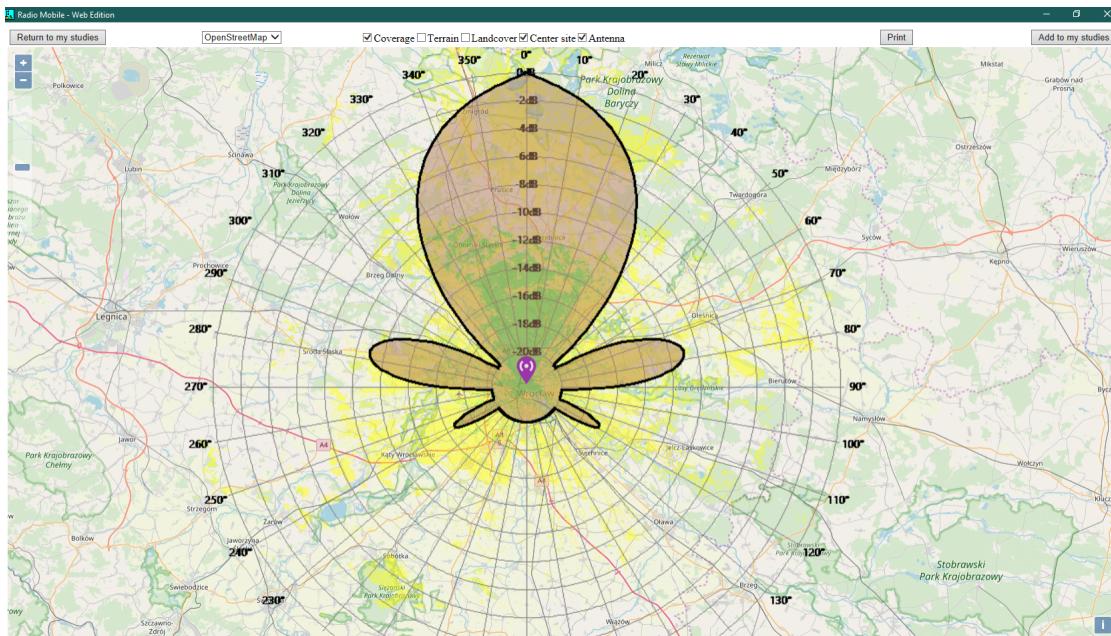
Rysunek 2.3. Rysunek przedstawiający kąt modyfikowany przy pomocy RET-a.
(Opracowanie własne)

Poniższe rysunki wygenerowano dzięki programowi Radio Mobile[13], Przedstawiono na nich szerokość wiązki promieni sygnału radiowego w osi poziomej[1] zmieniającą się dzięki modyfikacji elektrycznego kąta na wartość 40-tu stopni. Podczas symulacji użyto antenę Yagi, która aktualnie nie jest stosowana w technologii mobilnej z racji wspieranych przez nią częstotliwości, gdyż są one znacznie niższe aniżeli te wymagane przez WCDMA, LTE czy 5G, aczkolwiek bardzo dobrze odzwierciedla działanie tych rzeczywistych ze względu na swoją charakterystykę kierunkową.

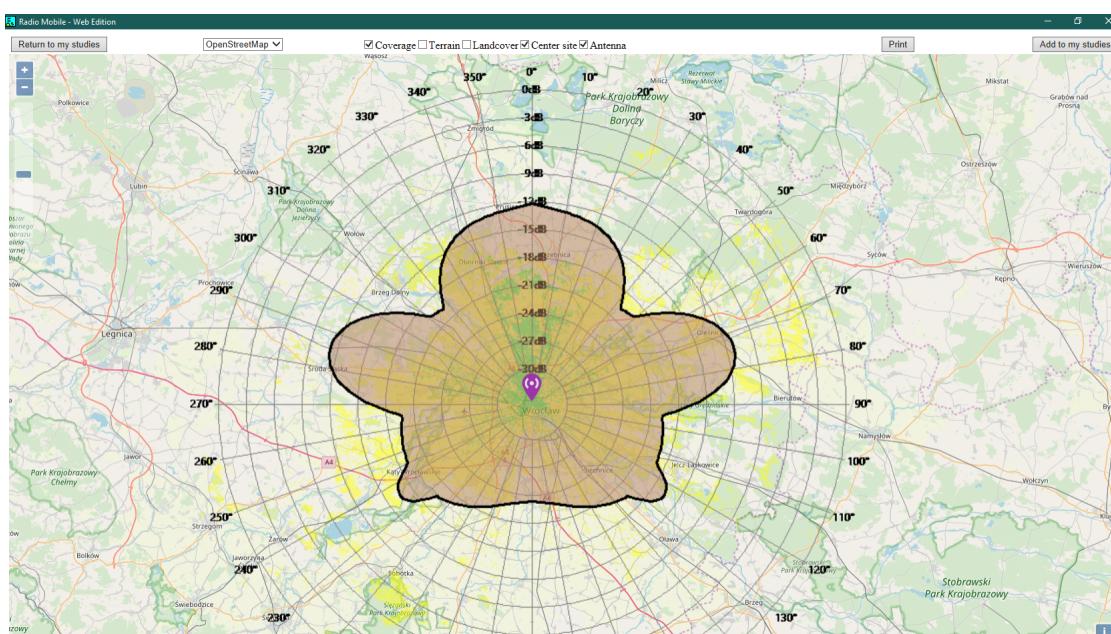
Antenę umieszczono przy WSIZ Copernicus.

Pomarańczowym kolorem przedstawiono charakterystykę zysku promieniowania anteny, które w przypadku kąta 0 stopni jest wydłużone oraz węższe, co pozwala pokryć śladowym zasięgiem nawet dalekie obszary, lecz do terenów bliżej zlokalizowanych dostarczony jest słabszy sygnał względem tego, który można otrzymać, zmieniając elektryczny kąt anteny na 40 stopni. Dzięki RET-owi, można skoncentrować wiązkę na mniejszym obszarze, oferując znacznie wyższe prędkości transmisji danych.

2. RET



Rysunek 2.4. Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 0 stopni.
(Opracowanie własne przy pomocy programu Radio Mobile)



Rysunek 2.5. Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 40 stopni.
(Opracowanie własne przy pomocy programu Radio Mobile)

3. Protokół AISG 2.0

Symulator sterownika ma za zadanie odebrać od użytkownika polecenie w postaci komendy wieloargumentowej, która zawierała będzie nazwę procedury rozpoznawanej przez odpowiednie warstwy protokołu komunikacyjnego *AISG v2.0* realizującego podzbiór *modelu OSI*[11].

3.1 Warstwy

3.1.1 1-sza - Fizyczna

Realizacja pracy w kierunku emulacji fizycznego połączenia do urządzenia niesie ze sobą pewne zmiany na tej warstwie.

Należy pominąć obszar mechaniczny i elektryczny[7], a skupić się na obszarze funkcjonalnym oraz proceduralnym.

3.1.2 2-ga - Łącza danych

Rola tej warstwy jest:

- Enkapsulacja[6] ramki *HDLC Body* do ramki *HDLC*[8]
 - Dodanie *bitów startu i stopu*;
 - Obliczenie oraz walidacja sumy *CRC*[4]
- Budowa ramki typu *XID* podczas procedur negocjacji:
 - Rozmiaru ramki;
 - Unikalnego identyfikatora urządzenia, na które składa się numer seryjny oraz kod producenta;
 - Wersji *3GPP* oraz *AISG* urządzenia podległego;
 - Adresu;
- Budowa ramki typu *U* w celu ustanowienia normalnego trybu komunikacji;

3.1.3 7-ma - Aplikacji

- Budowa ramki typu *I*;
- Rozpoznawanie wysokopoziomowej komendy kalibruj;

3.2 HDLC

Z języka angielskiego High-Level Data Link Control. Protokół warstwy łącza danych modelu OSI. Standard HDLC opisuje norma ISO, lecz szeroko stosuje się także implementację CISCO. HDLC jest stosowany w technologii WAN, ponieważ obsługuje zarówno połączenia dwupunktowe, jak i wielopunktowe. Jest protokołem o orientacji bitowej oraz jest przezroczysty informacyjnie.[9] W przypadku, jeśli przesyłana wartość jest wielobajtowa, zastosowane jest podejście *little endian*. Głównym powodem, dla którego protokół utworzony w roku 1979 roku wciąż znajduje zastosowanie jest fakt, że jego implementacja pozwala w maksymalnym stopniu wyeliminować możliwość utraty przesyłanej informacji dzięki mechanizmowi validacji sumy *CRC* oraz algorytmowi ewaluacji *bajtu kontrolnego*. Dzięki temu urządzenie nadzędne może nawet żądać ponownego przesłania wcześniej otrzymanej wiadomości.

3.2.1 Struktura ramki

1. Flaga startu - 0x7E;
2. Adres stacji docelowej;
3. Sterowanie - określa typ ramki oraz jej parametry w zależności od typu;
4. Dane;
5. Suma kontrolna FCS (dwubajtowa) - na przykład CRC-16;
6. Flaga stopu - 0x7E;

3.2.2 Typy ramek

- Ramka I - Informacyjna;
- Ramka U - Nienumerowana;
- Ramka S - Nadzorująca;
- Ramka XID - Identyfikująca urządzenia;

3.3 Typy ramek HDLC

3.3.1 Ramka XID

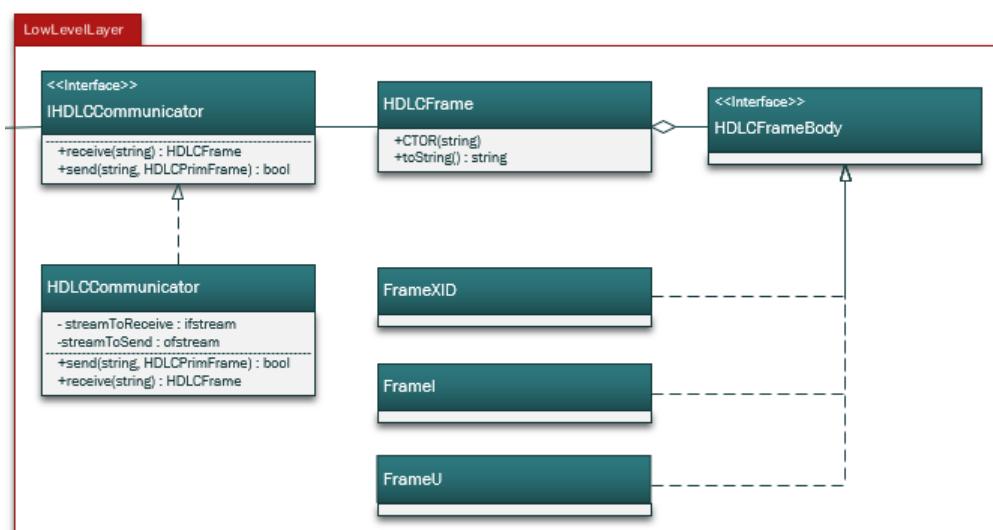
Jej nazwa pochodzi z języka angielskiego „exchange identification”. Służy do przekazania urządzeniu podlegającemu wiedzy na temat możliwości oraz charakterystyki komunikacji na warstwie łączącej danych.

Odpowiadając na tego typu wiadomości, urządzenie podlegające najczęściej zwraca tę samą wartość w przypadku zgodności, bądź najwyższą wspieraną, jeśli żądana jest zbyt duża. Szereg wysłanych i odebranych wiadomości XID nazywamy XID negocjacją. Tej ramki użyto również podczas ustalania prędkości komunikacji, co jest częścią procedury zestawienia warstwy fizycznej. Bajt kontrolny dla wiadomości przesyłanej ma zawsze wartość 0xBF.

Bajty budujące ramkę:

- Adresu;
- Kontrolny;
- Identyfikujący format;
- Identyfikujący grupę;
- Długości grupy;
- Parametrów HDLC:
 - Identyfikujący parametr;
 - Długości parametru;
 - Wartości parametru;

Wspomniano o parametrach HDLC, gdyż wiadomość negocjującą ich wartości może zawierać zarówno jeden jak i więcej parametrów.



Rysunek 3.1. Diagram klas przedstawiający relację pomiędzy abstrakcyjną oraz konkretną klasą poszczególnej ramki.

(Opracowanie własne)

3.3.2 Ramka U

Jej nazwa pochodzi z języka angielskiego „Unnumbered” co oznacza nienumerowana. Wzięło się to z tego, że wielokrotnie wysłana wiadomość tego typu, zawsze posiada tą samą wartość bajtu kontrolnego. Służy ona do zarządzania warstwą łącza danych, a czasami również do przesyłania pewnych informacji.

Bajty budujące ramkę:

- Adresu;
- Kontrolny;

Poszczególne wiadomości przesyłane przy pomocy tej ramki identyfikowane są dzięki charakterystycznej wartości bajtu kontrolnego.

3.3.3 Ramka I

Jej nazwa pochodzi z języka angielskiego „Information” co oznacza informacyjna. Dzięki niej można żądać od urządzenia podległego wykonania wysokopoziomowej operacji zdefiniowanej przez warstwę aplikacyjną, a nawet przesłać najnowszą wersję oprogramowania urządzenia. Jej długość definiowana jest podczas XID negocjacji w trakcie zestawiania warstwy łącza danych.

Bajty budujące ramkę:

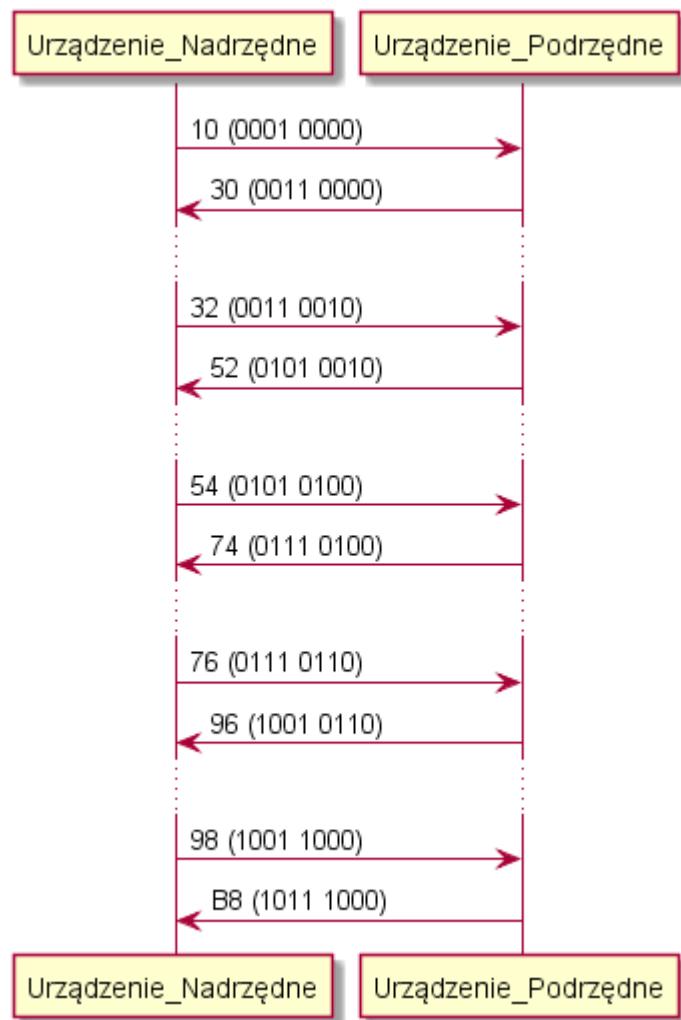
- Adresu;
- Kontrolny;
- Kodu procedury;
- Dodatkowe wartości procedury/raportujące proces wykonania procedury;

Wiadomości tego typu przy wielokrotnym zwołaniu żądania wykonania tej samej procedury posiadają zmienną wartość bajtu kontrolnego, co różni ją od ramki U czy XID.

Proces ten w przypadku bitu P/F zlokalizowanego na pozycji o indeksie 4 bajtu kontrolnego:

- Poinformowania urządzenia podległego, że urządzenie nadzorcze żąda odpowiedzi na właśnie przesyłaną wiadomość;
- Poinformowania urządzenia nadzorcze o zakończeniu nadawania odpowiedzi na wiadomość;

Wartość bitu na pozycji 0 jest zawsze równa 0. Bit od piątego do siódmego włącznie definiują nam licznik wiadomości odebranych przez urządzenie nadzorcze, czyli $N(R)$. Bit od pierwszego do trzeciego włącznie definiują nam licznik wiadomości wysłanych przez urządzenie nadzorcze, czyli $N(S)$. Na poniższym diagramie przedstawiono ewaluację bajtu kontrolnego dla kolejnych wiadomości warstwy aplikacyjnej.



Rysunek 3.2. Ramka informacyjna - ewaluacja bajtu kontrolnego.
(Opracowanie własne)

4. Dobre praktyki programowania obiektowego

Projektując sterownik, podjęto próbę podążania zgodnie z dobrymi praktykami programowania, powszechnie znanyymi jako SOLID. Pomyślna ich realizacja sprawia, że korzystanie, rozbudowywanie, jak i utrzymywanie powstałego kodu źródłowego przypomina przyjemne poruszanie się po świecie gry aniżeli walkę z napotkałymi przeszkodami. Tak powstały program umożliwia łatwe ponowne używanie wcześniej powstałego kodu. Poniższe reguły czasami bywają trudne w realizacji, zwłaszcza kiedy tworzony system osiągnie duże rozmiary, dlatego warto myśleć o nich już na początkowym etapie. Trzeba też pamiętać o tym, że nie jest rozsądne stosowanie którejkolwiek z reguł SOLID, jeśli nie ma ku temu wyraźnych powodów, dlatego jedynie doświadczenie oraz intuicja pozwoli nam wyczuć moment podjęcia decyzji projektowej. Poniżej przedstawiono prawa budujące zasadę SOLID wraz z ich głównymi założeniami.

4.1 Zasada pojedynczej odpowiedzialności

Z angielskiego Single Responsibility Principle w skrócie SRP definiuje, że powód modyfikacji klasy powinien być tylko jeden. [15, 103] Realizacja tej reguły uchroni nas przed niechcianą modyfikacją definicji dużej liczby klas w przypadku zmiany logiki tylko jednej z nich.

4.2 Zasada otwarте-zamknięte

Z angielskiego Open/Closed Principle w skrócie OCP definiuje, że encje programowanie (klasy, moduły, funkcje itp.) powinny być otwarcie na rozbudowę, ale zamknięte dla modyfikacji. [15, 117] Jeśli ta zasada zostanie właściwie zastosowana, to nowe zmiany uzyskuje się poprzez dodanie nowego kodu, aniżeli zmiane istniejącego.

4.3 Zasada podstawiania Liskov

Z angielskiego Liskov Substitution Principle w skrócie LSP definiuje, że musi być możliwość podstawienia typów pochodnych za ich typy bazowe. [15, 127] Trzeba pamiętać o tej regule w trakcie definiowania relacji dziedziczenia pomiędzy klasami.

4.4 Zasada segregacji interfejsów

Z angielskiego Interface Segregation Principle w skrócie ISP, definiuje że klienci nie powinny być zmuszone do zależności od metod, których nie używają. [15, 151] Dzięki postępowaniu

zgodnie z tą zasadą, utworzone interfejsy abstrakcyjnych klas bazowych będą zawierały minimalną liczbę funkcji czysto wirtualnych.

4.5 Zasada odwrócenia zależności

Z angielskiego Dependency Inversion Principle w skrócie DIP definiuje, że moduły wykonywane nie powinny zależeć od modułów niskopoziomowych. I jedne, i drugie powinny zależeć od abstrakcji. [15, 141] Kolejnym postulatem jest to, że abstrakcje nie powinny zależeć od szczegółów. To szczegóły powinny zależeć od abstrakcji. [15, 141]

Część Praktyczna

5. Wzorce projektowe

W latach, kiedy programowanie obiektowe stawało się bardziej popularne, wiele osób natrafiało na grupę problemów pewnej kategorii. Podczas wielokrotnych prób ich rozwiązywania, różne osoby dochodziły do wspólnych wniosków odnośnie do architektury kodu źródłowego tworzonego w celu ich rozwiązań. Napotykane wyzwania rozpoczęto dzielić według trzech kategorii: behawioralne, strukturalne oraz kreacyjne. Tak oto to powstały wzorce projektowe. Obszerna znajomość tej niemałej dziedziny informatyki pozwala spojrzeć na kolejne zadania w znacznie bardziej zaawansowany sposób aniżeli wcześniej. Można je porównać do znanych przekształceń oraz wzorów matematycznych w rachunku całkowym, gdyż zostały one ściśle zdefiniowane, a osiągniecie efektu końcowego różni się zastosowanie innych parametrów wejściowych. Podczas tworzenia wzorców usilnie trzymano się zbioru kolejnych zasad znanych pod nazwą SOLID.

Poniżej przedstawiono użyte podczas projektowania sterownika wzorce wraz z fragmentami kodu źródłowego.

5.1 Behawioralne

5.1.1 Komenda

Dzięki utworzeniu interfejsu komendy, zrealizowano regułę otwarcia na modyfikacje a zamknięcia na zmiany. W przypadku poprawnej definicji funkcjonalności, która powinna zostać zrealizowana, dodajemy jedynie implementację tego interfejsu, przez co cała regresja pozostaje bez zmian, a dzięki kontrolerowi, który kolejkuje komendy można pokusić się o realizację kompozytu oraz wprowadzenie systemu wag bądź poleceń terminujących aktualnie zaplanowane zadania[14]. Podczas implementacji sterownika zastosowano połączenie trzech wzorców projektowych a to wszystko dzięki późnemu podejściu do organizacji struktury kodu jakim jest użycie komend. Zestawiając implementację interfejsu komendy oraz budowania konkretnej wiadomości dzięki fabryce pozwala na zdefiniowanie *klas finalnych* zawierających charakterystyczne dla każdej z komend wywołan.

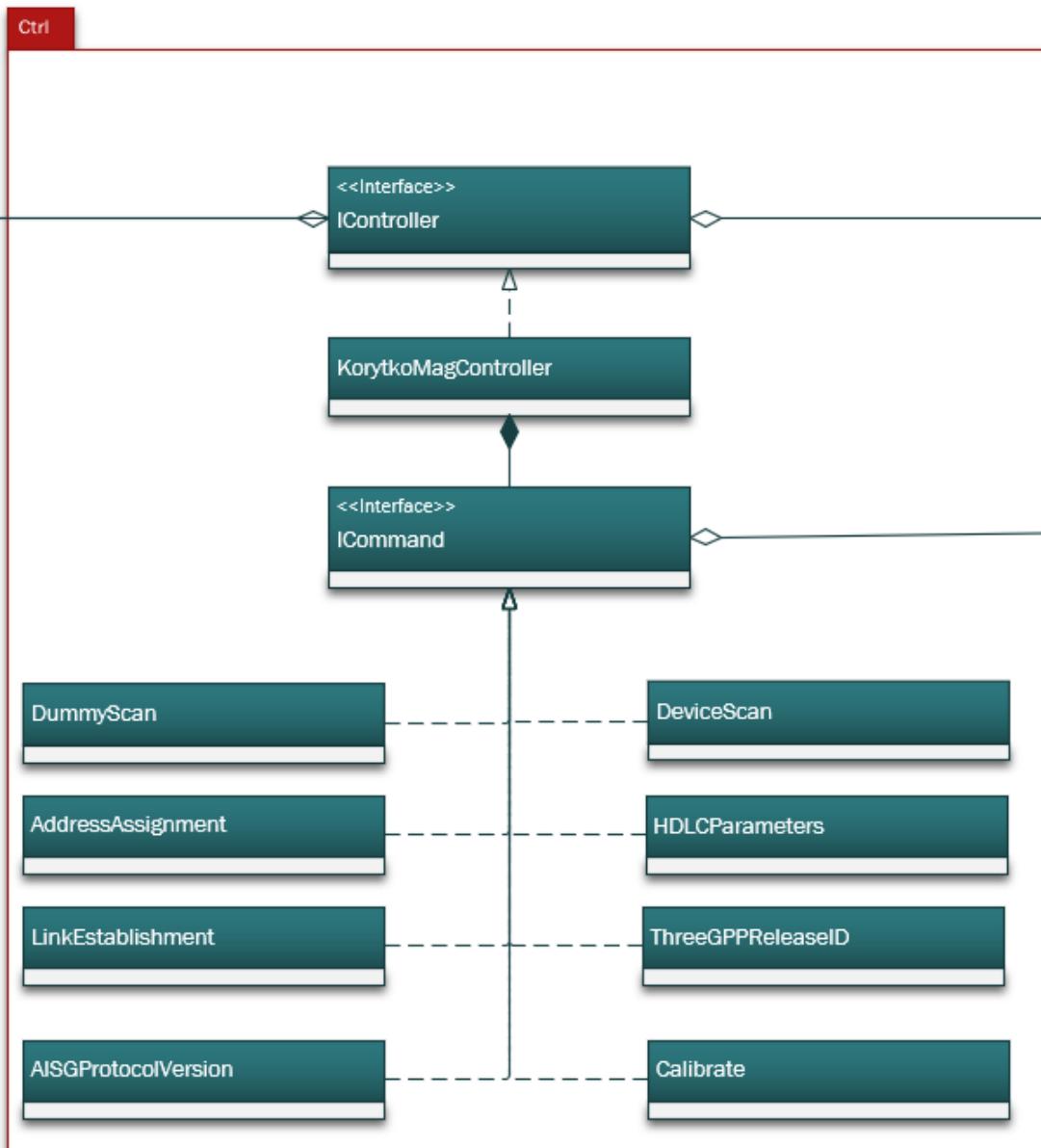
5. Wzorce projektowe

```
1 class ICommand
2 {
3 public:
4     virtual void execute() = 0;
5     virtual std::string handleResponse() = 0;
6     void registerResponseHandler(std::function<void(void)> responseHandler);
7     virtual ~ICommand();
8 protected:
9     ICommand();
10    using AlmagControllerInformer = boost::signals2::signal<void(void)>;
11    AlmagControllerInformer informControllerAboutResult_;
12 };
```

Listing 5.1. Interfejs komendy

```
1 void DeviceScan::execute()
2 {
3     executeImpl();
4     informControllerAboutResult_();
5 }
6
7 HDLCFrameBodyPtr DeviceScan::getFrameBody() const
8 {
9     return hdlcFrameBodyFactory_->get_FrameXID_DeviceScan();
10 }
11
12 void DeviceScan::executeImpl()
13 {
14     hdlcCommunicator_->communicate(validatedUserInput_[IDX_OF_ADDRESS_],
15                                         getFrameBody());
16 }
17
18 std::string DeviceScan::handleResponse()
19 {
20     return constraints::almag::L2::DEVICE_SCAN + DELIMITER;
21 }
```

Listing 5.2. Definicja klasy konkretnej komendy używającej fabryki oraz strategii



Rysunek 5.1. Diagram klas przedstawiający realizację wzorca komendy.
(Opracowanie własne)

5.1.2 Obiekt pusty

Znaną przypadłością w językach obiektowych jest wykonywanie dalszej akcji w zależności od stanu pewnego z obiektów. W przypadku korzystania ze wskaźników dobrą praktyką jest każdorazowa weryfikacja czy jego wartość nie jest równa *null*. Wykorzystanie klasy która implementuje interfejs kontrolera, w sposób neutralny sprawia, że zawsze bezpieczne będzie wywołanie na jej obiekcie instancjonującym którykolwiek z metod.

5. Wzorce projektowe

```
1 void AlmagControllerNull::addCommands(const StringsMatrix&
2     validatedUserInput)
3 { LOG(debug); }
4
5 bool AlmagControllerNull::executeCommand()
6 { LOG(debug); return true; }
7
8 void AlmagControllerNull::handleCommandsResult()
9 { LOG(debug); }
10
11 std::string AlmagControllerNull::getFinalresultCode()
12 { return defaultVals::FOR_STRING; }
```

Listing 5.3. Definicja klasy dla obiektu pustego

```
1 ReturnCode CMenu::interpretControllerCommand(const Strings& userInput)
2 {
3     LOG(debug) << "Start";
4     if (const auto validatedUserInput = almagCmdValidationMgr_->perform(
5         userInput))
6     {
7         almagCtrl_->addCommands({*validatedUserInput});
8         return almagCtrl_->executeCommand();
9     }
10    LOG(warning) << "Validation rejected the command";
11    return true;
12 }
```

Listing 5.4. Przykład użycia obiektu pustego

5.1.3 Metoda szablonowa

Wprowadzenie systemu walidacji argumentów podanych przez użytkownika gwarantuje bezpieczne wykonywanie akcji niskopoziomowych takich jak wysłanie wiadomości do urządzenia, bez zbędnego umieszczanie logiki weryfikacji w dalszym etapie. Zastosowanie tego wzorca umożliwi w przyszłości wdrożenie dodatkowych mechanizmów. Jednym z nich jest automatycznie pobranie długiego adresu portu na który ma zostać wysłana komenda, w przypadku podaniu klucza obiektu w bazie danych. Natomiast następnym jest funkcjonalność automatycznej korekty w przypadku zaistniałego błędu syntaktycznego we wprowadzanej przez użytkownika komendzie. Obecność komendy „execute” pozwala połączyć wywołanie powyższych funkcji za pomocą jednego polecenia[14], nie posiadając wiedzy o tym jakiego typu procedury weryfikacji sterownik będzie próbował dokonać. Powyższą funkcjonalność osiągnięto dzięki efektownemu wykorzystaniu funkcji wirtualnych.

```
1 class ICommandValidation
2 {
3     public:
4         virtual ~ICommandValidation() = default;
5         MaybeStrings execute(Strings userInput);
6     protected:
7         virtual MaybeStrings validateInputCorrectness(Strings userInput) = 0;
8         virtual MaybeStrings modifyIfRequired(Strings validatedUserInput) = 0;
9 }
```

Listing 5.5. Plik nagłówkowy dla metody szablonowej walidacji komendy

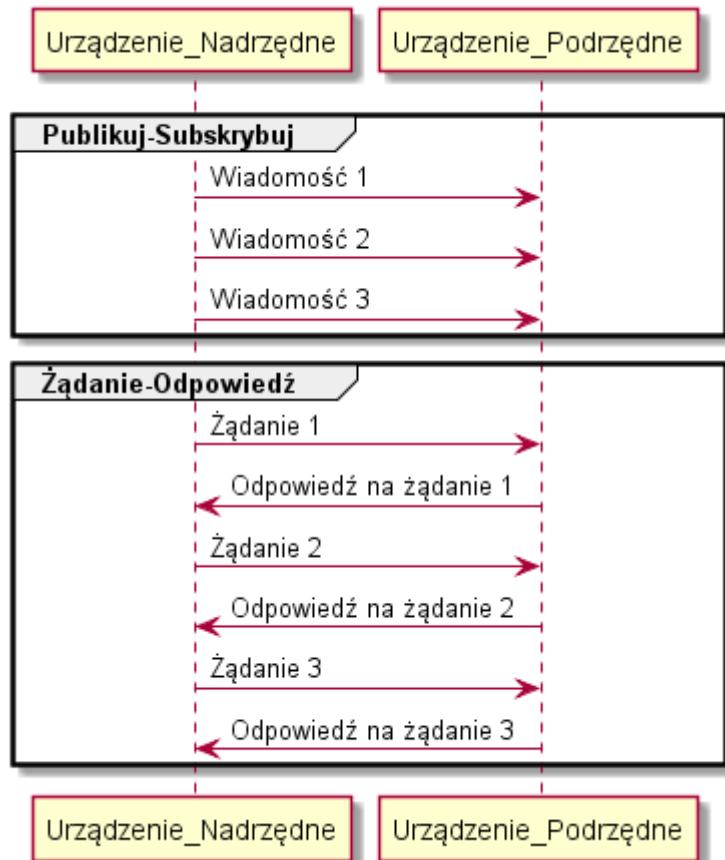
```
1 MaybeStrings ICommandValidation::execute(Strings userInput)
2 {
3     if (auto successfullyValidatedInput =
4         validateInputCorrectness(userInput))
5         return modifyIfRequired(*successfullyValidatedInput);
6     return boost::none;
7 }
```

Listing 5.6. Metoda szablonowa - Wywołanie metod wirtualnych z poziomu innej metody

5.1.4 Strategia

Znanych jest wiele wzorców komunikacji pomiędzy komponentami aczkolwiek w projekcie użyto dwa najbardziej znane: Publikuj-Subskrybuj oraz Żadanie-Odpowiedź.

Pierwszy z nich efektownie realizuje zasadę odwrócenia zależności, ponieważ sterownik urządzenia nadzorujące na początkowym etapie nie musi znać ilości podłączonych do linii urządzeń. Drugi natomiast pozwala zrealizować podejście do transmisji z urządzeniem znane jako półduplex, wymagane podczas wywoływania komend zestawiających warstwę łączącej danych oraz



Rysunek 5.2. Przepływy wiadomości dla wzorca Publikuj-Subskrybuj oraz Żądanie-Odpowiedź.
(Opracowanie własne)

aplikacyjnej. Głównym problem jest to, że obydwa wzorce wymagają innego zestawu komend w celu konfiguracji połączenia ze slotami systemowymi. W projekcie zaimplementowano jeden kontroler, który zarządza zestawianiem wymaganych warstw OSI w trakcie ciągłego uruchomienia sterownika, co osiągnięto dzięki dynamicznemu podmianie strategii komunikacji z Publikuj-Subskrybuj na Żadanie-Odpowiedź. Zaobserwowano pewne złe następstwo nieoprawnego użycia wzorca strategii polegające na prewencyjnemu rzuceniu wyjątku w przypadku gdyby programista wywołał niepoprawną metodę. W związku z tym, podczas przyszłej rozbudowy programu, zastosowany zostanie inny wzorzec o nazwie most.

```
1 void ZMqReqRepPrimaryStrategy::setupSend(const std::string& address)
2 {
3     tcpPortAddress = tcpPortAddressHeader + address;
4     socket_.connect(tcpPortAddress);
5 }
6
7 void ZMqReqRepPrimaryStrategy::setupReceive(const std::string& address)
8 { throw std::runtime_error("Redundant function"); }
9
10 bool ZMqReqRepPrimaryStrategy::send(const std::string &address,
11     HDLCFrameBodyPtr frame)
12 {
13     const std::string sentMessage = toString(frame->build());
14     return s_send(socket_, sentMessage);
15 }
16 HDLCFramePtr ZMqReqRepPrimaryStrategy::receive(const std::string &address
17 )
18 {
19     std::string message = s_recv(socket_);
20     auto recFrame{
21         std::make_shared<HDLCFrame>(HDLCFrameBodyInterpreter().apply(
22             message));
23     }
24     return recFrame;
25 }
26 HDLCFramePtr ZMqReqRepPrimaryStrategy::communicate(const std::string&
27     address, HDLCFrameBodyPtr frame)
28 {
29     send(tcpPortAddress, frame);
30     std::this_thread::sleep_for(1s);
31     receive(tcpPortAddress);
32     return nullptr;
33 }
```

Listing 5.7. Strategia komunikacji Żadanie-Odpowiedź dla urządzenia nadziednego

5.1.5 Wstrzykiwanie zależności

Klasa „HDLCCCommand” bezpośrednio dziedziczy po klasie „Command”. Dzięki implementacji interfejsu „execute” zrealizowany kontroler posiada możliwość, przyszłej rozbudowy nawet o komendy typu „włącz muzykę”. W przypadku obsługi urządzenia implementującego protokół AISG, zaobserwowano zapotrzebowanie na dodatkową klasę abstrakcyjną, która posiadała będzie wskaźnik na obiekt implementujący interfejs fabryki ramek oraz wzorca komunikacji. Podejście programowania sterowanego testami umożliwiło przedstawienie programisty przed koniecznością przekazania powyższych obiektów z poziomu testu, w celu wyeliminowania wymagania podłączania fizycznego urządzenia bądź uruchamiania aplikacji symulującej urządzenie oraz skrócenia czasu regresji programu, dzięki korzystaniu z atrap. Ten sposób zarządzania kolejnością tworzenia oprogramowania naturalnie wymusił realizację wstrzykiwania zależności polegającej na przekazywaniu obiektów z zewnątrz podczas wywoływanego konstruktora, aniżeli utworzeniu konstruktora zeroparametrowego, który uniemożliwia dalszą modernizację elementów składowych systemu.

```
1 HDLCCCommand(
2     IHDLCFrameBodyFactoryPtr frameBodyFactoryPtr,
3     IHDLCCommunicatorPtr hdlcCommunicator,
4     Strings userInput
5 );
```

Listing 5.8. Konstruktor zrealizowany podejściem wstrzykiwania zależności

5.1.6 Inicjowanie przy pozyskaniu zasobu

Do implementacji pracy użyto kompilatora GCC wspierającego język C++ w wersji 11-tej, który posiada wbudowany mechanizm inteligentnych wskaźników. „shared_ptr<T>” oraz „unique_ptr<T>” zmieniły sposób korzystania z dynamicznie alokowanej pamięci w sposób diametralny. Do tej pory dealokacja pamięci wskazywanej przez wskaźnik należała do obowiązków programisty. Dzięki podejściu tzw. RAII dla „shared_ptr<T>”, w przypadku destrukcji wszystkich wskaźników odnoszących się do wcześniej zaalokowanego obszaru pamięci, kompilator przy pomocy destruktora wywołuje komendę „delete” automatycznie, dzięki czemu programista jest ustrzeżony przed niepożądanym wyciekiem pamięci.

5.2 Kreacyjne

5.2.1 Budowniczy

Często zdarza się, że obiekt klasy wymaga modernizacji wielu z jego pól a realizacja konstruktora posiadającego trzy bądź więcej parametrów, uznawana jest za niepoprawną implementację oraz antywzorzec. W tej sytuacji z pomocą pojawia się wzorzec budowniczego, który podczas wywoływania metody zmieniającej stan obiektu, dokonuje zamierzonego celu, po czym zwraca referencję na obiekt na rzecz którego została wywołana *metoda*. Umożliwia to szeregowe wywoływanie kolejnych modyfikatorów co znacznie oczyściło i zwiększyło czytelność programu. Istnieje wiele interpretacji tego wzorca projektowego, w których dodana jest metoda finalizująca budowanie obiektu[14].

```
1 HDLCFrameBodyPtr HDLCReqFrameBodyFactory::get_FrameI_Calibrate() const
2 {
3     const auto retFrame = FrameI()
4         .setAddressByte(0x03)
5         .setControlByte(frameI::BYTE_CONTROL::CALIBRATE_REQ)
6         .setProcedureCode(PROCEDURE_CODE::CALIBRATE_SRET)
7         .setParameterLength(Hexes{ZERO, ZERO});
8     return std::make_shared<FrameI>(retFrame);
9 }
```

Listing 5.9. Budowniczy wraz z Fluent API podczas budowania ramki I - Kalibruj

5.2.2 Fabryka

Istnieje pewien wzorzec, który owiany jest złą sławą. Programiści w momencie usłyszenia o nim dostają ciarek, gdyż sądzą, że pod tym słowem, kryje się obiekt, który potrafi zachować się w nieprzewidziany sposób podczas każdorazowego wywoływanego jego metod. Z drugiej strony poprawna jego realizacja, umożliwia dynamiczną zmianę wartości zwracanych przed system, a w przypadku sterownika dała możliwość uwspólnienie kodu wraz z symulatorem urządzenia, na poziomie 90%. Mowa o fabryce[14]. W przypadku nieprzechowywania jakiegokolwiek stanu w jej instancji oraz zapoznania się w całości z realizowanym problemem komunikacji pomiędzy urządzeniem podrzędnym i nadzorowanym, prawdą jest to, że zaledwie na jedną komendę sterownik nie oczekuje odpowiedzi, a odpowiednie nazwanie metod interfejsu fabryki pozwoli zaobserwować, że po obu stronach trzeba obsłużyć komunikację oraz budowę wiadomości charakterystyczną na przykład dla polecenia „kalibruj” wprowadzając jedynie niewielkie modyfikacje.

```
1 class HDLCReqFrameBodyFactory : public IH DLCFrameBodyFactory
2 {
3     public:
4         HDLCFrameBodyPtr get_FrameI_Calibrate() const override;
5         HDLCFrameBodyPtr get_FrameU_LinkEstablishment() const override;
6         HDLCFrameBodyPtr get_FrameXID_3GPPReleaseId() const override;
7         HDLCFrameBodyPtr get_FrameXID_AddressAssignment() const override;
8         HDLCFrameBodyPtr get_FrameXID_AISGProtocolVersion() const override;
9         HDLCFrameBodyPtr get_FrameXID_DeviceScan() const override;
10        HDLCFrameBodyPtr get_FrameXID_DummyScan() const override;
11        HDLCFrameBodyPtr get_FrameXID_HDLCParameters() const override;
12        virtual ~HDLCReqFrameBodyFactory() = default;
13 }
```

Listing 5.10. Fabryka budowniczych dla sterownika urządzenia nadzorowanego

6. Zewnętrzne biblioteki użyte w projekcie

6.0.1 ZeroMQ

Jest to biblioteka, która umożliwia komunikację pomiędzy rozłącznymi komponentami, przy jednego wielu zdefiniowanych wzorców wymiany wiadomości. Jej implementacja została wykonana dla wielu różnych języków programowania, w tym dla C++. Realizacja wymiany wiadomości opiera się o ułatwienie korzystania z socketów, dzięki użyciu mechanizmu RAII. Połączenie pomiędzy programami może zostać zrealizowane przy pomocy różnych protokołów jak takich jak: TCP, UDP czy IPC.

6.0.2 GTest

Biblioteka służąca do tworzenia testów jednostkowych napisanych w języku C++. Umożliwia definiowanie testów zarówno pojedynczych jak i parametrycznych, które cechują się jednym ciałem testowym dla wielu argumentów wejściowych i wyjściowych. Oparta jest na mechanizmie makr. Dodatkową funkcjonalnością jest możliwość definicji dodatkowych funkcji, tzw. matcher-ów. Wspomagają one implementację asercji czy oczekiwania wobec obliczeń produkujących wyniki, w postaci zdefiniowanych przez użytkownika złożonych typów danych oraz kolekcji.

6.0.3 CMake

Wieloplatformowe narzędzie do automatycznego zarządzania procesem kompilacji programu. Jego główną cechą to niezależność od używanego kompilatora oraz platformy sprzętowej. CMake nie kompiluje programu samodzielnie, lecz tworzy pliki z regułami kompilacji dla konkretnego środowiska, np. w systemie GNU/Linux - Makefile. [3]

6.0.4 Boost

Zbiór klas zdefiniowanych w przestrzeni nazw „boost::” utworzonych oraz przeznaczonych do uruchamiania wraz z programami napisanymi w języku C++. Kod zdefiniowanych w nich często jest kandydatem aby dołączyć do biblioteki standardowej „std::”. Biblioteki zdefiniowane w bibliotece boost, licznie korzystają z mechanizmu metaprogramowania, co pozwala w łatwy sposób zintegrować je wraz z własnym kodem źródłowym. Klasy wykorzystane w projekcie to:

1. boost::optional - służy do przechowywania wartości zmiennej, która może być, lecz nie musi zostać zainicjalizowana. Korzysta się z niej też w przypadku, kiedy wartość zdefiniowana przez konstruktor domyślny danego typu (np. 0 dla zmiennej typu int), nie może zostać użyta jako wartość początkowa, gdyż ma już Ona znaczenie zdefiniowane przez specyfikację projektu.
2. boost::log - umożliwia wprowadzenie do projektu mechanizmu logowania wraz ze zdefiniowanymi priorytetami wiadomości; Dodatkową opcją jest też zdefiniowanie własnego nagłówka logowania oraz możliwość zapisania logów do pliku tekstowego;
3. boost::tokenizer - zawiera algorytmy służące do podziału łańcucha znaków na mniejsze tokeny, dzięki zdefiniowaniu; znaku separującego poszczególne frazy;
4. boost::algorithm - wspomaga oraz ułatwia korzystanie z kolekcji zdefiniowanych w bibliotece „std::”;

7. Wymagania funkcjonalne wraz z komendami je realizującymi

7.1 Warstwa fizyczna

7.1.1 Ustanowienie prędkości połączenia - SetLinkSpeed

Jako użytkownik chcę ustanowić prędkość transmisji na 9.6 kbps.

7.2 Warstwa łącza danych

7.2.1 Negocjacja roli - AddressAssignment

Jako użytkownik chcę zaadresować urządzenie podzielone typu SingleRET adresem 3.

7.2.2 Negocjacja parametrów HDLC - HDLCParameters

Jako użytkownik chcę ustalić maksymalny rozmiar nadawanej jak i odbieranej ramki typu I, oraz maksymalną liczbę ramek jednocześnie możliwych do wysłania oraz odebrania.

7.2.3 Ustanowienie normalnego trybu odpowiedzi - LinkEstablishment

Jako użytkownik chcę aby ustawić normalny tryb odpowiedzi podczas komunikacji z urządzeniem podzielonym.

7.2.4 Negocjacja parametrów HDLC - 3GPPReleaseID

Jako użytkownik chcę wynegocjować parametr HDLC urządzenia podzielonego, jakim jest wspierana wersja standardu 3GPP.

7.2.5 Negocjacja parametrów HDLC - AISGProtocolVersion

Jako użytkownik chcę wynegocjować parametr HDLC urządzenia podzielonego, jakim jest wspierana wersja protokołu AISG.

7.3 Warstwa aplikacyjna

7.3.1 Kalibracja - Calibrate

Jako użytkownik chcę skalibrować urządzenie.

8. Uruchomienie programu

8.1 Konfiguracja środowiska

Napisy ujęte w cudzysłowy są komendami które należy wywołać z lini poleceń.

1. Uruchomienie systemu Manjaro Linux;
2. Podłączenie do internetu w celu pobrania repozytorium;
3. Konfiguracja środowiska (pomiń w przypadku posiadania kompletu):
 - (a) „sudo pacman –sync boost”
 - (b) „sudo pacman –sync cmake”
 - (c) „sudo pacman –sync git”

8.2 Kompilacja kodu źródłowego oraz wystartowanie programu

1. otwarcie pierwszej konsoli w celu uruchomienia symulatora urządzenia:
 - (a) „git clone --recursive https://github.com/trunksBT/KorytkoMag_RetSimulator.git”
 - (b) „cd KorytkoMag_RetSimulator”
 - (c) „cmake .”
 - (d) „bash runBinary.sh”
2. otwarcie drugiej konsoli w celu uruchomienia sterownika:
 - (a) „git clone --recursive https://github.com/trunksBT/KorytkoMag_RetDriverSimulator.git”
 - (b) „cd KorytkoMag_RetDriverSimulator”
 - (c) „cmake .”
 - (d) „bash runBinary.sh”
 - (e) Wywołanie komend zgodnie z diagramem sekwencji zatytułowanym „Kalibracja RETa”
 - (f) Aby zakończyć trzeba wpisać „exit”

8.3 Efekt końcowy

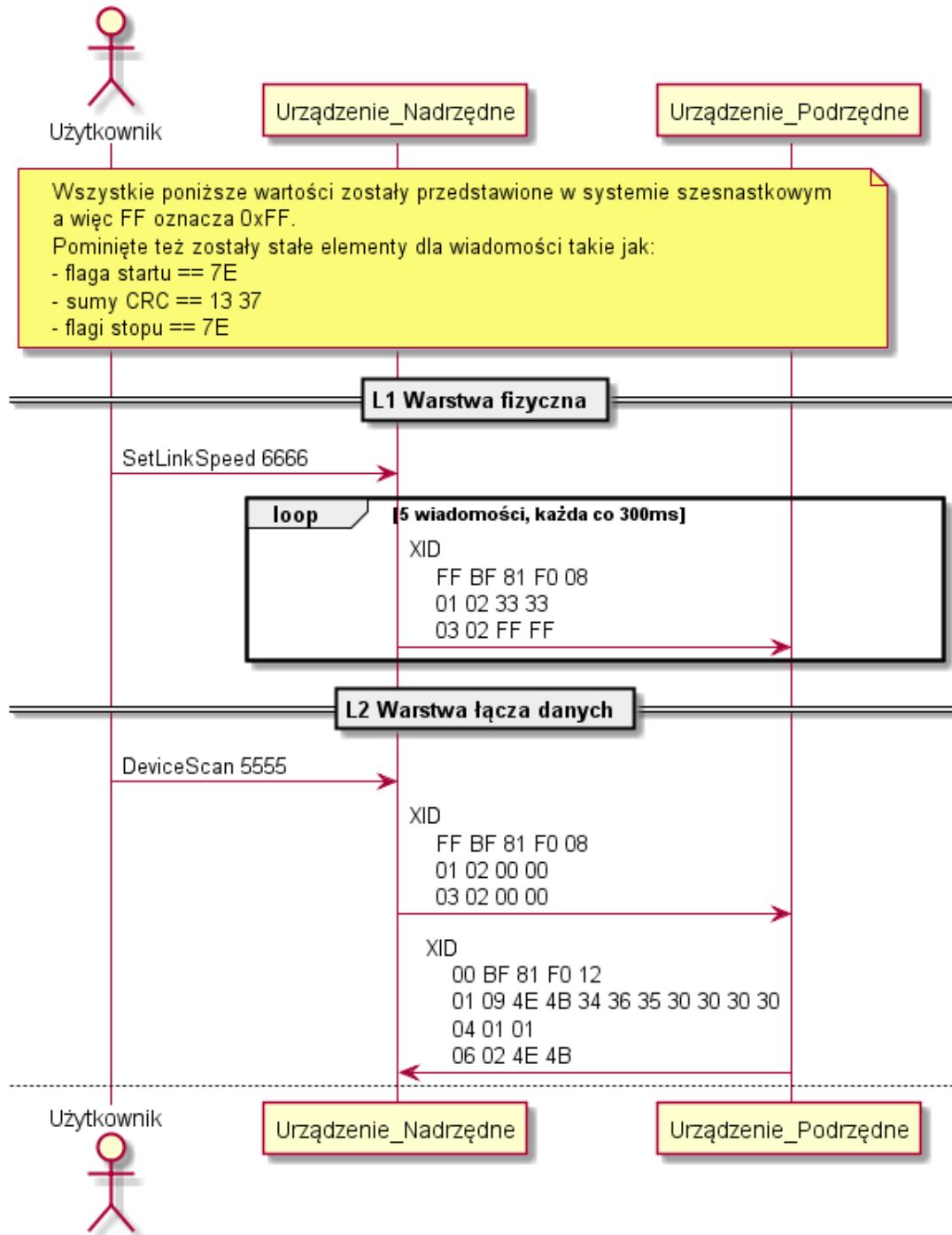
```

1 <23:14:19.498531> info [ZMqPubSubPrimaryStrategy::setupSend:22] on tcp://127.0.0.1:6666
2 <23:14:19.498859> info [ZMqReqRepPrimaryStrategy::setupSend:25] on tcp://127.0.0.1:5555
3 <23:14:19.498923> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
4 SetLinkSpeed 6666
5 <23:14:26.501229> info [DummyScan::execute:27]
6 <23:14:26.501378> info [ZMqPubSubPrimaryStrategy::send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
7 <23:14:26.812975> info [ZMqPubSubPrimaryStrategy::send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
8 <23:14:27.113751> info [ZMqPubSubPrimaryStrategy::send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
9 <23:14:27.415036> info [ZMqPubSubPrimaryStrategy::send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
10 <23:14:27.716261> info [ZMqPubSubPrimaryStrategy::send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
11 <23:14:28.022041> info [ZMqPubSubPrimaryStrategy::send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
12 <23:14:28.324300> info [ZMqPubSubPrimaryStrategy::send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
13 <23:14:28.631228> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
14 DeviceScan 5555
15 <23:15:05.546352> info [DeviceScan::execute:23]
16 <23:15:05.546552> info [ZMqReqRepPrimaryStrategy::send:37] 7e ff bf 81 f0 8 1 2 0 0 3 2 0 0 13 37 7e
17 <23:15:06.557544> debug [ZMqReqRepPrimaryStrategy::receive:46] 7e 0 bf 81 f0 12 1 9 4e 4b 34 36 35 30 30 30 30 4 1 1 6 2
    4e 4b 13 37 7e
18 <23:15:06.558102> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
19 AddressAssignment 5555
20 <23:15:15.679933> info [AddressAssignment::execute:21]
21 <23:15:15.680244> info [ZMqReqRepPrimaryStrategy::send:37] 7e ff bf 81 f0 11 1 9 4e 4b 34 36 35 30 30 30 30 2 1 3 4 1 1
    13 37 7e
22 <23:15:16.684906> debug [ZMqReqRepPrimaryStrategy::receive:46] 7e 3 bf 81 f0 12 1 9 4e 4b 34 36 35 30 30 30 30 4 1 1 6 2
    4e 4b 13 37 7e
23 <23:15:16.685376> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
24 DeviceScan 5555
25 <23:15:23.994520> info [DeviceScan::execute:23]
26 <23:15:23.994727> info [ZMqReqRepPrimaryStrategy::send:37] 7e ff bf 81 f0 8 1 2 0 0 3 2 0 0 13 37 7e
27 <23:15:25.000678> debug [ZMqReqRepPrimaryStrategy::receive:46] 7e 0 bf 81 f0 12 1 9 4e 4b 34 36 35 30 30 30 30 4 1 1 6 2
    4e 4b 13 37 7e
28 <23:15:25.000883> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
29 HDLCPParameters 5555
30 <23:15:31.159221> info [HDLCPParameters::execute:21]
31 <23:15:31.159425> info [ZMqReqRepPrimaryStrategy::send:37] 7e 3 bf 81 80 12 5 4 f0 2d 0 0 6 4 f0 2d 0 0 7 1 1 8 1 1 13
    37 7e
32 <23:15:32.168583> debug [ZMqReqRepPrimaryStrategy::receive:46] 7e 3 bf 81 80 12 5 4 f0 2 0 0 6 4 50 2 0 0 7 1 1 8 1 1 13
    37 7e
33 <23:15:32.168977> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
34 LinkEstablishment 5555
35 <23:15:38.008670> info [LinkEstablishment::execute:20]
36 <23:15:38.008757> info [ZMqReqRepPrimaryStrategy::send:37] 7e 3 93 13 37 7e
37 <23:15:39.017392> debug [ZMqReqRepPrimaryStrategy::receive:46] 7e 3 73 13 37 7e
38 <23:15:39.017504> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
39 3GPPReleaseID 5555
40 <23:15:46.348743> info [ThreeGPPReleaseID::execute:20]
41 <23:15:46.348923> info [ZMqReqRepPrimaryStrategy::send:37] 7e 3 bf 81 f0 3 5 1 8 13 37 7e
42 <23:15:47.407710> debug [ZMqReqRepPrimaryStrategy::receive:46] 7e 3 bf 81 f0 3 5 1 8 13 37 7e
43 <23:15:55.035056> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
44 AISGProtocolVersion 5555
45 <23:16:01.246151> info [AISGProtocolVersion::execute:22]
46 <23:16:01.246295> info [ZMqReqRepPrimaryStrategy::send:37] 7e 3 bf 81 f0 3 14 1 2 13 37 7e
47 <23:16:02.265720> debug [ZMqReqRepPrimaryStrategy::receive:46] 7e 3 bf 81 f0 3 14 1 2 13 37 7e
48 <23:16:02.266551> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
49 help
50 <23:16:03.940640> info [CMenu::runImpl:68] {
51 SetLinkSpeed
52 AddressAssignment
53 AISGProtocolVersion
54 DeviceScan
55 HDLCPParameters
56 LinkEstablishment
57 3GPPReleaseID
58 Calibrate
59 }
60 <23:16:03.940712> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
61 Calibrate 5555
62 <23:16:09.586609> info [Calibrate::execute:20]
63 <23:16:09.586733> info [ZMqReqRepPrimaryStrategy::send:37] 7e 3 10 31 0 0 13 37 7e
64 <23:16:10.592517> debug [ZMqReqRepPrimaryStrategy::receive:46] 7e 3 30 31 1 0 0 13 37 7e
65 <23:16:10.592664> info [UserCommandParser::receiveAndLex:34] Please pass your command ( Pascal Case )
66 exit

```

Listing 8.1. Wykonanie programu

9. Analiza nawiązanej komunikacji



Rysunek 9.1. Ustanowienie prędkości połączenia wraz z początkowym skanowaniem urządzeń
(Opracowanie własne)

W celu szczegółowej analizy wykonania programu przedstawionego na listingu 8.1, utworzono diagram sekwencji podzielony na kilka części (rysunek 9.1), co pozwoliło zobrazować zaistniałą komunikację pomiędzy użytkownikiem, urządzeniem nadrzędnym oraz urządzeniem podrzędnym. Na diagramie oraz podczas analizy pominięto cztery bajty stałe dla każdej wiadomości, czyli bajt startu równy 0x7E, dwa bajty sumy CRC = { 0x13, 0x37 } (powód dlaczego ta wartość jest stała przedstawiona w podsumowaniu) oraz bajt stopu równy 0x7E.

9.1 Ustanowienie prędkości połączenia

Parametrem tej komendy jest adres portu 6666, z którego korzysta sterownik do nawiązania połączenia typu tcp na adresie 127.0.0.1 wraz z symulatorem urządzenia przy zastosowaniu wzorca Publikuj-Subskrybuj. Podczas tego połączenia protokół AISG 2.0 nie zakłada oczekiwania na odpowiedź od urządzenia podrzędnego oraz użyta biblioteka ZeroMQ również nie udostępnia możliwości wysłania odpowiedzi na taką wiadomość.

Analiza wartości wiadomości wychodzącej (rysunek 8.1- linijki [3; 12]):

1. ADDR = 0xFF — *broadcast*;
2. CTRL = 0xBF — charakterystyczny dla ramki XID;
3. FI = 0x81 — charakterystyczna dla ramki XID oraz grupy wiadomości przypisania adresu;
4. GI = 0xF0 — grupa wiadomości przypisania adresu;
5. GL = 0x08;

Analizę kolejnych bajtów tej wiadomości pominięto, ponieważ mogą one przyjmować dowolną wartość.

9.2 Skanowanie urządzeń

Parametrem tej komendy jest adres portu 5555, z którego korzysta sterownik do nawiązania połączenia typu tcp na adresie 127.0.0.1 wraz z symulatorem urządzenia, przy zastosowaniu wzorca Żądanie-Odpowiedź. Jest to wzorzec, który zakłada otrzymanie odpowiedzi na każdą wyslaną wiadomość, zanim kolejna zostanie nadana. Opisany mechanizm komunikacji aplikowalny jest również do wszystkich poniższych komend.

Analiza wartości wiadomości wychodzącej (rysunek 8.1 - linijki 15, 16, rysunek 9.1):

1. ADDR = 0xFF — *broadcast*;
2. CTRL = 0xBF — charakterystyczny dla ramki XID;
3. FI = 0x81 — charakterystyczna dla ramki XID oraz grupy wiadomości przypisania adresu;

4. GI = 0xF0 — grupa wiadomości przypisania adresu;
5. GL = 0x08;
6. Pierwszy parametr HDLC:
 - (a) PI = 0x01 — unikalny identyfikator urządzenia podrzędnego;
 - (b) PL = 0x02;
 - (c) PV = { 0x00, 0x00 } — zestawienie tych wartości spowoduje, że każde urządzenie podłączone do portu zgłosi swoją obecność;
7. Drugi parametr HDLC:
 - (a) PI = 0x03 — maska unikalnego identyfikatora;
 - (b) PL = 0x02;
 - (c) PV = { 0x00, 0x00 } — zestawienie tych wartości spowoduje, że każde urządzenie podłączone do portu zgłosi swoją obecność;

Analiza wartości wiadomości przychodzącej (rysunek 8.1- linijka 17, rysunek 9.1):

1. ADDR = 0x00 — urządzenie podrzędne jest jest w stanie niezaadresowanym;
2. CTRL = 0xBF — charakterystyczny dla ramki XID;
3. FI = 0x81 — charakterystyczna dla ramki XID oraz grupy wiadomości przypisania adresu;
4. GI = 0xF0 — grupa wiadomości przypisania adresu;
5. GL = 0x12;
6. Pierwszy parametr HDLC:
 - (a) PI = 0x01 — unikalny identyfikator urządzenia podrzędnego;
 - (b) PL = 0x09;
 - (c) PV = { 0x4E, 0x4B, 0x34, 0x36, 0x35, 0x30, 0x30, 0x30, 0x30 };
7. Drugi parametr HDLC:
 - (a) PI = 0x04 — typ urządzenia podrzędnego;
 - (b) PL = 0x01;
 - (c) PV = 0x01 — pojedynczy RET;
8. Trzeci parametr HDLC:
 - (a) PI = 0x06 — kod producenta urządzenia podrzędnego;
 - (b) PL = 0x02;
 - (c) PV = { 0x4E, 0x4B };

9.3 Żądanie adresacji

Tutaj po raz pierwszy można zaobserwować zmienioną wartość pola adresu dla wiadomości przychodzącej. Oznacza to, że urządzenie podrzędne zaakceptowało żądanie adresacji oraz identyfikuje się w trakcie rozmowy z urządzeniem nadziednym pod adresem 0x03 co jest prawdą dla każdej następnej wiadomości.

Analiza wartości wiadomości wychodzącej (rysunek 8.1, linijki 20, 21 oraz 9.2):

1. ADDR = 0xFF — broadcast;
2. CTRL = 0xBF — charakterystyczny dla ramki XID;
3. FI = 0x81 — charakterystyczna dla ramki XID oraz grupy wiadomości przypisania adresu;
4. GI = 0xF0;
5. GL = 0x11;
6. Pierwszy parametr HDLC:
 - (a) PI = 0x01 — unikalny identyfikator urządzenia podrzecznego;
 - (b) PL = 0x09;
 - (c) PV = { 0x4E, 0x4B, 0x34, 0x36, 0x35, 0x30, 0x30, 0x30, 0x30 };
7. Drugi parametr HDLC:
 - (a) PI = 0x02 — docelowy adres urządzenia podrzecznego;
 - (b) PL = 0x01;
 - (c) PV = 0x03;
8. Trzeci parametr HDLC:
 - (a) PI = 0x04 — typ urządzenia podrzecznego;
 - (b) PL = 0x01;
 - (c) PV = 0x01 — pojedynczy RET;

Analiza wartości wiadomości przychodzącej (rysunek 8.1- linijka 22, rysunek 9.2):

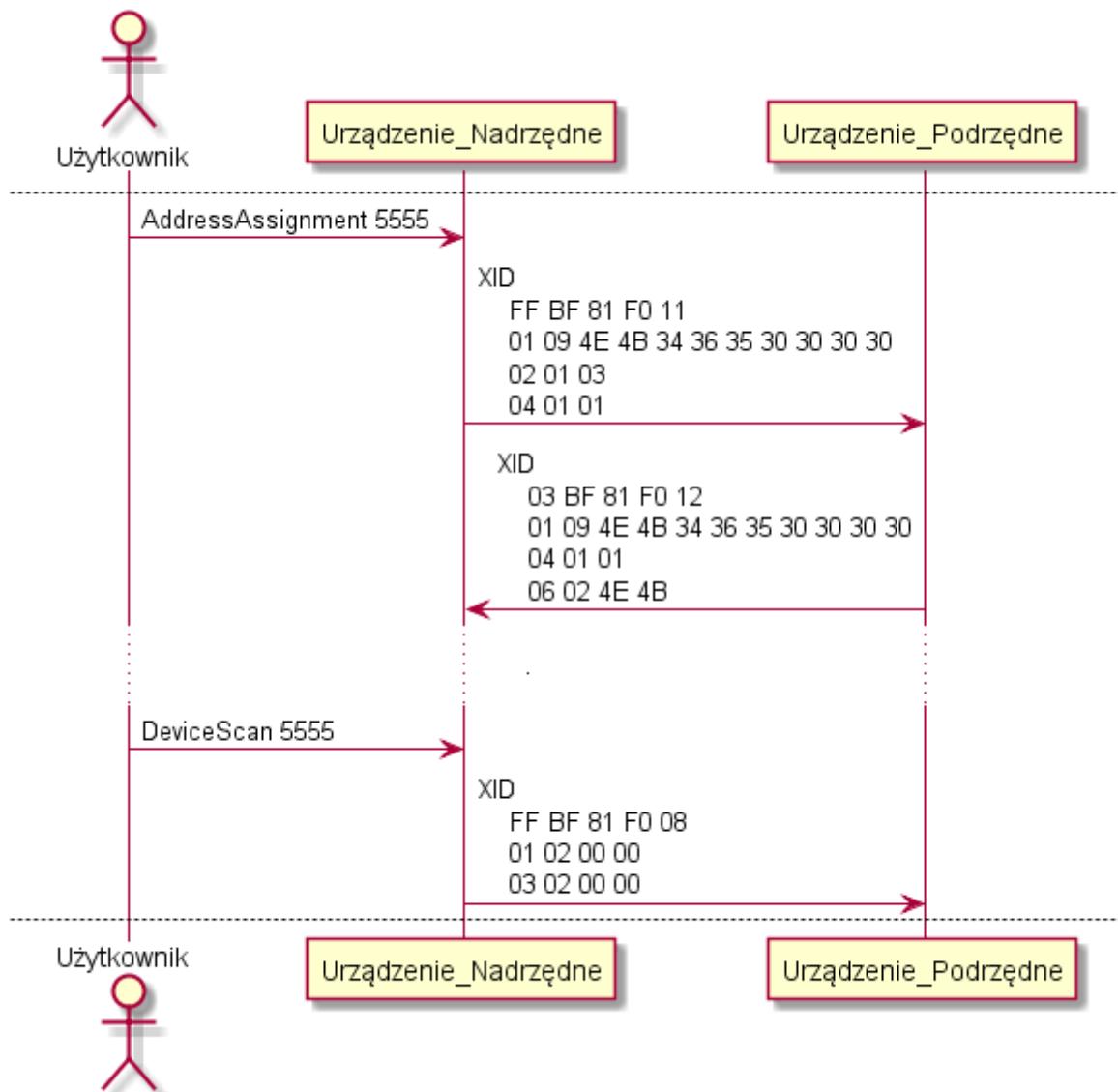
1. ADDR = 0x03 — urządzenie identyfikuje się żadanym adresem;
2. CTRL = 0xBF — charakterystyczny dla ramki XID;
3. FI = 0x81 — charakterystyczna dla ramki XID oraz grupy wiadomości przypisania adresu;
4. GI = 0xF0;
5. GL = 0x12;
6. Pierwszy parametr HDLC:
 - (a) PI = 0x01 — unikalny identyfikator urządzenia podrzecznego;
 - (b) PL = 0x09;
 - (c) PV = { 0x4E, 0x4B, 0x34, 0x36, 0x35, 0x30, 0x30, 0x30, 0x30 };
7. Drugi parametr HDLC:

9. Analiza nawiązanej komunikacji

- (a) PI = 0x04 — typ urządzenia podległego;
- (b) PL = 0x01;
- (c) PV = 0x01 — pojedynczy RET;

8. Trzeci parametr HDLC:

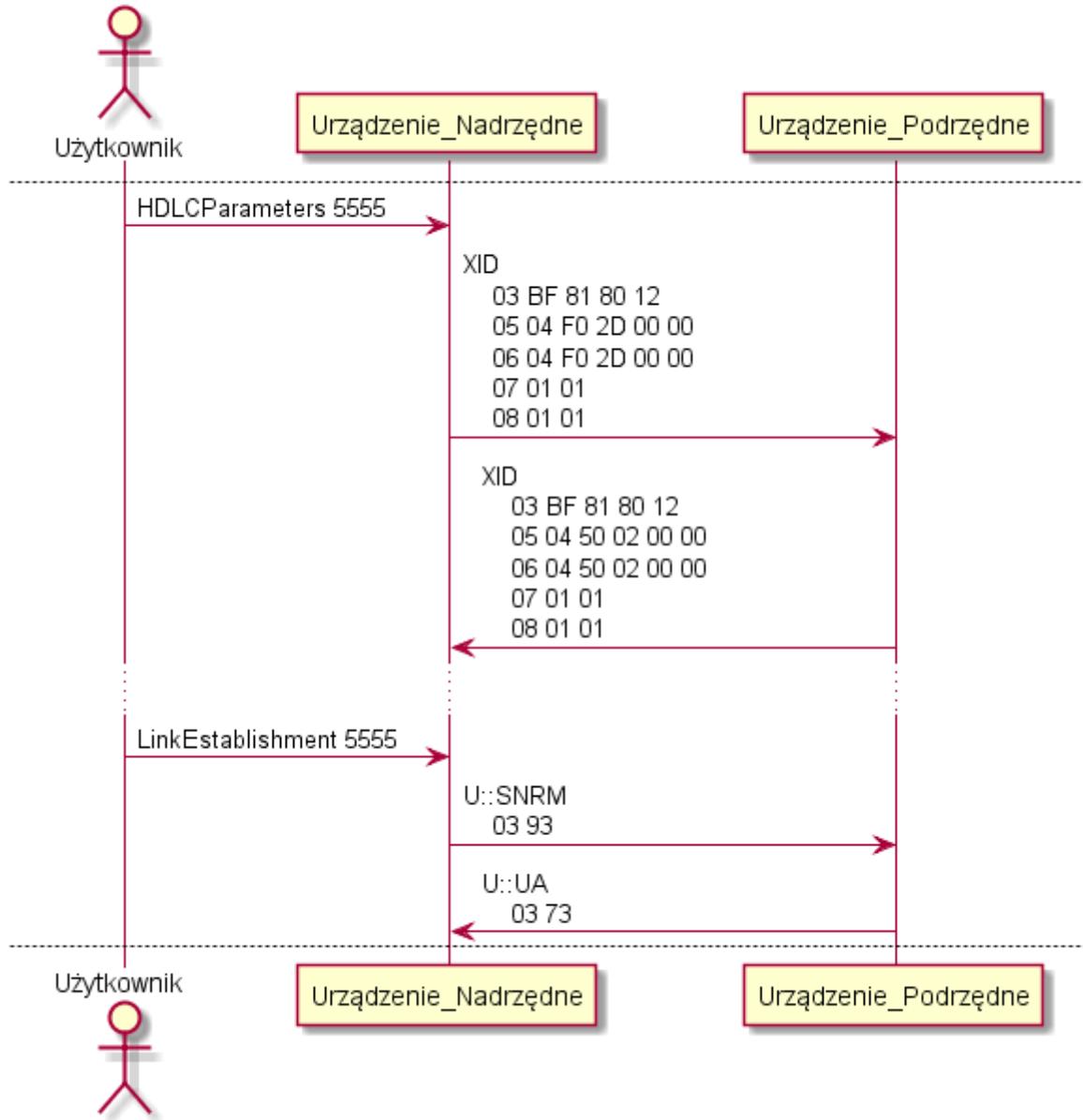
- (a) PI = 0x06 — kod producenta urządzenia podległego;
- (b) PL = 0x02;
- (c) PV = { 0x4E, 0x4B };



Rysunek 9.2. Żądanie adresacji oraz dodatkowe skanowanie urządzeń.
(Opracowanie własne)

9.4 Ponowne skanowanie urządzeń

Na rysunku 9.2 przedstawiono ponowne żądanie skanowania urządzenia na linii. Zostało to wykonane w celu upewnienia się, że poprzednia procedura adresacji powiodła się. Po zaadresowaniu urządzenia podlegającego, nie powinno ono odpowiedzieć na wiadomość typu broadcast.



Rysunek 9.3. Negocjacja rozmiaru okna oraz payloadu ramki informacyjnej oraz ustanowienie normalnego trybu komunikacji.
(Opracowanie własne)

9.5 Negocjacje parametrów HDLC

Cechą negocjacji parametrów przy pomocy ramek XID jest to, że jeśli żądana wartość jest wspierana przez urządzenie podrzędne, to odpowie ono wiadomością zawierającą te same parametry oraz te same wartości. W przeciwnym wypadku otrzymanymi wartościami parametrów będą największe możliwe przez nie wspierane. Zaobserwowano to zjawisko w przypadku negocjacji wielkości payloadu dla wysłanej oraz otrzymanej ramki informacyjnej.

Analiza wartości wiadomości wychodzącej (rysunek 8.1, linijki 30, 31 oraz 9.3):

1. ADDR = 0x03 — adres nadany urządzeniu podrzędnemu;
2. CTRL = 0xBF — charakterystyczny dla ramki XID;
3. FI = 0x81 — charakterystyczna dla ramki XID oraz grupy wiadomości przypisania adresu;
4. GI = 0xF0;
5. GL = 0x12;
6. Pierwszy parametr HDLC:
 - (a) PI = 0x05 — maksymalny rozmiar payloadu wysyłanej ramki I;
 - (b) PL = 0x04;
 - (c) PV = {0xF0, 0x2D, 0x00, 0x00} czyli 61485 bity;
7. Drugi parametr HDLC:
 - (a) PI = 0x06 — maksymalny rozmiar payloadu odbieranej ramki I;
 - (b) PL = 0x04;
 - (c) PV = {0xF0, 0x2D, 0x00, 0x00} czyli 61485 bity;
8. Trzeci parametr HDLC:
 - (a) PI = 0x07 — maksymalna liczba ramek wysłanych pod rzęd;
 - (b) PL = 0x01;
 - (c) PV = 0x01;
9. Czwarty parametr HDLC:
 - (a) PI = 0x08 — maksymalna liczba ramek odebranych pod rzęd;
 - (b) PL = 0x01;
 - (c) PV = 0x01;

Analiza wartości wiadomości przychodzącej (rysunek 8.1- linijka 32, rysunek 9.3):

1. ADDR = 0x03 — urządzenie identyfikuje się żądanym adresem;
2. CTRL = 0xBF — charakterystyczny dla ramki XID;
3. FI = 0x81 — charakterystyczna dla ramki XID oraz grupy wiadomości przypisania adresu;
4. GI = 0x80;
5. GL = 0x12;

6. Pierwszy parametr HDLC:

- (a) PI = 0x05 — maksymalny rozmiar payloadu wysyłanej ramki I;
- (b) PL = 0x04;
- (c) PV = {0x50, 0x02, 0x00, 0x00} czyli 20482 bity

7. Drugi parametr HDLC:

- (a) PI = 0x06 — maksymalny rozmiar payloadu odbieranej ramki I;
- (b) PL = 0x04;
- (c) PV = {0x50, 0x02, 0x00, 0x00} czyli 20482 bity

8. Trzeci parametr HDLC:

- (a) PI = 0x07 — maksymalna liczba ramek wysłanych;
- (b) PL = 0x01;
- (c) PV = 0x01;

9. Czwarty parametr HDLC:

- (a) PI = 0x08 — maksymalna liczba ramek odebranych;
- (b) PL = 0x01;
- (c) PV = 0x01;

9.6 Przejście na normalny tryb odpowiedzi

Normalny tryb odpowiedzi to jest taki, w którym urządzenie podrzędne nie może inicjować transmisji. Analiza wartości wiadomości wychodzącej (rysunek 8.1, linijki 35, 36 oraz 9.3):

1. ADDR = 0x03 — adres nadany urządzeniu podrzdnemu;
2. CTRL = 0x93 — charakterystyczny dla ramki U, SNRM;

Analiza wartości wiadomości przychodzącej (rysunek 8.1, linijka 37 oraz 9.3):

1. ADDR = 0x03 — adres nadany urządzeniu podrzdnemu;
2. CTRL = 0x73 — charakterystyczny dla ramki U, UA;

9.7 Wersja standardu 3GPP

Ciekawą obserwacją tej wiadomości jest fakt, że identyfikator parametru równy 0x05 pojawił się podczas definiowania długości payloadu dla ramki informacyjnej. Różnica polega na tym, że tamta wiadomość posiadała identyfikator grupy równy 0x80

Analiza wartości wiadomości wychodzącej (rysunek 8.1, linijki 40, 41 oraz 9.4):

1. ADDR = 0x03 — adres nadany urządzeniu podrzdnemu;
2. CTRL = 0xBF — charakterystyczny dla ramki XID;
3. FI = 0x81 — charakterystyczna dla ramki XID oraz grupy wiadomości przypisania adresu;

4. GI = 0xF0;
5. GL = 0x03;
6. Parametr HDLC:
 - (a) PI = 0x05 — wersja standardu 3GPP;
 - (b) PL = 0x01;
 - (c) PV = 0x08 — pionierska dla technologii LTE

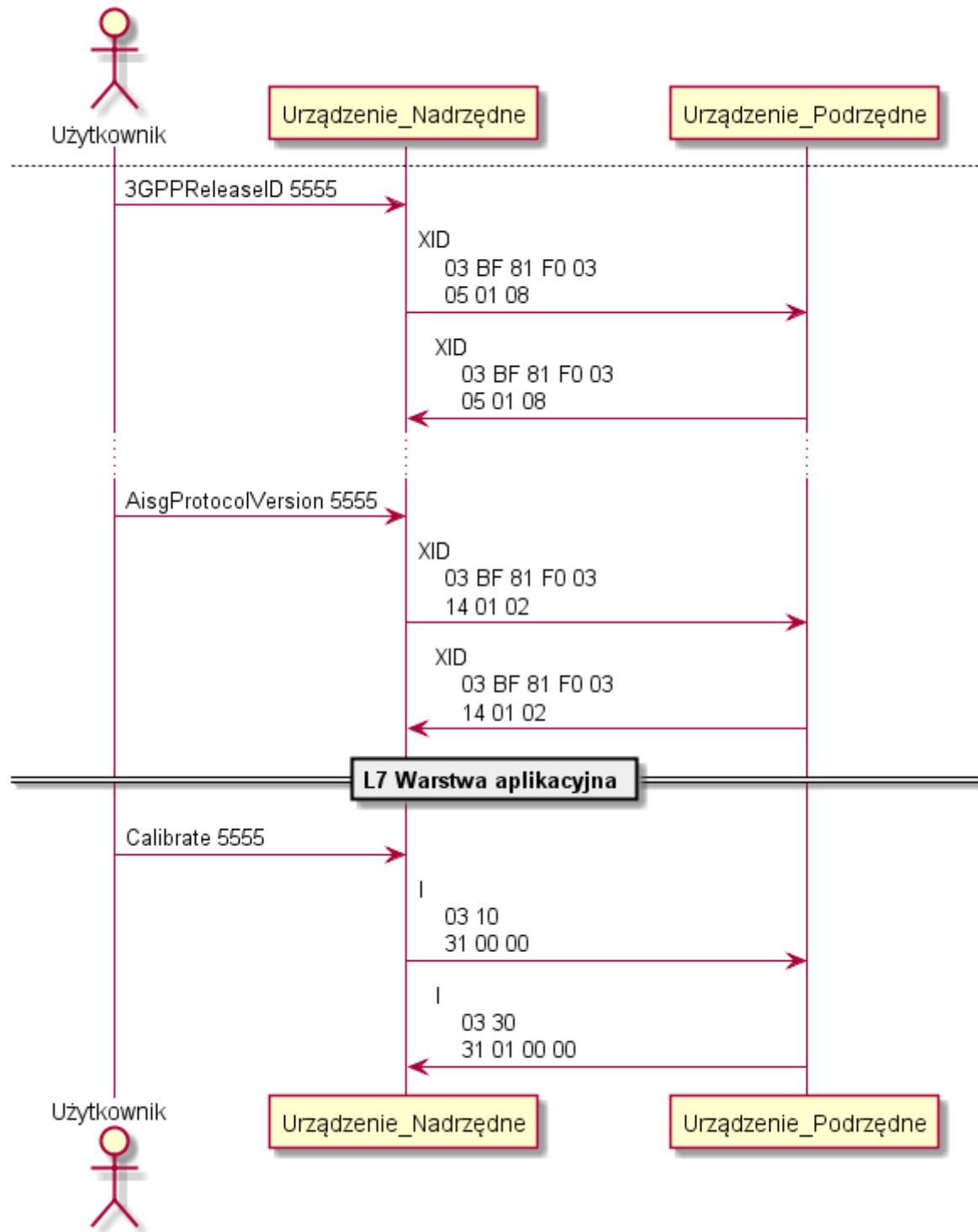
Wiadomość przychodząca posiada tę samą wartość co wychodząca, dlatego pominięto jej analizę.

9.8 Wersja protokołu AISG

Analiza wartości wiadomości wychodzącej (rysunek 8.1, linijki 45, 46 oraz 9.4):

1. ADDR = 0x03 — adres nadany urządzeniu podrzędnemu;
2. CTRL = 0xBF — charakterystyczny dla ramki XID;
3. FI = 0x81 — charakterystyczna dla ramki XID oraz grupy wiadomości przypisania adresu;
4. GI = 0xF0;
5. GL = 0x03;
6. Parametr HDLC:
 - (a) PI = 0x14 — wersja protokołu AISG;
 - (b) PL = 0x01;
 - (c) PV = 0x02 — wybrano wersję v2.0, istnieją jeszcze: 1.0, 1.1 oraz 3.0

Wiadomość przychodząca posiada tę samą wartość co wychodząca, dlatego pominięto jej analizę.



Rysunek 9.4. Negocjacja pozostałych parametrów HDLC oraz żądanie kalibracji urządzenia.
(Opracowanie własne)

9.9 Kalibracja

Kalibracja jest procedurą obowiązkową, jeśli użytkownik RET-a, planuje zmodyfikować kąt głównej wiązki. Polega ona na ustawieniu kąta anteny na wartość maksymalną wspieraną przez

9. Analiza nawiązanej komunikacji

urządzenia podrzędne, a następnie ustawieniu kąta na minimalną wspieraną wartość. Czynność ta jest czasochłonna gdyż może trwać aż do 3 minut, w związku z czym należy ją wykonać w początkowym etapie konfigurowania stacji nadawczej. Pozwala ona wykryć podstawowe błędy, które mogą pojawić się podczas próby ustawienia kąta, takie jak awaria silnika. W przypadku pominięcia tej procedury, poprawnie zaprogramowane urządzenie podrzędne, powinno zwrócić jako odpowiedź na żądanie kalibracji odpowiedni kod rezultatu. Poinformuje on o tym, że od momentu adresacji urządzenia, procedura kalibracji nie została przeprowadzona.

Analiza wartości wiadomości wychodzącej (rysunek 8.1, linijki [61; 63] oraz 9.4):

1. ADDR = 0x03 — adres nadany urządzeniu podrzdnemu;
2. CTRL = 0x10 — bajt kontrolny (rysunek 3.2)
3. PC = 0x31 — kod procedury dla kalibracji pojedynczego RET-a;
4. PL = { 0x00, 0x00 } — dwubajtowa długość dodatkowych wartości przesyłanych;

Analiza wartości wiadomości przychodzącej (rysunek 8.1, linijka 64 oraz 9.4):

1. ADDR = 0x03 — adres nadany urządzeniu podrzdnemu;
2. CTRL = 0x30 — bajt kontrolny (rysunek 3.2)
3. PC = 0x31 — kod procedury dla kalibracji pojedynczego RET-a;
4. PL = { 0x01, 0x00 } — dwubajtowa długość dodatkowych wartości przesyłanych;
5. PV = 0x00 — OK, procedura przebiegła pomyślnie;

10. Testowanie oprogramowania

Projekt zrealizowano zgodnie z zasadą TDD, którą budują trzy główne prawa.

Pierwsze prawo - nie można zacząć pisać kodu produkcyjnego do momentu napisania niespełnianego testu jednostkowego.

Drugie prawo - Nie można napisać więcej testów jednostkowych, które są wystarczające do niespełnienia testu, a brak komplikacji jest jednocześnie nieudanym testem.

Trzecie prawo - Nie można pisać większej ilości kodu produkcyjnego, niż wystarczy do spełnienia obecnie niespełnianego testu.[16]

Dzięki powyższym postulatom, projekt nie zawiera kodu nieużywanego, który wymagałby dodatkowego narzutu czasu podczas jego czytania. Prawdą jest to, że lepszy jest kod, którego nie ma, aniżeli ten który jest, aczkolwiek jego działanie jest niewiadome. Obszerna regresja na wielu poziomach abstrakcji pozwoliła na ciągłe usprawnianie zarówno architektury jak i dostarczonych funkcjonalności bez obaw wprowadzenia błędów. Podczas konstrukcji przypadków testowych zwrócono uwagę na to, że odpowiednie zdefiniowanie: argumentów wejściowych funkcji/klasy, wartości zwracanych przez funkcję/metodę, pozwoli zachować jedno ciało funkcji testowych. Podejście to nazywa się testowaniem parametrycznym.

10.1 Testy jednostkowe

```
1 [  3%] Built target gtest
2 [  6%] Built target gmock
3 [ 13%] Built target gmock_main
4 [100%] Built target KorytkoMag_RetDriver_UT
5 [=====] Running 28 tests from 5 test suites.
6 [-----] Global test environment set-up.
7 [-----] 7 tests from DBShould
8 [ RUN    ] DBShould.getObj_NotExisting_IOPaths2_template
9 [      OK ] DBShould.getObj_NotExisting_IOPaths2_template (1 ms)
10 [ RUN   ] DBShould.updateObj_NotExisting_IOPaths2
11 [      OK ] DBShould.updateObj_NotExisting_IOPaths2 (1 ms)
12 [ RUN   ] DBShould.createObj_WithGenerationOfUK_FromUK
13 [      OK ] DBShould.createObj_WithGenerationOfUK_FromUK (4 ms)
14 [ RUN   ] DBShould.createObj_WithGenerationOfUK_ThreeTimes
15 [      OK ] DBShould.createObj_WithGenerationOfUK_ThreeTimes (1 ms)
16 [ RUN   ] DBShould.updateObj_Existing_IOPaths2
17 [      OK ] DBShould.updateObj_Existing_IOPaths2 (1 ms)
18 [ RUN   ] DBShould.delete_Existing_IOPaths1
19 [      OK ] DBShould.delete_Existing_IOPaths1 (2 ms)
20 [ RUN   ] DBShould.delete_NotExisting_IOPaths2
21 [      OK ] DBShould.delete_NotExisting_IOPaths2 (1 ms)
22 [-----] 7 tests from DBShould (11 ms total)
23
24 [-----] 1 test from HDLCFrameTests
25 [ RUN   ] HDLCFrameTests.Transceive_XID_DummyScan
26 [      OK ] HDLCFrameTests.Transceive_XID_DummyScan (0 ms)
27 [-----] 1 test from HDLCFrameTests (0 ms total)
28
29 [-----] 3 tests from UserCommandParserShould
30 [ RUN   ] UserCommandParserShould.Accept_OnInput_StartPooling
31 [      OK ] UserCommandParserShould.Accept_OnInput_StartPooling (0 ms)
32 [ RUN   ] UserCommandParserShould.RejectUnknownCommand_OnInput_startPooling
33 [      OK ] UserCommandParserShould.RejectUnknownCommand_OnInput_startPooling (0 ms)
34 [ RUN   ] UserCommandParserShould.Shutdown_OnInput_Shutdown
35 [      OK ] UserCommandParserShould.Shutdown_OnInput_Shutdown (0 ms)
```

10. Testowanie oprogramowania

```
36 [-----] 3 tests from UserCommandParserShould (0 ms total)
37
38 [-----] 9 tests from HDLCReqFrameBodyTests/HDLCReqFrameBodyTests
39 [ RUN      ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/0
40 [ OK       ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/0 (0 ms)
41 [ RUN      ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/1
42 [ OK       ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/1 (0 ms)
43 [ RUN      ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/2
44 [ OK       ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/2 (0 ms)
45 [ RUN      ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/3
46 [ OK       ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/3 (0 ms)
47 [ RUN      ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/4
48 [ OK       ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/4 (0 ms)
49 [ RUN      ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/5
50 [ OK       ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/5 (0 ms)
51 [ RUN      ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/6
52 [ OK       ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/6 (0 ms)
53 [ RUN      ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/7
54 [ OK       ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/7 (0 ms)
55 [ RUN      ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/8
56 [ OK       ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/8 (0 ms)
57 [-----] 9 tests from HDLCReqFrameBodyTests/HDLCReqFrameBodyTests (1 ms total)
58
59 [-----] 8 tests from HDLCFrameInterpreterTests/HDLCFrameInterpreterTests
60 [ RUN      ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/0
61 [ OK       ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/0 (0 ms)
62 [ RUN      ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/1
63 [ OK       ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/1 (0 ms)
64 [ RUN      ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/2
65 [ OK       ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/2 (0 ms)
66 [ RUN      ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/3
67 [ OK       ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/3 (0 ms)
68 [ RUN      ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/4
69 [ OK       ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/4 (0 ms)
70 [ RUN      ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/5
71 [ OK       ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/5 (1 ms)
72 [ RUN      ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/6
73 [ OK       ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/6 (0 ms)
74 [ RUN      ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/7
75 [ OK       ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/7 (0 ms)
76 [-----] 8 tests from HDLCFrameInterpreterTests/HDLCFrameInterpreterTests (1 ms total)
77
78 [-----] Global test environment tear-down
79 [=====] 28 tests from 5 test suites ran. (14 ms total)
80 [ PASSED ] 28 tests.
```

Listing 10.1. Efekt uruchomienia testów jednostkowych

Testy jednostkowe zwane są często testami komponentów, czy też testami modułowymi. Ich głównym celem jest znalezienie błędów w implementacji danej jednostki / komponentu. [12] Uruchomienie testów odbywa się przy pomocy komendy „bash runUT.sh”, wywoywanej z poziomu głównego katalogu projektu.

10.1.1 Testowanie bazy danych

Przeprowadzone testy (listing 10.1- linijki [7; 22]):

1. Pobierz obiekt przy pomocy niestniejącego klucza.
2. Dodaj nowy obiekt przy pomocy klucza.
3. Wielokrotnie dodaj nowy obiekt przy pomocy klucza.
4. Dodaj nowy obiekt bez pomocy klucza.
5. Wielokrotnie dodaj nowy obiekt bez pomocy klucza.
6. Skasuj obiekt przy pomocy klucza.
7. Wielokrotnie skasuj obiekt przy pomocy klucza.

10.1.2 Testowanie budowania pełnej ramki AISG

Przeprowadzony test (listing 10.1- linijki [24; 27]) sprawdza czy konstruktor klasy „HDLC-Frame” poprawnie dokona enkapsulacji ciała ramki, wzbogacając ją o flagę startu, stopu oraz sumę kontrolną. Dzięki obiektowemu podejściu do tworzenia kodu źródłowego, brak jest konieczności wykonania testów dla każdej z budowanych ramek, ponieważ dokonano tego klasie testowej dedykowanej do weryfikacji poprawności budowania ciała ramki.

10.1.3 Testowanie parsera danych wejściowych użytkownika

Przeprowadzone testy (listing 10.1- linijki [29; 36]):

1. Wprowadź poprawnie prawidłową komendę
2. Wprowadź niepoprawnie prawidłową komendę, brak zachowania odpowiedniej wielkości liter
3. Wprowadź nieprawidłową komendę

10.1.4 Testowanie fabryki tworzącej ciało ramki AISG

Przeprowadzone testy (listing 10.1- linijki [38; 57]) pokrywają zbudowanie każdej z ramek wspomnianej w rozdziale „Analiza nawiązanej komunikacji”[9].

10.1.5 Testowanie interpretera ramki HDLC

Przeprowadzone testy (listing 10.1- linijki [59; 76]) pokrywają rozpoznanie każdej z ramek wspomnianej w rozdziale „Analiza nawiązanej komunikacji”[9], wprowadzonej w postaci łańcucha znaków.

10.2 Testy integracyjne

```

1 [ 2%] Built target gtest
2 [ 4%] Built target gmock
3 [ 9%] Built target gmock_main
4 [100%] Built target KorytkoMag_RetDriver_MT
5 [=====] Running 18 tests from 3 test suites.
6 [-----] Global test environment set-up.
7 [-----] 5 tests from UIAndDatabaseIntegrationShould
8 [ RUN   ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByOT_BufferToSend_Once
9 [       ] OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByOT_BufferToSend_Once (5 ms)
10 [ RUN  ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByOT_BufferToSend_Twice
11 [      ] OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByOT_BufferToSend_Twice (2 ms)
12 [ RUN  ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend
13 [      ] OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend (2 ms)
14 [ RUN  ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend_Twice_EqualKey
15 [      ] OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend_Twice_EqualKey (1 ms)
16 [ RUN  ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend_Twice_DifferentKey
17 [      ] OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend_Twice_DifferentKey (2 ms)
18 [-----] 5 tests from UIAndDatabaseIntegrationShould (12 ms total)
19
20 [-----] 8 tests from BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar
21 [ RUN   ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/0
22 [       ] OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/0 (2104 ms)
23 [ RUN  ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/1
24 [      ] OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/1 (2 ms)
25 [ RUN  ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/2
26 [      ] OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/2 (2 ms)

```

10. Testowanie oprogramowania

```
27 [ RUN      ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/3
28 [ OK       ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/3 (2 ms)
29 [ RUN      ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/4
30 [ OK       ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/4 (2 ms)
31 [ RUN      ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/5
32 [ OK       ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/5 (4 ms)
33 [ RUN      ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/6
34 [ OK       ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/6 (1 ms)
35 [ RUN      ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/7
36 [ OK       ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/7 (2 ms)
37 [-----] 8 tests from BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar (2119 ms total)
38
39 [-----] 5 tests from BaseFixtureWithDB/UI_Controller_RoundTripHDLC
40 [ RUN      ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/0
41 [ OK       ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/0 (2 ms)
42 [ RUN      ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/1
43 [ OK       ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/1 (1 ms)
44 [ RUN      ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/2
45 [ OK       ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/2 (2 ms)
46 [ RUN      ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/3
47 [ OK       ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/3 (8 ms)
48 [ RUN      ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/4
49 [ OK       ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/4 (2 ms)
50 [-----] 5 tests from BaseFixtureWithDB/UI_Controller_RoundTripHDLC (15 ms total)
51
52 [-----] Global test environment tear-down
53 [=====] 18 tests from 3 test suites ran. (2146 ms total)
54 [ PASSED ] 18 tests.
```

Listing 10.2. Testy integracyjne

Kolejnym poziomem są testy integracyjne, sprawdzają czy komponenty poprawnie współpracują ze sobą. [12] Uruchomienie testów odbywa się przy pomocy komendy „bash runMT.sh”, wywoływanej z poziomu głównego katalogu projektu.

10.2.1 Integracja bazy danych z interfejsem użytkownika

Przeprowadzone testy (listing 10.2- linijki [7; 18]) pokrywają przypadki testowe zdefiniowane w rozdziale „Testowanie bazy danych” 10.1.1. Różnicą jest sposób przekazania danych wejściowych. W poprzednim rozdziale w ciele funkcji testowej, wywoływano poszczególne funkcje interfejsu bazy danych. Testy integracyjne dokonują modyfikacji bazy danych, korzystając z funkcji obiektu klasy „CMenu”. Przy jego pomocy, końcowy użytkownik wchodzi w interakcję ze sterownikiem przy pomocy interfejsu konsoli. Dzięki odpowiedniej definicji metod klasy „CMenu”, na poziomie klasy testowej, pominięto fazę komunikacji przy pomocy konsoli. Zastąpiono go wywołaniem funkcji, która jako argument przyjmuje obiekt typu std::vector<std::string>. Przechowuje on podzieloną na tokeny rzeczywistą komendę, która ma za zadanie zmianę stanu bazy danych.

```
1 TEST_F(UIAndDatabaseIntegrationShould, Put_IOPaths_ByOT_BufferToSend_Once)
2 {
3     const auto& returnCode = ui_->runPredefinedCommands({
4         {PUT, IOPATHS_OT, "bufferToSend", BUFFER_TO_SEND_VAL_1} });
5     auto& received = db_.getObj<IOPaths>(IOPATHS_1_UK);
6
7     ASSERT_TRUE(returnCode);
8     ASSERT_TRUE(received);
9     ASSERT_TRUE(received->bufferToSend);
10    ASSERT_EQ(received->bufferToSend.value(), BUFFER_TO_SEND_VAL_1);
11 }
```

Listing 10.3. Ciało przykładowego testu integracyjnego interfejsu użytkownika wraz z bazą danych

10.2.2 Integracja interfejsu użytkownika z kontrolerem komend AISG

Przeprowadzone testy (listing 10.2- linijki [20; 37]) weryfikują działanie komend zdefiniowanych w rozdziale „Wymagania funkcjonalne” 7. Podobnie jak w przypadku testowania integracji bazy danych z interfejsem użytkownika. Dane wejściowe testu są obiektem typu std::vector<std::string>, który przechowuje rzeczywistą komendę podzieloną na tokeny. Ma ona za zadanie wykonanie jednej z komend rozpoznawaną przez warstwę fizyczną, łącza danyh czy aplikacyjną. Pomyślność integracji komponentów weryfikowana jest przy pomocy kodu rezultatu, zwracanego przez komendę „execute” należącą do klasy kontrolera komend AISG.

```

1 TEST_P(KorytkoMagControllerShouldPar, Execute_Command_Expect_ResultCode)
2 {
3     ctrl_ ->addCommands( GetParam().inCommands );
4     ctrl_ ->executeCommand();
5
6     ASSERT_EQ( ctrl_ ->getFinalresultCode(), GetParam().expectedHdIcFrame );
7 }
8 INSTANTIATE_TEST_CASE_P(BaseFixtureWithDBAndHDLC,
9     KorytkoMagControllerShouldPar,
10    ::testing::Values(
11        CommandsToExpectedFrame{
12            {{ L2::DEVICE_SCAN, BUFFER_TO_SEND_VAL_1 } },
13            L2::DEVICE_SCAN + DELIMITER
14        },
15        CommandsToExpectedFrame{
16            {{ L2::ADDRESS_ASSIGNMENT, BUFFER_TO_SEND_VAL_1 } },
17            L2::ADDRESS_ASSIGNMENT + DELIMITER
18        }
19    )
20 )

```

Listing 10.4. Ciało przykładowego testu integracyjnego interfejsu użytkownika oraz kontrolera komend AISG wraz z definicją przykładowych parametrów testu

10.2.3 Integracja interfejsu użytkownika, kontrolera komend AISG oraz fabryki ramek HDLC

Przeprowadzone testy (listing 10.2- linijki [39; 50]) weryfikują działanie komend zdefiniowanych w rozdziale „Wymagania funkcjonalne” 7. Na wejściu testu przekazywany jest obiekt typu `std::vector<std::string>`, który przechowuje rzeczywistą komendę podzieloną na tokeny. Rezultat wykonania weryfikowany jest z ramką HDLC budowaną przy pomocy fabryki użytej w kodzie produkcyjnym.

```

1 TEST_P(UI_Controller_RoundTripHDLC , ExecuteCommandAndExpectSentFrame)
2 {
3     const auto& returnCode = ui_.runPredefinedCommands(
4         GetParam().inCommands
5     );
6     auto sentFrames = hdlcCommunicators_.at(IDX_OF_REQUEST_RESPONSE_COMMUNICATOR)
7         ->receive(BUFFER_TO_SEND_VAL_1);
8
9     ASSERT_TRUE(returnCode);
10    ASSERT_TRUE(sentFrames);
11    ASSERT_EQ(sentFrames->build(), GetParam().expectedFrameHexes);
12 }
13 INSTANTIATE_TEST_CASE_P(BaseFixtureWithDB ,
14     UI_Controller_RoundTripHDLC ,
15     ::testing::Values(
16         ReceivedCommand_ExpectedFrameHexes{
17             {{ L2::ADDRESS_ASSIGNMENT, BUFFER_TO_SEND_VAL_1 } },
18             HDLCFrame(hdlcFrameBodyFactory ->get_FrameXID_AddressAssignment()).build()
19         },
20         ReceivedCommand_ExpectedFrameHexes{
21             {{ L2::LINK_ESTABLISHMENT, BUFFER_TO_SEND_VAL_1 } },
22             HDLCFrame(hdlcFrameBodyFactory ->get_FrameU_LinkEstablishment()).build()
23         }
24     )
25 )

```

Listing 10.5. Ciało przykładowego testu integracyjnego interfejsu użytkownika oraz kontrolera komend AISG wraz z definicją przykładowych parametrów testu

10.3 Manualne testy systemowe

Dzięki zaznajomieniu się z wieloma wzorcami projektowymi jak fabryka, komenda, budowniczy czy nakładanie obostrzeń na program dzięki listowaniu obsługiwanych komend, wspomniany wcześniej symulator urządzenia powstał przy minimalnym nakładzie pracy, co jest bardzo dużą oszczędnością. Świadczy to o sukcesie płynącego z dobrze wykonanego projektu architektury oprogramowania. Kolejną zaletą utworzenia symulatora urządzenia jest umożliwienie przeprowadzenia manualnych testów komponentowych, realizując regułę czarnej skrzynki. Symulator urządzenia wraz z sterownikiem komunikuje się przy pomocy biblioteki ZeroMQ.

10.4 Automatyczne testy systemowe

Implementacja automatycznych testów systemowych osiągalna będzie w momencie wystawienia przez programistę interfejsu aplikacji. Dzięki kompleksowemu pokryciu utworzonego kodu źródłowego testami na niższych poziomach, konfiguracja środowiska automatyzacji oraz tworzenie scenariuszy wykonania nie przynosi zbyt dużego zysku. Dopiero w momencie uruchomienia więcej niż jednej instancji sterownika, komunikującej się z wieloma fizycznymi bądź emulowanymi urządzeniami, faza automatycznych testów systemowych przyniesie wymierne korzyści. Z racji tego, że skalowalność w górę utworzonego systemu planowana jest na dalsze etapy implementacji, zdefiniowano zamiar wzbogacenia regresji o ten poziom testowania.

11. Dodatkowe informacje

11.1 Środowisko uruchomieniowe

- System operacyjny Linux
 - Dystrybucje:
 - * Manjaro 17.01;
 - Zależności:
 - * CMake 3.15.4;
 - * Boost C++ Libraries 1.71;
 - * Git -> Pobranie kodu produkcyjnego, GTest, GMock;
 - * GCC 9.2;

11.2 Licencje

Użyte biblioteki:

- Boost - Boost Software License, licencja typu OpenSource podobna do BSD i MIT;
- GTest - BSD 3;
- CMake - BSD 3;
- ZeroMQ - GNU GPL V3 wraz z koniecznością statycznego linkowania;

Kod źródłowy oparty jest na licencji CC BY-NC-ND 4.0 [2] czyli:

- dzielenie się - kopuj i rozpowszechniaj utwór w dowolnym medium i formacie;
- uznanie autorstwa - należy odpowiednio oznaczyć oraz podać link do recepcji oraz wskazać jeśli zostały dokonane w nim zmiany;
- użycie niekomercyjne - nie należy wykorzystywać utworu do celów komercyjnych;
- bez utworów zależnych - remiksując, przetwarzając lub tworząc na podstawie utworu, nie wolno rozpowszechniać zmodyfikowanych treści;

12. Podsumowanie

Sposób podejścia do implementacji pracy inżynierskiej polegający na zastosowaniu licznych wzorców projektowych sprawił, że powstały kod jest modularny oraz elastyczny pod kątem przyszłej rozbudowy czy utrzymania. Powstały produkt posiada wysoką wartość rynkową łatwą do zrealizowania. Zostanie to osiągnięte dzięki implementacji strategii komunikacji, odpowiedzialnej za konwersję ramek HDLC z postaci heksadecymalnej do postaci binarnej. Do tego momentu implementacja kolejnych scenariuszy oraz obsługi ramek nadzorujących może być kontynuowana. Analiza funkcjonalności wymaganych do zrealizowania poprzez podział na warstwy modelu OSI, umożliwiła zidentyfikować konieczność emulacji warstwy fizycznej. Sprawiło to, że zarówno wykonanie testów modułowych jak i manualnych jest realizowane bez konieczności podłączenia rzeczywistego urządzenia. Podział zadań przy pomocy tablicy Kanban, pozwolił w odpowiednim momencie zaobserwować konieczność wprowadzenia usprawnień w procesie jak i architekturze oprogramowania. Jednoczesne tworzenie diagramu sekwencji oraz klas, pozwalało postrzegać każdą z funkcjonalności z perspektywy całego systemu. Pomimo kilku potknięć, które sprawiły że komunikacja z fizycznym urządzeniem nie została zrealizowana,

12.1 Co zostało zrealizowane

Zaimplementowany został sterownik do urządzenia typu RET, który komunikuje się z użytkownikiem przy pomocy interfejsu konsolowego. Po zestawieniu trzech warstw OSI, wysłano na wybrany port żądanie kalibracji oraz otrzymano odpowiedź na tę wiadomość. Konfiguracja środowiska uruchomieniowego na systemie Manjaro Linux bądź Arch oraz komplikacja plików binarnych, sprowadza się do wywołania zaledwie kilku komend. Wiadomości zapisane są w języku angielskim, zrozumiałym dla człowieka. Ryzyko pojawienia się błędu w trakcie działań Osoba korzystająca z programu na bieżąco jest informowana o efektach komunikacji z urządzeniem dzięki rozbudowanemu systemowi logowania, z funkcją zapisu do pliku tekstuowego. W przypadku konieczności głębszej analizy działania programu, należy zmienić definicję filtrowania logów na *trace*. Tworzenie rozbudowanych wiadomości zapisanych za pomocą wartości heksadecymalnych sprowadzono do podania odpowiedniej komendy w języku angielskim oraz adresu portu, do którego podłączony jest symulator urządzenia.

12.2 Co nie zostało zrealizowane

Podążając zgodnie z założeniami projektowymi, poniższe wymagania stają się konieczne dopiero w momencie podłączenia rzeczywistego urządzenia poprzez port USB. Ważnym krokiem dokonanym, jest dokładne przeanalizowanie problemów, które programista napotka w przyszłości, co pozwoli odpowiednio zaplanować fazę ich wdrażania.

12.2.1 Walidacja sumy CRC

W celu przesłania wiadomości zapisanej w systemie heksadecymalnym przy pomocy interfejsu USB <-> RS-485, należy dokonać jej konwersji na postać binarną. Zarówno wiadomość odebrana, jak i wysłana, w trakcie transmisji może ulec wybrakowaniu bądź modyfikacji. W tym celu ramka HDLC protokołu AISG 2.0 posiada dwa bajty przeznaczone na walidację takowej wiadomości a jest to wykonywane dzięki wyliczeniu sumy CRC-16. W związku z tym, że istnieje wiele implementacji algorytmu wyliczania sumy CRC, a każda z nich może różnić się od siebie zarówno: definicją wyrażenia wielomianowego, jego postacią oraz wartością początkową, próbowałem na podstawie „podsłuchanej” wiadomości przy pomocy inżynierii wstępnej zdefiniować brakującą wiedzę, lecz aby tego dokonać konieczne było zdobycie informacji na temat zarówno endianowości jak i uporządkowania bitów, której na moment opracowywania algorytmu nie posiadało. Pomimo tego, że dla komputera jest to prosta operacja, to weryfikacja przez człowieka wartości sumy CRC dla wiadomości o długości 16-tu bajtów na kartce zajmuje naprawdę dużo czasu. W dodatku powszechnym problemem jest to, że producenci urządzeń linii antenowej pomimo tego, że deklarują pełną implementację protokołu AISG 2.0, to w rzeczywistości okazuje się, że wcale tak nie jest. Chcąc zapoznać się z protokołem komunikacyjnym oraz zrealizować projekt inżynierski, zamiast walczyć z wadliwym urządzeniem, postanowiono utworzyć symulator urządzenia. Pomyślna komunikacja z symulowanym podrzędnych zostaje osiągnięta dzięki przyjęciu stałej wartości dla pola przeznaczonego na dwubajtową sumę CRC.

12.2.2 Podmiana bajtów o wartości flagi startu/stopu

Z języka angielskiego „Byte stuffing”. Procedura polega na tym, że zarówno dla flagi startu jak i stopu zarezerwowana jest wartość 0x7E. Niestety urządzenie podrzędne inne niż posiadane, może odpowiedzieć na podczas procedur XID negocjacji, wiadomością zawierającą bajt 0x7E. Nieobsłużone takie zjawisko spowoduje, że urządzenie nadziedne zinterpretuje ten bajt jako bajt stopu, co jest niepożądane[10]. W tym celu standard AISG definiuje procedurę do jakiej należy się zastosować w takiej sytuacji, z którą nie zaznajomiłem się.

12.2.3 Interwały czasowe

Protokół AISG 2.0 definiuje szereg interwałów których obecność można zaobserwować podczas komunikacji z urządzeniem. Jednym z nich jest utrzymanie urządzenia w stanie za-

adresowanym. W przypadku kiedy urządzenie podrzędne nie otrzyma jakiekolwiek wiadomości w ciągu 3 minut od jego zaadresowania, przechodzi ono w stan niezaadresowany, po czym ponownie należy zestawić warstwę fizyczną oraz łącza danych wraz z całą procedurą XID negocjacji. Kolejnymi zależnościami czasowymi są odstępy pomiędzy wysłaniem wiadomości a rozpoczęciem nasłuchiwanego na odpowiedź. Problemem mógłby być scenariusz w którym wysyłamy wiadomość do urządzenia podrzędnego, a jest ono na przykład w trakcie procedury kalibracji która może trwać ok 2 minuty. Jako odpowiedź możemy wtedy otrzymać ramkę typu S której stan nazwano RNR czyli z angielskiego „Receiver Not Ready”. W tej sytuacji należy odczekać pewien kwant czasu, a następnie ponówić komunikację.

12.2.4 Ustalanie maski

RET jest urządzeniem które często posiada dodatkowy port RS-485, co oznacza że można podłączyć dwa bądź więcej urządzeń szeregowo. W takiej sytuacji na wiadomość pochodzącej z urządzenia nadzorowanego zaadresowaną wartością 0xFF, każde z nich wyśle swoją odpowiedź, co niesie ze sobą wiele problemów. Po stronie urządzenia nadzorowanego to zjawisko zostanie zaobserwowane jako złożenie zawartości wielu wiadomości co będzie skutkowało niepomyślnym procesem weryfikacji sumy CRC oraz ilości flag stopu. Algorytm definiowania maski skutecznie rozwiązuje ten problem. Podczas tej procedury ustala się wartość unikalnego identyfikatora każdego z urządzeń, w celu wysłania wiadomości z żądaniem zaadresowania, na którą odpowie tylko jedno urządzenie. Realizacja powyższego problemu często korzysta z algorytmów przeszukiwania binarnego, w trakcie którego zadaniem jest ustalenie unikalnego identyfikatora urządzenia podrzędnego. Z racji tego, że podczas pracy inżynierskiej przewidywano uruchomienie instancji sterownika dla każdego urządzenia osobno, pominięto implementację wyżej wymienionej logiki.

12.3 Możliwości rozwoju

Dzięki elastycznej i skalowej architekturze oprogramowania, bez większego problemu powinna być możliwość zaimplementowania rzeczywistej warstwy fizycznej. Pozwoli to na podłączenie prawdziwego urządzenia w celu wykonywania zabiegów testowych na nim. Po zrealizowaniu tego etapu planowane jest dodanie kolejnych komend definiowanych przez protokół AISG 2.0 takich jak: aktualizacja oprogramowania, reset twardy czy miękki oraz ustawienie kąta. Kolejnym usprawnieniem projektu jest wdrożenie do aplikacji mechanizmu wielowątkowości, co pozwoli obsłużyć komunikację z większą liczbą urządzeń z poziomu jednej instancji sterownika. Warstwy dynamicznego budowania wiadomości pozwalają również zmniejszyć czas wdrożenia najnowszej wersji protokołu AISG w wersji 3.0.

Bibliografia

- [1] Beamwidth. <http://www.itcmp.pwr.wroc.pl/~jwach/Techniczny%20EN-PL.htm>. (27.01.2020).
- [2] CC license. <https://creativecommons.org/licenses/by-nc-nd/4.0/deed.pl>. (12.02.2020).
- [3] CMake. <https://pl.wikipedia.org/wiki/CMake>. (12.02.2020).
- [4] CRC. http://wmii.uwm.edu.pl/~bnowak/userfiles/downloads/dydaktyka/Sieci_Komputerowe/CRC.rtf. (12.02.2020).
- [5] Elektryczny tilt. <https://www.kathreinusa.com/support/ret-products/>. (27.01.2020).
- [6] Enkapsulacja. <https://eduwiki.wmi.amu.edu.pl/pms/17sik/c6>. (12.02.2020).
- [7] ETSI TS 125 460 - UTRAN iuant interface: General aspects and principles. https://www.etsi.org/deliver/etsi_ts/125400_125499/125460/15.01.00_60/ts_125460v150100p.pdf. (5.14.2019).
- [8] High-level data link control. https://en.wikipedia.org/wiki/High-Level_Data_Link_Control. (25.01.2020).
- [9] High-level data link control. https://pl.wikipedia.org/wiki/High-Level_Data_Link_Control. (25.01.2020).
- [10] Information technology – telecommunications and information exchange between systems – high-level data link control (hdlc) procedures. <https://www.iso.org/standard/37010.html>. (01.01.2017).
- [11] Model OSI. https://pl.wikipedia.org/wiki/Model_OSI. (12.02.2020).
- [12] Poziomy testowania. <https://devenv.pl/poziomy-testow>. (10.02.2020).
- [13] Radio mobile. https://www.ve2dbe.com/rmonline_s.asp. (27.01.2020).
- [14] Erich Gamma, Richard Helm, Ralph Johnson, and John M Vlissides. *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku*. WNT, 2005.
- [15] RC Martin. *Zwinne wytwarzanie oprogramowania*. 2015.
- [16] Robert C Martin. *Czysty kod: poznaj najlepsze metody tworzenia doskonałego kodu*. Wydawnictwo Helion, 2014.

Spis rysunków

1.1	Diagram klas przedstawiający relacje klas tworzących bazę danych. (Opracowanie własne)	9
1.2	Diagram klas przedstawiający klasy wchodzące w skład interfejsu użytkownika. (Opracowanie własne)	11
2.1	RET - Miejsce na antenę. (Zdjęcie własne)	13
2.2	RET - podłączonym kablem RS-485 oraz włożoną atrapą anteny. (Zdjęcie własne)	13
2.3	Rysunek przedstawiający kąt modyfikowany przy pomocy RET-a. (Opracowanie własne)	14
2.4	Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 0 stopni. (Opracowanie własne przy pomocy programu Radio Mobile)	15
2.5	Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 40 stopni. (Opracowanie własne przy pomocy programu Radio Mobile)	15
3.1	Diagram klas przedstawiający relację pomiędzy abstrakcyjną oraz konkretną klasą poszczególnej ramki. (Opracowanie własne)	18
3.2	Ramka informacyjna - ewaluacja bajtu kontrolnego. (Opracowanie własne)	20
5.1	Diagram klas przedstawiający realizację wzorca komendy. (Opracowanie własne)	26
5.2	Przepływ wiadomości dla wzorca Publikuj-Subskrybuj oraz Żądanie-Odpowiedź. (Opracowanie własne)	29
9.1	Ustanowienie prędkości połączenia wraz z początkowym skanowaniem urządzeń (Opracowanie własne)	39
9.2	Żądanie adresacji oraz dodatkowe skanowanie urządzeń. (Opracowanie własne)	43
9.3	Negocjacja rozmiaru okna oraz payloadu ramki informacyjnej oraz ustanowienie normalnego trybu komunikacji. (Opracowanie własne)	44

Spis rysunków

9.4 Negocjacja pozostałych parametrów HDLC oraz żądanie kalibracji urządzenia. (Opracowanie własne)	48
--	----

Spis listingów

5.1	Interfejs komendy	25
5.2	Definicja klasy konkretnej komendy używającej fabryki oraz strategii	25
5.3	Definicja klasy dla obiektu pustego	27
5.4	Przykład użycia obiektu pustego	27
5.5	Plik nagłówkowy dla metody szablonowej walidacji komendy	28
5.6	Metoda szablonowa - Wywołanie metod wirtualnych z poziomu innej metody .	28
5.7	Strategia komunikacji Żadanie-Odpowiedź dla urządzenia nadzawanego	30
5.8	Konstruktor zrealizowany podejściem wstrzykiwania zależności	31
5.9	Budowniczy wraz z Fluent API podczas budowania ramki I - Kalibruj	32
5.10	Fabryka budowniczych dla sterownika urządzenia nadzawanego	33
8.1	Wykonanie programu	38
10.1	Efekt uruchomienia testów jednostkowych	50
10.2	Testy integracyjne	52
10.3	Ciało przykładowego testu integracyjnego interfejsu użytkownika wraz z bazą danych	53
10.4	Ciało przykładowego testu integracyjnego interfejsu użytkownika oraz kontrolera komend AISG wraz z definicją przykładowych parametrów testu	54
10.5	Ciało przykładowego testu integracyjnego interfejsu użytkownika oraz kontrolera komend AISG wraz z definicją przykładowych parametrów testu	55

Zawartość płyty DVD

1. Praca inżynierska w formacie .tex
2. Praca inżynierska w formacie .pdf
3. Kod źródłowy w postaci plików .cpp, .hpp, .txt
4. Adresy repozytorium z zamieszczonym kodem źródłowym
5. Listę wymaganych paczek dostępnych dzięki managerowi pacman dla systemu Manjaro Linux

Wrocław, dnia 2020-02-15

Wydział Informatyki

Kierunek: **informatyka**

Paweł Koryciński

(imię i nazwisko studenta)

6749

(nr albumu)

O ŚWIADCZENIE AUTORSKIE

Oświadczam, że niniejszą pracę dyplomową pod tytułem: Symulator sterownika do RET-a implementujący protokół AISG 2.0 napisałem/am samodzielnie. Nie korzystałem/am z pomocy osób trzecich, jak również nie dokonałem/am zapożyczeń z innych prac.

Wszystkie fragmenty pracy takie jak cytaty, rycinę, tabele, programy itp., które nie są mojego autorstwa, zostały odpowiednio zaznaczone i zamieszczono w pracy źródła ich pochodzenia. Treść wydrukowanej pracy dyplomowej jest identyczna z wersją pracy zapisaną na przekazywanym nośniku elektronicznym.

Jednocześnie przyjmuję do wiadomości, że jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdzi popełnienie przeze mnie plagiatu, skutkować to będzie niedopuszczeniem do dalszych czynności w sprawie nadania mi tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz złożenie zawiadomienia o podejrzeniu popełnienia przestępstwa.

.....
(podpis studenta)

Wrocław, dnia 2020-02-15

Wydział Informatyki

Kierunek: informatyka

Paweł Koryciński

.....
(imię i nazwisko studenta)

6749

.....
(nr albumu)

OŚWIADCZENIE O UDOSTĘPNIANU PRACY DYPLOMOWEJ

Tytuł pracy dyplomowej:

Symulator sterownika do RET-a implementujący protokół AISG 2.0

Wyrażam zgodę (nie wyrażam zgody)¹ na udostępnianie mojej pracy dyplomowej.

.....
(podpis studenta)

¹ Niepotrzebne skreślić