

**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

1 Wstęp	4
1.1 Wprowadzenie	4
1.2 Cel pracy	4
1.3 Motywacja	4
1.4 Zakres	5
1.4.1 Baza danych	5
1.4.2 Back-end	5
1.4.3 Front-end	6
1.4.4 Testowanie	6
Cześć Przeglądowa	7
2 RET	8
2.1 Prezentacja urządzenia	8
2.2 Zmiana szerokości głównej wiązki fali elektromagnetycznej	9
3 Protokół AISG 2.0	11
3.1 Warstwy	11
3.1.1 1-sza - Fizyczna	11
3.1.2 2-ga - Łącza danych	11
3.1.3 7-ma - Aplikacji	11
3.2 HDLC	12
3.2.1 Struktura ramki	12
3.2.2 Typy ramek	12
3.3 Typy ramek HDLC	13
3.3.1 Ramka XID	13
3.3.2 Ramka U	14
3.3.3 Ramka I	14
4 SOLID	16
4.1 SRP	16
4.2 OCP	16
4.3 LSP	16
4.4 ISP	17
4.5 DIP	17

Cześć Praktyczna	18
5 Wzorce projektowe	19
5.1 Behawioralne	19
5.1.1 Komenda	19
5.1.2 Obiekt pusty	21
5.1.3 Metoda szablonowa	22
5.1.4 Strategia	22
5.1.5 Wstrzykiwanie zależności	24
5.1.6 Zabór zasobu jest inicjalizacją	24
5.2 Kreacyjne	24
5.2.1 Budowniczy	24
5.2.2 Fabryka	25
6 Wymagania funkcjonalne	27
6.1 Historyjki użytkownika (Warstwa fizyczna łącza danych) - nazwa komendy	27
6.1.1 Ustanowienie prędkości połączenia - SetLinkSpeed	27
6.1.2 Negocjacja roli - AddressAssignment	28
6.1.3 Negocjacja parametrów HDLC - HDLCPARAMETERS	28
6.1.4 Ustanowienie synchronicznego trybu komunikacji - LinkEstablishment	29
6.1.5 Negocjacja parametrów HDLC - 3GPPReleaseID	29
6.1.6 Negocjacja parametrów HDLC - AISGProtocolVersion	30
6.2 Historyjki użytkownika (Warstwa aplikacyjna) - nazwa komendy	30
6.2.1 Kalibracja - Calibrate	30
7 Analiza nawiązanej komunikacji	32
7.1 Ustanowienie prędkości połączenia	33
7.2 Skanowanie urządzeń	33
7.3 Żadanie adresacji	33
7.4 Ponowne skanowanie urządzeń	33
7.5 Negocjacje parametrów HDLC	34
7.6 Przejście na normalny tryb komunikacji	35
7.7 Kalibracja	35
8 Wymagania niefunkcjonalne	38
8.1 Środowisko uruchomieniowe	38
8.2 Zrzuty ekranu	38

9 Plan dalszego rozwoju	41
9.1 Porażki odnośnie planu początkowego	41
9.1.1 Weryfikacja wiadomości pod względem obecności zarezerwowanych znaków	41
9.1.2 Interwały czasowe	42
9.1.3 Ustalanie maski	42
9.2 Rozbudowa aplikacji o nowe funkcjonalności	42
10 Podsumowanie	44
Bibliografia	45
Spis rysunków	47
Spis listingów	48
Zawartość płyty DVD	49
Załączniki	50
Oświadczenie autorskie	50
Oświadczenie o udostępnieniu pracy	51

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 smsa czy skorzystania ze skrzynki mailowej. Są w tym, 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 obszaru aktualnie pokrytego zasięgiem, anteny dipolowe usprawniane są o dodatkowe urządzenie które nosi nazwę RET, które jest szeroko stosowane w technologiach GSM, WCDMA, LTE.

1.2 Cel pracy

Jako cel obrałem zaznajomienie się z protokołem komunikacyjnym AISG 2.0, którego użyście 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-232 -> USB, w celu skrócenia czasu testowania urządzenia.

1.4 Zakres

Projekt 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;

1.4.2 Back-end

- Realizacja wzorców projektowych:
 - Komenda;
 - Metoda szablonowa;
 - Most;
 - Obiekt pusty;
 - Budowniczy;
- 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:
 - 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
 - * 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

Część Przeglądowa

2. RET

2.1 Prezentacja urządzenia

Z angielskiego Remote Electrical Tilt, czyli urządzenie służące do zdalnej modyfikacji kąta głównej wiązki anteny, zwanego elektrycznym tiltiem.¹



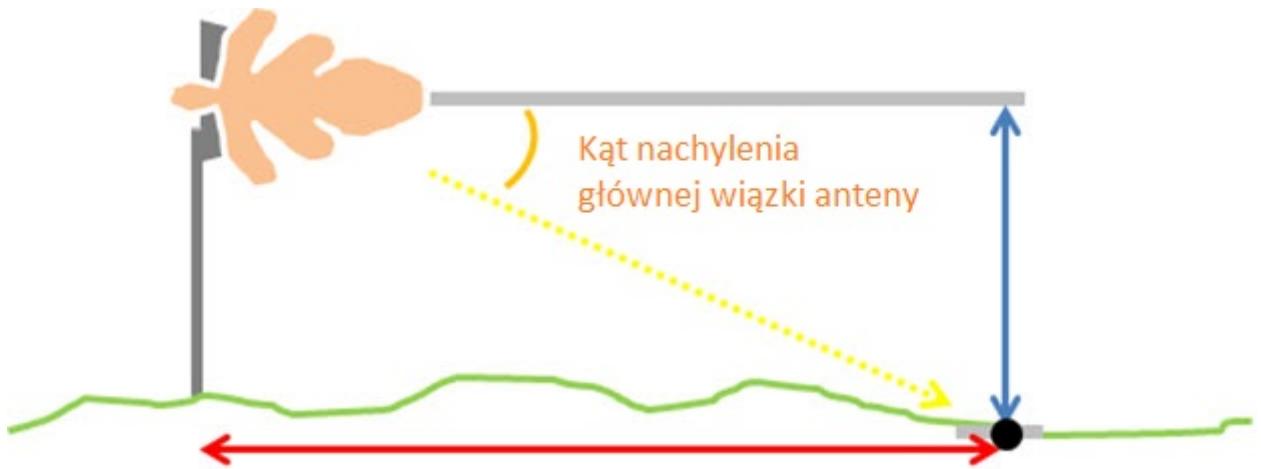
Rysunek 2.1. RetSimulator- Miejsce na antenę



Rysunek 2.2. RetSimulator podłączonym kablem RS-485 oraz włożoną atrapą anteny

¹Elektryczny tilt. <https://www.kathreinusa.com/support/ret-products/>. 27.01.2020.

2.2 Zmiana szerokości głównej wiązki fali elektromagnetycznej



Rysunek 2.3. Rysunek przedstawiający kąt modyfikowany przy pomocy RET-a

Poniższe rysunki wygenerowano dzięki programowi Radio Mobile² Przedstawiono na nich szerokość wiązki promieni sygnału radiowego w osi poziomej³ 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.

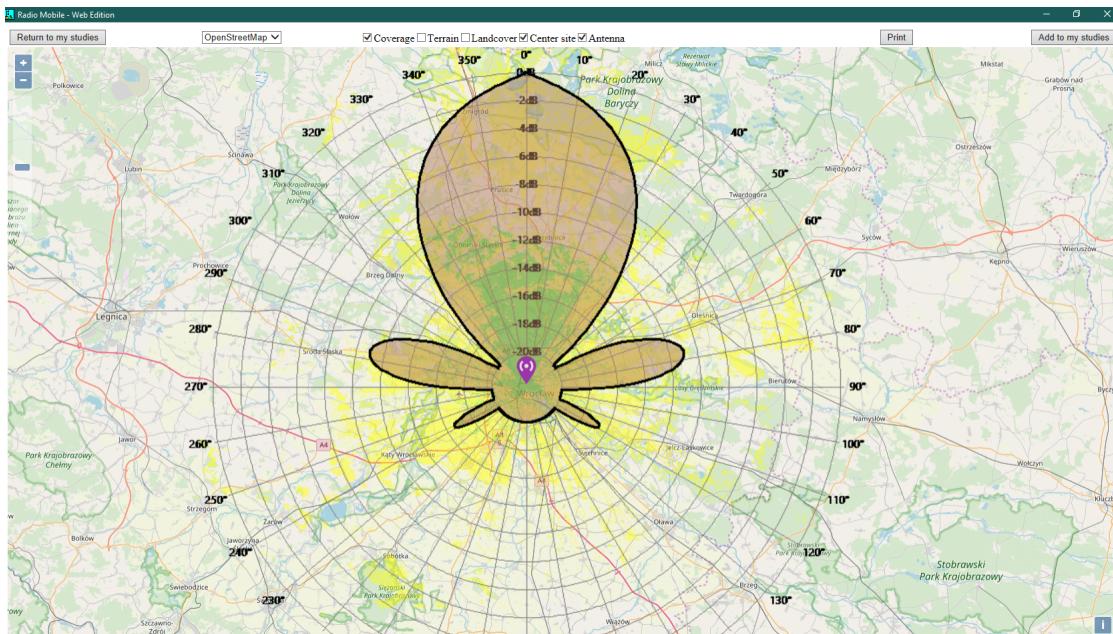
Antenę umieszczono przy WSIZ Copernicus.

Pomarańczowym kolorem widać tzw. kwiatek sygnału, który w przypadku kąta 0 stopni jest wydłużony oraz węższy, 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.

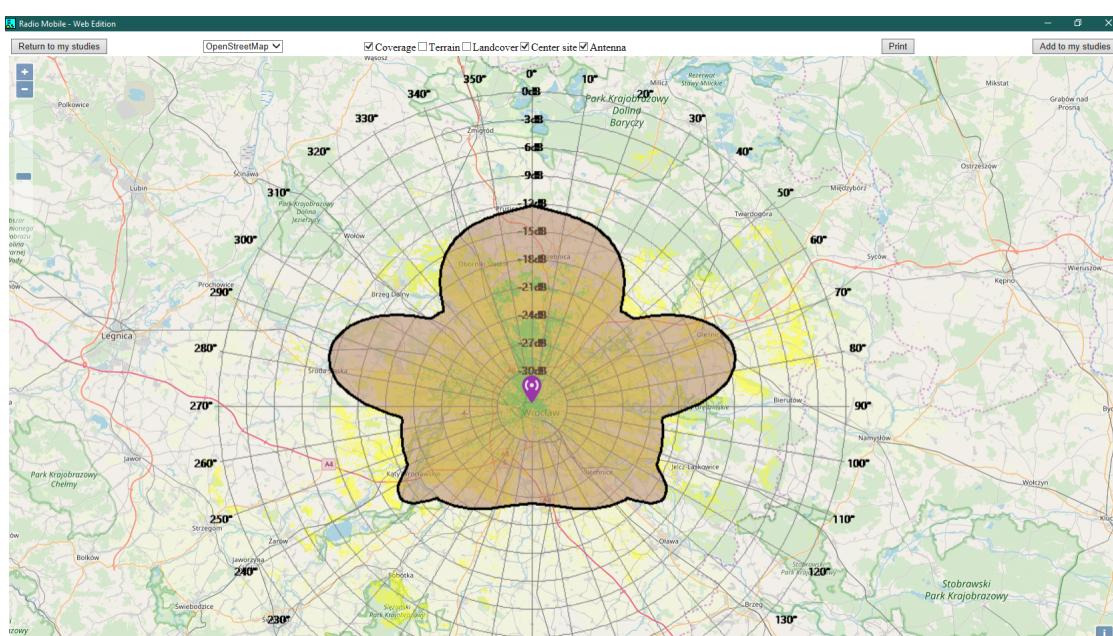
²Radio Mobile. https://www.ve2dbe.com/rmonline_s.asp. 27.01.2020.

³Beamwidth. <http://www.itcmp.pwr.wroc.pl/~jwach/Techniczny%20EN-PL.htm>. 27.01.2020.

2. RET



Rysunek 2.4. Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 0 stopni



Rysunek 2.5. Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 40 stopni

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 v2.0¹ realizującego podzbior modelu OSI².

3.1 Warstwy

3.1.1 1-sza - Fizyczna

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

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

3.1.2 2-ga - Łącza danych

Rola tej warstwy jest:

- Enkapsulacja³ ramki HDLC Body⁴ do ramki HDLC⁵:
 - Dodanie bitów startu i stopu;
 - Obliczenie oraz walidacja sumy CRC;⁶
- 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 podrzędnego;
 - 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;

¹ Protokół AISG v2.0 - <https://aisg.org.uk/files/AISG-v2.0.pdf>

² Model OSI - model odniesienia łączenia systemu otwartych opisujący struktury komunikacji sieciowej, https://pl.wikipedia.org/wiki/Model_OSI

³ Enkapsulacja - <https://pl.wikipedia.org/wiki/Kapsułkowanie>

⁴ HDLC Body - Ramka HDLC bez bajtów startu, stopu, sumy CRC.

⁵ HDLC - https://en.wikipedia.org/wiki/High-Level_Data_Link_Control

⁶ Suma CRC16 - https://en.wikipedia.org/wiki/Cyclic_redundancy_check

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, obsługuje zarówno połączenia dwupunktowe, jak i wielopunktowe. Jest protokołem o orientacji bitowej oraz jest przezroczysty informacyjnie.⁷ W przypadku jeśli przesyłana wartość jest wielobajtowa, zastosowane jest podejście cienkokońcowości.⁸

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 urządzenie;

⁷High-Level Data Link Control. https://pl.wikipedia.org/wiki/High-Level_Data_Link_Control. 25.01.2020.

⁸ Little endian - inaczej cienkokońcowość, forma zapisu danych o rozmiarze większym niż jeden bajt w której najmniej znaczący bajt umieszczony jest jako pierwszy.

3.3 Typy ramek HDLC

3.3.1 Ramka XID

Jej nazwa pochodzi z języka angielskiego „exchange identification”. Służy do przekazania urządzeniu podrzdnemu wiedzy na temat możliwości oraz charakterystyki komunikacji na warstwie łącza danych.⁹ Odpowiadając na tego typu wiadomości, urządzenie podrzędne 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 używamy również podczas ustalania prędkości komunikacji co jest częścią zestawiania warstwy fizycznej. Bajt kontrolny dla wiadomości przesyłanej będzie miał zawsze wartość 0xBF.

Bajty budujące ramkę:

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

Nie przez przypadek wspomniałem o parametrach HDLC zamiast o parametrze, gdyż równie dobrze można wysłać trzy wiadomości zawierające po jednym parametrze, jak i połączyć je w jedną większą.

⁹*High-Level Data Link Control.* https://en.wikipedia.org/wiki/High-Level_Data_Link_Control.
25.01.2020.

3.3.2 Ramka U

Jej nazwa pochodzi z języka angielskiego „Unnumbered” co oznacza nienumerowana.¹⁰ 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.¹¹ Dzięki niej można żądać od urządzenia podlegającego 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;

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

Proces ten w przypadku bitu który ustawiany jest na pozycji 4 bajtu kontrolnego P/F¹² służy do:

- Poinformowania urządzenia podlegającego, że urządzenie nadzorujące żąda odpowiedzi na właśnie przesyłaną wiadomość;
- Poinformowania urządzenia nadzorującego 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 nadzorujące czyli N(R)¹³. Bit od pierwszego do trzeciego włącznie definiują nam licznik wiadomości wysłanych przez urządzenie nadzorujące czyli N(S)¹⁴. Na poniższym diagramie możemy zaobserwować ewaluację bajtu kontrolnego dla kolejnych wiadomości.

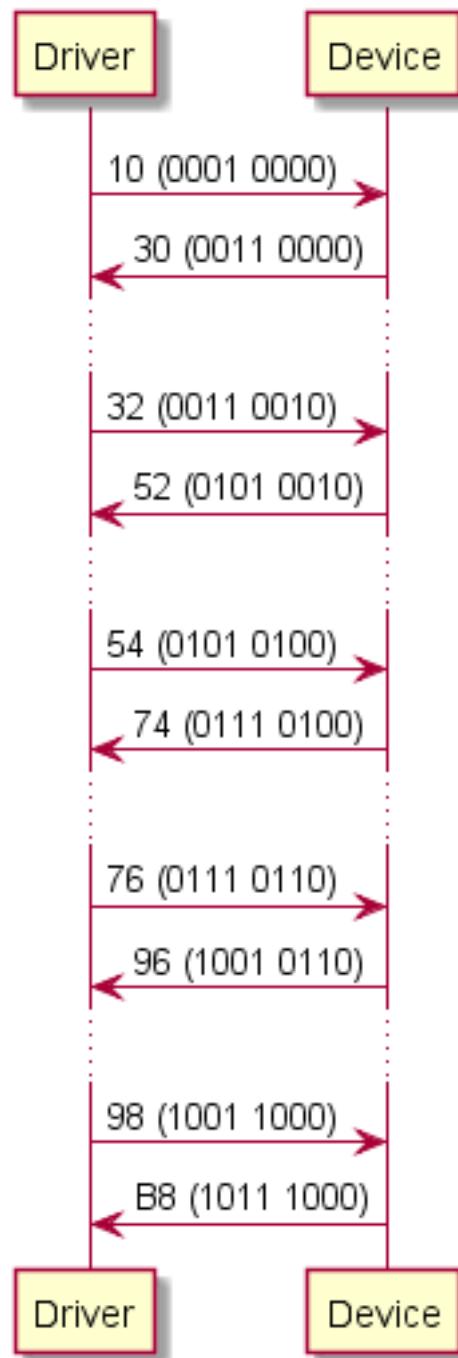
¹⁰Tamże.

¹¹Tamże.

¹² P/F - ang. bit Pool/Final

¹³ N(R) - ang. Receive sequence number

¹⁴ N(S) - ang. Send sequence number



Rysunek 3.1. Ramka informacyjna - ewaluacja bajtu kontrolnego

4. SOLID

Projektując sterownik starałem się postępować według dobrych praktyk programowania, powszechnie znanych jako skrót SOLID. Dążenie do ich realizacji 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. Zwrócić też trzeba uwagę na użyte wcześniej słowo, gdyż nie przez przypadek wspomniałem właśnie o dążeniu do ich wdrożenia. Poniższe reguły czasami bywają ciężkie w realizacji zwłaszcza, kiedy nasz 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. Pozwolę sobie przedstawić prawa budujące zasady SOLID oraz wspomnieć o ich głównych założeniach.

4.1 SRP

Z angielskiego Single Responsibility Principle, a więc zasada pojedynczej odpowiedzialności. Mówi ona, że powód modyfikacji klasy powinien być tylko jeden.¹ Realizacja tej reguły uchroni nas przed niechcianą modyfikacją definicji dużej liczby klas w przypadku zmiany logiki tylko jednej z nich.

4.2 OCP

Z angielskiego Open/Closed Principle, a więc zasada otwarte-zamknięte. Mówi ona, że encje programowanie (klasy, moduły, funkcje itp.) powinny być otwarte na rozbudowę, ale zamknięte dla modyfikacji.² Jeśli ta zasada zostanie właściwie zastosowana, to nowe zmiany uzyskuje się poprzez dodanie nowego kodu, aniżeli zmiane istniejącego.

4.3 LSP

Z angielskiego Liskov Substitution Principle, a więc zasada podstawiania Liskov. Mówi ona, że musi być możliwość podstawienia typów pochodnych za ich typy bazowe.³ Trzeba pamiętać o tej regule w trakcie definiowania relacji dziedziczenia pomiędzy klasami.

¹RC Martin. *Zwinne wytwarzanie oprogramowania*. 2015, s. 103.

²Tamże, s. 117.

³Tamże, s. 127.

4.4 ISP

Z angielskiego Interface Segregation Principle, a więc zasada segregacji interfejsów. Mówi ona, że klienci nie powinny być zmuszone do zależności od metod, których nie używają.⁴ Dzięki postępowaniu zgodnie z tą zasadą, utworzone interfejsy abstrakcyjnych klas bazowych będą zawierały minimalną ilość funkcji czysto wirtualnych.

4.5 DIP

Z angielskiego Dependency Inversion Principle, a więc zasada odwracania zależności. Mówi ona po pierwsze, że moduły wysokopoziomowe nie powinny zależeć od modułów niskopoziomowych. I jedne, i drugie powinny zależeć od abstrakcji.⁵ Kolejnym postulatem jest to, że abstrakcje nie powinny zależeć od szczegółów. To szczegóły powinny zależeć od abstrakcji.⁶

⁴Tamże, s. 151.

⁵Tamże, s. 141.

⁶Tamże, s. 141.

Część

Część Praktyczna

5. Wzorce projektowe

W latach kiedy programowanie obiektowe nabierały coraz większego rozprędu, wiele osób natrafiało na grupę problemów pewnej kategorii. Podczas wielokrotnych prób ich rozwiązywanie, różne osoby dochodziły do wspólnych wniosków odnośnie architektury kodu źródłowego. 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 efekt końcowy różni się jedynie od parametrów wejściowych. Podczas tworzenia wzorców usilnie trzymano się zbioru innych reguł 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 otwarcie na modyfikacje a zamknięcie 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. Podczas implementacji sterownika zastosowano połączenie trzech wzorców projektowych a to wszystko dzięki potężnemu podejściu do organizacji struktury kodu jakim jest użycie komend. Zestawiając implementację interfejsu komendy, wywołanie budowania konkretnej wiadomości dzięki fabryce pozwala na zdefiniowanie klas „finalnych”¹ zawierających charakterystyczne dla każdej z komend wywołań.

```
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 );
8     virtual ~ICommand();
9 protected:
```

¹ Klasa finalna - klasa po której nie można zdefiniować relacji dziedziczenia

5. Wzorce projektowe

```
9     ICommand();
10    using AlmagControllerInformer = boost::signals2::signal<void(void)>;
11    AlmagControllerInformer informControllerAboutResult_;
12 };
```

Tabela 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 }
```

Tabela 5.2. Definicja klasy konkretnej komendy używającej fabryki oraz strategii

5.1.2 Obiekt pusty

Znaną przypadłością w językach obiektowych jest wykonanie dalszej akcji w zależności od stanu pewnego obiektu. 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.

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

Tabela 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 }
```

Tabela 5.4. Przykład użycia obiektu pustego

5.1.3 Metoda szablonowa

Wprowadzenie systemu walidacji argumentów podanych przez użytkownika gwarantuje nam bezpieczne wykonywanie akcji niskopoziomowych takich jak wysłanie wiadomości do urządzenia, bez zbędnego umieszczanie logiki weryfikacji w dalszym etapie. W dodatku otworem przed nami staje możliwość lekkiej korekcji podanej komendy bądź umożliwienie procedury automatycznego pobrania wcześniej zapisanych w bazie danych wartości, których częste użycie może zmużyć klienta. Obecność komendy „execute” pozwala nam połączyć wywołanie powyższych funkcji dzięki jednemu poleceniu, nie posiadając wiedzy o tym jakiej procedury weryfikacji sterownik będzie próbował dokonać. Tę możliwość osiągamy dzięki efektywnemu 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 };

```

Tabela 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 }

```

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

5.1.4 Strategia

Znanych jest wiele wzorców komunikacji pomiędzy komponentowej, jako dwa najbardziej znane oznano Publish-Subscribe oraz Request-Response. Pierwszy z nich służy efektywnie realizuje odwrócenie zależności ponieważ urządzenie nadzorne na początkowym etapie nie musi znać ilości podłączonych do linii urządzeń. Drugi natomiast pozwala nam zrealizować podejście do transmisji z urządzeniem znane jako półduplex, wymagane podczas wywoływania komend zestawiających warstwę łącząca danych oraz aplikacyjnej. Sek w tym, że obydwa wzorce wymagają innego zestawu komend w celu konfiguracji połączenia ze slotami systemowymi. W projekcie zaimplementowana jeden kontroler, który zarządza zestawianiem wymaganych

warstw OSI w trakcie ciągłego uruchomienia sterownika, co osiągnięto dzięki dynamicznemu podmianie strategii komunikacji z Publish-Subscribe na Request-Response. Zaobserwowano pewien „zły zapach” kodu polegający na prewencyjnemu rzuceniu wyjątku w przypadku gdyby programista wywołał niepoprawną metodę ze strategii. Strategia jest potężnym narzędziem, nad którym zaplanowane zostały dalsze plany analizy oraz zrozumienia, w celu eliminacji tego typu wywołaniom.

```
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 }
```

Tabela 5.7. Strategia komunikacji request-response dla urządzenia nadzawanego

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 muzyk”. 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 zaobserwować konieczność przekazania powyższych obiektów z poziomu testu, w celu wyeliminowania konieczności 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 wołania konstruktora, aniżeli utworzeniu konstruktora zeroparametrowego, który uniemożliwia dalszą modernizację elementów składowych systemu.

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

Tabela 5.8. Konstruktor zrealizowany podejściem wstrzykiwania zależności

5.1.6 Zabór zasobu jest inicjalizacją

C++ w wersji 11-tej został rozbudowany o 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. RAII² 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 jesteśmy ustrzeżeni przed niepożdanym 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 jako „brzydki zapach kodu” oraz antywzorzec. W tej sytuacji z pomocą pojawia się wzorzec budowniczego, który

² RAII - ang. Resource acquisition is initialization

podczas wywoływania metody zmieniającej stan obiektu, dokonuje zamierzonego celu, po czym zwraca referencję na obiekt klasy modyfikowanej. To pozwala szeregowo wywołać kolejne modyfikatory co znacznie oczyściło i zwiększyło czytelność programu. Istnieje wiele interpretacji tego wzorca projektowego, w których istnieje również metoda finalizująca budowanie obiektu.

```
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 }
```

Tabela 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 podczas każdego wywołania inaczej. Natomiast 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. W przypadku nieprzechowywania jakiegokolwiek stanu w jej instancji oraz zapoznania się w całości z realizowanym problemem komunikacji pomiędzy urządzeniem podrzędnym i nadrzędnym, 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 obsługiwać komunikację oraz budowę wiadomości charakterystyczną 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;
```

5. Wzorce projektowe

13 } ;

Tabela 5.10. Fabryka budowniczych dla sterownika urządzenia nadzorującego

6. Wymagania funkcjonalne

6.1 Historyki użytkownika (Warstwa fizyczna łącza danych) - nazwa komendy

Częstą praktyką modelowania etapu łączenia z RET-em jest użycie przy maszynie stanowej w związku z czym, z poniższych funkcjonalności należy korzystać zgodnie z zadaną kolejnością. Dla każdej z poniższych ramek wyznaczono część wspólną składającą się z czterech bajtów. Pierwszym z nich jest bajt flagi startu, która oznacza początek wiadomości i ma wartość 0x7E. Koleje bajty budują ramkę XID, I, U, S o których wspomniano wcześniej. Następnie dodaje się dwubajtową sumę CRC oraz na samym końcu flagę stopu, która jest tej samej wartości co flaga startu i oznacza koniec nadawania wiadomości.

6.1.1 Ustanowienie prędkości połączenia - SetLinkSpeed

Jako użytkownik chcę ustanowić prędkość transmisji na 9.6 kbps.

Jest to komenda wysyłana do urządzenia podlegającego przy pomocy ramki XID.

Pierwsze dwa bajty o wartości 0xFF odpowiadają za zdefiniowanie adresu urządzenia docelowego. W tym przypadku celem jest synchronizacja prędkości połączenia ze wszystkimi urządzeniami na lini, które nie są zaadresowane a więc jest to wiadomość typu „broadcast”.

Kolejne dwa bajty kontrolne o wartości 0xBF są charakterystyczne dla ramki XID.

Następne dwa bajty o wartości 0x81 identyfikują format wiadomości XID, co oznacza że jest to wiadomość z kategorii przypisania adresu.

Potem napotykamy kolejne dwa bajty o wartości 0xF0 które są identyfikatorem grupy, co mówi nam jakie kolejne bajty dopuszczalne są w dalszej części wiadomości.

Urządzenie podlegające w celu umożliwienia mu zdekodowania wiadomości, otrzymuje w wiadomości długość grupy, czyli dwa bajty w tym przypadku o wartości 0x08 i jest to liczba bajtów która pojawi się następnie, aż do pierwszego bajta sumy CRC.

Przechodząc do przesyłanych parameterów, jako pierwszy będzie to unikalny identyfikator urządzenia. Rozpoznano go po identyfikatorze parametru o wartości 0x01. Następnie podajemy liczbę bajtów zajmowanych przez UniqueID czyli 0x02. Kolejnie podajemy wartość unikalnego identyfikatora. Z racji tego, że nie oczekujemy odpowiedzi na tę wiadomość, są to dowolne wartości. Drugim przesyłanym parametrem jest maska danych z identyfikatorem o wartości 0x03, potem podajemy liczbę bajtów budujących przesyłaną przez nas maskę czyli 0x02, a w wartości parametru umieszczały 4-ry bajty o wartości obu 0xFF. Jest to najbardziej ogólna maska urządzenia.

6.1.2 Negocjacja roli - AddressAssignment

Jako użytkownik chcę zaadresować urządzenie typu SingleRET identyfikujące się unikalnym identyfikatorem¹ == {0x4E, 0x4B, 0x34, 0x36, 0x35, 0x30, 0x30, 0x30} adresem 3, przy pomocy komendy „AddressAssignment”.

Budowa ramki wysyłanej przez tę komendę jest bardzo podobna do powyższej z racji tego, że powyższą można nazwać atrapą skanu a tą skanowaniem celowanym. Z racji tego, że różna jest liczba oraz wielkość poszczególnych parametrów HDLC, bajt rozmiaru grupy posiada wartość 0x11.

Pierwszy parametr identyfikuje się przy pomocy wartości 0x01 czyli jest to unikalny identyfikator urządzenia, który ma zostać zaadresowany. Kolejny bajt przypomina o rozmiarze parametru i w tym przypadku jest on równy 0x09. W powyższym opisie przedstawiono unikalny identyfikator urządzenia, który w przypadku równejści z rzeczywistym, pozwala urządzeniu podzielenemu przyjąć żądanie adresacji.

Następny parametr posiada wartość 0x02 i oznacza on, że w jego wartości urządzenie nadzędne zdefiniowało adres urządzeniu wewnętrznemu, dzięki któremu będzie identyfikowane oraz z racji tego, że długość parametru ma wartość 0x01, to wartość wynosi 0x03. W przypadku projektu, nie będzie miał on większego znaczenia aczkolwiek w momencie kiedy urządzenia typu Ret połączone są ze sobą szerogowo, potrzeba jest możliwość wysłania wiadomości do na przykład drugiego w łańcuchu.

Ostatnim parametrem jest 0x04 który mówi nam o tym, że urządzenie nadzędne uważa, że po drugiej stronie jest jedno z urządzeń zdefiniowanych według standardu AISG 2.0, w tym przypadku długość również wynosi 0x01 a wartość to 0x01. Oznacza ona, że próbujemy nawiązać komunikację z pojedynczym RET-em. Dla porównania istnieją również urządzenia takie jak MultiRET, dzięki któremu można zmieniać wartość nachylania kąta głównej wiązki anteny na wielu osiach czy płaszczyznach.

6.1.3 Negocjacja parametrów HDLC - HDLCParameters

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

Powyższą wiadomość można rozbić na 4-ry osobne aczkolwiek połączono je z racji podobnych funkcjonalności realizowanych podczas jej wysłania. Protokół AISG bazuje na HDLC, które używane jest w wielu miejscach gdzie potrzebne jest połączenie połączenie o wysokiej gwarancji otrzymania wiadomości bez utraty informacji. Istnieje nawet jego realizacja służąca do komunikacji satelit kosmicznych i różni się ona od AISG głównie ustalonimi wiadomościami HDLC, gdzie potrzeba wysłać znacznie większą liczbę ramek na raz aniżeli tylko jedna.

¹ UniqueID - Dla AISG v2.0 jest to złożenie numeru seryjnego wraz z kodem producenta

6. Wymagania funkcjonalne

Pierwszy bajt adresu może zostać zmieniony z wartości 0xFF na 0x03, gdyż w poprzedniej wiadomości ustanowiono adres urządzenia podległego. Ta wartość będzie aplikowana do każdej poniższej wiadomości a więc pominięto jej wspominanie, gdyż potraktowano ją jako stałą. Następnym bajtem jest identyfikator formatu, który dla wiadomości zawierającej parametry HDLC jest równy 0x81.

Kolejny bajt czyli identyfikator grupy ma wartość 0x80, a następny (rozmiar grupy) jest równy 0x12.

Pierwszym parametrem HDLC który jest negocjowany to maksymalny rozmiar tzw. payloadu dla ramki informacyjnej nadawanej przez urządzenie nadzorujące. Identyfikatorem w tym przypadku jest 0x05. Rozmiar payloadu jest równy 0x04 a wartościami są { 0xF0, 0x2D, 0x00, 0x00 }.

Kolejny bajty natomiast budują negocjowany maksymalny rozmiar payloadu ramki informacyjnej do wiadomości otrzymywanej przez urządzenie nadzorujące. Identyfikator ma wartość 0x06 a pozostały bajty są tożsame jak dla poprzedniej negocjacji.

Następnym parametrem jest maksymalna liczba ramek wysłanych pod rząd przez urządzenie nadzorujące. Identyfikator ma wartość 0x07 a zarówno długość jak i wartość to 0x01.

Ostatnim parametrem negocjowanym podczas tej wiadomości jest maksymalna liczba ramek otrzymanych pod rząd poprzez urządzenie nadzorujące. Wartość identyfikatora jest równa 0x08 a długość oraz wartość znów posiadają wartości 0x01.

6.1.4 Ustanowienie synchronicznego trybu komunikacji - LinkEstablishment

Jako użytkownik chcę ustawić synchroniczny tryb komunikacji z urządzeniem.

Bajt kontrolny posiada wartość 0x93 i oznacza żądanie ustalenia sposobu komunikacji z urządzeniem podległym na tryb normalny, a więc taki w którym wysyła ono ramkę jedynie jako odpowiedź na wcześniej nadaną przez urządzenie nadzorujące.

6.1.5 Negocjacja parametrów HDLC - 3GPPReleaseID

Jako użytkownik chcę wynegocjować parametry HDLC urządzenia, jakim jest wspierana wersja 3GPP.

Teraz można przejść do coraz bardziej szczegółowych parametrów jak ustanowienie numeru wersji standardu 3GPP.² Podczas tworzenia tej wiadomości znów skorzystano z ramki XID a więc należy zdefiniować identyfikator formatu, grupy oraz jej długość. Pierwsze dwie wartości

² 3GPP - 3rd Generation Partnership Project, komisja standaryzacyjna która tworzy protokoły telekomunikacyjne.

są równe tym z komendy „AddressAssignment” a długość ma wartość 0x03. Identyfikatorem parametru jest tym razem wartość 0x05, długością 0x01 natomiast wartością parametru 0x08. Jest to wersja standardu w wersji 8-mej czyli pionierska dla technologii LTE³. Wprowadzono w niej wparcie dla interfejsu radiowego opartego o OFDMA⁴, którego wykorzystanie można zaobserwować również przy technologii 5G.

6.1.6 Negocjacja parametrów HDLC - AISGProtocolVersion

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

Z racji tego, że implementowana wersja protokołu AISG 2.0 jest jedną z wielu (istnieją jeszcze dwie wcześniejsze: 1.0, 1.1 oraz najnowsza 3.0), potrzeba zdefiniować według którego standardu budowane są wiadomości przez urządzenie nadzędne oraz rozpoznawane przez podziale. Bajt identyfikujący parametr dla tej wiadomości ma wartość 0x14, długość parametru to 0x01 a jej wartość to 0x02.

6.2 Historyjki użytkownika (Warstwa aplikacyjna) - nazwa komendy

Po pomyślnym zestawieniu warstwy fizycznej posiadamy zdefiniowany wszystkie charakterystyki połączenia, które pozwolą w jednoznaczny sposób porozumieć się z urządzeniem oraz żądać odpowiedzi na wysłaną wiadomość. Kolejne komendy są już żądaniemi wysokociomowymi, które definiują użyteczne dla klienta polecenia służąca do zarządzania RET-em. Jest ich znacznie więcej, aczkolwiek obranym celem pracy inżynierskiej było pomyślne przeprowadzenie kalibracji urządzenia, co pozwala w dalszym etapie ustawić zadany kąt odchylenia głównej wiązki sygnału z anteny.

6.2.1 Kalibracja - Calibrate

Jako użytkownik chcę skalibrować urządzenie.

Z racji tego, że wiadomości tej warstwy korzystają z ramki informacyjnej, inną rolę pełni bajt kontrolny. Poza stałymi wartości bitów w bajcie, każdorazowe wysłanie i odebranie wiadomości zmienia wartość sekcji P/F wcześniej wspomnianej. Dzięki temu można w łatwy sposób identyfikować czy wiadomość którą otrzymujemy jest ta której oczekiwaliśmy. W związku z tym dla pierwszej wiadomości wysłanej przez urządzenie nadzędne poprawną wartością, będzie 0x10. Kolejnym bajtem jest kod procedury, który ma wartość 0x31, co oznacza, że oczekujemy kalibracji urządzenia będącego zwykłym pojedynczym RET-em.

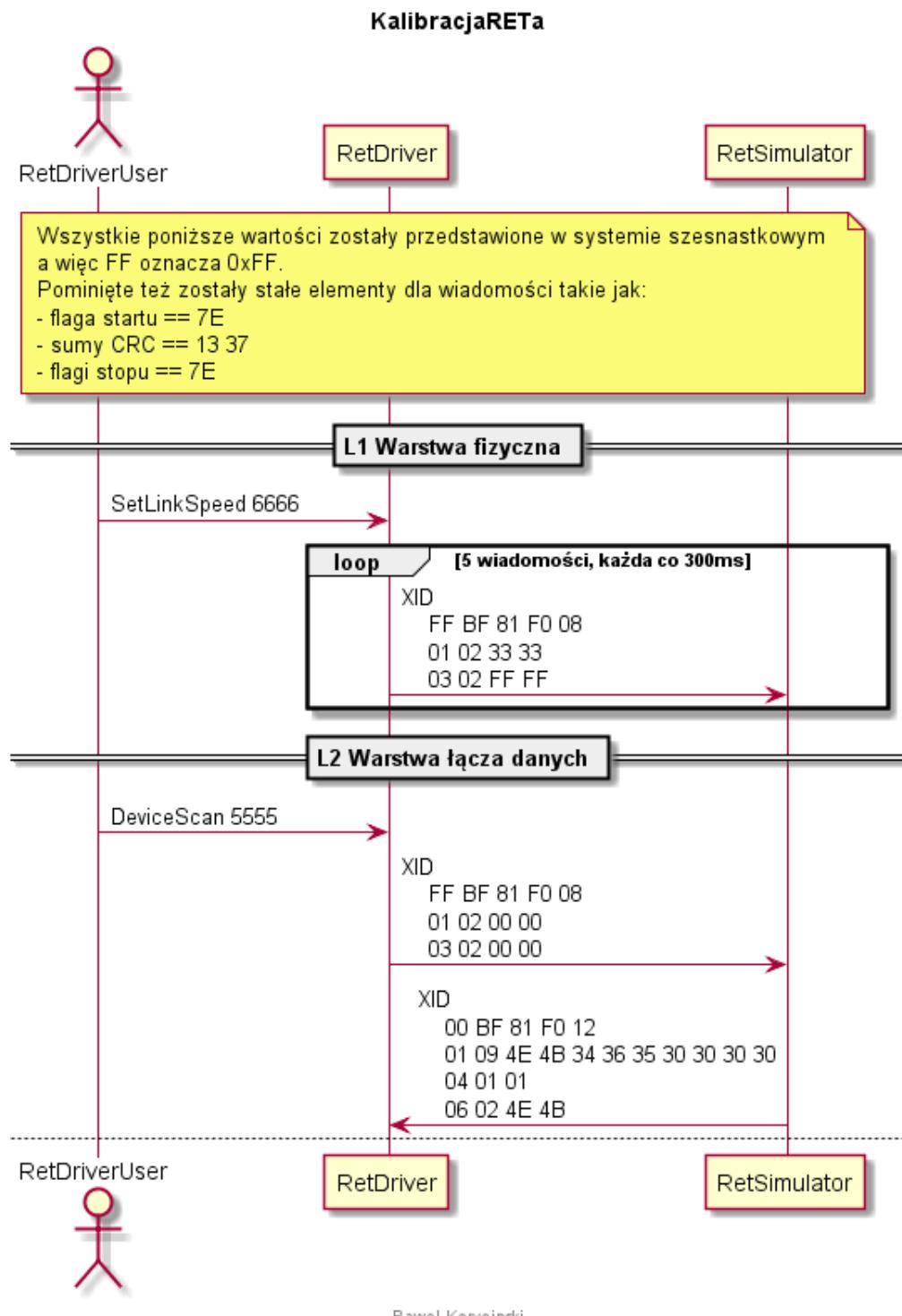
³ LTE - Long term evolution.

⁴ OFDMA - Orthogonal frequency-division multiple access.

6. Wymagania funkcjonalne

Z racji tego, że podczas tej komendy nie jest przesyłana żadna wartość dla tej procedury, dwa bajty zaalokowane dla długości parametru posiadają wartość {0x00, 0x00}

7. Analiza nawiązanej komunikacji



Rysunek 7.1. Ustanowienie prędkości połączenia wraz z początkowym skanowaniem urządzeń

Na diagramach sekwencji przedstawiono komendy wywoływanie przez użytkownika symulatora wraz z zawartościami ramek pochodzących z urządzenia nadziednego oraz wartościami ramek przychodzących z urządzenia podziednego. W celu analizy procesu komunikacji, w opisie skupiono się na cechach charakterystycznych dla poszczególnej ramki czy komendy, o których nie wspomniano we wcześniejszych rozdziałach.

7.1 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 zastosowania wzorca Publish-Subscribe. Podczas tego połączenia protokół AISG 2.0 nie zakłada oczekiwania na odpowiedź od urządzenia podziednego oraz użyta biblioteka ZeroMQ również nie udostępnia możliwości wysłania odpowiedzi na taką wiadomość.

7.2 Skanowanie urządzeń

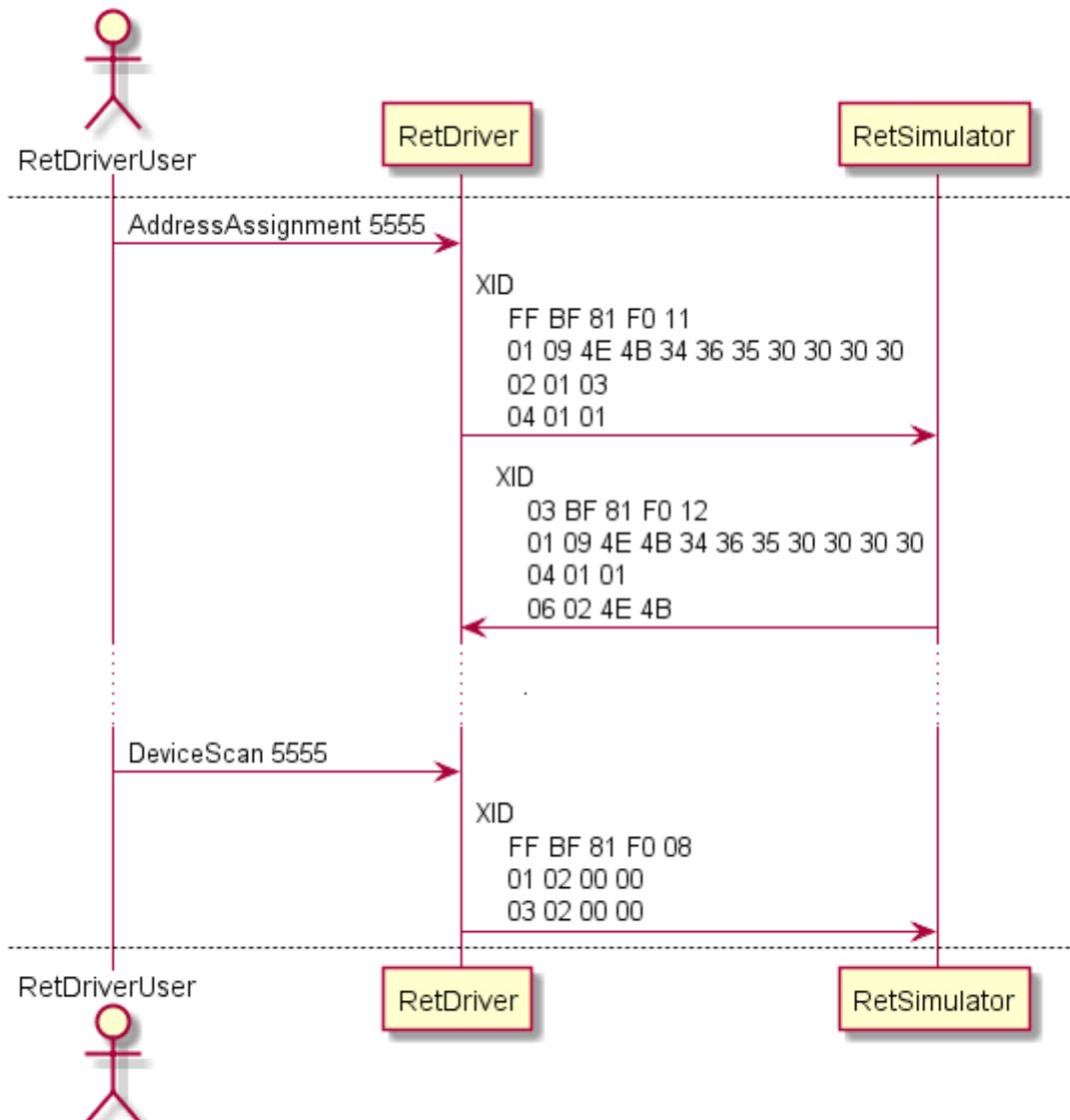
Parametrem tej komendy jest adres portu 5555, z którego korzysta sterownik do nawiązania połączenia typu tcp na adresie 127.0.0.1 wraz z symulatorem urządzenia, przy zastosowania wzorca Request-Response. Jest to wzorzec, który zakłada otrzymanie odpowiedzi na każdą wyslaną wiadomość, zanim kolejna zostanie nadana. Opisany mechanizm komunikacji aplikowalny jest również do wszystkich poniższych komend. Ciekawym elementem tej wiadomości jest to, że otrzymano kod producenta zarówno w osobnym parametrze jak i jako składową unikalnego identyfikatora urządzenia.

7.3 Żadanie adresacji

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

7.4 Ponowne skanowanie urządzeń

W zakresie tego projektu komunikacja nawiązywana jest z jednym urządzeniem, a więc dla czego ponownie wysłano wiadomość skanowania? Otóż na tę wiadomość urządzenie nadziedne nie powinno otrzymać odpowiedzi, gdyż jedynie urządzenia niezaadresowane mogą na nie odpowiedzieć, co jest podstawową metodą dodatkowej weryfikacji powodzenia procedury adresowania.



Pawel Korycinski

Rysunek 7.2. Żadanie adresacji oraz dodatkowe skanowanie urządzeń

7.5 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 payloadu¹ dla wysłanej oraz otrzymanej ramki informacyjnej. Urządzenie nadziedzne próbowało ustanowić dopuszczalną liczbę bitów na {0xF0, 0x2D, 0x00, 0x00} co daje wartość 61485, lecz urządzenie podrzędne odpowiedziało {0x50, 0x02, 0x00, 0x00} co po

¹ Payload - Fragment wiadomości, który nie zawiera bajtów takich jak nagłówki bądź metadane a jedynie „istotne” informacje.

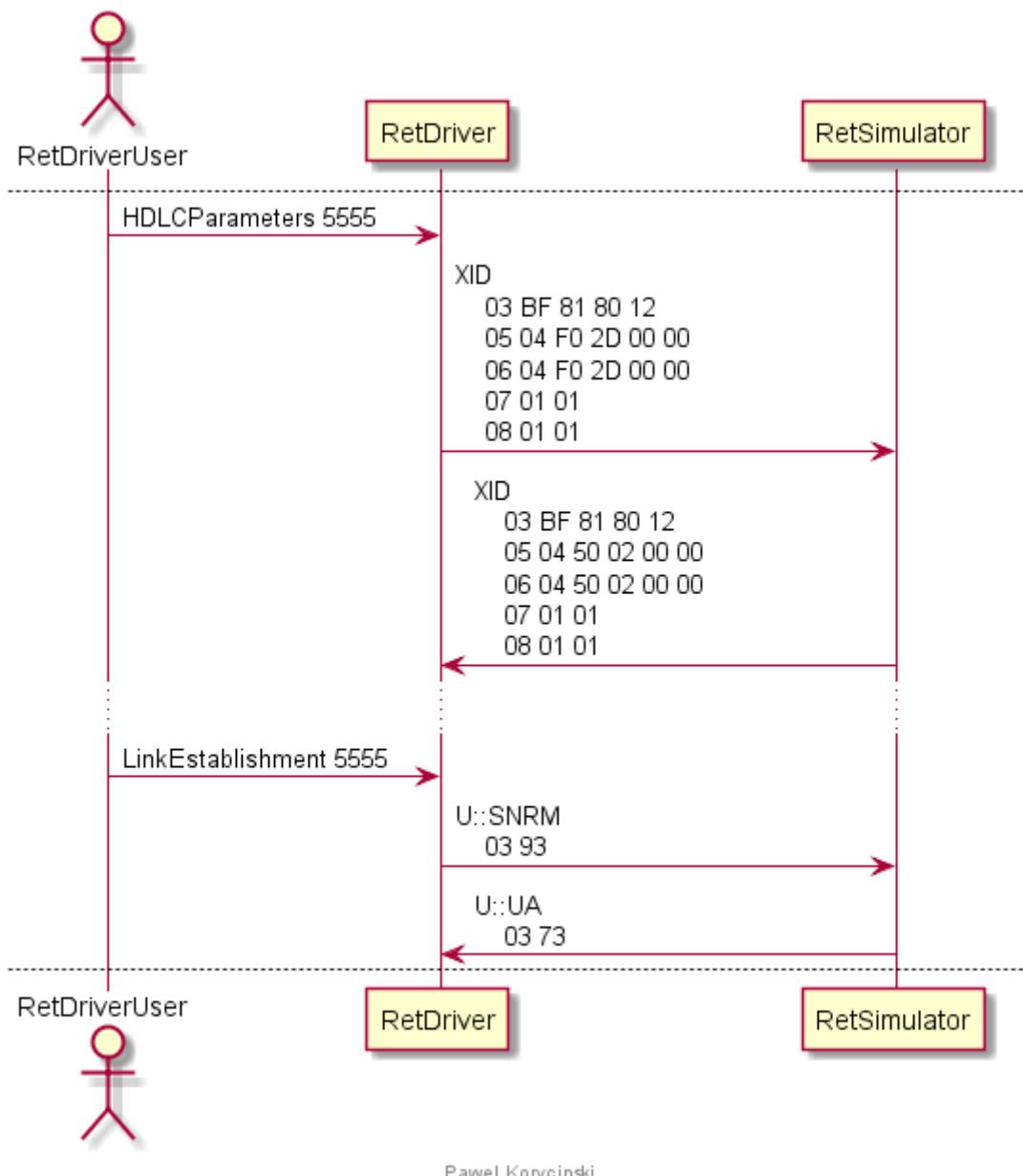
konwersji na system dziesiętny daje 20482 bity.

7.6 Przejście na normalny tryb komunikacji

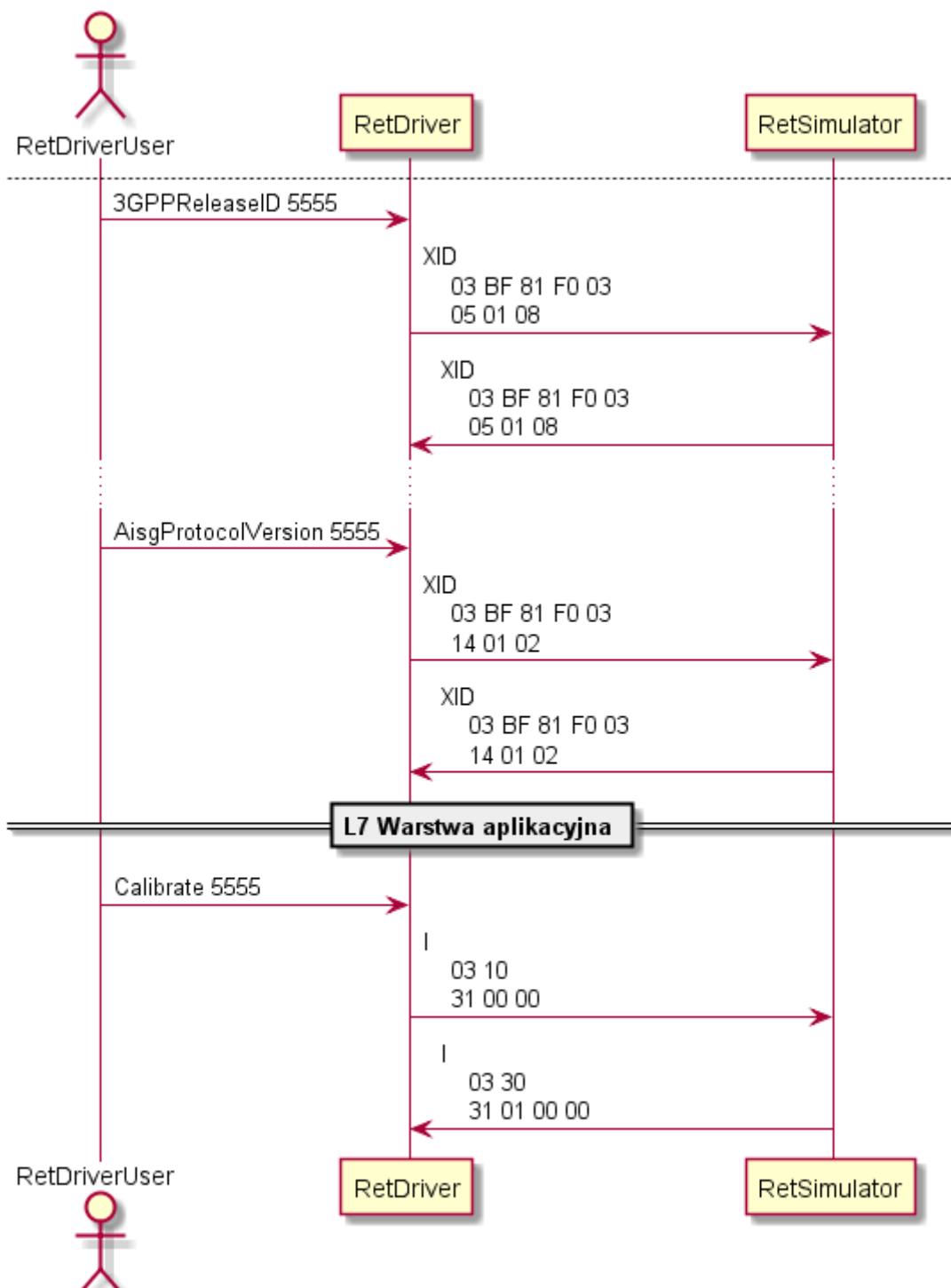
Jako odpowiedź na tę wiadomość urządzenie otrzyma ramkę, która zawiera wartość 0x73, co oznacza że potwierdza ono żądane oczekiwanie.

7.7 Kalibracja

W związku z tym, że oczekiwanie kalibracji przesłano przy pomocy ramki informacyjnej, bajt kontrolny posiada charakterystyczną metodę jego ewaluacji przedstawioną wcześniej. Diagram sekwencji potwierdza to, gdyż wiadomość odebrana przez urządzenie nadzędne posiada wartość równą 0x30. Ramka odebrana, na wyznaczenie liczby bajtów budujących odpowiedź posiada zarezerowane dwa bajty. Ciekawą obserwacją jest to, że odpowiedź równą 0x00 czyli OK można by zapisać przy pomocy jedynie jednego bajta, lecz optymalizacja pamięci nie jest tutaj zastosowana.



Rysunek 7.3. Negocjacja rozmiaru okna oraz payloadu ramki informacyjnej oraz ustanowienie normalnego trybu komunikacji



Pawel Korycinski

Rysunek 7.4. Negocjacja pozostałych parametrów HDLC oraz zadanie kalibracji urządzenia

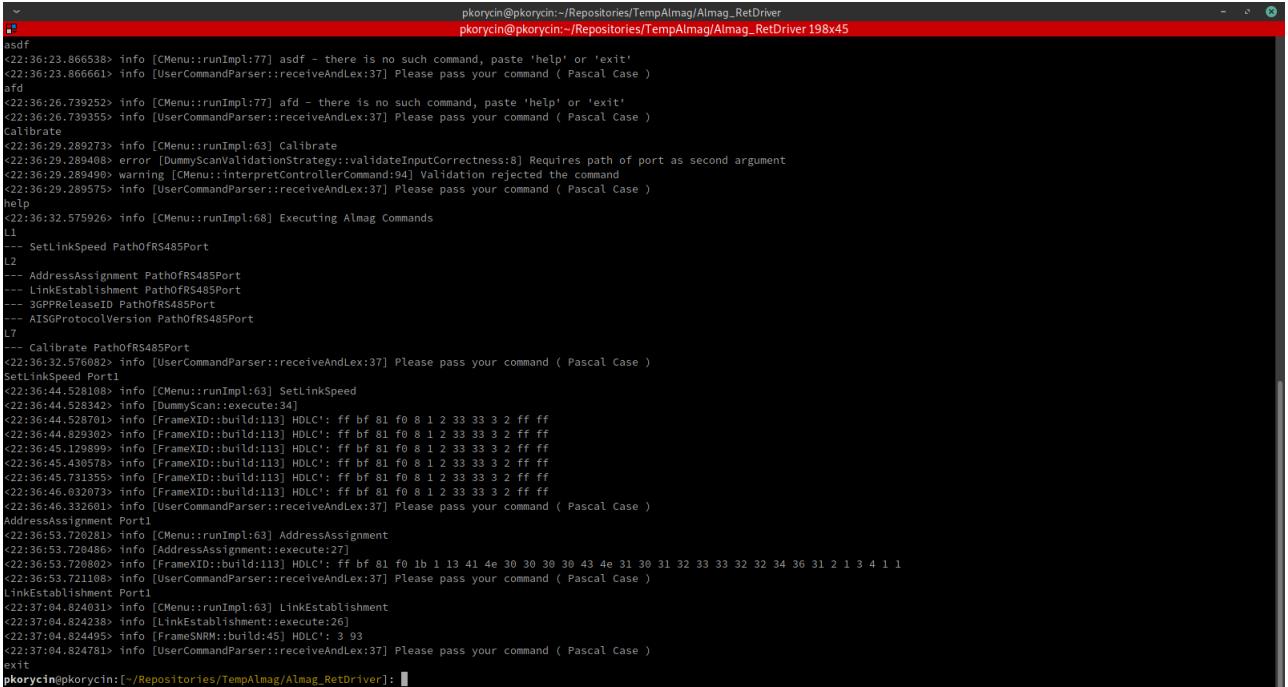
8. Wymagania niefunkcjonalne

8.1 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;
- Podłączenie do internetu w celu pobrania repozytorium oraz komplikacji programu uruchomieniowego;
- Komendy wywoływanie w pierwszym oknie konsoli, w celu uruchomienia symulatora urządzenia:
 1. git clone https://github.com/trunksBT/KorytkoMag_RetSimulator.git --recursive
 2. cd KorytkoMag_RetSimulator
 3. make .
 4. bash runBinary.sh
- Komendy wywoływanie w drugim oknie konsoli, w celu uruchomienia sterownika urządzenia:
 1. git clone https://github.com/trunksBT/KorytkoMag_RetSimulator.git --recursive
 2. cd KorytkoMag_RetSimulator
 3. make .
 4. bash runBinary.sh
-

8.2 Zrzuty ekranu

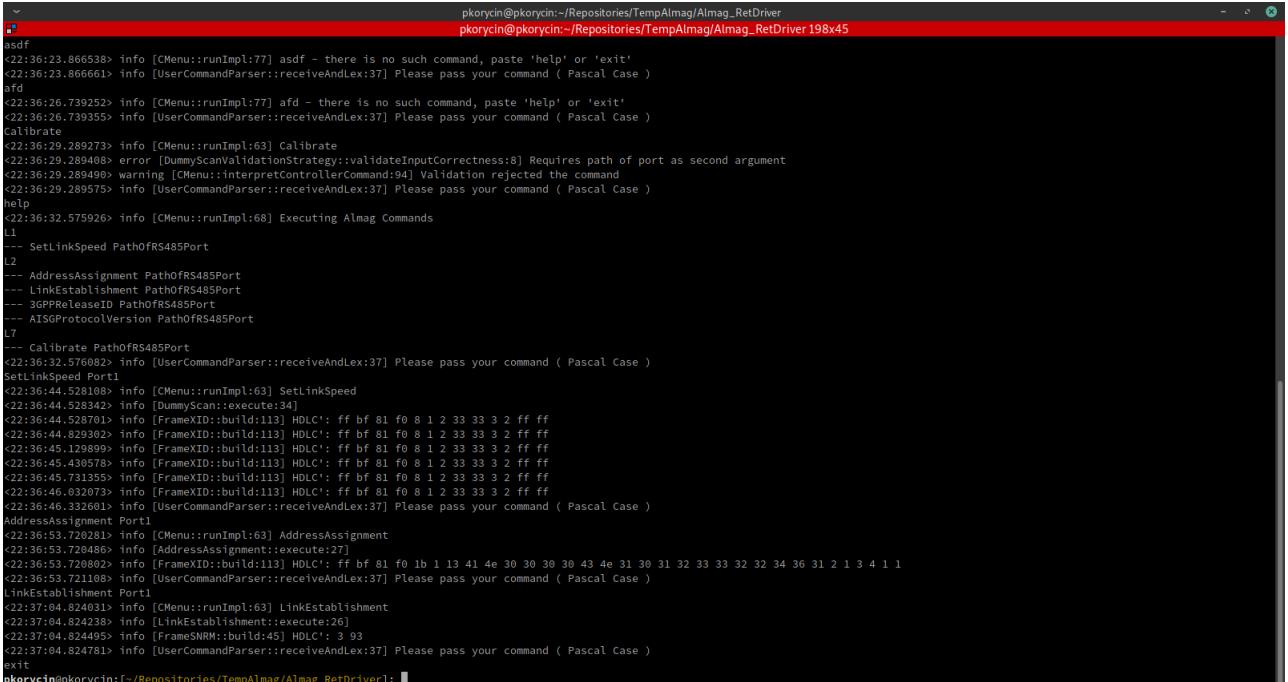
8. Wymagania niefunkcjonalne



```
pkorycin@pkorycin:~/Repositories/TempAlmag/Almag_RetDriver
pkorycin@pkorycin:~/Repositories/TempAlmag/Almag_RetDriver 198x45

asdf
<22:36:23.866538> info [CMenu::runImpl:77] asdf - there is no such command, paste 'help' or 'exit'
<22:36:23.866661> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
afd
<22:36:26.739252> info [CMenu::runImpl:77] afd - there is no such command, paste 'help' or 'exit'
<22:36:26.739355> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
calibrate
<22:36:29.289273> info [CMenu::runImpl:63] Calibrate
<22:36:29.289408> error [DummyScanValidationStrategy::validateInputCorrectness:8] Requires path of port as second argument
<22:36:29.289409> warning [CMenu::interpretControllerCommand:94] Validation rejected the command
<22:36:29.289575> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
help
<22:36:32.575926> info [CMenu::runImpl:68] Executing Almag Commands
L1
--- SetLinkSpeed PathOfRS485Port
L2
--- AddressAssignment PathOfRS485Port
--- LinkEstablishment PathOfRS485Port
--- 3GPPReleaseID PathOfRS485Port
--- AISGPProtocolVersion PathOfRS485Port
L7
--- Calibrate PathOfRS485Port
<22:36:32.576082> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
SetLinkSpeed Port1
<22:36:44.528108> info [CMenu::runImpl:63] SetLinkSpeed
<22:36:44.528342> info [DummyScan::execute:34]
<22:36:44.528701> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:44.829392> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.129899> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.430578> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.731355> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.932073> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:46.332601> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
AddressAssignment Port1
<22:36:53.720281> info [CMenu::runImpl:63] AddressAssignment
<22:36:53.720465> info [AddressAssignment::execute:27]
<22:36:53.720802> info [FrameID::build:113] HDLC: ff bf 81 f0 1b 1 13 41 4e 30 30 30 43 4e 31 30 31 32 33 32 34 36 31 2 1 3 4 1 1
<22:36:53.721108> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
LinkEstablishment Port1
<22:37:04.624091> info [CMenu::runImpl:63] LinkEstablishment
<22:37:04.624238> info [LinkEstablishment::execute:26]
<22:37:04.624495> info [FrameNRM::build:45] HDLC: 3 93
<22:37:04.624781> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
exit
pkorycin@pkorycin:[~/Repositories/TempAlmag/Almag_RetDriver]:
```

Rysunek 8.1. Zrzut ekranu z wykonania testów modułowych

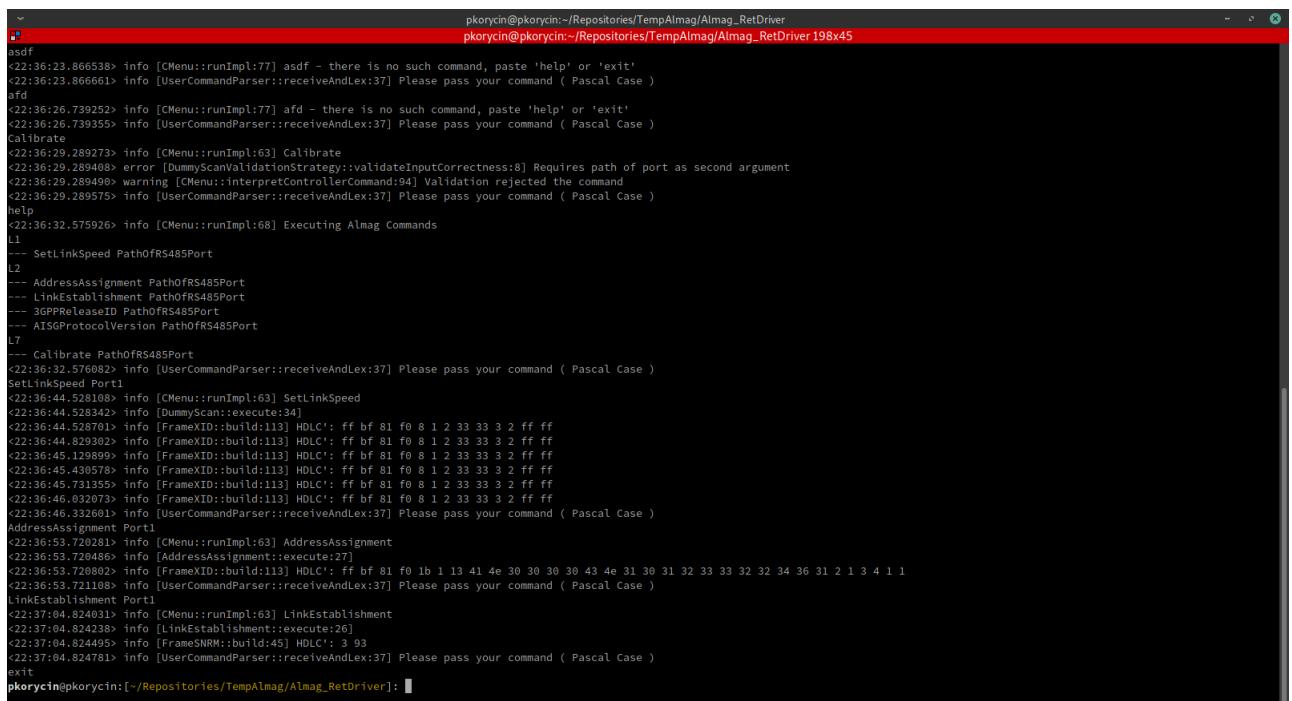


```
pkorycin@pkorycin:~/Repositories/TempAlmag/Almag_RetDriver
pkorycin@pkorycin:~/Repositories/TempAlmag/Almag_RetDriver 198x45

asdf
<22:36:23.866538> info [CMenu::runImpl:77] asdf - there is no such command, paste 'help' or 'exit'
<22:36:23.866661> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
afd
<22:36:26.739252> info [CMenu::runImpl:77] afd - there is no such command, paste 'help' or 'exit'
<22:36:26.739355> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
calibrate
<22:36:29.289273> info [CMenu::runImpl:63] Calibrate
<22:36:29.289408> error [DummyScanValidationStrategy::validateInputCorrectness:8] Requires path of port as second argument
<22:36:29.289409> warning [CMenu::interpretControllerCommand:94] Validation rejected the command
<22:36:29.289575> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
help
<22:36:32.575926> info [CMenu::runImpl:68] Executing Almag Commands
L1
--- SetLinkSpeed PathOfRS485Port
L2
--- AddressAssignment PathOfRS485Port
--- LinkEstablishment PathOfRS485Port
--- 3GPPReleaseID PathOfRS485Port
--- AISGPProtocolVersion PathOfRS485Port
L7
--- Calibrate PathOfRS485Port
<22:36:32.576082> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
SetLinkSpeed Port1
<22:36:44.528108> info [CMenu::runImpl:63] SetLinkSpeed
<22:36:44.528342> info [DummyScan::execute:34]
<22:36:44.528701> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:44.829392> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.129899> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.430578> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.731355> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.932073> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:46.332601> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
AddressAssignment Port1
<22:36:53.720281> info [CMenu::runImpl:63] AddressAssignment
<22:36:53.720465> info [AddressAssignment::execute:27]
<22:36:53.720802> info [FrameID::build:113] HDLC: ff bf 81 f0 1b 1 13 41 4e 30 30 30 30 43 4e 31 30 31 32 33 32 34 36 31 2 1 3 4 1 1
<22:36:53.721108> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
LinkEstablishment Port1
<22:37:04.624091> info [CMenu::runImpl:63] LinkEstablishment
<22:37:04.624238> info [LinkEstablishment::execute:26]
<22:37:04.624495> info [FrameNRM::build:45] HDLC: 3 93
<22:37:04.624781> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
exit
pkorycin@pkorycin:[~/Repositories/TempAlmag/Almag_RetDriver]:
```

Rysunek 8.2. Zrzut ekranu z wykonania testów jednostkowych

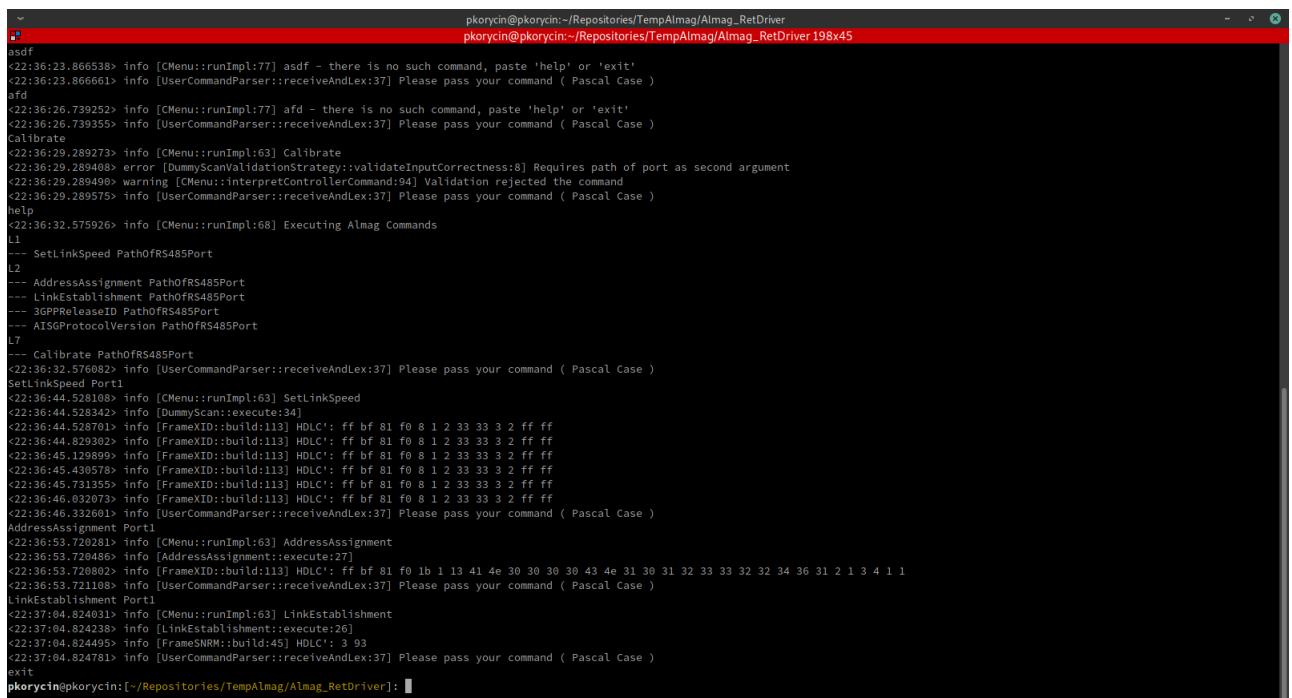
8. Wymagania niefunkcjonalne



```
pkorycin@pkorycin:~/Repositories/TempAlmag/Almag_RetDriver
pkorycin@pkorycin:~/Repositories/TempAlmag/Almag_RetDriver 198x45

asdf
<22:36:23.8666538> info [CMenu::runImpl:77] asdf - there is no such command, paste 'help' or 'exit'
<22:36:23.866661> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
afd
<22:36:26.739252> info [CMenu::runImpl:77] afd - there is no such command, paste 'help' or 'exit'
<22:36:26.739355> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
calibrate
<22:36:29.289273> info [CMenu::runImpl:63] Calibrate
<22:36:29.289408> error [DummyScanValidationStrategy::validateInputCorrectness:8] Requires path of port as second argument
<22:36:29.289409> warning [CMenu::interpretControllerCommand:94] Validation rejected the command
<22:36:29.289575> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
help
<22:36:32.575926> info [CMenu::runImpl:68] Executing Almag Commands
L1
--- SetLinkSpeed PathOfRS485Port
L2
--- AddressAssignment PathOfRS485Port
--- LinkEstablishment PathOfRS485Port
--- 3GPPReleaseID PathOfRS485Port
--- AISGProtocolVersion PathOfRS485Port
L7
--- Calibrate PathOfRS485Port
<22:36:32.576082> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
SetLinkSpeed Port1
<22:36:44.528108> info [CMenu::runImpl:63] SetLinkSpeed
<22:36:44.528342> info [DummyScan::execute:34]
<22:36:44.528701> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:44.829392> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.129899> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.430578> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.731355> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.932073> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:46.332601> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
AddressAssignment Port1
<22:36:53.720281> info [CMenu::runImpl:63] AddressAssignment
<22:36:53.720486> info [AddressAssignment::execute:27]
<22:36:53.720802> info [FrameID::build:113] HDLC: ff bf 81 f0 1b 1 13 41 4e 30 30 30 43 4e 31 30 31 32 33 32 34 36 31 2 1 3 4 1 1
<22:36:53.721108> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
LinkEstablishment Port1
<22:37:04.624091> info [CMenu::runImpl:63] LinkEstablishment
<22:37:04.624238> info [LinkEstablishment::execute:26]
<22:37:04.624495> info [FrameNRM::build:45] HDLC: 3 93
<22:37:04.624781> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
exit
pkorycin@pkorycin:[~/Repositories/TempAlmag/Almag_RetDriver]:
```

Rysunek 8.3. Zrzut ekranu z uruchomienia programu z odfiltrowaniem logów poniżej priorytetu info



```
pkorycin@pkorycin:~/Repositories/TempAlmag/Almag_RetDriver
pkorycin@pkorycin:~/Repositories/TempAlmag/Almag_RetDriver 198x45

asdf
<22:36:23.8666538> info [CMenu::runImpl:77] asdf - there is no such command, paste 'help' or 'exit'
<22:36:23.866661> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
afd
<22:36:26.739252> info [CMenu::runImpl:77] afd - there is no such command, paste 'help' or 'exit'
<22:36:26.739355> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
calibrate
<22:36:29.289273> info [CMenu::runImpl:63] Calibrate
<22:36:29.289408> error [DummyScanValidationStrategy::validateInputCorrectness:8] Requires path of port as second argument
<22:36:29.289409> warning [CMenu::interpretControllerCommand:94] Validation rejected the command
<22:36:29.289575> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
help
<22:36:32.575926> info [CMenu::runImpl:68] Executing Almag Commands
L1
--- SetLinkSpeed PathOfRS485Port
L2
--- AddressAssignment PathOfRS485Port
--- LinkEstablishment PathOfRS485Port
--- 3GPPReleaseID PathOfRS485Port
--- AISGProtocolVersion PathOfRS485Port
L7
--- Calibrate PathOfRS485Port
<22:36:32.576082> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
SetLinkSpeed Port1
<22:36:44.528108> info [CMenu::runImpl:63] SetLinkSpeed
<22:36:44.528342> info [DummyScan::execute:34]
<22:36:44.528701> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:44.829392> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.129899> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.430578> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.731355> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:45.932073> info [FrameID::build:113] HDLC: ff bf 81 f0 8 1 2 33 33 3 2 ff ff
<22:36:46.332601> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
AddressAssignment Port1
<22:36:53.720281> info [CMenu::runImpl:63] AddressAssignment
<22:36:53.720486> info [AddressAssignment::execute:27]
<22:36:53.720802> info [FrameID::build:113] HDLC: ff bf 81 f0 1b 1 13 41 4e 30 30 30 43 4e 31 30 31 32 33 32 34 36 31 2 1 3 4 1 1
<22:36:53.721108> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
LinkEstablishment Port1
<22:37:04.624091> info [CMenu::runImpl:63] LinkEstablishment
<22:37:04.624238> info [LinkEstablishment::execute:26]
<22:37:04.624495> info [FrameNRM::build:45] HDLC: 3 93
<22:37:04.624781> info [UserCommandParser::receiveAndLex:37] Please pass your command ( Pascal Case )
exit
pkorycin@pkorycin:[~/Repositories/TempAlmag/Almag_RetDriver]:
```

Rysunek 8.4. Zrzut ekranu z uruchomienia programu bez odfiltrowania logów

9. Plan dalszego rozwoju

9.1 Porażki odnośnie planu początkowego

W celu przesłania wiadomości zapisanej w systemie heksadecymalnym przy pomocy interfejsu USB <-> RS-485 należy dokonać konwersji na postać binarną. Zarówno wiadomość odebrana jak i wysłana może w trakcie transmisji ulec wybrakowaniu bądź zmianie zawartości. W tym celu ramka HDLC protokołu AISG 2.0 posiada dwa bajty przeznaczone na validację 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 potrzebowałem więcej informacji na temat zarówno endianowości jak i uporządkowania bitów, których na moment opracowywania algorytmu nie posiadałem. 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 podczas którego sprawdę umiejętności testowania, wdrażania wzorców projektowych czy też posługiwania się językiem C++, za miast walczyć z urządzeniem i jego możliwymi błędami, postanowiłem utworzyć symulator urządzenia, z którym sterownik będzie komunikował się przy pomocy biblioteki ZeroMQ, a samą wartość sumy wyznaczyłem na {0x13, 0x37}. 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 niemalże za darmo, co uznaję za bardzo duży sukces z dziedziny projektowania architektury oprogramowania. Kolejną zaletą utworzenia symulatora urządzenia jest umożliwienie testów komponentowych realizując regułę czarnej skrzynki, co pozwoli w przyszłości znacznie skrócić czas testowania sterownika pod względem błędów logicznych.

9.1.1 Weryfikacja wiadomości pod względem obecności zarezerwowanych znaków

Z języka angielskiego „Byte stuffing”. Procedura polega na tym, że zarówno dla flagi startu jak i stopu zarezerwowana jest wartość 0x7E. Niestety urządzenie podrzędne może odpowiedzieć na przykład podczas procedur XID negocjacji, wiadomością zawierającą bajt 0x7E. Nieobsłużone takie zjawisko spowoduje, że urządzenie nadziedne zinterpretuje ten bajt jako bajt

stopu, co jest niepożądane. W tym celu standard AISG definiuje procedurę do jakiej należy się zastosować w takiej sytuacji, z którą nie zaznajomiłem się.

9.1.2 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 zaadresowanym. W przypadku kiedy urządzenie podrzędne nie otrzyma jakiekolwiek wiadomości w przeciągu 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 ponowić komunikację.

9.1.3 Ustalanie maski

RetSimulator jest urządzeniem które często posiada wbudowane wyjście RS-485, co oznacza że można podłączyć dwa bądź więcej urządzeń szeregowo. W takiej sytuacji na wiadomość pochodząą z urządzenia nadzorującego zaadresowaną wartością 0xFF, każde z nich wyśle swoją odpowiedź co niesie ze sobą wiele problemów. Po stronie urządzenia nadzorującego zaobserwujemy to w ten sposób, że nasłuchiwanie w czasie standardowego interwału czasu bajty pochodzące będą naprzemian od dwóch bądź większej liczby urządzeń. Algorytm definiowania maski skutecznie rozwiązuje ten problem. Podczas tej procedury dążymy do ustalenia unikalnego identyfikatora każdego z urządzeń, w celu wysłania wiadomości z zadaniem 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.

9.2 Rozbudowa aplikacji o nowe funkcjonalności

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 serwowanych przez protokół AISG 2.0 takich jak: aktualizacja oprogramowania, reset twardy czy miękki oraz usta-

9. Plan dalszego rozwoju

wienie kąta. Wdrożenie do aplikacji mechanizmu wielowątkowości, podłączenie więcej niż jednego urządzenia również będzie osiągalne. Warstwy dynamicznego budowania wiadomości pozwalają również zmniejszyć czas wdrożenia najnowszej wersji protokołu AISG w wersji 3.0.

10. Podsumowanie

Podsumowanko

Bibliografia

- [1] *Control interface for antenna line devices*, (2016)
<https://aisg.org.uk/files/AISG-v2.0.pdf>
- [2] *UTRAN Iuant interface: General aspects and principles*, (2012)
https://www.etsi.org/deliver/etsi_ts/125400_125499/125460/11.00.00_60/ts_125460v110000p.pdf
- [3] *UTRAN Iuant interface: Layer 1*, (2016)
https://www.etsi.org/deliver/etsi_ts/125400_125499/125461/13.00.00_60/ts_125461v130000p.pdf
- [4] *UTRAN Iuant interface: Signalling transport*, (2016)
https://www.etsi.org/deliver/etsi_ts/125400_125499/125462/06.05.01_60/ts_125462v060501p.pdf
- [5] *UTRAN Iuant interface: Remote Electrical Tilting (RET) antennas Application Part (RETAP) signalling (2017) 125 463 V7.5.0*, (2007)
https://www.etsi.org/deliver/etsi_ts/125400_125499/125463/07.05.00_60/ts_125463v070500p.pdf
- [6] *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*, (2002)
<https://www.iso.org/standard/37010.html>
- [7] Robert C. Martin
Zwinne wytwarzanie oprogramowania. Najlepsze zasady, wzorce i praktyki, (2014)
<https://helion.pl/ksiazki/zwinne-wytwarzanie-oprogramowania-najlepsze-zasady-zwiwyv.htm#format/d>

Bibliografia

Beamwidth. <http://www.itcmp.pwr.wroc.pl/~jwach/Techniczny%20EN-PL.htm>.

27.01.2020.

Elektryczny tilt. <https://www.kathreinusa.com/support/ret-products/>. 27.01.2020.

High-Level Data Link Control. https://pl.wikipedia.org/wiki/High-Level_Data_Link_Control. 25.01.2020.

High-Level Data Link Control. https://en.wikipedia.org/wiki/High-Level_Data_Link_Control. 25.01.2020.

Martin, RC. *Zwinne wytwarzanie oprogramowania*. 2015.

Radio Mobile. https://www.ve2dbe.com/rmonline_s.asp. 27.01.2020.

Spis rysunków

2.1	RetSimulator- Miejsce na antenę	8
2.2	RetSimulator podłączonym kablem RS-485 oraz włożoną atrapą anteny	8
2.3	Rysunek przedstawiający kąt modyfikowany przy pomocy RET-a	9
2.4	Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 0 stopni	10
2.5	Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 40 stopni	10
3.1	Ramka informacyjna - ewaluacja bajtu kontrolnego	15
7.1	Ustanowienie prędkości połączenia wraz z początkowym skanowaniem urządzeń	32
7.2	Żadanie adresacji oraz dodatkowe skanowanie urządzeń	34
7.3	Negocjacja rozmiaru okna oraz payloadu ramki informacyjnej oraz ustanowienie normalnego trybu komunikacji	36
7.4	Negocjacja pozostałych parametrów HDLC oraz żadanie kalibracji urządzenia .	37
8.1	Zrzut ekranu z wykonania testów modułowych	39
8.2	Zrzut ekranu z wykonania testów jednostkowych	39
8.3	Zrzut ekranu z uruchomienia programu z odfiltrowaniem logów poniżej priorytetu info	40
8.4	Zrzut ekranu z uruchomienia programu bez odfiltrowania logów	40

Spis listingów

5.1	Interfejs komendy	19
5.2	Definicja klasy konkretnej komendy używającej fabryki oraz strategii	20
5.3	Definicja klasy dla obiektu pustego	21
5.4	Przykład użycia obiektu pustego	21
5.5	Plik nagłówkowy dla metody szablonowej walidacji komendy	22
5.6	Metoda szablonowa - Wywołanie metod wirtualnych z poziomu innej metody .	22
5.7	Strategia komunikacji request-response dla urządzenia nadzawanego	23
5.8	Konstruktor zrealizowany podejściem wstrzykiwania zależności	24
5.9	Budowniczy wraz z Fluent API podczas budowania ramki I - Kalibruj	25
5.10	Fabryka budowniczych dla sterownika urządzenia nadzawanego	25

Zawartość płyty DVD

1. praca.pdf;
2. Plik z kodem źródłowym: xxx.js;
3. Plik z kodem źródłowym: yyy.css;
4. Plik z kodem źródłowym: zzz.html;

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