Pomiary Automatyka Robotyka, ISSN 1427-9126, R. 29, Nr 1/2025, 99–106, DOI: 10.14313/PAR_255/99

Architektura oprogramowania systemu kontrolno-pomiarowego z interfejsem USB

Rafał Wojszczyk
Politechnika Koszalińska, ul. Śniadeckich 2, 75-453 Koszalin

Streszczenie: W pracy zaproponowano implementację wysokopoziomowej architektury oprogramowania, która pozwala na komunikację z urządzeniem zewnętrznym w sposób obiektowy. Opracowane rozwiązanie wraz z autorskim urządzeniem podłączanym do portu USB stanowią gotowy system kontrolno-pomiarowy. Wykorzystanie wielowarstwowej architektury oraz wybranych wzorców projektowych pozwoliło na opracowanie rozwiązania programowego, które zapewnia niski koszt rozbudowy, pozwala na łatwe wykorzystanie w sposób obiektowy oraz zapewnia możliwość wymiany modułów. Dodatkowo uniwersalność opracowanego narzędzia pozwala na wprowadzanie wybranych modyfikacji systemu bez konieczności ponownej kompilacji kodu źródłowego.

Słowa kluczowe: USB, .NET, MVP, architektura oprogramowania, wzorce projektowe, LibUSB, AVR, ATmega

1. Wpowadzenie

Komunikacja między komputerem a urządzeniem zewnętrznym jest możliwa na wiele sposobów. Większość urządzeń konsumenckich jest podłączanych za pomocą portu USB (ang. *Universal Serial Bus*), który jest stale rozwijany od 1996 r. Wydawać by się mogło, że bardzo duża popularność tego portu przyczyni się do używania również przez hobbystów i naukowców w budowie własnych urządzeń [6]. Praktyka pokazuje jednak, że dużo częściej w amatorskich oraz naukowych projektach stosowany jest port RS-232/UART, który już na etapie pierwotnych założeń oferował gorsze parametry niż USB. Wynika to z faktu, że środowisko hobbystów często nie ma odpowiedniej wiedzy, umiejętności lub zasobów finansowych, aby realizować komunikację przez interfejs USB.

Celem niniejszego rozdziału jest wprowadzenie wysokopoziomowej architektury oprogramowania, która dzięki dekompozycji na wiele warstw i użycie wzorców projektowych, umożliwia stosowanie urządzeń zewnętrznych podłączanych do portu USB. Proponowane rozwiązanie współdzieli wybrane korzyści oferowane przez port RS-232 (tj. łatwość i koszt implementacji) i jednocześnie umożliwia wykorzystanie możliwości oferowanych przez USB (np. komunikacja współbieżna, możliwość enumeracji wielu urządzeń, automatyczne ustawienie parametrów połączenia). Proponowaną architekturę oprogramowania zaimplementowano w programie użytkowym, co w połączeniu z dedykowanym urządzeniem zewnętrznym stanowi gotowy układ kontrolno-pomiarowy. W implementacji wykorzystano łatwo dostępne komponenty elektroniczne (mikrokontroler z rodziny AVR) oraz darmowe i powszechnie dostępne narzędzia programistyczne (Visual Studio, .NET Framework, WinForms, SQL Server). Opracowany system kontrolno-pomiarowy pozwala na rejestrowanie wyników pomiarów wejściowych sygnałów analogowych, odczyt cyfrowych stanów logicznych, sterowanie sygnałami cyfrowymi oraz generowanie sygnału PWM. Funkcje realizowane przez system wykonywane są zgodnie z zaplanowanym harmonogramem, natomiast planowanie harmonogramu odbywa się przez graficzny interfejs użytkownika, co nie wymaga umiejętności programowania. Oferowane możliwości są uniwersalne, co pozwoli na użycie tego rozwiązania przez techników w przemyśle, informatyków, pasjonatów DIY (ang. *Do It Yourself*) oraz na zastosowanie w badaniach naukowych.

Artykuł zawiera cztery rozdziały. W drugim rozdziale artykułu przedstawiono wymagania jakich oczekuje się od proponowanego systemu. Trzeci rozdział przedstawia opis implementacji. Piąty rozdział stanowi podsumowanie artykułu.

Autor korespondujący:
Rafał Wojszczyk, rafal.wojszczyk@tu.koszalin.pl

Artykuł recenzowany
nadesłany 10.03.2024 r., przyjęty do druku 20.12.2024 r.

2. Wymagania

W celu zebrania wymagań przeprowadzono analizę literatury, gdzie wykorzystano autorskie rozwiązania kontrolno-pomiarowe pełniące rolę narzędzi wspomagających badania naukowe oraz rolę głównej komunikacji w projektach hobbystycznych. Informacje otrzymane z przeprowadzonej analizy można zgrupować do omówionych dalej spostrzeżeń.

Nader często stosowana jest komunikacja przez port RS-232 (w tym emulacja USB za pomocą FT232R, CH340, CP2102) [4, 10, 11, 14]. Użycie RS-232 prowadzi do wielu trudności wynikających z założeń projektowych [15]:

— do jednego portu może być podłączone tylko jedno urządzenie, co dodatkowo wymaga wiedzy o parametrach połączenia,

– aplikacja obsługująca takie urządzenie musi cyklicznie odczytywać dane z portu RS-232, aby otrzymać powiadomienie (brak asynchroniczności),
– brak możliwości obsługi wielowątkowej, ponieważ urządzenie podłączone do portu RS-232 jest zasobem niepodzielnym, naruszenie tej zasady doprowadzi do występowania błędów klasy zakleszczenia oraz naruszenia niepodzielności [2].

Wśród stosowanych rozwiązań dominuje implementacja metodą strukturalną (nawet w językach obiektowych) [9, 7]. Potocznie nazywane jest to płaską implementacją, gdzie nie są stosowane filary paradygmatu programowania obiektowego. Ocena jakości takiego kodu źródłowego za pomocą popularnych metryk oprogramowania obiektowego (np. Chidamber-Kemerer, MOOD, metryki R.C. Martina [12]) przypuszczalnie będzie bardzo niska, a to oznacza wysokie koszty rozbudowy i naprawy błędów. Ponadto mała liczba instancji wzorców projektowych powoduje, że kod źródłowy będzie trudniejszy w zrozumieniu dla doświadczonych programistów, natomiast implementacja bez stosowania wzorców architektury znacznie utrudnia decentralizację aplikacji oraz wymianę komponentów.

Trzecie spostrzeżenie dotyczy dedykowania opracowanych rozwiązań wyłącznie do wybranych zastosowań (np. odczyt tylko wybranych wartości, komunikacji wyłącznie tekstowej) czy też wybranego standardu komunikacji (np. urządzenie w klasie Human Interface Device, co również może ograniczyć komunikację do trybu tekstowego) [3, 8]. W przypadku dedykowanych rozwiązań, ponowne wykorzystanie takiego narzędzia wymaga ponownej implementacji (zmiany kodu) i kompilacji programu, co nie jest trywialne.

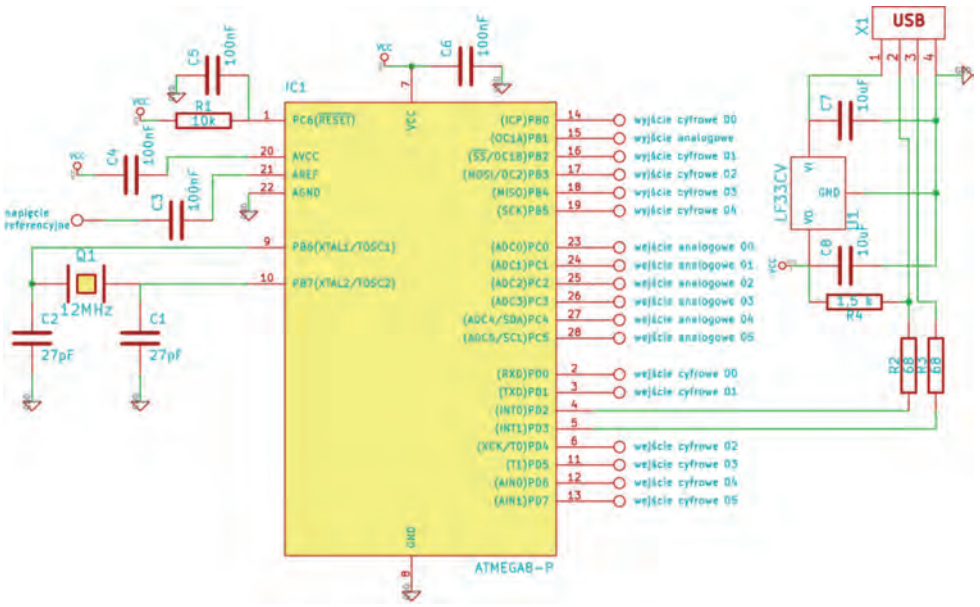
Na podstawie przedstawionych spostrzeżeń zaproponowano następujące wymagania, których spełnienie zapewni, że nowo-opracowane rozwiązanie spełni współczesne standardy techniczne i programistyczne:

1. Stosowanie interfejsu USB – koresponduje z wymienionymi spostrzeżeniami w stosunku do portu RS-232. USB u podstaw rozwiązuje większość problemów RS-232. Współczesny komputer typu laptop wyposażony jest w przynajmniej kilka takich portów, a liczbę portów można bez problemu zwiększyć. Każde urządzenie USB jest unikatowe dzięki numerom PID i VID, które są znane od razu po podłączeniu. W każdym urządzeniu może występować wiele punktów końcowych, z którymi wymiana danych może odbywać się asynchronicznie [16, 5]. Urządzenie USB jest gotowe do działania od razu po podłączeniu, nie wymaga podawania parametrów takich jak szybkość transmisji. Natomiast od strony elektrycznej linie zasilania oddzielone są od linii sygnałowych, a dodatkowo linie sygnałowe to sygnał symetryczny, co zwiększa odporność na zakłócenia.
2. Zastosowanie wielowarstwowej architektury oprogramowania oraz współczesnych standardów programistycznych koresponduje z wymienionymi spostrzeżeniami dotyczącymi sposobu implementacji programowej. Opracowanie odpowiedniej architektury oprogramowania ma na celu zapewnienia modularnej budowy aplikacji, co pozwoli na wymianę modułów (np. implementacja interfejsu użytkownika jako strony internetowej) oraz użycie modułów jako autonomicznych bibliotek. Wymagane jest stosowanie tzw. dobrych praktyk programistycznych (w tym wzorców projektowych oraz wzorców architektury), aby zapewnić niski koszt rozbudowy oraz naprawy błędów.
3. Uniwersalność aplikacji – koresponduje z wymienionymi spostrzeżeniami w zakresie dedykowania podobnych rozwiązań do wyłącznie jednego celu. Opracowane rozwiązanie powinno być konfigurowalne z poziomu interfejsu użytkownika, natomiast komunikacja z urządzeniem zewnętrznym powinna być realizowana z użyciem powszechnie dostępnych standardów USB, np. klasę urządzeń USB-CDC (ang. *Communications Device Class*).

3. Implementacja i testy

3.1. Implementacja sprzętowa

Testowe urządzenie zewnętrze zbudowano na bazie mikrokontrolera ATmega8, z wykorzystaniem biblioteki VUSB [17]. Biblioteka ta umożliwia emulację interfejsu USB w wersji 1.1 o szybkości FS [1]. Należy odróżnić realizację interfejsu USB w klasie CDC za pomocą emulacji przez mikrokontroler, od emulacji interfejsu RS-232 za pomocą gotowych układów scalonych (np. wspominany wcześniej FT232R), który jest podłączany do portu USB. W przypadku pierwszego komunikacja odbywa się zgodnie z protokołem USB (urządzenie będzie rozpoznawane przez system operacyjny jako USB), tj. przez odpowiednie punkty końcowe, w przypadku drugiego komunikacja odbywa się zgodnie z protokołem RS-232 (urządzenie będzie rozpoznawane przez system operacyjny jako port COM). Zaimplementowano klasę urządzenia USB-CDC, które przeznaczone jest do obsługi urządzeń zewnętrznych. Zdefiniowano również jeden punkt końcowy. Zgodnie z kodem deskryptora z listingu 1 został zdefiniowany zbiór rozkazów wraz z parametrami, które



Rys. 1. Schemat połączeń elektrycznych opracowanego urządzenia
Fig. 1. Wiring diagram of the developed device

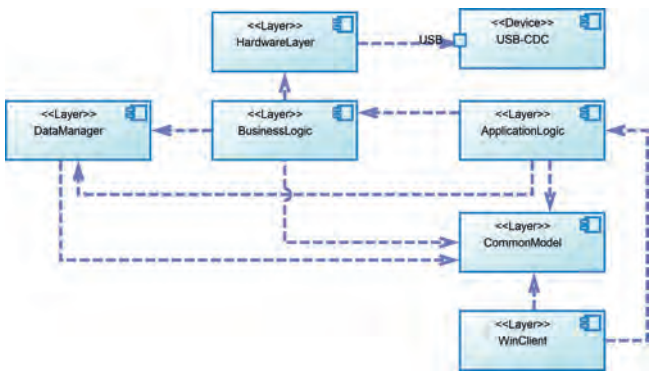
Tab. 1. Specyfikacja rozkazów
Tab. 1. Command specification

Rozkaz (pole *usbRequest*)	Argument (pole *wValue*)	Wyliczenie (pole *wIndex*)	Wartość zwracana	Opis
100	Bez znaczenia	Bez znaczenia	Dwubajtowa tablica o wartościach [100] [101]	Rozkaz testowy, sprawdzający jedynie czy urządzenie odbiera żądania i wysyła dane
102	Bez znaczenia	Młodszy bajt określa logiczny adres portu.	Brak	Rozkaz ustawia stan wysoki wybranego cyfrowego wyjścia
103	Bez znaczenia	Młodszy bajt określa logiczny adres portu.	Brak	Rozkaz ustawia stan niski wybranego cyfrowego wyjścia
104	Bez znaczenia	Młodszy bajt określa logiczny adres portu.	Dwubajtowa tablica, w której młodszy bajt ma wartość 0 lub 1, starszy zawsze 0	Rozkaz odczytuje i zwraca stan wybranego wejścia cyfrowego
105	Bez znaczenia	Młodszy bajt określa logiczny adres portu.	Dwubajtowa tablica, w której młodszy bajt zawiera osiem młodszych bitów z wyniku, a starszy bajt zawiera dwa starsze bity z wyniku	Rozkaz odczytuje i zwraca wartość napięcia z wybranego wejścia analogowego
106	Dwubajtowa tablica, w której młodszy bajt zawiera osiem młodszych bitów argumentu, a starszy bajt zawiera dwa starsze bity argumentu	Bez znaczenia	Brak	Rozkaz ustawia szerokość wypełnienia impulsu na wyjściu analogowym

umożliwiają obsługę portów wejścia/wyjścia mikrokontrolera, tab. 1 przedstawia specyfikację rozkazów. Zestawienie adresacji odpowiada wykonanym połączeniom elektrycznym, co przedstawia rys. 1. Użycie urządzenia polega na wysłaniu uzupełnionej struktury danych z listingu 1 danymi z tab. 1 oraz rys. 1, przykładowo: 102, 0, 01 – oznacza ustawienie stanu wysokiego na porcie PB2 mikrokontrolera.

```
typedef struct usbRequest{
uchar       bmRequestType;
uchar       bRequest;
usbWord_t   wValue;
usbWord_t   wIndex;
usbWord_t   wLength;};
```

Listing 1. Struktura deskryptora rozkazu zaimplementowanego w urządzeniu zewnętrznym
Listing 1. Structure of the command descriptor implemented in an external device



Rys. 2. Diagram komponentów w notacji UML reprezentujący zaimplementowane warstwy oraz relacje użycia pomiędzy warstwami
Fig. 2. Component diagram in UML notation representing the implemented layers and the use relationships between the layers

Alternatywna realizacja urządzenia może zostać wykonana za pomocą innych mikrokontrolerów, które wspierają komunikację za pomocą interfejsu USB, np. AT90USB82, ATmega16U4, STM32F103C8T6. W celu wykonania takiego urządzenia, należy zdefiniować w urządzeniu zerowy punkt końcowy i zaimplementować obsługę deskryptora z listingu 1 zgodnie z tabelą rozkazów z tab. 1. Zestawienie adresacji będzie unikatowe dla każdego z mikrokontrolerów.

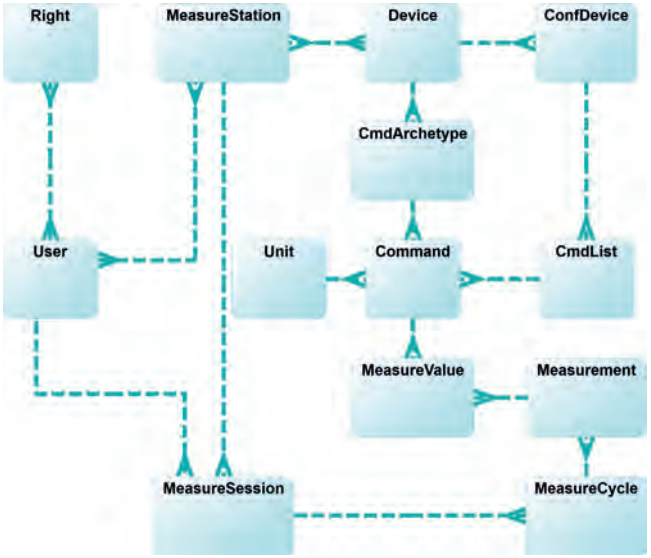
3.2. Implementacja programowa

Architektura oprogramowania

Implementacja oprogramowania obsługującego opracowane urządzenie zewnętrze, została wykonana w technologii Microsoft .NET Framework oraz z wykorzystaniem silnika baz danych Microsoft SQL Server. Implementacja została zdekomponowana na sześć osobnych warstw, które mogą być zastępowane innymi. Podział na warstwy oraz zależności między konkretnymi warstwami przedstawia rys. 2.

Warstwa o największej odpowiedzialności w hierarchii zależności to *CommonModel*. Najważniejszym celem tej warstwy jest dostarczanie wspólnego modelu danych dziedziny problemowej, który jest stosowany w pozostałych warstwach. Model danych został opracowany zgodnie z celem aplikacji, tj. sterowania i rejestrowania, zgodnie z harmonogramem. Na rys. 3 przedstawiono diagram związek-encja, który odwzorowuje model danych. W warstwie *CommonModel* model ten odwzorowany jest w strukturę obiektową. Dodatkowo w *CommonModel* znajdują się wspólne dane, wykorzystywane w całej aplikacji, jest to m.in. informacja o bieżąco zalogowanym użytkowniku. Dane tego typu udostępniane są za pomocą wzorca projektowego Singleton, co zwiększa spójność danych.

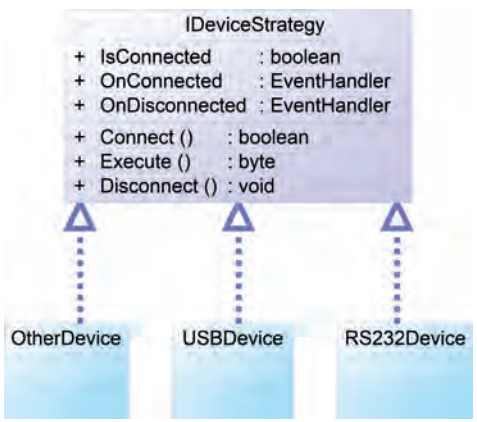
Dane z modelu dziedziny zapisywane i odczytywane są za pomocą warstwy *DataManager*. Warstwa odpowiada za komunikację z bazą danych, stąd w celu skorzystania z innego silnika baz danych, należy zamienić tą warstwę na inną, obsługującą inny silnik baz danych. Wymagane jest, aby dostęp do danych był realizowany z wykorzystaniem wzorców Repozytorium oraz Jednostka pracy (ang. *UnitOfWork*), wówczas połączenie z nowym silnikiem bazy danych będzie kompatybilne z istniejącym kodem.



Rys. 3. Diagram związek-encja w notacji Barkera, który odwzorowuje konceptualny model dziedziny problemowej
Fig. 3. Entity-relationship diagram in Barker notation that maps the conceptual model of the problem domain

Warstwa *Hardware* odpowiedzialna jest za komunikację z urządzeniem zewnętrznym, za pomocą biblioteki programistycznej LibUsbDotNet, do której została utworzona fasada (dzięki temu wzorcowi uproszczono użycie tej biblioteki). Obsługa opracowanego urządzenia z mikrokontrolerem Atmega8 jest zrealizowane w sposób generyczny. Odpowiada za to wzorzec Strategia, tj. zdefiniowany jest interfejs IDeviceStrategy, którego implementacja konkretyzowana jest na dane urządzenie. Dodanie nowego urządzenia, niezgodnego z deskryptorem z listingu 1 (w tym wykorzystującego inne porty komunikacyjne) polega na zaimplementowaniu tego interfejsu i odpowiedniego uzupełnienia kodu, przedstawia to diagram klas z rys. 4. Natomiast dodanie nowego urządzenia zgodnego z wspomnianym deskryptorem nie wymaga wprowadzania modyfikacji w kodzie programu, wystarczy uzupełnienie danych z poziomu interfejsu użytkownika. Opracowane w ten sposób rozwiązanie jest otwarte na rozbudowę i zamknięte na modyfikacje (jedna z zasad S.O.L.I.D). Przekazywanie parametrów poprzez metodę *Execute* realizowane jest przez implementację wzorca Kompozyt (przekazywany jest obiekt kompozytu, zamiast wielu drobnych parametrów).

Warstwą, która łączy wszystkie omówione dotąd warstwy, jest *BusinessLogic*, co odpowiada logice biznesowej. Jest to implementacja zbioru reguł sterujących całym systemem, inicjalizuje

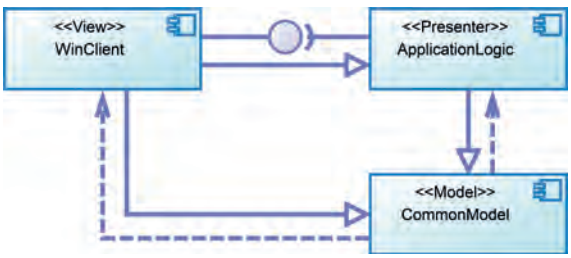


Rys. 4. Diagram klas przedstawiający koncepcję wykorzystania wzorca strategia do implementacji obsługi kolejnych urządzeń
Fig. 4. Class diagram showing the concept of using the strategy pattern to implement support for more devices

połączenie z warstwą *Hardware*, wykonuje schemat działania odczytany z bazy danych za pomocą *DataManager*. W tej warstwie zaimplementowano wzorzec stanu, który jest odpowiedzialny za bieżące monitorowanie stanu urządzenia. Natomiast powiadomienia o zmianach w urządzeniu USB (np. odłączenie, wystąpienie błędu, wystąpienie przerwania) realizowane jest za pomocą wzorca Obserwator, którzy przekazuje komunikaty za pomocą wzorca Łańcuch odpowiedzialności (ang. *Chain of Responsibility*). Połączenie trzech wzorców pozwoliło na realizację skutecznego mechanizmu nadzorującego prawidłową pracę urządzenia, reagowanie na zmiany w urządzeniu oraz uruchamianie odpowiednich działań w zależności od stanu.

Implementacja wzorca MVP

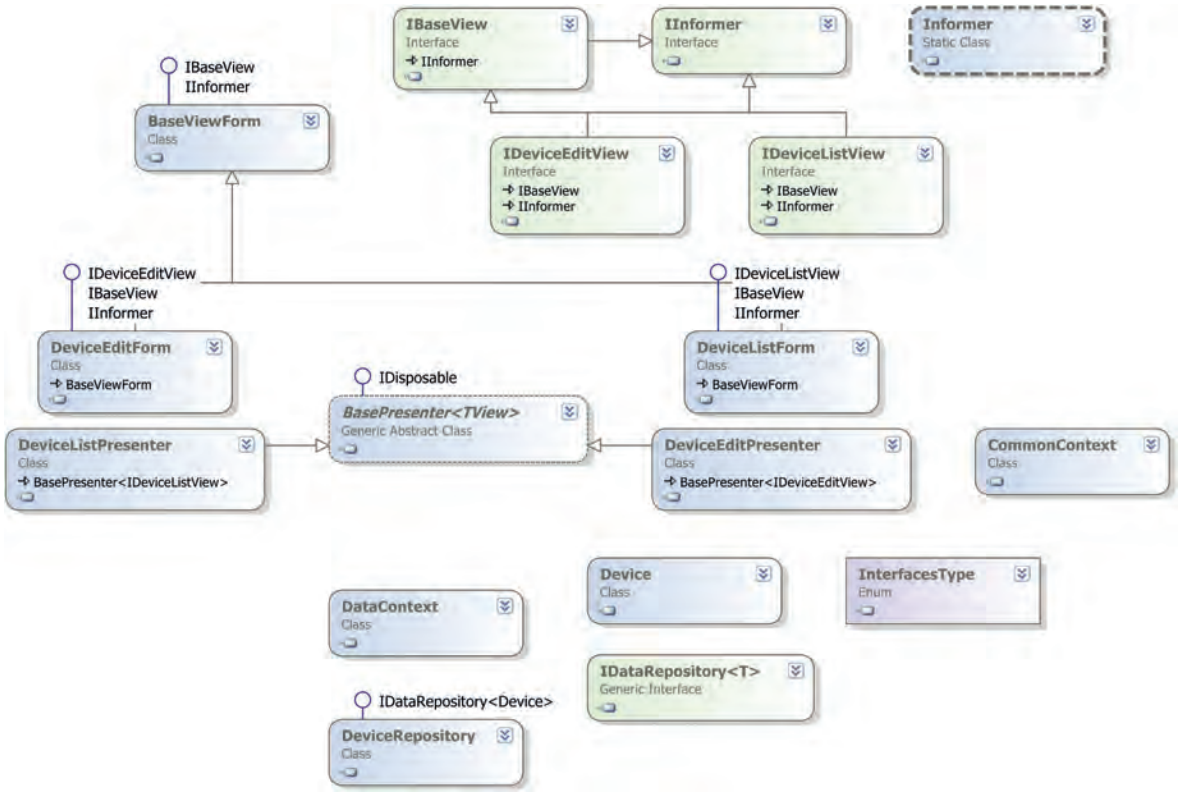
Ostatnie dwie warstwy odpowiedzialne są za implementację aplikacji w technologii WindowsForms, tj. klienta uruchamianego jako aplikacja stacjonarna. Aplikacja została zrealizowana zgodnie z wzorcem architektury oprogramowania MVP (ang. *Model-Vidok-Presenter*), w wariacji Supervising Controller. Strukturę wzorca przedstawia rys. 5. Model to warstwa *CommomModel*, widok to warstwa *WinClient*, natomiast prezenter to warstwa *ApplicationLogic*. Ze względu na wariant wzorca możliwe jest wykorzystanie struktury modelu przez widok, nie możliwe jest bezpośrednie manipulowanie danymi.



Rys. 5. Diagram komponentów przedstawiający koncepcję podziału warstw w wzorcu architektury MVP
Fig. 5. Component diagram showing the concept of layering in the MVP architecture pattern

Interfejs *IInformer* (rys. 6) należący do warstwy *ApplicationLogic* (oraz implementacja *Informer* należąca do warstwy *WinClient*) służą jedynie do pokazywania krótkich informacji dialogowych. Interfejs *IBaseView* należy do warstwy *ApplicationLogic* i zawiera prototypy metod odpowiedzialnych za wyświetlanie i ukrywanie widoków. Każdy interfejs widoku musi po nim dziedziczyć. Interfejsy *IDeviceListView* oraz *IDeviceEditView* opisują prototypy metod i właściwości, jakie musi implementować widok, czyli konkretne okno w aplikacji, należą do warstwy *ApplicationLogic*. Klasa abstrakcyjna *BasePresenter* jest typowana na interfejs *IBaseView* lub interfejs po nim dziedziczący. Typowany interfejs jest podstawowym widokiem dla danego prezentera, klasa należy do warstwy *ApplicationLogic*. Klasy *DeviceListPresenter* oraz *DeviceEditPresenter* dziedziczą po *BasePresenter* i są typowane na odpowiadający im interfejs widoku. Zawierają opis możliwych funkcji prezentera, należą do warstwy *ApplicationLogic*. Klasa *CommonContext* jest wspólnym kontekstem dla warstwy *ApplicationLogic*, przechowuje informacje, które prezentery muszą między sobą wymieniać i nie mogą tego zrobić bezpośrednio.

Klasa *Device* to zmapowana tabela bazy danych, przedstawia pojedynczy rekord. Lista obiektów jest przekazywana do interfejsu widoku, w tym przypadku *IDeviceListView*. Pojedynczy obiekt *Device* jest przekazywany do interfejsu widoku pozwalającego na edycję, w tym przypadku *IDeviceEditView*. Dodatkowo z tą klasą powiązany jest typ wyliczeniowy *InterfacesType* wskazujący na typ interfejsu, przez jaki jest podłączone urządzenie. Klasa *DataContext*



Rys. 6. Diagram klas przedstawiający aplikację WindowsForms zaimplementowaną w architekturze MVP oraz wybrane klasy z pozostałych warstw. Diagram wygenerowany na podstawie istniejącego kodu

Fig. 6. Class diagram showing a WindowsForms application implemented in MVP architecture and selected classes from other layers. Diagram generated from existing code

należy do warstwy *DataManager* i odpowiada za wykonywanie operacji związanych z bazą danych. Interfejs *IDataRepository* jest typowany na obiekt modelu danych, zawiera prototypy podstawowych metod związanych z odczytem, zapisem, edycją oraz usuwaniem danych z bazy danych. Interfejs należy do warstwy *DataManager*. Klasa *DeviceRepository* implementuje interfejs *IDataRepository* z typowaniem na *Device* i odpowiada za odczytanie, zapisanie, edycję i usuwanie obiektów typu *Device*. Klasa należy do warstwy dostępu do danych. Każda operacja wpływająca na dane wykonywana jest przy użyciu klasy *DataContext*. Klasa *BaseViewForm* implementuje interfejs *IBaseView* oraz w swojej implementacji interfejsu *IInformer* opisuje jego zachowania, aby klasy dziedziczące po *BaseViewForm* nie musiały tego robić ponownie. Klasa należy do warstwy widoku, dziedziczy po systemowej klasie *Form*. Klasy *DeviceListForm* oraz *DeviceEditForm* dziedziczą po *BaseViewForm* oraz implementują odpowiadający im interfejs widoku, odpowiednio *IDeviceListView* oraz *IDeviceEditView*. Każda klasa typu „Form" zawiera obiekt odpowiadającego jej prezentera i przy tworzeniu tego obiektu przekazuje do niego referencję do siebie samej. Prezenter dzięki tej referencji może wykonywać metody i korzystać z właściwości opisanych przez interfejs, który implementuje klasa typu „Form".

Możliwości rozwoju i rozbudowy

Implementacja wielowarstwowej architektury oraz wybranych wzorców projektowych dostarcza wiele korzyści, dostrzegalnych przede wszystkim z punktu widzenia programistów. Implementacja innego rodzaju widoku, np. aplikacji internetowej, wymaga zaimplementowania nowych warstw widoku i kontrolera (zamiast prezentera, co jest zgodne z wzorcem architektury oprogramowania MVC – ang. *Model-View-Controller*). Nowe warstwy będą korzystały z pozostałych, opisanych wcześniej warstw. Podobnie możliwe jest wykorzystanie wyłącznie wybranych warstw (*HardwareLayer, BusinessLayer)* w celu implementacji np. aplikacji konsolowych. Możliwe jest również rozproszenie systemu i uruchomienie wymienionych warstw jako usługi internetowe lub innych rozwiązań umożliwiających dostęp zdalny.

Inne korzyści wynikające ze stosowanej architektury to możliwość wymiany wybranych komponentów (warstw), a w związku z współdzieleniem znacznej części kodu, oznacza to niższe koszty budowy nowych aplikacji, bazujących na opracowanych komponentach.

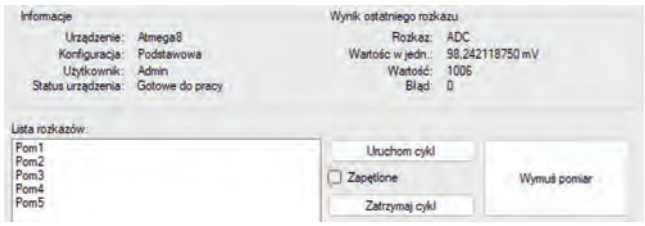
3.3. Użytkowanie systemu

Do prawidłowego funkcjonowania systemu wymagane jest podłączenie urządzenia do portu USB oraz zainstalowanie w systemie Windows 5.1 lub nowszym, generycznego sterownika LibUSB lub LibUsbDotNet.

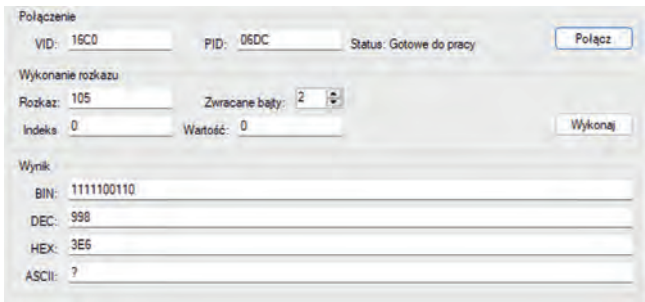
Użytkowanie aplikacji możliwe jest w kilku trybach: konfiguracyjnym, administracyjnym, wykonywania rozkazów, testowym. W trybie konfiguracyjnym możliwe jest zdefiniowanie rozbudowanej struktury użytkowników, komputerów (stacji pomiarowych) oraz urządzeń. Dany użytkownik może mieć dozwolony lub zabroniony dostęp do danego komputera i urządzenia, podobnie na danym komputerze może być dopuszczone lub zabronione dane urządzenie. Ustawieniem globalnym są jednostki miary, tj. przypisania odpowiednim wartością mierzonym wartości liczbowych lub słownych. Harmonogram wykonywania rozkazów sterujących i odczytujących to zagnieżdżona struktura, która na górze hierarchii rozpoczyna się od konfiguracji, w której zagregowane są listy rozkazów. Lista rozkazów zawiera zbiór rozkazów, które są egzemplarzami z pierwowzoru rozkazu (predefiniowanego słownika, zawierającego odpowiednio dobrane parametry do ustawiania lub odczytu wybranej własności). Dodanie nowych rozkazów wymaga modyfikacji programu danego urządzenia, natomiast nie wymaga modyfikacji kodu tego systemu, wystarczy aktualizacja danych słownikowych. Harmonogram wykonywany jest zgodnie z listą rozkazów.

Dostępność rozkazów warunkowana jest przypisaniem urządzenia do danej konfiguracji.

W trybie wykonywania rozkazów, użytkownik wybiera odpowiednie urządzenie i konfigurację, ma możliwość wprowadzenia opisu do całego działania, jak również do każdego rozkazu z osobna. Po uruchomieniu harmonogramu system działa autonomicznie, a użytkownik ma możliwość przerwania i wznowienia harmonogramu. Gdy harmonogram nie jest uruchomiony, użytkownik może wymusić wykonanie pojedynczego rozkazu. Rysunek 7 przedstawia fragment programu w trybie wykonywania rozkazów. W przypadku przerwania harmonogramu, np. przez odłączenie urządzenia, harmonogram jest zatrzymywany i wznawiany po ponownym podłączeniu. W przypadku nieprawidłowego zamknięcia aplikacji (np. w trakcie resetu komputera), zapamiętywany jest ostatni stan, a użytkownik otrzymuje odpowiedni komunikat.



Rys. 7. Widok program w trybie wykonywania rozkazów
Fig. 7. View of the programme in command execution mode



Rys. 8. Widok program w trybie testowym
Fig. 8. View of the programme in test mode

Warty omówienia jest tryb testowy, w którym można ręcznie (tj. poprzez wpisanie odpowiednich wartości w polach tekstowych) nawiązać komunikację z urządzeniem zewnętrznym i wykonać rozkaz zgodnie z tab. 1. Wynik rozkazu (jeśli występuje) prezentowany jest w zapisie binarnym, dziesiętnym, szesnastkowym oraz ASCII. Widok trybu testowego przedstawia rys. 8.

3.4. Weryfikacja systemu

Proponowane rozwiązanie zostało uruchomione i użytkowane przez studentów Koła Naukowego AnyCode Politechniki Koszalińskiej (dawniej Grupa .NET) oraz wybranych pracowników Wydziału Elektroniki i Informatyki Politechniki Koszalińskiej. Wykorzystanie całego systemu polegało na sterowaniu diodami LED zgodnie z zaplanowanym harmonogramem oraz sterowanie i odczyt wartości w przygotowywanym torze testowym dla dronów. Poza tym, została zaimplementowana aplikacja konsolowa, która wykorzystywała bezpośrednio warstwę *Hardware* w celu prototypowania automatyki w projektach realizowanych przez Koło Naukowe. W opinii członków Koła Naukowego AnyCode, opracowane rozwiązanie jest szczególnie użyteczne w budowie prototypowych urządzeń, ponieważ pod względem kompilacji i debugowania przewyższa np. Arduino. W proponowanym rozwiązaniu nie ma potrzeby wgrywania kodu maszynowego do mikrokontrolera po kompilacji, ponieważ jednostką sterującą jest komputer PC. W związku z tym debugowanie odbywa się również na poziomie komputera PC. Wykorzystane urządzenia USB bazowały m.in. na schemacie PCB opublikowanym przez AVT-Korporacja Sp. z o.o. oraz na układach budowanych z wykorzystaniem uniwersalnych płytek PCB. Zastosowane urządzenia przedstawia rys. 9.

Rys. 9. Wykonanie urządzenia USB z użyciem schematu płytki PCB dostarczonej przez AVT-Korporacja Sp. z o.o. Do urządzenia podłączona jest płytka testowa
Fig. 9. Realisation of the USB device using the PCB schematic provided by AVT Corporation. A test board is connected to the device

Przeprowadzono również analizę porównawczą opracowanego rozwiązania względem analogicznego urządzenia podłączanego do portu RS-232. Urządzenie z interfejsem RS-232 zostało zrealizowane również z wykorzystaniem mikrokontrolera ATmega8, gdzie wykorzystano interfejs UART oraz konwerter napięć MAX232. Dodanie urządzenia do systemu zostało zrealizowane zgodnie z wzorcem Strategia. W przypadku USB dane wysyłane są w pakietach danych, które składają się z rozkazu i dwóch szesnastobitowych parametrów. W komunikacji RS-232 nie występuje podział na pakiety danych [18]. Przesyłane są kolejne bajty danych. Stąd konieczne było utworzenie mechanizmu, który zinterpretuje odbierane dane w podobny sposób jak przy interfejsie USB. Ponadto w przypadku USB zwracany wynik jest częścią jednej transakcji i programista ma 100 % pewności, że otrzymane dane są odpowiedzią na wysłane żądanie. W przypadku RS-232 programista jest zmuszony do odczytywania danych z portu. Dodatkowo nie można tego wykonać zaraz po wysłaniu rozkazu, należy jeszcze odczekać pewien czas na gotowość portu do transmisji. W tym przypadku bezpieczny czas oczekiwania to 20 milisekund. Kolejną różnicą zauważoną w trakcie implementacji jest brak możliwości sprawdzenia czy urządzenie jest podłączone. Jest możliwość jedynie sprawdzenia logicznego stanu, czy połączenie z wybranym portem zostało otwarte. Jednak w przypadku odłączenia urządzenia nie zostanie to wykryte i zasygnalizowane. Jedyny skutek to brak zwracanych danych z rozkazów, które powinny zwracać wyniki pomiarów. Utrudniło to w znaczący sposób monitorowanie stanu urządzenia, co ma związek z przerywaniem i wznawianiem pracy harmonogramu.

Wykonano również testy weryfikujące wydajność i stabilność w działaniu systemu, wykorzystującego zarówno urządzenie z interfejsem USB oraz z RS-232. Test wydajności polegał na jak najczęstszym wykonywaniu rozkazów. Zostały wykonane dwa rodzaje testów: pierwszy rodzaj to ustawianie wartości logicznych przez jedną minutę, test powtórzony 10 razy, drugi rodzaj to odczyt wartości analogowej przez jedną minutę, test powtórzony 10 razy. Podział na dwa rodzaje był konieczny ze względu na zastosowany dodatkowy czas opóźnienia, przy odczytywaniu wartości z portu RS-232, co z góry stawia port RS-232 na gorszej pozycji. Wyniki zostały zweryfikowane na podstawie czasu zapisanego w bazie danych systemu, po wykonaniu każdego z rozkazów. Czas został zapisany jako typ

Tabela 2. Wyniki weryfikacji wydajności
Tab. 2. Performance verification results

Heading level	Minimalna szybkość [rozkaz/sekundę]	Średnia szybkość [rozkaz/sekundę]	Maksymalna szybkość [rozkaz/sekundę]
Ustawianie USB	100	110	114
Ustawianie RS-232	59	100	106
Odczyt USB	82	91	103
Odczyt RS-232	34	36	38

danych DateTime z dokładnością do 0,00333 sekund. Czas ten uwzględnia wykonanie kodu wszystkich warstw architektury programu. Wyniki testu dla obu rodzajów z podziałem na interfejsy przedstawia tab. 2.

Wyniki eksperymentu potwierdziły wstępne założenie, że urządzenie z RS-232 będzie mniej wydaje przy odczycie danych. Wyniki testu, w których ustawiana była wartość są zbliżone, co wynika z wykorzystania tej samej jednostki obliczeniowej obu urządzeń, w obu przypadkach jest to mikrokontroler ATmega8. Jednakże wyniki eksperymentu wskazują większą wydajność w przypadku interfejsu USB. Warto dodać, że mikrokontroler wyposażony jest w sprzętowy port UART, natomiast nie jest wyposażony w sprzętowy interfejs USB (co oznacza, że w przypadku USB wymagana jest dodatkowa moc obliczeniowa).

Realizację weryfikacji stabilności działania, poprzedzono kalibracją szybkości obu urządzeń, tj. dodano dodatkowe opóźnienie w przypadku interfejsu USB, aby uzyskać taką samą liczbę rozkazów wykonywanych w trakcie jednej godziny, ile wykonuje urządzenie z RS-232. Eksperyment polegał na wykonywaniu listy rozkazów, które były zapętlone w nieskończoność. Każdy z testów trwał równo 24 godziny. W przypadku urządzenia z USB wykonano 147 580 rozkazów. W przypadku RS-232 wykonano 146 282 rozkazów, z czego 2,5 % (3657 rozkazów) zostało zaklasyfikowanych jako błąd przez zaimplementowany mechanizm kontroli błędów. Uzyskany błąd wskazuje jednoznacznie niższą stabilność komunikacji za pomocą RS-232, dodatkowo o 1298 mniejsza liczba wykonanych rozkazów wskazuje, że wystąpiły kolejne problemy w tym sposobie komunikacji.

4. Podsumowanie

W pracy przedstawiono propozycję architektury oprogramowania, którą zastosowano do budowy systemu kontrolno-pomiarowego wykorzystującego interfejs USB. Opracowane rozwiązanie stanowi gotowe narzędzie do sterowania stanami logicznymi cyfrowych wyjść urządzenia zewnętrznego, sterowanie wyjściem analogowym, odczyt wejść cyfrowych i odczyt wejść analogowych w urządzeniu zewnętrznym, zgodnie z zaplanowanym harmonogramem.

Opracowane rozwiązania spełnia wszystkie wymagania postawione w drugim rozdziale niniejszej pracy. Oznacza to, że wady wykorzystania portu RS-232 zostały poprawione poprzez wykorzystanie portu USB. Dodatkowo opracowana architektura oprogramowania zapewnia niskie koszty rozbudowy, ponownego użycia kodu i łatwego wykorzystania przez programistów języków wysokopoziomowych. Uniwersalność rozwiązania pozwala na wprowadzanie modyfikacji w działaniu urządzenia, dodawanie nowych urządzeń, bez potrzeby modyfikacji kodu źródłowego.

Opracowane rozwiązanie poddano weryfikacji poprzez wykorzystanie w autorskich projektach studentów z Koła Naukowego oraz przez bezpośrednie porównanie z portem RS-232. Przeprowadzone badania potwierdziły, że interfejs USB gwarantuje nie gorsze parametry niż RS-232. Porównywalna wydajność komunikacji wynika z wykorzystania takiej samej jednostki sterującej w każdym z urządzeń, tj. mikrokontrolera ATmega8. Różnice wystąpiły natomiast w stabilności działania, co jak potwierdziły eksperymenty, jest korzystniejsze w przypadku USB.

Dalsze prace będą skupiać się na wykorzystaniu zaawansowanych mikrokontrolerów, które wyposażone są w sprzętowy interfejs USB (np. ESP32-S3, STM32F411CEU6) oraz inne interfejsy, aby rozbudować system o obsługę dodatkowych urządzeń. Równolegle mogą być prowadzone prace implementacyjne zwiększające użyteczność oprogramowania, np. umożliwiające dynamiczne sterowanie urządzeniem zewnętrznym. Wersja rozwojowa oprogramowania jest dostępna w repozytorium: https://github.com/anycode-pk/usb_cdc.

Bibliografia

1. Daniluk A., *USB Praktyczne programowanie z Windows API w C++*, Helion 2009.
2. Giebas D., Wojszczyk R., *Atomicity Violation in Multithreaded Applications and Its Detection in Static Code Analysis Process*, "Applied Sciences", Vol. 10, No. 22, 2020, DOI: 10.3390/app10228005.
3. Kavaliauskas Ž., Šajev I., Blažiūnas G., Gecevičius G., Čapas V., Adomaitis D., *Electronic System for the Remote Monitoring of Solar Power Plant Parameters and Environmental Conditions*, "Electronics", Vol. 11, No. 9, 2022, DOI: 10.3390/electronics11091431.
4. Kowol A., *Internet Rzeczy – przykład implementacji protokołu LoRaWAN*, „Pomiary Automatyka Robotyka", R. 23, Nr 2, 2019, 61–68, DOI: 10.14313/PAR_232/61.
5. Mielczarek W., *USB uniwersalny interfejs szeregowy*, Helion 2005.
6. Nasution A.S., Adhitama B.S., Rasyidi F.H., Adi Aufarachman Putra Bambang Dwi, Imdad M.T., Jatmiko N.W., *Development of the NOAA-19 Satellite Data Receiver Ingest System based on FPGA*, 2022 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), Bandung, Indonesia, 2022, 132–137, DOI: 10.1109/ICRAMET56917.2022.9991236.
7. Nithya M.R., Lakshmi P., Roshmi J., Sabana R., Swetha R.U., *Machine Learning and IoT based Seed Suggestion: To Increase Agriculture Harvesting and Development*, 2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS), Erode, India, 2023, DOI: 10.1109/ICSCDS56580.2023.10104981.
8. Ladino K.S., Sama M.P., Stanton V.L., *Development and Calibration of Pressure-Temperature-Humidity (PTH) Probes for Distributed Atmospheric Monitoring Using Unmanned Aircraft Systems*, "Sensors", Vol. 22, No. 9, 2022, DOI: 10.3390/s22093261.
9. Tambaro M., Bisio M., Maschietto M., Leparulo A., Vassanelli S., *FPGA Design Integration of a 32-Microelectrodes*

Low-Latency Spike Detector in a Commercial System for Intracortical Recordings, "Digital", Vol. 1, No. 1, 2021, 34–53, DOI: 10.3390/digital1010003.
10. Tiboni M., Remino C., *Condition Monitoring of Pneumatic Drive Systems Based on the AI Method Feed-Forward Backpropagation Neural Network*, "Sensors", Vol. 24, No. 6, 2024, DOI: 10.3390/s24061783.
11. Thalman R., *Development and Testing of a Rocket-Based Sensor for Atmospheric Sensing Using an Unmanned Aerial System*. "Sensors", Vol. 24, No. 6, 2024, DOI: 10.3390/s24061768.
12. Wojszczyk R., *Zestawienie metryk oprogramowania obiektowego opartych na statycznej analizie kodu źródłowego*, „Zarządzanie projektami i modelowanie procesów", Zeszyty Rady Naukowej Polskiego Towarzystwa Informatycznego, 2013, 95–107, Warszawa 2013.
13. Wojszczyk R., Giebas D., *Multithreading Errors in Data Reading Automation*, [In:] Szewczyk, R., Zieliński C., Kaliczyńska M. (eds) Automation 2022: New Solutions and Technologies for Automation, Robotics and Measurement Techniques. AUTOMATION 2022. Advances in Intelligent Systems and Computing, Vol. 1427, DOI: 10.1007/978-3-031-03502-9_10.
14. Wójcicki S., Rutkowski T., *Projekt i budowa uniwersalnego sterownika programowalnego*, „Pomiary Automatyka Robotyka", Nr 2, 2013, 436–442.

Inne źródła

15. Gocławski P., *Komunikacja szeregowa*, artykuł online: https://www.elmark.com.pl/blog/komunikacja-szeregowa-1-jak-dzialaja-standardy-komunikacyjne-rs-232-422-485/.
16. Jamrógiewicz T., *Magistrala USB*, Instytut Radioelektroniki Politechniki Warszawskiej
17. *V-USB*, www.obdev.at/vusb.
18. Szymczyk N., *Jak działa interfejs RS232*, https://ntronic.pl/rs232/.

Software Architecture of the Control and Measurement System Using the USB Interface

Abstract: The paper proposes the implementation of a high-level software architecture that allows communication with an external device in an object-oriented manner. The developed solution, together with a proprietary device connected to a USB port, constitutes a ready-made control and measurement system. The use of a multilayered architecture and selected design patterns allowed for the development of a software solution that ensures low development costs, allows easy use in an object-oriented manner and ensures the possibility of module replacement. In addition, the versatility of the developed tool allows selected modifications to be made to the system without recompiling the source code.

Keywords: USB, .NET, MVP, software architecture, design patterns, LibUSB, AVR, ATmega

dr inż. Rafał Wojszczyk
rafal.wojszczyk@tu.koszalin.pl
ORCID: 0000-0003-4305-7253

Adiunkt na Wydziale Elektroniki i Informatyki Politechniki Koszalińskiej, gdzie uzyskał tytuł magistra inżyniera Informatyki, magistra Pedagogiki oraz stopień doktora w dyscyplinie Informatyka. Dydaktycznie i naukowo zajmuje się inżynierią oprogramowania, w szczególności statyczną analizą kodu źródłowego oraz tzw. dobrymi praktykami w programowaniu zorientowanym obiektowo. Jest autorem i współautorem ponad 40 publikacji, w tym artykułów w międzynarodowych czasopismach (JCR), rozdziałów w książkach i materiałów konferencyjnych. Jest recenzentem wielu międzynarodowych czasopism i konferencji.