Seminarium III - Tworzenie i oprogramowanie działania GUI:

Mirosław Franczak,

Informatyka przemysłowa

14.01.2026

Co znajduje się w moim projekcie (stan na teraz):

-Warstwa danych (symulacja):

- ARXModel
- PIDController
- SignalGenerator
- Simulation

- Warstwa usług (logika pośrednia):

- UARService

- Warstwa prezentacji (GUI):

- Mainwindow
- SecDialog

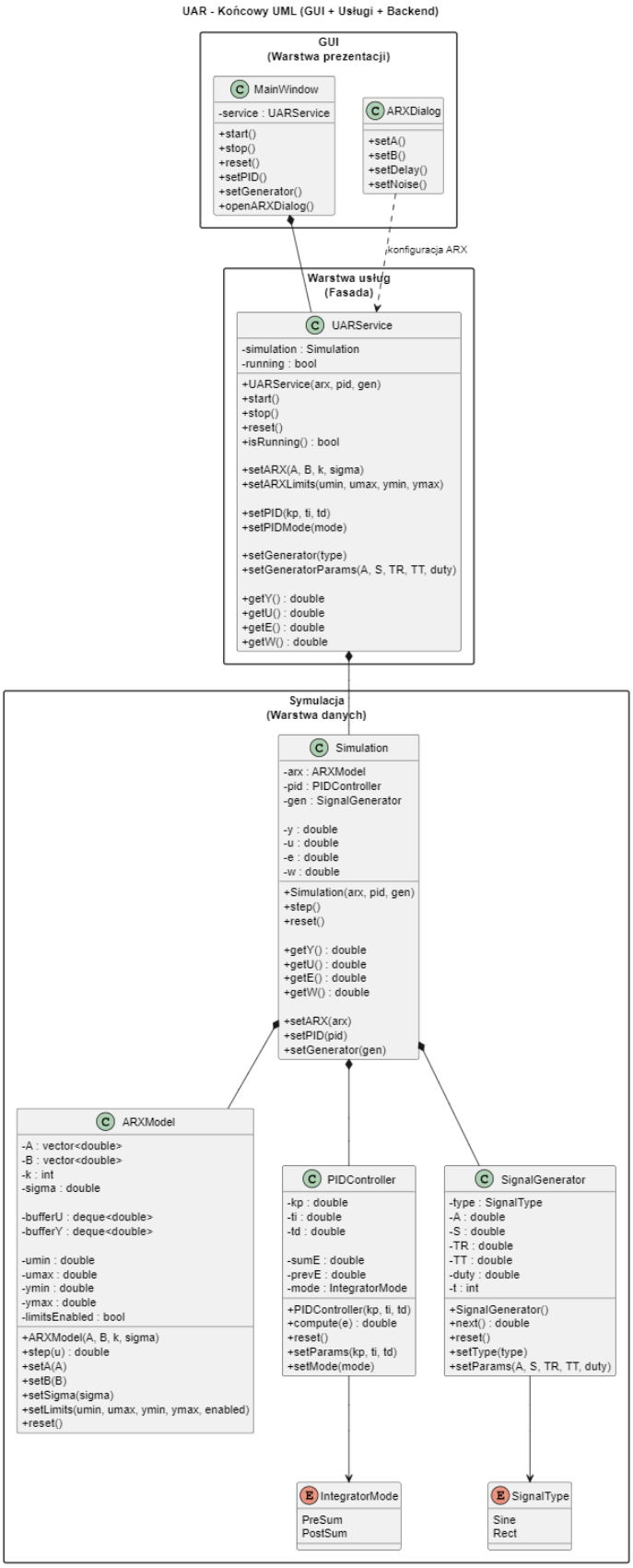
GUI → UARService → Simulation

Film z działania aplikacji:

Na etapie implementacji interfejsu graficznego zdarza się, że prezentowane są niepoprawne wartości, co wynika z niedostatecznego dopracowania warstwy prezentacji. Sam backend symulacji został jednak przetestowany niezależnie i działał poprawnie. Jest to kwestia dopracowania która będzie wykonana na następne zajęcia jednak samo gui w tym wypadku działa.

https://youtu.be/-KghER41YRU

UML


UAR - Końcowy UML (GUI + Usługi + Backend)
GUI
(Warstwa prezentacji)
MainWindow
-service : UARService
+start()
+stop()
+reset()
+setPID()
+setGenerator()
+openARXDialog()
ARXDialog
+setA()
+setB()
+setDelay()
+setNoise()
konfiguracja ARX
Warstwa usług
(Fasada)
UARService
-simulation : Simulation
-running : bool
+UARService(arx, pid, gen)
+start()
+stop()
+reset()
+isRunning() : bool
+setARX(A, B, k, sigma)
+setARXLimits(umin, umax, ymin, ymax)
+setPID(kp, ti, td)
+setPIDMode(mode)
+setGenerator(type)
+setGeneratorParams(A, S, TR, TT, duty)
+getY() : double
+getU() : double
+getE() : double
+getW() : double
Symulacja
(Warstwa danych)
Simulation
-arx : ARXModel
-pid : PIDController
-gen : SignalGenerator
-y : double
-u : double
-e : double
-w : double
+Simulation(arx, pid, gen)
+step()
+reset()
+getY() : double
+getU() : double
+getE() : double
+getW() : double
+setARX(arx)
+setPID(pid)
+setGenerator(gen)
ARXModel
-A : vector<double>
-B : vector<double>
-k : int
-sigma : double
-bufferU : deque<double>
-bufferY : deque<double>
-umin : double
-umax : double
-ymin : double
-ymax : double
-limitsEnabled : bool
+ARXModel(A, B, k, sigma)
+step(u) : double
+setA(A)
+setB(B)
+setSigma(sigma)
+setLimits(umin, umax, ymin, ymax, enabled)
+reset()
PIDController
-kp : double
-ti : double
-td : double
-sumE : double
-prevE : double
-mode : IntegratorMode
+PIDController(kp, ti, td)
+compute(e) : double
+reset()
+setParams(kp, ti, td)
+setMode(mode)
SignalGenerator
-type : SignalType
-A : double
-S : double
-TR : double
-TT : double
-duty : double
-t : int
+SignalGenerator()
+next() : double
+reset()
+setType(type)
+setParams(A, S, TR, TT, duty)
IntegratorMode
PreSum
PostSum
SignalType
Sine
Rect

Implementacja która sprawiła największą trudność:

Największe problemy w projekcie pojawiły się na etapie **implementacji wykresów w warstwie GUI**.
Trudność nie wynikała z logiki symulacji (backend działał poprawnie), lecz z **problemów narzędziowych i integracyjnych w Qt**.

Brak dostępności QtCharts w Qt Creator.

Finalnie rozwiązałem problem przez reinstalacje programu, dalej nie wiem dlaczego QtCharts nie działały.

Próby rozwiązania problemu:

Podjąłem próbę ręcznej instalacji QtCharts z repozytorium (qtcharts-dev) po której

pojawiały się błędy:

- No rule to make target
- cannot find ... src/lib
- brak rozpoznania formatu biblioteki

Efekt:
Ręczna instalacja QtCharts okazała się niepraktyczna w warunkach projektu.

Znalazłem alternatywę QCustomPlot.

Zastosowano bibliotekę **QCustomPlot** jako alternatywę:

- pliki qcustomplot.cpp i qcustomplot.h zostały dodane do projektu,
- jednak pojawiły się błędy linkera:

-multiple definition of QCPLayerable

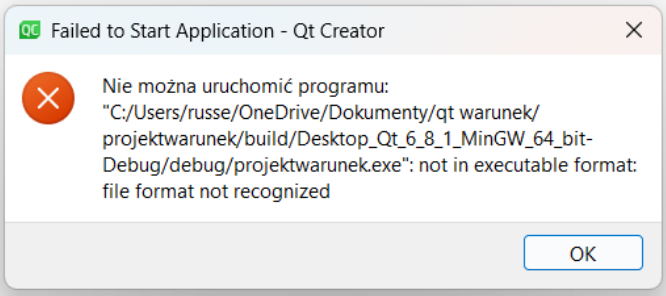
-multiple definition of QCPLayerable::~QCPLayerable

Przyczyna:

- wielokrotna kompilacja tego samego pliku .cpp,
- konflikty definicji w procesie linkowania (MinGW).

Po dodaniu tych plików(mimo tego że je później usunąłem z projektu jak i z pliku .pro) nie wiem dlaczego cały projekt się rozsypał i więcej nie udało mi się go poprawnie skompilować.

```
make your code fail to compile if it uses deprecated APIs.
 to do so, uncomment the following line.
= QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.

 \
mplot.cpp \
el.cpp \
troller.cpp \
Generator.cpp \
tion.cpp \
vice.cpp \
pp \
ndow.cpp \
mplot.cpp

 \
mplot.h \
el.h \
troller.h \
Generator.h \
tion.h \
```



Po częściowej kompilacji pojawił się błąd:

-Failed to Start Application

-file format not recognized

Największa satysfakcja

Największą satysfakcję dała mi implementacja możliwości modyfikacji parametrów regulatora PID oraz modelu ARX w trakcie trwającej symulacji.
Zmiany są przekazywane przez warstwę usług i nie powodują resetu pamięci obiektu ani

regulatora.

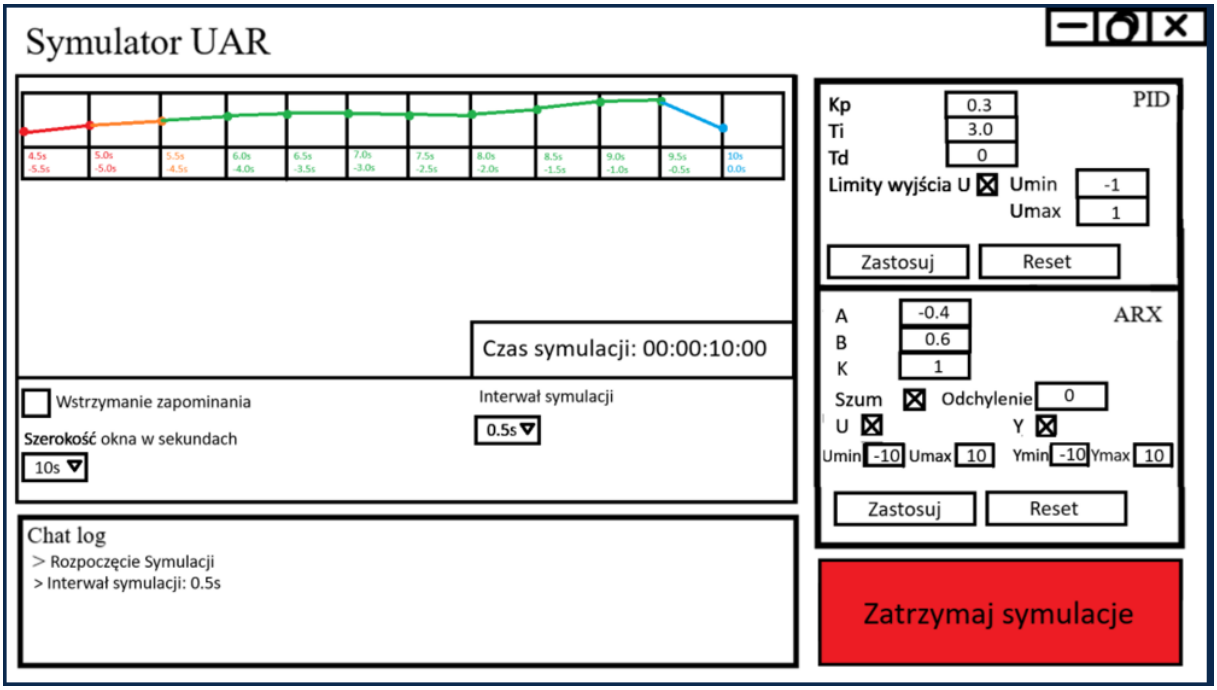
KOD

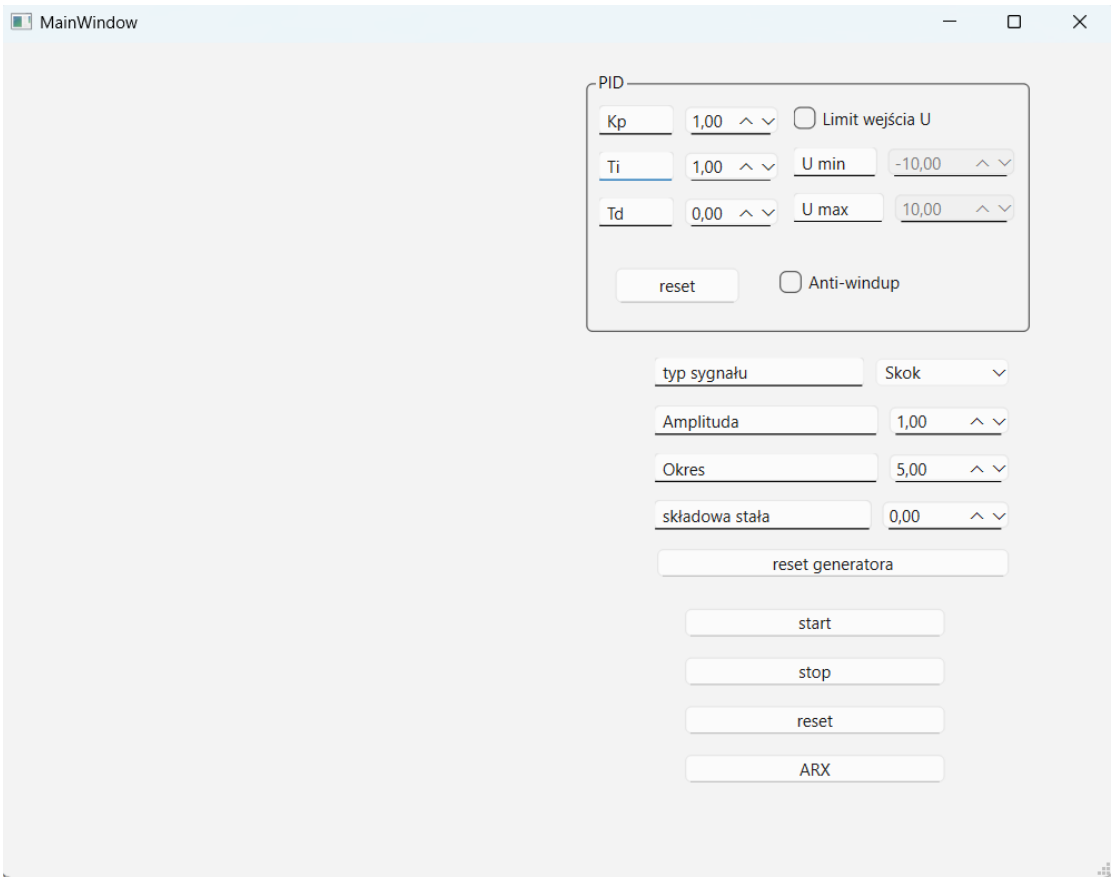
```
// Połączenie sygnałów z odpowiednimi slotami
connect(ui->kpSpinBox, QOverload<double>::of(&QDoubleSpinBox::valueChanged), this, &MainWindow::onKpChanged);
connect(ui->tiSpinBox, QOverload<double>::of(&QDoubleSpinBox::valueChanged), this, &MainWindow::onTiChanged);
connect(ui->tdSpinBox, QOverload<double>::of(&QDoubleSpinBox::valueChanged), this, &MainWindow::onTdChanged);
```

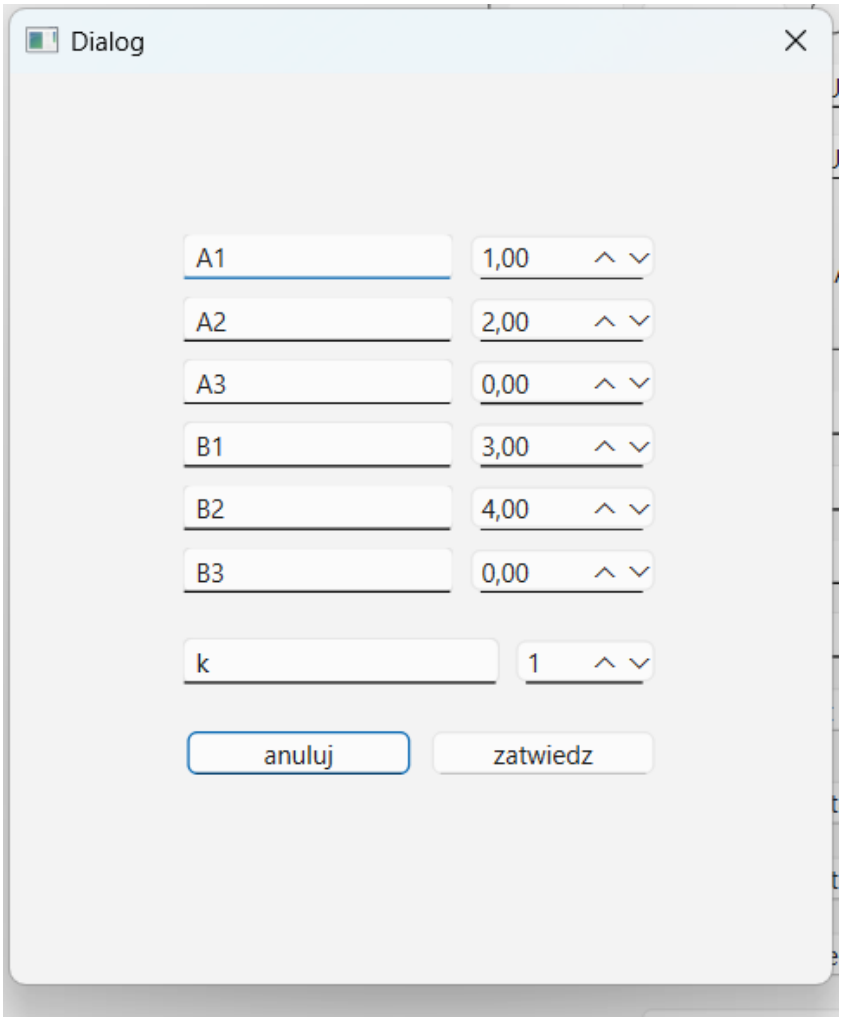
```
// Sloty dla PID
void MainWindow::onKpChanged(double value) {
    service->setPID(value, ui->tiSpinBox->value(), ui->tdSpinBox->value());  // Aktualizowanie PID w UARService

    // service->setPID(
    //     value,
    //     ui->tiSpinBox->value(),
    //     ui->tdSpinBox->value()
    //     );
}
```

Zmiana parametrów PID jest przekazywana z GUI do warstwy usług i działa w trakcie trwania symulacji.

Omówienie modyfikacji wstępnego projektu GUI, które musiały zostać poczynione.


Symulator UAR
4.5s -5.5s
5.0s -5.0s
5.5s -4.5s
6.0s -4.0s
6.5s -3.5s
7.0s -3.0s
7.5s -2.5s
8.0s -2.0s
8.5s -1.5s
9.0s -1.0s
9.5s -0.5s
10s 0.0s
Czas symulacji: 00:00:10:00
Wstrzymanie zapominania
Interwał symulacji
0.5s
Szerokość okna w sekundach
10s
Chat log
> Rozpoczęcie Symulacji
> Interwał symulacji: 0.5s
PID
Kp 0.3
Ti 3.0
Td 0
Limity wyjścia U
Umin -1
Umax 1
Zastosuj
Reset
ARX
A -0.4
B 0.6
K 1
Szum
Odchylenie 0
U
Y
Umin -10 Umax 10
Ymin -10 Ymax 10
Zastosuj
Reset
Zatrzymaj symulacje


MainWindow
PID
Kp
1,00
Limit wejścia U
Ti
1,00
U min
-10,00
Td
0,00
U max
10,00
reset
Anti-windup
typ sygnału
Skok
Amplituda
1,00
Okres
5,00
składowa stała
0,00
reset generatora
start
stop
reset
ARX


Dialog
A1
1,00
A2
2,00
A3
0,00
B1
3,00
B2
4,00
B3
0,00
k
1
anuluj
zatwiedz

Udokumentowanie na bazie próbek kodu, jak warstwa prezentacji komunikuje się z warstwą usług.

Pokazanie, w jaki sposób zmiana parametru regulatora PID w GUI jest przekazywana do warstwy usług, a następnie do warstwy symulacji, bez bezpośredniego dostępu GUI do regulatora.

KROK 1:

MainWindow.cpp – połączenie sygnału z kontrolki GUI

```
// Połączenie sygnałów z odpowiednimi slotami
connect(ui->kpSpinBox, QOverload<double>::of(&QDoubleSpinBox::valueChanged), this, &MainWindow::onKpChanged);
```

Zmiana wartości w kontrolce QDoubleSpinBox generuje sygnał, który wywołuje metodę warstwy prezentacji.

```
// Sloty dla PID
void MainWindow::onKpChanged(double value) {
    service->setPID(value, ui->tiSpinBox->value(), ui->tdSpinBox->value());  // Aktualizowanie PID w UARService

    // service->setPID(
    //     value,
    //     ui->tiSpinBox->value(),
    //     ui->tdSpinBox->value()
    //     );
}
```

Co jest ważne architektonicznie:

- GUI **nie modyfikuje regulatora PID bezpośrednio**
- GUI komunikuje się **wyłącznie z warstwą usług (UARService)**

Krok 2 – Warstwa usług

```
void UARService::setPIDMode(PIDController::IntegralMode mode)
{
    simulation_.pid().setIntegralMode(mode);
}

// ----------------------------
```

Znaczenie:

- UARService pełni rolę **pośrednika**
- Warstwa usług decyduje, **jak i gdzie** zmieniane są parametry regulatora

Krok 3 – Warstwa danych (symulacja)

```cpp
void PIDController::setParams(double Kp, double Ti, double Td)
{
    if (Ti < 0.0 || Td < 0.0)
        throw std::invalid_argument("Ti i Td musza byc >= 0");

    Kp_ = Kp;
    Ti_ = Ti;
    Td_ = Td;

    qDebug()<<"zmiany";
}
```

```cpp
void UARService::setPID(double Kp, double Ti, double Td)
{
    qDebug() << "Setting PID: Kp = " << Kp << ", Ti = " << Ti << ", Td = " << Td;


    simulation_.pid().setParams(Kp, Ti, Td);
}
```