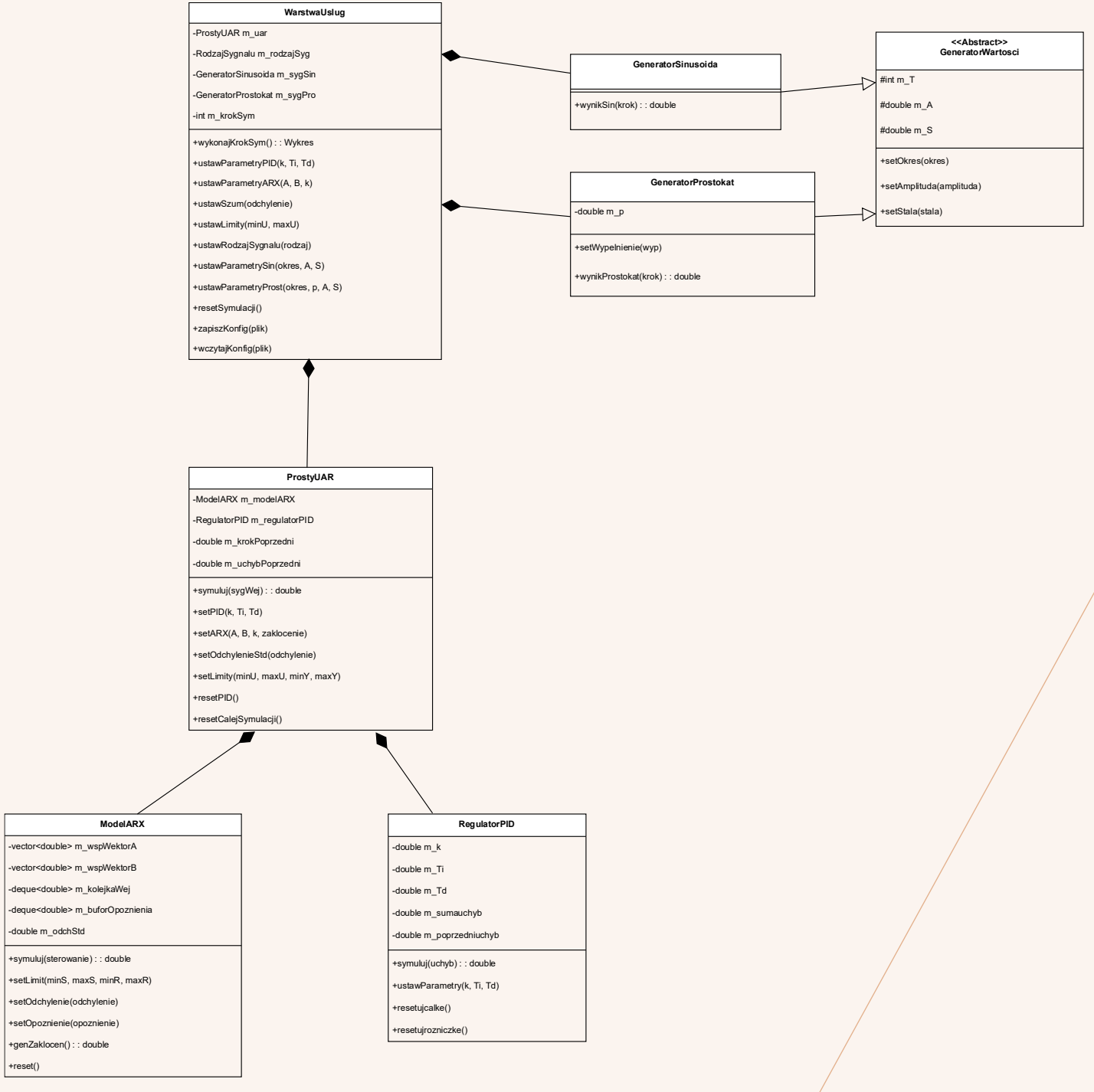
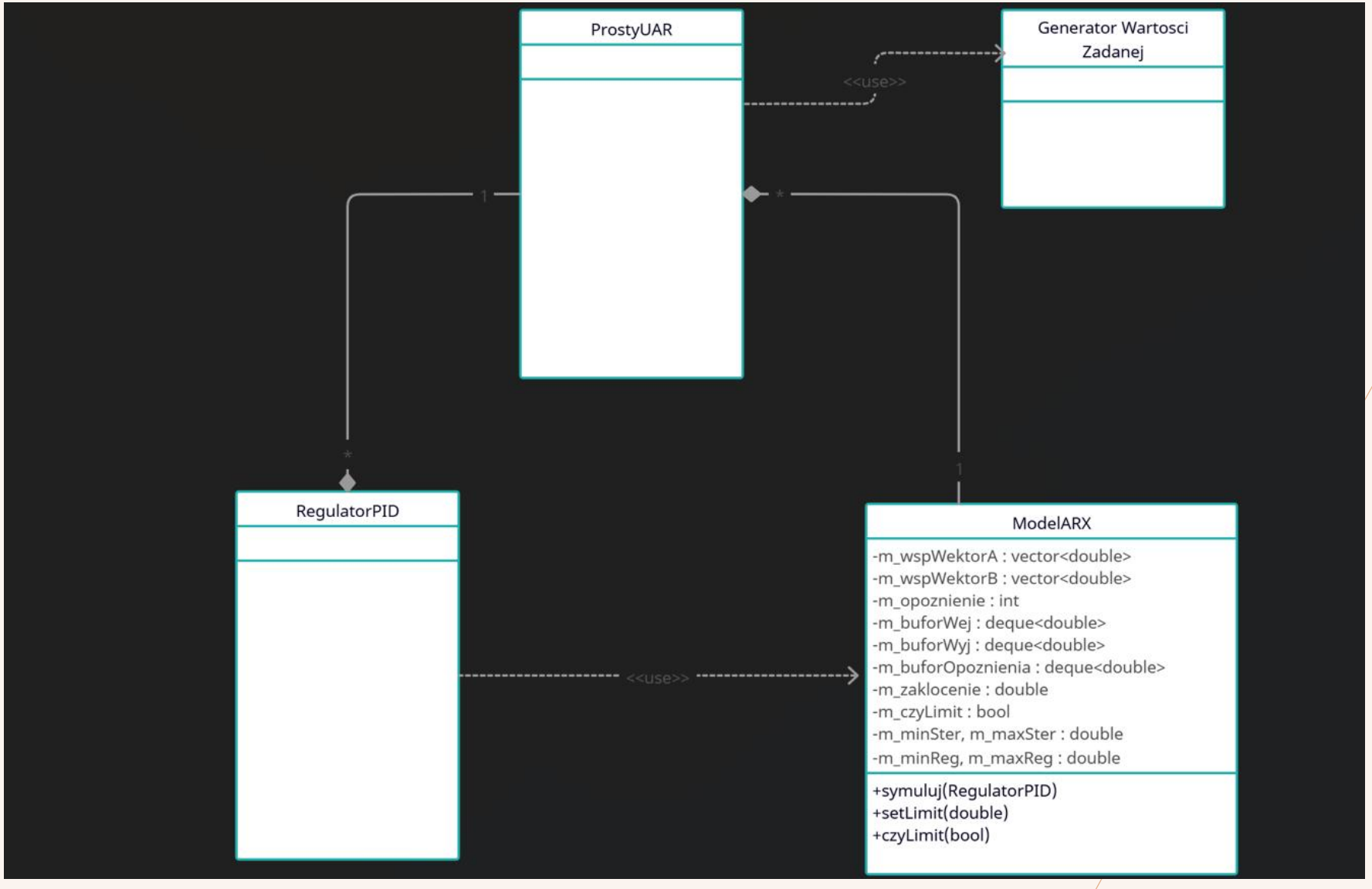
SYMULATOR UKŁADU AUTOMATYCZNEJ REGULACJI

Kacper Skowroński i Igor Juraszek

UML


WarstwaUslug
-ProstyUAR m_uar
-RodzajSygnalu m_rodzajSyg
-GeneratorSinusoida m_sygSin
-GeneratorProstokat m_sygPro
-int m_krokSym
+wykonajKrokSym() : : Wykres
+ustawParametryPID(k, Ti, Td)
+ustawParametryARX(A, B, k)
+ustawSzum(odchylenie)
+ustawLimity(minU, maxU)
+ustawRodzajSygnalu(rodzaj)
+ustawParametrySin(okres, A, S)
+ustawParametryProst(okres, p, A, S)
+resetSymulacji()
+zapiszKonfig(plik)
+wczytajKonfig(plik)
GeneratorSinusoida
+wynikSin(krok) : : double
GeneratorProstokat
-double m_p
+setWypelnienie(wyp)
+wynikProstokat(krok) : : double
<<Abstract>>
GeneratorWartosci
#int m_T
#double m_A
#double m_S
+setOkres(okres)
+setAmplituda(amplituda)
+setStala(stala)
ProstyUAR
-ModelARX m_modelARX
-RegulatorPID m_regulatorPID
-double m_krokPoprzedni
-double m_uchybPoprzedni
+symuluj(sygWej) : : double
+setPID(k, Ti, Td)
+setARX(A, B, k, zaklocenie)
+setOdchylenieStd(odchylenie)
+setLimity(minU, maxU, minY, maxY)
+resetPID()
+resetCalejSymulacji()
ModelARX
-vector<double> m_wspWektorA
-vector<double> m_wspWektorB
-deque<double> m_kolejkaWej
-deque<double> m_buforOpoznienia
-double m_odchStd
+symuluj(sterowanie) : : double
+setLimit(minS, maxS, minR, maxR)
+setOdchylenie(odchylenie)
+setOpoznienie(opoznienie)
+genZaklocen() : : double
+reset()
RegulatorPID
-double m_k
-double m_Ti
-double m_Td
-double m_sumauchyb
-double m_poprzedniuchyb
+symuluj(uchyb) : : double
+ustawParametry(k, Ti, Td)
+resetujcalke()
+resetujroznicze()

PIERWOTNY UML



KŁOPOTLIWA FUNKCJONALNOŚĆ

Aby zresetować symulację, należy wyzerować całą jej historię, co pozwala programowi rozpocząć pracę od stanu początkowego, zamiast korzystać z poprzednio zapamiętanych danych.

```
}
void ProstyUAR::resetSymulacji()
{
    m_krokPoprzedni = 0.0;
    m_uchybPoprzedni = 0.0;
    m_WartSterPoprzedni = 0.0;
    m_regulatorPID.resetujcalke();
    m_regulatorPID.resetujrozniczke();
    m_modelARX.resetARX();
}
```

SATYSFAKCJONUJĄCA FUNKCJONALNOŚĆ

Zastosowanie DRY – wspólne parametry sygnałów dla obu generatorów

Polimorfizm – przeciążenie metody setAmplituda, w sygnale prostokątnym ograniczamy zakres tak, aby amplituda była nieujemna

```
class GeneratorWartosci
{
protected:
    int m_T; //okres
    double m_A; //amplituda
    double m_S; //skladowa stala
public:
    void setOkres(int okres);
    virtual void setAmplituda(double amplituda);
    void setStala(double stala);

    int getOkres();
    double getAmplituda();
    double getStala();
};

class GeneratorSinusoida : public GeneratorWartosci
{
public:
    double wynikSin(int krok);
};

class GeneratorProstokat : public GeneratorWartosci
{
private:
    double m_p; //wypelnienie
public:
    void setWypelnienie(double wyp);
    double getWypelnienie();
    void setAmplituda(double amplituda) override;
    double wynikProstokat(int krok);
};
```

OMÓWIENIE INTERFEJSU WARSTWY USŁUG

```
class WarstwaUslug
{
public:
    enum class RodzajSygnalu { Brak, Sinusoida, Prostokatny };
    struct Wykres { double wartZad, wartReg, uchyb, sterowanie, p, i, d; };  //wartzad i reg -> 1 wykres, p,i,d -> 2 wykres, uchyb -> 3 wykres, ster -> 4 wykres

    Wykres wykonajKrokSym(); //do tworzenia wykresow

    //settery
    void ustawParametryPID(double k, double Ti, double Td);
    void resetPID();
    void ustawParametryARX(const std::vector<double>& A, const std::vector<double>& B, int k);
    void ustawOdchylenie(double odchylenie);
    void ustawLimity(double minU, double maxU);
    void ustawRodzajSygnalu(RodzajSygnalu rodzaj);
    void ustawParametrySin(int okres, double amplituda, double skladowaStala);
    void ustawParametryProst(int okres, double wypelnienie, double amplituda, double skladowaStala);


    void resetSymulacji();
private:
    ProstyUAR m_uar;
    RodzajSygnalu m_rodzajSyg;
    GeneratorSinusoida m_sygSin; //do generowania sinusoidy
    GeneratorProstokat m_sygPro; //do generowania prostokata
    int m_krokSym;               //krok symulacji
```

WYNIKI TESTÓW

```
ModelARX (-0.4 | 0.6 | 1 | 0 ) -> test zerowego pobudzenia: OK!
ModelARX (-0.4 | 0.6 | 1 | 0 ) -> test skoku jednostkowego nr 1: OK!
ModelARX (-0.4 | 0.6 | 2 | 0 ) -> test skoku jednostkowego nr 2: OK!
ModelARX (-0.4, 0.2 | 0.6, 0.3 | 2 | 0 ) -> test skoku jednostkowego nr 3: OK!
RegP (k = 0.5) -> test zerowego pobudzenia: OK!
RegP (k = 0.5) -> test skoku jednostkowego: OK!
RegPI (k = 0.5, TI = 1.0) -> test skoku jednostkowego nr 1: OK!
RegPI (k = 0.5, TI = 10.0) -> test skoku jednostkowego nr 2: OK!
RegPID (k = 0.5, TI = 10.0, TD = 0.2) -> test skoku jednostkowego: OK!
RegPI (k = 0.5, TI = 10.0 -> 5.0 -> 10.0) -> test skoku jednostkowego nr 3: OK!
UAR_1 -> test zerowego pobudzenia: OK!
UAR_1 PID -> test skoku jednostkowego: OK!
UAR_2 PID (k = 2) -> test skoku jednostkowego: OK!
UAR_3 PID (kP=1.0,Ti=2.0) -> test skoku jednostkowego: OK!

PID -> test zerowania calki: OK!

ARX -> test obciecia wejscia (Umax=5): OK!
```

WYNIKI TESTÓW

```cpp
void TESTY_Wlasne::test_1_PID_resetPamieci()
{
    std::cerr << "PID -> test resetu pamieci calkujacej: ";
    try
    {
        RegulatorPID test(1.0, 1.0);

        test.symuluj(1.0);
        test.symuluj(1.0);

        test.resetujcalke(); //sumauchyb=0

        std::vector<double> spodz = { 1.0 };
        std::vector<double> fakt = { test.symuluj(1.0) };

        myAssert(spodz, fakt);
    }
    catch (...)
    {
        std::cerr << "INTERUPTED! (niespodziwany wyjatek)\n";
    }
}
```

```cpp
void TESTY_Wlasne::test_2_ARX_testOgraniczeniaSterowania()
{
    std::cerr << "ARX -> test ograniczenia wejscia U (Umax=5): ";
    try
    {
        ModelARX test({ -0.4 }, { 0.6 }, 1, 0.0);
        //Ustawienie limitu max U = 5.0
        test.setLimit(1.0, 5.0, -10.0, 10.0);

        constexpr size_t LICZ_ITER = 2;
        std::vector<double> sygWe = { 10.0, 3.0 };

        std::vector<double> spodzSygWy = { 0.0, 3.0 };
        std::vector<double> faktSygWy(LICZ_ITER);

        for (int i = 0; i < LICZ_ITER; i++)
            faktSygWy[i] = test.symuluj(sygWe[i]);

        myAssert(spodzSygWy, faktSygWy);
    }
    catch (...)
    {
        std::cerr << "INTERUPTED! (niespodziwany wyjatek)\n";
    }
}
```