PROGRAMOWANIE KOMPUTERÓW PROJEKT (SPOTKANIE II)

SKŁAD SEKCJI:
ARTUR ZABOR,
MICHAŁ WALTEROWSKI

PODSUMOWANIE SPOTKANIA I

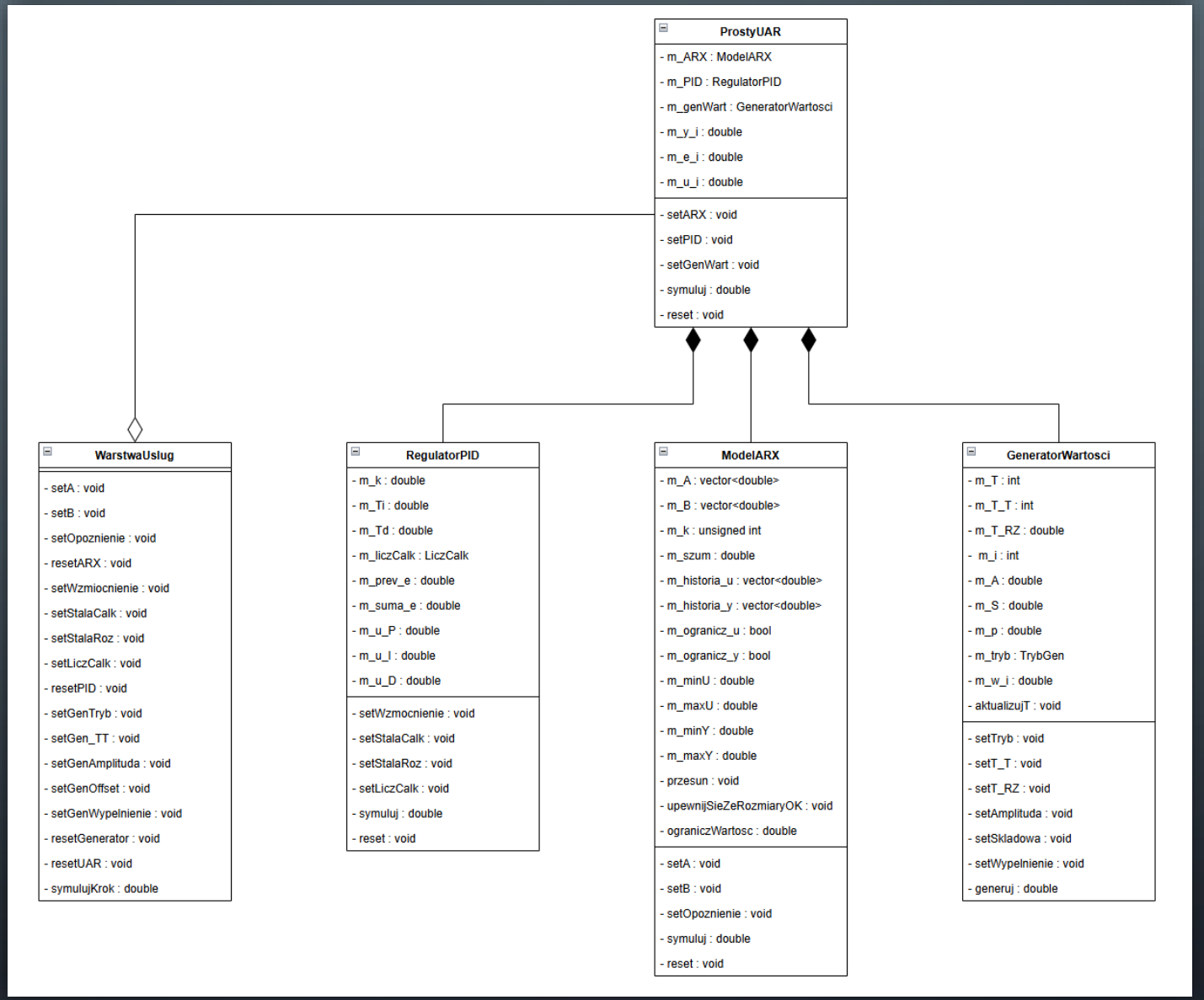
ZAPREZENTOWANIE WSTĘPNEGO PROJEKTU APLIKACJI

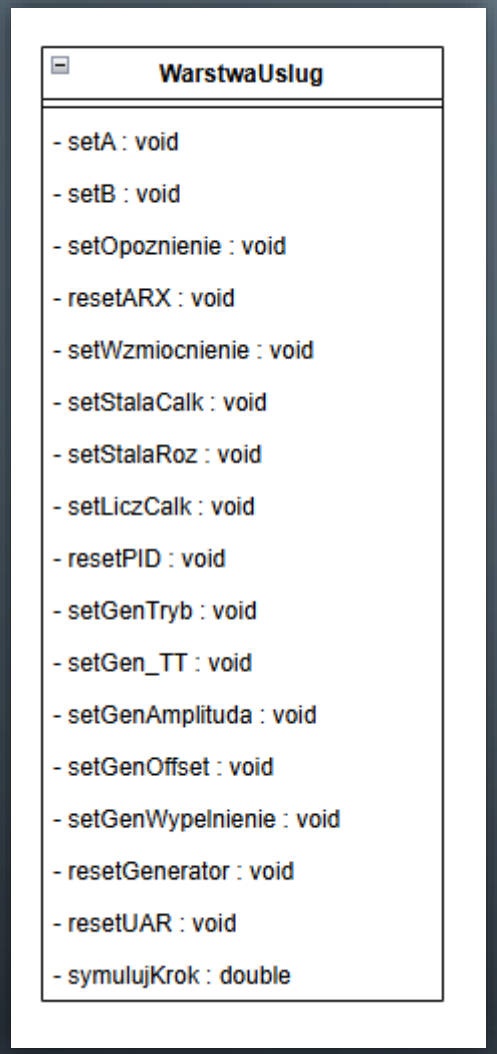
ZAPREZENTOWANIE WSTĘPNEGO PODZIAŁU PRAC

WSTĘPNY SCHEMAT HIERARCHII KLAS UML

ZAIMPLEMENTOWANA KLASA ARX WRAZ Z ZAŁĄCZONYMI TESTAMI

SCHEMAT UML - SZCZEGÓŁOWY





OMÓWIENIE INTERFEJSU KLASY WARSTWY USŁUG

- Relacja między warstwą usług a UARem to agregacja
- W warstwie usług zawierają się wszystkie setery z wszystkich klas

KOD – NAJWIĘCEJ KŁOPOTÓW

Najwięcej kłopotów sprawiło zaimplementowanie funkcji symuluj, która odpowiada za jeden krok symulacji modelu PID. Trudności wynikały z sposobu jak zaimplementować wszystkie działania w odpowiedniej kolejności

```
double RegulatorPID::symuluj(double e_zadane)
{
    m_u_P = m_k * e_zadane * 1.0;

    m_u_I = 0.0;

    if (m_Ti != 0.0)
        {
            if(m_liczCalk == LiczCalk::Wew)
            {
                m_suma_e += e_zadane / m_Ti;
                m_u_I = m_suma_e * 1.0;
            }
            else
            {
                m_suma_e += e_zadane;
                m_u_I = m_suma_e / m_Ti * 1.0;
            }
        }


    m_u_D = m_Td * (e_zadane - m_prev_e) * 1.0;


    double u = m_u_P + m_u_I + m_u_D;


    m_prev_e = e_zadane;

    return u;
}

void RegulatorPID::reset()
{
    m_prev_e = 0.0;
    m_suma_e = 0.0;
}
```

KOD – NAJWIĘKSZA SATYSFAKCJA

Najwięcej satysfakcji sprawiło zaimplementowanie funkcji generuj, w generatorze wartości odpowiadającej za tryb generatora prostokątny lub sinusoidalny. Powodem dlaczego ten fragment nas najbardziej zadowolił był z faktu, że na początku nam nie działał, a było to związane z błędnie użytym znakiem w równaniu. Po użyciu odpowiedniego symbolu wszystko działało.

```
227
228  double GeneratorWartosci::generuj()
229  {
230
231      switch(m_tryb)
232      {
233          case TrybGen::Pros:
234              if ((m_i % m_T) < (m_p * m_T))
235                  m_w_i = m_A + m_S;
236              else
237                  m_w_i = m_S;
238              break;
239          case TrybGen::Sin:
240              m_w_i = m_A * sin((m_i % m_T * 1.0) / m_T * 2 * 3.1416 * 1.0) * 1.0 + m_S;
241              break;
242      }
243
244      m_i++;
245
246      return m_w_i;
247  }
248
```

KOD - PID

```h
// Klasa implementująca regulator PID
class RegulatorPID
{
private:
    // Wzmocnienie regulatora (pierwsza nastawa regulatora)
    double m_k;

    // Stała całkująca (druga nastawa regulatora)
    double m_Ti;

    // Stała różniczkowania (trzecia nastawa regulatora)
    double m_Td;

    // Tryb pracy części całkującej ((0) - ze stałą przed sumą; (1) - ze stałą pod sumą)
    LiczCalk m_liczCalk;

    // Bufory historii sterowania i wyjścia
    double m_prev_e = 0.0; // poprzednie wartości e
    double m_suma_e = 0.0;

    double m_u_P;
    double m_u_I;
    double m_u_D;

public:
    // Konstruktor
    RegulatorPID(double wzmocnieneie_k = 0.0, double stalaC = 0.0, double stalaR = 0.0, LiczCalk tryb = LiczCalk::Wew);

    void setWzmocnienie(double wzmocnienie_k);
    void setStalaCalk(double stalaC);
    void setStalaRoz(double stalaR);
    void setLiczCalk(LiczCalk rodzaj);

    // Główna funkcja modelu
    double symuluj(double sterowanie);

    void reset();
};
```

```cpp
RegulatorPID::RegulatorPID(double wzmocnieneie_k, double stalaC, double stalaR, LiczCalk tryb)
    :m_k(wzmocnieneie_k), m_Ti(stalaC), m_Td(stalaR), m_liczCalk(tryb)
{}

void RegulatorPID::setWzmocnienie(double wzmocnienie_k)
{
    m_k = wzmocnienie_k;
}

void RegulatorPID::setStalaCalk(double stalaC)
{
    m_Ti = stalaC;
}

void RegulatorPID::setStalaRoz(double stalaR)
{
    m_Td = stalaR;
}

void RegulatorPID::setLiczCalk(LiczCalk rodzaj)
{
    if (rodzaj != m_liczCalk)
    {
        if (rodzaj == LiczCalk::Wew)
            m_suma_e = m_suma_e / m_Ti * 1.0;
        else
            m_suma_e = m_suma_e * m_Ti * 1.0;
    }


    m_liczCalk = rodzaj;
}
```

```cpp
double RegulatorPID::symuluj(double e_zadane)
{
    m_u_P = m_k * e_zadane * 1.0;

    m_u_I = 0.0;

    if (m_Ti != 0.0)
        {
            if(m_liczCalk == LiczCalk::Wew)
            {
                m_suma_e += e_zadane / m_Ti;
                m_u_I = m_suma_e * 1.0;
            }
            else
            {
                m_suma_e += e_zadane;
                m_u_I = m_suma_e / m_Ti * 1.0;
            }
        }


    m_u_D = m_Td * (e_zadane - m_prev_e) * 1.0;


    double u = m_u_P + m_u_I + m_u_D;


    m_prev_e = e_zadane;

    return u;
}

void RegulatorPID::reset()
{
    m_prev_e = 0.0;
    m_suma_e = 0.0;
}
```

KOD - GENERATOR

```h
class GeneratorWartosci
{
private:
    int m_T;

    int m_T_T;
    double m_T_RZ;

    int m_i = 1;

    double m_A;
    double m_S;
    double m_p;
    TrybGen m_tryb;

    double m_w_i;

    void aktualizujT();
public:
    GeneratorWartosci(TrybGen tryb, int t_T, double t_RZ, double a, double s, double p = 0.5);

    void setTryb(TrybGen tryb);
    void setT_T(int t_T);
    void setT_RZ(double t_RZ);
    void setAmplituda(double a);
    void setSkladowa(double s);
    void setWypelnienie(double p);

    double generuj();
    void reset();
};
```

```cpp
GeneratorWartosci::GeneratorWartosci(TrybGen tryb, int t_T, double t_RZ, double a, double s, double p)
    :m_tryb(tryb), m_T_T(t_T), m_T_RZ(t_RZ), m_A(a), m_S(s)
{
    setWypelnienie(p);
    aktualizujT();
}

void GeneratorWartosci::setTryb(TrybGen tryb)
{
    m_tryb = tryb;
}

void GeneratorWartosci::setT_T(int t_T)
{
    m_T_T = t_T;
    aktualizujT();
}

void GeneratorWartosci::setT_RZ(double t_RZ)
{
    m_T_RZ = t_RZ;
    aktualizujT();
}

void GeneratorWartosci::aktualizujT()
{
    m_T = m_T_RZ * m_T_T / 10;
}

void GeneratorWartosci::setAmplituda(double a)
{
    m_A = a;
}

void GeneratorWartosci::setSkladowa(double s)
{
    m_S = s;
}
```

```cpp
void GeneratorWartosci::setWypelnienie(double p)
{
    if (p < 0.0)
    {
        m_p = 0.0;
        return;
    }
    if (p > 1.0)
    {
        m_p = 1.0;
        return;
    }
    m_p = p;
}

double GeneratorWartosci::generuj()
{
    switch(m_tryb)
    {
        case TrybGen::Pros:
            if ((m_i % m_T) < (m_p * m_T))
                m_w_i = m_A + m_S;
            else
                m_w_i = m_S;
            break;
        case TrybGen::Sin:
            m_w_i = m_A * sin((m_i % m_T * 1.0) / m_T * 2 * 3.1416 * 1.0) * 1.0 + m_S;
            break;
    }

    m_i++;

    return m_w_i;
}

void GeneratorWartosci::reset()
{
    m_i = 1;
    m_w_i = 0.0;
}
```

KOD - UAR

```h
// Klasa implementująca UAR
class ProstyUAR
{
private:
    ModelARX m_ARX;
    RegulatorPID m_PID;
    GeneratorWartosci m_genWart;

    // double w_i = 0.0;
    double m_y_i = 0.0;
    double m_e_i;
    double m_u_i;

public:
    // Konstruktor
    ProstyUAR(ModelARX arx, RegulatorPID pid, GeneratorWartosci genWart);

    void setARX(ModelARX arx);
    void setPID(RegulatorPID pid);
    void setGenWart(GeneratorWartosci genWart);


    double symuluj();

    void reset();
};
```

```cpp
ProstyUAR::ProstyUAR(ModelARX arx, RegulatorPID pid, GeneratorWartosci genWart)
    :m_ARX(arx), m_PID(pid), m_genWart(genWart)
{}

void ProstyUAR::setARX(ModelARX arx)
{
    m_ARX = arx;
}

void ProstyUAR::setPID(RegulatorPID pid)
{
    m_PID = pid;
}

void ProstyUAR::setGenWart(GeneratorWartosci genWart)
{
    m_genWart = genWart;
}

double ProstyUAR::symuluj()
{
    m_e_i = m_genWart.generuj() - m_y_i;

    m_u_i = m_PID.symuluj(m_e_i);

    m_y_i = m_ARX.symuluj(m_u_i);

    return m_y_i;
}

void ProstyUAR::reset()
{
    m_ARX.reset();
    m_PID.reset();
    m_genWart.reset();

    m_e_i = 0.0;
    m_u_i = 0.0;
    m_y_i = 0.0;
}
```

KOD – WARSTWA USŁUG

```
class UARService {
public:

    // ARX
    void setA(ModelARX& arx, const std::vector<double>& A);
    void setB(ModelARX& arx, const std::vector<double>& B);
    void setOpoznienie(ModelARX& arx, unsigned int k);
    void resetARX(ModelARX& arx);

    // PID
    void setWzmocnienie(RegulatorPID& pid, double k);
    void setStalaCalk(RegulatorPID& pid, double Ti);
    void setStalaRoz(RegulatorPID& pid, double Td);
    void setLiczCalk(RegulatorPID& pid, LiczCalk tryb);
    void resetPID(RegulatorPID& pid);

    // Generator
    void setGenTryb(GeneratorWartosci& gen, TrybGen tryb);
    void setGen_TT(GeneratorWartosci& gen, int TT);
    void setGen_TRZ(GeneratorWartosci& gen, double TRZ);
    void setGenAmplituda(GeneratorWartosci& gen, double A);
    void setGenOffset(GeneratorWartosci& gen, double S);
    void setGenWypelnienie(GeneratorWartosci& gen, double p);
    void resetGenerator(GeneratorWartosci& gen);

    // UAR
    void resetUAR(ProstyUAR& uar);
    double symulujKrok(ProstyUAR& uar);
};
```

```
// ARX
void UARService::setA(ModelARX& arx, const vector<double>& A) { arx.setA(A); }
void UARService::setB(ModelARX& arx, const vector<double>& B) { arx.setB(B); }
void UARService::setOpoznienie(ModelARX& arx, unsigned int k) { arx.setOpoznienie(k); }
void UARService::resetARX(ModelARX& arx) { arx.reset(); }

// PID
void UARService::setWzmocnienie(RegulatorPID& pid, double k) { pid.setWzmocnienie(k); }
void UARService::setStalaCalk(RegulatorPID& pid, double Ti) { pid.setStalaCalk(Ti); }
void UARService::setStalaRoz(RegulatorPID& pid, double Td) { pid.setStalaRoz(Td); }
void UARService::setLiczCalk(RegulatorPID& pid, LiczCalk tryb) { pid.setLiczCalk(tryb); }
void UARService::resetPID(RegulatorPID& pid) { pid.reset(); }

// Generator
void UARService::setGenTryb(GeneratorWartosci& g, TrybGen t) { g.setTryb(t); }
void UARService::setGen_TT(GeneratorWartosci& g, int TT) { g.setT_T(TT); }
void UARService::setGen_TRZ(GeneratorWartosci& g, double TRZ) { g.setT_RZ(TRZ); }
void UARService::setGenAmplituda(GeneratorWartosci& g, double A) { g.setAmplituda(A); }
void UARService::setGenOffset(GeneratorWartosci& g, double S) { g.setSkladowa(S); }
void UARService::setGenWypelnienie(GeneratorWartosci& g, double p) { g.setWypelnienie(p); }
void UARService::resetGenerator(GeneratorWartosci& g) { g.reset(); }

// UAR
void UARService::resetUAR(ProstyUAR& uar) { uar.reset(); }
double UARService::symulujKrok(ProstyUAR& uar) { return uar.symuluj(); }
```

TESTY JEDNOSTKOWE

```
arturzabor@MacBook-Air-Artur projekt % g++ -std=c++17 UAR.cpp Testy_UAR.cpp -o projekt
./projekt
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!
```

```
arturzabor@MacBook-Air-Artur projekt % g++ -std=c++17 UAR.cpp Testy_UAR.cpp -o projekt
./projekt
ModelARX -> test reset(): OK!
ModelARX -> test setA/setB(): OK!
ModelARX -> test ograniczenia sterowania U: OK!
RegPID -> test reset(): OK!
GeneratorWartosci(Sin) -> test pierwszej próbki: OK!
GeneratorWartosci(Pros) -> test 50% wypełnienia: OK!
GeneratorWartosci -> test reset(): OK!
ProstyUAR -> test reset(): OK!
ProstyUAR -> test sygnalu prostokatnego: OK!
UARService -> test setterów GUI: OK!
```

DZIĘKUJEMY ZA UWAGĘ