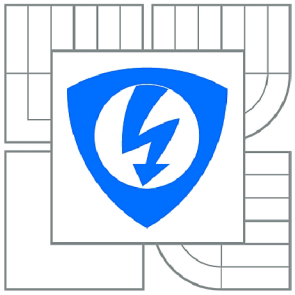
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

OPC KLIENT PRO MODELOVÁNÍ REGULACE V SYSTÉMU COMES

OPC CLIENT FOR REGULATORY CONTROL MODELLING IN COMES SYSTEM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

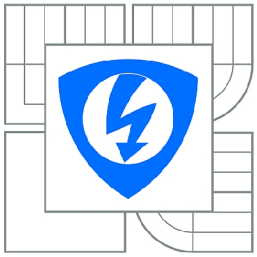
AUTOR PRÁCE — JAN BRZOBOHATÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN PÁSEK, CSc.

SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Ing. Jan Brzobohatý	***ID:***	125136
Ročník:	3	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

OPC klient pro modelování regulace v systému COMES

POKYNY PRO VYPRACOVÁNÍ:

Navrhnout způsob vytvoření OPC klienta, který bude simulovat odezvu systému popsaného diferenciální rovnicí. Realizovat tohoto OPC klienta a ověřit jeho funkceschopnost.

DOPORUČENÁ LITERATURA:

[1] COMPAS AUTOMATIZACE. Výrobní informační systém COMES verze 3: COMES CCI – Příručka pro nastavení. Ždár nad Sázavou: COMPAS, 2010
[2] FAJMON, Břetislav a Irena RŮŽIČKOVÁ. Matematika 3. Brno, 2005. Skriptum. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav matematiky.
[3] JURA, Pavel. Signály a systémy: Část 1: Spojité signály. 2.opr.vyd. Brno, srpen 2010. Skriptum. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií.
[4] JURA, Pavel. Signály a systémy: Část 2: Spojité systémy. 2.opr.vyd. Brno, srpen 2010. Skriptum. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií.

Termín zadání:	11.2.2013	***Termín odevzdání:***	27.5.2013

Vedoucí práce: Ing. Jan Pásek, CSc.
Konzultanti bakalářské práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

Abstrakt

Vytvoření programu OPC klienta, který simuluje chování spojitého řízeného systému. Simulovaný spojitý systém je popsán diferenciální rovnicí. Numerické řešení diferenciální rovnice simuluje odezvu spojitého systému. OPC klient je připojen k řídicímu systému s regulačním programem prostřednictvím OPC serveru. Odezva vypočtená OPC klientem a akční zásah řídicího systému přenášené OPC komunikací umožňují modelovat a odladit regulační program implementovaný v řídicím systému.

Klíčová slova

Simulace spojitého systému, modelování spojitého systému, OPC klient, diferenciální rovnice, metoda Runge-Kutta.

Abstract

It was created program of the OPC client which simulates behavior of a continuous dynamical system. The simulated continuous dynamical system is described by the differential equation. The numerical solution of the differential equation simulates response of the dynamical system. The OPC client is connected to the control system with the regulatory program via the OPC server. The response calculated by OPC client and the action of the control system program transferred via OPC communication allow modeling and tuning the continuous control program implemented in the control system.

Key words

Continuous dynamical system simulation, continuous dynamical system modeling, OPC client, differential equation, Runge-Kutta method.

Bibliografická citace:

BRZOBOHATÝ, J. *OPC klient pro modelování regulace v systému COMES*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2013. 38s. Vedoucí bakalářské práce byl Ing. Jan Pásek, CSc.

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma OPC klient pro modelování regulace v systému COMES jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne: 25. května 2013

.....

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Janu Páskovi za metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: 25. května 2013

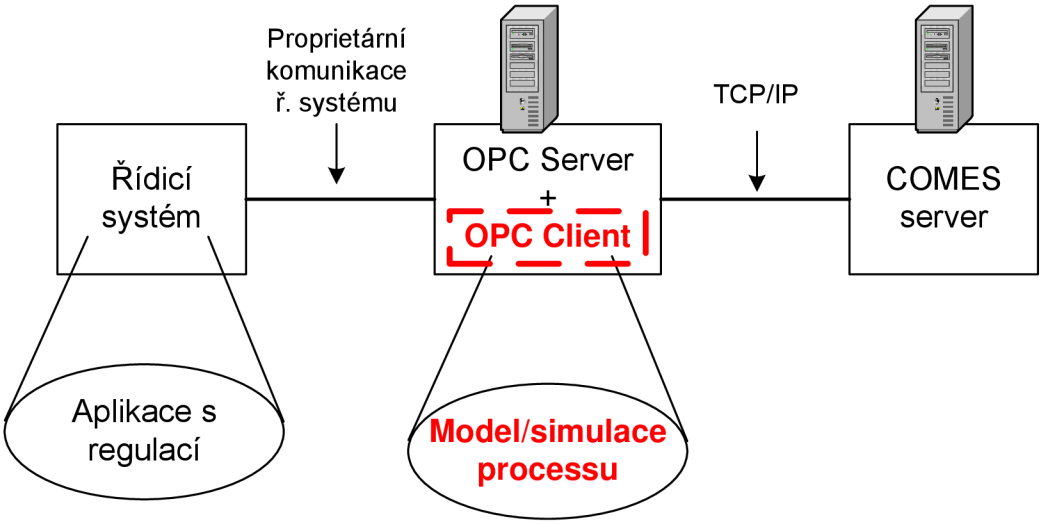
.....

podpis autora

Obsah

1. Úvod

Úkolem bakalářská práce bylo vytvořit program OPC klienta, který simuluje chování spojitého systému popsaného diferenciální rovnicí. Hotový program OPC klienta má sloužit jako pomůcka pro ladění regulací v řídicím systému. OPC klient je propojen s řídicím systémem přes OPC server. Podstatou simulace je numerické řešení diferenciální rovnice. Výsledek řešení je hodnota, která simuluje odezvu spojitého řízeného systému. OPC klient/simulátor prostřednictvím OPC rozhraní a OPC serveru může komunikovat s regulátorem implementovaným v aplikačním programu řídicího systému – viz Obr. 1.1. Komunikace umožňuje přenos akčního zásahu z řídicího systému do OPC klienta a přenos simulované odezvy z OPC klienta do řídicího systému. Tímto způsobem je možné odladit regulátor v řídicím systému již během vytváření aplikačního programu před fyzickou instalací řídicího systému.
OPC klient je primárně určen pro výrobní informační systém COMES fy COMPAS Automatizace, ale obecně ho lze použít pro jakýkoliv systém o podobné konfiguraci. OPC klient, který je navržen v této práci, nemá sám o sobě smysl. Měl by být používán jako část konfigurace podobné na Obr. 1.1.



Obr. 1.1: *Modelování regulace s použitím OPC klienta simulujícího spojitý systém.*

2. Základní návrh

Informační systém COMES je naprogramován převážně v jazyce C#, proto program pro modelování regulace je také napsán v C#. Program OPC klienta obsahuje dva základní moduly:

- Modul simulátoru
- Modul OPC klienta

Vlastní simulátor řeší diferenciální rovnici popisující spojitý systém. Počítá odezvu simulovaného spojitého systému na základě zadané diferenciální rovnice a dále podle akční veličiny obdržené z modulu OPC klienta. Vypočítaná odezva se pak odešle zpět do modulu OPC.

Modul OPC komunikuje s OPC serverem. Přes OPC server komunikuje s řídicím systémem (popřípadě to může být i něco jiného, například jiný OPC klient komunikující s tímto OPC serverem). OPC klient se musí připojit k OPC serveru, nakonfigurovat komunikaci s OPC serverem, načíst akční zásah z řídicího systému a předat akční zásah do modulu simulátoru. Dále musí zapsat do OPC serveru simulovanou odezvu vypočítanou a předanou modulem simulátoru.

Oba moduly budou běžet nezávisle na sobě ve dvou různých vláknech. Budou si jen vyměňovat potřebná data. Modul OPC klienta přitom je nadřazen simulátoru, může ho startovat, pozastavit, a nastavovat případně některé parametry simulátoru, a přebírat některé základní indikační údaje o jeho chodu.

3. Simulace spojitého systému

Budeme se zabývat simulací spojitého systému – viz skripta *Signály a Systémy* [3],[4]. Chování spojitého systému se dá popsat diferenciální rovnicí. Předpokládáme, že budeme simulovat spojitý systém následujících vlastností:

- Systém se soustředěnými parametry. Takový spojitý systém se dá popsat obyčejnými diferenciálními rovnicemi.

- Systém může být lineární i nelineární. Řešení diferenciální rovnice popisující nelineární systém musí být spojité včetně derivací až do řádu n-1.

- Systém je časově invariantní. To znamená, že odezva systému nezávisí na časovém počátku (tj. na stejné počáteční podmínky a stejný vstup systém odpoví vždy stejnou odezvou bez ohledu na aktuální čas počátku). V tomto případě se v diferenciální rovnici explicitně neobjevuje proměnná času.

- Systém má jeden vstup *u* a jeden výstup *y*. K popisu systému musí stačit jedna diferenciální rovnice maximálně 4 řádu.

3.1. Metoda Runge-Kutta 4. řádu

Modelovat spojitý systém znamená řešit diferenciální rovnicí. Podle skripta *Matematika 3* [2] je vhodné řešit diferenciální rovnici numerickou *metodou Runge-Kutta 4. řádu*. Aby řešení dosahovalo potřebné přesnosti, numerický výpočet obsahuje také řízení délky kroku *metodou polovičního kroku* [2].

Obyčejná diferenciální rovnicí [2] popisuje spojitého *t*-invariantního systému má obecný tvar

$$\frac{d^n y(t)}{dt^n} = f\left(u, y, y^{(1)}, ..., y^{(n-1)}\right), \tag{1}$$

kde je

t	čas,
$u(t)$	vstup (akční zásah) do spojitého systému,
$y(t)$	výstup (odezva) spojitého systému,
$y^{(i)}$	je *i*-tá derivace výstupu *y*.
n	řád diferenciální rovnice,

Pokud se bude počítat lineární diferenciální rovnice, očekává se její zadání v podobném tvaru

$$\frac{d^n y(t)}{dt^n} = b \cdot u(t) - a_{n-1}\frac{d^{n-1} y(t)}{dt^{n-1}} - ... - a_1 \frac{dy(t)}{dt} - a_0 y(t) \tag{2}$$

kde

a_i	konstantní koeficienty lineární diferenciální rovnice pro výstup *y* ($i = 0, ..., n-1$),
b	konstantní koeficient pro vstup *u*.

Abychom byli schopni řešit diferenciální rovnici (1), je nutné znát počáteční podmínky

$$y^{(n-1)}(0), y^{(n-2)}(0), ..., y^{(1)}(0), y(0) \ . \tag{3}$$

Metoda Runge-Kutta 4. řádu [2] je jako taková určena pro řešení počáteční úlohy obyčejné diferenciální rovnice 1. řádu

$$y' = f(t, y) \quad , \qquad y(0) = y_0 \tag{4}$$

Abychom byli schopni řešit diferenciální rovnici *n*-tého řádu (1) metodou Runge-Kutta, musíme rovnici (1) převést na soustavu *n* rovnic 1. řádu (5). S ohledem na to, že rovnice je *t*-invariantní, čas *t* není v rovnicích explicitně obsažen. Zavedeme *n* nových proměnných x_i, kde vzhledem k implementaci v jazyce C# index i má rozsah $0,...,n-1$. Pak soustava *n* rovnic 1. řádu včetně počátečních podmínek vypadá

$$\begin{aligned}
x_0 &= y & x_0(0) &= y_0 \\
x_1 &= y^{(1)} = x_0' & x_1(0) &= y_0' \\
x_2 &= y^{(2)} = x_1' & x_2(0) &= y_0^{(2)} \\
&..... & &..... \\
x_{n-1} &= y^{(n-1)} = x_{n-2}' & x_{n-1}(0) &= y_0^{(n-1)} \\
y^{(n)} &= x_{n-1}'
\end{aligned} \tag{5}$$

Rovnici musí být zadaná ve tvaru (1) nebo (2), kde nejvyšší derivace je na levé straně a vše ostatní na straně pravé je vyjádřeno funkcí $f\left(u, y, y^{(1)},..., y^{(n-1)}\right)$. Dostaneme soustavu diferenciálních rovnic (6), která naznačuje postup výpočtu od x_{n-1} k x_0.

$$\begin{aligned}
x_{n-1}' &= f_{n-1}(x_0,....,x_{n-1},u) = f\left(u, x_0, x_1,..., x_{n-1}\right) \\
x_{n-2}' &= f_{n-2}(x_{n-1}) = x_{n-1} \\
&..... \\
x_1' &= f_1(x_2) = x_2 \\
x_0' &= f_0(x_1) = x_1
\end{aligned} \tag{6}$$

A vyřešením poslední diferenciální rovnice v soustavě dostaneme výstup – odezvu simulovaného spojitého systému.

$$y = x_0 \tag{7}$$

Metodou Runge-Kutta se tedy řeší diferenciální rovnic *n*-tého řádu. Metoda Runge-Kutta počítá v zadaném časovém úseku $\langle 0, T \rangle$ hodnoty proměnné y (resp. x_i) po časových krocích o délce *h*. Zavedeme si index *j*, který bude označovat číslo výpočetního kroku, tj. například pro proměnnou x_i použití indexu *j* znamená

$$\begin{aligned}
x_{i,j} &= x_i(t) \\
x_{i,j+1} &= x_i(t+h)
\end{aligned} \tag{8}$$

Jeden výpočtový krok je popsán v tomto odstavci. Výpočtový algoritmus Runge-Kutta 4. řádu potřebuje spočítat čtyři parametry k_1, k_2, k_3, k_4 v každém výpočetním kroku. Tyto parametry se musí spočítat zvlášť pro každou stavovou proměnnou x_i z hodnot posledního celého spočteného kroku *j* podle následujících vztahů

$$k_{1,i} = f_i(x_{0,j}, ..., x_{n-1,j}, u) \tag{9}$$

$$k_{2,i} = f_i(x_{0,j} + \frac{h \cdot k_{1,i}}{2}, ..., x_{n-1,j} + \frac{h \cdot k_{1,i}}{2}, u) \tag{10}$$

$$k_{3,i} = f_i(x_{0,j} + \frac{h \cdot k_{2,i}}{2}, ..., x_{n-1,j} + \frac{h \cdot k_{2,i}}{2}, u) \tag{11}$$

$$k_{4,i} = f_i(x_{0,j} + h \cdot k_{3,i}, ..., x_{n-1,j} + h \cdot k_{3,i}, u) \tag{12}$$

Krok metody Runge-Kutta je zakončen výpočtem hodnoty s indexem *j+1* pro všechny stavy x_i

$$x_{i,j+1} = x_{i,j} + \frac{h}{6}\left(k_{1,i} + 2 \cdot k_{2,i} + 2 \cdot k_{3,i} + k_{4,i}\right) \tag{13}$$

Stavy x_i se počítají v algoritmu v sestupném pořadí od $i = n-1$ po $i = 0$.

3.2. Přesnost výpočtu

Přesnost numerických metod typu Runge-Kutta 4. řádu závisí na délce časového kroku *h*. Pokud počítané hodnoty rychle rostou nebo klesají, pak se zvětšuje jejich výpočtová chyba oproti exaktnímu analytickému výpočtu. Zvětšování chyby se dá zabránit zkrácením časového kroku *h*. V tomto simulátoru se používá jednoduchá metoda poloviční kroku. Spočte se rozdíl nově vypočtené hodnoty a hodnoty z minulého kroku a tento rozdíl se porovná s požadovanou přesností výpočtu.

$$\left|x_{i,j+1} - x_{i,j}\right| > PožadovanáPřesnost \tag{14}$$

Pokud je rozdíl větší, viz rovnice (14), pak se výpočet opakuje s tím, že nový časový krok je poloviční a výpočtový krok se provádí dvakrát, jednou pro interval $\langle 0, h/2 \rangle$ a podruhé pro interval $\langle h/2, h \rangle$. Implementace v programu je provedena s použitím rekurzívní funkce, takže se dá zjemňování časového kroku opakovat i pro již poloviční krok.

3.3. Provedení výpočtu

Výpočet je naprogramován v separátní třídě C# jako samostatná třída "DiffEq", takže v případě použití v jakémkoliv programu stačí vytvořit objekt téhle třídy, zadat diferenciální rovnici ve tvaru (1), rsp. (2), případně změnit parametry výpočtu a dát povel ke startu výpočtu "CmdStart". Výpočet by se měl startovat až po navázání komunikace s OPC serverem a po přečtení vstupu *u*.

Výpočet běží ve dvou hlavních smyčkách. Nadřazená smyčka běží periodicky v časovém intervalu daném parametrem "Cyklus". Velikost tohoto časového intervalu je dána schopnostmi OPC komunikace, a to především, jak rychle a často se může komunikovat. Tato smyčka zavádí do programu práci v reálném čase, přesně časuje předávání vstupu *u* do výpočtu, start výpočtu a odeslání vypočteného výstupu *y* do OPC serveru.

Na začátku nadřazené smyčky "ModelCyklus" (
Obr. 3.1) se

1) přečte hodnota vstupu *u* do výpočtu,

spustí se podřízená smyčka "RungeKuttaCyklus" – viz

2) Obr. 3.2. Podřízená smyčka provádí vlastní výpočet metodou Runge-Kutta.

```
// Hlavni cyklus modelu/simulatoru
public void ModelCyklus()
{
  .....
  while (Run && !konec)
  {
    .....
    // predani vstupu - akce "u" do algoritmu
    uAction = U;
    // vypocet
    RungeKuttaCyklus( ioCyklus);
    // cekani na dobeh cyklu
    do
    {
      ...
    } while ( ! konecCyklus );
    Y = yOutput;
    // poslani vystupu nekam
    if ( writeY!=null )
      writeY( Y );
    ....
  };  // while
}  // void
```

***Obr. 3.1**: Struktura nadřazeného cyklu simulátoru "ModulCyklus"*

Počítá hodnoty v časovém intervalu (0, Cyklus) se zadaným krokem *h*. Opakovaně se provádí jeden krok metody Runge-Kutta podle popisu v kapitole 3.1 a 3.2 – viz Obr. 3.3. Předpokládá se, že podřízená smyčka běží značně rychleji než je velikost intervalu "Cyklus". Doba běhu podřízené smyčky se měří a na základě tohoto měření se počítá zatížení (Load) modulu simulátoru. Po ukončení podřízené smyčky "RungeKuttaCyklus" se čeká na konec smyčky "ModelCyklus". V tom okamžiku se vypočtená hodnota výstupu *y* odešle přes odesílací funkci "writeY" (typu SendYEventHandler) z modulu OPC klienta do OPC serveru.

```
// Hlavni cyklus metody RK4
// extCyklus - casovy interval zadany z nadrazeneho procesu
// pro tento zadany interval se ma dodavat vypocetny vystup "yOutput"
// Aby se to stihlo, tak vlastni vypocet neprobiha v realnem case
void  RungeKuttaCyklus( double extCyklus)
{
  .....
  // cely cyklus Runge-Kutt. metody 4.radu
  jRK4 = 0;
  // počet cyklu prováděných v RungeKuttaCyklus
  int ncykl = (int)(Math.Round(extCyklus / hStep));
  int jcykl = 0;
  do
  {
    if (krok1)
    {
      // nastaveni pocatecnich hodnot
      InitVypocet();
      krok1 = false;
      .....
    }
    jcykl++;
    neniPresne = false;

    // jeden krok Runge Kutt. metody
        RungeKuttaKrok(jcykl*hStep,  hStep);

  } while ( jcykl < ncykl );
  yOutput = x_j[0];
  .....
  Load = ms / (10 * extCyklus);
  if (Load > MaxLoad) MaxLoad = Load;
}
```

***Obr. 3.2**: Struktura podřízené smyčky simulátoru "RungeKuttalCyklus"*

Výpočet se dá ukončit povelem "CmdStop".

Při prvním běhu podřízené smyčky "RungeKuttaCyklus" se nastaví hodnoty $x_{i,j}$ (tj. naposled spočtený krok) podle zadaných počátečních podmínek. V případě, že počáteční podmínky nejsou zadány, tak jsou nulové. Pokud existuje možnost zpětně přečíst hodnotu výstupu y z OPC serveru, dá se tato hodnota použít jako počáteční podmínka pří startu výpočtu. Také je možné vnucením této hodnoty během výpočtu simulovat poruchu.

```
// Vypočet jednoho kroku metody RK4 ,
// vcetne rizeni delku kroku metodou puleni intervalu
void RungeKuttaKrok(double cas, double h)
{
  .....
  // zaloha hodnot posledniho kroku pro pripad puleni kroku,
  .....
  // vypocet k1
  kvypocet(x_jm1, uAction, k1);
  // vypocet k2
  for (int i = 0; i < n; i++)
    xaux[i] = x_jm1[i] + k1[i] * h / 2.0;
  kvypocet(xaux, uAction, k2);
  // vypocet k3
  for (int i = 0; i < n; i++)
    xaux[i] = x_jm1[i] + k2[i] * h / 2.0;
  kvypocet(xaux, uAction, k3);
  // vypocet k4
  for (int i = 0; i < n; i++)
    xaux[i] = x_jm1[i] + k3[i] * h;
  kvypocet(xaux, uAction, k4);
  // Vypocet hodnot v aktualnim kroku - Runge-Kutt. 4 radu
  for (int i = 0; i < n; i++)
    x_j[i] = XNew(x_jm1[i], k1[i], k2[i], k3[i], k4[i], h);
  // Zjistit presnost vypoctu
  neniPresne = neniPresne || TestPresnostiNeprosel();
  if (neniPresne && presnostPov)
  {
    neniPresne = false;
    // Musime obnovit puvodni hodnoty z minuleho kroku
    for (int i = 0; i < n; i++)
      x_j[i] = x_zal[i];
    // Zjemnime krok na polovinu a musime provest 2 výpočty
    // 1. vypocet pro "t-h/2", 2.výpočet pro "t"
    for (int i = -1; i <= 0; i++)
      RungeKuttaKrok(cas + i * h / 2.0, h / 2.0);
  }
}
```

***Obr. 3.3**: Struktura funkce výpočtu jednoho kroku metodou Runge-Kutta 4.řádu*

3.4. Zadání výpočtu

Parametry při startu programu OPC klienta se načtou z konfiguračního XML souboru. Všechny parametry potřebné pro výpočet simulace jsou v Tab. 3.1.

```
  <Equation_Order>4</Equation_Order>
  <Differential_Equation>dy(4)=u-y-2*dy(1)-dy(2)-3*dy(3)
</Differential_Equation>
  <Linear_Equation Enabled="false" b_u="1">
    <a_y>
      <Coefficient ID="a(0)" Value="-1" />
      <Coefficient ID="a(1)" Value="-2" />
      <Coefficient ID="a(2)" Value="-1" />
      <Coefficient ID="a(3)" Value="-3" />
    </a_y>
  </Linear_Equation>
  <Simulation_Parameters CalculationCycle="1000" CalculationStep="100"
Accurracy="1E-05" Debug="false">
    <InitialConditions_dy>
      <Initial_Condition ID="y_0" InitValue="0" />
      <Initial_Condition ID="dy(1)_0" InitValue="0" />
      <Initial_Condition ID="dy(2)_0" InitValue="0" />
      <Initial_Condition ID="dy(3)_0" InitValue="0" />
    </InitialConditions_dy>
  </Simulation_Parameters>
```

Tab. 3.1: *Zadání diferenciální rovnice a parametry simulačního výpočtu.*

Parametr **<Equation_Order>** obsahuje řád diferenciální rovnice.

Diferenciální rovnice je možno zadat dvěma způsoby. Prvním způsob je s použitím parametru **<Differential_Equation>**, který umožňuje zadat obyčejnou diferenciální rovnici, a to jak lineární, tak i nelineární. Rovnice se zadává jako matematický výpočet v jazyce Visual Basic. Přitom se mohou používat všechny matematické operátory a funkce, které Visual Basic obsahuje. Výraz zadávající diferenciální rovnici však může obsahovat jen povolené proměnné "**u**", "**y**", a "**dy(*i*)**", kde i představuje řád derivace veličiny "**y**". Proměnná "**dy(4)**" znamená čtvrtou derivaci výstupu *y*. Podobně **dy(3), dy(2), dy(1)** znamenají 3., 2. a první derivaci výstupu *y*. "**y**" je samozřejmě výstup *y* z výpočtu modelu. Například zadaní diferenciální rovnice (15)

$$\frac{d^{(4)}y}{dt^4} = \frac{d^{(3)}y}{dt^3} + \frac{d^{(2)}y}{dt^2} + \frac{dy}{dt} + y + u \qquad (15)$$

vypadá

<Differential_Equation>
dy(4)=u+y+dy(1)+dy(2)+dy(3)
</Differential_Equation>

Písmeno "**u**" znamená akční veličinu *u* načtenou OPC klientem z řídicího systému přes OPC server. Všechny proměnné se píší malým písmem. Rovnice v parametru **<Differential_Equation>** musí být kompatibilní s jejím řádem zadaným parametrem **<Equation_Order>**. Derivační člen s nejvyšším řádem musí být sám jediný na levé straně rovnítka, všechny ostatní členy jsou umístěny na pravé straně rovnice.
Zadávaná rovnice by měla být vypočitatelná numerickou metodou. Například při dělení některou z proměnných "**dy(4), dy(2), dy(1), y, u**" může dojít k dělení nulou, nebo některá hodnota může být třeba jen velice malá. Po načtení parametru **<Differential_Equation>** z konfiguračního XML souboru, se vzorec z parametru zkompiluje. Zkompilovaná výpočetní funkce se umístí v paměti počítače, a používá se stejným způsobem jako jakákoliv jiná funkce vytvořená kompilátorem Visual Basic a umístěná například v DLL souboru.

Druhý způsob zadání umožňuje zadat jen lineární diferenciální rovnicí parametrem **<Linear_Equation>**. Tento způsob je volitelný. Dá se povolit atributem "**Enabled**". Koeficienty *b* a a_i pro lineární diferenciální rovnice jsou nastaveny atributem "**b_u**" a položkami "**<a_y>**". Pole položek **<a_y>** musí uvádět všechny koeficienty počínaje 0.řádem, žádný řád nesmí být vynechán. Hodnota "0" pak znamená, že příslušný člen se v diferenciální rovnici nevyskytuje. Tento způsob výpočtu je asi o 5% rychlejší než způsob s parametrem **<Differential_Equation>**. Pokud je lineární diferenciální rovnice povolena atributem "**Enabled**", tak se výpočet podle **<Differential_Equation>** neprovádí.

Další parametr potřebný pro výpočet simulované odezvy je atribut "**CalculationCycle**", který nastavuje délku výpočetního intervalu metodou Runge-Kutta. Hodnota je v milisekundách. Je to také doba nazývaná výše externí cyklus.

Krok metody Runge-Kutta je zadán atributem "**CalculationStep**". Hodnota je opět v milisekundách. Bývá obvykle alespoň desetkrát menší než hodnota výpočetního intervalu metody "**CalculationCycle**". Měl by to být celočíselný podíl výpočetního cyklu "**CalculationCycle**".

Požadovaná přesnost výpočtu je zadána atributem "**Accurracy**". Pokud není požadovaná přesnost dosažena během jedno kroku, tak se aktuální krok rozdělí na dvě poloviny, a výpočetní funkce metody Runge-Kutta se rekurzívně volá pro oba poloviční intervaly. Touto *metodou polovičního kroku* se dá sice dosáhnout požadované přesnosti, ale někdy vede k opravdu zdlouhavým výpočtů, kdy hloubka rekurze dosahuje čísla 20, což vede k neúnosně dlouhým výpočtů. Například výpočet intervalu 1000 ms s krokem 100 ms trval i 20 sekund, což je nepoužitelné pro simulátor, který má fungovat online. To se samozřejmě stává v místech, kde odezva strmě stoupá, či klesá. To bylo potlačeno zavedením tzv. minimální délky kroku. Když přidělení kroku je velikost kroku menší než minimální, tak v tomto bodě půlení kroku končí. Minimální délka kroku je v

programu pevně fixována na délku kroku jako desítitisícina délky kroku "**CalculationStep**". To omezuje hloubku rekurze asi na 14 úrovní. Pokud zatížení (Load) překročí 50%, doporučuji buď snížit přesnost parametrem atributem "**Accurracy**", anebo zvětšit výpočetní krok "**CalculationStep**", to také zvedne minimální délku kroku.

Parametr "**<InitialConditions_dy>**" umožňuje zadat počáteční podmínky pro výpočet metodou Runge-Kutta. Je to pole položek, a musí uvádět všechny podmínky počínaje 0.řádem, žádná podmínka nesmí být vynechána. Program však ignoruje počáteční podmínku "y_0". Výpočet je totiž zahájen až po začátku OPC komunikace. První hodnota *y* načtená z řídicího systému (rsp.OPC serveru) se pak použije jako počáteční podmínka pro výpočet.

Atribut "**Debug**" může povolit generování souboru DEBUG.LOG s ladícími informacemi, které se generují pro každý výpočetní cyklus (interval). Obsahuje tvar výpočetní rovnice, čas výpočtu, délku výpočtu, zatížení, výsledky a dosažená přesnost. Doporučuje se povolit generování tohoto souboru jen krátkodobě po ladicí účely.

3.5. Ovládání výpočtu a stavy výpočtu

Výpočet simulátoru je možné ovládat. Následující základní povely jsou součástí třídy "DiffEq" – viz Tab. 3.2.

Povel /Funkce	Popis
OpcClient.SimModel.CmdStart	Tato funkce nastartuje výpočet simulace spojitého systému,nebo restartuje pozastavený výpočet.
OpcClient.SimModel.CmdPause	Tato funkce pozastaví výpočet simulace spojitého systému.
OpcClient.SimModel.CmdStop	Tato funkce úplně zastaví výpočet.

Tab. 3.2: *Seznam povelů pro ovládání výpočtu simulátoru*

Třída simulátoru "DiffEq" vrací některé stavy výpočtu – viz Tab. 3.3.

Povel /Funkce	Popis
OpcClient.SimModel.Y	Výstup "y" simulovaného spojitého systému.
OpcClient.SimModel.Run	Indikace typu "bool". Indikuje chod výpočtu simulátoru.
OpcClient.SimModel.Load	Aktuální zatížení simulátoru výpočtem. Závisí na rychlosti počítače. Pokud je více než 60-70%, může se stát, že při nějaké dodatečné činnosti počítače se výpočet nestihne během intervalu "Cyklus". Pak je třeba upravit přesnost výpočtu – viz parametr "Accuracy",nebo upravit délku kroku "Step" tak, aby se zatížení snížilo.
OpcClient.SimModel.MaxLoad	Maximální zatížení dosažené během chodu programu OPC klienta.
OpcClient.SimModel.CalcMsMax	Maximální dosažená délka výpočtu metodou Runge-Kutta v cyklu "RungeKuttaCyklus". Vynuluje se při použití některého z povelů pro ovládání výpočtu – viz **Chyba! Nenalezen zdroj odkazů.** . Hodnota je v milisekundách.

***Tab. 3.3**: Stavy indikující chod simulátoru*

3.6. Zápis výstupu *y* do OPC serveru

Třída simulátoru "DiffEq" implementuje delegáta

public delegate void SendYEventHandler(double y);

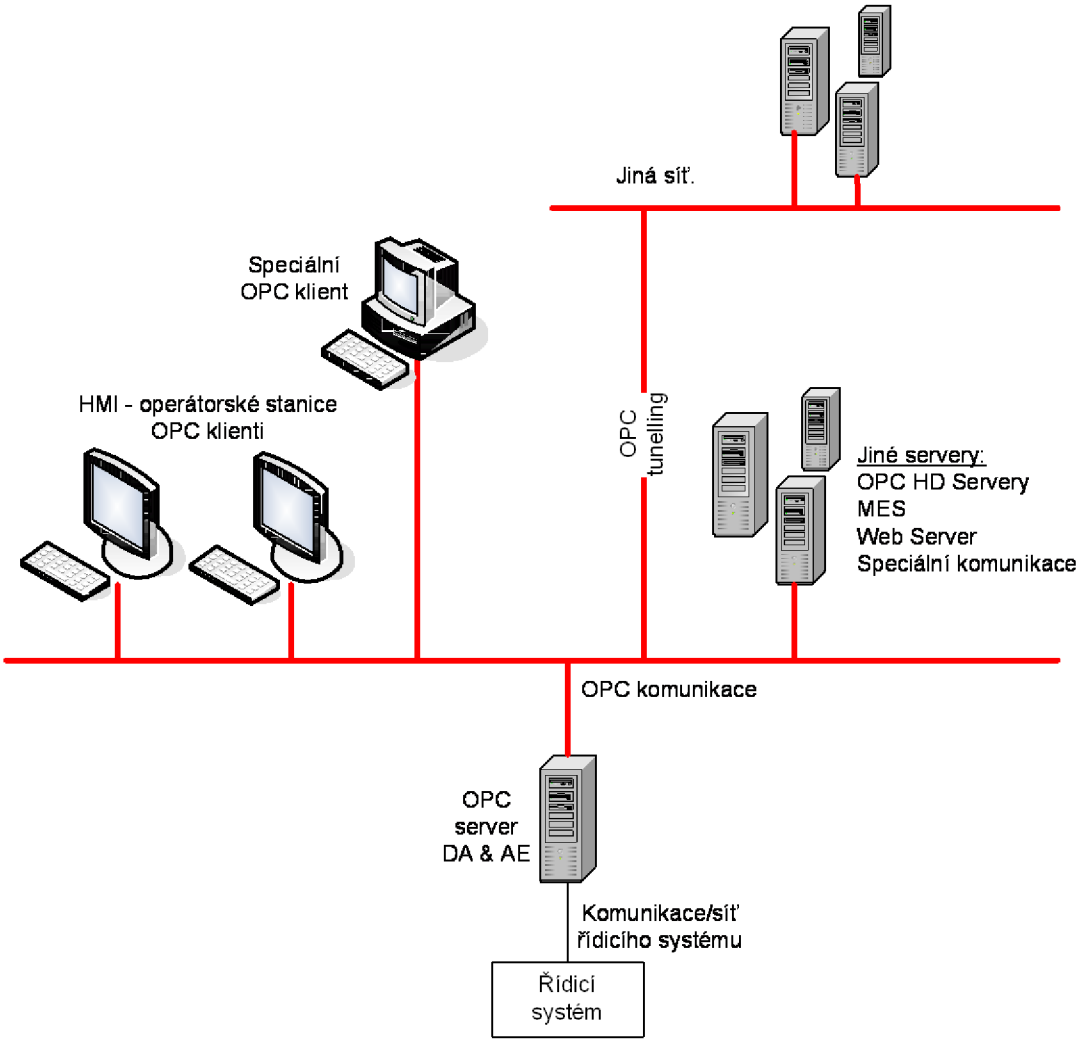
Dále třída "DiffEq" obsahuje ukazatel na funkci typu "SendYEventHandler(double y)", tj. stejného typu jako výše zmíněný delegát. Předpokládá se, že program, který používá třídu "DiffEq" (v našem případě modul OPC) vytvoří funkci, která parametr funkce "double y" nějakým způsobem zpracuje (např. zapíše do OPC serveru). Při vytváření instance "DiffEq" se adresa takovéto funkce předá jako parametr konstruktoru. V cyklu "ModelCyklus" se funkce typu "SendYEventHandler" volá okamžitě při ukončení cyklu "Cyklus" – viz též odstavec 3.3 .

4. Modul OPC klienta

Modul OPC je v podstatě jednoduchý OPC DA klient, který přes OPC server komunikuje s řídicím systémem podle Obr. 4.1. OPC server a OPC klient jsou softwarové aplikace. Mohou běžet na stejném počítači, ale i na různých počítačích. V případě, že OPC klient běží na jiném počítači než OPC server, je třeba splňovat požadavky kladené specifikací DCOM [11], které se týkají nastavení některých služeb operačního systému Windows.

4.1. Seznámení s OPC

OPC jsou komunikační standardy založené na technologii Microsoft Windows,



***Obr. 4.1** : Možná architektura řízení procesu s použitím OPC komunikace*

konkrétně rozhraní OLE/COM [5], [16]. Zkratka OPC byla odvozena z názvu "OLE for Process Control". V současné době zkratka OPC je odvozena z nového názvu "Open Productivity & Connectivity", který byl zaveden s ohledem na vývoj nového standardu OPC Unified Architecture (OPC UA), který již není závislý na OLE technologii. Závislost starších OPC standardů (tzv. OPC Classic) na operačním systému Windows omezovala OPC komunikaci na vyšší úrovně řízení. OPC standardy se úspěšně prosazují v posledních 15 letech. To značně zjednodušuje výběr prostředků HMI, MES, protože aplikace od různých výrobců mohou navzájem komunikovat díky OPC komunikaci. Možnosti OPC jsou zobrazeny na Obr. 4.1.

Standard OPC byl vyvinut koncem 90. let organizací OPC Foundation (www.opcfoundation.org). OPC Foundation umožňuje členům a zájemcům přístup k OPC specifikacím, toolkitům, hotovým knihovnám, a vzorovým programům. Registrovaná fyzická osoba s úrovní Non-Member může stáhnout a použít některý z balíčků hotových DLL knihoven "OPC Redistributable" [10], který je nutno použít při spouštění OPC klienta naprogramovaného v jazyce C#. Hotové programy pro komunikaci OPC, které nemají tyto softwarové balíčky jako součást instalace, vyžadují instalaci těchto knihoven. Programy napsané jazykem C# se překládají do specifického kódu, tzv. "Managed Code". Aby takový program bylo možné spustit, musí být na počítači nainstalovány také potřebné verze .Net Framework.

Dokumentace pro OPC komunikaci na stránkách OPC Foundation [5], [6], [10] je spíše referenčního charakteru bez jasného popisu, jakým způsobem se jednotlivé funkce řadí v programu. OPC Foundation předpokládá, že se to zájemci naučí sami studiem vzorových programů. Další možnost, jak se naučit programovat OPC komunikaci, je využít zdrojů na internetu, jako jsou zdroje [13], [14] a [15]. Informace z těchto volných zdrojů byly použity při tvorbě programu.

Rozhraní OPC objektů jsou implementovány v OPC serverech. OPC klient se může připojovat k OPC serverům, které mohou být i od různých výrobců. Jeden OPC server může být mít připojeny i více OPC klientů. Pokud jsou nějaká omezeni, musí být specifikována výrobcem OPC serveru, rsp klienta.

V současné době standard OPC nabízí starší a nový způsob komunikace .

A. Starší standardy, které obsahují několik způsobů přenosu podle typu přenášených dat (v současné době nazývaný Classic OPC). Přitom pro každý tento typ musí existovat speciální OPC server a speciální OPC klient. Jde o:
   - OPC Data Access – přenos aktuálních hodnot mezi procesem a vyšší úrovní řízení, popřípadě horizontálně ve stejné úrovni řízení.

- OPC Historical Data Access – přenos archivovaných dat pro další zpracování (trendy, reporty, MES, aj.).
- OPC Alarms & Events – přenos alarmů a událostí z procesu do seznamů alarmů, či událostí, a také do archivace, včetně stavu a časové značky.

B. Novější standard, nazývaný OPC Unified Architecture (OPC UA). Tento standard pokrývá všechny zmíněné druhy přenášených dat, a má řadu dalších vylepšení a změn. Zatím ale není příliš rozšířený. Dvě ze zásadních změn jsou, že OPC standard přestal být závislý na technologii Microsoft OLE/COM, a jako další změna či vylepšení se uvádí, že OPC UA bude možno používat i na jiných platformách než jen MS Windows, například pod Unixem, či v embedded zařízeních.

4.2. Zkratky

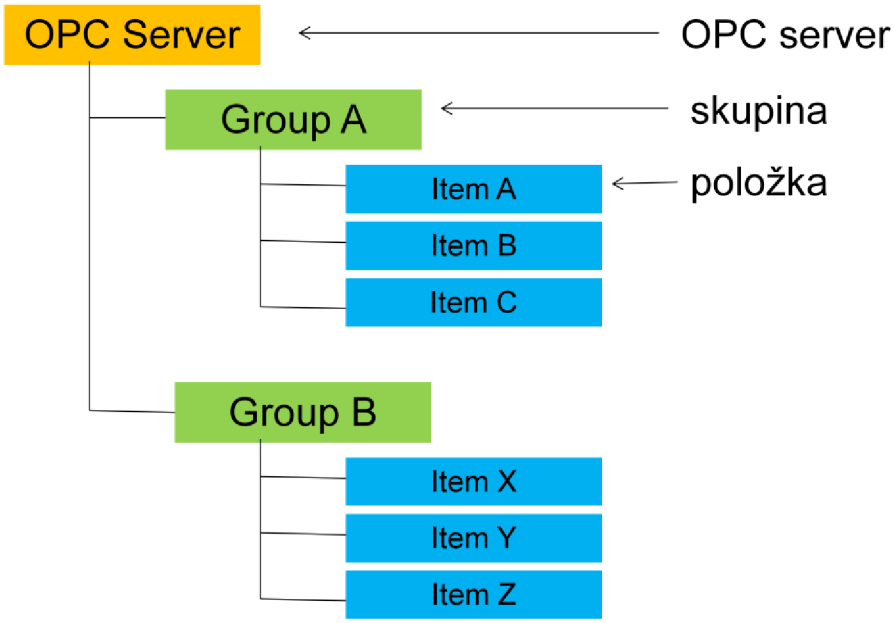
Na základě zdroje Wikipedia [9] je uveden popis zkratek často používaných v OPC dokumentaci.

OPC	Open Productivity & Connectivity – nový název OLE for Process Control - starší název (Informace viz OPC Foundation, http://www.opcfoundation.org)
OLE	Object Linking and Embedding (softwarová technologie používaná v operačním systému Windows od fy Microsoft)
COM	Component Object Model (softwarové rozhraní od fy Microsoft)
DCOM	Distributed Component Object Model (rozšíření technologie COM o distribuci komponent mezi počítači propojených počítačovou sítí)
MMI, HMI, HSI	Man Machine Interface, Human Machine Interface, Human System Interface (jakékoliv rozhraní mezi systémem/strojem a člověkem – ovládací pult, obrazovka, klávesnice, operátorská stanice, apod.)
MES	Manufacturing Execution System (výrobní informační systém)
OPC DA	OPC Data Access (standard pro přenos datových údajů)
OPC HDA	OPC Historical Data Access (standard pro přenos historizovaných datových údajů)
OPC AE	OPC Alarm & Events (standard pro přenos alarmů a událostí)

Tab. 4.1: *Seznam zkratek používaných v OPC terminologii.*

4.3. OPC Data Access

Objekty uvnitř OPC Data Access serveru mají tři úrovně: Server, Group (skupina), Item (položka) . Jsou uspořádány hierarchickým způsobem – viz Obr. 4.2.



Obr. 4.2: *Hierarchie OPC*

Objekt "Server" obsahuje informace o serveru.

Objekt "Group" obsahuje informace o sobě a umožňuje OPC klientovi zorganizovat data. "Group" je obvykle vytvořena na požadavek klienta, takže "Group" reprezentuje data, která mají něco společného, například údaje zobrazované na aktuální obrazovce.

OPC klient také specifikuje, jakým způsobem jsou data ve skupině "Group" čtena z OPC serveru, nebo zapisována do OPC severu. OPC klient také specifikuje, jakou rychlostí OPC server posílá "Group" data OPC klientovi.

OPC položka "Item" představuje propojení ke zdroji dat v OPC serveru. OPC klient nemá rozhraní, kterým by se bylo možno dostat k položkám přímo v OPC serveru. OPC klient se může dostat k OPC položkám pouze skrz "OPC Group".

Každá OPC DA položka má minimálně následující vlastnosti:

- Value (hodnota)
- Quality (kvalita,stav)
- Time Stamp (časová značka)

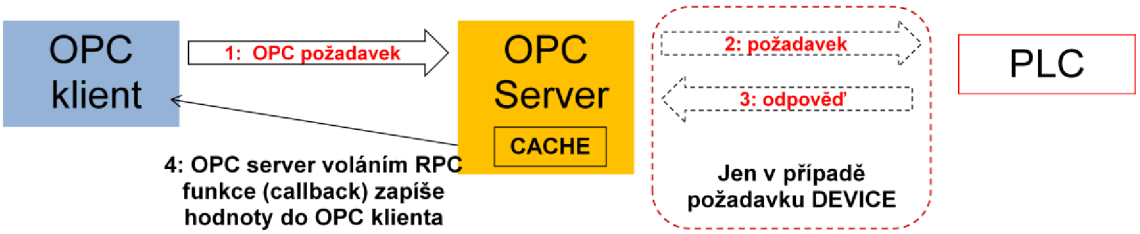
Většina OPC serverů může při předávání dat OPC klientům používat dva zdroje:

- DEVICE (zařízení) – v tomto případě se data přečtou z např. z PLC do OPC serveru a pak se pošlou OPC klientovi.
- CACHE – zvláštní oblast paměti v OPC serveru. OPC klient specifikuje přes OPC Group vlastnost "UpdateRate", jak často má OPC server číst data z procesního kontrolleru (např.PLC) a ukládat je do paměti CACHE. Interval čtení z DEVICE je pak obvykle rovný hodnotě "UpdateRate" milisekund. Interval posílání dat do OPC klienta bude ""UpdateRate" a nebo větší než "UpdateRate". Z různých důvodů OPC server může mít tento interval delší. Proto se "UpdateRate" také často nazývá minimální komunikační interval čtení do OPC klienta. Když "UpdateRate" je nastaven na 0, tak OPC server by měl periodicky komunikovat s PLC podle svých možností co nejčastěji.

OPC klient také určuje jakým způsobem se bude komunikovat mezi OPC klientem a OPC serverem. Existují tři druhy možné komunikace:

- **synchronous** (synchronní) – komunikace je podobná jednoduchým sériovým protokolům - Obr. 4.3. OPC klient pošle požadavek na přečtení dat a čeká na vrácená data. OPC server v tomto případě okamžitě čte požadovaná data z PLC. Po přečtení dat z PLC je ihned odesílá OPC klientovi. Důsledkem je, že komunikace je pomalá, ale přečtená data jsou tzv. čerstvá. OPC server je navíc zatížený mimořádnou komunikací, a OPC klient je zatížený čekací smyčkou.

Obr. 4.3: *Schéma synchronní komunikace OPC.*

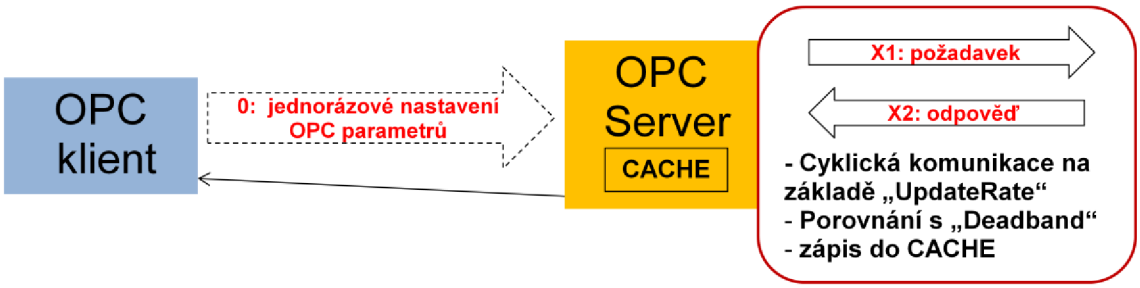


Obr. 4.4: *Schéma asynchronní komunikace OPC.*

- **asynchronous** (asynchronní) – OPC klient pošle požadavek na čtení dat, a pak si tak zvaně jde po svých - Obr. 4.4. OPC klient specifikuje, zda data mají být ze zdroje DEVICE nebo CACHE. V případě zdroje DEVICE OPC server musí přečíst data z PLC. Pokud OPC klient specifikuje jako zdroj dat CACHE, tak OPC server

vezme poslední data z pamětí CACHE. Jakmile OPC server má data, tak je pošle OPC klientu voláním callback funkce "OnReadComplete". V tomto případě OPC klient omezil svoji aktivitu jen na vyslání požadavku.

- **subscription** (odběr dat) – Obr. 4.5. OPC klient nastavuje parametry "UpdateRate", "KeepAlive", "Deadband". OPC server podle parametru "UpdateRate" opakovaně čte data z PLC a ukládá je do paměti CACHE. Dále kontroluje hodnotu právě načtenou z PLC podle parametru "Deadband". Pokud procentuální změna hodnoty je větší než "Deadband" a doba od posledního zaslání dat OPC klientovi je alespoň "UpdateRead", tak OPC server zašle data OPC klientovi voláním callback funkce "OnDataChanged". V tomto případě OPC klient neposílá OPC serveru žádné požadavky na přečtení dat. Pokud parametr "Deadband" je nulový, tak OPC server má posílat data při každé změně hodnoty. Pokud ale změna hodnoty je detekována před uplynutím doby "UpdateRate", tak OPC server nic nepošle (je jedno jestli "Deadband" je nula nebo různý od nuly). Aby se zabránilo situaci, že OPC server delší dobu nic neposílá, protože hodnota se nemění, tak OPC server odešle i nezměněná data, když doba od poslední komunikace s OPC klientem je stejná nebo větší než parametr "KeepAlive".



Obr. 4.5: *Schéma odběru dat OPC (subscription)*

Komunikace "subscription" je nejméně náročná na prostředky počítače i síťové komunikace. Cena ale je, že hodnoty dat nemusí být právě aktuální. Naopak komunikací "synchronous" OPC klient dostane ty nejaktuálnější data z PLC, ale komunikace na jistou dobu blokuje prostředky jak OPC serveru, tak i OPC klienta. V případě, že OPC server nebo klient ještě komunikují s jinými OPC servery či klienty, tak to vede i ke znatelnému omezování provozu někde jinde.

4.4. Program modulu OPC klienta

4.4.1. Připojení k OPC serveru

Program OPC modulu obsahuje typický kód pro připojení k OPC serveru – viz Tab. 4.1.

```
// Nastaveni OPC komunikace podle konfiguračního souboru
xmlConfig.SetOpcClient(this);
// Vytvoreni objektu OPC serveru
System.Net.NetworkCredential mCred = new
System.Net.NetworkCredential();
Opc.ConnectData mConnectData = new Opc.ConnectData(mCred);
Opc.URL opcUrl = new Opc.URL("opcda://" + opcServerComputer + "/" +
      opcServerName);
Opc.Factory fact = new OpcCom.Factory();
opcServer = new Opc.Da.Server(fact, opcUrl);
// Pripojeni k vytvorenemu OPC serveru
opcServer.Connect(opcUrl, mConnectData);
```

Tab. 4.2: *Kód pro připojení k OPC serveru*

4.4.2. Vytvoření skupiny OPC položek

Dále se vytvoří skupina položek, které budou komunikovány s OPC serverem. OPC klient potřebuje ke komunikaci jen dvě OPC položky:

1. Akci *u* kterou nastavuje regulátor v PLC.
2. Výstup *y* simulace z tohoto modelu.

Proměnné odpovídající těmto dvěma položkám musí být vytvořeny v PLC a pak musí být nahrány z PLC do OPC serveru (tzv. upload OPC serveru). Způsob provedení závisí na PLC a na OPC serveru příslušném PLC.

S pomocí testovacího OPC klienta (např. program Matrikon OPC Explorer [12] lze ověřit existenci položek v OPC serveru a zjistit jejich přesná jména. Název OPC serveru a jména OPC položek se zadají v konfiguračním souboru.

OPC klient čte z OPC serveru obě položky, ale zapisuje jen jednu, výstup *y*. Program OPC klienta vytvoří skupinu (Group) dvou OPC položek, nastaví jejich typ asynchronní komunikace typu "subscription", nastaví komunikační intervaly podle konfiguračního souboru. OPC položky jsou také nakonfigurovány v konfiguračního souboru.

4.4.3. Čtení OPC položek

Pro asynchronní OPC komunikaci jsou vytvořeny v OPC klientovi callback funkce "**ReadCompleteCallback**", "**OnDataChange**", "**WriteCompleteCallback**", které jsou zaregistrovány v OPC serveru. Pak OPC klient vždy zahájí komunikaci funkcí "**BeginRead**" a od té doby se nemusí starat o čtení položek z OPC serveru. OPC server posílá položky do OPC klienta sám

a) buď funkcí "**ReadCompleteCallback**" jako odpověď na povel "**BeginRead**",
b) nebo funkcí "**OnDataChange**", kterou OPC server volá,
    i. když je jakákoliv změna hodnoty v jedné ze dvou OPC položek,
    ii. nebo když uplynul interval "KeepAlive",
    iii. anebo když OPC klient použije funkci "Refresh".

4.4.4. Ukládání hodnot do souboru

Hodnoty načtené z OPC serveru funkcí asynchronního čtení "**ReadCompleteCallback**" nebo funkcí odběru položek (item subscription) "**OnDataChange**" mohou být za chodu programu ukládány do textoveho souboru s nazvem RECrrrrmmddhh.DAT. Časové značky a hodnoty položek jsou odděleny tabulátorem, takže je možné je importovat do nějakého vyhodnovacího programu (např. MS Excel). Toto ukládání se umožňuje atributem „Record“ v konfiguračním souboru. Ukládání datového souboru samozřejmě zvyšuje režii programu, proto doporučuji používat tuto funkci jen pro testovací účely.

4.4.5. Zápis vypočteného výstupu *y* do OPC serveru

V modulu OPC klienta je vytvořena funkce typu „SendYEventHandler“, která asynchronně zapisuje vypočtenou výstupní hodnotu *y* do OPC serveru. Adresa funkce je předánan do třídy „DiffEq“ a v okamžiku, kdy nastane konec výpočetního cyklu, je volána uvnitř externího výpočtového cyklu. Časová značka je nastavena v okamžiku provádění funkce. OPC server předává výsledek zápisu *y* funkcí "**WriteCompleteCallback**". Zajímavou vlastností je zápis výstupu *y* do OPC serveru většinou změní hodnotu této položky, takže OPC server vzápětí vrátí hodnoty obou položek *u* a *y*.

4.5. Realizace

Vlastní program byl naprogramován v prostředí Microsoft Visual Studio 2008, v jazyce C#. Programy napsané v jazyce C# jsou obvykle kompilovány do tzv. řízeného kódu (managed code) a potřebují ke svému provozu instalaci podpůrných softwarových prostředků Microsoft .NET Framework, Microsoft Visual C++ 2008 Redistributable a Program OPC klienta potřebuje ke svému spuštění instalaci následujících softwarových produktů:

- Microsoft .NET Framework version 2.0 (x86) [17].
- OPC Core Components Redistributable (x86) 105.1 [10].
- Microsoft Visual C++ 2008 Redistributable Package (x86) [18].

Všechny tyto produkty lze volně stáhnout z internetových stránek poskytovatele a používat podle licence na stránce poskytovatele. Tyto instalační balíčky jsou také uloženy na přiloženém CD.

OPC klient byl vytvořen ve dvou verzích, ve verzi DLL knihovny vhodné pro implementaci do jiného programového prostředí, např. systém COMES, a ve verzi EXE, kterou lze rovnou použít pro simulaci spojitého řízeného systému.

4.5.1. Verze DLL - Integrace do systému COMES

Verze DLL je vhodná pro integraci do systému COMES. Obsahuje veškerou logiku OPC klienta. Při integraci se musí používat jmenný prostor "OpcClientModel". OPC klient včetně simulace se spustí povelem/funkcí "**OpcClient.OpcClientStart()**" a zastaví povelem/funkcí "**OpcClient.OpcClientStop()**". V rámci DLL běží modul simulace a modul OPC klienta každý ve svém odděleném vlákně. Předpokládá se, že process, který bude spouštět DLL verzi, poběží také v jiném vlákně. Pro provoz DLL knihovny je nutné nainstalovat podpůrný software zmíněný na začátku kap. 4.5. Dále je nutné mít umístěny v jednom adresáři následující soubory:

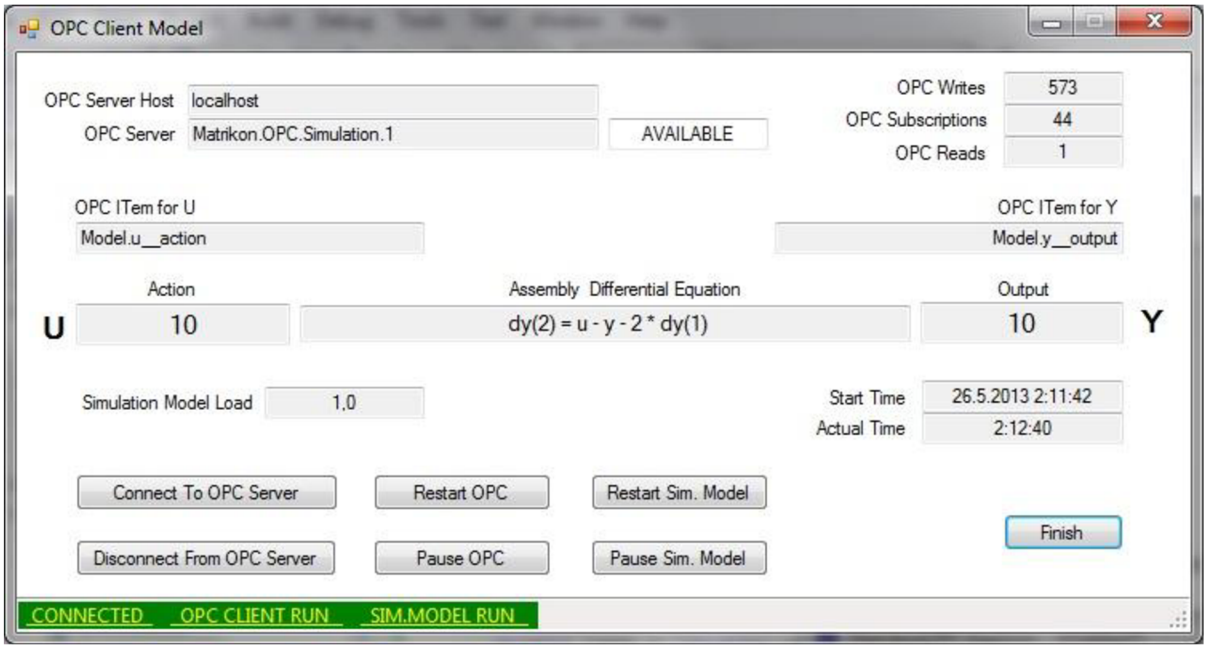
- "OpcClientModelDll.dll"
- "OpcNetApi.dll"
- "OpcNetApi.Com.dll"
- "OpcClientModel.xml"

Soubor "OpcClientModelDll.dll" je hlavní knihovna OPC klienta. Soubor "OpcClientModel.xml" je konfigurační soubor. Parametry v něm uložené se mohou měnit v textovém editoru, např. v Notepad (Poznámkový Blok). Soubory

"OpcNetApi.dll" a "OpcNetApi.Com.dll" jsou součásti softwarového balíku "OPC .NET API 2.00 Redistributables 105.1" [19].

4.5.2. Verze EXE – Spustitelný program

Verze EXE z hlediska logiky obsahuje to samé co DLL verze. Je navíc vybavena jednoduchým uživatelským rozhraním, které umožňuje sledovat práci OPC klienta, a popřípadě zastavovat a opět spouštět modul simulace, modul OPC komunikace, a také se odpojit a znovu připojit k OPC serveru – viz Obr. 4.6.



Obr. 4.6: *Vzhled ovládacího a kontrolního panelu programu "OPC Client Model"*

Tento hotový program lze rovnou použít pro připojení k OPC serveru. Soubory nutné ke spuštění programu jsou:

- "OpcClientModel.exe"
- "OpcNetApi.dll"
- "OpcNetApi.Com.dll"
- "OpcClientModel.xml"

Soubor "OpcClientModel.exe" je vlastní spustitelný program. Ostatní soubory jsou stejné jako u verze DLL.

Spustitelná verze OPC klienta disponuje funkcemi pro pozastavení běhu modulu OPC klienta a jeho znovuspuštění. Přitom se automaticky také zastaví výpočet v modulu simulace řízeného spojitého systému.

OPC Server Host	Jméno (popřípadě CLSID) počítače, na kterém běží OPC server.
OPC Server	Název OPC serveru
OPC Item for U	Název OPC položky v OPC serveru, která se používá pro přenos akční veličiny z řídicího systému (např. PLC) do zde popisovaného OPC klienta.
OPC Item for Y	Název OPC položky v OPC serveru, která se používá pro přenos výstupu simulovaného spojitého systému, zadaného diferenciální rovnici, z OPC klienta do řídicího systému.
pole za jménem OPC serveru	Podává informaci o to, zda OPC server existuje na daném počítači, a zda je v provozu. OPC server je obvykle instalován jako služba operačního systému Windows, a může být deaktivován. Jako služba operačního systému se nemusí OPC server objevovat jako běžící program.
Differential Equation	Zadaná diferenciální rovnice určující chování simulace spojitého řízeného systému. Může být zadána dvěma různými způsoby – kap. 3.4. Popis "Assembly" znamená použití kompilované diferenciální rovnice, lineární i nelineární. Popis "Linear" znamená použití lineární diferenciální rovnice zadanou svými koeficienty. Tvar rovnice je schematicky zobrazen.
U / Action	Hodnota akční veličiny posílaná z OPC serveru do OPC klienta.
Y / Output	Hodnota výstupu simulačního algoritmu.
Simulation Model Load	Zatížení výpočtem. Závisí na tom, jaký je právě průběh výstupní veličíny *y*. Pokud *y* strmě klesá či stoupá, je zatížení vyšší. Pokud gradient *y* je malý, tak zatížení je malé. Zatížení dále závisí na konfigurovaném výpočetním intervalu a kroku metody Runge-Kutta, a na požadované přesnosti výpočtu.
OPC Writes	Počet OPC zápisů simulovaného výstupu *y* z OPC klienta do OPC serveru.
OPC Subscriptions	Počet OPC čtení akční veličiny *u* z OPC serveru do OPC klienta metodou odběru OPC položek.
OPC Reads	Počet OPC čtení akční veličiny *u* z OPC serveru do OPC klienta metodou asynchronního čtení OPC položek.

Tab. 4.3: *Přehled stavů panelu spustitelné verze OPC klienta.*

Dále existují funkce, která odpojí modul OPC klienta od OPC serveru, popřípadě ho znovu připojí. Při odpojení se samozřejmě zastaví OPC komunikace a výpočet simulace. Při znovupřipojování se OPC klient opět připojuje k OPC serveru uvedeném

v konfiguračním souboru. Tyto funkce byly zavedeny hlavně kvůli ztrátě komunikace z vnějších důvodů. Je lepší řešit tento problém manuálně odpojením a novým připojením.

Stavy zobrazené na panelu programu jsou vysvětleny v tabulce Tab. 4.3.

Stavová lišta okna OPC klienta zobrazuje barevně a popisně stavy tří základních procesů - Tab. 4.4.

Stav připojení k OPC serveru.	CONNECTED (zeleně) DISCONNECTED (bezbarvě)
Stav chodu OPC komunikace (čtení a zápis položek).	RUN (zeleně) HALTED (červeně nebo bezbarvě)
Stav chodu simulačního výpočtu podle zadané diferenciální rovnice.	RUN (zeleně) HALTED (červeně nebo bezbarvě)

Tab. 4.4: *Stavy procesů OPC klienta na stavové liště programu*

4.6. Konfigurace OPC komunikace

Parametry OPC komunikace jsou při startu programu OPC klienta také načteny z konfiguračního souboru "**OpcClientModel.xml**" – viz .

```
<OPC_Configuration UpdateRate="10" KeepAlive="5000" MaxAge="30000"
Deadband="0" Record="true">
    <Computer>localhost</Computer>
    <OpcServer>Matrikon.OPC.Simulation.1</OpcServer>
    <OpcItemGroup Name="OPC_Client_Model">
      <ReadAction ModelName="U" ItemName="Model.u__action"
Limitation="false" Min="-100" Max="100" Description="Control System
Action Value" />
      <WriteOutput ModelName="Y" ItemName="Model.y__output"
Limitation="false" Min="-100" Max="100" Description="Model Calculated
Output" />
    </OpcItemGroup>
  </OPC_Configuration>
```

Tab. 4.5: *Parametry OPC komunikace v konfiguračním souboru.*

Atribut "**UpdateRate**" je požadavek OPC klienta na OPC server na minimální obnovovací interval hodnot pro OPC klienta. Minimálně za tento čas musí OPC server obnovit hodnotu z řídicího systému. Atribut "**KeepAlive**" je požadavek OPC klienta na OPC server na maximální obnovovací interval hodnot pro OPC klienta. I když OPC server nestihne dostat hodnotu z řídicího systému anebo se hodnota nezmění, přesto OPC server musí hodnotu položky poslat do OPC klienta v čase "**KeepAlive**". Pokud OPC server není schopen dostat hodnotu z řídicího systému za čas "**MaxAge**", nastaví kvalitu takové OPC položky jako špatnou (Bad). Atributy "**UpdateRate**", "**KeepAlive**", "**MaxAge**" mají hodnoty v milisekundách.

Atribut "**Deadband**" je hodnota v procentech, která umožňuje omezit posílání hodnot položek z OPC Serveru OPC klientům. Pokud je nula, každá změna hodnoty je poslána OPC klientům. Pokud je atribut "**Deadband**" různý od nuly, tak OPC server pošle hodnotu OPC položky jen když změna hodnoty položky je větší než zadané procento. V tomto případě OPC server musí mít konfiguraci také pro dolní a horní rozsah položky.

Atribut "**Record**" umožňuje zapisovat hodnoty načtené z OPC serveru do souboru. Vždy se čte jak akční veličina, tak i zpětně nasimulovaný výstup. Soubory mají jméno RECrrrrmmddhh.DAT. Pro každou hodinu je generován jeden soubor. Položky jsou odděleny tabulátorem. Údaje o datu a čase se berou z časové značky OPC položky, nezávisí na čase, v kterém se do souboru zapisuje. Soubor lze otevřít např. v programu Excel, a následně vyhodnotit zapsaná data. Doporučuji povolit zápis dat do soboru atributem "**Record**" jen pro testovací účely.

Parametr "**<Computer>**" obsahuje název počítače, na kterém běží OPC server. Pokud OPC server a OPC klient běží na stejném počítači, zadává se název "localhost". Pokud OPC server běží na jiném počítači než OPC klient, komunikuje se protokolem DCOM. Většinou je nutné zadat místo jména počítače CLSID (Class ID) vzdáleného počítače, a je nutno splnit provést nastavení uživatelských účtů a služeb operačního systému Windows na obou počítačích tak, aby fungovalo vzdálené volání procediry (RPC) – viz dokument " *Using OPC via DCOM with Microsoft Windows*" [11].

Parametr "**<OpcServer>**" obsahuje název OPC serveru.

Skupina položek "**OpcItemGroup**" obsahuje konfiguraci dvou základních OPC položek používaných OPC klientem. Atribut skupiny "**Name**" může obsahovat libovolný text.

Parametr "**<ReadAction>**" popisuje OPC položku pro přenos akční veličíny z OPC serveru do OPC klienta. Atribut **ModelName="U"** by se neměl nikdy modifikovat. Další atribut "**ItemName**" obsahuje název této OPC položky v OPC serveru. Atribut "**Limitation**" umožňuje omezit tuto hodnotu při čtení z OPC komunikace na rozsah daný

atributy "**Min**" a "**Max**". Atribut "**Description**" je popis OPC položky a může obsahovat libovolný text.

Parametr "**<WriteOutput>**" popisuje OPC položku pro přenos vypočtené hodnoty odezvy simulovaného systému z OPC klienta do OPC serveru. Atribut **ModelName="Y"** by se neměl nikdy modifikovat. Ostatní atributy "**ItemName**", "**Min**", "**Max**", "**Description**" mají stejný význam jako u parametru "**<ReadAction>**". Atribut "**Limitation**" umožňuje omezit vypočtenou hodnotu *y* při odesílání OPC komunikací. Během výpočtu hodnota *y* nemá žádné omezení.

Pro zjištění jména OPC serveru, identifikaci vzdáleného počítače a nalezení přesných jmen OPC položek v OPC serveru doporučuji použít některého z volně použitelných OPC klientů, např. Matrikon OPC Explorer [12].

4.6.1. Oprava konfiguračního souboru

Konfigurační XML soubor "OpcClientModel.xml" je nutné editovat ručně. Může se stát, že syntaxe souboru byla porušena, popřípadě soubor byl smazán, nebo z nějakých jiných důvodů se konfigurace nenačte. V tom případě soubor přejmenujte nebo smažte, a spusťte program. Pokud program nenajde konfigurační soubor, vygeneruje nový XML soubor se správnou XML syntaxí a implicitním nastavením. Bude ale nutné do něj doplnit ručně jméno OPC serveru a jména OPC položek, nastavit správnou diferenciální rovnici, a popřípadě změnit či korigovat i jiné parametry.

4.6.2. Výstupní soubory

Program OPC klienta může generovat několik různých souborů. Viz Tab. 3.1.

OpcClientError.Log	Je generován v případě odchycení nějaké nespecifické chyby programu.
Debug.Log	Obsahuje ladicí informace o výpočtu simulace v případě, že je to povoleno atributem "**Debug="true"**".
RECrrrrmmddhh.DAT	Soubor se záznamem OPC položek pro akční vstup *u* a vypočítaný výstup *y*. Jeden soubor obsahuje data za 1 hodinu. Generování souboru je povoleno atributem "**Record="true"**".
OpcClientModel.xml	Je vygenerován tento implicitní konfigurační soubor v případě, že neexistuje.

Tab. 4.6: *Přehled výstupních souborů*

4.7. Testování

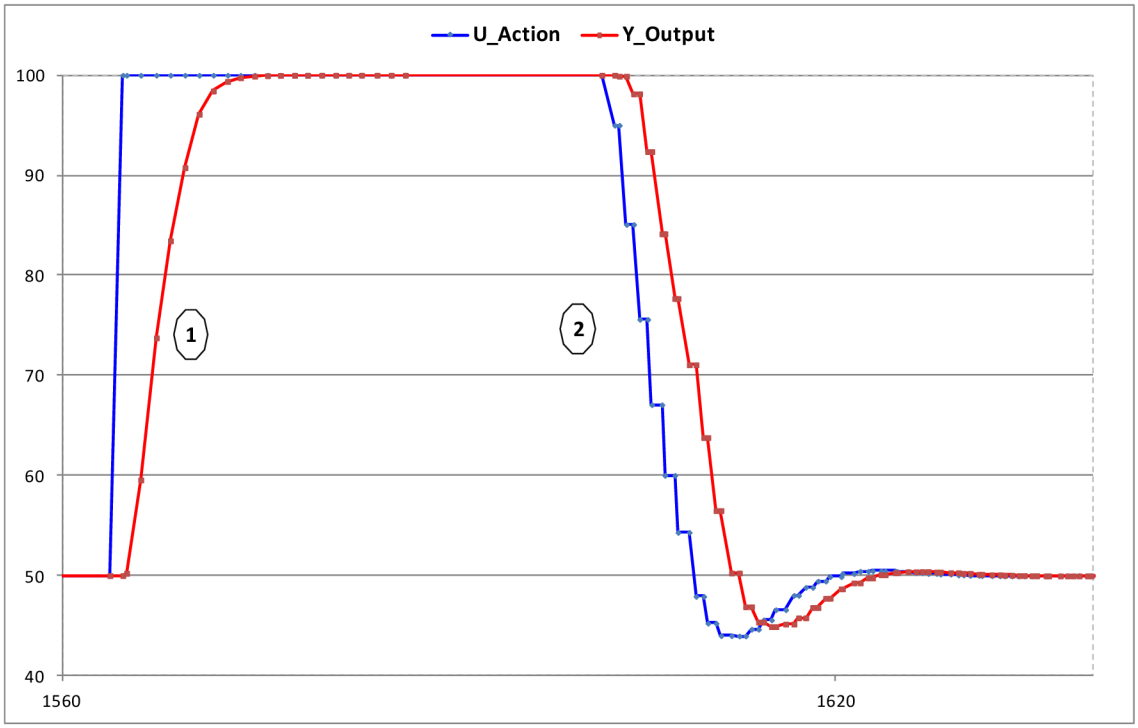
Program OPC klienta byl testován s pomocí testovacího OPC serveru "Matrikon OPC Server for Simulation" [12]. V OPC serveru byly nakonfigurovány dvě potřebné OPC položky. Akční veličina v nich byla měněna buď manuálně (s pomoci Matrikon OPC Exploreru), popřípadě jednoduchým proporcionálním regulátorem, který byl za tím účelem speciálně naprogramován jako separátní OPC klient. Výsledek testu pro rovnici

$$\frac{d^2y(t)}{dt^2} = u(t) - 2 \cdot \frac{dy(t)}{dt} - y(t) \qquad (16)$$

je na Obr. 4.7 a Obr. 4.7

Obr. 4.8. Akční hodnota *u* je modrou barvou, výstup *y* vypočtení OPC kloientem je červenou barvou. Na počátku jsou obě veličiny ustálené na hodnotě 50.
Následovaly 4 různé akce:

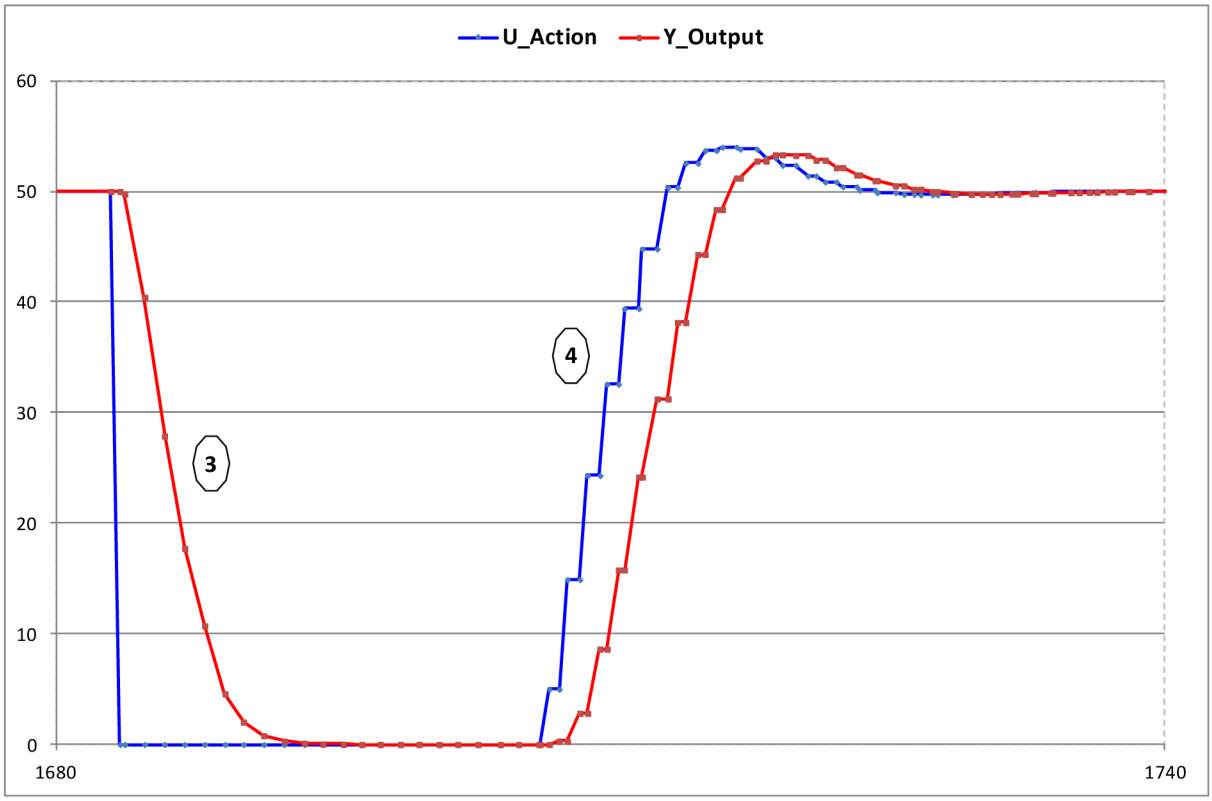
1) Obr. 4.7 - Ručně byla nastavena akční veličina u na hodnotu 100. Modul simulace dopočítal výstupní hodnotu také na hodnotu 100, protože zadaná diferenciální rovnice popisuje stabilní systém.



Obr. 4.7: *Výsledek testu OPC klienta – 1.část*

2) Obr. 4.7 - Po ustálení byl spuštěn jednoduchý OPC proporcionální regulátor s konstantou k=0,1. Ten zreguloval simulovaný systém s jedním větším překmitem

dolů na hodnotu 50 . Skoky na čáře akční veličiny jsou způsobeny opakovanou OPC komunikací, když dva OPC klienti komunikují tytéž položky. Po ustálení byl regulátor zastaven.



Obr. 4.8: *Výsledek testu OPC klienta – 2.část*

3) Obr. 4.8 - Ručně byla nastavena akční veličina u na hodnotu 0. Modul simulace dopočítal výstupní hodnotu také na hodnotu 0.

4) Obr. 4.8 - Po ustálení byl opět spuštěn jednoduchý OPC proporcionální regulátor s konstantou k=0,1. Ten zreguloval simulovaný systém s jedním větším překmitem na hodnotu 50.

Během testů se ukázalo, že použitý simulační OPC server není schopen posílat požadované OPC položky rychleji než jednou za sekundu bez ohledu na jakékoliv nastavení. Přitom bylo možné, aby OPC klient zapisoval do OPC serveru až 10x za sekundu. Podobné chování se dá očekávat i od jiných OPC serverů. Všeobecně OPC nemá pověst nejrychlejší komunikace.
V případě, že OPC server a OPC klient jsou na různých počítačích, je nutné počítat i s několika sekundovými prodlevami. Je jisté, že jedním povelem se dá poslat rychle velké

množství dat, ale opakovaná cyklická komunikace bude mít interval cyklu sotva pod 1 sekundu.

Proto se při výběru systému musí s tímto počítat. Systém vhodný pro simulaci tímto způsobem by měl mít znatelnou odezvu alespoň 10 sekund.

5. Závěr

Během návrhu OPC klienta pro modelování regulace spojitého systému byl navržen a odladěn modul simulátoru lineárního spojitého systému na základě zadané diferenciální rovnice. Simulátor používá k simulaci numerickou metodu Runge-Kutta 4. řádu a využívá k výpočtu hodnot požadované přesnosti metodu poloviční kroku. Vnitřní architektura simulátoru byla navržena tak, aby se docílilo jak přesného výpočtu v reálném čase, tak i dosáhlo dostatečné rychlosti výpočetního algoritmu. Parametrizace simulátoru umožňuje spouštět simulátor efektivně i na počítačích, které nejsou příliš rychlé.

Další část vytvořeného programu je modul OPC klienta. Komunikační modul byl odladěn s pomocí simulátoru OPC serveru. Během ladění a testování programu se ukázalo, že OPC komunikace má jisté meze, co se týče rychlejšího cyklického opakování komunikace. K tomu se musí přihlížet při volbě spojitého systému pro simulaci.

Výsledný program má dvě formy. Jedna ve formě DLL knihovny vhodná pro integraci do jiného systému. Druhá forma představuje spustitelný program, spustitelný za podmínek popsaných v kapitole 4.5.

Literatura

[1] COMPAS AUTOMATIZACE. *Výrobní informační systém COMES verze 3*: *COMES CCI – Příručka pro nastavení*. Ždár nad Sázavou: COMPAS, 2010

[2] FAJMON, Břetislav a Irena RŮŽIČKOVÁ. *Matematika 3*. Brno, 2005. Skriptum. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav matematiky.

[3] JURA, Pavel. *Signály a systémy: Část 1: Spojité signály*. 2.opr.vyd. Brno, srpen 2010. Skriptum. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií.

[4] JURA, Pavel. *Signály a systémy: Část 2: Spojité systémy*. 2.opr.vyd. Brno, srpen 2010. Skriptum. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií.

[5] OPC FOUNDATION: *OPC Overview*. OPC FOUNDATION, 1998. Dostupné z: http://www.opcfoundation.org/

[6] OPC FOUNDATION: *OPC Data Access Custom Interface Specification version 3.00*. OPC FOUNDATION, 1998, vydáno 4.3.2003
Dostupné z: http://www.opcfoundation.org/

[7] SHARP, John. *Microsoft Visual C# 2008: Step by Step.* Redmond (Washington): Microsoft Press, 2008

[8] WATSON, Ben. *C# 4.0 – řešení praktických programátorských úloh.* Přeložil Jan POKORNÝ. Brno: ZONER Press, 2010

[9] Wikipedia. http://en.wikipedia.org/

[10] OPC FOUNDATION: "*OPC Core Components Redistributable (x86) 105.1*". OPC FOUNDATION, 2011, vydáno 18.11.2011. Dostupné z: http://www.opcfoundation.org/Downloads.aspx?CM=1&CN=KEY&CI=286

[11] DEIRETSBACHER, K.H., J.LUTH, R.MODY a K.T.HAUS. *Using OPC via DCOM with Microsoft Windows SP Service Pack 2 Version 1.10.* OPC FOUNDATION, 2006. Dostupné z: http://www.opcfoundation.org/

[12] MatrikonOPC, Canada. *Matrikon OPC Explorer. Matrikon OPC Server for Simulation.*
http://www.matrikonopc.com/products/opc-desktop-tools/opc-explorer.aspx

[13] Codeguru, stránky pro programování. http://www.codeguru.com

[14] CodeProject, stránky pro programování. http://www.codeproject.com

[15] Mesta Automation, Blog about WPF, C# and PLC software development. http://www.mesta-automation.com

[16] BĚHÁLEK Martin. *Komponenty COM a distribuované aplikace*. VŠB-TU Ostrava 2007. http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text/ch10.html

[17] Microsoft. *Microsoft .NET Framework 2.0* . http://www.microsoft.com/en-us/download/details.aspx?id=16614

[18] Microsoft. *Microsoft Visual C++ 2008 Redistributable Package (x86).* http://www.microsoft.com/en-us/download/details.aspx?id=29

[19] OPC Foundation. *OPC .NET API 2.00 Redistributables 105.1.* http://www.opcfoundation.org/Downloads.aspx?CM=1&CN=KEY&CI=285