VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH DESKTOPOVÝCH APLIKACÍ S VYUŽITÍM KNIHOVNY PRO WORKFLOW

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

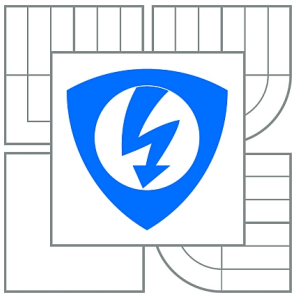
AUTOR PRÁCE
AUTHOR

KRYŠTOF ZEMAN

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH DESKTOPOVÝCH APLIKACÍ S VYUŽITÍM KNIHOVNY PRO WORKFLOW

DESIGN OF DESKTOP APPLICATIONS USING THE WORKFLOW LIBRARY

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

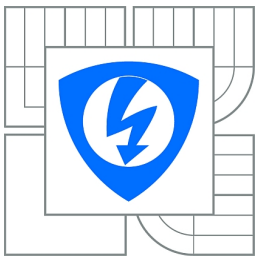
AUTOR PRÁCE — KRYŠTOF ZEMAN

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. IVO LATTENBERG, Ph.D.

SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Kryštof Zeman	***ID:***	134671
Ročník:	3	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

Návrh desktopových aplikací s využitím knihovny pro workflow

POKYNY PRO VYPRACOVÁNÍ:

S využitím programovacího jazyka C# a technologie Windows Workflow Foundation 4.0 pro platformu .NET navrhněte desktopovou aplikaci - hru "Piškvorky" pro dva hráče. Aplikace by měla fungovat peer-to-peer mezi dvěma počítači. K definici stavového chování aplikace využijte právě Workflow Foundation 4.0. Aplikaci řádně okomentujte a pokuste se ji rozdělit do jednotlivých etap, ze kterých bude zřejmý postup při návrhu.

DOPORUČENÁ LITERATURA:

[1] SHARP, J. Microsoft Visual C# 2010, Nakladatelství Computer Press, a.s. 2010, 696 s., ISBN 978-80-251-3147-3
[2] PETZOLD, CH. Mistrovství ve Windows Presentation Foundation, Nakladatelství Computer Press, a.s. 2008, 928 s., ISBN 978-80-251-2141-2
[3] WATSON, B. C# 4.0 - řešení praktických programátorských úloh, Zoner press, 2010, 656 s., ISBN 978-80-7413-094-6

Termín zadání:	11.2.2013	***Termín odevzdání:***	5.6.2013

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.
Konzultanti bakalářské práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

V teoretické části se práce obecně zaměřuje na historii a vývoj knihovny workflow, stručně popisuje její nejdůležitější vlastnosti a možnosti, čímž poskytuje čtenáři základy, potřebné k porozumění a orientaci v dané tematice. V praktické části je pak podrobně rozebrán jeden ukázkový příklad, ukazující možnosti, flexibilitu a výhody spojení knihoven WF a WPF. Především práce v příkladě využívá nejrozšířenější model workflow a ukazuje čtenářům jak předávat informace mezi workflow a WPF.

KLÍČOVÁ SLOVA

Windows Workflow Foundation, Windows Presentation Foundation, Windows Communication Foundation, C#, .NET 4.0, WPF

ABSTRACT

The theoretical part of the thesis generally focuses on history and development of workflow library, briefly describing most important features and options, which gives the reader basics needed to understand and orientation in the topic. In practical part of the thesis is detailly analized prime example, which is showing options, flexibility and advantages of connecting WF and WPF libraries. Primarily the thesis uses the most widely used model and shows how to pass information between workflow and WPF.

KEYWORDS

Windows Workflow Foundation, Windows Presentation Foundation, Windows Communication Foundation, C#, .NET 4.0, WPF

ZEMAN, Kryštof *Návrh desktopových aplikací s využitím knihovny pro workflow*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2013. 41 s. Vedoucí práce byl doc. Ing. Ivo Lattenberg, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Návrh desktopových aplikací s využitím knihovny pro workflow“ 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/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Ivo Lattenbergovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

V dnešní době se programovací jazyky snaží co nejvíce přizpůsobit problémům, usnadnit práci programátorům a obsáhnout co největší množství příkazů a hlavně řěšení různých problematik. Windows Workflow Foundation (WF) je programovací model, sada nástrojů a runtime prostředí, které umožňuje psát deklarativní workflow tvořící model průběhu vašich programů na platformě Windows. Verze WF o které budu pojednávat v této práci byla vydána jako součást .NET Framework verze 4 a je označována jako WF4. Workflow Foundation je díky možnosti modelování workflow pomocí visuálního designeru velmi přehledné a díky možnostem užití jak předdefinovaných, tak uživatelem vytvořených aktivit velmi lehce implementovatelné na jakýkoliv problém. Díky možnosti tvoření vlastních aktivit se nabízí uplatnění workflow například ve velkých týmech, kde se programátoři zaměří na vytvoření zmíněných aktivit, a manažeři z nich následně sestaví workflow dle jejich potřeb. V této práci se na příkladu pokusím ukázat, jak v jedné aplikaci spojit výhody WF a WPF.

1 ŘEŠENÍ STUDENTSKÉ PRÁCE

1.1 Historie Workflow Foundation

- Workflow Foundation se poprvé objevilo jako část .NET Frameworku verze 3.0, a využívalo hlavně jmenné prostory: System.Workflow.Activities, System.-Workflow.ComponentModel, a System.Workflow.Runtime. Ve verzi 3 byly Workflow tvořeny za použití Sekvenčního modelu (Sequential model), ve kterém jsou aktivity vykonávány v pořadí, v jakém byly sestaveny, a to vždy tak, že ukončení jedné aktivity vede ke spuštění aktivity následující, nebo Stavového modelu (State machine model) ve kterém jsou aktivity spouštěny jako reakce na vnější události.
- V .NET 3.5, byla představena nová aktivita s názvem ReceiveActivity, díky které mohou Workflow reagovat na příchozí Windows Communication Foundation zprávy. Nové funkce Workflow ve verzi 3.5 používají jmenný prostor System.ServiceModel.
- V .NET 4.0,byla Windows Workflow Foundation široce modernizována o nové funkce jako Data Contract Resolver, Flowchart, a ostatní flow control aktivity. Workflow ve verzi .NET 4 používá jmenný prostor System.Activities. Nejvýznamější změnou je, že zde už není Workflow Runtime objekt jako v předchozích verzích. Workflow se namísto něj spouštějí přímo za použití WorkflowApplication nebo WorkflowInvoker. Aktivity vytvořené v přechozích verzích .NET Frameworku se dají spouštět v .NET 4 workflow za použití Interop aktivity.
- WF4 následovala aktualizace .NET 4.0 Platform Update1, která hlavně přinesla možnost využívat StateMachineWorkflow v designeru a nové třídy: StateMachine, State, FinalState, Transition, StateMachineQuery, StateMachine-StateRecord.
- Poslední dosud vydanou verzí je WF4.5, která je součásti .NET Frameworku 4.5. Tato verze nově podporuje využívání jak C# tak VB výrazů. Dále přináší Workflow Identity & Versioning, který pomáhá v kompatibilitě mezi jednotlivými verzemi.

1.2 Workflow knihovna

Workflow knihovna umožňuje:

- Plánovat a spouštět workflow a aktivity. Workflow může být spuštěno třemi způsoby:
  1. Pomocí WorkflowInvokeru, který spustí workflow na stejném vlákně, ze kterého je volán(to znamená, že není vytvořeno nové vlákno pro workflow). V tomto případě proces, který workflow zavolal počká, dokud workflow neskončí, a následně pokračuje. Hodí se pro malé workflow, kde není potřeba rozsáhlých možností WorkflowApplication.
  2. Pomocí WorkflowApplication, která spustí workflow na novém vlákně (aplikace tedy není pozastavena po dobu, kdy je je vykonáváno workflow). Vhodné pro větší worfklow, kde může být potřeba například pozastavit workflow pomocí bookmarků či jinak ovládat průběh.
  3. Pomocí WorkflowServiceHost, který spustí workflow jako WCF Službu. Vzniklá workflow služba následně typicky získává vstupní data ze sítě pro aktivity, které obsahuje.
- Určovat v jakém pořadí se budou vykonávat jednotlivé aktivity. Model vykonávání Workflow může být sestaven pomocí aktivit jako Flowchart, If, Sequence, Pick a Parallel vizuálně přes designer. Ve verzi 4 a výše lze nově používat StateMachine aktivity, přinášející nový způsob tvorby modelu.
- Uchovávání workflow. Při uchovávání workflow se data z workflow uloží na přechodné medium (například SQL Server) a uvolní workflow z paměti. Workflow může být znovunačteno po určité době nebo pokud obdrží zprávu. Odstraněním nečinných workflow z paměti můžeme velmi navýšit počet aktivních workflow, které systém může ovládat, a tedy zvýšit škálovatelnost.
- Spravovat data pro probíhající aktivity. Data spotřebovávají aktivity používáním argumentů a proměnných, které jsou potřebné pro běh aplikace. Používání argumentů a proměnných k ukládání dat pro aktivity znamená, že má aplikace kdykoliv ve svém průběhu přístup ke konečnému stavu aktivity pro případ, že by jej bylo nutné zachovat. Aplikace může též v průběhu předávat příchozí zprávy a data konkrétním workflow instancím v případě, že běží několik workflow současně.
- Využívat vestavěný „tracking provider“, který zaznamenává vestavěné Workflow události (jako začátek, dokončení či selhání aktivity), nebo vlastní aktivity (sledování určitých dat vlastní aktivity). Základní „tracking provider“ v .NET Frameworku verze 4 zaznamenává sledované události do originálního záznamového souboru Windows, ale uživatelem definovaný „tracking provider“ může být naprogrmován k ukládání sledovaných událostí i do jiných uložišť.

- Poskytuje budoucí rozšiřitelnost formou Workflow Extensions(=rozšíření). Rozšíření jsou uživatelem definované objekty, přidané do workflow prostředí, které poskytují uživatelem definované funkce, jako rozšířené možnosti komunikace s hlavním procesem nebo specifické funkce sledování procesů.
- Využívat visualní ladění projektu pomocí workflow designeru. Workflow mohou být spouštěna a laděna ve vývojovém prostředí za použití stejné zarážky a krokování jako v ladícím kódu.

1.3 Druhy Workflow modelů

Různé druhy aplikací potřebují různý přístup k jejich řešení, a tím pádem také různé modely vykonávání. Některé programy potřebují být co nejrychleji vykonány, zatímco ostatní mohou běžet týdny nebo měsíce. Jiné zas potřebují být vykonány ve velmi přesných krocích s pokud možno žádnou odchylkou od naplánovaného průběhu. Další potřebují být více flexibilní, aby mohli reagovat na akce uživatelů a podle nich měnit svůj průběh. Díky Windows Workflow Foundation (WF) tohoto můžeme dosáhnout pomocí vytvoření modelů, složených z aktivit, které jsou použitím Activity typu(přesněji System.Workflow.ComponentModel.Activity). Pro vytvoření workflow použijeme jednu či více aktivit z WF knihovny, vetšinou ve spojení s námi definovanými aktivitami, a spojíme je v určitém pořadí, aby výsledný model vyhovoval našemu řešení. Aktivity můžeme rozdělit na dvě základní skupiny: První jsou jednoduché aktivity(Simple activities, často označované jako též „leaf“ activities), které dokáží spouštět libovolný kód, zapisovat data do databází, posílat emaily nebo spouštět sady pravidel. Druhým typem jsou složené aktivity(Composite activities), které nemají konkrétní funkci po spuštění, ale spíše spouštějí jejich potomky v předem známém pořadí, čímž ovládají tok ve workflow. Jednoduchým příkladem složené aktivity může být například aktivita, která spouští dva potomky: jeden potomek pošle email a druhý zapíše do systému data o odeslaní. Ve většině případů tedy používáme složené aktivity.

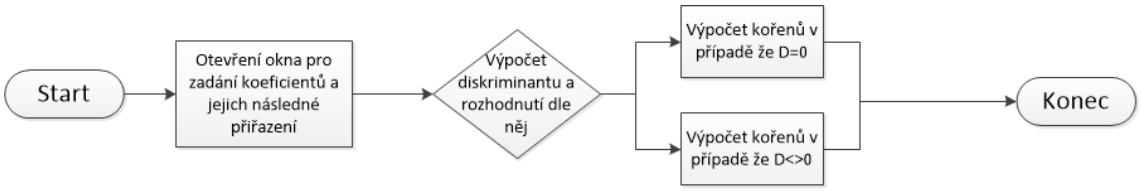
Samotné WF je vydáváno s velkým množstvím složených aktivit, které mohou být spouštěny přímo pomocí WorkflowRuntime. Je také vydáváno se třemi speciálními aktivitami, které jsou často označovány jako „Kořenové“ modely. Jsou to SequentialWorkflowActivity,FlowChartWorkflowActivity a StateMachineWorkflowActivity. Důležité je ale vědět že i přes jejich časté označování jako „Kořenové“ modely se k nim WorkflowRuntime chová naprosto stejně jako k ostatním aktivitám.

1.4 Vzorový příklad

V této práci bych Vás rád seznámil s praktickým příkladem využití workflow verze 4.0. Bohužel se mi nepodařilo zprovoznit aplikaci piškvorek za pomocí workflo a tak nám jako ukázkový model poslouží poměrně jednoduchá aplikace pro výpočet kvadratických rovnic. Tato aplikace bude používat několik Windows Presentation oken, které budou komunikovat s workflow modelem pomocí bookmarků.

1.4.1 Návrh

Vzniku každé aplikace by měl předcházet její návrh. Prvním krokem návrhu by mělo být ujasnění si našich cílů,tedy soupis věcí,které od aplikace požadujeme. Poté se přesuneme k vlastnímu návrhu průběhu aplikace. V našem případě od aplikace požadujeme aby byla schopna počítat kořeny kvadratických rovnic, otevírat několik oken a také by bylo vhodné aby aplikace dobře vypadala, o což se postará dříve zmíněná WPF knihovna. Nic dalšího už od aplikace nepotřebujeme a tak se přesuneme k návrhu jejího průběhu. Nejlepším způsobem návrhu stále zůstává metoda, kdy si vezmeme papír, tužku a nakreslíme jakými stavy bude aplikace procházet. Tyto stavy by měli být řazeny za sebou v co nejlogičtějším sledu, aby aplikace byla co nejefektivnější. V našem případě by průběh mohl vypadat zhruba nějak jako na obrázku **??**.



Obr. 1.1: Návrh průběhu

Jak vidíte, schema průběhu naší aplikace není nijak velké, avšak i v takto malých aplikacích je dobré si jej nakreslit hned z několika důvodů:

- Při vytváření modelu si ujasníme jak bude aplikace fungovat, což je vždy důležité udělat před začátkem psaní kódu.
- Pokud se při psaní kódu objeví problémy v průběhu, pomocí tohoto schematu se lépe zorientujeme a snadněji identifikujeme chybu.

Dále je vhodné si do modelu, popřípadě k němu dopsat stručný průběh aplikace v jednotlivých stavech. V našem modelu to bude vypadat zhruba takto:

1. Ve stavu Start se spustí aplikace a zobrazí první okno ve kterém si uživatel vybere mezi spuštěním okna pro výpočet, nebo zavřením aplikace(v reálném

případě by to nebylo nutné, ale pro ukázku předávání informací se to velmi hodí).

2. Ve stavu druhém, do kterého se aplikace přesune po stisku tlačítka ve stavu prvním, aplikace otevře okno pro zadání koeficientů a následně bude čekat na jejich zadání. Po zadání koeficientů uživatel stiskne tlačítko a aplikace postoupí do dalšího stavu.
3. Ve třetím stavu aplikace pouze vypočítá diskriminant, a dle něj rozhodne zda bude následovat stav pro výpočet kořenů v případě $D = 0$ a nebo $D \neq 0$. Po tomto rozhodnutí se přesune do jednoho z dalších stavů.
4. Ve stavech do kterých se přesune z třetího stavu, se vypočítají kořeny kvadratické rovnice a následně se aplikace přesune do posledního stavu.
5. Ve stavu Konec aplikace otevře okno s výsledky, kam zapíše hodnoty získané z předchozích stavů. Tím je celý průběh aplikace ukončen a může tedy začít znovu.

Po dokončení všech těchto kroků máme připraveno vše, co je nutné k vytvoření aplikace. Můžeme se tedy přesunout k samotné tvorbě.

1.4.2 Vytvoření aplikace

Vytvoření nového Solution a projektu

1. Otevřete Visual Studio 2012.
2. Z horní lišty Visual Studia zvolíme File⇒ New ⇒ Project....
3. V nově otevřeném okně zvolte v jeho levé části položku **Other Project Types** a vyberte z ní **VisualStudioSolution**. Do kolonky **Name** napište Workflow a klikněte na **OK**.
4. V okně **Solution Explorer** pravým tlačítkem kliněte na položku: Solution 'Workflow' (0 projects) a z vyskakovacího okna zvolte: Add⇒ New Project.... Otevře se vám znovu okno s výběrem projektů. Teď ovšem rozklikněte položku **Visual C#**. Ujistěte se, že v horní části okna máte zvolený **.NET Framework 4 Platform Update 1**, vyberte **WPF Application**, do položky **Name** napište Wpf_Ukazka a stiskněte **OK**. Nakonec stiskněte CTRL+SHIFT+B k buildnutí a uložení změn.

Tím jsme vytvořili Solution a v něm WPF Aplikaci s kterou budeme dále pracovat.

Vytvoření hlavního okna

Pokud jste udělali správně vše z předchozího kroku, měli by jste v okně **Solution Exploreru** vidět soubor **MainWindow.xaml** a po jeho rozkliknutí i **Ma-**

inWindow.xaml.cs(lze též otevřít pomocí kliknutí pravým tlačítkem na MainWindow.xaml a následně na **View Code**). Přistoupíme tedy k jejich editaci:

1. Otevřete MainWindow.xaml. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **MainWindow**. V **Properties** okně **Labelu** mu do okna **Content** vložte *Dobrý den, pokud chcete vypočítat kořeny kvadratické rovnice, klikněte zde:*. Editovat vlastnosti prvků lze i přímo v XAML kódu, který by jste měli vidět hned pod oknem designeru. Doporučuji však editovat výše uvedeným způsobem, neboť je to rychlejší a pro méně zkušené uživatele i přehlednější.
2. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **MainWindow**. V **Properties** okně **Labelu** mu do okna **Content** vložte *Pokud chcete aplikaci zavřít klikněte zde:*.
3. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Button** a vložte jej do okna **MainWindow**. V **Properties** okně **Buttonu** mu do okna **Content** vložte *Otevřít okno pro zadání koeficientů*. V okně **Properties** klikněte na ikonku **blesku**, čímž se přepnete do **events** části okna. Dvojklikem na okénko vedle textu **Click** vytvoříte on_click_event tlačítka. Pokud vás Visual Studio přepne do **MainWindow.xaml.cs** vraťte se zpět do **MainWindow.xaml**.
4. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Button** a vložte jej do okna **MainWindow**. V **Properties** okně **Buttonu** mu do okna **Content** vložte *Zavřít*. V okně **Properties** klikněte na ikonku **blesku**, čímž se přepnete do **events** části okna. Dvojklikem na okénko vedle textu **Click** vytvoříte on_click_event tlačítka. Pokud jste vše udělali správně, po přepnutí zpět do MainWindow.xaml by vaše okno mělo vypadat podobně jako na obr.1.2.
5. Nyní se přepneme do okna **MainWindow.xaml.cs** a kód, který obsahuje nahradíme následujícím kódem a poté stiskněte CTRL+SHIFT+B k buildnutí a uložení změn.

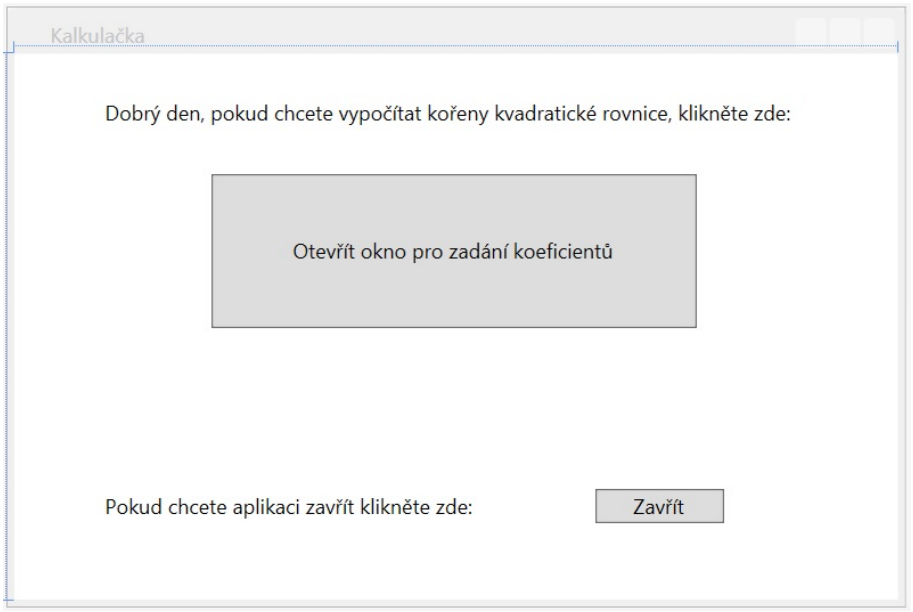
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Threading;
using System.Activities;

namespace Wpf_Ukazka
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        //po zapnutí aplikace se spustí tento stav a otevře
        public MainWindow()
        {
            InitializeComponent();
        }

        //při kliknutí na tlačítko 1 spustí WindowState1
        public void Button_Click_1(object sender, RoutedEventArgs e)
        {
            worker.getInstance().spust();

        }
        //po kliknutí na tlačitko 2 ukončí aplikaci
        private void Button_Click_2(object sender, RoutedEventArgs e)
        {
            Application.Current.Shutdown();
        }

    }

    //třída starající se o zapnutí a běh workflow
    public class worker
    {
        public string Koreny;
        public string Numberx1;
        public string Numberx2;
        public string Numberxi1;
        public string Numberxi2;

        //singleton instance
        private static worker instance;

        //singleton Getter
```

```
public static worker getInstance()
{
    if (worker.instance == null)
    {
        worker.instance = new worker();
    }
    return worker.instance;
}

//vytvoří novou proměnnou wfApp
WorkflowApplication wfApp;
//votvoří novou proměnnou thread
public Thread childThread;

//po zavolání metody spustí workflow aplikaci
public void spust()
{
    //vytvoří novou workflow aplikaci,kterou namapuje z
        ModeluPrubehu
    wfApp = new WorkflowApplication(new ModelPrubehu());

    AutoResetEvent syncEvent = new AutoResetEvent(false);
    AutoResetEvent idleEvent = new AutoResetEvent(false);

    // pri dokonceni wfApp udělá následující
    wfApp.Completed = delegate(
        WorkflowApplicationCompletedEventArgs f)
    {

        // přiřadí stringu Koreny hodnotu, kterou získá z
            výstupu workflow
         Koreny = Convert.ToString(f.Outputs["koreny"]);
         Numberx1 = Convert.ToString(f.Outputs["x1"]);
         Numberx2 = Convert.ToString(f.Outputs["x2"]);
         Numberxi1 = Convert.ToString(f.Outputs["xi1"]);
         Numberxi2 = Convert.ToString(f.Outputs["xi2"]);

         //votvoří nový ThreadStart pro běh WindowState2
         ThreadStart childref = new ThreadStart(OpenWindow2.
             getInstance().openWindow2);
         //vytvoří nový thread
         childThread = new Thread(childref);
         //nastaví thread do STA stavu
         childThread.SetApartmentState(ApartmentState.STA);
         //spustí thread
```

```
            childThread.Start();
            syncEvent.Set();

        };

        //při ukončení workflow v průběhu vyhodí chybu do konzole
        wfApp.Aborted = delegate(
            WorkflowApplicationAbortedEventArgs f)
        {
            Console.WriteLine(f.Reason);
            syncEvent.Set();
        };

        //při neošetřené podmínce v průběhu aplikace vyhodí chybu
            do konzole
        wfApp.OnUnhandledException = delegate(
            WorkflowApplicationUnhandledExceptionEventArgs f)
        {
            Console.WriteLine(f.UnhandledException.ToString());
            return UnhandledExceptionAction.Terminate;
        };

        //při Idle stavu aplikace zapne idleEvent
        wfApp.Idle = delegate(WorkflowApplicationIdleEventArgs f)
        {
            idleEvent.Set();
        };

        //vytvoření a namapování extension
        OpenWindow1 okno1 = new OpenWindow1();

        OpenWindow2 okno2 = new OpenWindow2();


        wfApp.Extensions.Add(okno1);
        wfApp.Extensions.Add(okno2);

        wfApp.Run();
    }

    //metoda vracející na bookmarky
    public void resBook1(string text0,string text1,string text2,
        string text3)
    {
        wfApp.ResumeBookmark("Bookmark0", text0);
        wfApp.ResumeBookmark("Bookmark1", text1);
```

```
                wfApp.ResumeBookmark("Bookmark2", text2);
                wfApp.ResumeBookmark("Bookmark3", text3);

            }

        }
}
```



Obr. 1.2: MainWindow

Díky tomu, že MainWindow je vytvořeno automaticky při vytvoření projektu, bude po spuštění aplikace hned spuštěna metoda MainWindow() a tedy zobrazeno první okno. Dále v tomto kódu naleznete metody starající se o on_click_eventy tlačítek. Třída worker obsahuje dvě metody: první z nich se jmenuje spust a stará se o vytvoření, spuštění a běh workflow. Můžete si také všimnout, že při dokončení workflow přiřadí do proměnných výsledky z worfklow a zobrazí okno ve kterých budou vypsané. Dále třída worker obsahuje ještě singleton instanci, kterou obsahuje z důvodu potřeby volání jejích metod z jiných tříd. Druhá metoda má jméno resBook1 a stará se o volání bookmarků. Vytvořením tohoto kódu končí úprava hlavního okna.

Vytvoření tříd starajících se o otevírání oken

1. V okně **Solution Explorer** klikněte pravým tlačítkem na **Wpf_Ukázka** a zvolte **Add⇒Class**. V okně **Add New Item** zvolíme **Class**, do **Name** napíšeme *OpenWindow1* a stiskneme **Add**.

2. V nově vytvořené třídě nahradíme aktuální kód tímto:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Wpf_Ukazka
{
    //třída zajišťující otevření okna
    class OpenWindow1
    {
            public void openWindow1()
            {
                //vytvoří nové okno a zobrazí ho
                new WindowState1().ShowDialog();
            }
    }
}
```

Tato třída je velmi jednoduchá a obsahuje jedinou metodu, jejímž úkolem je vytvořit nové okno a zobrazit jej. Nakonec stiskněte CTRL+SHIFT+B k buildnutí a uložení změn.

3. V okně **Solution Explorer** klikněte pravým tlačítkem na **Wpf_Ukázka** a zvolte **Add⇒Class**. V okně **Add New Item** zvolíme **Class**, do **Name** napíšeme *OpenWindow2* a stiskneme **Add**. V této třídě kód nahradíme následujícím:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace Wpf_Ukazka
{
    class OpenWindow2
    {
        public void openWindow2()
        {
            new WindowState2().ShowDialog();
        }

        //deklarování singleton instance
        private static OpenWindow2 instance;
```

```
        //pokud je instance nulová, vytvoří novou. pokud ne, vrátí
            aktuální
        public static OpenWindow2 getInstance()
        {
            if (OpenWindow2.instance == null)
            {
                OpenWindow2.instance = new OpenWindow2();
            }
            return OpenWindow2.instance;
        }
    }
}
```

Tato třída obsahuje též metodu na vytvoření a zobrazení okna. Navíc je zde použita instance, aby bylo možné na třídu odkazovat. Nakonec stiskněte CTRL+SHIFT+B k buildnutí a uložení změn. Tím jsme vytvořili všechny třídy, které budeme potřebovat k ovládání aplikace.

Vytvoření vlastních CodeActivit

1. V okně **Solution Explorer** klikneme pravým tlačítkem na **Wpf_Ukázka** a zvolíme **Add⇒New Item...**. V okně **Add New Item** klikneme na **Visual C#**,z nabídky zvolíme **Workflow** a vybereme **CodeActivity**. Do **Name** napíšeme *OpenWindowActivity* a stiskneme **Add**.
2. Kód uvnitř aktivity nahradíme :

```
  using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
using System.Threading;

namespace Wpf_Ukazka
{

    public sealed class OpenWindowActivity : NativeActivity<string
        >
    {
        // Definuje vstupní argument aktivity tupu string
        [RequiredArgument]
        public InArgument<string> BookmarkName { get; set; }

        public static Thread childThread;
```

```
        //po zavolání CodeActivity udělá následující
        protected override void Execute(NativeActivityContext
            context)
        {
            //načte extension
            OpenWindow1 ext = context.GetExtension<OpenWindow1>();
            //vytvoří nový thread
            childThread = new Thread(ext.openWindow1);
            //nastaví thread do STA stavu
            childThread.SetApartmentState(ApartmentState.STA);
            //spustí thread
            childThread.Start();
            //vytvoří Bookrmark na který čeká dokud není zavolán
            context.CreateBookmark(BookmarkName.Get(context),
            new BookmarkCallback(OnResumeBookmark));
        }

        protected override bool CanInduceIdle
        {
            get { return true; }
        }

        //po zavolání bookmarku spustí tuto metodu
        public void OnResumeBookmark(NativeActivityContext context
            , Bookmark bookmark, object obj)
        {
            //přiřadí do Result hodnotu přinesenou bookmarkem
            this.Result.Set(context, (obj));

        }
    }
}
```

Díky zvolení dědění vlastností z **NativeActivity** jsme získali možnost používat **Bookmarky** a díky agrumentu string možnost předat jí string hodnotu do Result proměnné. Tato CodeActivita při spuštění získá **extension OpenWindow1** a přidělí jí nový thread, který nastaví do STA stavu a na kterém následně spustí metodu **OpenWindow1**. Poté začne čekat na zavolání jejího **Bookmarku**. Po vyvolání **Bookmarku** přířadí do její proměnné **Result** hodnotu, kterou **Bookmark** dostane při jeho volání.

3. V okně **Solution Explorer** klikneme pravým tlačítkem na **Wpf_Ukázka** a zvolíme **Add⇒New Item...**. V okně **Add New Item** klikneme na **Visual C#**,z nabídky zvolíme **Workflow** a vybereme **CodeActivity**. Do **Name** napíšeme *ReadLine* a stiskneme **Add**.

4. Kód uvnitř aktivity nahraďte následujícím a poté stiskněte CTRL+SHIFT+B k buildnutí a uložení změn.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;

namespace Wpf_Ukazka
{

    public sealed class ReadLine : NativeActivity<int>
    {
        [RequiredArgument]
        public InArgument<string> BookmarkName { get; set; }

        protected override void Execute(NativeActivityContext
            context)
        {
            // Vytvoří Bookmark a čeká dokud nebude zavolán
            context.CreateBookmark(BookmarkName.Get(context),
                new BookmarkCallback(OnResumeBookmark));
        }

        // NativeActivity obsahuje activitiy, které mohou dělat
            asynchronní operace voláním jednoho
        //z CreateBookmark overloadů definovaných v System.
            Activities.NativeActivityContext
        // musí přepisovat CanInduceIdle vlastnosti a vracet true.

        protected override bool CanInduceIdle
        {
            get { return true; }
        }

        public void OnResumeBookmark(NativeActivityContext context
            , Bookmark bookmark, object obj)
        {
            // Po zavolání Bookmarku nastaví do Result hodnotu z
                obj
            this.Result.Set(context, Convert.ToInt32(obj));

        }
    }
}
```

Tato aktivita po spuštění začne čekat na zavolání Bookmarku. Po jeho vyvolání zapíše do své proměnné Result hodnotu, kterou Bookmark obdržel při jeho volání.

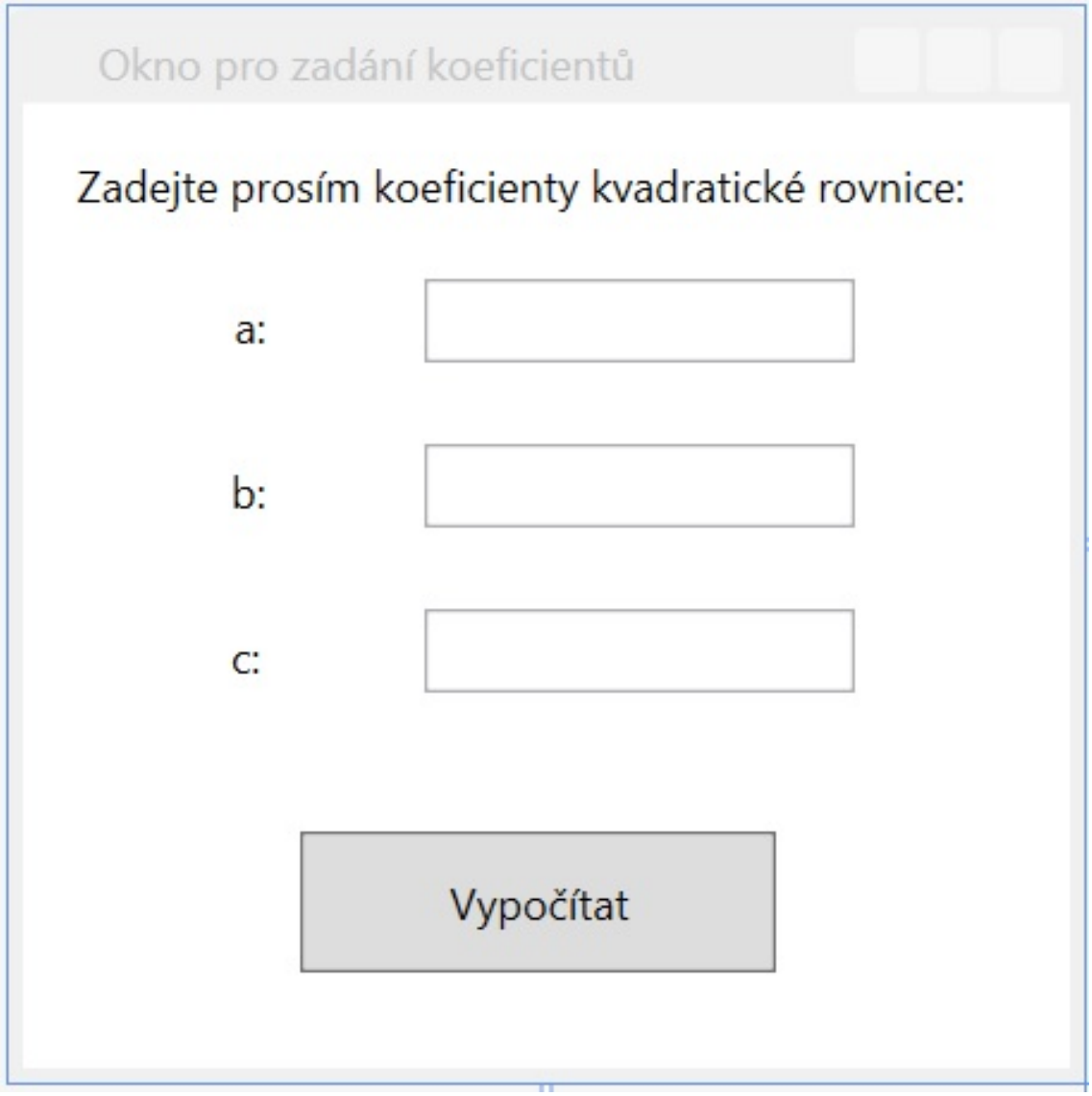
Další CodeActivity nebudeme v našem případě potřebovat a tak se přesuneme k dalšímu kroku.

Vytvoření oken pro zadání koeficientů a zobrazení výsledků

1. V okně **Solution Explorer** klikněte pravým tlačítkem na **Wpf_Ukázka** a zvolte **Add⇒New Item...**. V okně **Add New Item** klikněte na **Visual C#**,z nabídky zvolte **WPF** a vyberte **Window(WPF)**. Do **Name** napíšeme *WindowState1* a stiskněte **Add**.
2. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Button** a vložte jej do okna **WindowState1**. V **Properties** okně **Buttonu** mu do okna **Content** vložte *Vypočítat*. V okně **Properties** klikněte na ikonku **blesku**, čímž se přepnete do **events** části okna. Dvojklikem na okénko vedle textu **Click** vytvoříte on_click_event tlačítka. Pokud vás Visual Studio přepne do **WindowState1.xaml.cs** vraťte se zpět do**WindowState1.xaml**.
3. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState1**. V **Properties** okně **Labelu** mu do okna **Content** vložte *a:*.
4. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState1**. V **Properties** okně **Labelu** mu do okna **Content** vložte *b:*.
5. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState1**. V **Properties** okně **Labelu** mu do okna **Content** vložte *c:*.
6. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState1**. V **Properties** okně **Labelu** mu do okna **Content** vložte *Zadejte prosím koeficienty kvadratické rovnice:*.
7. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **TextBox** a vložte jej do okna **WindowState1**. V **Properties** okně **TextBoxu** mu smažte obsah **Text** okna. Tím zajístíte, že v **TextBoxu** nic nebude. Dále mu v okně **Properties** nastavte do **Name** okna jméno *TextBox1*.
8. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **TextBox** a vložte jej do okna **WindowState1**. V **Properties** okně **TextBoxu** mu smažte obsah **Text** okna. Tím zajístíte, že v **TextBoxu** nic nebude. Dále mu v okně **Properties** nastavte do **Name** okna jméno *TextBox2*.
9. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **TextBox**

a vložte jej do okna **WindowState1**. V **Properties** okně **TextBoxu** mu smažte obsah **Text** okna. Tím zajístíte, že v **TextBoxu** nic nebude. Dále mu v okně **Properties** nastavte do **Name** okna jméno *TextBox3*.

10. Klikněte na WindowState1. V Properties okně WindowState1 mu do **Name** okna nastavte jméno *Okno pro zadání koeficientů*. Výsledné okno by mělo vypadat jako na obr.1.3



Obr. 1.3: Okno pro zadání koeficientů

11. Ve **WindowState1.xaml.cs** nahradíme vygenerovaný kód následujícím:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Wpf_Ukazka
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class WindowState1 : Window
    {
        public WindowState1()
        {
            InitializeComponent();
        }

        //po kliknuti na tlacitko spusti
        private void Button_Click_1(object sender, RoutedEventArgs
            e)
        {
            //vytori string
            string a;string b;string c;
            string fn = "ahoj";

            // pokud v textboxu po stisku tlačitka nebude nic,
                vyhodí messagebox
            if (string.IsNullOrEmpty(TextBox1.Text.ToString()))
            {
                MessageBox.Show("Zadejte prosím číslo a");
                return;
            }
            if (string.IsNullOrEmpty(TextBox2.Text.ToString()))
            {
                MessageBox.Show("Zadejte prosím číslo b");
                return;
            }
            if (string.IsNullOrEmpty(TextBox3.Text.ToString()))
            {
                MessageBox.Show("Zadejte prosím číslo c");
                return;
            }
```

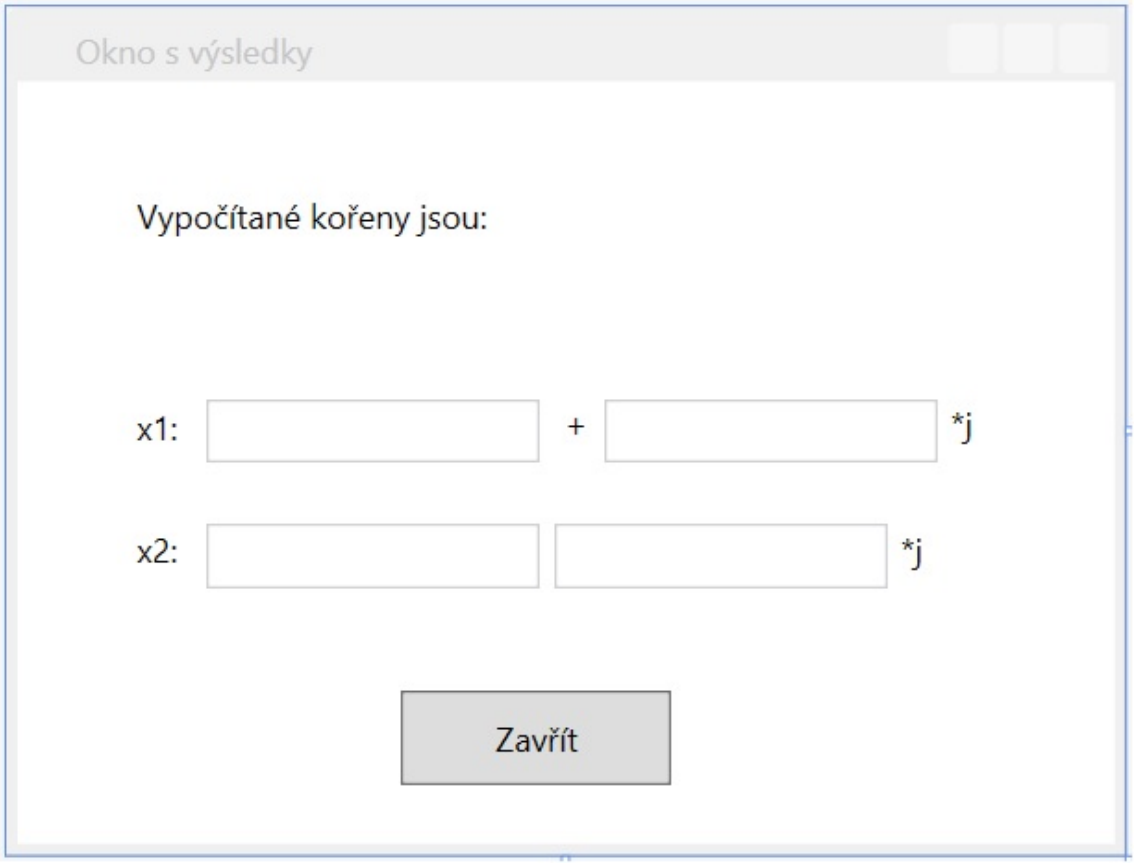
```
            // přiřadí do stringu obsah TextBoxu
            a = TextBox1.Text.ToString();
            b = TextBox2.Text.ToString();
            c = TextBox3.Text.ToString();
            //vrátí se na bookmark s obsahem textboxu (a)
            worker.getInstance().resBook1(fn,a,b,c);
            //zavře okono
            this.Close();
        }

        private void TextBox1_TextChanged(object sender,
            TextChangedEventArgs e)
        {
            double a;
            if (!double.TryParse(TextBox1.Text, out a))
            {
                //Pokud do textboxu nenapíšete Int, vyčistí ho
                TextBox1.Clear();
            }
        }
        private void TextBox2_TextChanged(object sender,
            TextChangedEventArgs e)
        {
            double a;
            if (!double.TryParse(TextBox2.Text, out a))
            {
                TextBox2.Clear();
            }
        }
        private void TextBox3_TextChanged(object sender,
            TextChangedEventArgs e)
        {
            double a;
            if (!double.TryParse(TextBox3.Text, out a))
            {
                TextBox3.Clear();
            }
        }
    }
}
```

Tento kód hned po spuštění otevře okno a po kliku na tlačítko zkontroluje, zda nejsou TextBoxy prázdné(pokud ano tak vyhodí MessageBox s žádostí o zadání daného čísla), pokud nejsou přiřadí do stringů hodnoty z TextBoxů, které následně předá pomocí bookmarku do workflow. Metody TextChanged

zajišťují, že uživatel bude moci zadávat do TextBoxů pouze čísla. Zajišťují to pomocí jednoduchého TextChanged eventu, při kterém zjistí typ znaku zadaného do TextBoxu a pokud to není integer, vymaže ho.

12. V okně **Solution Explorer** klikněte pravým tlačítkem na **Wpf_Ukázka** a zvolte **Add⇒New Item...**. V okně **Add New Item** klikněte na **Visual C#**,z nabídky zvolte **WPF** a vyberte **Window(WPF)**. Do **Name** napíšeme *WindowState2* a stiskněte **Add**.
13. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Button** a vložte jej do okna **WindowState2**. V **Properties** okně **Buttonu** mu do okna **Content** vložte *Zavřít*. V okně **Properties** klikněte na ikonku **blesku**, čímž se přepnete do **events** části okna. Dvojklikem na okénko vedle textu **Click** vytvoříte on_click_event tlačítka. Pokud vás Visual Studio přepne do **WindowState2.xaml.cs** vraťte se zpět do**WindowState2.xaml**.
14. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState2**. V **Properties** okně **Labelu** mu do okna **Content** vložte *Vypočítané kořeny jsou:*.
15. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState2**. V **Properties** okně **Labelu** mu do okna **Content** vložte *x1:*.
16. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState2**. V **Properties** okně **Labelu** mu do okna **Content** vložte *x2:*.
17. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState2**. V **Properties** okně **Labelu** mu do okna **Content** vložte *+*.
18. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState2**. V **Properties** okně **Labelu** mu do okna **Content** vložte **j*.
19. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState2**. V **Properties** okně **Labelu** mu do okna **Content** vložte **j*.
20. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **Label** a vložte jej do okna **WindowState2**. V **Properties** okně **Labelu** mu okno **Content** nechte prázdné(smažte jeho obsah).
21. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **TextBox** a vložte jej do okna **WindowState2**. V **Properties** okně **TextBoxu** mu smažte obsah **Text** okna. Tím zajístíte, že v **TextBoxu** nic nebude. Dále mu v okně **Properties** nastavte do **Name** okna jméno *TextBoxX1*.

22. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **TextBox** a vložte jej do okna **WindowState2**. V **Properties** okně **TextBoxu** mu smažte obsah **Text** okna. Tím zajístíte, že v **TextBoxu** nic nebude. Dále mu v okně **Properties** nastavte do **Name** okna jméno *TextBoxX2*.
23. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **TextBox** a vložte jej do okna **WindowState2**. V **Properties** okně **TextBoxu** mu smažte obsah **Text** okna. Tím zajístíte, že v **TextBoxu** nic nebude. Dále mu v okně **Properties** nastavte do **Name** okna jméno *TextBoxXi1*.
24. V okně **Toolbox** ze záložky **Common WPF Controls** zvolte **TextBox** a vložte jej do okna **WindowState2**. V **Properties** okně **TextBoxu** mu smažte obsah **Text** okna. Tím zajístíte, že v **TextBoxu** nic nebude. Dále mu v okně **Properties** nastavte do **Name** okna jméno *TextBoxXi2*.
25. Klikněte na WindowState2. V Properties okně WindowState2 mu do **Name** okna nastavte jméno *Okno s výsledky*. Výsledné okno by mělo vypadat jako na obr.1.4 (label bez obsahu je vpravo od labelu s obsahem Vypočítané kořeny)



Obr. 1.4: Okno s výsledky

26. Nyní nahradíme kód WindowState2.xaml.cs následujícím:

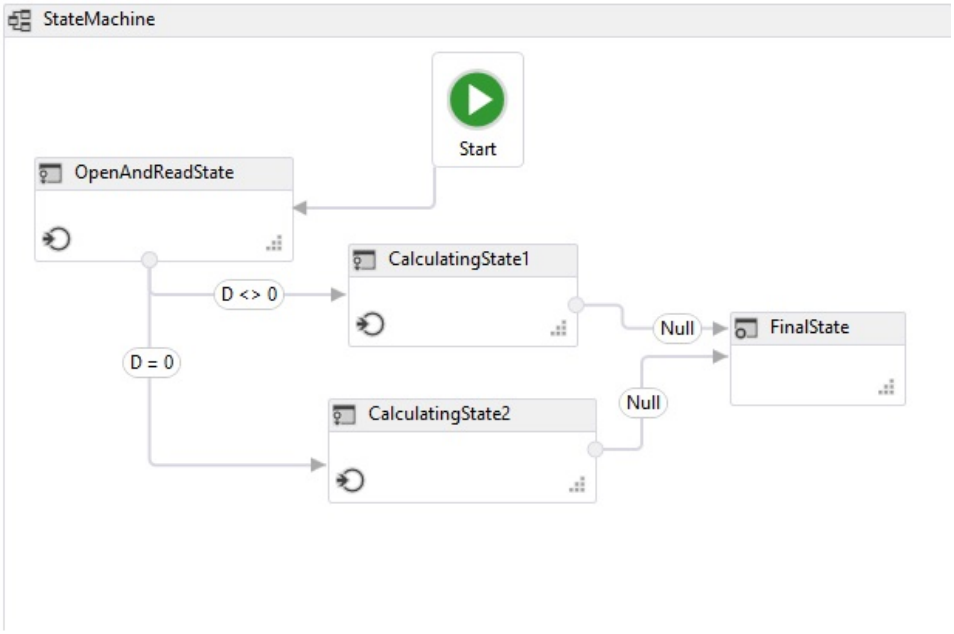
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Wpf_Ukazka
{
    /// <summary>
    /// Interaction logic for Window2.xaml
    /// </summary>
    public partial class WindowState2 : Window
    {
        public WindowState2()
        {
            //zapni okno
            InitializeComponent();
            TextBoxX1.Text = worker.getInstance().Numberx1;
                        //do textboxu přiřadí hodnotu z Numberx1
            TextBoxXi1.Text = worker.getInstance().Numberxi1;
            TextBoxX2.Text = worker.getInstance().Numberx2;
            TextBoxXi2.Text = worker.getInstance().Numberxi2;
            Koreny.Content = worker.getInstance().Koreny;
        }

        private void Button_Click_1(object sender, RoutedEventArgs
            e)
        {
            //ukončí thread a zavře okno. ukončení threadu je zde
                jen pro jistotu
            worker.getInstance().childThread.Abort();
            this.Close();
        }
    }
}
```

Tento kód hned po spuštění otevře okno a přiřadí TextBoxům hodnoty uložené v proměnných Numberx1,Numberxi1,Numberx2,Numberxi2 a Labelu přiřadí

hodnotu uloženou v proměnné Koreny. Při stisku tlačítka ukončí Thread na kterém běží a zavře se.

Vytvoření modelu průběhu

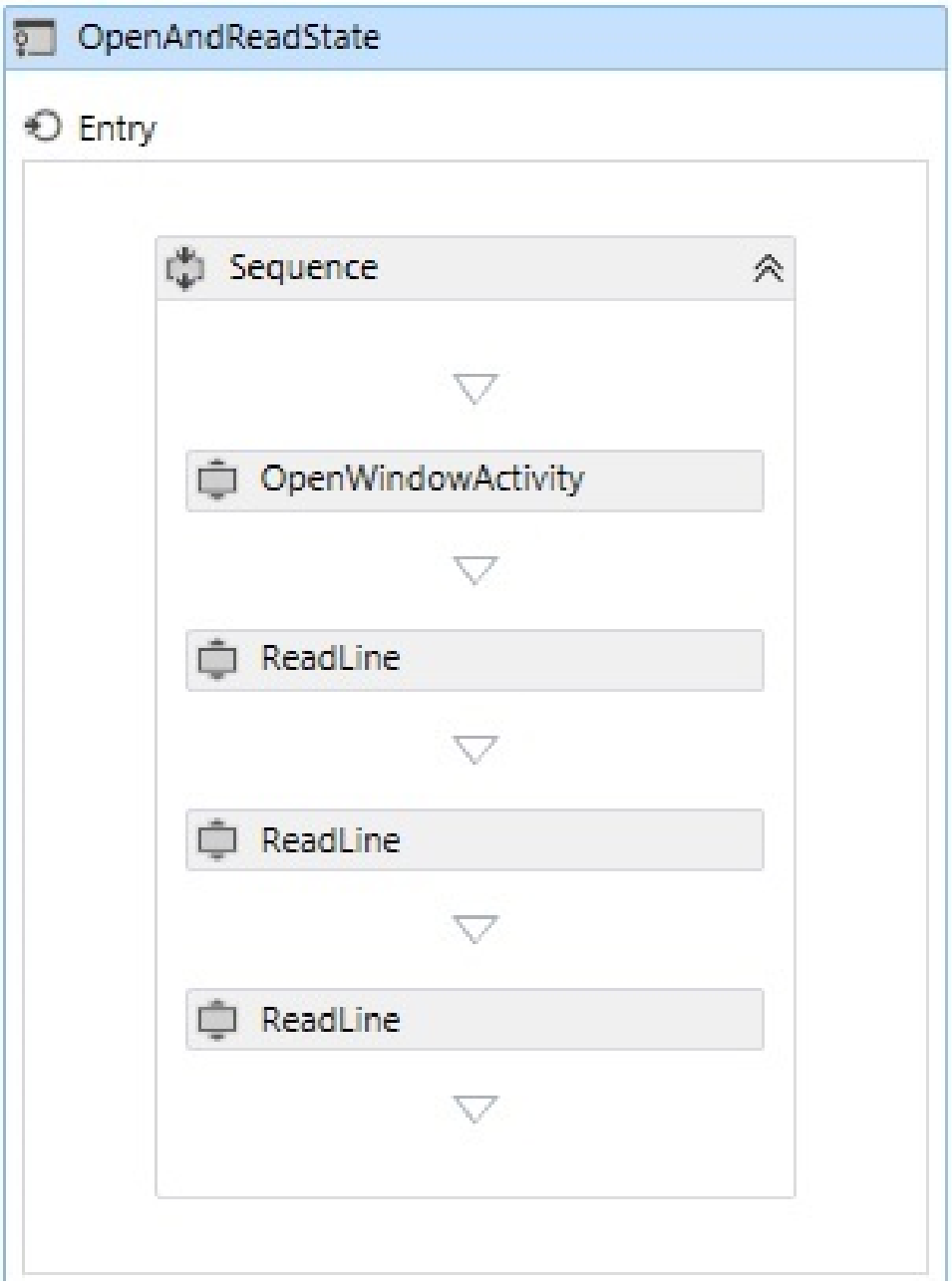
1. V okně **Solution Explorer** klikneme pravým tlačítkem na **Wpf_Ukázka** a zvolíme **Add⇒New Item...**. V okně **Add New Item** klikneme na **Visual C#**,z nabídky zvolíme **Workflow** a vybereme **Activity**. Do **Name** napíšeme *ModelPrubehu* a stiskneme **Add**.
2. V **Toolboxu** vybereme záložku **State Machine** a z ní **StateMachine**, které vložíme do v předchozím bodě vytvořené Activity.
3. Ze stejného místa v **Toolboxu** do Activity vložíme ještě dvakrát **State** a jednou **FinalState**. První State při vkládaní přibližte k automaticky vygenerovanému stavu(State1). Objeví se malý trojúhelník, na který když jej přiložíte, automaticky se mezi stavy vytvoří transition.
4. U druhého **Statu** to již nebude možné, protože chceme aby sdíleli **trigger**. Takže jej vložte pouze do **StateMachine** Activity. Následně najeďte myší na kolečko, odkud vychází **transition** z prvního Statu a kliknutím a tažením jej napojte na **State3**.
5. Nakonec z **Toolboxu** vyberte **FinalState** a vložte jej do **StateMachine** Activity. Z obou Statů vytvořených v předchozích bodech(stavy State2 a State3) vytvořte transition do **FinalState**. Celé workflow by mělo vypadat jako na obr. 1.5



Obr. 1.5: Model průběhu

6. Dole na liště **WorkflowDesigneru** klikněte na **Variables**. V zobrazeném okně klikněte na **Create Variable**. Do name napište *D*, Variable type zvolte **Double**(poprvé se nezobrazí a tak musíte z nabídky zvolit Browse For Types, v nově otevřeném okně napsat nahoře do **Type Name:** Double a vybrat System→Double.).
7. Dole na liště **WorkflowDesigneru** klikněte na **Arguments**. V zobrazeném okně klikněte na **Create Argument**. Do name napište *x1*, Direction zvolte **Out**, Argument type zvolte **Double**(poprvé se nezobrazí a tak musíte z nabídky zvolit Browse For Types, v nově otevřeném okně napsat nahoře do Type Name: Double a vybrat System→Double.).
8. Klikněte na **Create Argument**. Do name napište *x2*, Direction zvolte **Out**, Argument type zvolte **Double**.
9. Klikněte na **Create Argument**. Do name napište *xi1*, Direction zvolte **Out**, Argument type zvolte **Double**.
10. Klikněte na **Create Argument**. Do name napište *xi2*, Direction zvolte **Out**, Argument type zvolte **Double**.
11. Klikněte na **Create Argument**. Do name napište *koreny*, Direction zvolte **Out**, Argument type zvolte **String**.
12. Klikněte na **Create Argument**. Do name napište *a*, Direction zvolte **In**, Argument type zvolte **Int32**.
13. Klikněte na **Create Argument**. Do name napište *b*, Direction zvolte **In**, Argument type zvolte **Int32**.
14. Klikněte na **Create Argument**. Do name napište *c*, Direction zvolte **In**, Argument type zvolte **Int32**.
15. Dvakrát klikněte na **State1**, čímž se přesunete dovnitř stavu. Klikněte na **State1** v titulku stavu, a přejmenujte jej na *OpenAndReadState*.
16. V **Toolboxu** ze záložky **Control Flow** vyberte **Sequence** a vložte ji do okna **Entry**.
17. V **Toolboxu** ze záložky Wpf_Ukazka vyberte **OpenWindowActivity** a vložte ji do **Sequence**. Označte OpenWindowActivity a v okně **Properties** do **BookmarkName** zadejte "*Bookmark*0".
18. V **Toolboxu**ze záložky Wpf_Ukazka vyberte **ReadLine** a vložte ji do **Sequence** pod **OpenWindowActivity**. Vytvořenou **ReadLine** označte a v okně **Properties** do **BookmarkName** zadejte "*Bookmark*1" a do Result *a*.
19. V **Toolboxu** ze záložky Wpf_Ukazka vyberte **ReadLine** a vložte ji do **Sequence** pod přechozí **ReadLine**. Vytvořenou **ReadLine** označte a v okně **Properties** do **BookmarkName** zadejte "*Bookmark*2" a do Result *b*.
20. V **Toolboxu** ze záložky Wpf_Ukazka vyberte **ReadLine** a vložte ji do **Sequence** pod přechozí **ReadLine**. Vytvořenou **ReadLine** označte a v okně **Proper-**

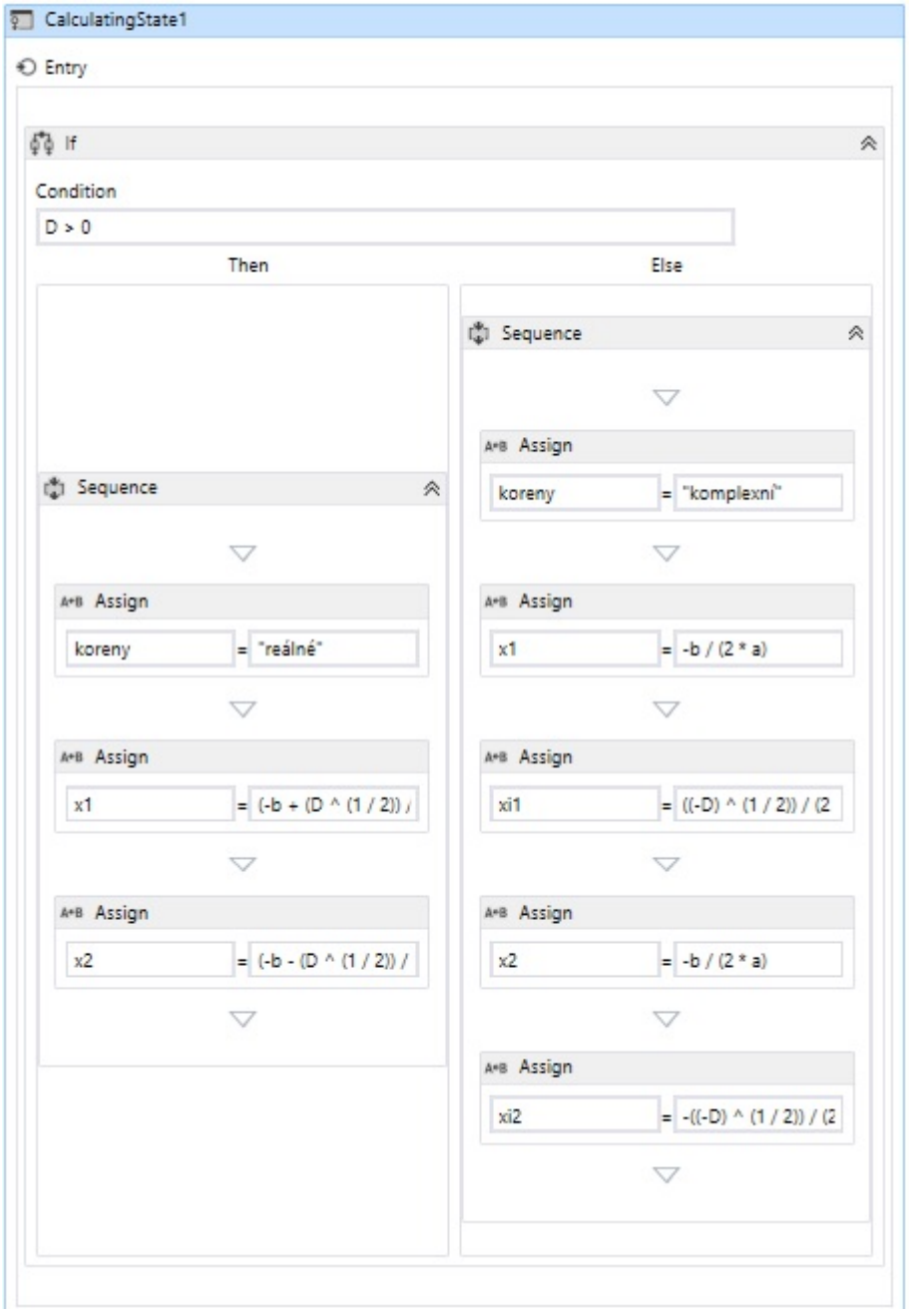
ties do **BookmarkName** zadejte "*Bookmark*3" a do Result *c*.Výsledný stav by měl vypadat jako na obr.1.6. Klikněte na **ModelPrubehu** pro návrat zpět k zobrazení celého workflow.



Obr. 1.6: OpenAndReadState

21. Dvojklikem na **Transition** vedoucí z **OpenAndReadState** do něj přejděte. Následně jej přejmenujte na D<>0. Do okna **Trigger** vložte z **Toolboxu** ze sekce **Primitives** aktivitu **Assign**. V **Properties** okně do jejího okna **To** vložte D a do **Value** $(b * b) - 4 * a * c$. Do **Condition** vložte D<>0.

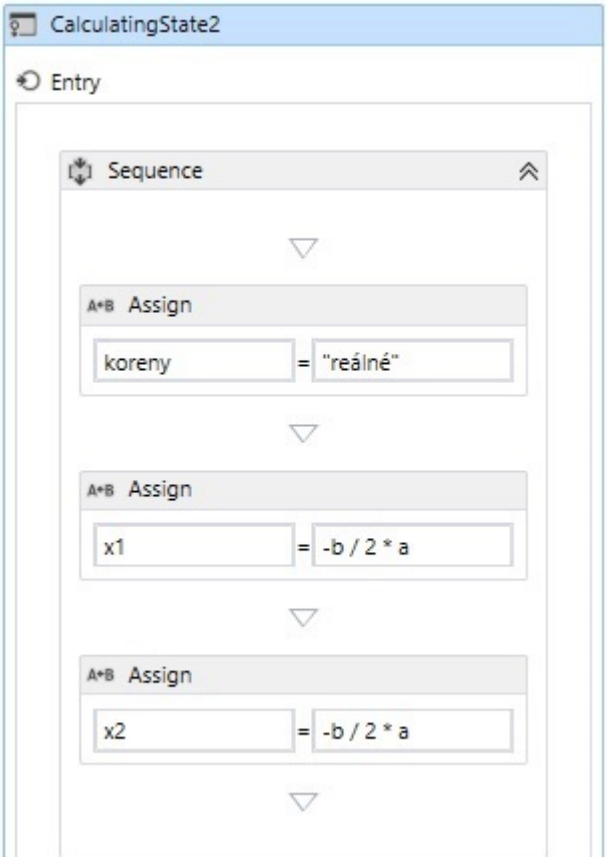
22. Kliknutím na **State2** v dolní části **Transition** okna do něj přejděte. Po otevření jej přejmenujte na *CalculatingState1*.
23. V **Toolboxu** ze záložky **Control Flow** vyberte **If** aktivitu a vložte ji do **Entry** okna **CalculatinState1**. Do **Condition If** aktivity vložte $D > 0$.
24. V **Toolboxu** ze záložky **Control Flow** vyberte **Sequence** a vložte ji do **Then** okna **If** aktivity. Do ní vložte z **Toolboxu** ze sekce **Primitives** aktivitu **Assign**. V **Properties** okně Assing aktivity do jejího okna **To** vložte *koreny* a do Value "*reálné*".
25. V **Toolboxu** ze sekce **Primitives** vyberte aktivitu **Assign** a vložte ji za **Assign** aktivitu z předchozího bodu. V **Properties** okně nové **Assign** aktivity do jejího okna **To** vložte *x1* a do Value $(-b + (D\hat{}(1/2))/2 * a)$.
26. V **Toolboxu** ze sekce **Primitives** vyberte aktivitu **Assign** a vložte ji za **Assign** aktivitu z předchozího bodu. V **Properties** okně nové **Assign** aktivity do jejího okna **To** vložte *x2* a do Value $(-b - (D\hat{}(1/2))/2 * a)$.
27. V **Toolboxu** ze záložky **Control Flow** vyberte **Sequence** a vložte ji do **Else** okna **If** aktivity. Do ní vložte z **Toolboxu** ze sekce **Primitives** aktivitu **Assign**. V **Properties** okně **Assing** aktivity do jejího okna **To** vložte *koreny* a do Value "*komplexní*".
28. V **Toolboxu** ze sekce **Primitives** vyberte aktivitu **Assign** a vložte ji za **Assign** aktivitu z předchozího bodu. V **Properties** okně nové **Assign** aktivity do jejího okna **To** vložte *x1* a do Value $-b/(2 * a)$.
29. V **Toolboxu** ze sekce **Primitives** vyberte aktivitu **Assign** a vložte ji za **Assign** aktivitu z předchozího bodu. V **Properties** okně nové **Assign** aktivity do jejího okna **To** vložte *xi1* a do Value $((-D)\hat{}(1/2))/(2 * a)$.
30. V **Toolboxu** ze sekce **Primitives** vyberte aktivitu **Assign** a vložte ji za **Assign** aktivitu z předchozího bodu. V **Properties** okně nové **Assign** aktivity do jejího okna **To** vložte *x2* a do Value $-b/(2 * a)$.
31. V **Toolboxu** ze sekce **Primitives** vyberte aktivitu **Assign** a vložte ji za **Assign** aktivitu z předchozího bodu. V **Properties** okně nové **Assign** aktivity do jejího okna **To** vložte *xi2* a do Value $-((-D)\hat{}(1/2))/(2 * a)$. Celý CalculatingState1 by měl vypadat jako na obr.1.7.
32. Klikněte na **ModelPrubehu** pro návrat a následně se dvojklikem na druhou **transition** přesuňte do jejího designu. Uvnitř by měl již být **Trigger**(sdílený s první transition), takže pouze do pole **Condition** vložte $D = 0$.
33. Kliknutím na **State3** vedle **Destination:** se přesuntě do stavu 3. Přejmenujte jej na *CalculatingState2*.
34. V **Toolboxu** ze záložky **Control Flow** vyberte **Sequence** a vložte ji do **Entry** okna **CalculatingState2**. Do ní vložte z **Toolboxu** ze sekce **Primitives**



Obr. 1.7: CalculatingState1

aktivitu **Assign**. V **Properties** okně **Assing** aktivity do jejího okna **To** vložte *koreny* a do Value "*reálné*".

35. V **Toolboxu** ze sekce **Primitives** vyberte aktivitu **Assign** a vložte ji za **Assign** aktivitu z předchozího bodu. V **Properties** okně nové **Assign** aktivity do jejího okna **To** vložte *x1* a do Value $-b/(2 * a)$.
36. V **Toolboxu** ze sekce **Primitives** vyberte aktivitu **Assign** a vložte ji za **Assign** aktivitu z předchozího bodu. V **Properties** okně nové **Assign** aktivity do jejího okna **To** vložte *x2* a do Value $-b/(2 * a)$. CalculatingState2 by měl vypadat jako na obr. 1.8



Obr. 1.8: CalculatingState2

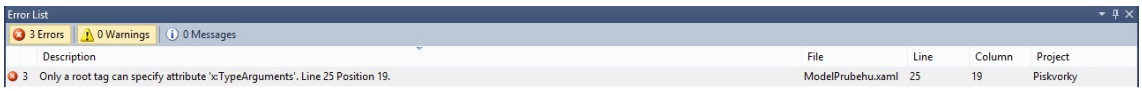
Tímto jsme vytvořili model průběhu naší aplikace, což bylo posledním krokem naší tvorby. Stiskem CTRL + SHIFT + B buildněte solution a následně spusťte aplikaci.

2 VÝSLEDKY STUDENTSKÉ PRÁCE

2.1 Workflow peer-to-peer piškvorky

Úkolem mé práce bylo navrhnout peer-to-peer piškvorky pro dva hráče pomocí WPF a workflow. Bohužel při vytváření této aplikace nastaly problémy, které mi nakonec znemožnili v aplikaci plně zprovoznit workflow model. Aplikace peer-to-peer piškvorek je přiložena v přílohách ve stavu, do jakého jsem ji byl schopen zprovoznit. Nyní bych se tedy rád pokusil nejzásadnější a nejčastější problémy v následujícím textu popsat.

- Prvním a dosud nevyřešeným problémem je eror, zobrazený na obr.2.1



Obr. 2.1: Error při použití InvokeMethod Aktivity

  Tento error se objeví kdykoliv ve workflow modelu použiji InvokeMethod aktivitu. Problém se objevuje i v případě, že aktivitu používám přesně podle návodu vývojářů. Podle mnou získaných informací by měl jít odstranit správným nastavením namespaců XAML souboru. Bohužel nemohu tuto teorii potvrdit, jelikož se mi to nepovedlo. Díky tomuto eroru jsem tedy přišel o možnost volat jednotlivé metody z přímo z workflow. Zůstala tedy možnost sestavit workflow celé z custom CodeActivit.
- Druhým problémem který mě velice dlouho trápil bylo, že jsem nemohl najít v žádných zdrojích jak zkombinovat WPF a workflow v jedné aplikaci. (Všechny zdroje předpokládají že workflow bude konzolová či webová aplikace a WPF neřeší, stejně tak ve WPF aplikacích a jejich dokumentacích nikdo nepočítá s workflow.) Nemohl jsem zjistit, jak předávat informace z workflow do oken WPF, ani jak z workflow spustit WPF okna a naopak. Tento problém jsem nakonec vyřešil, jak je vidět z vzorové aplikace popsané výše. Workflow se totiž dá spustit z jakéhokoliv eventu (například ze stisku tlačítka), avšak díky poněkud nešťastnému řešení WPF oken bylo nutné pro spouštění workflow vytvořit speciální třídu obsahující metodu na spouštění a spravování průběhu workflow. V této třídě bylo nutné vytvořit ještě singleton instanci, pro možnost na třídu odkazovat. Naopak pro spouštění WPF oken z workflow, bylo nutné použít extensions. Jako extension se do workflow aplikace přidá třída obsahující metodu spouštějící okno. Ta se následně například pomocí bookmarku vyvolá a nové okno se zobrazí. Tato cesta je sice zdlouhavá a nejspíše jsou i jiná řešení, avšak důležité je, že funguje.

- Třetím problémem bylo, že po vytvoření workflow aplikace a jejím spuštění worfklow proběhlo, avšak visuálně se nic nezměnilo a Visual Studio 2010 nenašlo žádné errory. Tato chyba byla nejspíše způsobena absencí všech stavů workflow aplikace. Při jejím vytvoření tedy bylo nutné vždy definovat stavy completed,idle atd. . . .
- Čtvrtým problémem, který bylo nutné odstranit pro správný běh aplikace, byla skutečnost, že při pokusu otevřít jakékoliv WPF okno z workflow aplikace vyhodila error. Tento error vypadal následovně: *The calling thread must be STA, because many UI components require this.* V praxi šlo o problém který se běžně řeší Background workerem. V mnou prezentovaném kódu(je součásti Vzorového příkladu) je tento problém vyřešen pomocí extensions a následným vytvořením nového Threadu, který se po vytvoření nastaví do STA stavu.
- Posledním významným problémem byla samotná peer-to-peer komunikace. Workflow by k podobným případům mělo používat SendActivity. Tyto aktivity jsou součástí workflow a vypadají jako velmi vhodný a rozumně definovaný prvek. Jsou zde i RecieveAndSendReply a SendAndRecieveReply aktivity, které jsou jako stvořené pro komunikaci mezi aplikacemi. Bohužel však nikde není k sehnání dokumentace jak komunikaci pomocí těchto aktivit zprovoznit. Vzhledem k tomuto faktu jsem se nakonec rozhodl k této komunikaci použít klasické funkce knihovny WCF. K přijímání a odesílání dat jsem vytvořil třídy, které jsou součástí přiložené aplikace s názvem piškvorky. Fungují za pomocí definice EndPointu a následného sestavení spojení mezi dvěmi aplikacemi. Samotná komunikace pak probíhá asynchronně pomocí asynchronních callbacku.
- Dále se vyskytlo několik menších problémů týkajících se workflow, avšak tyto problémy byly způsobeny pouze mou neznalostí a špatnou dostupností materiálů, které by daný problém objasňovali. Všechny tyto problémy jsem nakonec tedy dokázal nějakým způsobem buď obejít, nebo vyřešit. Bohužel zkoušení různých řešení a hledání podkladů mi zabralo tolik času, že jsem již nestihl aplikaci piškvorek přepsat a tak jsem funkce a možnosti aplikací vytvořených pomocí workflow a WPF demonstroval na jednoduchém příkladu výpočtu Kvadratických rovnic.

2.2 Aplikace pro výpočet kořenů kvadratických rovnic

V první části této práce jsem popsal návrh a následně i postup vytvoření aplikace pro výpočet kořenů kvadratických rovnic. Účelem této aplikace je hlavně uživatele seznámit s možnostmi kombinace WPF a workflow. Součástí kódu je několik tříd, ve kterých je za pomocí Bookmarků, Instancí a eventů zajištěna komunikace mezi uživatelem, WPF okny a samotným workflow. Je zde také krásně vidět řešení problému, popsaných v předchozí kapitole. Celý kód je přehledně komentován pro snašší orientaci a lepší přehlednost.

3 ZÁVĚR

Účelem celé mé práce bylo seznámit čtenáře s možnostmi aplikací tvořených za pomoci Windows Workflow Foundation a Windows Presentation Foundation knihoven z .NET Frameworku 4.0. Ve vzorovém příkladu se snažím co nejpřehledněji a nejnázorněji předvést postup návrhu a tvorby aplikace. Při samotné tvorbě se snažím popsat jednotlivé funce a metody, aby měl čtenář stále přehled o právě používaných metodách a funkcích kódu. Nejdůležitějšími poznatky, které tato práce ukazuje jsou spolupráce workflow a WPF a práce s metodami podporujícími tuto spolupráci.

Na závěr bych se ještě rád zmínil o workflow technologii a přístupu jejích autorů k ní. Nápad a směr, jakým se snaží WF jít jsou dle mého názoru správné a velice ambiciózní. Pokud by se WF rozšířilo a začalo užívat například ve velkých vývojových týmech, mohlo by ušetřit práci a hlavně dovolit například manažerům pouze tvořit WF modely z aktivit, které by jim programátoři předepsali. Tím by se docílilo toho, že by se manažeři mohli plně soustředit na vývoj workflow, a nemuseli se rozptylovat kódem, který pracuje na pozadí. Na druhou stranu já osobně bych dal přednost „klasickému“ programování, protože celá workflow nadstavba mi přijde zbytečně náročná a zpomalující běh celého programu. Navíc když se podíváme na dostupnost materiálů k workflow, zjistíme, že sehnat kvalitní materiály je velmi náročné. Microsoft na svých stránkách(MSDN Library) sice nějaké materiály k workflow má, ale většinou jsou hodně stručné a občas by mnohem více pomohli názorné příklady. Microsoft navíc vydává nové verze WF tak rychle, že často nestačí aktualizovat dokumenty a pak méně zkušeným uživatelům nezbývá nic jiného než používat starší verzi, neboť v nové jsou například změněny syntaxe příkazů. Nejvíce mě ale překvapil fakt, že jediným člověkem snažícím se uživatelům představit možnosti workflow je Ron Jacobs, který na praktických příkladech vysvětluje jednotlivé funkce. Do budoucna doufám že materiálů k worfklow přibude, a bude tak mnohem snaží se s touto technologií naučit pracovat a plně ji využívat.

LITERATURA

[1] SHARP, J. *Microsoft Visual C♯ 2010*, Nakladatelství Computer Press, a.s. 2010, 696 s., ISBN 978-80-251-3147-3

[2] PETZOLD, CH. *Mistrovství ve Windows Presentation Foundation*, Nakladatelství Computer Press, a.s. 2008, 928 s., ISBN 978-80-251-2141-2

[3] WATSON, B. *C♯ 4.0 - řešení praktických programátorských úloh*, Zoner press, 2010, 656 s., ISBN 978-80-7413-094-6

[4] JACOBS, R. *Getting Started Tutorial*. JACOBS, Ron. MSDN [online]. 2012, 3.2.2012 [cit. 2012-12-15]. Dostupné z: http://msdn.microsoft.com/en-us/library/dd489454(v=vs.100).aspx

[5] *Designing Workflows*. MICROSOFT. MSDN [online]. 2012, 3.2.2012 [cit. 2012-12-15]. Dostupné z: http://msdn.microsoft.com/en-us/library/dd489-422(v=vs.100).aspx

[6] *Windows Workflow Foundation Data Model*. MICROSOFT. MSDN [online]. 2012, 3.2.2012 [cit. 2012-12-15]. Dostupné z: http://msdn.microsoft.com/en-us/library/dd489457(v=vs.100).aspx

A PŘÍLOHY

A.1 Obsah Přiloženého CD

Přiložené DvD obsahuje dva adresáře:

`/UkazkovePriklady` obsahuje jeden podadresář a tři soubory:

Podadresář: `/Piskvorky`

Tento podadresář obsahuje zdrojové kódy a soubory potřebné pro aplikaci piškvorek

1.soubor: `/packages`

Tento soubor obsahuje zdrojové kódy a soubory potřebné pro správný debug aplikace piškvorek

2.soubor: `/ReadMe`

Tento soubor obsahuje informace k aplikaci uložené v adresáři /UkazkovePriklady

3.soubor: `/UkazkovePriklady` Solution soubor Visual Studia

`Workflow` obsahuje jeden adresář a jeden soubor:

Podadresář: `/Piskvorky`

Tento podadresář obsahuje zdrojové kódy a soubory potřebné pro ukázkovou aplikaci

Soubor: `/packages`

Tento soubor obsahuje zdrojové kódy a soubory potřebné pro správný debug ukázkové aplikace