JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Pedagogická fakulta, Katedra informatiky

Kontrolky ve WPF

Bakalářská práce

Jan Lasac

Vedoucí bakalářské práce: **Ing. Václav Novák, CSc.**

Akademický rok: **2008/2009**

Návrh zadání:
Název: Kontrolky ve WPF , Controls inside WPF

Zadání:
Nasazením aplikačního rámce .NET Frameworku ve verzi 3.5 se naskytla i možnost nového přístupu k tvorbě programů. Zejména je možno posílit jejich grafickou stránku. Díky vektorové grafice a skvělé implementaci animací ve WPF můžeme nyní vytvářet efekty, o kterých jsme si ve WinForms mohli nechat pouze zdát. .NET Framework obsahuje spoustu předvytvořených kontrol (Button, TextBox, ComboBox, CheckBox, RadioButton, ProgressBar ...), což usnadňuje práci programátora, kter nemusí neustále implementovat prvky, které používá opakovaně v různých aplikacích. Kontrola je samostatný interaktivní prvek s grafickým rozhraním.

Úkolem diplomanta je:

1. Seznámit se s modelem Windows Presentation Foundation (WPF) uvnitř .NET Frameworku v 3.5.
2. Porovnat tvorbu kontrol ve stávajícím WinForms s Windows Presentation Foundation.
3. Ukázat postup tvorby uživatelských kontrolek, jejich sloučenído samostatně distribuovaných knihoven.
4. Vytvořit příklad knihovny kontrolek s cílem dosáhnou jednotný dising.

Hlavním cílem práce je vytvoření vzorové audiovizuální prezentace doplněné o příklady knihoven kontrolek vedoucí k ujednocení dising aplikací. Programátoři, jež chtějí používat WPF by měli být upozorněni na záludnosti použití.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích dne 22.4.2009

…….……...

Jan Lasac

Anotace

Tato publikace se zabývá novým způsobem programování aplikací pro Windows a popisuje možnosti a přednosti této technologie a jejích vizuálních komponent. V teoretické části práce se seznámíme s modelem Windows Presentation Foundation a bude provedeno srovnání se starší technologií WinForms. V praktické části bude vytvořena knihovna a dále ukázka postupu tvorby prvků formou audiovizuální prezentace.

Abstract

This publication deals with a new way of programming the applications for Windows and describes possibilities and advantages of this technology and its visual components. In the theoretic part we will get to know the model Windows Presentation Foundation and it will be made comparison with the older technology WinForms. In the practical part it will be created the library and then the audio-visual presentation which will show the procedure of the creation controls.

Obsah

Seznam obrázků

Seznam příkladů

Seznam tabulek

Seznam grafů

1 Úvod

V dnešní moderní době je téměř každý den nalezen nový objev nebo zdokonalení již stávajícího, jejichž cílem je usnadnit a zpříjemnit nám život. Tato skutečnost se výrazně projevuje právě v informační technologii, bez které si už život umíme jen těžce představit, protože se vyskytuje „na každém kroku“.

Jednou z novinek, která se snaží o vylepšení a usnadnění práce v informační technologii, je Windows Presentation Foundation (dále jen WPF). I když technologie WPF je stažitelná pro Windows XP SP2 a Windows Server 2003, moji pozornost upoutala hlavně v souvislosti se systémem Windows Vista, který používám a nyní je i přes některé nedostatky na vzestupu.

Technologie WPF se zabývá hlavně vzhledem aplikace, která mě velmi zajímá, a právě díky tomu jsem si jako téma své bakalářské práce zvolil „Kontrolky ve WPF“.

Na následujících stránkách tuto technologii popíši a vysvětlím. Také se pokusím zhodnotit její inovace oproti konkurenční technologii WinForms a jejich smysl a prospěšnost. Dalším mým cílem je vytvořit knihovnu ovládacích prvků ve WPF a dále ukázat postup tvorby prvků formou audiovizuální prezentace.

2 Windows Presentation Foundation

2.1 Vývoj uživatelských rozhraní

Zpočátku fungovaly počítače jen jako pomůcka pro zjednodušení výpočtu, ale postupem času se ukázalo, že mají pro člověka význam především ve vědě, zbrojním průmyslu nebo komunikaci na velké vzdálenosti. Po vynálezu polovodičového tranzistoru se vývoj počítačové techniky zrychlil a umožnil její zdokonalení až do dnešní podoby. V průběhu jejího vývoje se vedle velkých místností, které představovaly počítače, začaly objevovat menší osobní počítače. Díky menším rozměrům a vyspělejší technice se počítače staly dostupnější pro širší skupinu lidí, což zapříčinilo vznik požadavků na vývoj ovládacích rozhraní, které by zpřístupnily osobní počítače i neodborné veřejnosti.

Ovládání počítačů bylo zpočátku plně funkcionálně orientované a i díky míře využití se jednalo o konkrétní koncepty založené na množství signálů (mohly být světelné nebo zvukové), ovládacích tlačítek či přepínačů.

Později se vynalezly děrné štítky, které zjišťovaly potřebu konfigurovat chování počítačů na základě měnících se cílů výpočtu. Štítky mohli používat jen vyškolení počítačoví specialisté, protože představovaly metodu pro programování i zadávání vstupních dat a ne každý uměl pracovat s takovým zařízením.

Pokročilejším zařízením pro ovládání přístroje se později stala *klávesnice*. Zdokonalení se projevilo hlavně v zadávání příkazů pro obsluhu počítače, které velmi zjednodušilo práci na počítači. Klávesnice a související *příkazový řádek* dále rozšířily skupinu uživatelů, neboť to, co jste napsali přístrojem, bylo zobrazováno vyspělejší *textovou obrazovkou*.

Jedním z dalších větších pokroků byla implementace *polohovacího zařízení* neboli *myši* a *barevného grafického displeje*, které ještě více zlepšily ovládání tím, že usnadnily práci s vizuálními objekty. Díky tomu získalo uživatelské rozhraní další dimenzi. Velký přínos to byl i pro výrobce programů, kteří získali prostředky pro efektivní ovládání, což dalo významný impulz tvorbě nových aplikací. Vznikly první operační systémy s grafickým prostředím (MacOs1, Windows2). Postupem času se osobní počítače stávají součástí vybavení podniků a později i domácností. Trh s počítačovými programy se rychle rozrůstá a s tím stoupá i význam uživatelských rozhraní. Cílem je nabízet lehce ovladatelné rozhraní, které akceptuje veškeré podmínky koncových uživatelů a nevyžaduje pokročilé zkušenosti.

Vývoj a pokroky ve výpočetní technice neustále posunují možnosti programového vybavení. V dnešní době má stále významnější roli využití a dostupnost programu. Uživatel musí mít pocit uspokojení z užívání programu a z toho, že program ovládá. V dnešní široké nabídce produktů mají šanci na uplatnění jen kvalitní uživatelská rozhraní, ostatní zanikají, ačkoliv u distribuovaných produktů, které jsou zdarma, jsou uživatelé mnohdy méně žádostiví.

I z mnoha dalších důvodů je návrh uživatelského rozhraní odvětví, které se za několik desítek let rozrostlo do samostatného specializovaného odvětví. Snaží se zkoumat lidské chápání, reakce na podněty a předvídat způsob zacházení s počítačem. Současně vyvíjené systémy se snaží uživateli maximálně ulehčovat práci. Jelikož se setkáváme s nelegálním užíváním programů, úspěšnost produktu se neodvozuje pouze množstvím spokojených uživatelů. Významným faktorem je pro programátora navrácení investice, které vynaložil, poptávka po dalších produktech a aktivní zájem uživatelů o nová vylepšení.

2.2 Východiska pro WPF

Využití osobních počítačů se s rozvojem techniky a nabídky programového vybavení rozšířilo i do oblastí, kde se uplatňovaly jen málo. Už i běžné počítače nabízí práci s digitální fotografií, videem, rozhlasovým a televizním signálem a s dokumenty díky zvyšování výpočetního výkonu i v dalších jednotkách osobních počítačů. To vše je závislé na schopnosti zobrazit data na displeji kvalitně a rychle. Hlavní důraz při výrobě zákaznického softwaru je kladen na užitečné možnosti aplikace. U vysoce specializovaných programů (např. letectví, konstruktérství,…) se hlavně v zobrazovací diagnostice projevily možnosti grafických karet.

V současnosti lze počítače ovládat celou řadou zařízení. Kromě dnes už běžné klávesnice a myši existují dotykové displeje a různé další ovladače. Vysoký komfort ovládání zabezpečují nové požadavky na aplikace. Rozhraní by mělo uživateli umožňovat atraktivní zážitek a odlišovat použití aplikace od jiných způsobů realizace.

Mezi hlavní požadavky na nová rozhraní patří:

- nezávislost na zobrazovacím médiu a jeho rozlišení
- efektivní a přehledné ovládání
- atraktivní uživatelsky nastavitelný vzhled
- atraktivní design
- nastavitelné a variabilní zobrazování dat
- modularita zobrazování
- víceúrovňová správa zdrojů a definic vzhledu

Tyto požadavky je bohužel možné v dnešní technologii splnit jen některé, protože rozhraní procházejí složitým vývojem a laděním, především pak tvorba bohatých a lákavých designů. K uspokojení stále se zvyšující náročnosti uživatelů slouží vizuálně přitažlivé prostředí, které je vytvořeno díky nové a jednodušše použitelné technologii. Často vyžadovaná je přenositelnost produktu mezi různými platformami a zařízeními, která ale většinou není bez problům a lze ji splnit jen omezeně, protože mezi systémy existuje řada velkých odlišností. Svou roli zde hraje také obchodní strategie dodavatele technologie.

2.3 Příchod nové technologie

Mezi nejnovější technologii vytvořenou pro podporu lepších uživatelských rozhraní:

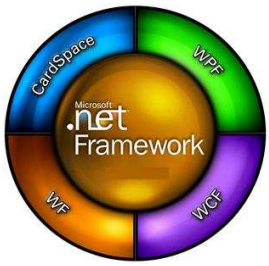
- OpenGL (1990)
- DirectX (1995)
- GTK+ (1997)
- GDI+ (2001)

Většina z těchto technologií je neustále vylepšována, ale i přesto je nutné, aby programátoři tvořili vlastní nové technologie, protože v závislosti na rozličnosti grafických systémů mívají některé architektury problémy a musejí proto být přizpůsobovány různým parametrům.

Nejnovější verzí operačního systému Windows od společnosti Microsoft je systém Windows Vista. Oproti minulým systémům byl vylepšen jeho vzhled a zabezpečení. V základní struktuře je využita technologie .NET Framework, nový grafický a zvukový subsystém, lepší podpora pro instalaci programů a výraznější podpora pro digital rights management (DRM, ochrana proti neoprávněnému kopírování). Visty mají uživateli nabídnout jednoduché a efektivní rozhraní pro ovládání a využívání všech poskytovaných služeb. Uvedením nového systému poskytl Microsoft vývojářům softwaru nový balík technologií pro tvorbu aplikací – specifikaci .NET Framework verze 3.0, která obsahuje řadu funkcí předchozích verzí i množství nových prostředků.

Mezi nové technologie patří:

- ✓ **Windows Communication Foundation** – systém přenosu zpráv mezi programy pro podporu lokální i vzdálené spolupráce, podobně jako u web services
- ✓ **Windows Workflow Foundation** – podpora pro bussiness scénáře a programování složitých workflow procesů
- ✓ **Windows Presentation Foundation** – systém grafického uživatelského rozhraní a API založených na technologiích XML, .NET a vektorové grafice, který využívá schopností moderních grafických adaptérů a technologie DirectX



Obrázek 1: Přehled technologií v NET Frameworku 3.0

Windows Presentation Foundation se snaží dosáhnout posunu tvorby uživatelských rozhraní více do působností designérů s lepšími předpoklady pro moderní a neokoukaný návrh. Je těžké dopředu uvádět, jestli WPF představuje správný prostředek pro všechny typy aplikací, protože se jedná o novou technologii, ale jisté je, že přináší nový způsob jak efektivně a rychle sestavovat zajímavé návrhy, které mohou být v krátké době hodnoceny a posléze vylepšovány.

2.4 Hlavní rysy

Za zkratkou WPF (Windows Presentation Foundation) se skrývá nový grafický framework (3.0 a vyšší) společnosti Microsoft pro psaní Windows aplikací. WPF bylo dříve označované jako Avalon. Pomocí WPF lze psát aplikace, které byste jinak zvládali jen s velkou námahou (nebo vůbec). Tím je samozřejmě myšlena grafická stránka aplikací. Na psaní WPF aplikací se používá XAML (eXtensible Application Markup Language) spolu s .NET jazykem, jako je například C# nebo Visual Basic.NET. Tyto aplikace se renderují na grafické kartě, a proto není nutné obávat se příliš velkého zatížení procesoru při vytváření graficky bohatých aplikací. Cílem je sjednotit celou řadu aplikačních služeb (např. uživatelské rozhraní, 2D a 3D kreslení, data binding, vektorová a rastrová grafika…).

WPF navazuje na Windows Forms, více ho rozvíjí hlavně v grafických dovednostech a nabízí bohatší sadu vizuálních prvků. Nabízí nový způsob, zapouzdření vykreslovací logiky včetně vizuálních efektů, předpokládané umístění na jiné vývojové platformy (WPF/E), dostupnost napojení na všechny „managed-code“ programovací jazyky a šířenitelnost knihovnami. Veškerou práci usnadní a zrychlí vytváření uživatelského rozhraní, a dále poskytne lepší podporu pro následné úpravy. Porozumění a praktické zvládnutí technologie může poskytnout vysoký přínos a další řadu nových možností.

V následujícím výčtu uvádím hlavní oblasti, kterých se inovace týkají:

Hardwarová akcelerace – vykreslování se provádí vektorově, přes grafický hardware (vrstva DirectX), tím je procesor skoro nezatížený a plně využívá schopností výkonných grafických karet. Využití procesoru se uskutečňuje jen v případě, že technické vybavení chybí nebo jeho prostředky nejsou dostačující. Zajímavý návrh aplikace již nemusí znamenat nižší výkon.

Nezávislost na zobrazovacím rozlišení - rozměry prvků a jejich parametrů (velikost písma, odsazení, tloušťka apod.) jsou definovány v nezávislých jednotkách. Neodpovídají počtu pixelů, jednotkou je 1/96 palce – využívají se převážně u systémů Windows (rozměr pak odpovídá počtu pixelů). Nezávislost na rozlišení cílového zařízení znamená, že při vyšším rozlišení displeje (př. 140 bodů/ palec) se výsledné zobrazení nezmenší a nebude hůře čitelné, tudíž zachová shodný rozměr. Automaticky se zobrazení přizpůsobí displeji a ponechá si kvalitní a hladké tvary.

Kompozice – obsahová rozmanitost je důležitým hlediskem při kompozici. Možnost vkládat skoro cokoliv a pouze omezit cílový prostor dodává monitorům velký výběr vizualizace. Ovládací prvky lze vzájemně jednoduše kombinovat, zapouzdřovat či překrývat.

Deklarativní jazyk – výrazný rys WPF, který má za úkol vnořit odborníky na vizuální vzhled do procesu vývoje. Představuje širokou škálu výrazu pro stylizaci. Poskytuje rychle viditelnou odezvu na provedené změny a přístup k nim, včetně dalších nových způsobů jak dynamicky měnit vzhled.

Oddělení vzhledu od kódu – deklarativní jazyk odděluje specifikaci vizuálního vzhledu od programovaného kódu.

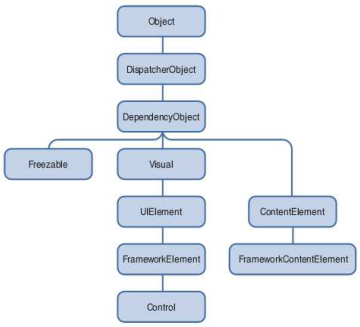
Stylizace a nastavitelnost – technologie nabízí velké množství způsobů, jak demonstrovat data i jejich další stavy. S WPF návrhář může používat geometrické tvary, animace, multimediální prvky, stíny, průhlednost a jiné. U objektů může používat transformace (polohy, velikost, zkosení), což jsou nové techniky, které jsme znali jen u vektorových grafických editorů.

Integrace – technologie, která podporuje 2D a 3D grafiku, multimédia (obrazová informace, video a zvuk), zobrazování dokumentů. Poskytuje kompaktní sadu tříd a ovládacích prvků.

Rozmístění – výsledné řešení lze poskytnout jako běžný program (EXE) či aplikaci dostupnou přes webový prohlížeč (ClickOnce). Přispívá tomu vnořená implementace navigačních kroků, které lze využít v obou formách distribuce.

2.4.1 Architektura

Architekturu WPF můžeme rozdělit do hierarchických tříd. Model celého WPF je založený z uživatelského hlediska na managed kódu. Při jeho vývoji bylo potřeba stanovit hranici mezi managed a unmanaged kódem. Výsledkem je vysoce produktivní, robustný systém zahrnující správu pamětí, ošetření chyb, typovost a jiné. V této části se zmíním především o hlavních subsystémech WPF a popíši, jak jsou vzájemně propojeny.



Obrázek 2: Základní hierarchie pro prvky

Na obrázku 2 je znázorněna základní hierarchie, ze které vycházejí všechny ovládací prvky - Controls (např. Button, Label). Každá třída diagramu přidává soubor vlastností, čímž každá samotná kontrolka přichází s velkou funkcionalitou. Blíže si popíšeme význam jednotlivých tříd v dané hierarchii. Nyní rozebereme jednotlivé subsystémy:

2.4.1.1 Systém.Object

Hlavní prvku WPF jsou znázorněny na obrázku níže. Červené body na diagramu (PresentationFramework, PresentationCore a milcore) jsou hlavní částí kódu WPF. PresentationFramework, PresentationCore jsou napsané v managed kódu. Milcore je napsaný v unmanaged kódu proto, aby byla zajištěna rychlá spolupráce s DirectX. Vše je ve WPF zobrazeno pomocí DirektůX , což umožňuje efektivní hardwarové a softwarové vykreslování. WPF slouží též k ovládání pamětí a výkonu.

2.4.1.2 Systém.Threading.DispatcherObject

Většina objektů ve WPF pochází z **DispatcherObject**, který poskytuje základní příkazy pro běh více vláken současně. WPF je založen na systému zasílání zpráv realizované dispečerem, které funguje stejně jako u známých Win32, ale ve WPF dispečer používá User32 zprávy pro vykonávání “tunelových“ volání vláken. Pro běh více vláken současně ve WPF potřebujeme dispečera a vlákno afinitu.

Nejběžnější formou, kde používat vlákno, je lokální paměť (TLS) k uložení stavu. Vlákno afinitu vyžaduje, aby každé logické vlákno vlastnilo jedno fyzické vlákno v operačním systému, který se může stát pamětí. Systémy jako OLE 2.0, schránky a Internet Explorer vyžadují provedení s jediným vláknem afinitu (STA). Jestliže máte objekty se STA provedením vláken, potřebujete způsob, jak komunikovat mezi vlákny a potvrdit, že používáte správné vlákno. Zde spočívá úloha odesílatele. Dispečer obsahuje základní zprávu pro stanovení priority fronty. Příklady zpráv zahrnují vstupní oznámení (pohyb myši), framework funkce (layout) nebo uživatelské příkazy.

2.4.1.3 Systém.Windows.DependencyObject

Jedna ze základních teorií používaná při realizaci WPF byla přednost vlastností před metodami nebo událostmi. Vlastnosti jsou deklarativní a umožňují snadněji specifikovat záměr náhradou za činnost. Tato základní teorie úmyslně vytvořila více vlastností, které slouží k tomu, aby se lépe kontrolovalo chování aplikace.

Ve snaze navázat chování na hodnoty vlastností, musíte vědět, že hodnota vlastností byla změněná. Microsoft .NET Framework má rozhraní **INotifyPropertyChange**, které umožňuje, aby objekt zveřejnil oznámení o změně (ale to není povinné).

Jak už jsem zmínil, jednou z nových myšlenek WPF je preference vlastností před metodami a událostmi. Daty řízený model upřednostňuje bohatou sadu vlastností, na které lze vázat dynamiku uživatelského rozhraní v aplikaci. Proto WPF definuje nový typ vlastnosti – *DependencyProperty*. Ačkoli se navenek nijak neliší od obyčejných vlastností, mají několik dalších nepřehlédnutelných výhod. WPF samo poskytuje tomuto typu záložní systém nahrazující zažitý způsob kombinování skrytých a veřejných členů třídy. Implementací propracovaného notifikačního systému, jehož jádrem je třída *DependencyObject* (pouze jeho potomci mohou vlastnit DependencyProperty), se stará o vyhodnocování změn hodnot. Konfigurací zvláštních parametrů (třída *PropertyMetaData*) pak můžeme určit, k jakým akcím dojde při této události.

Kromě toho lze k vlastnosti doplnit:

- přednastavenou hodnotu – předvolbu
- metodu pro validaci a metodu pro úpravu hodnoty (udržení v určitých mezích)
- metodu pro provedení dalších návazných operací

Hlavním důvodem pro existenci tohoto typu vlastnosti, je použití v oddělené definici vizuálního vzhledu.

Finální nový rys property systému je představa o připojených vlastnostech. WPF elementy jsou postavené na základě principu kompozice a opětovném využití komponentů. Často se totiž stává, že některé prvky obsahují (jako Grid layout prvek) další údaje o dítěti prvku ke kontrole jeho chování (stejně jako řádková / sloupcová informace). Místo toho, aby se sjednotily všechny tyto vlastnosti s každým prvkem, má každý objekt povoleno definovat vlastnost jiného objektu. Toto je podobné jako v JavaScriptu.

2.4.1.4 Systém.Windows.Media.Visual

V tomto systému je definováno, jak vykreslit pixely na obrazovku. *Visual* třída tvoří základ vizuálních objektů a slouží k vykreslení scény. Každé vykreslování obsahuje instrukce a metadata o tom, jak scénu vykreslit (ořezávání, transformace apod.). Visual systém je navržený tak, aby byl jednoduchý a flexibilní a je závislý na zabezpečených callback metodách.

WPF zobrazuje data do datové struktury. Tyto struktury představují hierarchické zobrazení stromu s instrukcemi vykreslení. Při programování ve WPF, lze vytvářet vizuální prvky a odvozené typy, které mohou vnitřně komunikovat o složení stromu prostřednictvím protokolu zpráv.

Je velmi důležité si uvědomit architektonický detail, že kompletní strom vizuálních a vykreslovacích instrukcí je cache instrukce. WPF používá systém renderování ke grafickému vyjádření. To je umožněno schopností systému překreslovat ve vysokých obnovovacích kmitočtech a zachováním odpovídajícího uživatelského rozhraní i při náročných výpočtech na pozadí. Toto pomáhá zabránit výskytu neodpovídající aplikace.

Další důležitý detail, který není tak úplně vidět ve schématu, je to, jak systém ve skutečnosti vykonává kompozici.

WPF používá malířský model - malířův algoritmus. To znamená, že namísto ořezávání jednotlivých složek je každá složka seřazena podle vzdálenosti od průmětny. Výhodou tohoto modelu je možnost mít komplexní, částečně transparentní tvary**.** S dnešním moderním grafickým hardwarem je tento model poměrně rychlý (což nebyl případ, kdy byly vytvořeny User32 / GDI).

2.4.1.5 Systém.Windows.UIElement

UIElement definuje základní subsystémy obsahující uspořádání (Layout), vstup od uživatele akce (Events). Layout je základní koncept ve WPF. V mnoha systémech je buď pevně stanovený layout model (HTML podporuje 3 modely rozložení…) nebo relativní model pro layout (User32 opravdu podporuje pouze absolutní umístění). WPF začalo s předpokladem, že vývojáři a designéři chtěli flexibilní, rozšiřitelný layout model, který je znovu deklarativní a ne imperativní. UIElement prochází dvěma fázemi - měřením (Measure), kde si sám určí, kolik místa potřebuje, a potom uspořádáním (Arrange), kde ho jeho rodičovský komponent umístí podle požadované velikosti a nastaveného uspořádání.

Měření (Measure) umožňuje komponentům určit, kolik místa potřebují. Měření je oddělená fáze od uspořádání (Arrange), protože tu existuje mnoho situací, kdy zdrojový element bude několikrát žádat dětský element o měření k určení jeho optimální pozice a velikosti. Skutečnost, že zdrojové prvky žádají dětské prvky o měření, ukazuje další klíčovou filozofii WPF – uspokojit velikost. Všechny kontrolky ve WPF podporují schopnost určit velikost přirozené velikosti jejich obsahu. Toto dělá lokalizaci jednodušší a umožňuje dynamické rozvržení prvků (layout) a jak změnit jeho velikost. Fáze uspořádání umožňuje rodičovským prvkům umístit a určit konečnou velikost každého dětského prvku.

Hodně času je stráveno diskutováním o výstupní straně WPF – o vizuálních a souvisejících objektech. Nicméně na vstupní straně existuje obrovské množství inovací. Pravděpodobně nejzásadnější změnou vstupního modelu pro WPF je konzistentní model, kterým jsou vstupní akce směřovány do systému.

Vstup vznikne jako signál na jádru režimu řídícího zařízení a směřuje ke správnému procesu a vlákny skrz složitý proces zahrnující jádro systému Windows a User32. Jakmile je zpráva User32 korespondující se vstupem směřována do WPF, je převedena do WPF hrubé vstupní zprávy a poslána dispečerovi. WPF dovoluje vstupním akcím (events), aby byly převedeny do vícenásobných současných akcí, umožňujících realizaci vlastností jako „MouseEnter“ na nízké úrovni systému s garantovaným doručením.

Každá vstupní akce (events) je převedena alespoň do dvou akcí – ukázka akce a aktuální akce. Všechny akce ve WPF představují směr cesty přes strom prvků. Akcím se říká „bubble“ - bublina, jestliže se pohybují od cíle vrcholu stromu ke kořeni, a „tunnel“ tunelová, jestliže začínají v kořeni a pokračují dolu do cíle. Vstup náhledových tunelových akcí umožňuje

prvkům ze stromu filtrovat akci nebo přijmout opatření. Obvykle se ale používají (nenáhledové) akce.

UIElement zavádí pojem „CommandBindings“. Příkazový systém (Command systém) WPF umožňuje vývojářům definovat funkčnost v případě příkazového koncového bodu. Command bindings umožňuje prvkům definovat zobrazení mezi pohybem vstupu (Ctrl+N) a příkazem (New). Oba vstupní pohyby a příkazové definice jsou rozšiřitelné a mohou být zaslány společně. Proto triviální, například umožňují konečnému uživateli přizpůsobit klíčové vazby (key bindings), které chce používat s aplikací.

Ve WPF jsou vlastnosti zavedené ve skupině PresentationCore. Při stavbě WPF bylo požadavkem pro výsledek úplné oddělení foundational části (jako je layout - měření a uspořádání) a framework části (jako je zavádění specifického layout-např. Grid). Cílem bylo poskytnout rozšiřitelný systém, který by umožňoval externím vývojářům vytvářet jejich vlastní framework.

2.4.1.6 Systém.Windows.FrameworkElement

Na FrameworkElement lze pohlížet ze dvou hledisek. Zavádí sadu metod a přizpůsobení systému potřebám uživatele na subsystémech, zavedených v nižších vrstvách WPF, a také zavádí sadu nových subsystémů. První metoda zavedená FrameworkElementem se týká návrhu aplikace. FrameworkElement navazuje na základní layout zavedený UIElementem a přidává pojem rozložení – layout. Vlastnosti jako HorizontalAlignment, VerticalAlignment, MinWidth, a Margin dávají všem komponentům odvozeným od FrameworkElement konzistentní chování uvnitř layout kontejnerů.

FrameworkElement také poskytuje jednodušší API expozici s více funkcemi v klíčové vrstvě WPF. Například FrameworkElement poskytuje přímý přístup do animace prostřednictvím metody BeginStoryboard. Storyboard nabízí možnost skript vícenásobné animace oproti souboru vlastností.

Dvě rozhodující věci, které FrameworkElement zavádí, jsou data binding a styly.

Subsystém data binding ve WPF by měl být poměrně známý všem, kteří užívají Windows Forms nebo ASP.NET k tvorbě aplikačního uživatelského rozhraní. V každém z těchto systémů je jednoduchá cesta vyjadřující, že chcete, aby jedna nebo více vlastností z daných základních prvků byly vázány na část dat. WPF má plnou podporu data binding, transformace a list binding (seznamů).

Jeden z nejzajímavějších rysů data binding ve WPF je zavedení šablon dat. Šablony dat vám umožní deklarativně určit, jak by část dat měla být zviditelněna. Místo tvorby obyčejného uživatelského rozhraní, které může být vázáno na data, můžete problém otočit a nechat data určit, které zobrazí.

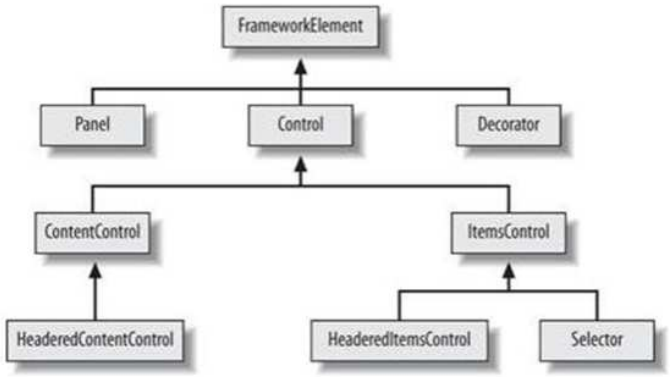
2.4.1.7 System.Windows.Controls.Control

Nejvýznamnějším rysem kontrolek jsou šablony. Pokud uvažujeme o skladbě systému WPF jako o zachovaném režimu zobrazení systému, šablony umožňují kontrolce popsat jeho převod do parametrizovaného, deklarativního stylu. ControlTemplate není opravdu nic jiného než skript pro vytvoření souboru dalších prvků, s vazbami na vlastnosti nabízených kontrolkou.

Prvek poskytuje sadu kmenových vlastností (stock properties), Foreground, Background, Padding, které pak mohou tvůrci použít k přizpůsobení zobrazení kontrolky. Realizace kontrolky poskytuje datový model a interakční model. Interakční model definuje sadu příkazů (jako Zavřít okno) a vazby na vložení pohybů (jako kliknutí na červené X v horním rohu okna). Datový model poskytuje sadu vlastností buď k přizpůsobení interakčního modelu, nebo k přizpůsobení zobrazení (určené šablonou).

Toto rozdělení mezi datový model (vlastnosti), interakční model (příkazy a události) a zobrazení modelu (šablony) umožňuje kompletní přizpůsobení systému vzhledu a chování kontrolek potřebám uživatele.

Běžný aspekt datového modelu kontrolek je obsahový model. Pokud se podíváte na kontrolku jako je Button (tlačítko), uvidíte, že má vlastnost nazvanou „Content“ - typ objektu. Ve WinForms a v ASP.NET by tato vlastnost měla obvykle být řetězcem. Bohužel limituje obsah, který můžete dát do tlačítka. Obsah tlačítka může být jednoduchý řetězec, složený datový objekt nebo kompletní strom základních prvků. V případě datového objektu je datová šablona užívaná k vytvoření zobrazení.



Obrázek 3: Hierarchie tříd

Na obrázku 3 je viditelná hierarchie tříd. Třída Control může být typu ContentControl, pokud v sobě nese nějaký obsah. Příkladem je Button (může v sebě obsahovat text, obrázek neboj jiný Control). ItemsControl naopak obsahuje více položek, představuje nějaký seznam. Selector umožňuje uživateli vybrat některé z těchto položek. Příkladem je ListBox nebo ComboBox.

2.4.1.8. Shrnutí architektury

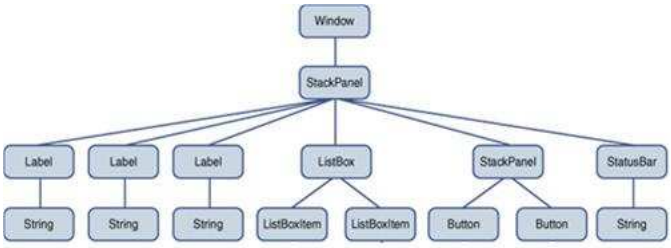
WPF je vytvořen tak, aby vám umožnil vytvořit dynamické, daty řízené návrhové systémy. Každá část systému je navržena k vytvoření objektů prostřednictvím vlastnosti sady, která řídí chování. Data binding je základní částí systému a je sjednocená na každé vrstvě. Ve WPF je všechno okolo ovládacích prvků, každé hledisko zobrazení, je vytvořeno nějakým typem data bindingu. Ve WPF můžete sestavit vlastnosti, použít objekty a navázat data více způsoby. Můžete použít také Windows Forms nebo ASP.NET. Hlubším zkoumáním architektury WPF zjistíte, že poskytuje možnost vytvoření bohatších aplikací.

2.4.2 Stromy

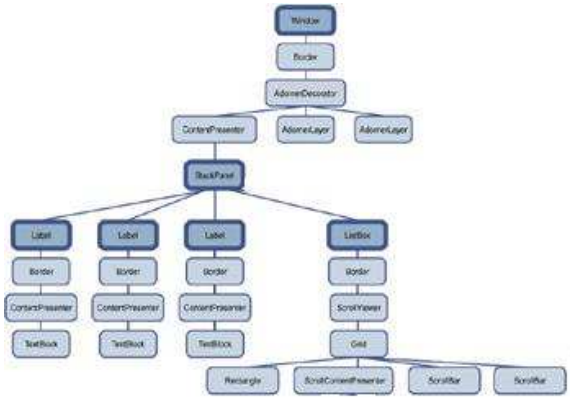
Novinkou ve WPF je transformace celého zobrazení do stromové struktury. Každé zobrazení aplikace je navrženo jako strom, který kopíruje postup zapouzdřování prvků. Má jeden kořenový element, na který se v dalších úrovních váže jeho obsah. Takhle postavená struktura je vstupním bodem pro skládání prvků a používá se i při vykreslování. WPF umí rozlišovat logický a vizuální strom. Rozdíl mezi logickým a vizuálním stromem je v chápání vlastního obsahu:

Logický strom - odpovídá zapouzdření elementů zobrazení a jejich datových zdrojů

Vizuální strom - rozkládá každý element do vizuálních primitiv nejnižší úrovně



Obrázek 4: Logický strom



Obrázek 5: Vizuální strom

Logický strom je aplikovatelný pro procházení po struktuře zobrazení a vizuální strom je zejména vstupním bodem při vykreslování obrazovky. Zpočátku bylo běžné, že objekt byl požádán, aby se vykreslil do ohraničeného prostoru a jako jediný tedy určoval výsledné zobrazení v tomto rámečku. Ve WPF je obsah postupně vykreslován v pořadí od pozadí (nejvzdálenější) k popředí (nejbližší), tudíž využívá Malířova algoritmu. Bližší vrstva se tak může transparentně zobrazit přes již vykreslený objekt v pozadí. Důležitý je v tomto případě přechod k datovému modelu. Parametry pro vykreslení se aplikují prostřednictvím nastavení odpovídajících vlastností jednotlivých prvků.

Logický strom je používán jako prostředník pro správnou funkci zmiňovaných systémů.

2.5 XAML

Objektově orientované programování je pro psaní uživatelského rozhraní velice težkopádné. Proto součástí WPF je nový programovací jazyk uživatelského rozhraní – e**X**tensible **A**pplication **M**arkup **L**anguage, který představuje značkovací jazyk pro zjednodušenou deklarativní specifikaci vzhledu aplikace, jejích jednotlivých součástí (layout prvky, ovládací prvky) a jejího obsahu.

XAML je jazyk založený na specifikaci XML dokumentů, což zrychluje zápis, usnadňuje čitelnost, přenositelnost, rozšiřitelnost a manipulaci. Další výhodou XAMLu je oddělení programové části od uživatelského rozhraní. Ve WPF aplikaci se každé okno skládá ze dvou souborů, jednoho ".xaml" a připojeném ".cs" souboru (záleží na použitém .NET jazyku).

Použitím značkovacího jazyka **XAML** získáváme některé z výhod:

- ✓ snížení nároků na úpravu jednotlivých částí
- ✓ možnost paralelně vyvíjet aplikaci za spolupráce specializovaných odborníků
- ✓ různé prostředky pro různě úrovně specializace (grafické/textové editory)
- ✓ oddělení pracovních prostorů designérů a programátorů
- ✓ okamžité zobrazení deklarace bez překladu

Uživatelské rozhraní se často stává předmětem problémů mezi grafickými specialisty a programátory. Grafik mívá většinou zajímavé a kreativní nápady, které ale mohou programátoři mnohdy jen těžce implementovat. Pro WPF je typické, že se snaží už zpočátku oddělit práci grafika od rutinního kódování programátora. Poskytuje dostupné nástroje pro komplexní definici chování a vizuální stránku elementů. Programátorům nadále zůstává k dispozici známé prostředí programovacích jazyků (např C#,VB.Net).

Již na začátku je třeba upozornit, že XAML je pouze jinou formou zápisu (zjednodušená verze pro návrháře a hodí se zejména k vytváření a inicializaci objektů). Vše z deklarace v jazyce XAML je možné stejným (a mnohdy pracnějším) způsobem zapsat i do zdrojových

souborů spravovaného kódu. Tím ale přicházíme o snadnou a efektní cestu k vytváření atraktivních designů aplikací.

2.5.1 Struktura aplikace

Zápis v jazyce XAML je založen na vnořování elementů do stromové struktury. Každá část představuje rodičovský prvek, který vkládá další prvek do stromu. Element *Application* se nachází na kořenové úrovni zobrazení a představuje vstupní bod pro spuštění programu (vyvolá okno aplikace), a také tento element poskytuje infrastrukturu pro správu zdrojů, oken, jejich navigaci, předávání parametrů, detekci a zpracování výjimek. Pro bezproblémový provoz systému nabízí možnost reagovat na aktivaci i deaktivaci aplikace nebo různé způsoby ukončení. Základním grafickým elementem pro aplikaci je třída *Window*. Představuje konturu okna se standardními ovládacími prvky pro minimalizaci, maximalizaci a ukončení. Redefinice vzhledu se využije zejména v situacích, kdy požadujeme netradiční vzhled dialogů a nápověd. Do okna se pak vkládá požadovaný obsah.

2.6 Zdroje

Pojmem „zdroje“ jsou označovány neprogramové části aplikací, mezi které zahrnujeme například texty popisků nebo obrázky. Jejich chápání a implementace jsou společné pro mnoho prvků. Jako obvykle se používají binární zdroje zapojené přímo do projektu, protože technologie je součástí .NET Framework. Práce s nimi je stejná jako v minulých verzích .NET prostředí. Lze je snadno určovat pro odlišná národní nastavení a jsou přístupné i pro XAML deklaraci.

WPF poskytuje více možností. Například pro kompletnost XAML zápisu se využívají logické zdroje, které jsou navrhovány jako slovníky, jejichž části jsou označené přidělenými klíči. Ve WPF je tato součást zabudovaná a přístupná v kódu i přes vlastnost Resources. Na rozdíl od dřívějších modelů však má každý prvek zobrazení svůj prostor pro modifikace. Zásadou je, aby každý objekt ve slovníku byl identifikovaný klíčem, jenž ale nemusí být jedinečným v celém logickém stromu. Pokud změníme objekt s konkrétním klíčem, dojde k ovlivnění jen následujících prvků stromu, protože zde účinkuje priorita pro nejbližší výskyt. Při vyhledávání klíče je procházen celý strom až do nalezení shody u některého z nadřazených (taktéž objektu Application). Díky tomuto lze upravovat vzhled aplikace i jinou a rychlejší cestou, než jsme doposud byli zvyklí. Mimo rozšiřování knihovny zdrojů nebo přepisování referencí u cílových prvků můžeme pozměnit související součásti v příslušné úrovni stromu (u vybraného prvku), čímž se vyhneme přepisování odkazů v hotových částech aplikace.

WPF ovládá dva typy modelů referencí na zdroje – StaticResource a DynamicResource. Jednoduchou substituci klíče jeho obsahem v době překladu představuje rozšíření StaticResource. V případě, že by klíč nebyl známý, bude při překladu nahlášena chyba. Aplikovat rozšíření DynamicResource je nutné při používání zdrojů připojených až při běhu aplikace, které provádí nalezení vždy, když je reference použita.

Zápis reference na barvu pozadí může vypadat následovně:

```
<Window.Resources>
      <SolidColorBrush x:Key="MyBrush" Color="Gold"/>
</Window.Resources>
…
<Button Background="{StaticResource MyBrush}" Content="Stiskni mne"/>
```

Příklad 1: Statická reference na zdroj

Požadujeme-li zohlednění změny hodnoty zdroje i za běhu aplikace, musíme použít druhé rozšíření:

```
<Button Background="{DynamicResource MyBrush}" Content="Stiskni mne"/>
```

Příklad 2: Dynamická reference na zdroj

Další znak WPF se projeví při nenalezení dynamické reference. Jelikož je dynamická reference pravým odkazem na zdrojový objekt, při změně zdroje se zaktualizují všechny dynamicky připojené prvky. Pokud se vyhledávání dynamické reference nezdařilo, WPF postoupí prázdnou hodnotu a neohlásí chybu.

Metody FindResource a TryFindResource (testuje i existenci klíče) jsou poskytovány ovládacími prvky ke zjednodušení operací v kódových jednotkách. Tyto metody pátrají po identifikacích také v celém stromu zobrazení.

Kromě seskupování definic zdrojů je úlohou slovníků poskytování postupů pro operace přidání, nalezení a odebrání objektu. Také umožňují podporu pro vzájemné vnořování, která je důležitá při složitější struktuře, kdy se slovníky sestavují pro objemnější řešení.

2.7 Vzhled

U WPF se setkáváme s novou schopností jak měnit vzhled jakékoli části zobrazení. Hlavní pokrok v možnostech tvorby vzhledu je oddělení vizuální definici od funkční. WPF nejen, že nabízí bohatší výběr vlastností, ale zavádí i zcela novou schopnost definovat vlastní strukturu či tvar prvku. U prvků WPF je rozšířená i nastavitelnost grafického vzhledu - různé barvy lze definovat pro okraj, pozadí a popředí. Barvám lze nastavit plnost nebo průhlednost a můžeme používat i barevné přechody s libovolným průběhem a počtem změn.

Vzhled prvků lze rozšířit i některým z efektů jako jsou stín, rozostření či sražení. Máme na výběr více možností jak nastavovat okraje, odsazení a zarovnání a typ fontu. Prvek lze vektorově posunout, pootočit (dle libovolně umístěného bodu), změnit měřítko či zkosení. Další možností jak změnit vzhled aplikace je pokročilejší model reprezentovaný styly a šablonami.

Určitě si umíte představit pracnost, s jakou lze nastavit stejné atributy pěti tlačítkům v jedné nabídce. Jestliže chceme nějakou vlastnost změnit, budeme muset u každého prvku přepsat potřebné hodnoty. Se styly a šablonami je to o mnoho jednodušší, jak ukáži v další části.

2.8 Vazba dat

Základním smyslem uživatelského rozhraní je ovládat zařízení při plnění úkolů. Při tomto procesu dochází k výměně údajů mezi operátorem a programem.

Aby bylo možné používat atraktivní design v souvislosti s potřebami interakce mezi uživately a technikou, poskytuje technologie WPF komplexně zpracovaný způsob vázání dat – *data binding*. Je velmi důležitý pro použitelnost deklarativního jazyka, neboť skvěle doplňuje oddělitelnost vzhledu od aplikační logiky v kategorii zobrazování dat. Přináší návrhářům mohutný nástroj k lepší práci s datovými objekty a jejich vizuálním designem.

Vázat data lze k vlastnostem typu *DependencyProperty*, resp. vlastnostem implementujícím rozhraní *INotifyPropertyChanged*. Vázat lze jak jednotlivé objekty, tak i celé seznamy – implementující rozhraní *INotifyCollectionChanged*. Obě rozhraní jsou důležitá pro správnou funkci vazby, jelikož ta musí zajišťovat správný přenos hodnot.

Při svazování musíme identifikovat zdroj a jméno datové položky. Zdroj může být nahrazen datovým kontextem, který představuje schopnost vyhledávání zdroje dat ve struktuře stromu elementů. Nemá-li vazba určen zdroj, prochází stromem a hledá první nenulový objekt *DataContext*.

```
<TextBox Text= "{Binding Path=Label, Mode=TwoWay, Source={StaticResource
                                    xdata}}"/>
```

Příklad 3: Vazba mezi ovládacím prvkem a zdrojem dat

Způsob aktualizace mezi účastníky vazby pak určuje parametr *BindingMode:*

- OneWay – jen ve směru od zdroje k cíli
- TwoWay – obousměrná aktualizace
- OneWayToSource – jen ve směru od cíle ke zdroji
- OneTime – ve směru od zdroje k cíli, a to pouze jednou

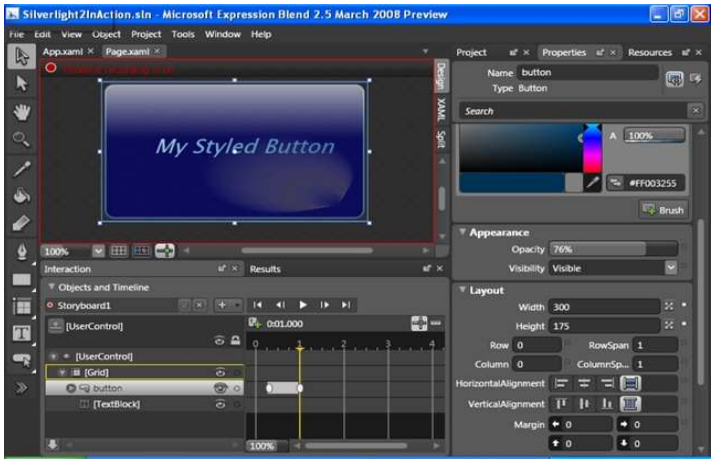
WPF poskytuje dva typy poskytovatele dat – *XMLDataProvider* a *ObjectDataProvider*. První typ představuje cestu, jak lze připojit datový zdroj ve formátu XML dokumentu, a druhý způsob zastřešuje objekty .NET Frameworku.

2.9 Nástroje

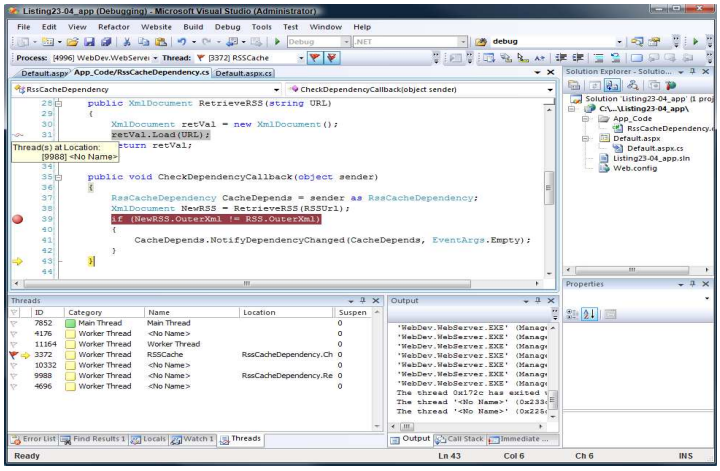
WPF je integrován do vývojového prostředí *Microsoft Visual Studio*. Po instalaci .NetFrameworku 3.0 můžeme WPF využívat i v Microsoft Visual Studiu 2005, ale kvalitnější integrace je pak ve verzi 2008, která obsahuje propracovaný editor XAML. Další produkt společnosti Microsoft – součást produktové řady Expression, *Expression Blend* (ver.1 a 2), nabízí formu vizuálního editoru společně se zobrazením deklarativního zápisu. Obdobným

grafickým editorem je Aurora od společnosti Mobiform. Pro tvorbu v jazyce XAML jsou dostupné i neplacené nástroje, například:

- XAMLPad – součást Windows SDK
- XamlCruncher
- Kaxaml



Obrázek 6: Prostředí editoru Expression Blend



Obrázek 7: Prostředí Microsoft Visual Studio 2008

3 Porovnání WinForms a WPF

3.1 Úvod

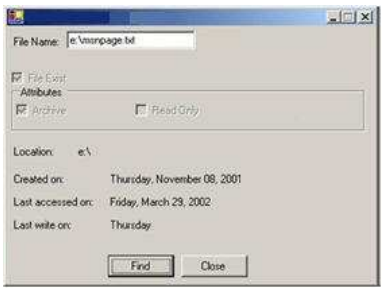
Myslím si, že WPF je očekávaným krokem vpřed ve vývoji Windows aplikací. Co se ale programování a zpracování týče, jedná se určitě o příjemné překvapení. Typická aplikace ve WinForms má alespoň jeden formulář. Bez formuláře je to jen „aplikace“, což není moc

zajímavé. Ve WinForms byly aplikace "formulářoidního" vzhledu, ve WPF můžete ale popustit uzdu fantazie a využívat plně dalších možností.

WPF navazuje na Windows Forms , více ho rozvíjí hlavně v grafických dovednostech a nabízí bohatší sadu vizuálních prvků. Nabízí nový způsob zapouzdření vykreslovací logiky včetně vizuálních efektů, předpokládané umístění na jiné vývojové platformy (WPF/E), dostupnost napojení na všechny „managed-code“ programovací jazyky a rozšíření knihoven. Veškerou práci usnadní a zrychlí vytváření uživatelského rozhraní, a dále poskytne lepší podporu pro následné úpravy. Porozumění a praktické zvládnutí technologie může poskytnout vysokou přínosnost a další řadu nových možností.

Hlavní změny oproti WinForms:

- ✓
- ✓
- ✓ **3**.1.1 Vektorová grafika - elementy jsou vektorové, nikoli bitové mapy, tedy nezávislé na rozlišení
- ✓ **Oddělení vzhledu od kódu**
- ✓ **Deklarativní programování**
- ✓ **Nastavitelnost vzhledu** - styly, šablony, triggery, události, animace, efekty, transformace, skinování a témata
- ✓ **Multimédia** - práce s audiem a videem, dokumenty
- ✓ **Interaktivní 3D aplikace** - poskytuje podmnožinu funkcí z Direct3D zaměřenou na multimédia, dokumenty a uživatelské rozhraní
- ✓ **Hardwarová akcelerace**



Obrázek 8: Aplikace ve WinForms

Obrázek 9: Aplikace ve WPF

3.1.1 Vektorová grafika

Bohatost vizuálního vzhledu by nebylo možné poskytnout bez integrace vyspělejší vektorové grafiky. Výkon výpočetních jednotek již dostačuje pro kompletní vektorové uspořádání obrazovky. WPF obsahuje podporu dvou i třírozměrné grafiky. Objektový model zapouzdřuje definici grafických primitiv, jejich seskupování a vykreslování. Všechny elementy jsou vektorové, nikoli bitové mapy, tedy nezávislé na rozlišení. Grafické tvary jsou uspořádány do stromu elementů uživatelského rozhranní**,** tudíž odpadá nutnost neustále překreslovat okna a složitě manipulovat s vykreslenou grafikou. Dále model poskytuje mnoho způsobů jak vizuální objekty dotvářet (transformovat, barvit) – v mnohém se přibližují nabídce specializovaných vektorových a modelovacích editorů. Navíc můžeme grafické objekty opatřovat i ovládacími mechanismy.

Grafický symbol může začít rotovat, přesune-li se nad něj kurzor myši, nebo vyvolat jinou akci, například po stisku tlačítka. WPF umožňuje kombinovat různorozměrné objekty v jediném zobrazení společně s prezentací aplikačních dat. Dává tak designérům obrovský potenciál uvnitř programovacího prostředí aplikace, čímž zjednodušuje přenos atraktivní grafiky do budoucích softwarových produktů.

3.1.2 Oddělení vzhledu od kódu

Výhodou použití jazyka XAML je oddělení vzhledu od kódu, který umožňuje roztřídit práci mezi uživatelskými návrháři rozhraní a programátory. Je to jeden z významných rysů WPF, jestliže vlastníte návrháře i vývojáře, potom máte velkou výhodu.

Visual Studio umožnuje jednoduše oddělit uživatelské rozhraní od kódu. Můžete nechat vývojáře pracovat na jiných kopiích a potom je sloučit, ačkoli tento proces je v některých případech trochu nešikovný.

3.1.3 Deklarativní programování

Microsoft propaguje deklarativní charakter XAML jako jednu ze svých velkých výhod, ale nevysvětluje, v čem jeho výhoda spočívá. V určitém smyslu se zdá být rozumné, že byste mohli definovat uživatelské rozhraní jen pomocí deklarativního jazyka. Avšak když využijete všechny dynamické propojení a animace, které WPF poskytuje, je jasné, že uživatelské rozhraní zvládne mnohem více.

V určitém okamžiku se může rozhraní tak zkomplikovat, že jste v podstatě donuceni sestavit ho spíše pomocí návrháře jako je Expression Blend nebo Visual Studiem než jeho psaním prostým XAML. V tomto bodě záleží na tom, zda je rozhraní napsáno v čistě deklarativním jazyce nebo je uloženo jako série příkazů pro program automaticky spouštěný při startu. V každém případě, je vaše práce hotova po navržení uživatelského rozhraní. Nemusíte psát kód, aby bylo skutečně vytvořeno.

Deklarativní programování může být hezké, ale když používáte designéry k sestavení rozhraní, je to v každém případě diskutabilní bod.

3.1.4 Nastavitelnost vzhledu

3.1.4.1 Styly

Při vytváření uživatelského rozhraní někdy potřebujeme zachovat jednotný vzhled a jsme nuceni psát některé vlastnosti pořád dokola (např. stejnou barvu nebo velikost textu). Abychom jsme se tomuto problému vyhli, můžeme použít styly.

Styly jsou pojmenovaným seskupením hodnot vlastností, které se váží na konkrétní třídu (*atribut TargetType*). Umísťují se do oblasti *Resources*, odkud je prostřednictvím klíče přiřadíme konkrétním instancím třídy:

```
<DockPanel>
    <DockPanel.Resources>
      <Style x:Key="GreenButtonStyler" TargetType="Button">
        <Setter Property="Background" Value="LightGreen" />
        <Setter Property="Margin" Value="5" />
        <Setter Property="FontWeight" Value="Bold" />
      </Style>
    </DockPanel.Resources>
    <StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
      <Button Style="{StaticResource GreenButtonStyler}">Nový</Button>
      <Button Style="{StaticResource GreenButtonStyler}">Otevřít</Button>
      <Button Style="{StaticResource GreenButtonStyler}">Uložit</Button>
      <Button Style="{StaticResource GreenButtonStyler}">Zavřít</Button>
    </StackPanel>
  </DockPanel>
```

Příklad 4: Definice pojmenovaného stylu

Obrázek 10: Pojmenovaný styl

V příkladu 4 je předvedeno, jak se používá pojmenovaný styl ovládacího prvku. Další možností je použít styl jako nepojmenovaný. Nepojmenované styly nemají jméno a jsou aplikované na všechny prvky typu uvedeného v *TargetType.*

Nepojmenovaným stylem je styl, u kterého chybí atribut *x:Key*:

```
<StackPanel>
    <WrapPanel>
      <WrapPanel.Resources>
        // Nepojmenovaný styl
        <Style TargetType="Button">
          <Setter Property="Background" Value="LightGreen" />
          <Setter Property="Width" Value="50" />
          <Setter Property="Height" Value="30" />
        </Style>
      </WrapPanel.Resources>
      <Button>Nový</Button>
      <Button>Otevřít</Button>
      <Button>Uložit</Button>
      <Button>Zavřít</Button>
    </WrapPanel>
    <WrapPanel>
      <Button>Zkopírovat </Button>
      <Button>Vložit</Button>
      <Button>Odstranit</Button>
    </WrapPanel>
  </StackPanel>
```

Příklad 5: Definice nepojmenovaného stylu

Obrázek 11: Nepojmenovaný styl

Modifikovat lze i pojmenovaný styl. Můžeme například změnit některý parametr pro určený počet prvků, ale tento způsob je pracný a pro spravování nevhodný. Model stylování ve WPF zná lepší možnost – styly je možné odvozovat (dědit). Dědičnost má význam, neboť vyjadřuje potřebu upravovat či obohacovat. Odvozovat jde pouze od stylů pojmenovaných. Použitím atributu *BasedOn* v elementu *Style* se určí, ze kterého stylu mají být vlastnosti nového stylu zděděny.

```
<Style BasedOn="{DynamicResource GreenButtonStyler}" TargetType=" Button ">
    <Setter Property="Background" Value="DarkGreen" />
</Style>
```

Příklad 6: Úprava stylu pomocí dědičnosti

Nepojmenovávání v sobě přináší jednu nevýhodu. Neuvedením identifikačního klíče přiřazujeme stylu pojmenování odpovídající typu třídy. WPF procesor přiřazuje tento styl všem prvkům tohoto typu jako předvolbu. Tento proces ale prochází celou strukturou vizuálního stromu a nepojmenovaný styl se tak objeví i na neočekávaných místech, např. přiřadí se i instancím uvnitř vizuálního stromu.

Při definici stylu je důležitý element *Setter*. Nastavuje vlastnost, identifikovanou hodnotou atributu *Property*, na hodnotu v atributu *Value*.

Myslím si, že stejně jako u moderních webových stránek se už neobejdeme bez CSS stylů, tak ani ve WPF se neobejdeme bez použití stylů, které nám ulehčují práci při psaní aplikace.

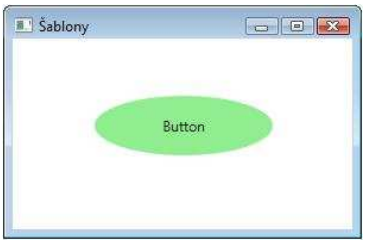
3.1.4.2 Šablony

Jednou z vlastností ovládacích prvků je šablona. Reprezentuje významný rys WPF, který designérům umožňuje úplně přepsat vzhled prvku. Šablona určuje, z jakých součástek se prvek skládá a jak jsou vůči sobě rozloženy.

Na příkladu je ukázka kulatého tlačítka:

```
<Button Width="150" Height="50">
    <Button.Template>
      <ControlTemplate>
        <Grid>
          <Ellipse Fill="LightGreen" />
          <Label VerticalAlignment="Center"
                 HorizontalAlignment="Center"
                 Content="Button" />
        </Grid>
      </ControlTemplate>
    </Button.Template>
  </Button>
```

Příklad 7: Zápis šablony „kulatého“ prvku



Obrázek 12: Šablona kulatého tlačítka

Na příkladu7 je dobře zřetelné oddělení funkční části. V šabloně nastavíme vnitřní obsah, který napojíme na hodnoty vlastností deklarovaných typem prvku. V příkladu je povinný element *ControlTemplate*, do kterého vložíme prvky, ze kterých se má naše kontrola nově skládat.

Výhodné na tomto rozdělení je možnost jeden funkční celek zobrazovat více způsoby s různými styly i rozsahem chování. Na jednom programovém základu designér navrhne sadu různých komponent rozlišených mírou znalosti, zpřístupněné funkcionality a požadovaných parametrů. Je to snadná cesta pro rozšiřování a vylepšování prvků bez nebezpečí zanesení chyb do již existujících návrhů. Šablona je jednou z vlastností každého ovládacího prvku.

Obdobou šablon jsou datové šablony (typ *DataTemplate*). Jedná se o prostředek, který dovoluje designérům modifikovat způsob vizuálního zobrazení datových objektů. Datovou šablonu přisuzujeme konkrétnímu typu atributem *DataType*.

Oba typy šablon mohou být pojmenované i nepojmenované. Šablony prvků se tak mohou aplikovat na všechny instance konkrétního typu ovládacího prvku, datové šablony se aplikují na všechny instance typu specifikovaného datového objektu.

3.1.4.3 Triggery

Vlastnosti, styly nebo šablony jsou zatím jen ve statické podobě, která nenabízí žádnou schopnost reagovat na události uživatelského prostředí. Měli bychom vědět, že ovládací prvky obsahují sadu vlastností notifikující systém o jejich změnách, které nastávají nejen pomocí uživatelských vstupů. Pro zvýraznění změn vlastností WPF obsahuje triggery. Triggery jsou součástí stylů i šablon a dokonce i ovládacích prvků.
Triggery rozlišujeme na:

- *Property triggers* - reagují na hodnotu vlastnosti (property) kontroly
- *Data triggers* - reagují na hodnotu proměnné
- *Event triggers* - reagují na události

Property trigry reagují pouze na hodnoty *dependency properties* (vlastnosti) kontroly. V *property* triggrech můžeme nastavit hodnoty vlastností pomocí elementu *Setter* a nebo pracovat s animacemi.

Příklad: Chceme, aby po najetí myší nad tlačítko byl text červený a zvýrazněný.

```
<StackPanel>
    <StackPanel.Resources>
      <Style TargetType="Button">
        <Style.Triggers>
          <Trigger Property="Button.IsMouseOver" Value="True">
            <Setter Property="Button.FontWeight" Value="Bold" />
            <Setter Property="Button.Foreground" Value="Red" />
          </Trigger>
        </Style.Triggers>
      </Style>
    </StackPanel.Resources>
      <Button>Nový</Button>
      <Button>Otevřít</Button>
      <Button>Uložit</Button>
      <Button>Zavřít</Button>
  </StackPanel>
```

Příklad 8: Property trigger najetí kurzoru myši

Obrázek 13: Property trigger najetí kurzoru myši

Nabude-li vlastnost specifikované hodnoty, nastaví se všechny vlastnosti uvedené uvnitř bloku. Takto napsaný zápis obsahuje další velkou výhodu. Trigger totiž představuje akci, která pouze změní odpovídající hodnoty. Dojde-li k zneplatnění jeho podmínky, nemusíme navracet původní hodnoty, protože WPF provede zpětné nastavení samo. Pokud bude potřeba více vstupů, existuje třída *MultiTrigger*, která se stane aktivní, platí-li všechny podmínky v seznamu.

Aby byla schopnost definice vzhledu skutečně komplexní, poskytuje WPF i zvláštní trigger pro vlastnosti typově různé od DependencyProperty. *Data triggry* nám umožní reagovat na hodnotu proměnných v seznamech, jako je například ListBox.

Potřebuje-li designér reagovat na vznik události, použije *EventTrigger*:

```
<EventTrigger RoutedEvent="Button.MouseEnter">
            <EventTrigger.Actions>
              <BeginStoryboard>
                <Storyboard>
                  <DoubleAnimation To="30"
                                   Storyboard.TargetProperty="Height"
                                   Duration="0:0:0.5" />
                </Storyboard>
              </BeginStoryboard>
            </EventTrigger.Actions>
          </EventTrigger>
```

Příklad 9: Trigger události roztáhnoutí prvku

3.1.4.4 Události

WPF zavádí i neobvyklý typ pro události – *RoutedEvent*. Tyto události také využívají existenci stromové struktury zobrazení k tomu, aby šlo vytvářet jakoukoliv událost na jakémkoliv prvku. Hlavní myšlenkou jejich přínosu je opět použití při nastavení vzhledu. Událost jako vlastnosti musí připadat konkrétnímu vlastníku a při její deklaraci je také nutné určit směr jejího šíření logickým stromem:

- Ve směru od zdroje ke kořenovému elementu zobrazení (Bubble)
- Ve směru od kořene zobrazení ke zdroji události (Tunnel)
- Bez šíření stromem zachovávajíc ostatní výhody (Direct)

Úprava události se zajišťuje pomocí zástupce, u kterého jde především o typ parametru, jež probíhá společně s událostí. V původní verzi parametr přenáší zdroje vzniku (element logického stromu), zdrojový grafický primitiv (element vizuálního stromu) a typ události. Součástí parametru je také atribut *Handled*, který podává informaci o tom, zda již byla událost vytvořená. Aby byla funkce stálá, je nabízena schopnost reagovat na ni libovolnému počtu objektů ve stromu elementů. Subsystém WPF zajistí průchod celým stromem. Tento rys poskytuje možnost sledování děje v podstromu elementu a následné vizuální reakce při zaznamenání sledované akce.

3.1.4.5 Animace

Nepostačuje-li designérovi k provedení dynamiky rozhraní skoková změna hodnot vlastností, může využít dalšího nástroje WPF – *animace*. Vložit prvek, pulzovat barvou pozadí či zabavit uživatele může návrhář v propracovaném systému vektorově definovaných *scénářů*. Přestože příručky pro designéry varují před animacemi, které ruší soustředění, aplikace ve WPF mají být atraktivní a tudíž se v nich s animacemi počítá. Navíc přiřazovat animace můžeme jednotlivým ovládacím prvkům, stylům i šablonám. Vytváření animací je velice jednoduché, dokonce ve většině případů si vystačíme pouze s XAMLem. Ve WPF ---pracujeme s animacemi jinak, než možná čekáte, nevytváříme a nespravujeme časové osy, dokonce ani neanimujeme jednotlivé kontroly, ale ve WPF vytváříme animace prostřednictvím vlastností typu DependencyProperty, kdy v definovaném čase mění jejich hodnoty.

Animovat lze skoro každou vlastnost – barvu, velikost, pozici, orientaci či velikost. Základní animaci nastavujeme pouze počáteční, koncovou hodnotu a délku trvání. Průběh změn je standardně lineární, pochopitelně nastavit jde i nelineární posun po křivce. Pro rozsáhlejší efekty použijeme animaci s klíčováním, kde zadáváme sled hodnot v navazujících časových okamžicích. Především pak pro třírozměrné scény máme k dispozici i animaci podle geometrické prostorové křivky. Součástí WPF je i objektový model animací, pomocí něhož mohou vývojáři vytvořit vlastní typy animací.

3.1.4.6 Skinování a témata

Technologie WPF poskytuje způsob pro uživatelskou přizpůsobitelnost vzhledu – *skiny*. Strukturováním slovníku zdrojů a pečlivým dynamickým napojením všech vzhledových vlastností na jeho objekty můžeme za běhu aplikace měnit parametry zobrazení. Jejich změny promítneme do aplikace náhradou konkrétních objektů, což způsobí automatickou aktualizaci všech postižených elementů.

Součástí definice vzhledu je možnost sestavovat *témata*. Představují komplexní vizuální styl zobrazení všech elementů aplikace. Uživatel může definovat jakékoli téma ve vzhledových parametrech systému, kde se témata automaticky rozeznávají podle operačního systému Windows:

- **Classic - Windows 2000**
- **Luna - Windows XP**
- **Aero - Windows Vista**

3.1.4.7 Efekty

WPF poskytuje velké množství efektů (stíny, záře, odlesky, rozostření, sražení…), které bychom nejspíše hledali v 3D grafice a u WinForms by se vytvářeli těžce nebo vůbec. Efekty vypadají úžasně, nejsou náročné na výkon a jsou zpestřením pro každou aplikaci. Všechny efekty se nacházejí v namespace System.Windows.Media.Effects, kde je i jejich společná abstraktní třída BitmapEffect.

BevelBitmapEffect

Vytvoří sklon, který sníží nebo zvýší povrch daného elementu.

Obrázek 14: Efekt BevelBitmapEffect

BlurBitmapEffect

Simuluje pohled na element přes nezaostřený objektiv, element je rozmazaný. Technika a míra rozmazaní jsou nastavitelné.

Obrázek 15: Efekt BlurBitmapEffect

DropShadowBitmapEffect

Vytvoří stín za elementem. Tím simuluje situaci, kdy jakoby element vrhal stín při určitém umístění světelného zdroje, a vytváří dojem 3D. Velikost, barva, umístění stínu jsou nastavitelné.

Obrázek 16: Efekt DropShadowBitmapEffect

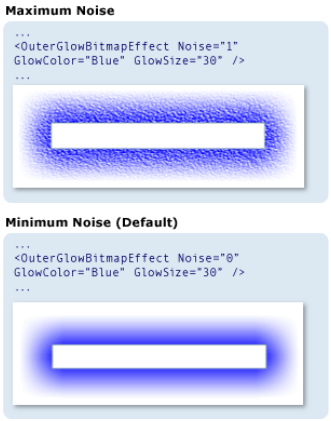
EmbossBitmapEffect

Emboss způsobí, že element vypadá na některých místech vyvýšený, někde zase snížený. Vytváří reliéf. Lépe než na příkladě Buttonu je vidět tento efekt na objektu Image.

Obrázek 17: Efekt EmbossBitmapEffect

OuterGlowBitmapEffect

Obrázek 18: Efekt OuterGlowBitmapEffect



Obrázek 19: Efekt Efekt OuterGlowBitmapEffect

BitmapEffectGroup

Obrázek 20: Efekt BitmapEffectGroup

V tomto příkladu si ukážeme, jak vytvořit ovladací prvek používající efekty jako je odlesk, vykreslení pozadí a 3D vzhled. Nejdříve v okně, kde chceme používat tlačítko Button definujeme jeho styl, který bude obsahovat Trigger měnící barvu tlačítka při přechodu a stlačení myší.

Ve stylu definujeme vlastní šablonu pro prvek Button, která je složená z několika obdélníků sloužících pro vykreslení pozadí a 3D vzhledu.

```
<Window.Resources>
  <Style TargetType="{x:Type Button}">
  <Setter Property="Foreground" Value="White" />
  <Setter Property="Background" Value="Black" />
  <Setter Property="Margin" Value="l" />
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type Button}">
        <Grid>
          <Rectangle x:Name="GelBackground" Opacity="l" RadiusX="9 RadiusY="9"
                   Fill="{TemplateBinding Background}" StrokeThickness="0.35">
            <Rectangle.Stroke>
              <LinearGradientBrush StartPoint="O,0" EndPoint="0,1">
                <GradientStop Color="White" Offset="0" />
                <GradientStop Color="#666666" Offset="l"/>
              </LinearGradientBrush>
            </Rectangle.Stroke>
          </Rectangle>
         <Rectanglex:Name="GelShine"Margin="2,2,2,0"VerticalAlignment="Top"
          RadiusX="6"RadiusY="6"Opacity="l" Stroke="Transparent" Height="15px">
            <Rectangle.Fill>
              <LinearGradientBrush StartPoint="0,O" EndPoint="O,1">
                <GradientStop Color="#ccffffff" Offset="0"/>
                <GradientStop Color="Transparent" Offset="l">
               </LinearGradientBrush>
            </Rectangle.Fill>
          </Rectangle>
          <ContentPresenter VerticalAligrunent="Center" HorizontalAlignment=""
        <ControlTemplate.Triggers>
          <Trigger Property="IsMouseOver" Value="True">
            <Setter Property="Fill" TargetName="GelBackground">
              <Setter.Value>
                <RadialGradientBrush>
                  <GradientStop Color="Lime" Offset="0" />
                  <GradientStop Color="DarkGreen" Offset="l" />
                </RadialGradientBrush>
              </Setter.Value>
          </Trigger>
          <Trigger Property="IsPressed" Value="true">
            <Setter Property="Fill" TargetName="GelBackground">
              <Setter.Value>
                <RadialGradientBrush>
                  <GradientStop Color="#ffccOO" Offset="0"/>
                  <GradientStop Color="#cc9900" Offset="l"/>
                </RadialGradientBrush>
              </Setter.Value>
            </Setter>
          </Trigger>
        </ControlTemplate.Triggers>
```

```
      </ControlTemplate>
    </Setter.Value>
  <Style.Triggers>
    <Trigger Property="IsMouseOver" Value="True">
      <Setter Property="Foreground" Value="Black"/>
    </Trigger>
    <Trigger Property="IsPressed" Value="True">
      <Setter Property="Foreground" Value="Black"/>
    </Trigger>
  </Style.Triggers>
  </Style>
</Window.Resource>
```

Příklad 10: Prvek Button

Prvek *Button* umístíme do mřížky, ve které jsou dva řádky. V jednom je samotné tlačítko Button a v druhém jeho odlesk. Ten vytváříme pomocí čtverce, který na pozadí má VisualBrush zobrazující naše tlačítko Button otočené horizontálně. Čtverec má navíc ještě definovanou vlastnost *OpacityMask* pro vytvoření efektu odlesku. Samotný Button používá také některé bitmapové efekty.

```
<Grid Height="60"Width="125">
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Button Name="button" FontWeight="Bold">
    <Button.BitmapEffect>
      <BitmapEffectGroup>
        <DropShadowBitmapEffect/>
        <BevelBitmapEffect/>
      </BitmapEffectGroup>
    </Button.BitmapEffect> Click Me!
  </Button>
  <Rectangle Grid.Row="l">
    <Rectangle.Filll>
      <VisualBrush Visual="{Binding ElementName=button}">
        <VisualBrush.Transform>
          <TransformGroup>
            <ScaleTransform ScaleY="-l"/>
            <TranslateTransform Y="30"/>
          </TransformGroup>
        </VisualBrush.Transform>
      </VisualBrush>
      </Rectangle.Fill>
      <Rectangle.OpacityMask>
        <LinearGradientBrush StartPoint="0,O" EndPoint="0,1">
          <GradientStop Color="#55000000" Offset="l"/>
          <GradientStop Color="#FF000000" Offset="0"/>
        </LinearGradientBrush>
      </Rectangle.OpacityMask>
    </Rectangle>
</Grid>
```

Příklad 11: Prvek Button s efektem odlesk

Obrázek 21: Prvek button

Obrázek 22: Prvek button-přechod myší

Obrázek 23: Prvek button kliknutí

Efekty se dají bez problémů kombinovat. Na obrázku 23, 24, 25 vidíme prvek *Button*, na kterém je aplikovaný DropShadowBitmapEffect a BevelBitmapEffect, přičemž na polovinu jeho textu je použitý BlurBitmapEffect.

3.1.4.8 Transformace

WPF používá DirectX, který umožnuje transformovat cokoliv v uživatelském rozhraní. To znamená, že můžete překládat, natáhnout, ovládát a otočit výkresy nebo text poměrně snadno. Například podle obr. 26 je použita jednoduchá transformace pootočení tlačítka o 30 stupňů a další.

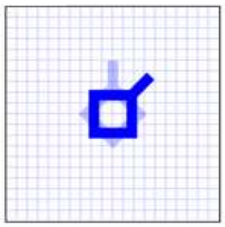
Obrázek 24: Transformace ve WPF

Transformace nevypadají až tak efektně, ale jsou také velmi užitečné a v málokterém frameworku najdeme něco podobného. Nachází se v namespace System.Windows.Media, kde je jejich společná třída Transform.

Následně uvedu některé z druhů transformací:

RotateTransform

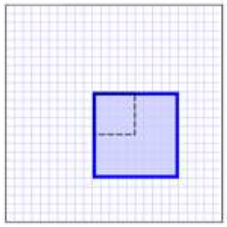
Jednoduchá transformace, kde je element otočený o určitý uhel. Možné je zvolit samotný úhel otočení jako bod elementu, vzhledem ke kterému se otáčí.



Obrázek 25: Transformace RotateTransform

ScaleTransform

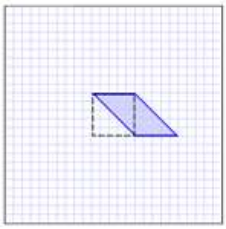
Představuje zvětšení elementu. Pokud je WPF plně vektorové a používá vyhlazování hran (Anti-Aliasing), všechny hrany jsou pěkné a nedochází k žádné deformaci.



Obrázek 26: Transformace ScaleTransform

SkewTransform

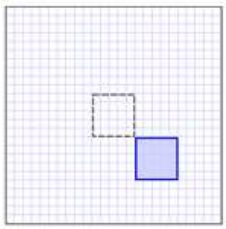
Transformace zešikmující element. Míra zešikmení je nastavitelná.



Obrázek 27: Transformace SkewTransform

TranslateTransform

Jednoduché posunutí, vhodné pokud chceme element umístit mimo rozsah jeho rodičovského elementu.



Obrázek 28: Transformace TranslateTransform

MatrixTransform

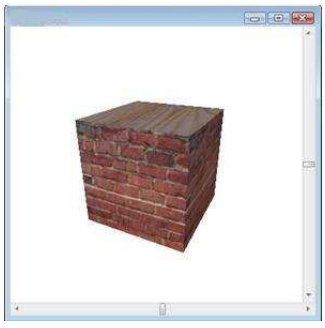
Transformace určená maticí, která slouží k vytváření a definovaní vlastních transformací.

TransformGroup

Ve WPF je možné na jeden element použít i více transformací najednou. Pro tento účel slouží právě TransformGroup.

3.1.5 Interaktivní 3D scény

Jak už víme, jednou z klíčových komponent WPF je i DirectX, který je primárně určený na práci s multimédii. Jeho nejpoužívanější částí je Direct3D, rozhraní pro práci s 3D grafikou. Pomocí tohoto rozhraní WPF umožňuje jednoduchým stylem vytvářet 3D scény. Direct3D je vysoce výkonný třídimenzionální vykreslovací nástroj. A když říkám "vysoce výkonný“, myslím to vážně. Pomocí Direct3D může i můj 1,83 GHz dvoujádrový notebook čerpat desítky tisíc dekorativně upravených trojúhelníků pohybujících se v reálném čase. WPF využívá Direct3D k relativně jednoduchému vykreslení třídimenzionálních scén. Kreslení scén s Direct3D je dost složité. WPF poskytuje objekty, které výrazně zjednoduší kreslení, ale i přesto je to stále složité. Velkou slabinou v Direct3D je, že vyžaduje, abyste zjistili, co grafiký hardware uživatele obsahuje a dokáže využít. Direct3D vás přinutí pracovat se spoustou neobvyklých speciálních případů (například pokud grafické zařízení není k dispozici) a nechá vás vybudovat složité datové struktury.



Obrázek 29: 3D krychle

Obrázek 31 ukazuje poměrně jednoduchý program pro zobrazení kostky, která má cihlový a dřevěný povrch. Program nabízí horizontální a vertikální posuvníky, takže uživatelé mohou změnit úhel pohledu. Pohybování posuvníky otáčí zobrazení kamery kolem osy X a Y podle úhlů, které jsou vázány k „Hodnotě vlastnosti“ posuvníků.

Domnívám se, že vytvoření třídimenzionálních modelů ručně je poměrně zdlouhavé a složité. Například kreslení kostky zobrazené na obr. zabralo 358 řádků XAML kódu, což zde není vidět.

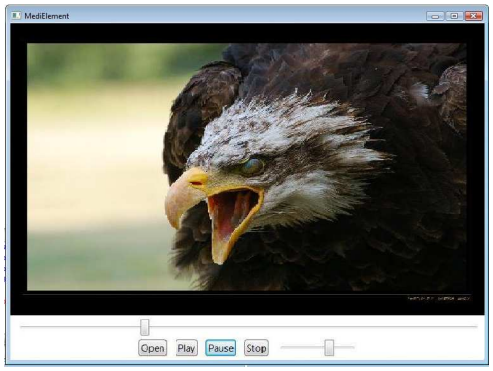
Designér může kombinovat 2-D a 3-D grafiku ke zvýšení atraktivity aplikace či výmluvnosti zobrazovaných dat. Základní ovládací prvek pro zobrazení 3D je *Viewport3D*. Zapouzdřuje celou třírozměrnou scénu a snímání kamerou (pohled na scénu), prostředky pro modelování objektů, jejich nasvícení a opatření materiálem jsou jeho vnořenými elementy. WPF umožňuje však zobrazit i běžné ovládací prvky v trojrozměrném prostoru. Díky tomu můžeme vytvářet futuristické aplikace, například přehrávač videa.

3.1.6 Multimédia

Jedním ze způsobů, jak více obohatit aplikace, je prostřednictvím využívání audiovizuálních médií. S novým přístupem k designu uživatelského rozhraní souvisí i podpora médií jako jsou obrázky, videa a audio. WPF integruje práci s grafickými daty. K zobrazování běžných formátů přidává i jejich zápis a konverzi. U obrázků můžeme využít nabízenou řadu úprav a transformací, včetně nedestruktivního zvětšování, ořezávání a rotace. Navíc šetří systémové prostředky schopností dekódovat obrázek pouze na požadovanou velikost, aniž by se v paměti aplikace musel udržovat originál v plné velikosti. WPF nabízí větší podporu obrazových formátů např. BMP, JPEG, JPG, PNG, TIFF, Windows Media Photo, GIF a ICON a zpracování obrazu, změnu jasu, kontrastu, vyvážení, barev, rozmazání.

Obrázek 30: Práce s obrázky ve WPF

WPF nám umožňuje integrovat zvuk a video do svých aplikací s cílem zvýšit uživatelský komfort. Pro práci s video a audio nahrávkami slouží prvek *MediaElement*, který podporuje audio formáty (WMA, MP3) a video formáty (AVI, MPG, MPEG, WMV).



Obrázek 31: Ovládací prvek MediaElement

WPF pro práci s dokumenty podporuje tři typy dokumentů: Flow dokumenty, Fixed dokumenty XML a XPS dokumenty. Dále také poskytuje služby jako tvorba, prohlížení, úpravy a tisk dokumentů.

Flow Dokumenty

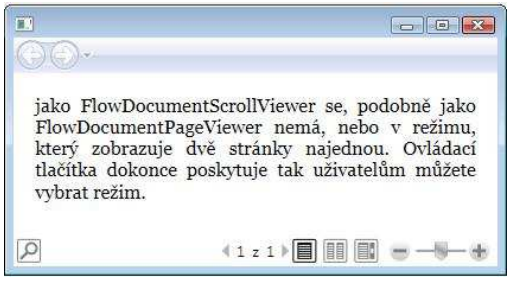
Flow dokumenty jsou navrženy tak, aby optimalizovaly prohlížení a čtení, které se dynamicky upravují a aktualizují obsah při velikosti okna a zobrazení změny nastavení. FlowDocument obsahuje text v různých stylech podle volného místa. Může obsahovat také objekty, jako jsou obrázky nebo dokonce WPF ovládácí prvky.

WPF nabízí tři ovládací prvky pro zobrazení FlowDocuments: *FlowDocumentScrollViewer*, *FlowDocumentPageViewer* a *FlowDocumentReader*

Následující příklad FlowDocument

```
<FlowDocument
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Paragraph>
      jako FlowDocumentScrollViewer se,podobně jako FlowDocumentPageViewer
       nemá,nebo v režimu, který zobrazuje dvě stránky najednou.
      Ovládací tlačítka dokonce poskytuje tak uživatelům můžete vybrat režim.
    </Paragraph>
</FlowDocument>
```

Příklad 12: FlowDocument XAML



Obrázek 32: FlowDocumentReader ve WPF

Fixed Dokumenty

Fixed dokumenty jsou určeny pro aplikace, které vyžadují přesné "co vidíte, je to, co dostanete" prezentace, zejména s ohledem na tisk. Typické použití pevných dokumentů je v DTP, zpracování textu a formu uspořádání.

Fixed dokumenty udržují přesné uspořádání obsahu v zařízení. Například Fixed dokument, který zobrazí displej na 96 bodů-per-inch (dpi) se zobrazí stejně, jako když je vytištěn buď na 600 dpi laserové tiskárně nebo 4800 dpi foto (typesetter)sazeči. Rozvržení zůstává stejné ve všech případech, i když v tomto dokumentu kvalita závisí na schopnostech jednotlivých zařízení.

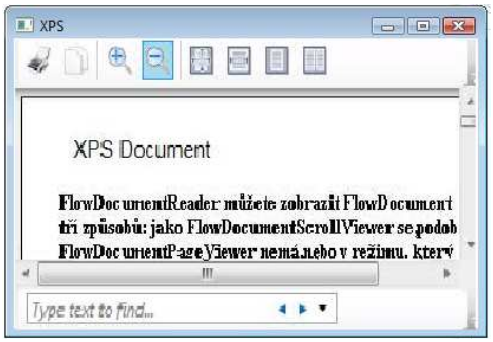
Dokumenty XPS

XPS dokumenty jsou založené na XML, které je v podstatě stránkovaným zastoupím elektronického papíru. XPS je otevřený dokument ve formátu, který je navržen tak, aby se usnadnilo vytváření, sdílení, tisk a archivaci stránkovaných dokumentů.

Důležité prvky z XPS technologie:

- XPS dokumenty jsou zabaleny jako *ZipPackage* soubory
- Hostování v prohlížeči.
- Manuální vývoj a manipulace s dokumenty z XPS WPF aplikací.
- Vysoce kvalitní renderování se zaměřením na maximální kvalitu výstupního zařízení.
- Windows Vista zařazování tisku.

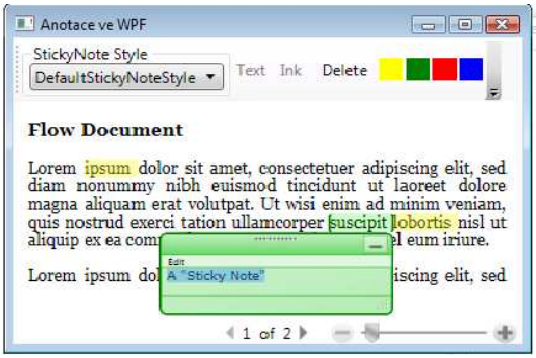
Následující obrázek ukazuje XPS dokument, který se zobrazí pomocí ovládacího prvku *DocumentViewer*.



Obrázek 33: XPS Dokument

Anotace

Anotace jsou poznámky či připomínky, které se přidávají k dokumentům nebo chtějí vyzdvihnout zájem pro pozdější reference. Ačkoli psaní poznámek do dokumentů je snadné, schopnost "psát" poznámky do elektronické podoby je často omezené nebo nedostupné. Ve WPF, ale je anotační systém, kde je poskytnuta podpora poznámek. Navíc, tyto anotace můžou být použité v ovládacím prvku *DocumentViewer*, jak ukazuje následující obrázek.



Obrázek 34: Anotace

3.1.7 Hardwarová akcelerace

Ve WPF jsou aplikace renderovány na grafické kartě, čímž jsou “rychlejší”, ale zároveň i méně zatěžují procesor. To ale neplatí vždy. Renderování neprobíhá pokaždé na grafické kartě, záleží totiž na tom, co všechno daná grafická karta umí. Někdy se může jednat o starší kousek hardwaru, jindy zase grafická karta nenabízí všechno, co WPF potřebuje, což způsobí, že na některých počítačích vaše aplikace bude pouze z části renderována na grafické kartě, v tom horším případě celá procesorem počítače. Proto rozlišujeme 3 stupně renderování:

- ✓ aplikace je renderována na procesoru (nejpomalejší)
- ✓ část je renderována na grafické kartě, část na procesoru
- ✓ renderována na grafické kartě (samozřejmě nejrychlejší)

Jak zjistíme, jak je aplikace renderována a na čem to závisí? Způsob renderování je, jak už jsem napsal, závislý na schopnostech grafické karty počítače. Aby celá aplikace byla renderována grafikou, musí podporovat všechny funkce od DirectX 7 až po DirectX 9 “hardwarově”. Také záleží na verzi Vertex a Pixel shaderu karty.

3.2 Ovládací prvky

Termín *ovládací prvek* v grafickém subsystému WPF má oproti WinForms poněkud užší význam. V technologii WinForms se například za ovládací prvek jistého typu považuje vše, co je zobrazeno na obrazovce. Ve WPF je tento termín vyhrazen pro prvky, které jsou

interaktivní, tj. zpravidla poskytují uživateli určitou zpětnou vazbu na zmáčknutí myši nebo stisknutí klávesové zkratky. Ovládací prvky uživatelský vstup aktivně sledují a zpracovávají.

Ovládací prvky můžeme zjednodušeně rozdělit na ovládací prvky, které dělají všechno samy, na standardní ovládací prvky, jako je textové pole (TextBox), a na uživatelské ovládací prvky (ovládací prvky, které obsahuji jiné ovládací prvky).

3.2.1 Standardní ovládací prvky

Ovládací prvek (*control*) je nějaká třída, která je odvozena ze základní třídy System.Windows.Forms.Control (u WinForms) nebo System.Windows.Controls (u WPF).

Ovládací prvky se dají dále rozčlenit do několika dalších kategorií:

- ✓ **Akční ovládací prvky**
- ✓ **Hodnotové ovládací prvky**
- ✓ **Ovládací prvky pro seznam**
- ✓ **Ovládací prvky pro správu velikosti a rozmístění prvků**

3.2.1.1 Akční ovládací prvky

Akční ovládací prvky jsou tlačítko, panel nástrojů, hlavní nabídka a kontextová nabídka - *Button*, *ToolBar*, *MainMenu* (WinForms), *Menu* (WPF) a *ContextMenu*. Tyto ovládací prvky mají ve WinForms i WPF téměř stejný vzhled. Existují proto, aby uživatel mohl tím, že na ně klikne, spustit v aplikaci nějakou akci. Každá z dostupných akcí má nějaký popisek, a v případě panelu nástrojů může mít i nepovinný obrázek. Hlavní událostí akčních ovládacích prvků je událost *Click*.

S výjimkou tlačítka (Button) jsou všechny ostatní ovládací prvky ve skutečnosti kontejnery pro více podobjektů, s nimiž teprve uživatel komunikuje. Například objekt MainMenu obsahuje jeden nebo více objektů MenuItem, jeden pro každý prvek nabídky, který může spustit událost Click.

3.2.1.2 Hodnotové ovládací prvky

Hodnotové ovládací prvky tvoří sadu ovládacích prvků, které slouží k zobrazení (a někdy také k editování) jediné hodnoty. Dají se dále rozčlenit podle datového typu hodnoty:

- **Řetězcové hodnoty -** popisek simulující hypertextový odkaz, textové pole, textové pole s formátováním a stavový řádek
- **Číselné hodnoty -** číselník, vodorovný a svislý posuvník, ukazatel průběhu a „reostat“

- **Booleovské hodnoty** - zaškrtávací políčko a přepínač
- **Datum a čas** - výběr data a času, několikaměsíční kalendář
- **Grafické hodnoty** - obrázek, náhled před tiskem

Řetězcové hodnoty

Ovládací prvky pro řetězec vystavují vlastnost Text, která obsahuje hodnotu ovládacího prvku v řetězcovém formátu. Ovládací prvek Label (popisek) pouze zobrazí text. Ovládací prvek *LinkLabel* zobrazí text tak, jakoby to byl odkaz HTML, a při kliknutí se také jako odkaz otevře. Ovládací prvek StatusBar (ve WPF i WinForms) zobrazí text stejně jako Label (až na to, že je stavový řádek standardně přichycen ke spodní straně svého kontejneru), ale také umožňuje rozčlenit text do několika panelů.

Nejjednodušším prvkem pro zobrazení kratších textů ve WPF je prvek TextBlock. Umožňuje bohatší formátování, ale je použitelný jen pro čtený text.

```
<TextBlock Margin="0,0,-417,0">
  <Run FontWeight="Bold" FontSize="14" Text="Toto je ukázka prvku TextBlock ve
WPF. " />
  <Run FontStyle="Italic" Foreground="Red" Text="Toto je červený text. " />
  <Run FontStyle="Italic" FontSize="18" Text="Tady je linear gradient text.">
       <Run.Foreground>
              <LinearGradientBrush>
                   <GradientStop Color="Green" Offset="0.0" />
                   <GradientStop Color="Purple" Offset="0.25" />
                   <GradientStop Color="Orange" Offset="0.5" />
                   <GradientStop Color="Blue" Offset="0.75" />
              </LinearGradientBrush>
       </Run.Foreground>
     </Run>
    <Run FontStyle="Italic" Foreground="Green" Text="Jak se Vám to líbí? " />
   </TextBlock>
```

Příklad 13:Prvek TextBlock

Obrázek 35: Ovládaci prvek TextBlock ve WPF

TextBox u obou technologií umožňuje uživateli kromě zobrazení textu i text editovat, a to buď v jednořádkovém, nebo ve víceřádkovém režimu. Ovládací prvek RichTextBox umožňuje editovaní podobně jako TextBox, ale podporuje také data RTF (Rich Text Format), což zahrnuje formátování různými typy písma a barvami a také grafiky. Nezbytným prvkem pro

zabezpečené zadání citlivých údajů, který najdeme ve WPF, je *PasswordBox* (namísto znaků zobrazuje substituční symbol).

Číselné hodnoty

Všechny ovládací prvky pro číselné hodnoty vystavují číselnou vlastnost *Value*, jejíž hodnota může být v rozsahu od hodnoty vlastnosti Minimum až po hodnotu vlastnosti Maximum. Rozdíl v nich je jenom v tom, jaké uživatelské rozhraní chcete uživateli zobrazit.

U WinForms jsou to ovládací prvky např. *NumericUpDown, HScrollBar, VScrollBar, ProgressBar, TrackBar* a ve WPF *ScollBar, ScollViewer, ProgressBar, Slider.*

Booleovské hodnoty

Ovládací prvky u WPF i WinForms CheckBox a RadioButton mají vlastnost Checked, která odráží skutečnost, jestli jsou zaškrtnuté nebo ne. Oba ovládací prvky lze ještě nastavit na třetí (indeterminate) stav, který vystavuje vlastnost CheckedState. Pokud se Checked změní, spustí se události CheckedChanged a CheckStateChanged.

Datum a čas

Ovládací prvky pro datum a čas ve WinForms umožňují uživateli pohodlně vybírat jednu nebo více instancí typu DateTime. MonthCalendar umožňuje uživateli zvolit si počáteční a koncové datum přes vlastnost SelectionRange (signalizuje to událost DateChanged). DateTimePicker umožňuje uživateli zadat jediné datum a čas, což se vystavuje vlastnosti Value (a signalizuje to událost ValueChanged). Ale ve WPF jsem bohužel tyto ovládací panely nenašel.

Grafické hodnoty

U WinForms ovládací prvky (PictureBox, PrintPreviewControl) pro grafické hodnoty zobrazují obrázky, ale ani jeden z nich nepovoluje obrázky měnit. Ovládací prvek PictureBox zobrazí jakýkoliv obrázek nastavený ve vlastnosti Image. *PrintPreviewControl* zobrazí náhled dat z objektu PrintDocument určených k vytištění. WPF je inovační technologie a proto nabízí i ovládací prvky pro práci s komplexními dokumenty jak např. *RichTextBox, DocumentViewer, StickyNoteControl…* Tyto ovládací prvky umí prezentovat texty s různým obsahem, včetně formátovacích konstrukcí s podporou operace vyhledávání a editace. Další tři ovládací prvky jsou používané pro zobrazení FlowDocumentů: *FlowDocumentScrollViewer*, *FlowDocumentPageViewer* a *FlowDocumentReader*

- ✓ *FlowDocumentScrollViewer* zobrazuje FlowDocument v dlouhém vertikálním rolovacím listu - jako webový prohlížeč zobrazuje Web stránky.
- ✓ *FlowDocumentPageViewer* zobrazuje obsah jedné stránky najednou pomocí zvětšování a rolování, pro snadnější prohlížení celého dokumentu.

- ✓ *FlowDocumentReader* může zobrazit FlowDocument kombinací obou způsobu co jsem uvedl výše.

Prvek *InkCanvas* implementuje kreslení štětcem po plátně (prostřednictvím polohovacího zařízení), včetně možnosti malovat přes zobrazený element – například obrázek. Načrtnuté tahy převádí do vektorové reprezentace (položky kolekce *Strokes*), se kterými lze dále operovat. S novým přístupem k designu uživatelského rozhraní souvisí i podpora médií. WPF integruje práci s grafickými daty. K zobrazování běžných formátů přidává i jejich zápis a konverzi. Ovládací prvek *Image* umí s obrázkem pracovat inteligentně. Nabízí řadu úprav a transformací, včetně nedestruktivního zvětšování, ořezávání a rotace. Navíc šetří systémové prostředky schopností dekódovat obrázek pouze na požadovanou velikost, aniž by se v paměti aplikace musel udržovat originál v plné velikosti.

3.2.1.3 Ovládací prvky pro seznam

Myslím si, že ovládací prvky pro seznam jsou hodně využívané jak ve WPF tak ve WinForms. Nejznámější seznam (*ListBox*) či rolovací seznam (*ComboBox*) jsou již nedílnou součástí mnoha aplikací. V technologii WPF jsou tyto ovládací prvky odvozené z typu *ItemsControl*, která mimo jiné definuje kolekci *Items*. Ve WinForms se většina ovládacích prvků pro seznam zobrazuje jako seznam objektů vystaveny kolekcí Items. Podle mého názoru se WPF snaží nijak neomezovat design. Každý jednotlivý objekt může být v rámci jednoho seznamového elementu vykreslen odlišně. Do výše uvedených prvků patří ještě třída *ListView* – potomek typu *ListBox*, jež zjednoduššuje zobrazení tabulek. Nicméně u tohoto prvku se projevuje nedostatečný záběr komponent technologie WPF. U *ListView* mi naprosto chybí implementace editačního režimu, který byl kvalitně vytvořen u prvku *DataGridView* předešlé technologie WinForms. WPF tak naprosto chybí obdobný nativní element.

Další zajímavým prvkem ve WPF je *Popup*. Jedná se o princip dobře známého rozvinutí rolovacího seznamu (*ComboBox*) či vyvolání kontextového menu pravým tlačítkem myši. WPF tak zcela obecně poskytuje tento překrývající mechanismus pro libovolnou funkcionalitu a libovolný prvek. Například popup prvek můžete použít jako kontextové menu pro objekt, který poskytne další informace a možnosti uživatelů.

Mezi seznamové prvky patří také *TreeView*, který slouží pro zobrazování hierarchické struktury s rozbalováním a sbalováním větví stromu

3.2.1.4 Ovládací prvky pro správu velikosti a rozmístění prvků

Největším rozšířením u WPF jsou prvky pro rozložení a správu vložených elementů. Nejvyužívanější množinou prvků ve WinFoms jsou prvky typu *Panel*. Ve WPF máme daleko větší výběr možností ovládacích prvků (např.Grid, DockPanel, StackPanel, WrapPanel, UniformGrid, Canvas, ToolBarOverflowPanel, VirtualizingStackPanel), které popíši později.

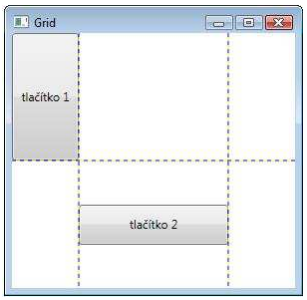
U WPF se při vytváření kompozice *Panely* stávají mocnými součástmi, které fungují nejen jako kontejnery pro vkládané prvku (kolekce *Children*), ale také přidávají schopnost jejich rozprostření v přiděleném prostoru a výpočtu jejich velikosti. Při tomto pochodu probíhá komunikace mezi ovládajícím rodičem a ovládanými potomky. V první fázi se zjišťují požadované rozměry a následně jsou pak prvky vykreslovány do určených oblastí. Postup výpočtu velikosti a vykreslení implementuje každý prvek podle vlastních parametrů (např. okraj, šířka, výška, zarovnání, transformace). Rodič jen rozhoduje o cílových rozměrech a pozici. Každý panel však tyto procesy provádí podle určitých pravidel.

Tyto panely pak můžeme nazývat *Layout controls* a najdeme je jen ve WPF:

Ovládací prvek Grid představuje tabulku nebo spíše mřížku podobnou HTML tabulce. Definice prostoru rozlišuje sloupce a řádky, přičemž ve výchozím stavu obsahuje pouze jedinou buňku (po jedné řádce i sloupci). Vložené prvky pak dostávají attached property Column, Row, RowSpan a ColumnSpan. Těmi se určuje, ve které buňce rastru se má prvek zobrazit a přes kolik buněk se má roztáhnout. U sloupců i řádků mřížky lze dále konfigurovat způsob, kterým si rozdělí celý vymezený prostor a jak budou reagovat na změnu jeho velikosti. Grid je díky tomuto nejrobustnějším kontejnerem, ale pro mnohé požadované aplikace návrháři raději využijí některý z ostatních prvků.

```
<Grid ShowGridLines="True">
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition Width="150" />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Button Grid.Column="0" Grid.Row="0">tlačítko 1</Button>
  <Button Grid.Column="1" Grid.Row="1" Height="40">tlačítko 2</Button>
</Grid>
```

Příklad 14: Ukázka panelu Grid v jazce XAML



Obrázek 36: Grid

Ovládací prvek *DockPanel* doplňuje vloženým prvkům atribut Dock, kterou je prvek svázán s jednou ze 4 stran (Top, Left, Right, Bottom). Prvky umisťuje vedle nebo pod sebe od uvedené strany. Priorita pořadí od kraje je dána pořadím při vložení. Poslední prvek v celé kolekci vyplňuje zbylý prostor.

```
<DockPanel>
   <Button DockPanel.Dock="Top">tlačítko 1</Button>
   <Button DockPanel.Dock="Left">tlačítko 2</Button>
   <Label BorderBrush="Blue" BorderThickness="2">Text label</Label>
                                        </DockPanel>
```

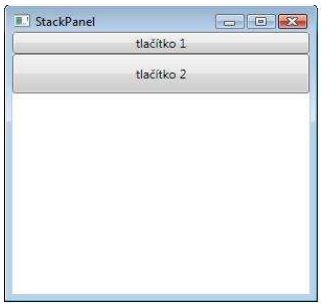
Příklad 15: Ukázka panelu DockPanel v jazyce XAML

Obrázek 37: DockPanel

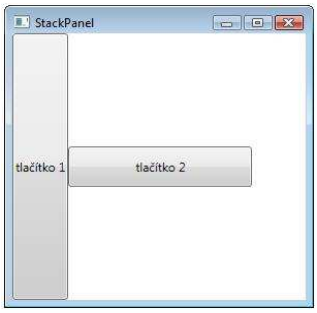
Ovládací prvek *StackPanel* vkládá prvky vedle sebe (horizontálně) nebo pod sebe (vertikálně), tak jak za sebou následují v kolekci. Nepřidává podřízeným prvkům žádnou vlastnost, protože není potřeba, všem poskytne požadovaný prostor nehledě na vlastní rozměry.

```
// vertikálně

  <StackPanel>
    <Button>tlačítko 1</Button>
    <Button Height="40">tlačítko 2</Button>
  </StackPanel>

// horizontálně

  <StackPanel Orientation="Horizontal">
    <Button>tlačítko 1</Button>
    <Button Height="40">tlačítko 2</Button>
  </StackPanel>
```

Příklad 16: Ukázka panelu StackPanel v jazyce XAML



Obrázek 38: StackPanel vertikálně

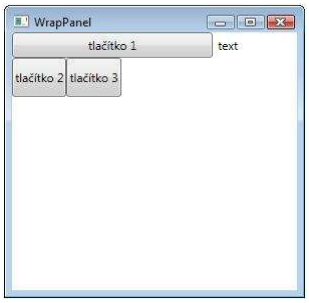


Obrázek 39: StackPanel horizontálně

Ovládací prvek *WrapPanel* - obdobně jako u textu jsou zde jednotlivé prvky zobrazeny vedle sebe na řádek, pokud se některý prvek již na řádek nevejde, dojde k zalomení řádku.

```
<WrapPanel>
   <Button Width="200">tlačítko 1</Button>
   <Label>muj text</Label>
   <Button Height="40">tlačítko 2</Button>
   <Button>tlačítko 3</Button>
 </WrapPanel>
```

Příklad 17: Ukázka panelu WrapPanel v jazyce XAML

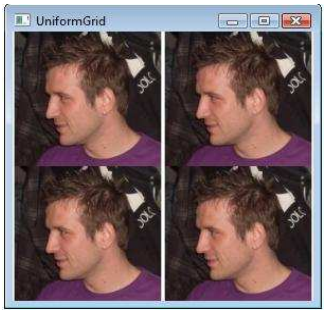


Obrázek 40: WrapPanel

Ovládací prvek *UniformGrid* je obdoba mřížky, která však vloženým objektům přiděluje stejný prostor. Na základě atributů Columns a Rows propočítává, kolik prvků se zobrazí v jednom řádku a jakou šířku jim přidělí.

```
<UniformGrid Columns="2" Rows="2" Name="uniformGrid1" >
            <Image Source="mujobrazek.jpg"></Image>
            <Image Source="mujobrazek.jpg "></Image>
            <Image Source="mujobrazek.jpg "></Image>
            <Image Source="mujobrazek.jpg "></Image>
  </UniformGrid>
```

Příklad 18: Panel UniformGrid má dva řádky a dva sloupce se čtyřmi Image kontrol

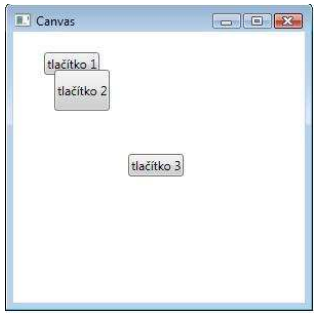


Obrázek 41: UniformGrid

Ovládací prvek *Canvas* - prvky uvnitř se absolutně pozicují. Při změně velikosti pak ponechává rozměry prvků nedotčené.

```
<Canvas>
  <Button Canvas.Top="20"
          Canvas.Left="30">tlačítko 1</Button>
  <Button Canvas.Top="37"
          Canvas.Left="40"
          Height="40">tlačítko 2</Button>
  <Button Canvas.Bottom="40"
          Canvas.Right="10">tlačítko 3</Button>
</Canvas>
```

Příklad 19: Ukázka panelu Canvas v jazyce XAML



Obrázek 42: Canvas

Ovládací prvek *Expander* je zcela novým prvkem spadajícím do skupiny *Layout prvků* ve WPF, který navíc přidává zajímavou funkcionalitu. Popis prvku ovládá dynamické zobrazení a skrytí oblasti s obsahem. Nově tedy WPF nabízí cestu ke kompozici, která kompaktně sdružuje mnoho informací, ale současně je přehledná.

```
<Expander Name="Expander1" Header="Můj Expander" Background="Linen"
Width="110">
            <StackPanel Background="Azure">
                <ListBox Name="list2">
                    <ListBoxItem>
                        <TextBlock>Bakalářské práce</TextBlock>
                    </ListBoxItem>
                    <ListBoxItem>
                        <TextBlock>Jan Lasac</TextBlock>
                    </ListBoxItem>
                </ListBox>
            </StackPanel>
        </Expander>
```

Příklad 20: Expander v jazyce XAML

Obrázek 43: Expander ve WPF

Mezi panely patří i další úzce specializované ovládací prvky:

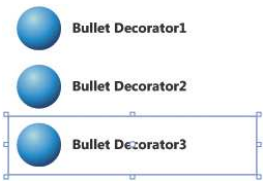
- ✓ Které mají stejnou funkci ve WinForms i ve WPF:

Ovládací prvek *TabControl***,** což je kontejner pro kolekci záložek neboli jednoprvkových elementů (TabPage). Ovládací prvek *Splitter* ve WinForms a *GridSplitter* ve WPF jsou dělícími pruhy, které samy o sobě nejsou kontejnerem, ale používájí se pro změnu velikosti kontejnerů, které jsou přichycené k nějaké straně svého kontejneru. Zvláštní podkategorií tohoto typu ovládacích prvků jsou ovládací prvky nabízející navíc i popisnou informaci o jejich obsahu – známým zástupcem je třída *GroupBox*.

- ✓ Ovládací prvky jen ve WPF:

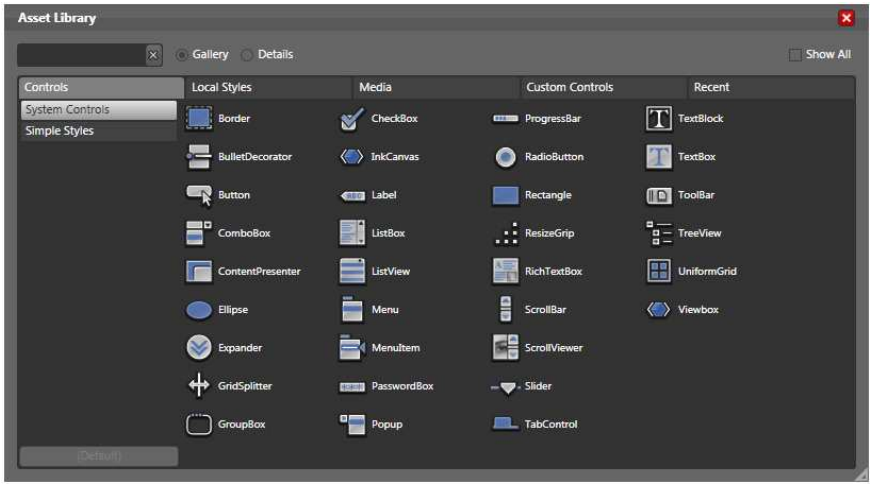
Border je jednoduchý prvek, který určuje hranici, pozadí nebo obojí kolem jiného prvku. Dále *ToolBarOverflowPanel*, který aranžuje neviditelné součásti tlačítkové lišty, a *VirtualizingStackPanel,* který funguje jako StackPanel, ale specificky nakládá s podřízenými prvky. Umí vykreslovat jen aktuálně viditelné prvky. Používá se v seznamových ovládacích prvcích k úspoře systémových prostředků. Ovládací prvek *Viewbox* slouží k vyběru a přiblížení určité oblasti z obrázku. Zobrazení vybrané oblasti pomocí čtyř hodnot - první dvě hodnoty specifikují pozici levého horního rohu oblasti, třetí hodnota udává šířku oblasti a čtvrtá hodnota

udává výšku prostoru. *Separator* je prvek, který vykreslí horizontální či vertikální meze mezi položkami v ovládacích prvcích, jako ListBox, Menu a ToolBar. *BulletDecorator* je prvek, který může vytvořit dva elementy, kterými jsou obvykle textový řetězec a glyf (představuje kontrolu jako zaškrtávací políčko - CheckBox).



Obrázek 44: Ovládací prvek BulletDecorator

3.2.1.5 Přehled Ovládacích prvků v produktu Expression Blend



Obrázek 45: Ovládací prvky v programu Expresion Blend

3.2.1.6 Přehled Ovládacích prvků ve Visual Studiu 2008

WinForms **WPF**

Obrázek 46: Porovnání ovládacích prvků ve Visual Studiu 2008

3.2.2 Způsoby rozvoje ovládacích prvků

Způsoby rozvoje prvků poskytují ve WPF i WinForms několik úrovní pro tvorbu nových prvků. Vývojáři mají možnost odvozovat své třídy od předešlých prvků implementujících obvykle požadovanou logiku a vlastnosti nebo mohou záměrně zvolit jako základ výrazně obecnější typ, což výsledný prvek odlehčí a programátorům poskytne větší kontrolu. Obecně se předpokládají dvě schémata tvorby nových prvků.

3.2.2.1 UserControl ve WPF

Uživatelský ovládací prvek *UserControl* ve WPF se skládá ze dvou souborů – xaml jednotky pro nastavení vzhledu a obsahu, dále pak kódové jednotky pro přiřazení nových vlastností, událostí a metod. Představuje nový rozměr návrhu prvku, neboť je více nastaven na vlastní způsob návrhu, takzvaně „*lookfilled*“. Tento postup návrhu začíná od vizuální reprezentace a končí u programování logiky. Obsahem prvku (v souboru xaml) jsou totiž přímo ovládací prvky WPF, což vytváří přímou vazbu mezi vizuálním stromem a kódem. Současně to i znamená, že vyvíjený prvek nelze dále dědit (rozvíjet v potomcích), a nelze mu ani předdefinovat šablonu (vizuální strukturu). Na druhou stranu, na rozdíl od běžných ovládacích prvků, jsou vizuální součásti přímo v logickém stromu a tedy z kódu lépe dostupné. Tvůrce prvku snadněji vizuálně vylepší v grafickém prostředí editoru (např. Expression Blend a jiné), a následně přenechá programátorovi jednotku s kódem pro doplnění potřebné funkcionality.

Nevýhodou tohoto schématu je složitější zpřístupnění atributů vizuálních prvků obsahu. Paradoxně je i komplikovanější použití nových vlastností při definici vzhledu a jeho chování. Důsledkem odvození od třídy *UserControl* je existence závislosti na tomto předku. Ve výsledku můžeme v deklarativním prostředku používat jen vlastnosti specifikované pro tohoto předka. Hodnota nově přidávané vlastnosti může být reflektována jen prostřednictvím kódu, což je při maximálním upřednostňování deklarativního návrhu značným záporem. Tudíž je použitelný především tam, kde tvoříme vizuálně bohatý prvek s neměnným obsahem a vizuální prezentací. Z pohledu objektového modelu se nejedná o dobré řešení, ale i přesto může často splnit cíl (například grafický symbol se schopností vyvolat akci) a přitom nabízí snazší způsob implementace – vhodný zejména pro grafické designéry.

3.2.2.2 UserControl ve WinForms

Uživatelský ovládací prvek *UserControl* ve WinForms je způsob, jak připravit nějakou sadu ovládacích prvků pro opětovné využití tak, aby se dala použít jako celek. Vyprodukuje se jakýsi druh „podformuláře“. Všechny techniky pro uspořádávání ovládacích prvků, na něž jste zvyklí jako např. kotvení a přichycování, fungují stejně v uživatelském ovládacím prvku jako na vlastním formuláři. Stejné postupy se používají i při nastavování vlastností nebo zpracování událostí. Uživatelské ovládací prvky umožňují budovat opětovně využitelné ovládací prvky se

stejnými nástroji, s jakými vytváříte své formuláře, ale navíc s výhodou, že uživatelský ovládací prvek můžete táhnout a upustit na čemkoliv, co může obsahovat ovládací prvky, tedy na kontejnerových ovládacích prvcích, formulářích, a dokonce i na jiných uživatelských ovládacích prvcích.

3.2.2.3 CustomControl ve WPF

Vlastní prvek *CustomControl* již plně zapadá do objektového modelu WPF. Za základ může sloužit jakákoli třída splňující naše představy. Předpokládá se použití některého z potomků *UIElement*, ať už hotového ovládacího prvku nebo obecné třídy *Control*.

Prvek prezentuje programová jednotka. Vizuální reprezentace prvku, pokud je požadavek na její redesign, se vkládá do společného slovníku stylů. Ten může být společný pro celou knihovnu prvků, lze ho však hierarchicky rozdělit do více fyzických souborů. Centrální slovník je jediné místo, kde se hledá vzhled prvku. Odlišné vzhledy jsou umožněny existencí více centrálních slovníků, kde každý je cílený na vlastní operační systém – *Luna* pro WindowsXP, *Aero* pro Windows Vista, *Classic* pro Windows NT, *Generic* pro případy, kdy správný slovník schází. Kompletně zpracovan grafický prvek má mít tedy vytvořeny styly do všech typů slovníků. Podívejme se však na schéma návrhu. V tomto případě je postup obrácený než u *UserControl* – nejprve naprogramujeme logiku, včetně vlastností a událostí, a následně k prvku přidáme vizuální vzhled napojený na sadu vlastností nového prvku („*lookless*“). Získáváme možnost prvek rozvíjet v dalších potomcích, vytvářet další alternativní zobrazení a uživatelsky modifikovat vzhled. Také v tomto případě můžeme pro vizuální definici použít grafický editor a výsledný styl přenést do slovníku. Výhodou je flexibilnější a rozmanitější přístup, na druhou stranu je i pracnější. Nejvýraznější rozdíl je z hlediska spolupráce s vizuálními elementy. Prvky *CustomControl* jsou chápany jako nezávislé na své reprezentaci. Umožňují vytvořit požadovanou funkcionalitu, ale ponechávají na designérovi, jak tuto funkcionalitu znázorní.

3.2.2.4 CustomControl ve WinForms

Vlastní prvek (*CustomControl*) ve WinForms můžeme rozdělit do tří druhů:

- **Ovládací prvky přímo odvozené ze základní třídy Control.** Umožnují kompletně zpracovávat vstup i výstup ovládacího prvku.
- **Ovládací prvky odvozené ze ScrollableControl**, které jsou podobné ovládacím prvkům odvozeným z Control, ale poskytují zabudovanou podporu pro posouvání.
- **Ovládací prvky odvozené z nějakého existujícího ovládacího prvku,** jehož úkolem je rozšíření jejich chování. Je mnohem jednodušší vylepšit již vytvořený prvek, než začínat úplně od začátku odvozováním od třídy Control.

Někteří lidé nejsou nikdy spokojeni a pro programátory to zřejmě platí dvojnásobně. I když je v kolekci Windows Forms k dispozici velmi mnoho ovládacích prvků, často je vhodné učinit ještě další krok (nebo dokonce několik kroků) vpřed a posunout se od připravených prvků do říše těch vlastních.

Z hlediska programování je takový vlastní ovládací prvek třídou, kterou definujete a jež je odvozena přímo nebo nepřímo od Control. Vlastní ovládací prvek může být vylepšením již existujícího prvku nebo zcela novým prvkem. Přestože lze chování ovládacího prvku výrazně upravit pouhou implementací handlerů událostí existujících prvků a jejich důkladným zpracováním, potřebujete-li zcela změnit výchozí zpracování událostí, budete muset použít novou třídu. Můžete například přidat nový vizuální vzhled ovládacímu prvku typu Button implementací handleru události Paint, nemůžete ale tlačítku zabránit v zobrazení jeho vlastních vizuálních součástí, pokud nevytvoříte novou třídu a nepřekryjete *OnPaint*.

Novou třídu budete také muset vytvořit, budete-li chtít k nějakému existujícímu prvku přidat určité atributy nebo vlastnosti. Potřebujete-li však prvku zadat jen nějaké zvláštní údaje, zvažte použití vlastnosti Tag, která slouží právě k tomuto účelu. Uvedená vlastnost je definovaná jako typ *object*, takže při práci s ní bude zapotřebí používat přetypování. Je však ideální k jednoduchému identifikování ovládacího prvku a připojení libovolných doplňkových dat.

Jak už tomu při programování bývá, vlastní ovládací prvky jsou skutečným přínosem až v okamžiku, kdy je opakovaně používáte v několika aplikacích nebo je zpřístupňujete jiným programátorům, ať už za peníze nebo jen pro slávu.

3.2.3 Porovnání tvorby ovládacích prvků ve WimForms a WPF

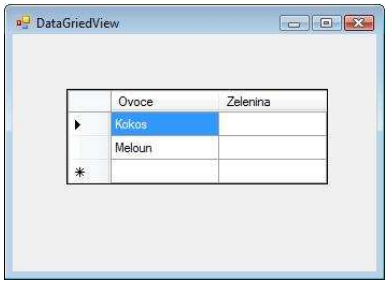
V této části práce budu porovnávat u těchto vybraných prvků (*TreeView, DataGriedView, ListView,Button*), jaké mají možnosti nastavení vzhledu, složitost jejich vytvoření, přehlednost kódu a jejich použitelnost, ve které chceme, aby prvek měl efektivní a ergonomické ovládání, přehlednou navigaci, zřetelné rozlišení funkčních elementů a mnoho dalších rysů, které zlepšují a ulehčují práci se systémem.

3.2.3.1 Prvek TreeView

<table>
<tr><th></th><th colspan="2">WinForms</th><th colspan="2">WPF</th></tr>
<tr><td>Název</td><td colspan="2">TreeView</td><td colspan="2">TreeView</td></tr>
<tr><td>Vzhled</td><td colspan="2"> 
</td><td colspan="2"> 
</td></tr>
<tr><td>Kód</td><td colspan="2">treeView1.Location = new Point(12, 42);
treeView1.Size = new Size(150, 180);
treeView1.AllowDrop = true;
treeView1.Nodes.Add("Káva");
treeView1.Nodes.Add("Čaj");
treeView1.Nodes.Add("Džus");
treeView1.Nodes.Add("Mléko");
treeView1.Nodes.Add("Pivo");</td><td colspan="2"><TreeView Name="TreeView1">
<TreeViewItem Header="Káva">
<TreeViewItem
Header="Čaj"></TreeViewItem>
<TreeViewItem
Header="Džus"></TreeViewItem>
<TreeViewItem
Header="Mléko"></TreeViewItem>
<TreeViewItem
Header="Pivo"></TreeViewItem>
</TreeViewItem>
</TreeView></td></tr>
<tr><td>Porovnání</td><td colspan="2">+ jednoduchost</td><td colspan="2">+ více možnosti nastavení vzhledu
+ přehlednost kódu</td></tr>
<tr><td rowspan="5">Hodnocení</td><td>Vzhled</td><td>50 %</td><td>Vzhled</td><td>80 %</td></tr>
<tr><td>Jednoduchost</td><td>90 %</td><td>Jednoduchost kódu</td><td>70 %</td></tr>
<tr><td>Přehlednost kódu</td><td>70 %</td><td>Přehlednost kódu</td><td>90 %</td></tr>
<tr><td>Použitelnost</td><td>80 %</td><td>Použitelnost</td><td>90 %</td></tr>
<tr><td>Celkem</td><td>58 %</td><td>Celkem</td><td>66 %</td></tr>
</table>

Tabulka 1: Porovnání prvků TreeView

3.2.3.2 Prvek DataGridView a ListView

<table>
<tr><th></th><th colspan="2">WinForms</th><th colspan="2">WPF</th></tr>
<tr><td>Název</td><td colspan="2">DataGriedView</td><td colspan="2">ListView</td></tr>
<tr><td>Vzhled</td><td colspan="2"> 
</td><td colspan="2"> 
</td></tr>
<tr><td>Kód</td><td colspan="2">

```
DataGridViewTextBoxColumn ovoce =
new DataGridViewTextBoxColumn();
ovoce.HeaderText = "Ovoce";
ovoce.Name = "Ovoce";

    string[] row0 = {"Kokos"};
    string[] row1 = {"Meloun"};

dataGriedView1.Columns.Add(ovoce)
;
dataGriedView1.Rows.Add(row0);
dataGriedView1.Rows.Add(row1);

DataGridViewTextBoxColumn
zelenina = new
DataGridViewTextBoxColumn();
zelenina.HeaderText = "Zelenina";
zelenina.Name = "Zelenina";
dataGriedView1.Columns.Add(zeleni
na)
```

</td><td colspan="2">

```
<ListView>
    <ListViewItem
Content="Ovoce">
    </ListViewItem>
</ListView>

  <ListView>
    <ListViewItem
Content="Kokos">
    </ListViewItem>
    <ListViewItem
Content="Meloun">
    </ListViewItem>
  </ListView>
    <ListView >
    </ListView>
 <ListView >
<ListViewItem>Zelenina</ListVie
wItem>
</ListView>
```

</td></tr>
<tr><td>Porovnání</td><td colspan="2">+ jednoduchost
+ kvalitní způsoby kompozice i prezentace seznamu</td><td colspan="2">+ více možností nastavení vzhledu
+ přehlednost kódu
- chybí zdrojový kod</td></tr>
<tr><td rowspan="5">Hodnocení</td><td>Vzhled</td><td>60 %</td><td>Vzhled</td><td>80 %</td></tr>
<tr><td>Jednoduchost</td><td>100 %</td><td>Jednoduchost</td><td>50 %</td></tr>
<tr><td>Přehlednost kódu</td><td>70 %</td><td>Přehlednost kódu</td><td>90 %</td></tr>
<tr><td>Použitelnost</td><td>100 %</td><td>Použitelnost</td><td>50 %</td></tr>
<tr><td>Celkem</td><td>82 %</td><td>Celkem</td><td>72 %</td></tr>
</table>

Tabulka 2: Porovnání prvků DataGridView a ListView

3.2.3.3 Prvek TabControl

<table>
<tr><th></th><th colspan="2">WinForms</th><th colspan="2">WPF</th></tr>
<tr><td>Název</td><td colspan="2">TabControl</td><td colspan="2">TabControl</td></tr>
<tr><td>Vzhled</td><td colspan="2"> 
</td><td colspan="2"> 
</td></tr>
<tr><td>Kód</td><td colspan="2">

```
TabControl tabControl1=new
TabControl();
tabControl1.Location=new
Point(12,42);
tabControl1.AllowDrop = true;

tabControl1.TabPages.Add("Sýr"
);
tabControl1.TabPages.Add("Rajč
e");
tabControl1.TabPages.Add("Másl
o");
tabControl1.TabPages.Add("Moza
rela");
```

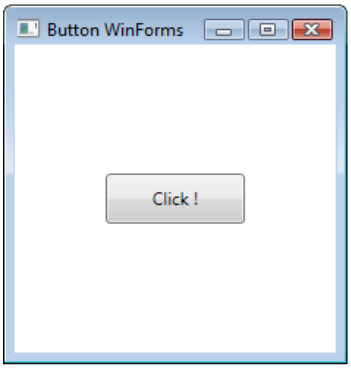
</td><td colspan="2">

```
<TabControl Name="tabControl1">
 <TabItem Header="Sýr"></TabItem>
 <TabItem Header="Rajče"></TabItem>
 <TabItem Header="máslo"></TabItem>
 <TabItem
Header="Mozarel"></TabItem>
</TabControl>
```

</td></tr>
<tr><td>Porovnání</td><td colspan="2"></td><td colspan="2">+ Více možností nastavení vzhledu</td></tr>
<tr><td rowspan="5">Hodnocení</td><td>Vzhled</td><td>70 %</td><td>Vzhled</td><td>90 %</td></tr>
<tr><td>Jednoduchost</td><td>70 %</td><td>Jednoduchost</td><td>70 %</td></tr>
<tr><td>Přehlednost kódu</td><td>80 %</td><td>Přehlednost kódu</td><td>80 %</td></tr>
<tr><td>Použitelnost</td><td>80 %</td><td>Použitelnost</td><td>80 %</td></tr>
<tr><td>Celkem</td><td>75 %</td><td>Celkem</td><td>80 %</td></tr>
</table>

Tabulka 3: Porovnání prvků TabControl

3.2.3.3 Prvek Button

<table>
<tr><th></th><th colspan="2">WinForms</th><th colspan="2">WPF</th></tr>
<tr><td>Název</td><td colspan="2">Button</td><td colspan="2">Button</td></tr>
<tr><td>Vzhled</td><td colspan="2"> 
</td><td colspan="2"> 
</td></tr>
<tr><td>Kód</td><td colspan="2">

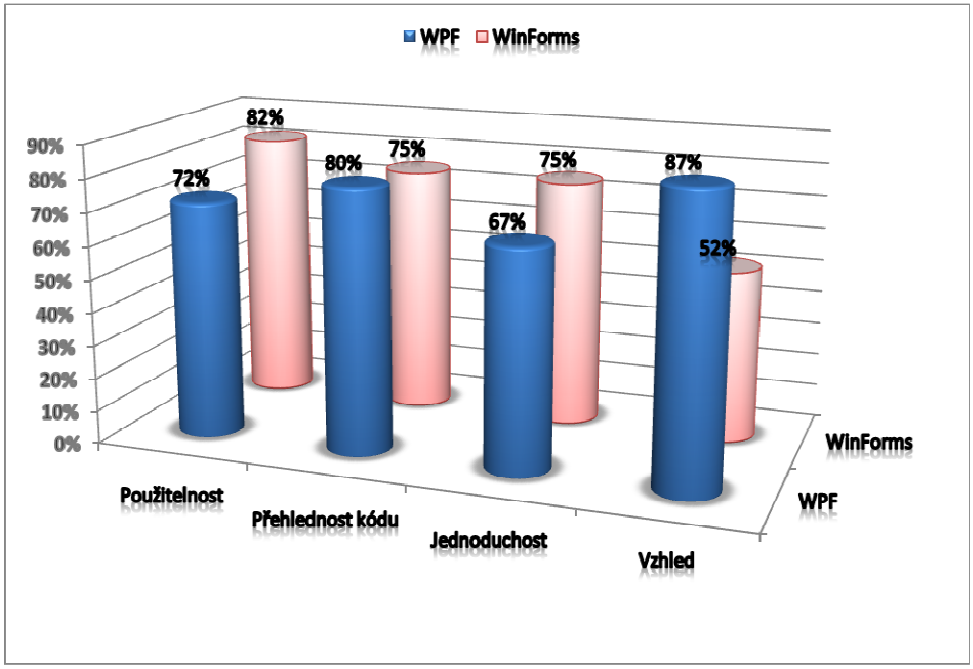
```
Button buttonl = new Button();
button1.Text = "Click!";
button1.Height = 25;
button1.Width = 75;
```

</td><td colspan="2">

```
<Button Name="button1" Content="
Click ! " Height="35" Width="95">
 <Button.BitmapEffect>
   <OuterGlowBitmapEffect
GlowColor="Blue" GlowSize="10"/>
   </Button.BitmapEffect>
 <Button.RenderTransform>
    <RotateTransform Angle="45"/>
 </Button.RenderTransform>
</Button>
```

</td></tr>
<tr><td>Porovnání</td><td colspan="2">- složitější nastavení různých efektů a transformací</td><td colspan="2">+ více možností nastavení vzhledu</td></tr>
<tr><td rowspan="5">Hodnocení</td><td>Vzhled</td><td>30 %</td><td>Vzhled</td><td>100 %</td></tr>
<tr><td>Jednoduchost</td><td>40 %</td><td>Jednoduchost</td><td>80 %</td></tr>
<tr><td>Přehlednost kódu</td><td>80 %</td><td>Přehlednost kódu</td><td rowspan="2">80 %
70 %</td></tr>
<tr><td>Použitelnost</td><td>70 %</td><td>Použitelnost</td></tr>
<tr><td>Celkem</td><td>60 %</td><td>Celkem</td><td>87 %</td></tr>
</table>

Tabulka 4: Porovnání prvků Button



Graf 1: Porovnání WPF a WinForms

3.2.3.5 Vyhodnocení Porovnávání

Jak je vidět i na grafu 1, ovládací prvky ve WPF, které jsem testoval, mají daleko více možností jak nastavit vzhled, jsou složitěji vytvořitelné (vyjímka byla u prvku Button), protože WPF je stále ještě mladou technologií a programátoři jsou zvyklí na známou technologii WinForms. Při testování použitelnosti prvků pro tabulkové zpracování seznamů, jako je *DataGridView* ve WinForms a *ListView* ve WPF, jsem zjistil, že *DataGriedView* poskytuje bohatý design jednotlivých elementů zobrazení a navíc nabízí netradiční způsoby kompozice i prezentace seznamu. U prvku *ListView* jsem postrádal řadu vlastností, které byly často řešeny technikami neodpovídajících WPF. Například zde nejsou jasné výkonové parametry, ani nebyl k dispozici zdrojový kód apod.

3.3 Programovací jazyk

3.3.1 Srovnání C# vs XAML

Už víme, že XAML je nový značkovací jazyk pro psaní grafického rozhraní, ale ve WPF existuje stále možnost psát uživatelské rozhraní i starým způsobem, stejně jako ve WinForms - programově v jazyce C#.

Zde je pro porovnání ukázka stejné části aplikace v XAMLu a v C#.

XAML

```
<DockPanel>
  <Label Content="Ahoj"
         FontStyle="Italic"
         FontWeight="Bold"
         Foreground="Black"
         HorizontalAlignment="Center"
         VerticalAlignment="Center" />
</DockPanel>
```

Příklad 21: Příklad v XAMLu

C#

```
DockPanel myDockPanel = new DockPanel();

Label myLabel = new Label();
myLabel.Content = "Ahoj";
myLabel.FontStyle = FontStyles.Italic;
myLabel.FontWeight = FontWeights.Bold;
myLabel.Foreground = Brushes.Black;
myLabel.HorizontalAlignment = HorizontalAlignment.Center;
myLabel.VerticalAlignment = VerticalAlignment.Center;

myDockPanel.Children.Add(myLabel);
```

Příklad 22: Příklad v C#

3.3.2 Shrnutí

Ve výsledku je XAML převeden do zdrojového kódu a přeložen do binární formy. Nemusí sloužit jen k definici designu, ale také k definici systémových zdrojů nebo celé aplikace. Myslím si, že kód jazyka XAML je v mnoha případech stručnější než odpovídající kod C# a lépe reprezentuje hierarchické struktury, které vznikají například při návrhu rozložení oken s panely a ovládacími prvky. Značkovací jazyky jsou však obecně mnohem více omezeny než procedurální jazyky, zejména proto, že nezahrnují princip řízení toku. V některých situacích XAML nestačí (např. při vypisování tlačítek v cyklu) a musíme pro definici uživatelského rozhraní použít C#. V kódu XAML se zpravidla nesetkáte dokonce ani s jednoduchým sdílením proměnných. Podrobnějším prozkoumáním jazyka XAML zjistíte, že obsahuje mnohé vlastnosti, které tyto nedostatky kompenzují.

4 Knihovna

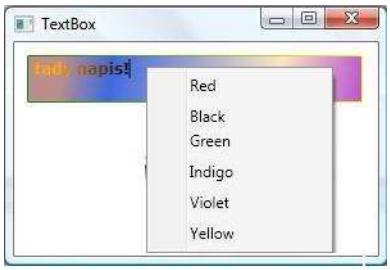
4.1 Textové prvky

Celá skupina požadovaných prvků úzce souvisí s úpravami textového řetězce různého charakteru. Za jejich základ jsem zvolil hotový ovládací prvek pro vstup textu - třída *TextBox*, který byl doplněn zejména o rozsáhlejší parametrizaci zobrazovaného popisku položky (např. definovat štětce pozadí a popředí). Poskytuje funkcionalitu pro práci s textem (nejen krátkými řetězci), dodává integrovaný rolovací posuvník a nabízí i stylizaci zobrazeného obsahu. Dále jsem tomuto prvku vytvořil směrovou událost, která klepnutím pravým tlačítkem myši zobrazí místní nabídku, kde si můžeme vybrat, jakou chceme barvu textu.

V elementu *ContexMenu* jsou metody *MenuItem*. Objekt *MenuItem*, který událost aktivuje , slouží jako vlastnost *Source* objektu *RoutedEventArgs*. Tato obslužná rutina převádí text položky *MenuItem* na objekt typu *Color* pomocí statické metody *ColorConverter.ConvertFromString*.

Vlastnosti jsem implementoval jako typ attached property, aby byly dostupné pro všechny elementy rozhraní, a také proto, abych využil výhodu průchodu hodnoty stromem elementů. Například:

- ✓ ***TextBoxForeground***
- ✓ ***TextBoxBackground***
- ✓ ***TextBoxFontSize***
- ✓ ***TextBoxFontFamily***
- ✓ ***TextBoxVisibility***
- ✓ ***TextBoxWraping***
- ✓ ***TextBoxBorderBrush***



Obrázek 47: Ovládací prvek TextBox

4.1 Tlačítkové prvky

Může se zdát, že tlačítko není zrovna prvek pro další rozvíjení. WPF ale díky bohatému obsahu poskytuje větší variabilitu. Nejsme omezeni na jediný obsahový element. Navíc tlačítko v aplikaci nabývá různých podob, které se od sebe významově liší.

U prvku na obrázku 50 jsem definoval vlastnosti, které odpovídají uživatelským akcím, jako je vlastnost *IsMouseOver*, která je nastavena na hodnotu true, a jakmile se uživatel pohybuje kurzorem myši přes prvek, změní barvu prvku, nebo vlastnost *IsPressed*, která reaguje na stisknutí tlačítka na myši. K provedení akce při změně hodnoty vlastnosti typu DependencyProperty jsem použil *Property* trigger. Tento zápis skrývá velkou výhodu. Trigger totiž představuje akci, která pouze změní patřičné hodnoty. Dojde-li k zneplatnění jeho podmínky, nemusíme navracet původní hodnoty, neboť zpětné nastavení se provede automaticky.

Obrázek 48: Prvek Button

Obrázek 49: Prvek Button při najetí

Obrázek 50: Prvek Button při stisknutí

U ovládacího prvku na obrázku 53 jsem také použil *Property* trigger k provedení akce při změně hodnoty vlastnosti typu DependencyProperty, jen s tím rozdílem, že při stiskutí tlačítka na myši se prvek zmenší pomocí šalovacího faktoru prostřednictvím třídy *ScaleTransform* (viz. 3.1.4.8 Transformace) Transformace. Transformaci můžeme totiž aplikovat na libovolný objekt odvozený z objektu UIElement.

Obrázek 51: Prvek ButtonEllipse

Obrázek 52: Prvek ButtonEllipse při najetí

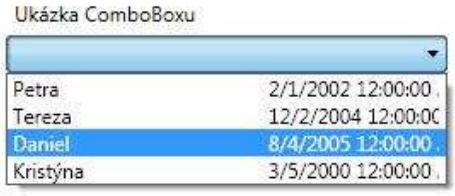
Obrázek 53: Prvek ButtonEllipse při stisknutí

Prvek *RadioButton* je odvozený od třídy *RadioButton*, implementuje uživatelskou změnu hodnoty a směrovou událost typu *Routed Event*, která nastavuje vlastnosti *Foreground* a *Background*, pomocí kterých se mění barva přepínače, jak to vidíme na obrázku.

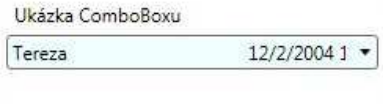
Obrázek 54: Prvek RadioButton

4.2 Seznamové prvky

Ovládací prvek roletové nabídky ***LabelComboBox*** nesmí chybět v nabídce žádné knihovny prvků. Jeho užitečnost pro úsporu místa a snadné ovládání je nepřehlédnutelná. Tento prvek odvodíme od třídy *ComboBox*. Rolovací seznam bude spojován se speciálním datovým zdrojem. Načtení seznamu se realizuje až při vzniku události *Window_Loaded.* Ideální by byl okamžik, kdy se rozvíjí seznam, ale i zde existuje způsob použití, který se bez rozvinutí obejde (šipka dolů). Proto je nutné vyvolat nahrání datového zdroje již při zisku zaměření. Další rozšíření dodává prvku příkaz *ClearSelection*, který umožňuje (jinak nedostupné) vyprázdnění aktuálního výběru.



Obrázek 55: Prvek LabelComboBox otevřený



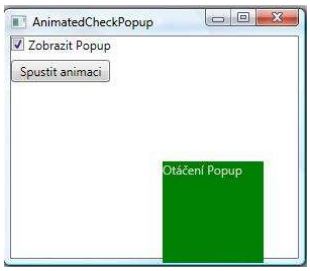
Obrázek 56: Prvek LabelComboBox

Následující prvek pracuje s kolekcí záznamů - ***CheckedListBox***. Je odvozen od prvku *ListBox*, který umožňuje uživateli vybrat jednu položku (nebo volitelně více položek) z jejich kolekce. Ve své výchozí podobě je prvek *Listbox* jednoduchý a strohý. Prvek *ChecekListBox* nevyvolává událost *Window_Loaded*, neboť se jedná o prvek, jehož nabídka je přímo viditelná. Šablona tohoto prvku reflektuje přidané části jako prvek *CheckBox*, obrázek znázorněný pomocí prvku *Image* a *TextBlock* prvek, který nám umožní zobrazit text. Umisťování ošetřuje prvek StackPanel, jak vidíte na obrázku 59. Pro prvek ***CheckedListBox*** jsem nevytvořil zapojení datových událostí seznamových prvků, ale díky přiložitelné realizaci nebude pracné je doplnit.

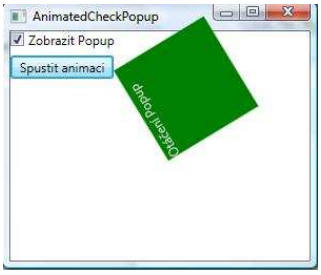
Obrázek 57: Prvek CheckedListBox

Dalším zajímavým seznamovým prvkem je ***AnimatedCheckPopup***. V tomto prvku jsem implementoval animaci, kterou jsem umístil do definice stylu v jazyce XAML, neboť se jedná o vizuální prostředek. Pro zápis deklarativního XAMLu můžeme například využít prostředí produktu Expression Blend. Pohyb prvku se potom podobá práci v grafickém programu. Interaktivně vytvoříme scénáře akcí pro obě události, ale zjistíme, že konfigurace této implementace je poněkud složitá. Pro programátora je realizace v kódu jednodušší a při použití směrovaných událostí i jediná možná. Avšak největší rys směrovaných událostí zapříčiňuje podstatný problém, se kterým jsem se setkal právě při vytváření animace v prostředí XAMLu. Událost totiž prochází logickým stromem.

Zobrazení prvku *AnimatedCheckPopup* se vyvolá pomocí bindingu, který je realizován zašrtnutím *CheckBoxu*. Animace se spustí prostřednictvím události typu Routed Event. Ke spuštění procesu slouží objekt *DoubleAnimation*, který otáčí prvek kolem své osy a zpět pomocí transformace (viz. 3.1.4.8 Transformace).



Obrázek 58: Prvek AnimatedCheckPopup



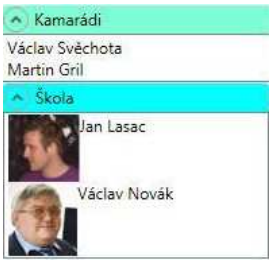
Obrázek 59: Otáčení prvku AnimatedCheckPopup

4.3 Ostatní prvky

K vytváření ovládacího prvku ***UserControlButton*** jsem použil třídu *UserContol*. Namapoval jsem namespace *UserControlButton* na element *rc*, abych přes tento element mohl přistupovat k prvkům v tomto namespace. Následně jsem vytvořil vlastnost *Items*, a do této vlastnosti jsem vloži tvar - elipsu. Poté jsem přidal jednoduchý efekt odlesk, který jsem vytvořil pomocí *Gridu* o dvou řádcích, v prvním je celé rozhraní ovládacího prvku a ve druhém odlesk do ztracena. Do druhého řádku *Gridu* jsem vložil prvek *Border*, na jehož pozadí jsem nabindoval ovládací prvek. Výhoda použití bindování je, že automaticky reflektuje změny prvku v odlesku (například rámeček po najetí myší nad jakýkoliv tvar se objeví i v odrazu). Nakonec jsem přidal OpacityMask, což je lineární gradient definující průhlednost.

Obrázek 60: Prvek UserControlButton

Novinka mezi ovládacími prvky – třída *Expander* – nabízí nové pohledy na sestavování obrazovek. Vzhledem k jednoduchému ovládání se stane často využívanou technikou pro kompozici informačně bohatých prezentací. Hlavička obsahuje stručný přehled. Rozkrytí pak může uživatel provádět s již konkrétní představou o výsledku akce. Prvek přináší větší možnosti při dělení prostoru nebo členění informačního obsahu. V prvku ***Expander*** jsem vnořil jeden svůj odvozený *Expander* do druhého. Pro zobrazovení obrázků jsem zvolil prvek *Image*, který ulehčuje práci s rozličnými grafickými soubory a poskytuje netradiční přístup k jejich zobrazování.



Obrázek 61: Prvek Expander při rozvinutí

U ovládacího prvku ***MediaElement*** jsou metody jednoduché, jen volají příslušnou funkci MediaElementu. Při otevření videa se zobrazí dialog, ve kterém uživatel vybere video nebo audio. To následně spustí. Video lze zastavit, pozastavit, spustit, měnit hlasitost a rychlost přehrávaní. Stejným způsobem bychom mohli přidat i jiné prvky na kontrolu videa, například posouvat video a jiné.

Vytvořený přehrávač samozřejmě nemůže konkurovat Windows Media Playeru, spíše slouží jen na demonstraci vlastností WPF.

Obrázek 62: Prvek MediaElement

SliderProgressBar je prvek, který prostřednictvím posuvníku mění hodnotu *Progressbaru* (pomocí bindingu). Využil jsem třídu *TransformGroup* (viz. 3.1.4.8 Transformace)., která umožňuje použít více transformací najednou. Nadefinoval jsem čtyři transformace *SkewTransform, RotateTransform, TranslateTransform ScaleTransform*, které si uživatel může později zvolit podle svých představ.

Obrázek 63: Prvek SliderProgressBar

Ovládací prvek ***CheckedTreeView***, jak vidíme na obrázku 67, zobrazuje stromovou hierarchii. Jelikož ovládací prvek *CheckedTreeView* je odvozen od prvku *TreeView*, jsou všechny položky v ovládacím prvku objekty typu *TreeViewItem*. V objektu typu *TreeViewItem* je implementováno zaškrtávátko *CheckBox*, prvek *Image* pro vykreslení obrázku a prvek *TextBlock* pro zobrazení textového řetězce. Ovládací prvek *CheckedTreeView* se velmi podobá prvku *CheckedListBoxu*.

Obrázek 64: Prvek CheckedTreeView

4.5 Zhodnocení práce na knihovně

WPF je stále poměrně nová technologie s řadou otazníků. Hned na začátku vytváření knihovny prvků jsem se nevyhnul potížím. Například nefungovaly některé funkce vývojového prostředí Microsoft Visual Studio nebo jsem musel řešit problémy samotného programovacího prostředí, proto jsem byl nucen udělat jisté kompromisy ohledně výběru budoucí knihovny prvků. Také se objevily problémy s vytvářením odvozených prvků typu CustomControl. Problém se vyznačoval absencí společného slovníku stylů, což jsem nakonec vyřešil explicitně vytvořeným slovníkem.

Zkoušel jsem vytvářet prvky v grafickém editoru Expression Blend, ale nakonec jsem od toho upustil, protože vývojové prostředí Microsoftu Visual Studio a grafický editor Expression Blend spolu moc dobře nespolupracovaly, ačkoli oba programy jsou od stejného výrobce a podporují stejné technologie.

Na začátku vývoje knihovny mi chyběly komplexní informační zdroje v češtině, tudíž jsem musel hledat v zahraničních publikacích, které byly většinou v angličtině.

Musím přiznat, že jsem při vytváření knihovny neustále objevoval nové vymoženosti technologie WPF. Zpočátku byly mé kroky pracné a vyžádaly si mnoho zkoušek, než jsem pochopil, jak správně problém vyřešit. To se netýkalo pouze jazyka XAML, ale především programování funkční logiky za použití nových prostředků. Některé novinky jsem pochopil právě až při řešení tvorby knihovny.

5 Závěr

Cílem Bakalářské práce bylo seznámit se s modelem nové technologie WPF, porovnat tvorbu ovládacích prvků ve WinForms a WPF, vytvořit knihovnu ovládacích prvků ve WPF a dále ukázat postup tvorby prvků formou audiovizuální prezentace.

S technologií WPF jsem se seznamoval až při nainstalování nového systému Windows Vista. V práci uvádím její hlavní přednosti, vlastnosti a prostředky (nepopisuji technologii kompletně).

Vytvořená knihovna prvků, s přihlédnutím k vlastním zkušenostem, je vcelku zajímavá a pestrá. Obsahuje sadu rozšířených ovládacích prvků, včetně výchozího vizuálního vzhledu. Při řešení vznikaly větší i menší problémy, které bylo nutné řešit hledáním jiných způsobů realizace.

Mladost technologie znázorňuje zhoršenou dostupnost znalostní báze, která se ale díky rozšiřující se komunitě WPF vývojářů rozrůstá.

Součástí bakalářské práce bylo také porovnat předešlou technologii WinForms a WPF a dále u těchto dvou technologií porovnat tvorbu ovládacích prvků. Při porovnávání těchto dvou technologií jsem popisoval především hlavní změny technologie WPF oproti WinForms. U porovnání tvorby ovládacích prvků jsem došel k výsledku, jak už je známo u technologie WPF, že má daleko bohatší možnosti nastavení vzhledu, ale zároveň i k tomu, že má některé nedostatky, což je způsobeno stářím technologie a absencí některých prvků z technologie WinForms. Porovnání bylo prováděno na 4 ovládacích prvcích z každé technologie, což může snižovat výsledek komplexního hodnocení.

Myslím se, že tato práce naplňuje představy ze zadání, ačkoliv v některých atributech vývoje knihovny jsem mohl více využít přednosti nové technologie WPF, když se na to teď dívám zpětně. Při seznamování s modelem WPF jsem musel nakonec u některých prvků slevit z původních požadavků na jejich implementaci. Významný vliv na snížení požadavků měly i uvedené faktory (omezená znalostní báze, zcela nová technologie). Celkově však přínosy řešení s použitím WPF oproti WinForms převládají a zejména pro aplikace s důrazem na vizuální stránku ovládacích prvků bych tuto technologii doporučil.

6 Literatura

[1] Introduction to Windows Presentation Foundation [online]. c2007 [cit. 2009-01-01]. Text v angličtině. Dostupný z WWW:

http://msdn.microsoft.com/en-us/library/aa970268.aspx

[2] WPF tutorial.net [online]. c2007 [cit. 2009-01-01]. Text v angličtině. Dostupný z WWW:

http://www.wpftutorial.net/

[3] WPF – vytváříme kontroly [online]. c2007 [cit. 2009-01-01]. Text v češtině. Dostupný z WWW:

http://www.vyvojar.cz/Articles/460-6-wpf-vytvarime-kontroly.aspx

[4] Windows Presentation Foundation [online]. c2009 [cit. 2009-01-01]. Text v angličtině. Dostupný z WWW:

http://en.wikipedia.org/wiki/Windows_Presentation_Foundation

[5] MSDN Library - sekce Windows Presentation Foundation
http://msdn2.microsoft.com/en-us/library/ms754130.aspx

[6] PETZOLD, Charles. *Mistrovství ve Windows Presentation Foundation*. Jiří Fadrný . 1. vyd. Brno : Computer press, 2008. 928 s.

[7] SELLS, Chris. *C# a WinForms : programování formulářů Windows*. Ing Pavel Kristián; RNDr. Jan Pokorný. 1. vyd. Brno : Zoner Press, c2005. 648 s. ISBN 80-86815-25-0.

[8] Ruihua Jin : MSDN Hands On Lab - Building Line of Business Applications with Windows Presentation Foundation, květen 2007

[9] Adam Nathan, Lehenbauer Daniel: *Windows Presentation Foundation Unleashed*
Sams 2007, Indianapolis

[10] Steve Krug: *Webdesign : Nenuťte uživatele přemýšlet*
Computer Press Brno 2006, 2. aktualizované vydání

A Seznam použitých zkratek a pojmů

.NET Framework – vývojová platforma společnosti Microsoft
ActiveX – objektový model ovládacích prvků vyvinutý společností Microsoft
DirectX – technologie společnosti Microsoft pro vykreslování grafiky
DLL – Dynamic-link library, dynamicky připojované knihovny
EXE – koncovka spustitelného souboru
primitiva – nejjednodušší objekty
kód
spravovatelný – kód spouštěný pod správou virtuálního stroje Microsoft CLR
nespravovatelný – kód spouštěný přímo na procesoru počítače

RTF – Rich Text Format, text se stylizačními parametry
SDK – Software Development Kit, balík programů pro vývoj aplikací
SW - software, programové vybavení
Win32 – rozhraní pro vývoj aplikací pod Microsoft Windows
WPF – Windows Presentation Foundation, technologie pro vizuální vzhled aplikací
WPF/E – WPF/Everywhere, verze pro distribuci na jiné platformy
XAML – eXtensible Application Markup Language, značkovací jazyk pro definici vzhledu aplikací
XML – eXtensible Markup Language, značkovací jazyk

B Obsah přiloženého CD