Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

Propojení .NET s MS Office

Bakalářská práce

Daniel Domin

Vedoucí práce : Ing. Petr Vaněček, Ph.D.

České Budějovice 2007

Poděkování

Děkuji Ing. Petru Vaněčkovi, Ph.D. za odborné vedení mé práce a Mgr. Miloši Prokýškovi za cenné rady a připomínky.

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.

….....

Daniel Domin

V Č. Budějovicích 24. dubna 2007

Abstrakt

Tato bakalářská práce se zabývá propojením technologie .NET s kancelářským balíkem MS Office (momentálně ve verzi 2007). Realizace je provedena především v prostředí Visual Studia 2005 a jazyka C# s využitím vývojářského balíku VSTO (Visual Studio Tools for Office). Ukázány jsou i jiné technologie jako COM a ActiveX, které se však pro tento konkrétní úkol neukázaly být vhodné. V příloze jsou zdrojové kódy praktických příkladů.

Abstract

This bachelor thesis deals with connection between .NET technology and MS Office (current version 2007). It is realized mainly in Visual Studio 2005 and programming language C# using developers kit VSTO (Visual Studio Tools for Office). Other technologies how to connect applications, like COM and ActiveX, are shown here, but for this task were not applicable. Source codes are shown in appendixes.

Obsah

1. Úvod

Životnost každého programu je dána jeho užitnou hodnotou a především flexibilitou - tím, jak je možné ho uzpůsobit svým požadavkům. Program, který má minimum funkcí, ale umožňuje je uživateli dodatečně přidat má rozhodně větší potenciál, než srovnatelný program bez další možnosti rozšíření.

Tato bakalářská práce vychází z potřeby projektu GREG, zobrazit své grafické výstupy v kancelářském balíku firmy Microsoft. Mým cílem tedy bylo najít způsob, jak tento požadavek realizovat. Jako tři možné cesty se ukázaly technologie COM, ActiveX a vývojářský balík VSTO pro MS Visual Studio. Nejdříve si nastíníme problematiku pluginů a rozšiřování programů obecně za použití API. Potom se od počátku vývoje COM technologie dostaneme až k technologiím odvozeným včetně ActiveX a momentálnímu nástupci .NET. Nakonec se zaměříme na konkrétní produkty Microsoft Visual Studio 2005, Microsoft Office 2007 a ukážeme si možnosti jejich propojení na praktickém příkladu.

Cílem této práce je ukázat princip a výhody rozšiřování aplikací o vlastní funkce se zaměřením na technologii .NET a produkt MS Office. Zjištěné poznatky budou využity v projektu GREG.

2. Interakce mezi procesy

2.1 Plugin

Plugin (také označován jako „plug-in“ z anglického „to plug in“ - zasunout, zapojit nebo česky „zásuvný modul“) je software, který většinou nepracuje samostatně, ale jako doplňkový modul jiné aplikace a rozšiřuje tak její funkčnost. Protože plugin není samostatným programem, potřebuje, aby jeho mateřská aplikace byla k tomuto účelu přizpůsobena, tzn. musí mít nějaké rozhraní (API - Application Programming Interface), pomocí kterého s ní komunikuje, více lit. [1].

Počátky pluginů můžeme najít již polovině 70. let 20. století. Jednalo se konkrétně o textový editor EDT bežící na platformě Unisys VS/9, který umožňoval spustit externí program. Tomu byla zpřístupněna vyrovnávací paměť editoru s aktuálně editovaným souborem. Program mohl k datům volně přistupovat a dokonce nad nimi volat funkce editoru. Této možnosti bylo využito kompilátorem Fortranu vyvinutém na University of Waterloo, což přinášelo výhodu přímé kompilace programů psaných v editoru. Nebylo tedy třeba zdrojový soubor zpracovávat ve více programech. Po editaci bylo možné otevřený soubor rovnou i zkompilovat.

2.2 API (Application Programming Interface)

API (rozhraní pro programování aplikací) nabízí k případnému využití knihovna nebo operační systém (např. DirectX, OpenGL, Windows API). Je specifikováno pomocí programovacího jazyka a kompiluje se společně s mateřskou aplikací. Rozdílem oproti ABI (Application Binary Interface) je nutnost kompilovat program pro různé operační systémy (podporující dané API) zvlášť. Jde o sbírku procedur, funkcí či tříd knihovny (potažmo i jiného programu nebo jádra operačního systému), které může programátor využívat, více lit. [1].

API určuje, jakým způsobem se funkce mají volat ze zdrojového kódu programu. Samotné pouze specifikuje *rozhraní*, software (kód), který se stará o vykonávání funkcí popisovaných v API, se nazývá *implementace*. API bývá velmi často součástí SDK (Software Development Kit – sada nástrojů pro vývoj aplikací) a umožňuje programátorům vyvíjet aplikace pro konkrétní software nebo hardware.

2.2.1 Windows API (WinAPI)

Jako příklad jednoho z mnoha API je standardní API operačního systému Microsoft Windows navrženo pro programovací jazyk C a C++. Je obsaženo již v samotném systému a není nutné ho dodávat zvlášť. Nabízí programátorům mnoho možností jak využít prostředků operačního systému. Tyto by se daly rozdělit do několika kategorií:

- **Základní služby**

Přístup k nezbytným zdrojům poskytovaných systémem Windows. Obsaženy jsou: souborový systém, periferie, procesy a vlákna, registry Windows a ošetření chyb. Tyto funkce jsou uloženy na 16bitových Windows v souborech kernel.exe, krnl286.exe nebo krnl386.exe a na 32bitových Windows v kernel32.dll a advapi32.dll.

- **GUI (Graphical User Interface) - Grafické Uživatelské Rozhraní**

Funkce pro výstup grafického obsahu na monitory, tiskárny a jiná výstupní zařízení. Na 16-bitových Windows uloženo v gdi.exe a na 32bit Windows v gdi32.dll.

- **UI (User Interface) - Uživatelské Rozhraní**

Zahrnuje funkce pro tvorbu a řízení oken a dalších základních kontrolek jako jsou tlačítka a posuvníky, zpracovává vstup z periferií (například klávesnice, myš) a jiných funkcí spojených s GUI. Tato funkční jednotka se na 16bitových Windows nachází v souborech user.exe a na 32bitových Windows v user32.dll. Od Windows XP se základní prvky nachází v comctl32.dll, společně s běžnými prvky (Common Control Library).

- **Knihovna běžných dialogových oken**

Spadá do uživatelského rozhraní a poskytuje aplikacím standardní dialogová okna pro otevření a ukládání souborů, volbu barvy a fontů, apod. Knihovna je na 16bitových Windows uložena v souboru commdlg.dll a na 32bitových Windows v comdlg32.dll.

- **Knihovna Běžných Prvků (Common Control Library)**

Poskytuje aplikaci přístup k pokročilejším prvkům operačního systému. Zahrnuje věci jako stavový řádek, zobrazení průběhu výpočtu, toolbary a záložky. Knihovna je na 16bitových Windows umístěna v commctrl.dll a na 32bitových Windows v comctl32.dll.

- **Windows Shell**

Umožňuje aplikacím přístup k funkcím poskytovaných shellem Windows. Komponenta je na 16bitových Windows v shell.dll, později ve Windows 95 v shell32.dll a na 32bitových Windows v shlwapi.dll.

- **Síťové služby**

Poskytuje přístup k různým počítačovým sítím. Zahrnuje také NetBIOS, Winsock, NetDDE, RPC (Remote procedure call) a mnoho dalších funkcí.

2.3 ABI (Application Binary Interface)

ABI je funkčně stejné jako API s tím rozdílem, že je zkompilované a jako takové je schopné fungovat na různých operačních systémech (podporujících dané ABI). Aplikační rozhraní je definováno na strojové úrovni. Problémem jsou Unixové (a odvozené) systémy, kde se nepodařilo ABI standardizovat a je v některých případech nutné aplikaci pro daný systém portovat, viz lit. [1].

2.4 DDE (Dynamic Data Exchange)

Jednou z prvních metod meziprocesové komunikace ve Windows byla DDE (Dynamic Data Exchange), která umožňovala přijímat a posílat zprávy v takzvaných *konverzacích* mezi jednotlivými aplikacemi na bázi klient-server. Každá konverzace je definována třemi objekty:

- **Application – Jméno aplikace**

Identifikace serveru. Například „Excel“

- **Topic – téma konverzace**

Určuje objekt v rámci kterého se budou data přenášet, například pro aplikace operující s dokumenty je to obvykle jméno souboru. Společně se jménem aplikace jednoznačně identifikuje konverzaci.

- **Item – položka**

Konkrétní data vztahující se ke konverzaci. Lze použít jakýkoliv z předdefinovaných formátů dat používaných pro schránku, nebo nadefinovat vlastní formát dat.

Výhodou je možnost poslat libovolnému programu implementujícímu komunikaci DDE potřebná data, popřípadě je získat (například data z tabulky Excelu). Na druhé straně je nevýhodou jisté omezení, jelikož k této komunikaci je nutné, aby používaná aplikace běžela (v našem případě Excel), což není vždy možné, ani žádoucí viz lit.[1] a [2].

2.5 OLE (Object Linking and Embedding)

Základem pro OLE byla technologie DDE. OLE se stala jednou z hlavních technologií uvedených ve Windows 3.0 (roku 1992). Verze OLE 1.0 byla navržena speciálně pro tzv. *compound documents* (např. dokument wordu; obecně jakýkoliv

dokument obsahující kromě textu i další data jako listy nebo obrázky). Umožňovala vkládání objektů (např. dokumentů, obrázků) do jiných aplikací. Při editaci vloženého objektu se automaticky spustila instance mateřské aplikace (pokud již neběžela), ve které bylo následně možné vybraný objekt editovat.

Textové konverzace a systém zpráv Windows se však neukázaly být natolik flexibilní, aby bylo možné plně využít sdílení vlastností různých aplikací. To vedlo k přepracování technologie na verzi OLE 2.0 (rok 1993). Ta obsahovala již myšlenku komponentového software a byla navržena obecně pro jakékoliv softwarové komponenty. Klient byl nyní schopen využívat služby serveru ve svém vlastním prostředí.

Tuto funkcionalitu umožňovaly již předtím klasické DLL (Dynamic Linked Library), které obsahovaly funkce serveru, ale programátoři naráželi na mnohé problémy. Technologie OLE 2.0 všechny tyto řešila, ale byla komplikovaná a vhodná pouze pro určitý typ úloh.

V roce 1994 byly uvedeny OCX (OLE controls) jako nástupce VBX (Visual Basic Extensions). Téhož roku vydal Microsoft prohlášení, že OLE 2.0 bude známo pouze pod názvem „OLE“ a nebude již dále zkratkou, ale názvem pro všechny komponentové technologie Microsoftu, viz lit. [1] a [2].

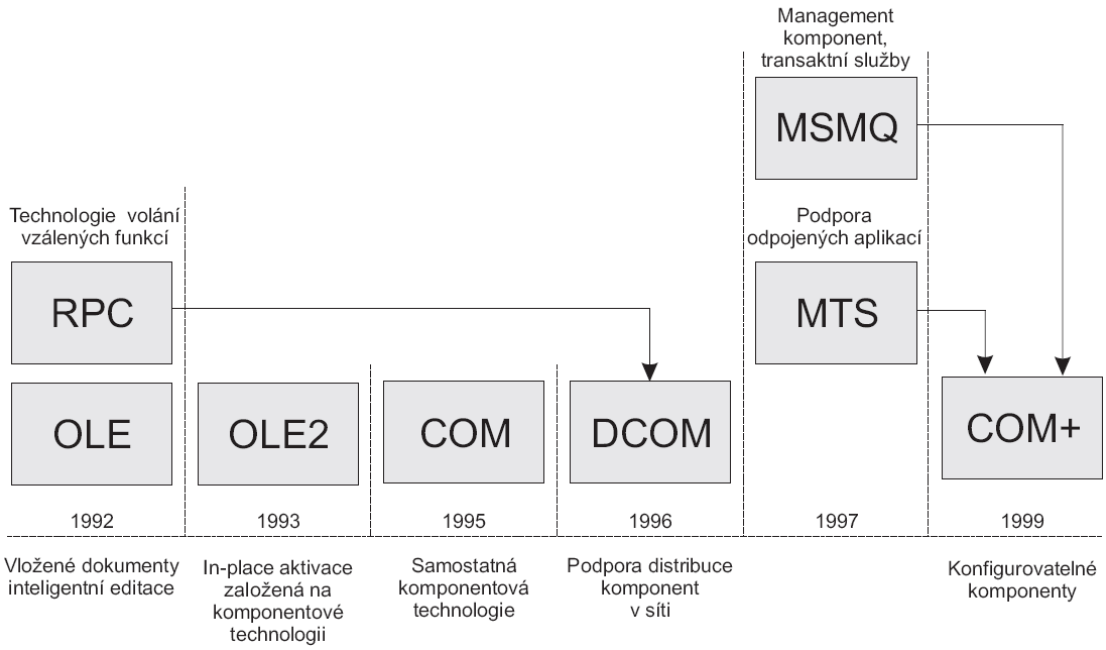
2.6 COM (Component Object Model)

2.6.1 Úvod

Komponentový model definuje specifické interakce mezi komponentami a kompoziční standardy. Implementace komponentového modelu je specializovaná množina spustitelných softwarových elementů vyžadovaná k podpoře spouštění komponent odpovídajících tomuto modelu. Komponentový model COM byl zveřejněn

firmou Microsoft v roce 1993, ačkoliv k jeho prosazování došlo o několik let později. Vývoj komponentových modelů je zachycen na obrázku níže (viz Obrázek 2.1).

Jako jeden z možných způsobů meziprocesové komunikace ve Windows má COM i další výhody, kterou je například rozdělení původně *monolitických aplikací* (tj. aplikací tvořené jedním zdrojovým textem, který implementuje jediný algoritmus, mnohdy velmi složitý a nepřehledný) na jednoznačně oddělitelné části. Jinou možností jak rozdělit program jsou DLL, avšak někdy nastávají problémy s verzemi potřebných komponent. Rozvoj komponentové tvorby softwaru tímto krokem neustal. COM technologie se stala také základem pro mnoho dalších. Detailně se těmito technologiemi zabývá lit. [2] a [3] .

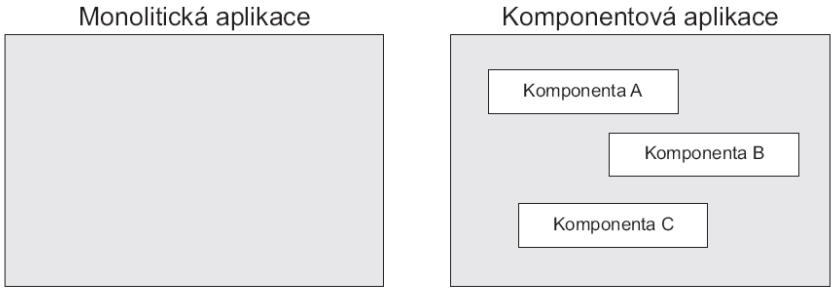


Obrázek 2.1: Historie komponentových modelů

2.6.2 Komponenty a rozhraní

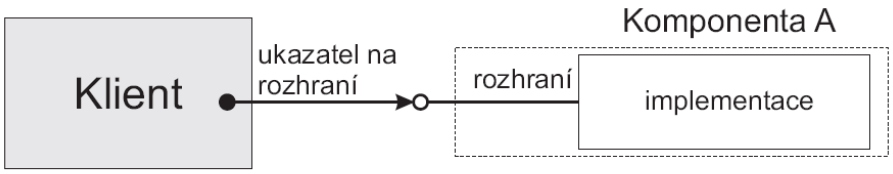
Softwarová komponenta je softwarový element, který odpovídá komponentovému modelu, může být nezávisle šířen a může se skládat s ostatními

komponentami podle definovaného kompozičního standardu bez jakýchkoliv změn. Softwarové komponenty tedy umožňují rozdělit aplikaci na jednotlivé části podle předem daného modelu, (viz Obrázek 2.2)



Obrázek 2.2: Monolitická a komponentová aplikace

Každá komponentová aplikace, neboli COM server, poskytuje pomocí instancí svých komponent služby jedné nebo více jiným aplikacím, tzv. COM klientům. Model COM popisuje, co musí splňovat klient a server a zajištění mechanismu jejich vzájemného propojení, bez tohoto předpokladu by nebylo možné využít všech výhod komponentového modelu. Každou komponentu COM lze rozdělit na dvě části - *interface* (rozhraní) a *implementaci* (viz Obrázek 2.3).



Obrázek 2.3: Komponenta a spojení s klientem

Komponenta může obsahovat více rozhraní a jim odpovídajících implementací. Z pohledu programovacích jazyků je interface seznamem metod a implementace definicí výkonných těl těchto metod. Rozhraní nesmí obsahovat žádná členská data,

která by mohla způsobit nekompatibilitu komponentové aplikace (jako se často stává u klasických DLL). Klientská aplikace pak získává pouze ukazatel na rozhraní dané instance komponenty serveru a není třeba, aby cokoliv věděla o implementaci tohoto rozhraní (viz Obrázek 2.3).

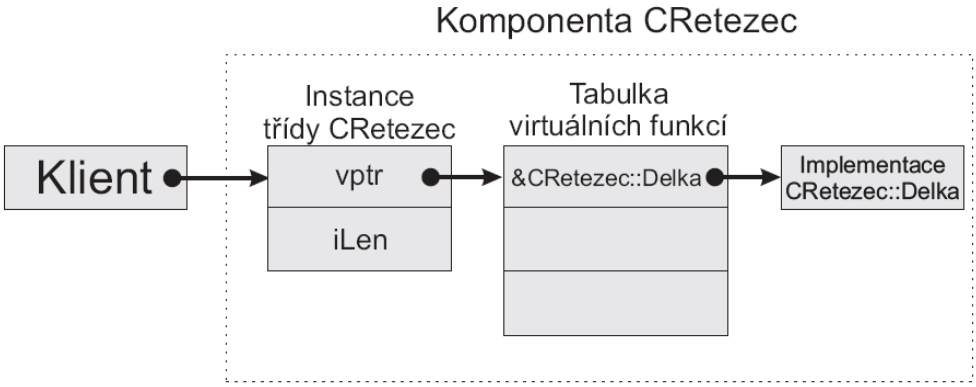
Ukázka implementace rozhraní komponenty v jazyce C++ pomocí abstraktní třídy. Zde si nadeklarujeme vlastní rozhraní:

```
class IRetezec //I před jménem označuje Interface
{
    public:
          virtual int Delka() = 0; //deklarace virtuální metody Delka
                                   //vracející hodnotu typu integer
}
```

A následně implementujeme:

```
class CRetezec : public IRetezec
{
    int iLen;  // proměnná iLen typu integer
    public:
          int Delka() {return iLen;};   //implementace metody Delka
                                        //vrací hodnotu proměnné iLen
    //dále pak doplněný konstruktor, destruktor ...
}
```

Vnitřně je tato komunikace realizována pomocí tabulek virtuálních funkcí. Každá abstraktní třída má tabulku virtuálních funkcí. Pro každou třídu existuje pouze jedna a je sdílena všemi instancemi třídy. Jednotlivé instance pak vlastní ukazatel na tabulku. Samotná tabulka obsahuje ukazatele na implementace všech virtuálních metod dané třídy. Klient pak ve skutečnosti získává ukazatel na ukazatel na implementaci dané metody. (viz Obrázek 2.4)



Obrázek 2.4: Realizace spojení klienta s komponentou

V každém programovacím jazyce podporujícím tvorbu komponent se deklarace rozhraní provádí různě a ostatní jazyky této deklaraci nerozumí. Proto je nutné vytvořit *binárně kompatibilní* rozhraní. Binární kompatibilita zaručuje, že komponenty mohou být po svém přeložení využívány i klienty vytvořenými v jiných programovacích jazycích.

2.6.3 Rozhraní definované COM

V předchozí kapitole jsme si ukázali deklaraci rozhraní v jazyce C++. Aby vyhovovala pravidlům COM, je třeba ji ještě upravit.

```
class IRetezec : public IUnknown
{
    public:
          virtual HRESULT __stdcall Delka(int *piVysledek) = 0;
}
```

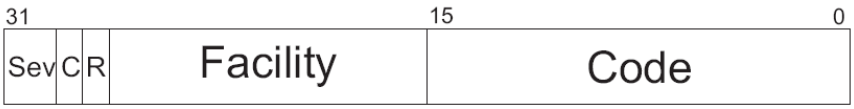
Změny oproti původní deklaraci:

- **volací konvence __stdcall**
- **návratová hodnota funkce typu HRESULT**
- **implementace rozhraní IUnknown**

Volací konvence __stdcall určuje, jak budou funkci předávány parametry a kdo se postará o odstraňování proměnných (volající, nebo volaná strana). V jazyce C++ je nutno ji použít, pokud chceme zajistit kompatibilitu s ostatními programovacími jazyky.

2.6.4 Návratová hodnota HRESULT

Návratová hodnota funkce typu HRESULT je 32bitové číslo obsahující kód, který popisuje úspěšnost metody. Všechny kódy používané systémem Microsoft Windows lze najít v hlavičkovém souboru winerror.h. Pokud nevyhovují, nebo je třeba některé dodefinovat, je možné vytvořit vlastní hlavičkové soubory.



Obrázek 2.5: Struktura kódu HRESULT

Bity C (Customer Code Flag), resp. R (Reserved Bit) jsou rezervovány pro Microsoft a mají konstantní hodnotu 1, resp. 0.

Bity Sev (Severity Code) můžou nabývat 4 hodnot :

- **Success(0)**
- **Informational(1)**
- **Warning(2)**
- **Error(3)**

Bity zdroje události (Facility) musí být pro uživatelské kódy nastaveny na hodnotu FACILITY ITF(4), zbytek je rezervován pro Microsoft.

Z bitů pro vyjádření konkrétní chyby (Code), lze pro vlastní návratové hodnoty použít rozsah 0x0200-0xFFFF. Každý návratový kód by měl mít nadefinovanou symbolickou hodnotu, která obvykle obsahuje první písmeno nebo dokonce celé slovo z hodnot Sev, např. S OK (0x00000000), E NOINTERFACE (0x80004002). Tyto

návratové hodnoty používá většina metod, ale jsou i výjimky, např. metody AddRef() a Release(). Původní návratovou hodnotu funkce typu int (integer) z našeho příkladu lze nyní předat odkazem přes ukazatel piVysledek.

2.6.5 Rozhraní IUnknown

Rozhraní IUnknown je základním rozhraním každé komponenty. Řídí životnost instance komponenty a umožňuje klientům dotázat se na rozhraní, která komponenta podporuje. Každá komponenta musí implementovat metody IUnknown.

- **AddRef()**

Inkrementuje počítadlo referencí, neboli počet odkazů na instanci komponenty od všech připojených klientů. Metoda vrací aktuální počet referencí.

- **Release()**

Dekrementuje počítadlo referencí. Pokud počítadlo klesne na hodnotu 0, komponenta se může sama odstranit z paměti. Metoda vrací aktuální počet referencí.

- **QueryInterface()**

Slouží k dotazu na existující rozhraní implementované komponentou.

Nyní můžeme náš příklad opět rozšířit:

```
class CRetezec : public IRetezec
{
    int iLen;
    ULONG ulRefCount;
    public:
          //metody rozhrani IRetezec
          HRESULT __stdcall Delka(int *piVysledek);
          //metody rozhrani IUnknown
```

```
            ULONG __stdcall AddRef(VOID);
            ULONG __stdcall Release(VOID);
            HRESULT __stdcall QueryInterface (REFIID riid,
                  LPVOID *ppvObject);
    //dále pak doplněný konstruktor, destruktor ...
}
```

Implementace konstruktoru a destruktoru je závislá na realizaci komponenty a na jejím *apartmentu*. Ostatní metody lze implementovat třeba takto :

```
HRESULT __stdcall CRetezec::Delka (int *piVysledek)
{
    if (!piVysledek)
          return E_INVALIDARG;
    *piVysledek = iLen;
    return S_OK;
}
```

Implementace metody Addref():

```
ULONG __stdcall CRetezec::AddRef (VOID)
{
    ++ulRefCount;
    return ulRefCount;
}
```

Implementace metody Release():

```
ULONG __stdcall CRetezec::Release(VOID)
{
    if (--ulRefCount != 0)
          return ulRefCount;
    delete this;
    return 0;
}
```

Implementace metody QueryInterface():

```
HRESULT __stdcall CRetezec::QueryInterface(IID riid,
    LPVOID *ppvObject)
{
    if (!ppvObject)
          return E_INVALIDARG;

    if (riid == IID_IRetezec)
    {
          *ppvObject = (IRetezec*) this;
    }
    else
    {
          if (riid == IID_IUnknown)
          {
                *ppvObject = (IUnknown*) this;
          }
          else
          {
                *ppvObject = NULL;
                return E_NOINTERFACE;
          }
    }
    AddRef();
    return S_OK;
}
```

Metoda QueryInterface() musí splňovat následující požadavky:

- **rozhraní IUnknown je jedinečné**

Vždy získáme stejné rozhraní IUnknown.

- **výsledek je vždy stejný**

Pokud se nám podaří pomocí QueryInterface() získat ukazatel na rozhraní, musí se nám to podařit i pří příštím pokusu. Naopak, pokud jednou metoda při dotazu na rozhraní skončí neúspěšně, ani v budoucnu úspěšná nebude.

- **metoda je reflexivní**

Můžeme vyžádat ukazatel na rozhraní, které již máme.

- **metoda je symetrická**

Kdykoliv se můžeme vrátit k rozhraní, ze kterého jsme vyšli. Máme-li např. ukazatel na rozhraní IUnknown a pomocí jeho metody QueryInterface() získáme rozhraní IRetezec, musíme být schopni pomocí rozhraní IRetezec a jeho metody QueryInterface() získat zpátky ukazatel na rozhraní IUnknown.

- **metoda je tranzitivní**

Vždy se můžeme dostat k danému rozhraní z jiného libovolného rozhraní téhož objektu, pokud jsme se k němu již odněkud dostali.

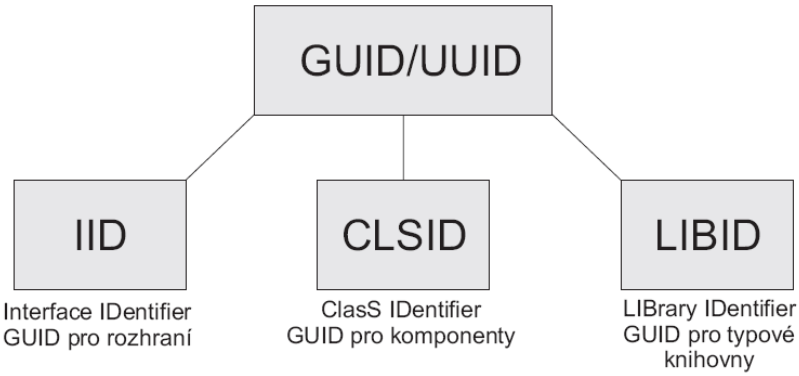
- **metoda automaticky inkrementuje počítadlo referencí**

Toho je docíleno voláním metody AddRef(). O dekrementaci počítadla se pak musí postarat klient zavoláním metody Release().

2.6.6 GUID (Globally Unique Identier)

Datová struktura zastupující celosvětově jedinečné 128bitové číslo ve formátu {3F2504E0-4F89-11D3-9A0C-0305E82C3301}. Obecně se tomuto číslu říká GUID (Globally Unique IDentifier) nebo UUID (Universally Unique IDentifier). Zkratka IID (Interface IDentifier) se většinou používá v kontextu s identifikací rozhraní. Nejen rozhraní, ale i komponenty a další objekty musí mít své GUID, pro každý z nich se číslu GUID říká jinak. Záleží na typu objektu, jaká zkratka se použije (viz Obrázek 2.6), struktura čísla je ale stejná. Firma Microsoft nabízí různé nástroje, které umí generovat

GUID. Jsou to např. programy guidgen.exe nebo uuidgen.exe.



Obrázek 2.6: Přehled identifikátorů

2.6.7 IDL (Interface Definition Language)

Jazyk IDL byl původně navržen organizací OSF (Open Software Foundation), která ho využívala pro metody vzdáleného spouštění funkcí RPC. IDL i RPC byly firmou Microsoft přejaty, rozšířeny a využity v technologiích COM a DCOM. IDL není plnohodnotným programovacím jazykem. Slouží pouze k popisu rozhraní komponenty a k přeložení tohoto rozhraní do binárně kompatibilní formy. Vlastní implementace rozhraní komponenty se již provede v C++, Java, VB apod. Ukážeme si, jak přepsat deklaraci našeho příkladu pomocí jazyka IDL.

```
import "unknwn.idl";
[object,uuid (12345678 - 1234 - ABCD - 1234 – 123456789012)]

interface IRetezec : IUnknown
{
    HRESULT Delka ([out, retval] int *piVysledek);
}
```

Deklarace se skládá z těchto hlavních částí

- **výraz import "unknwn.idl"**

Přiloží obsah souboru unknwn.idl, což je IDL popis rozhraní IUnknown.

- **Object**

Definujeme rozhraní

- **uuid**

GUID identifikující rozhraní

- **interface**

Název rozhraní : jméno rodičovského rozhraní.

Deklarace vlastní metody Delka(). Základní atributy, které jí můžeme přiřadit:

- **[in]**

Data jsou zajištěna volající stranou a přenesena pouze v jednom směru - od klienta ke komponentě

- **[out]**

Data jsou naplněna volanou stranou a přenáší se od komponenty ke klientovi

- **[in, out]**

Parametr je přenášen oběma směry

- **[out, retval]**

Stejné jako [out], jen s konkrétním označením návratové hodnoty pro jazyky jako VB nebo Java

Abychom mohli vygenerovat binárně kompatibilní rozhraní (typovou knihovnu), je třeba ještě přidat tento kód:

```
[uuid (12345678-0000-0000-0000-000000000003),
helpstring("Retezec Type Library"),
version (1.0)]
library Retezec
{
    importlib("stdole32.tlb");
    interface IRetezec;
    [uuid(12345678-0000-0000-0000-000000000002]
    coclass CGenerator
    {
          interface IGenerator;
    }
}
```

Funkce jednotlivých atributů:

- **uuid**

LIBID identifikující typovou knihovnu. (Podruhé GUID komponenty)

- **helpstring**

Výraz blíže definující jméno typové knihovny. Tento výraz se pak zobrazuje v různých OLE prohlížečích

- **version**

Výraz definující verzi knihovny, taktéž zobrazovaný v OLE prohlížečích

- **library**

Název souboru knihovny - typová knihovna bude uložena v souboru "Retezec.tlb", vše pod tímto řádkem bude součástí knihovny

- **importlib**

Vloží již hotovou knihovnu - stdole32.tlb, která obsahuje definice standardních rozhraní (IUnknown apod.)

- **interface**

Vloží do knihovny definici rozhraní.

- **coclass**

Definuje komponentu CGenerator, ve složených závorkách je pak seznam rozhraní, která komponenta implementuje.

K vygenerování typové knihovny již stačí jen zkompilovat.

2.7 Další komponentové modely

2.7.1 CORBA (Common Object Request Broker Architecture)

Standard definovaný OMG (Object Management Group), umožňující spolupráci komponent napsaných v různých programovacích jazycích a jejich běh na vzdálených počítačích. Toho je docíleno použitím IDL a následným „mapováním“ na konkrétní programovací jazyk. Konkurencí byl model DCOM, více lit. [1].

2.7.2 DCOM (Distributed Component Object Model)

Původně nazývaný „Network OLE“. Poskytuje nadstavbové služby pro zavádění a komunikaci s komponentami na vzdálených počítačích. K modelu COM přidával *marshalling* a distribuovaný *garbage collector*. Společně s CORBA měli za cíl použití při komunikaci přes internet, což se především kvůli potížím s firewally nepodařilo, viz lit. [1].

2.7.3 COM+

Model postavený na COM a DCOM s využitím dalších služeb: MTS (Microsoft Transaction Server), prostředí řešící problémy současné obsluhy více klientů, řízení bezpečnosti, administrace a robustního ošetření chybových stavů, a MSMQ (Microsoft Message Queue Server), produkt který umožňuje zaznamenávat požadavky na straně klienta do tzv. fronty zpráv a spolehlivě je přenášet na stranu serveru i při poruchách sítě. Výhodou byla možnost spuštění v tzv „komponentových farmách“ (správně napsaná komponenta mohla být znovu použita inicializací bez nutnosti mazat ji z paměti) a volání komponent na vzdálených počítačích (což bylo dříve možné pouze pomocí DCOM), viz lit. [1].

2.7.4 OPC (OLE for Process Control)

Průmyslový standard postavený na technologii COM, určený k zprostředkování komunikace mezi aplikacemi Windows a konkrétním hardware používaným ve výrobě. Cílem OPC je definovat společné rozhraní, které by bylo dále použito v různých odvětvích. Jakmile je jednou napsán server pro dané zařízení, může ho libovolný software využívající OLE (COM) technologii použít. Více lit. [2] a [3].

2.7.5 ActiveX

ActiveX je sada internetových (intranetových) technologií uvedených Microsoftem na konci 90. let minulého století. Jako mnoho dalších spadá pod jednotné označení COM. Termín ActiveX se sám o sobě dnes již příliš nepoužívá a mnoho technologií bylo přesunuto, nebo přejmenováno. Idea Microsoftu zapouzdřit aktivní obsah na webových stránkách pomocí komponent COM/ActiveX (např. místo Java appletů) společně s problémy s Internet Explorerem (jako jediný umí pracovat s těmito komponentami) vedla k otevření cesty mnohým virům a trojským koním. Je to

především proto, že COM a ActiveX komponenty jsou spouštěny jako nativní kód na hostitelském stroji a mají minimální omezení své činnosti, viz lit [1].

Pomineme-li špatnou pověst této technologie, je to jedno z možných řešení spojení s MS Office, jelikož ty standardně podporují vkládání ovládacích prvků ActiveX. Bohužel od podpory ActiveX a možnosti vytvářet vlastní ActiveX komponenty bylo ve verzi Visual Studia 2005 zcela upuštěno, není možné tímto způsobem propojit přímo .NET a MS Office. Visual Studio sice nabízí možnost interoperability, ale funkčnost takovýchto prvků je velmi omezená a především nespolehlivá (není zaručena na všech strojích a za každých podmínek).

Zcela bez problémů lze takové prvky vytvořit v prostředí Visual Basic 6.0, který však není součástí .NET a nelze tak využít jeho potřebné funkce. Jistou možností je spouštět externí aplikaci .NET z kódu komponenty ActiveX vytvořené ve Visual Basicu. Jako příklad (Příloha A) uvádím jednoduchou kontrolku s jedním tlačítkem, po jehož stisknutí se zobrazí okno s informací o stisknutí. Tu je možné jednoduše přidat do MS Office. Výsledný projekt je zkompilován jako soubor .ocx

2.8 .NET

.NET je souborný název pro technologie v softwarových produktech, tvořící celou novou platformu, která je dostupná pro Web, MS Windows, mobilní Pocket PC i Unixové systémy. Se svým příchodem odsunula do pozadí technologii COM. Microsoft však neuvažuje o ukončení COM, ale ani o jeho dalším rozvoji a podpoře. V některých případech COM stále zůstává zásadním nástrojem, například DirectX SDK je založen na technologii COM.

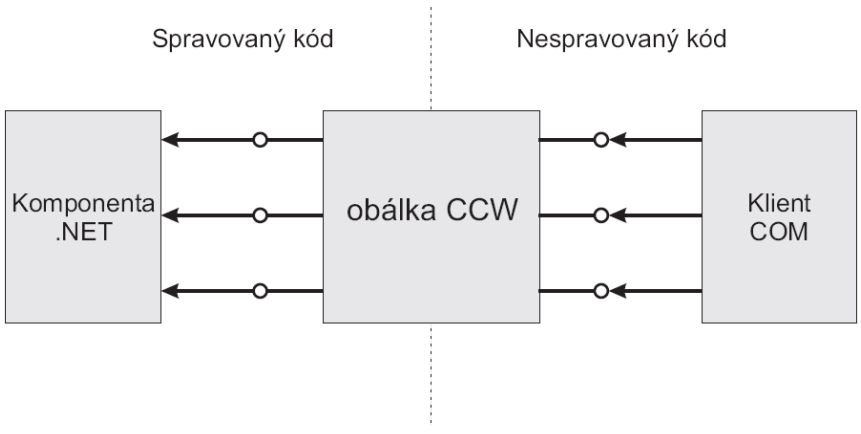
Pro podporu vývoje aplikací v .NET vydal Microsoft Visual Studio .NET (momentálně ve verzi 2005), které je oproti původnímu VS rozšířeno návrh webových XML služeb .NET a .NET Framework (pro platformu Pocket PC – Compact Framework), zajišťující prostředí potřebné pro běh aplikací a nabízející jak spouštěcí rozhraní, tak potřebné knihovny.

.NET Framework je možné použít v OS MS Windows od verze 98 a je k dispozici zdarma jako samostatná komponenta, která se do systému doinstaluje. .NET Compact Framework pro platformu Pocket PC s operačním systémem Windows Mobile umožňuje vytvářet kompatibilní aplikace pro PDA. Compact Framework obsahuje méně metod a knihoven z důvodu úspory místa, ale pokud používáme pouze standardní prvky, je taková aplikace spustitelná jak na PDA, tak na PC. V opačném případě je třeba pro každou platformu udělat aplikaci zvlášť.

Díky standardizaci .NET existuje i jeho GNU obdoba a nazývá se DotGNU. Její část DotGNU Portable.NET umožňuje spouštět všechny .NET aplikace na unixových platformách. V prostředí operačních systémů Linux, UNIX a Mac OS X je k dispozici i sada nástrojů kompatibilní přímo s Microsoft .NET pod názvem Mono (opensource projekt). Detaily se zabývá lit. [4]

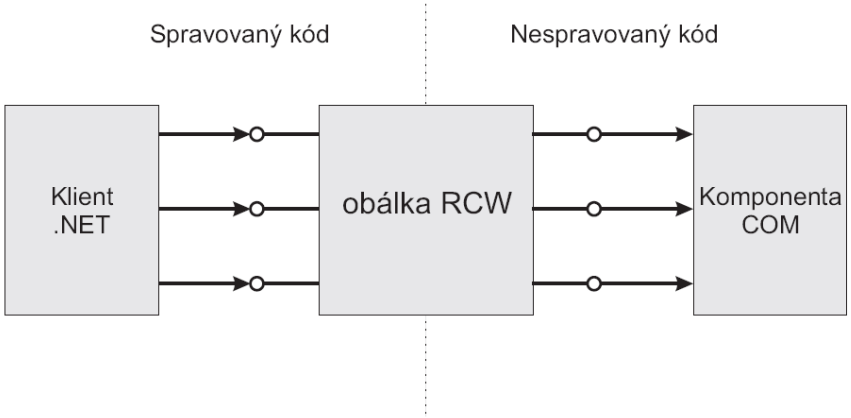
2.8.1 Interoperabilita

IEEE (Institute of Electrical and Electronics Engineers) definuje interoperabilitu jako schopnost dvou a více systémů nebo komponent vyměňovat informace a následně je použít.



Obrázek 2.7: Komunikace COM s .NET pomocí obálky

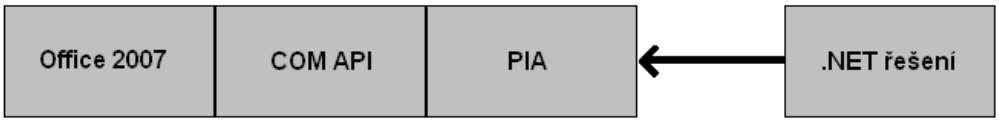
Pokud budeme hovořit o COM a .NET, je možné jednotlivé technologie vzájemně použít (.NET v COM a naopak). Toho je docíleno pomocí *obálek* (wrappers). Takováto obálka překládá specifická volání vysílaná klientem na požadavky specifického volání metod komponenty modelu. Při komunikaci .NET s COM je použita RCW (Runtime Callable Wrapper) obrácená komunikace zas vyžaduje CCW (COM Callable Wrappers) (ukázka viz Obrázek 2.8 a Obrázek 2.7). Detailněji je tato problematika rozvedena v lit. [5].



Obrázek 2.8: Komunikace .NET s COM pomocí obálky

3. VSTO (Visual Studio Tools for Office)

VSTO je vývojářský balík (SDK) pro Visual Studio a umožňuje vytváření .NET řešení v rámci MS Office. Je ke stažení na na internetových stránkách Microsoft Corporation a nainstaluje se jako doplněk VS. Podporován je MS Office minimálně ve verzi 2003. Aby bylo možné přidávat vlastní rozšíření (add-ins), je nutné při instalaci MS Office zvolit u jednotlivých produktů podporu PIA (Primary Interop Assemblies – podpora programovatelnosti v rozhraní .NET). S jeho využitím pak dochází ke spojení .NET a MS Office (viz Obrázek 3.1) .



Obrázek 3.1: Realizace propojení .NET a MS Office v rámci VSTO

VSTO prošel již několika verzemi, s každou přibývá podpora dalších aplikací z MS Office:

- **VSTO 2003**

První verze. Podporován byl pouze Word 2003 a Excel 2003. Striktní bezpečnostní model zakazoval spouštění .NET kódu, bylo nutné explicitně potvrdit jeho věrohodnost.

- **VSTO 2005**

Podpora aplikací Word 2003, Excel 2003, InfoPath 2003 a Outlook 2003. Možnost vytváření vlastních ovládacích prvků. Add-iny byly pouze na úrovni konkrétního dokumentu vytvořeného ve VS.

- **VSTO 2005 SE**

Aktuální verze. Přibyla podpora Word 2007, Excel 2007, InfoPath 2007, Outlook 2007, Visio 2007 a PowerPoint 2007. Aplikace z balíku 2003 jsou sice podporovány, avšak některé funkční prvky pro tyto aplikace již chybí. Add-iny se již vytváří na úrovni celé aplikace, je možné také vytvářet add-in pro více aplikací.

- **VSTO 3.0**

Již je k dispozici beta verze a Microsoft slibuje podporu Access 2007, tvorbu vlastního uživatelského rozhraní a podporu mobilních zařízení.

Tento nástroj nabízí jak tvorbu vlastních ovládacích prvků (např. TaskBar, ActionPane, CustomPane a v Office 2007 i kompletní editaci uživatelského rozhraní Ribbon pomocí XML souboru), tak i možnost práce s daty v konkrétních dokumentech, jejich načítání i zapisování. Detailní popis jednotlivých funkcí a vlastností je k dispozici v lit. [6].

Výhodou těchto add-inů je jejich schopnost vykonávat libovolný .NET kód. Programátor je sice trochu omezen ohledně ovládacích prvků, jelikož lze přidávat pouze prvky standardní ve smyslu jejich standardní pozice a zobrazení v dané aplikaci. Toto řešení tvorby vlastní kontrolky se však ukázalo jako nejlepší možné, především kvůli již zmiňované podpoře .NET a nativní podpoře MS Office. Příklad využití je uveden v příloze C, kdy na standardní kontrolku vykreslujeme barvu pomocí OpenGL.

4. Praktické řešení

První pokus vložit do MS Office vlastní kontrolku byl realizován pomocí ActiveX. MS Office ho standardně podporuje a nabízí registraci a vkládání ovládacích prvků. Problémem je, že VS od verze 2005 zcela upustilo od podpory ActiveX a není tedy možné vytvářet projekty s touto technologií. Vytvořit ActiveX kontrolku lze ve Visual Basicu verze 6.0, ten ale není součásti VS .NET a je tedy v našem případě obtížně použitelný, jelikož celý projekt GREG je realizován v .NET. Jistou možností by bylo pokusit se spouštět v rámci kontrolky externí aplikaci .NET. Tato kontrolka se nachází v příloze A.

Druhou možností bylo obejít chybějící podporu ActiveX pomocí přímého zápisu hodnot do registrů MS Windows. Toto řešení funguje v testovací aplikaci VS a Internet Exploreru. Dokonce je možné takovouto kontrolku použít na vykreslování pomocí OpenGL. MS Office však toto řešení neakceptuje. Kontrolku nelze přidat jako dynamickou knihovnu, jelikož neobsahuje prvky ActiveX a při pokusu o vložení do dokumentu se zobrazí chybová hláška. Toto řešení je v příloze B.

Poslední možností, která se objevila až v nedávné době, je použití balíku VSTO. Tento SDK je koncipován přímo na práci s aplikacemi MS Office a nabízí tvorbu a úpravu ovládacích prvků a práci s dokumenty. Tímto způsobem je možné přidat svoji kontrolku (avšak pouze v rámci standardních ovládacích prvků) a na tu opět provádět vykreslování pomocí OpenGL. Ukázka je v příloze C.

Ačkoliv se podařilo nalézt několik možných způsobů, každý má své výhody a omezení. Vykreslování 3D scény není ještě zcela bez problémů a na jejich odstranění se bude v budoucnu pracovat.

5. Závěr

Všechny zde uvedené technologie (především COM, ActiveX, VSTO) umožňují komunikaci mezi jednotlivými aplikacemi a lze je použít i v jiných případech, než kterými se zde zabývám. Vytváření vlastních kontrolek a jejich vkládání do MS Office se však ukázalo být velmi obtížným, jelikož Office s některými z nich nekomunikuje, nebo je v některých případech nekorektně zobrazuje. Problémem jsou také různé verze MS Office (2003 a 2007) a VSTO (2005 a 2005 SE), jelikož nejsou vzájemně zcela kompatibilní a je třeba vytvářet kontrolku (add-in) pro každou verzi jiným způsobem. Nelze tedy jeden doplněk použít v libovolném balíku.

Navzdory těmto omezením se podařilo primární cíl, vytvoření prvku zobrazujícího pracujícího s OpenGL, realizovat a příkladný zdrojový kód se nachází v přílohách. Tato práce slouží jako základní přehled možností jak propojit libovolné aplikace. Detailní popis jednotlivých technologií kvůli svému rozsahu nemohl být obsažen a je tedy nutné v případě zájmu konkrétní technologii blíže nastudovat.

6. Použitá literatura

[1] *Wikipedia The Free Encyclopedia*. http://en.wikipedia.org/wiki, získáno dne 20.3.2007

[2] Šťavík O., *OPC server*. Diplomová práce. Praha2006.

[3] Mastný R., *Technologie COM a OPC*. Diplomová práce. Praha2002.

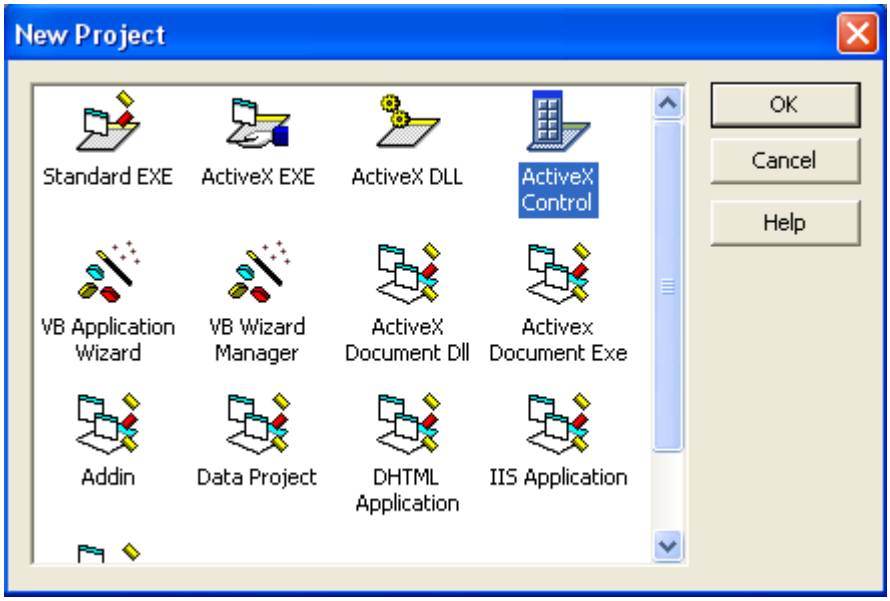
[4] Microsoft Corporation, *The .NET Show*. http://msdn.microsoft.com/theshow/, získáno dne 18.4.2007

[5] Troelsen A., *COM and .NET interoperability*. 2002. ISBN 1-59059-011-2

[6] Microsoft Corporation, *Visual Studio 2005 Tools for Office Second Edition*. http://msdn2.microsoft.com/en-us/office/aa905543.aspx, získáno dne 18.4.2007

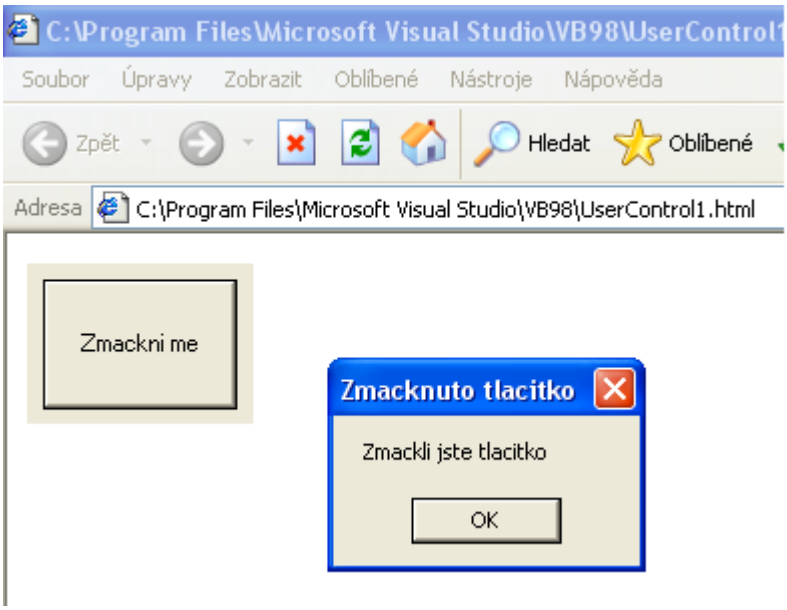
Příloha A - ActiveX kontrolka ve VB

Příklad popisuje vytvoření kontrolky ve Visual Basic 6.0. Nejprve založíme nový projekt ActiveX Control (viz Obrázek A.1). Poté napíšeme zdrojový kód uvedený níže a celý projekt zkompilujeme jako soubor .ocx. Jeho funkčnost v jednotlivých aplikacích je znázorněna na obrázcích.

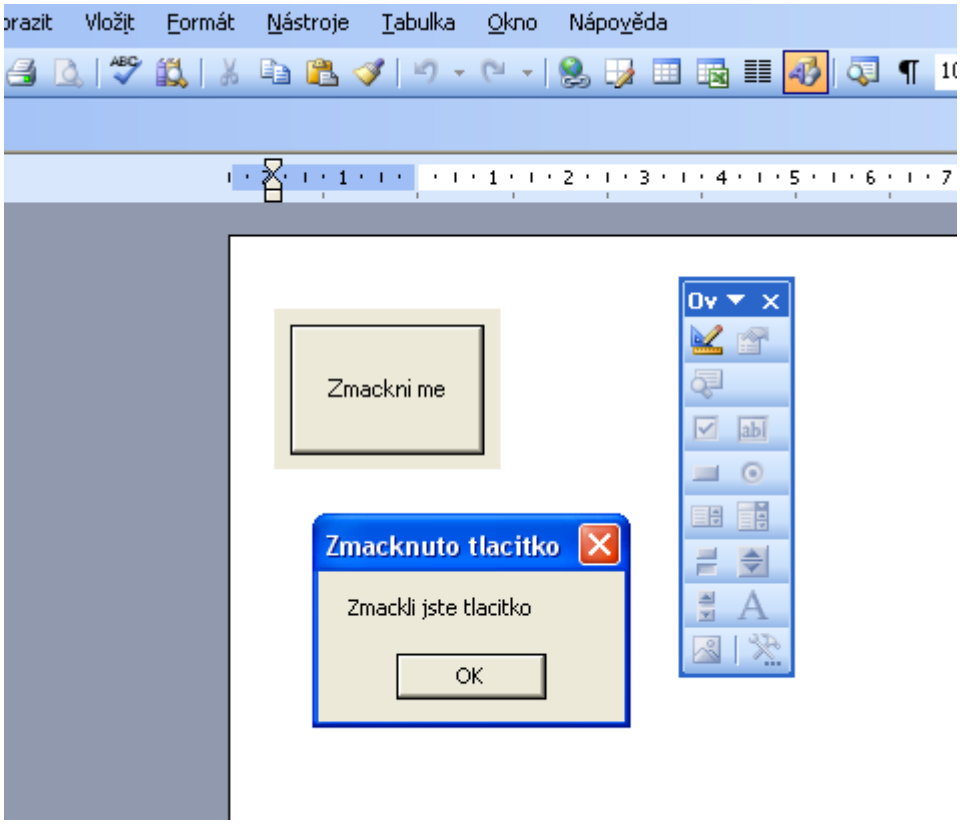


Obrázek A.1: Založení nového projektu ActiveX kontrolky v VB 6.0

Jak je vidět do Internet Exploreru (viz Obrázek A.2) můžeme bez problému kontrolku přidat a funguje bez jakýchkoliv omezení. Zobrazeno je také okno s hláškou o stisknutí tlačítka. To samé platí i pro MS Office (viz Obrázek A.3), kde je ukázka sešitu Word a přidané kontrolky, vše je stejně funkční.



Obrázek A.2: Naše kontrolka v Internet Exploreru



Obrázek A.3: Kontrolka umístěná v MS Word 2003

Zde je zdrojový kód ukázkové kontrolky ve VB 6.0:

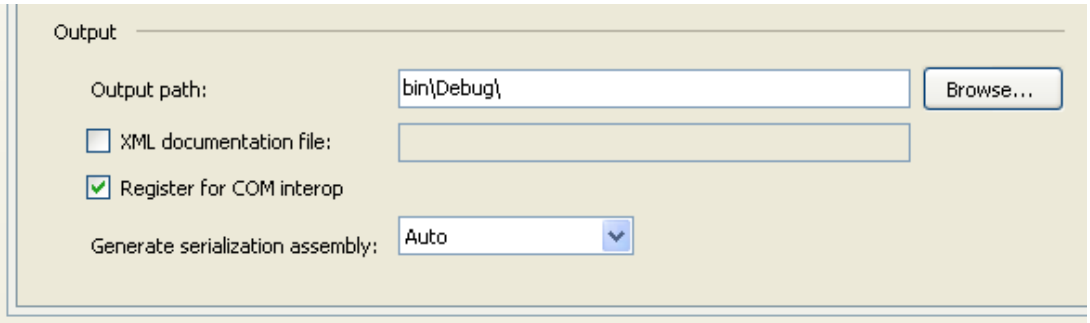
```
Begin VB.UserControl UserControl1
   ClientHeight    =   1200
   ClientLeft      =   0
   ClientTop       =   0
   ClientWidth     =   1695
   ScaleHeight     =   1200
   ScaleWidth      =   1695
   Begin VB.CommandButton Tlacitko
      Caption         =   "Zmackni me"
      Height          =   975
      Left            =   120
      TabIndex        =   0
      Top             =   120
      Width           =   1455
   End
End
Attribute VB_Name = "UserControl1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Private Sub Tlacitko_Click()
    MB = MsgBox("Zmacknuto tlacitko", vbOKOnly, "Zmacknuto")
End Sub
```

Za zmínku pak stojí ještě část projektového souboru, kde jsou uloženy informace o GUID a jménu kontrolky.

```
Type=Control
UserControl=MojeKontrolka.ctl
Reference=*\G{00020430-0000-0000-C000-000000000046}
#2.0#0#C:\WINDOWS\system32\stdole2.tlb#OLE Automation
```

Příloha B - ActiveX kontrolka v C#

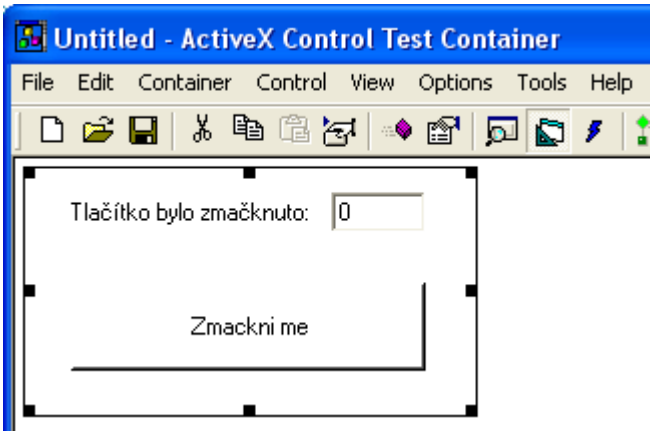
Tento příklad využívá zápis do registrů, čímž trochu obchází chybějící podporu přímé tvorby ActiveX prvků ve VS 2005. Důležité je ve vlastnostech projektu nastavit registraci komponenty pro Interop služby (viz Obrázek B.1).



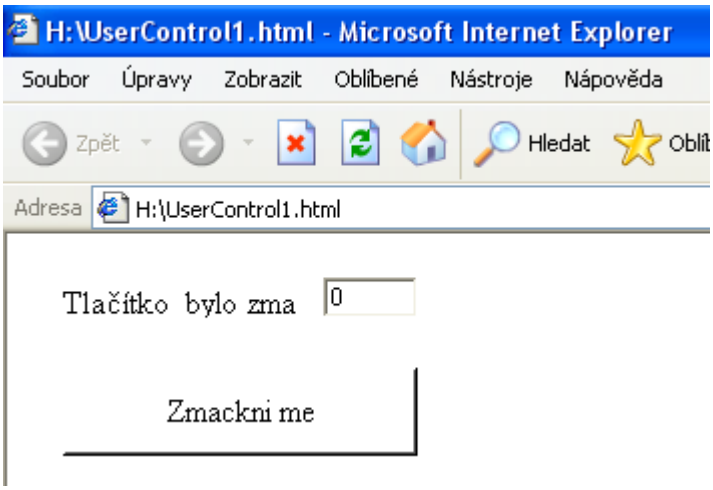
Obrázek B.1: Nastavení vlastností projektu

Ačkoliv výsledná kontrolka funguje v testovací aplikaci VS (viz Obrázek B.3) a Internet Exploreru (viz Obrázek B.4), MS Office nejsou schopny takto vytvořený objekt přijmout a vrací chybovou hlášku (viz Obrázek B.5)

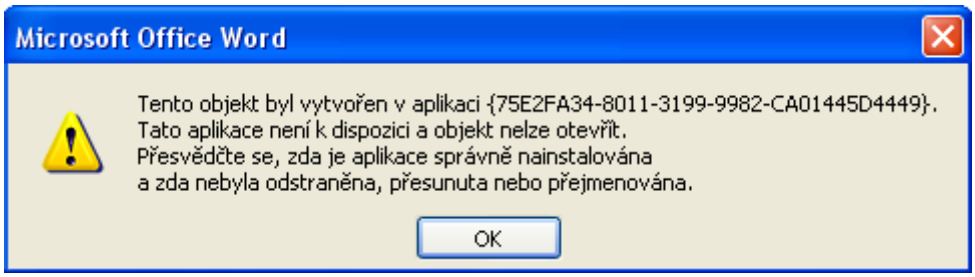
Obrázek B.2: Návrh kontrolky ve VS 2005



Obrázek B.3: Kontrolka v testovaní utilitě VS



Obrázek B.4: Kontrolka v Internet Exploreru



Obrázek B.5: Chybová hláška při pokusu o vložení do Wordu 2003

Zdrojový kód příkladu:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;

// Pro náš příklad je třeba přidat tyto knihovny
using System.Runtime.InteropServices;
using System.Text;
using System.Reflection;
using Microsoft.Win32;

namespace VlastniCOMKontrolka
{
    ///   <summary>
    ///   VlastniCOMKontrolka
    ///   </summary>
    [ProgId("PrikladCOM.VlastniCOMKontrolka")]
    [ClassInterface(ClassInterfaceType.AutoDual)]
    public class VlastniCOMKontrolka :
          System.Windows.Forms.UserControl
    {
      private Button tlacitko;
      private TextBox pocetKliknutiText;
      private Label popisek;
      private int pocetKliknuti = 0;
```

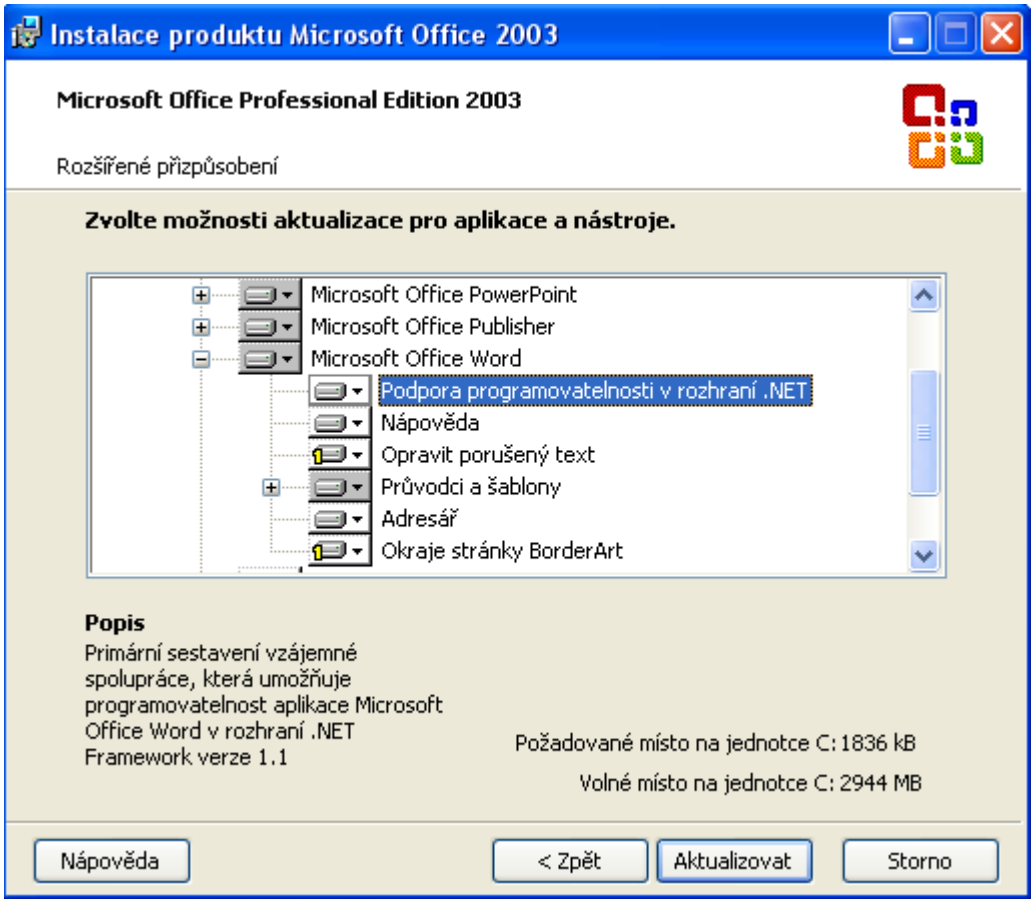
```
        private System.ComponentModel.Container components = null;

        public VlastniCOMKontrolka()
        {
    // This call is required by the     Windows.Forms Form Designer.
              InitializeComponent();
          // TODO: Add any initialization after the InitForm call
        }

            ///   <summary>
            ///   Pri odstraneni kontrolky uklidime pouzivane zdroje
            ///   </summary>
            protected override void Dispose(bool disposing)
            {
                  if(disposing)
                  {
                        if(   components != null )
                              components.Dispose();
                  }
                  base.Dispose(disposing);
            }

            //Component Designer generated code


            public void Pripocti()
            {
               pocetKliknuti++;
               pocetKliknutiText.Text = pocetKliknuti.ToString() ;
```

```
        }

        ///   <summary>
        ///   Zaregistrujeme tridu jako kontrolku a nastavime
        ///   hodnoty registru
        ///   </summary>
        ///   <param name="klic">klic registru</param>
        [ComRegisterFunction()]
        public static void RegistrujTridu(string klic)
        {
              // Odzrizneme cast klice - HKEY_CLASSES_ROOT\ jelikoz
ho neni treba
              StringBuilder sb = new StringBuilder(klic) ;
              sb.Replace(@"HKEY_CLASSES_ROOT\","") ;

              // Otevreme klic CLSID\{guid}
              RegistryKey rk    =
Registry.ClassesRoot.OpenSubKey(sb.ToString(),true);

           // Vytvorime klic 'Control' zajisti zobrazeni v nabidce
          // kontrolek
          RegistryKey kontrolka = rk.CreateSubKey("Control");
          kontrolka.Close();

              // Dale pak CodeBase
              RegistryKey inprocServer32 =
rk.OpenSubKey("InprocServer32", true);
              inprocServer32.SetValue("CodeBase",
Assembly.GetExecutingAssembly().CodeBase);
              inprocServer32.Close();
```

```
                // Zavreme cely klic
                rk.Close();
            }
            ///   <summary>
            ///   Odregistrace kontrolky
            ///   </summary>
            ///   <param name="klic">klic registru</param>
            [ComUnregisterFunction()]
            public static void OdregistrujTridu(string klic)
            {
                StringBuilder sb = new StringBuilder(klic);
                sb.Replace(@"HKEY_CLASSES_ROOT\","");
                // Smazeme vsechny zapsane hodnoty - false zajisti,
    // ze nebude v pripade jejich neexistence vyvolana vyjimka
                RegistryKey rk =
    Registry.ClassesRoot.OpenSubKey(sb.ToString(),true);
                rk.DeleteSubKey("Control",false);
                RegistryKey inprocServer32 =
    rk.OpenSubKey("InprocServer32",true);
                rk.DeleteSubKey("CodeBase", false);
                // Zavreme cely klic
                rk.Close();
            }
          private void tlacitko_Click(object sender, EventArgs e)
          {
              Pripocti();
          }
    }
}
```

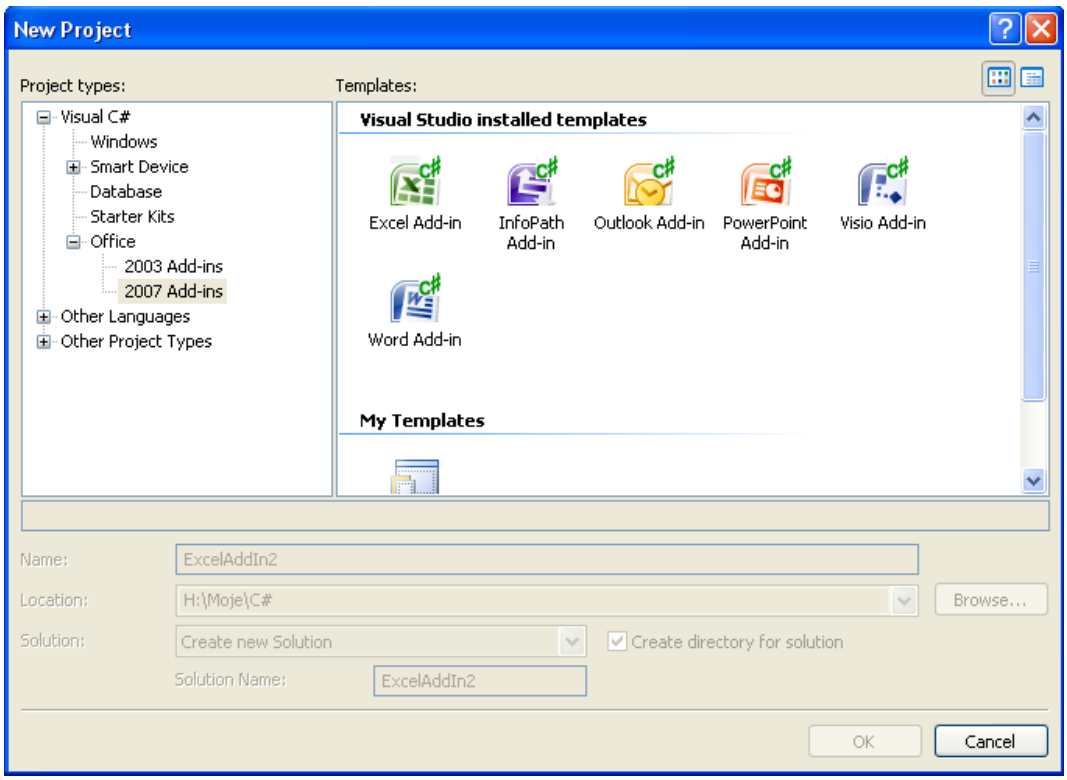
Příloha C - Add-in PowerPoint 2007 v C#

Jednoduchý příklad add-inu vyžaduje instalaci PIA do balíku MS Office (viz Obrázek C.1)

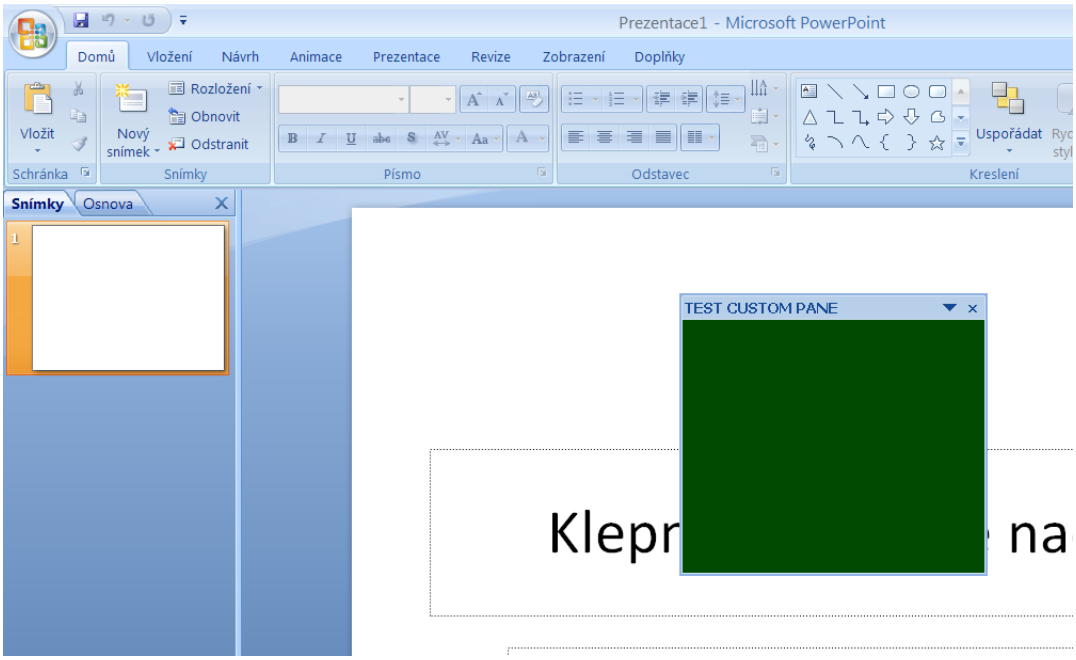


Obrázek C.1: Přidání podpory PIA v instalaci MS Office

V VS pak založíme nový projekt (viz Obrázek C.2) a použijeme zdrojový kód uvedený níže. Náš ovládací prvek se pak po zkompilování automaticky přidá do MS Office, případně pouze do konkrétní aplikace. Add-in zde zůstává, dokud ho sami v nastavení neodebereme.



Obrázek C.2: Vytvoření nového projektu add-inu pro MS Office



Obrázek C3: Takto vypadá výsledný projekt

Do souboru ThisAddIn.cs doplníme do metody ThisAddIn_Startup inicializaci našeho nového ovládacího prvku

```
using System;
using System.Windows.Forms;
using Microsoft.VisualStudio.Tools.Applications.Runtime;
using PowerPoint = Microsoft.Office.Interop.PowerPoint;
using Office = Microsoft.Office.Core;

namespace  PowerPointAddIn
{
    public partial class ThisAddIn
    {
        private void ThisAddIn_Startup(object sender,
    System.EventArgs e)
        {
            this.CustomTaskPanes.Add(new MyControl(),"TEST CUSTOM
    PANE").Visible = true;
            // it is important to set our pane visible
        }

        private void ThisAddIn_Shutdown(object sender,
    System.EventArgs e)
        {
        }

        //region VSTO generated code
    }
}
```

Dále pak vytvoříme kontrolku. V našem případě bylo pro projekt GREG důležité, aby na ni bylo možné zobrazovat scénu pomocí OpenGL. V tomto příkladu

však budeme pomocí OpenGL pouze měnit barvy. Vytvoříme tedy nový objekt MyControl, což není nic jiného, než UserControl jejíž událost MouseMove spouští vykreslování pomocí OpenGL. (Knihovna pro práci s OpenGL není součástí VS ke stažení na http://www-user.rhrk.uni-kl.de/~larsmidd/csopengl/)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using OpenGL;

namespace PowerPointAddIn
{
    public partial class MyControl : UserControl
    {
        private Context context;

        public MyControl()
        {
                // add event MouseMove
            this.MouseMove += new
    System.Windows.Forms.MouseEventHandler(this.control_MouseMove);
                // OpenGL context set to this control
            context = new Context(this, 32, 32, 0);
            InitializeComponent();
        }
```

```
        private void control_MouseMove(object sender, MouseEventArgs e)
            {
                    //random numbers are used to set color
                Random r = new Random();
                gl.ClearColor(0, (float)r.NextDouble(),
    (float)r.NextDouble(), 0);
                    // and OpenGL uses it to draw on our control
                gl.Clear(gl.COLOR_BUFFER_BIT);
                context.SwapBuffers();
            }
        }
    }
```