MOŽNOSTI JAZYKA C# A JEHO INTEGRACE S JINÝMI JAZYKY.

Vladimír Sklenář[a]
Václav Snášel[b]

a) KMI PřF UP, Tomkova 40, 779 00 Olomouc - Hejčín, ČR, E-mail: vladimir.sklenar@upol.cz
b) FEI VŠB-TU, 17.listopadu 15, 708 33 Ostrava - Poruba, ČR, E-mail: vaclav.snasel@vsb.cz

Abstrakt

C# is a simple, modern, object oriented, and type-safe programming language derived from C and C++. It aims to combine the high productivity of Visual Basic and the raw power of C++. C# has been specifically designed by Microsoft to be the language of choice for writting applications for their new .NET platform. .NET supports multiple languages and allows integration among those languages.

1. Úvod

Programovací jazyk C# byl ohlášen v červnu roku 2000 jako nativní programovací jazyk .NET platformy firmy Microsoft. Důvodem pro jeho vytvoření byla zřejmě snaha o zjednodušení a zrychlení vývoje komponent, které byly doposud vytvářeny v C++. Přes několik pokusů o usnadnění totiž stále není snadné používat technologii COM, která je na platformě Microsoft pro vytváření komponent v současné době určena. Ještě složitější je práce s DCOM, který podporuje distribuované prostředí. Vývoj COM komponent tedy v současnosti klade vysoké nároky na znalosti a dovednosti programátorů. Cílem .NET platformy je vytvořit prostředí pro snadnou tvorbu internetových aplikací.

2. Přehled vlastností jazyka

Autor jazyka C# Anders Hejlsberg jej charakterizuje jako jednoduchý, moderní, typově bezpečný programovací jazyk, který je odvozen z jazyků C a C++. Obsahuje však také vlastnosti převzaté z jiných jazyků (Java, Visual Basic). Oproti C++ je C# výrazně jednodušší. Neobsahuje totiž některé z vlastností C++, jejichž nekorektní použití mohlo vést k zanesení chyb do vytvářených programů. Je to silně typový jazyk používající unifikovaný typový systém.
Hlavní cíl při návrhu jazyka byla jeho jednoduchost a bezpečnost. Za cenu jistého snížení výrazových možností jazyka je kód stabilnější. Oproti C++ nabízí C# programátorům následující výhody :

1. Jednoduchost

- Pro přístup k prvkům objektů a tříd se jednotně používá pouze tečkové syntaxe (např. objekt.metoda()), nejsou tedy zahrnuty symboly -> a ::
- Soustava předdefinovaných typů je podstatně jednodušší a logičtější. Např. znak je reprezentovaný jediným typem char (neexistují unsigned char, wchar_t, …) a je zahrnut typ string.
- Je zahrnut typ bool a není zaměnitelný s typem integer. Tím se potlačí chyby vzniklé záměnou = a ==.

- V příkazu switch se implicitně vkládá break před každý příkaz case. Tím se zabrání chybnému provedení více větví.

2. Konzistence

- Každý typ je potomkem třídy Objekt. Lze např. zapsat 3.ToString(). Objekty jsou sdruženy do prostorů jmen, které jsou do programu vkládány pomocí příkazu using (např. using system). Odpadají hlavičkové soubory a příkazy #include. Všechno musí být součástí nějaké třídy. Neexistují globální proměnné.

3. Robustnost

- Garbage collection zajišťuje korektní správu paměti.
- Podpora pro ošetření chybových stavů vyvoláním vyjímek. (příkazy throw, try…catch a try…finally).

4. Objektová orientovanost

- Pouze jednoduchá implementační dědičnost. Je podporována vícenásobná dědičnost rozhraní.
- Je explicitně vyžadován modifikátor virtual. Je tedy těžší přepsat omylem metodu.
- COM+ komponentový model je podporován konstrukcí delegate – objektově orientovaného ekvivalentu ukazatele na funkci v C++.

5. Typová bezpečnost

- Všechny dynamicky alokované objekty a pole jsou inicializovány na 0.
- Nelze přepsat nealokovanou paměť.
- Překladač upozorní na použití lokální proměnné před její inicializací
- Přístup k polím je automaticky testován na rozsah
- Všechny přetypování musí být bezpečná (nelze přetypovat integer na referenci)
- Garbage collection – nezůstanou nepoužívané reference
- Kontrola přetečení
- Pole jsou chápány jako objekty, ne jako adresovatelný proud bajtů

6. Škálovatelnost

- Kombinuje deklaraci a definici typů
- Přímo importuje COM+ metadata

7. Kompatibilita

- Trasparentní přístup ke COM a OLE automation

8. Flexibilita

- Lze přistupovat k nativnímu kódu – unsafe code (pointry, …).

C# nabízí tyto primitivní typy:

C# primitivní typ	CLR typ	Popis	Příklad použití
Sbyte	System.SByte	8 bitů se znaménkem	sbyte val = 12;
Byte	System.Byte	8 bitů bez znaménka	byte val1 = 12; byte val2 = 34U;
Short	System.Int16	16-bitů se znaménkem	short val = 12;
Ushort	System.UInt16	16- bitů bez znaménka	ushort val1 = 12; ushort val2 = 34U;
Int	System.Int32	32-bitů se znaménkem	int val = 12;
Uint	System.UInt32	32- bitů bez znaménka	uint val1 = 12; uint val2 = 34U;
Long	System.Int64	64- bitů se znaménkem	long val1 = 12; long val2 = 34L;

Ulong	System.UInt64	64- bitů bez znaménka	ulong val1 = 12;
			ulong val2 = 34U;
			ulong val3 = 56L;
			ulong val4 = 78UL

Char	System.Char	Znak, 16-bitů Unikód	char val = 'h';
Float	System.Single	IEEE 32-bitů float	float val = 1.23F;
Double	System.Double	IEEE 64-bitů float	double val1 = 1.23; double val2 = 4.56D;
Bool	System.Boolean	True/false	bool val1 = true; bool val2 = false;
Decimal	System.Decimal	Desítkové číslo s 28 významnými ciframi	Decimal val = 1.23M;
Object	System.Object	Základ pro všechny typy	Object o = null; Object o = new C1();
String	System.String	Typ String; řetězec je posloupnost znaků v Unikódu	String s = "Hello";

3. Porovnání s jinými jazyky

C++, Java a C# jsou definovány jako objektově orientované programovací jazyky. Tyto jazyky jsou si velmi podobné, protože používají stejné konvence pro deklaraci proměnných, pro zápis poznámek a stejné omezovače bloků {}.
Pro představu uvedeme příklady programů v těchto jazycích. Příklad je ukázkou konzolového výstupu nejprve v C# a potom v C++ a jazyku Java. Od RPG přes Cobol, Fortran, Basic, C, C++, Visual Basic, Javu, a nyní C#, se museli programátoři znovu učit nejjednodušší úlohu, a to vstup a výstup znaku. Bohužel tato tradice nebyla s příchodem C# porušena.

```
// toto je poznámka v C#
using System;
class HelloWorld
{
  static void Main()
  {
    for(int i = 1; i <= 100; i++)
      Console.WriteLine("Hello World, repeated another {0} times. ", i);
```

```
  }
}
```

```
#include <iostream.h>
// toto je poznámka v C++
int main()
{
  for(int i = 1; i <= 100; i++)
     cout << "Hello World, repeated another " << i << " times. " << '\n';
}
```

```
// toto je poznámka v Java
class HelloWorld
{
  public static void main(String[] args)
  {
    for(int i = 1; i <= 100; i++)
      System.out.println("Hello World repeated another " + i + " times.");
  }
}
```

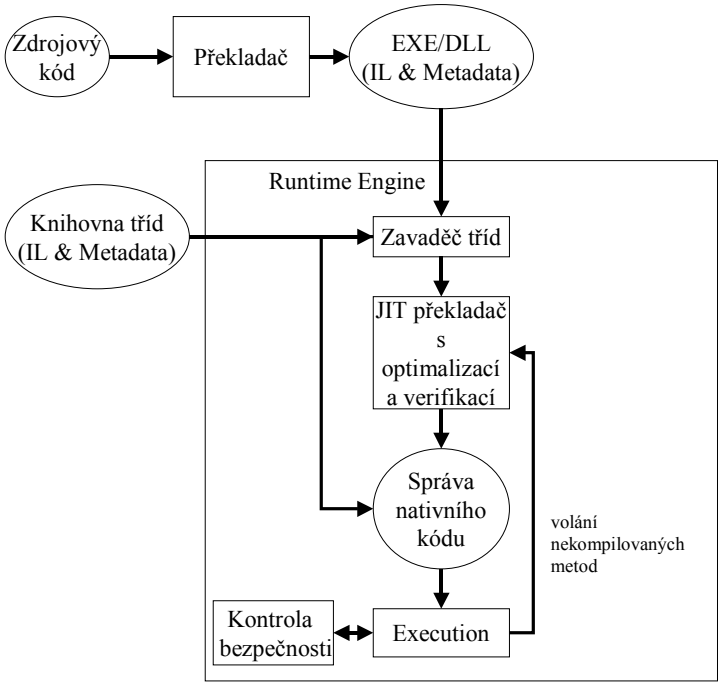
Z uvedených příkladů je vidět, že jedním ze základních rozdílů mezi C++ , Javou a C# je to že program v Javě a C# se nedá napsat bez deklarace alespoň jedné třídy. V následující tabulce uvedeme některé vybrané vlastnosti jazyků C++, Java, C# . Pro úplnější pohled jsme přidali i jazyk Visual Basic 7.0.

Vlastnost jazyka	C++	C#	Java	VB7
Jednoduchá dědičnost všechny objekty jsou potomkem základní třídy Object	N	A	A	A
Překlad do intermediate code	N	A	A	A
Unikód pro char, string, identifikátory	N	A	A	A
Všechny primitivní typy jsou potomkem Object	N	A	A	A
Struct, enum	A	A	N	A
Union	A	N	N	N
Ošetření vyjímek throw, try-finally-catch	A, ne finally	A	A	A
Přetížení operátorů	A	A, ne pro =	N	N
Přímé použití XML	N	A	N	A
Přímé použití XML pro rozhraní a RPC	N	A	N	A
Garbage collection	N	A	A	A
Preprocesor #define a pod.	A	A	N	A
Templates, generické programování	A	N	N	N
Optimalizece překladu, inline, register	A	N	N	N

Z tabulky je zřejmé, že podobnou funkci, kterou má splnit C# by mohl plnit i Visual Basic. Jak dále uvidíme, rozdíly mezi jednotlivými jazyky budou v rámci .NET platformy daleko menší než v současnosti. Jeden z možných důvodů pro zavedení nového programovacího jazyka může být nechuť programátorů v C++ k práci ve Visual Basicu. C# bude pro ně zřejmě průchozí varianta.

4. Integrace programovacích jazyků

Mezi nejvýznamnější přínosy .Net platformy pro tvůrce software patří její podpora pro vzájemnou integraci různých programovacích jazyků. Bude například možné vytvořit novou třídu (např. v C#), která je potomkem třídy implementované v jiném programovacím jazyku (např. Visual Basicu). Překladače spolupracující s .NET platformou vytvoří soubory, které obsahují spolu s přeloženým kódem v mezijazyce MSIL (Microsoft Intermediate Language) také metadata obsahující informace o typech a objektech deklarovaných v daném modulu. Jakmile takovýto soubor existuje může být importován do modulů zapsaných v libovolném jiném jazyce podporovaným v rámci .NET platformy. Proces importování typů může být prováděn opakovaně mezi různými jazyky. V době vykonávání modul nazývaný CLR (Common Languge Runtime) zajistí přeložení a vykonání MSIL kódu (viz obrázek).



Aby to bylo možné musí překladače splňovat pravidla stanovená specifikací CLS (Common Language Specification). Ta stanovuje tři úrovně:

- Framework – garantuje, že třídy zapsané v daném jazyce mohou být využívány jinými jazyky.
- Consumer – podporuje využívání tříd zapsaných v jiném jazyce.
- Extender – podporuje rozšiřování tříd. To znamená, že podporuje dědičnost a předefinování operací.

Microsoft poskytne v rámci nové verze Visual studia překladače pro C++, C#, Visual Basic a JScript. Jiné firmy, popřípadě univerzity, připravují překladače pro řadu dalších jazyků, např. Cobol, Smalltalk, Scheme, Mercury, Python, Perl, Eiffel.

Dále uvedem jednoduchý příklad na dědění a přepis virtuálních metod mezi C# a VB.NET. V jazyce C# implementujeme třídu Complex pro práci s komplexními čísly. Tato třída implementuje metodu ToString (, která je obsažena v univerzálním předku všech tříd - třídě Object) tak, že vypíše obecně používaný zápis komplexního čísla. Ve VB.NET vytvoříme

třídu GoniometricComplex, která modifikujeme třídu Complex tak, že metoda ToString bude vypisovat goniometrický tvar komplexního čísla.

Třída Complex je potomkem System.ComponentModel.Component. To zajistí, že získá vlastnosti potřebné k tomu, aby mohla být využívána jako komponenta ve všech jazycích v rámci .NET platformy, tedy i ve VB.NET.

```
namespace TvorbaSW
{
   using System;

public class Complex : System.ComponentModel.Component
{
          private void InitializeComponent ()
          {
          }

  protected double x, y;

  public Complex()
  {
  }
  public Complex(double X, double Y)
  {
    x=X; y=Y;
  }

  public override string ToString()
  {
    return(x.ToString() + " + " + y.ToString() + " i");
  }

  public double Norm
  {
    get
    {
      return Math.Sqrt(x*x + y*y);
    }
  }
}
}
```

Program ve VB.NET, který využívá třídu Complex a definuje vlastního potomka GoniometricComplex musí použít prostor jmen TvorbaSW, ve kterém je třída Complex realizována. Příkazem Inherits Complex zdělujeme, že budeme dědit ze třídy Complex. Definujeme nový konstruktor (metoda New) a přepíšeme metodu ToString. Klíčové slovo MyBase obsahuje odkaz na bázovou třídu a příkaz MyBase.New(X, Y) zavolá konstruktor Complex(double X, double Y) třídy Complex. Metoda ToString přepisuje metodu implementovanou v jazyce C# tak aby výpis komplexního čísla byl realizován jako výpis komplexního čísla v goniometrickém tvaru.

```
Imports System.Math
Imports System.ComponentModel
Imports TvorbaSW

Namespace TvorbaSW

   Public Class GoniometricComplex
      Inherits Complex

      Public Sub New(ByVal X As Double, ByVal Y As Double)
         MyBase.New(X, Y)
      End Sub

      Public Overrides Function ToString() As String
         Dim r As Double
         Dim mSin As Double
         Dim mCos As Double
         r = Norm
         mSin = x / r
         mCos = y / r
         If (r = 0) Then
            ToString = 0.ToString
         Else
            ToString = r.ToString  &  ".("  &  mSin.ToString  &  " + i " & mCos.ToString & ")"
         End If

      End Function

   End Class

End Namespace
```

Z příkladu je vidět, že používání tříd mezi C# a VB.NET je velmi jednoduché a přirozené. Implementace třídy Complex v C++ by do značné míry odpovídala implementaci v C# a lišila by se pouze syntaktickým zápisem. Na závěr je třeba poznamenat, že obdobné vlastnosti mají všech programovací jazyky implementované v rámci platformy .NET.
Vzhledem k historickým vazbám semináře Tvorba Softwaru na programovací jazyk COBOL, a faktu, že jsme jistou část své programátorské kariéry v COBOLu programovali, uvedeme ještě příklad kódu v tomto jazyce. Současně také chceme ukázat, že .NET platforma je otevřená i pro produkty jiných firem.

```
      CLASS-ID. HELLO INHERITS FORM.
      ENVIRONMENT DIVISION.
      CONFIGURATION SECTION.
      REPOSITORY.
         PROPERTY WIN-TEXT AS "Text"
         CLASS APPLICATION AS "System.WinForms.Application"
         CLASS FORM AS "System.WinForms.Form".
      FACTORY.
```

```
PROCEDURE DIVISION.
METHOD-ID. MAIN.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 APP-OBJ USAGE OBJECT REFERENCE FORM.
PROCEDURE DIVISION.
   INVOKE HELLO "NEW" RETURNING APP-OBJ.
   MOVE "Hello COBOL World!" TO WIN-TEXT OF APP-OBJ.
   INVOKE APPLICATION "Run" USING BY VALUE APP-OBJ.
END METHOD MAIN.
END FACTORY.
END CLASS HELLO.
```

V sekci REPOSITORY jsou definovány třídy a metody, které budou dále použity. V sekci PROCEDURE DIVISION je realizován vlastní program, který zobrazí klasický pozdrav. Implementace COBOLU v .NET platformě je od firmy Fujitsu Software (viz [6]).

5. Závěr

Lze předpokládat, že na platformě firmy Microsoft se jazyk C# stane poměrně rychle rozšířeným. Již v současné době je používán pro vývoj firemních aplikací. Vzhledem k tomu, že je vázán na existenci CLR je jeho další rozšíření otázkou implementace .NET na jiných platformách. Často citovaný termín „Java killer“ se z tohoto pohledu nejeví jako příliš aktuální. Je spíše možné, že k používání C# přejdou někteří zastánci Visual Basicu.

Literatura

1. Hejlsberg, Anders, Wiltamuth, Scott. C# Language Reference. Microsoft Corp., 2000
2. Richter, Jeffrey. NGWS Programming, 2000
3. Hrawey, B., Robinson, S., Templeman, J., Watson, K. C# Programming. Wrox Press 2000
4. Conrad, J., Dengler, P., Francis, B., Glynn, J., Harvey, B., Hollis, B., Ramachandran, R., Schenken, J., Short, S., Ullam, C. Introducting .NET. Wrox Press 2000
5. Sklenář, Vladimír, SNÁŠEL, Václav. Porovnání C# s jinými OO jazyky. Objekty 2000
6. Kadhim, Basim. COBOL for the Microsoft .NET Framework. Fujitsu Software, 2000