

# PV178: Programming for .NET Framework

## Introduction to .NET and C#

Vojtěch Forejt, forejt@fi.muni.cz  
Martin Osovský, osovsky@ics.muni.cz

**Faculty of Informatics and Institute of Computer Science**  
Masaryk University

February 19, 2009

# Microsoft.Net Technology Suite

- **standards** (CLI) and their **implementations** (CLR)
- **programming languages** (C# among others)
- **development tools** (Microsoft Visual Studio, Microsoft .Net Framework SDK)
- **runtime** (Microsoft .Net Framework)
- **application toolkits** (ASP.Net, ADO.Net, . . . )

- *Common Language Infrastructure*
- international standard (ECMA #335, ISO/IEC 23271:2003), see [7]
- a standard base for creating execution and development environments
- interoperability – languages and libraries conforming to the standard should work together seamlessly

# CLI Components

## CLI itself defines

- the Common Type System (CTS)
- the Common Language Specification (CLS)
- metadata (description of the code units, such as visibility, security requirements, etc.)
- portable and platform-agnostic file format for managed code
- common Intermediate Language (CIL) instruction set
- basic requirements on the Virtual Execution System
- a programming framework (a class library)

# CTS: the Common Type System

- The Complete set of types available to a CLI-compliant language
- based both on representation of values and their behaviour
  - designed for language interoperability
  - designed via set of rules – types are extensible (by derivation), type system is not
  - designed for broad coverage: object-oriented, procedural and functional languages (C#, JScript, C++, F#, COBOL, J#, etc. – for more comprehensive list see [10]).

# The Common Language Specification

Defines a subset of CTS types that can be used for external calls.

- actually a set of restrictions on the CTS
- used by standardised framework to ensure compatibility

# CLI, CTS and CLS relationship

## CLI

Defines the Virtual Execution System and the executable code that runs in it. To produce executable code, the CLI defines types (CTS), metadata, the CIL instruction set, a file format and a factored base class library.

## CTS

Defines the complete set of types available to a CLI-compliant language.

## CLS

Defines the subset of CTS types that can be used for external calls.

# Metadata

- stored directly in the executable file in the form of *attributes* and *tables*
- attributes attached to code units (assemblies, methods, types, etc.)
- tables list selected code units (methods, types, etc.)
- used to
  - manage code execution
  - versioning, deployment and maintenance
  - enable cross language interoperability
- contains
  - description of an executable file (Manifest)
  - description of all types stored in an executable file
  - signatures of all methods stored in an executable file
  - custom user defined attributes – the metadata system is extensible
  - ...



# Executable File Format

- executable file: Assembly (unit of deployment)
- conforms to the Portable Executable file format (PE)
- CLI PE file format is an extension to the Microsoft PE<sup>1</sup> file format
- Contains
  - Assembly metadata,
  - Type metadata,
  - IL code,
  - resources
- Microsoft PE is described in [9]

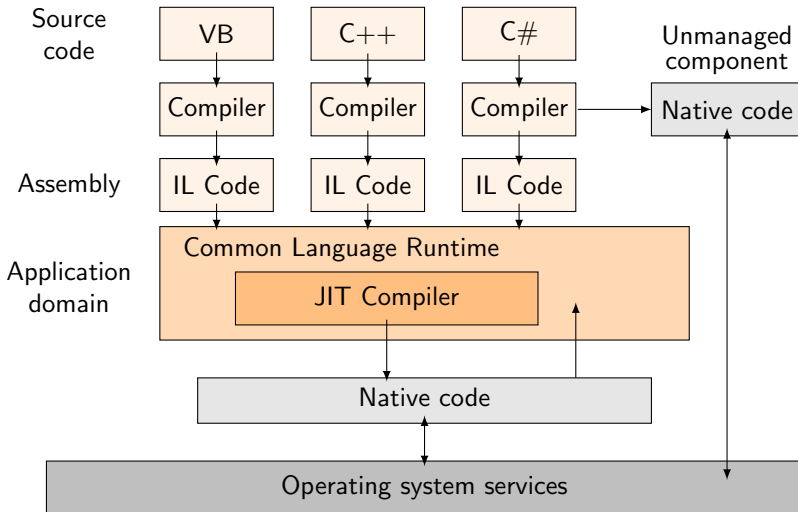
---

<sup>1</sup>Microsoft Portable Executable

# VES – Virtual Execution System

- provides an environment for executing managed code
- provides direct support for built-in data types
- defines a virtual machine that provides support for executing the CIL instruction set
- VES implementation is usually denoted as CLR (*Common Language Runtime*) – Microsoft.Net Framework, Mono [4], Rotor – Shared Source CLI

# CLR Execution Model



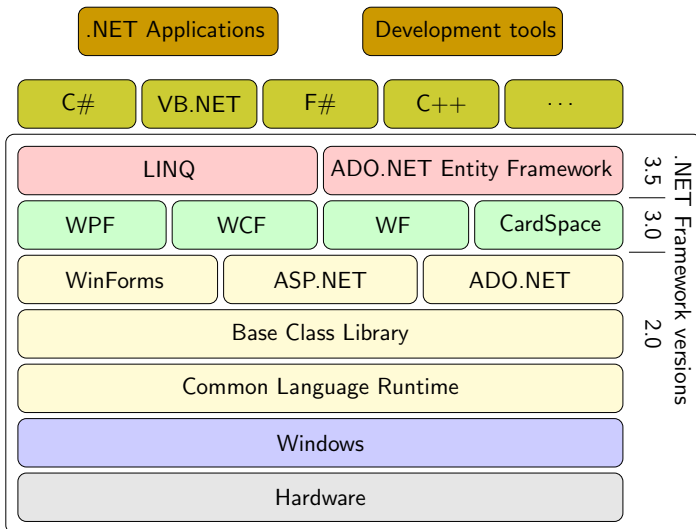
# Compilation and Execution

- 1 source code is compiled by the compiler into the intermediate code (CIL)
- 2 generated CIL code is stored (by the compiler) in an assembly along with metadata
- 3 the assembly is deployed on the target machine (assembly usually platform independent)
- 4 the assembly is executed by CLR
- 5 the code stored in the assembly is compiled on demand

# Microsoft.Net Framework

- ECMA #335 Compact Profile
- ASP.Net, ADO.Net, Windows Forms, C# 3.0, Silverlight. . .
- runs on Microsoft Windows x86, x64, IA64
- Performance of Microsoft.Net Framework and Java VMs (Sun, IBM, BEA) is generally comparable (let the flame begin :-))

# .NET Framework Structure



# Mono

- ECMA #335 Compact Profile
- ASP.Net, ADO.Net, Windows Forms, GTK#, C#, Moonlight. . .
- [http://www.mono-project.com/Supported\\_Platforms](http://www.mono-project.com/Supported_Platforms)
- Microsoft.Net Framework is faster (better GC and JIT)

What is C#

# Example code

CSharpFeatures.cs



# C# in buzzwords

- simple, easy to learn and use
- modular and block structured
- object oriented (class based language)
- component oriented
- strongly typed (static typing with implicit RTTI – *RunTime Type Information*), see [2], [3]
- descended from C, C++ ([8])
- part of the Microsoft.Net suite of technologies

# C/C++

- curly braces
- C# more typesafe than C++ (e.g. only safe coercions)
- tricky features (templates) reinvented
- C# has support for low-level pointer manipulation (`unsafe` keyword) as has C++ (Java has not)

# Java

- similar idea and motivation
- subtle syntactic differences
- C# includes lower level constructs (strictly optional)
- C# includes higher level constructs (convenience)
- Java is truly multi-platform – Microsoft has (almost) no interest in supporting non-Microsoft platforms
- **when C# came, Java was already here and well established**



- research programming language ([1])
- far beyond C# (and everything you've probably met)
- XML made easy (LINQ and beyond)
- a notion of *choice types*
- parallel constructs
- very advanced

# Spec#

- experimental extension to C#, see [12]
- implements *Design by Contract* concept, pioneered by Eiffel (see [11])
- adds preconditions, postconditions, invariants to the language
- may eventually become mainstream
- for introductory material and samples refer to: [13]

# Spec# Example

```
public float SquareRoot(float f)
    requires f > 0 // PRECONDITION
{
    return Math.sqrt(f);
}
```

# C# Evolution

- C# language evolves
- four editions so far:
  - C# 1.0 (year 2002)
  - C# 2.0 (year 2005)
  - C# 3.0 (year 2007)
  - C# 4.0 (in development)

# C# 1.0

- first edition of C# available to public (sometimes denoted as C# 1.2)
- unveiled with Microsoft Visual Studio.Net 2002
- major features were there
- filled the language gap in Microsoft portfolio
- “killer” feature of Microsoft.Net suite



- interesting new features:
  - generic types (*see example*)
  - anonymous methods
  - partial classes
  - ...
- mostly syntactic sugar, but generics are important

- LINQ (*Language INtegrated Query* – query languages (SQL, XPath, XQuery) embedded (*see example*) directly in C# source code
- type inference (*see example*)
- lambda expressions (*see example*)
- bytecode compatible with C# 2.0 (interoperability)

# C# Language Specification

- ECMA: *European Computer Manufacturers' Association*, <http://www.ecma-international.org>
- ECMA #334 (see [6]) – freely downloadable
- current: 3<sup>rd</sup> edition (corresponds to C# 2.0)
- 2<sup>nd</sup> (C# 1.0) edition submitted and approved by ISO (ISO/IEC 23270)
- useful reference, but not lightweight reading

# Development Tools

Tools which are available to students at FI (Thanks go to the CVT FI):

- Microsoft Visual Studio 2008 (B117)
- Mono (computer hall – Linux and Windows), see: [4]
- Rotor: Shared Source CLI (A104, computer hall), see: [5]

Additional tools:

- MonoDevelop (open source IDE for Mono)

# MSDN Academic Alliance

- Microsoft operating systems, development tools and server products for research and education
- for more information refer to:  
<http://www.fi.muni.cz/tech/win/msdnaa.shtml>

## HelloWorld.cs

```
using System; //like import of package in Java

namespace PV178 /* like package in Java */
{
    public class HelloWorld
    {
        static void Main(string [] agrs)
        {
            Console.WriteLine(" Hello _world" );
        }
    }
}
```

# HelloWorld.cs cont.

- compile with `csc HelloWorld.cs`
- then run `HelloWorld.exe`

## References

- [1] Microsoft Research, *Cω Website*.  
<http://research.microsoft.com/Comega/>.
- [2] Wikipedia Article, *Data types*.  
<http://en.wikipedia.org/wiki/Datatype>.
- [3] ———, *RunTime Type Information*.  
<http://en.wikipedia.org/wiki/RTTI>.
- [4] *Mono Project Website*. <http://www.mono-project.org>.
- [5] *Rotor: Shared Source CLI*. <http://msdn.microsoft.com/net/sscli>.
- [6] ECMA International, *C# Language Specification*. <http://www.ecma-international.org/publications/standards/Ecma-334.htm>.
- [7] ECMA International, *Common Language Infrastructure*. <http://www.ecma-international.org/publications/standards/Ecma-335.htm>.
- [8] Lambda the Ultimate, *Genealogical Diagrams*.  
<http://lambda-the-ultimate.org/node/7>.
- [9] *Microsoft Portable Executable and Common Object File Format Specification*, 1999. <http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.msp>.
- [10] *.NET Languages*.  
<http://www.dotnetlanguages.net/DNL/Resources.aspx>.
- [11] *Building bug-free O-O software: An introduction to Design by Contract*.  
<http://archive.eiffel.com/doc/manuals/technology/contract/>.
- [12] *Spec# Programming System*.  
<http://research.microsoft.com/specsharp/>.
- [13] *The Spec# Wiki at Channel9*. <http://channel9.msdn.com/wiki/default.aspx/SpecSharp.HomePage>.