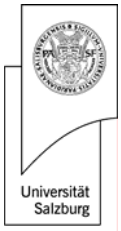
Software Produktentwicklung



Wintersemester 2004 / 2005

Guido Menkhaus and Sebastian Fischmeister
University of Salzburg

SOFTWARE RESEARCH LAB

Referenzen

- *Msdn Roadshow .NET*
- *.NET Overview. Developmentor*
- Damien Watkins, *Handling Language Interoperability with the Microsoft .NET Framework*, Monash University, 2000
- Erik Meijer (Microsoft Redmond WA) and John Gough (QUT Brisbane, Australia), *Technical Overview of the Common Language Runtime*

UNIVERSITÄT SALZBURG

Interoperability

- Representation standards
  - External Data Representation (XDR) and Network Data Representation (NDR), address the issue of passing data types between different machines. These standards compensate for such matters as big-endian and little-endian issues, and different word sizes.
- Architecture standards
  - Distributed Computing Environment's (DCE) Remote Procedure Call (RPC), the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA), and Microsoft's Component Object Model (COM), address the issue of calling methods across language, process, and machine boundaries.
- Language standards
  - ANSI C allows the distribution of source code across compilers and machines.
- Execution environments
  - Virtual machines (VM) of SmallTalk and Java, allow code to execute on different physical machines by providing a standardized environment for execution.
- **Language Interoperability**
  - Allows classes and objects in one language to be used as first class citizens in another language. Ideally, it should be possible for Eiffel code to instantiate a C++ object that inherits from an C# class → **Intermediate language**

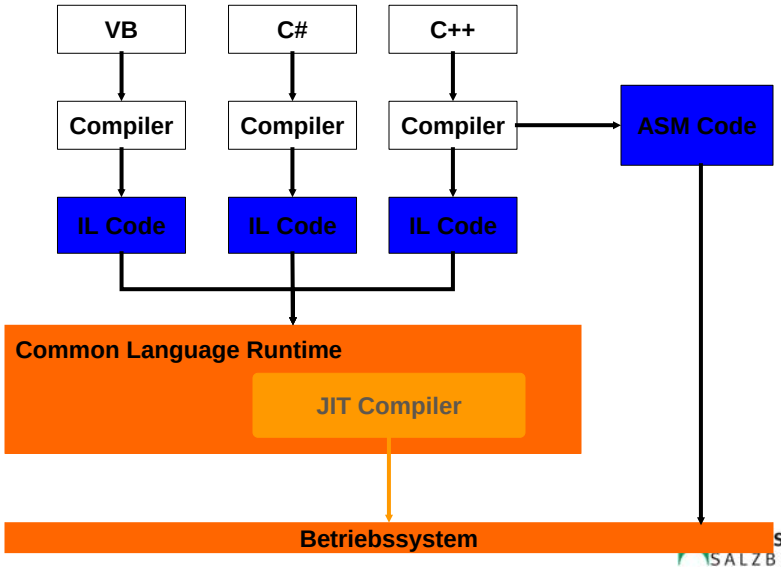
UNIVERSITÄT SALZBURG

Benefits

- **Portability**
  - By using an intermediate language, you need only n + m translators instead of n * m translators, to implement n languages on m platforms.
- **Compactness**
  - Intermediate code is often much more compact than the original source. This was an important property back in the days when memory was a limited resource, and has recently regained importance in the context of dynamically downloaded code.
- **Efficiency**
  - By delaying the commitment to a specific native platform as much as possible, the execution platform can adapt to the dynamic behavior of the program.
- **Security**
  - High-level intermediate code is more amenable to deployment and runtime enforcement of security and typing constraints than low level binaries.
- **Interoperability**
  - By sharing a common type system and high-level execution environment (that provides services such as a common garbage collected heap, threading, security, etc), interoperability between different languages becomes easier than binary interoperability. Easy interoperability is a prerequisite for multi-language library design and software component reuse.
- **Flexibility**
  - Combining high level intermediate code with metadata enables the construction of (typesafe) metaprogramming concepts such as reflection, dynamic code generation, serialization, type browsing etc.

UNIVERSITÄT SALZBURG

Übersicht



5

2004, S. Fischmeister, G. Menkhaus

SITÄT SALZBURG

Basics

Managed Code

- Sämtlicher Code wird unter Aufsicht der Common Language Runtime ausgeführt
  - Runtime führt Sicherheitsüberprüfungen aus
  - Runtime übernimmt Speicherverwaltung und Fehlerbehandlung (→ GC, Exceptions)
  - Runtime führt Versionsprüfungen aus
  - Dieser Code wird als Managed Code bezeichnet

6

2004, S. Fischmeister, G. Menkhaus

UNIVERSITÄT SALZBURG

Basics
Microsoft Intermediate Language

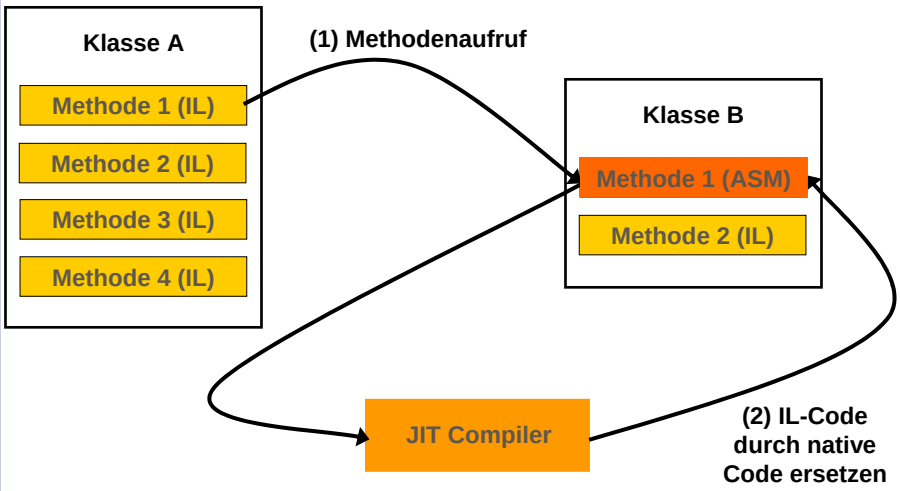
- Compiler erzeugen keinen native Code sondern eine prozessorunabhängige Zwischensprache
  - Sprachintegration erfolgt auf Codeebene
  - MSIL – Microsoft Intermediate Language
  - MSIL wird oft auch mit IL bezeichnet

Basics
Code wird kompiliert

- IL-Code wird vor der Ausführung immer (!) durch Compiler in echten Maschinencode übersetzt
  - Unabhängigkeit von Hardwareplattformen
  - unter Windows CE bereits mit einem IL-Vorläufer im Einsatz

Basics

Code wird kompiliert



UNIVERSITÄT SALZBURG

Basics

Implikationen

- Die IL unterscheidet sich von „reinen“ Assemblersprachen
  - komplexe Datentypen und Objekte sind fester Bestandteil
  - Konzepte wie Vererbung und Polymorphie werden von vornherein unterstützt

UNIVERSITÄT SALZBURG

Basics
Implikationen

- Sprachen werden gleichwertig, da alle Compiler MSIL-Code erzeugen
  - „eine C# Klasse kann von einer VB.NET Klasse abgeleitet sein“
  - einheitliche Fehlerbehandlung
- Compilerbau wird einfacher
  - kein eigenes Typsystem
  - keine eigene Standardbibliothek
  - Sprachen sind „per Definition“ interoperabel

Basics
Common Type System

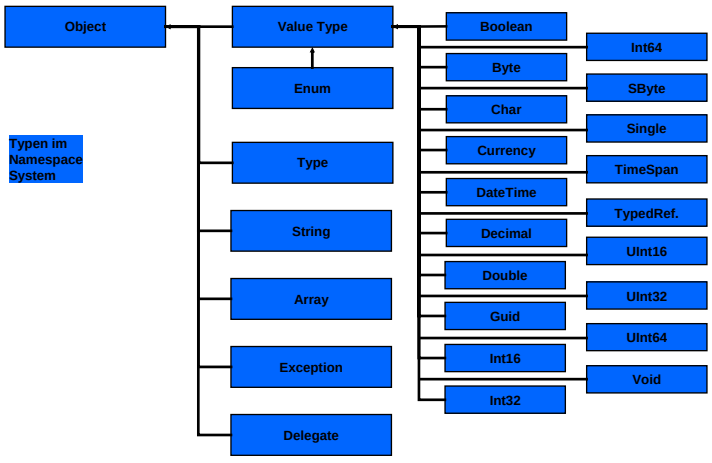
- Das Typsystem
  - Typen werden eindeutig
    - „ein String unter C# und ein String unter VB.NET sind identisch“
  - Sprachen werden interoperabel, da sie das gleiche Typsystem benutzen
  - CTS – Common Type System

Wrap up.

- Common Language Specification (CLS) defines type subset
  - required to be supported by all .NET languages
  - limiting code to CLS maximizes language interoperability
  - code limited to CLS called *CLS compliant*
- Compilers produce *Intermediate Language* (IL)
  - IL is not executable
  - similar to assembly language
  - processor independent
- *Common Language Runtime* (CLR) is the execution engine
  - loads IL
  - compiles IL
  - executes resulting machine code
- IL is compiled into machine code at runtime by the CLR
  - compiles methods as needed
  - called *just in time* (JIT) compile
- JIT compilation model:
  - first time method is called the IL is compiled and optimized
  - compiled machine code is cached in transient memory
  - cached copy used for subsequent calls

Common Type System

Das Objektmodell



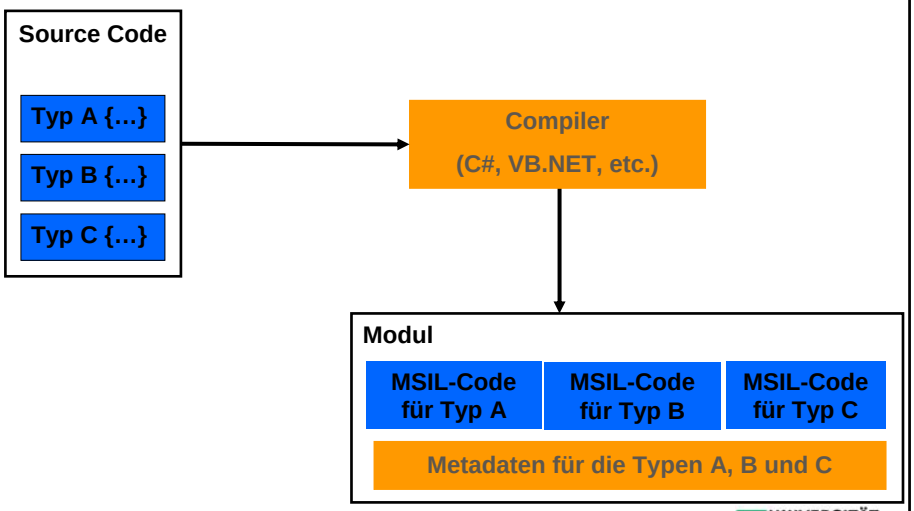
Common Type System

Gleichheit und Identität von Objekten

- Zwei Objekte sind gleich, wenn deren Inhalte gleich sind
- Zwei Objekte sind identisch, wenn sie die gleiche Instanz referenzieren
- Gleichheit definiert sich über die virtuelle Methode System.Object.Equals
  - identisch: System.Object.Equals = true
  - gleich: System.Object.Equals.Value = true

Metadaten und Reflection

Übersetzen von Sourcen



Metadaten und Reflection

- Ein Modul dient als Container für Typen
- Ein Modul enthält
  - den IL-Code der Typen
  - Beschreibung der Typen
- Die Beschreibung der Typen wird mit Metadaten bezeichnet
- Jedes Modul enthält Metadaten
  - Compiler erstellt Metadaten „on the fly“

Metadaten und Reflection

- Metadaten sind für alle Module auf die gleiche Art und Weise aufgebaut
  - Einheitliches Format !!!
- Metadaten eines Moduls können zur Laufzeit ausgelesen und geändert werden
  - Diesen Vorgang nennt man Reflection
  - .NET Framework stellt entsprechende Klassen über den Namespace System.Reflection bereit

IL instructions

- ldarg *n*
  - pushes the contents of the *n*-th argument on the evaluation stack.
- ldarga *n*
  - pushes the *address* (as a managed pointer of type *T*&) of the *n* argument on the evaluation stack.
- starg *n*
  - pops a value from the stack and stores it in the *n*-th argument.
- ldloc *n*
  - instruction pushes the contents of the *n*-th local variable onto the evaluation stack
- ldloca *n*
  - Pushes the *address* of the *n*-th local variable on the evaluation stack as a managed pointer.
- stloc *n*
  - instruction pops a value from the stack and stores it in the *n*-th argument.
- ldind.*t*
  - expects an address (which can be a native int, or a unmanaged or managed pointer) on the stack, dereferences that pointer and puts the value on the stack.
- stind.*t v*
  - stores a value *v* of type *T* at address found at the top of the stack.
- The JIT can figure out the types of these values from the context.

Example

- For example, here is the IL version of the Swap function that swaps the values of two variables:

```
.method static void Swap(int32& xa, int32& ya) {
.maxstack 2
.locals (int32 z)
ldarg xa; ldind.i4; stloc z
ldarg xa; ldarg ya; ldind.i4
stind.i4; ldarg ya; ldloc z
stind.i4; ret // return
}
```

- To call this function, we just pass the addresses of the local variables as arguments to function Swap:

```
.locals (int32 x, int32 y)
// initialize x and y
ldloca x
ldloca y
call void Swap(int32&, int32&)
```

Hello, World!

C#

```
using System;

namespace HelloWorld
{
    public class Class1
    {
        public static void Main()
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

VB.NET

```
Imports System

Namespace HelloWorld
    Class Class1
        Shared Sub Main()
            Console.WriteLine("Hello, World!")
        End Sub
    End Class
End Namespace
```


Hello, World: C# IL

```
.namespace HelloWorld
{
  .class public auto ansi Class1
         extends [mscorlib]System.Object
  {
    .method public hidebysig static void Main() il managed
    {
      .entrypoint
      .maxstack  8
      IL_0000:  ldstr      "Hello, World!"
      IL_0005:  call       void [mscorlib]System.Console::WriteLine(class System.String)
      IL_000a:  ret
    }
    .method public hidebysig specialname rtspecialname
           instance void .ctor() il managed
    {
      .maxstack  8
      IL_0000:  ldarg.0
      IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
      IL_0006:  ret
    }
  }
}
```