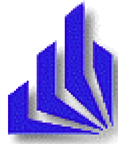

Universität Paderborn

Fachbereich Informatik

Vertiefung Softwaretechnik

Seminararbeit

Sprachumgebung im .NET
Übersetzung, Ausführung, gemeinsame Zwischensprache, gemeinsames Typsystem

vorgelegt von

Jens Heger, 6054796
jheger@upb.de

vorgelegt bei

Prof. Dr. Uwe Kastens

Paderborn, 16/17. September 2002

Inhaltsverzeichnis

1 Einleitung und Motivation 3
1.1 Abgrenzung.. 3
1.2 Microsofts Ziele 3

2 Das MS® .NET Framework.. 5
2.1 Die MS® .NET Plattform: Komponenten und Begriffe 5
2.1.1 Common Language Infrastructure (CLI) und Common Language Runtime (CLR) 6
2.1.2 Common Language Specification (CLS) und das Common Type System (CTS) 6
2.1.3 Verwaltete Module und die Intermediate Language (IL) 8
2.2 Ausführungsmodell 10
2.2.1 Vergleich: Java Ausführungsmodell.. 11
2.3 Klassenbibliothek des .NET Frameworks 12
2.3.1 Vergleich zur Java Klassenbibliothek.. 14
2.4 Assemblies 15
2.5 Vergleich Java: Classfiles 16
2.6 Fazit 17

3 Literaturverzeichnis.. 18

1 Einleitung und Motivation

1.1 Abgrenzung

Diese Arbeit beschäftigt sich mit der Sprachumgebung im Microsoft .NET Framework. Es wird gezeigt, wie die Übersetzung, die Ausführung, die gemeinsame Zwischensprache und das gemeinsames Typsystem zusammengehören und funktionieren. Es findet jeweils ein Vergleich zu bekannten Java Komponenten statt.

1.2 Microsofts Ziele

Nachdem Microsoft nun mehrere Jahre an dem MS® .NET Framework gearbeitet hat, ist es vor einiger Zeit vorgestellt worden. Es verbergen sich hinter diesem Konzept verschiedene Ideen und Ziele.

Microsoft hat sich in den letzten Jahren einen schlechten Ruf eingehandelt, was die Win-32-Api betrifft, da es hier durch gemeinsam genutzte Bibliotheken von verschiedenen Programmen, regelmäßig zu Problemen kommt. Jedes Programm bringt die Bibliotheken, die es benötigt mit und registriert diese zentral. So kann es sein, dass eine alte durch eine neue oder eine deutsche durch eine englische ersetzt wird, diese aber nicht kompatibel zueinander sind. Diese „DLL-HELL" wurde nun abgeschafft, bzw. wird es, wenn alle Windows-Programme auf dem neuen Framework basieren, denn da wird eine zentrale Registrierung nicht mehr benötigt, da die Bibliotheken mit Metadaten ausgestattet wurden.

Eine weitere Neuerung sind die so genannten **Web Services**. Sie sollen verschiedene Computer, aber auch PDAs, Handys usw. miteinander über das Internet/Intranet stärker als bisher miteinander verbinden. Bisher ist es sehr schwierig bestimmte Informationen, die man aus dem Internet abrufen kann, direkt in anderen Programmen zu nutzen oder weiterzuverarbeiten. Hier setzen die Web Services an, die anhand offener Schnittstellen Daten miteinander austauschen können. Microsoft hält sich dabei an offene Standards, wie http, XML oder SOAP, um

plattformübergreifend eine möglichst hohe Akzeptanz und Kompatibilität zu erreichen.

Als ein einfaches Beispiel kann man sich das folgende Szenario vorstellen: Ein Webseitenbetreiber eines Urlaubhotels möchte den Besuchern seiner Seite das aktuelle Wetter präsentieren. Dies ist mit herkömmlichen Mitteln im Augenblick nur schwer möglich. Es müsste regelmäßig ein manuelles Update durchgeführt werden oder automatische Systeme müssten regelmäßig angepasst werden, sobald sich der Wetter-Webserver leicht ändert. Dies ist wesentlich leichter möglich, wenn der Wetterdienst zusätzlich zu seinem Webserver einen Web-Service anbieten würde.

Eine andere Idee ist die weitere Zerteilung von Software in Komponenten, die mit dem neuen MS Office XP begonnen hat. Hiermit ist ein direkter Zugriff auf Web Services möglich und so können z.B. einerseits Daten direkt in Excel oder Word verarbeitet werden oder aber bestimmte Funktionen auf Application Servern ausgelagert werden, was eine wesentlich leichtere Administration ermöglicht, da nicht die Software auf jedem Client installiert werden müsste. Dies sind alles sicherlich keine neuen Ideen, allerdings werden sie jetzt, besonders was die Windows-Welt angeht, konsequent umgesetzt.

Microsoft möchte hierzu nicht nur die Technik anbieten, sondern mittelfristig auch als führender Dienstleister von Web Services auftreten. Unter dem Stichwort *.NET MY Services* soll ein großes Angebot geschaffen werden, was mit dem umstrittenen, zentralen Passwort Dienst *Passport.NET* begonnen hat.

2 Das MS® .NET Framework

In diesem Kapitel wird gezeigt, aus welchen Komponenten sich das MS® .NET Framework zusammensetzt und welche Aufgaben die einzelnen dabei erfüllen. An geeigneter Stelle finden Vergleiche zu Java statt.

2.1 Die MS® .NET Plattform: Komponenten und Begriffe

Die Abbildung 1 zeigt die wichtigsten Bestandteile: Basis ist ein Betriebssystem, auf das die abstrakte Maschine, die CLR, aufsetzt. Sie ist für die Ausführung verantwortlich. Die Klassenbibliothek, FCL, und die speziellen Daten und XML – Klassen bieten die Funktionalität, wie z.B. Festplattenzugriff (IO), oder aber auch Anschluss an Datenbanken (ADO.NET) usw.. Die oberste Reihe stellt die Ausgabemöglichkeiten des Frameworks dar. Hier finden sich sowohl die Web Services, wie auch die normalen Windowsprogramme (Windows Forms), Web Server (Web Forms), Dienste usw.. Die Richtlinien für die einzelnen Komponenten werden von der Common Language Infrastructure (CLI) definiert.

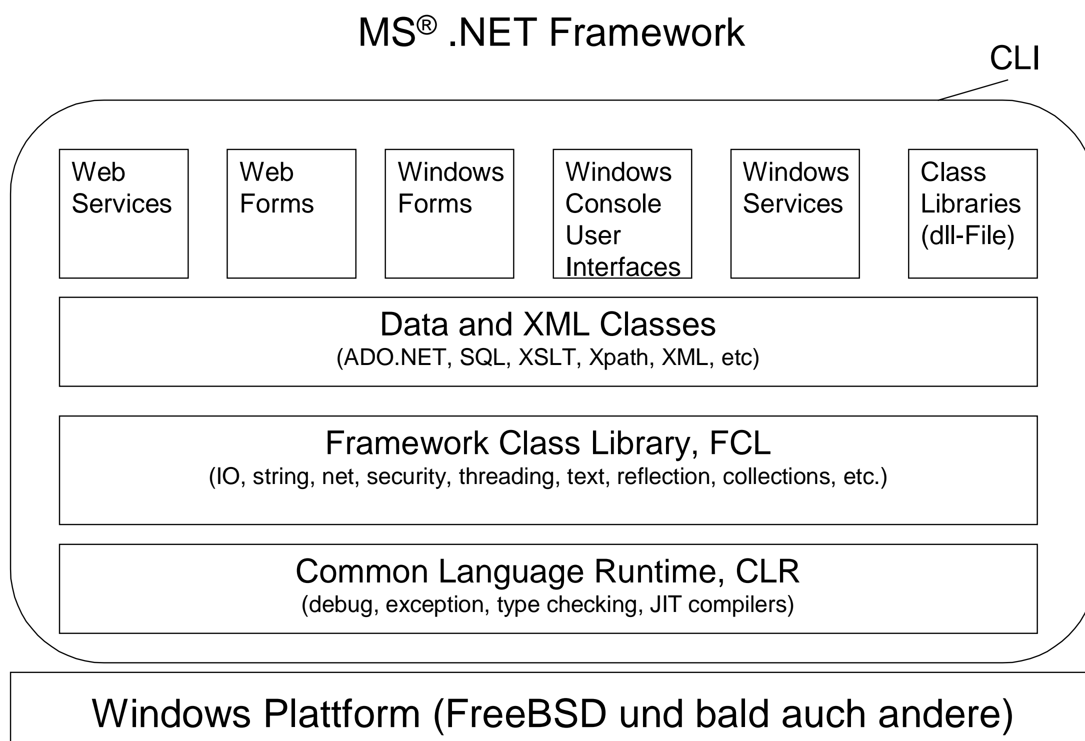

Abbildung 1: Vergl. [Thai & Lam01] Seite 10

2.1.1 Common Language Infrastructure (CLI) und Common Language Runtime (CLR)

Das MS® .NET Framework setzt auf ein beliebiges Betriebssystem auf. Implementierungen gibt es bereits für die aktuellen Windows Versionen und FreeBSD. Grundsätzlich ist es bei einem Ausführungssystem, das auf einer abstrakten Maschine basiert, praktisch immer möglich diese auf andere Plattformen zu portieren, deswegen kann man davon ausgehen, dass es in Zukunft auch noch andere Hersteller und andere Betriebssysteme geben wird, die das .NET Framework anbieten. Dies ist natürlich eine technische Sichtweise, es ist noch nicht ganz klar, ob es lizenzrechtliche Probleme geben wird. Bemerkungen hierzu finden sich in der ([CT4/02]). Unter Linux gibt es bereits zwei Ansätze. Einmal eine Art „Umbau“ der FreeBSD Version ([Bangay02]) und einmal die komplette Neuimplementierung. ([Mono]). Diese Implementierungen richten sich nach der **Common Language Infrastructure (CLI)**, die Microsoft, genau wie die Sprache **C#** bei der ECMA eingereicht hat. (ECMA-334 (C#), ECMA-335 (CLI)) Beide sind von dieser zu einem Industriestandard erklärt worden, was allerdings Lizenzgebühren und ähnliches nicht ausschliesst.

Die **Common Language Runtime (CLR)** ist die Implementierung der CLI von Microsoft selbst. Sie ist der Kern des .NET Frameworks und eine Laufzeitumgebung vergleichbar mit der Java Virtual Machine. Eine Besonderheit der CLR ist, dass sie mit verschieden Sprachen zusammenarbeitet. Dies ist möglich, weil sich alle .NET Sprachen an die **Common Language Specification (CLS)** und das **Common Type System (CTS)** halten.

2.1.2 Common Language Specification (CLS) und das Common Type System (CTS)

Die CLS ist eine Spezifikation, die für Compilerbauer die Mindestanforderungen festlegt, welche ihre Sprache erfüllen muss, wenn sie CLR-kompatibel sein soll. Grundsätzlich ist es möglich, für praktisch jede Sprache einen .NET Compiler zu bauen, solange er sich an die CLS hält. Da sich die Sprachen teilweise erheblich unterscheiden, wird so ein gemeinsamer Nenner festgelegt. Unterschiede sind z.B. die Groß- und Kleinschreibung, vorzeichenlose Integer, Operatorüberladung usw. Es

ist dabei so, dass die CLR einen gewissen Funktionsumfang anbietet, der von den einzelnen Sprachen zum Teil umgesetzt und angeboten wird. Pflicht ist allerdings nur der CLS-Teil. Da C# passend zum .NET Framework entwickelt wurde, bietet es (fast) alle Funktionen der CLR an. Also müsste in der Abbildung 2 der C#-Kreis, der den Funktionsumfang darstellen soll, richtigerweise fast deckungsgleich mit dem äußeren sein!

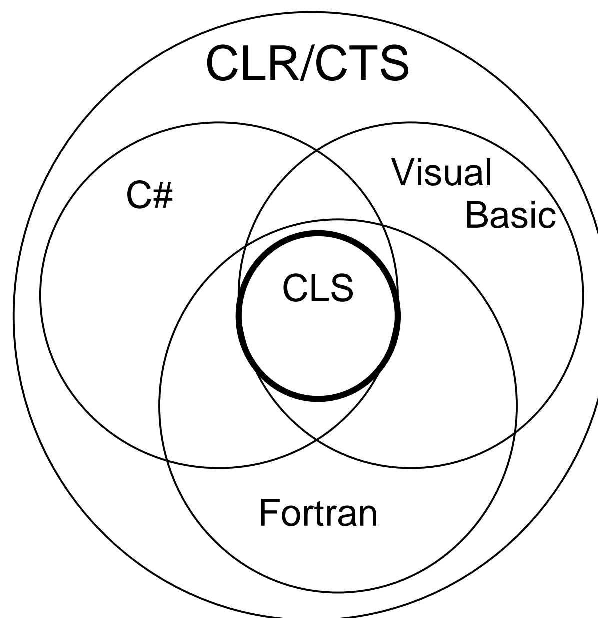

Abbildung2: Programmiersprachen bieten eine Teilmenge von CLR/CTS und eine Obermenge des CLS (aber nicht unbedingt dieselbe Obermenge) vgl. [Richter02] Seite 23

Es gibt ein gemeinsames Typsystem für alle Sprachen, das es ermöglicht, dass Programme, die in unterschiedlichen Sprachen programmiert wurden, gegenseitig Funktionen aufrufen können. Mit diesen so genannten Typen sind Klassen gemeint, die das grundlegende Konzept der objektorientierten CLS darstellen. Auch benutzerdefinierte Typen können über Sprachgrenzen hinweg benutzt werden. Hier nun eine Übersicht, was eine Typdefinition beinhaltet:

- alle definierten Attribute
- die Typsichtbarkeit
- den Namen des Typs

- den Basistyp
- alle Interfaces, die der Typ implementiert
- alle Definitionen der Member des Types. Ein Typ kann keinen oder mehrere Member beinhalten. Hier eine kurze Übersicht über Member:
    - Feld (field): Datenvariable, die Teil des Objektzustandes ist.
    - Methode (method): Eine Funktion, die den Objektzustand ändern kann.
    - Eigenschaft (property): Eine Eigenschaft sieht für den Aufrufer aus wie ein Feld, es ist für den Entwickler allerdings eine Methode. Auf diese Art und Weise kann der Entwickler die übergebenen Argumente und den Objektzustand überprüfen, bevor er auf den Wert zugreift. Es ist so auch möglich Felder zu implementieren, die nur gelesen oder nur eingestellt werden können.
    - Ereignis (event): Mit Hilfe eines Ereignisses ist er möglich ein anderes Objekt zu benachrichtigen.

Die CTS – Regeln besagen z.B. außerdem, dass es nur eine einfache Vererbung gibt, oder dass jeder Typ von `System.Object` abgeleitet werden muss.

2.1.3 Verwaltete Module und die Intermediate Language (IL)

Damit die CLR mit Programmen aus verschiedenen Sprachen arbeiten kann, bedient man sich einer Zwischensprache, vergleichbar mit dem Java Bytecode. Im .NET Framework heißt diese **Intermediate Language IL**, manchmal auch **MSIL**. Die CLR arbeitet nur mit der IL, also nicht mit Quelltexten in C#, Visual Basic .NET oder anderen Sprachen. Jede Sprache des .NET Frameworks besitzt einen eigenen Compiler mit, der aus ihrem Quelltext den IL-Code erzeugt. Dieser wird dann von der CLR ausgeführt. Wie dies im einzelnen funktioniert, folgt im nächsten Kapitel.

Die Compiler erzeugen nicht nur den IL Code, sondern ein **verwaltetes Modul (managed module)**, das aus dem IL Code und Metadaten besteht. Diese Metadaten enthalten weitere Informationen über verwendete und selbst angebotene Typen usw. Die kleinste ausführbare Einheit ist ein Assembly, welches aus einem verwalteten Modul mit einem Manifest (weiteren Metadaten) besteht. Dazu mehr in Kapitel 2.4.

Es liegt im Windows-Standard-PE-Format (portable executable) vor und kann nur mit der CLR ausgeführt werden, obwohl es rein äußerlich eine EXE-Datei ist. Seine Struktur ist in Abbildung 3 dargestellt.

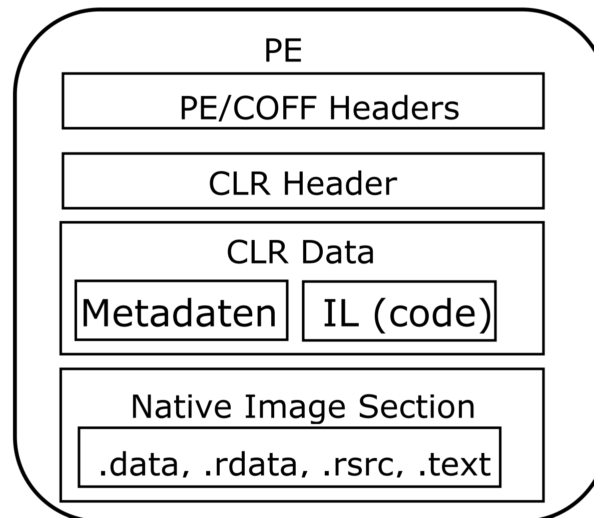

Abbildung3: .NET PE –File [Thai & Lam01] S.17

Der PE-Header ist für verwaltete Module nicht so wichtig. Er definiert den Dateityp und enthält einen Zeitstempel.

Der CLR-Header speichert einige zusätzliche Informationen, die von der CLR oder anderen Dienstprogrammen ausgewertet werden. Dies sind z.B. die CLR-Version, einige Flags, das Metadatentoken MethodDef, die Einstiegsmethode des verwalteten Moduls (Methode Main) und Adresse und Größe von Metadaten des Moduls usw.

Die Metadaten enthalten mehrere Metadatentabellen, dabei gibt es zwei Typen: Definitionstabellen (definition table) und Verweistabellen (reference table). Die Definitionstabellen enthalten die Typen und Member, die im Quellcode definiert sind und die Verweistabellen die Typen und Member, auf die zugegriffen wird.

Der IL-Code ist der Zwischencode, der später von der CLR in CPU-spezifische Befehle kompiliert wird.

Es gibt nicht nur **verwaltete Module (managed module)**, sondern auch **nicht verwaltete Module (unmanaged module)**. Diese nicht verwalteten Module sind Standard Windows Programme oder Bibliotheken, z.B. in C++ programmiert. (Nicht

mit dem C++ .NET Compiler übersetzt!) Damit man diese weiterbenutzen kann, gibt es die Möglichkeit sie aus .NET Anwendungen aufzurufen. Sie werden dann nicht von der CLR ausgeführt, sondern direkt, d.h. es findet keine Garbage Collection usw. statt.

2.2 Ausführungsmodell

Nachdem nun ein Programm z.B. in C# geschrieben wurde, wird es zuerst in die Intermediate Language übersetzt und dann von der CLR ausgeführt. Dies sieht im einzelnen etwa so aus:

1. Der Quelltext wird mit dem C#-Compiler (csc.exe) übersetzt. Der Compiler erzeugt eine EXE-Datei im PE-Standardformat. Diese ist ein Assembly (verwaltetes Modul mit Manifest (weitere Metadaten) (s.u.)). Der Compiler importiert dabei immer die Methode _CorExeMain aus der MSCorEE.dll.

2. Wird die Anwendung nun ausgeführt, lädt das Betriebssystem ganz normal die PE-Datei und alle eingebundenen DLL–Dateien (dynamic-link library), wie es auch bei anderen Windowsanwendungen üblich ist. Es wird dabei vor allem die _CorExeMain geladen, die für die weitere Ausführung existentiell ist.

3. Nun startet der Lader des Betriebssystems im Eintrittspunkt der PE-Datei (siehe .text in der Native Image Section), wo sich der Aufruf JMP _CorExeMain befindet.

5. Die_CorExeMain kümmert sich nun um den IL-Code und da dieser nicht direkt ausgeführt werden kann, muss er erst noch in Maschinensprache übersetzt werden. Die _CorExeMain kompiliert jetzt nach Bedarf den IL-Code mit einem Just-In-Time Compiler. Es werden dabei allerdings nur die Methoden übersetzt, die auch benötigt werden. Diese werden nach der Übersetzung in einem Cache gespeichert und können so bei einem erneuten Aufruf direkt ausgeführt werden.

6. Als letztes startet die _CorExeMain den übersetzten Code; das Programm läuft.

Es gibt noch zwei weitere Methoden zu der hier beschriebenen Standardvariante. Für Systeme mit wenig Speicher, z.B. PDAs u.ä. gibt es einen EconoJIT, der die

Möglichkeit hat, den Methoden Cache während der Laufzeit wieder zu leeren, um Speicherplatz zu gewinnen. Natürlich müssen dann Methoden neu übersetzt werden, wenn sie aus dem Speicher gelöscht wurden. Eine andere Variante bietet die Codeerzeugung bei der Installation. Hier wird während der Installation der gesamte Code optimiert und in Maschinensprache übersetzt. Hierfür wird das Tool *EGen.exe* benutzt.

2.2.1 Vergleich: Java Ausführungsmodell

Das Java Ausführungsmodell funktioniert grundsätzlich ähnlich, denn es arbeitet ebenfalls mit einer abstrakten Maschine.

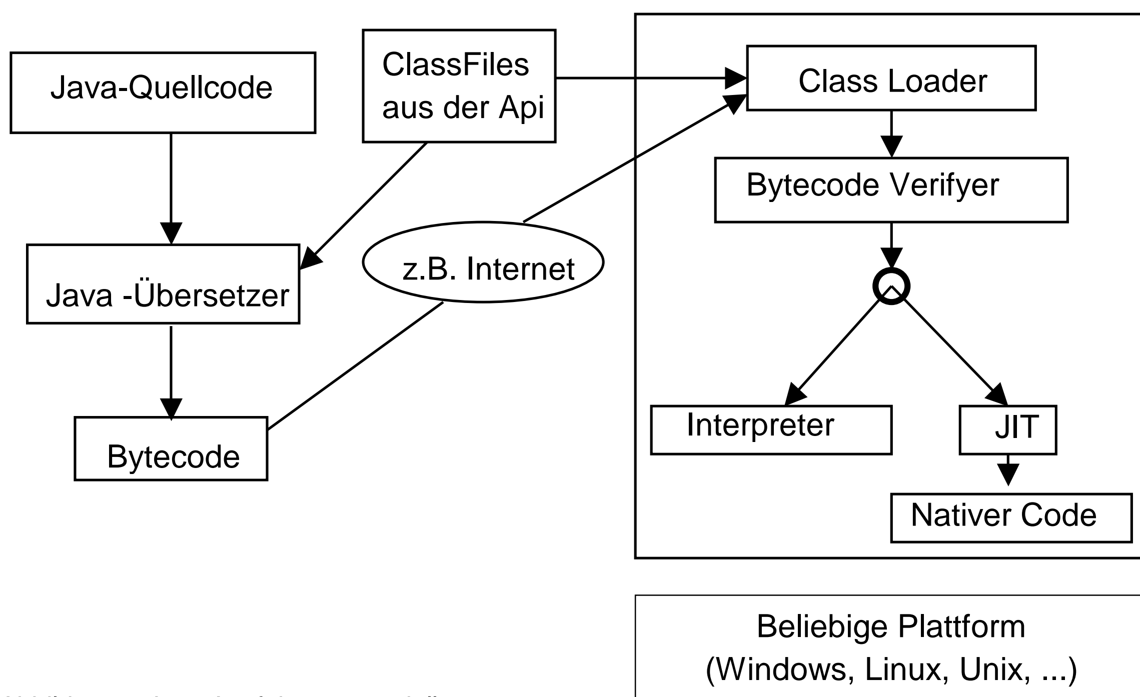

Abbildung4: Java Ausführungsmodell

Zuerst wird der Java Quellcode in den Bytecode übersetzt. Der Bytecode wird dann, z.B. per Internet zur Java Virtual Maschine übertragen. Dort lädt der Class Loader die übertragenen Klassen und die aus der API. Anschließend wird der Bytecode, bevor er nun ausgeführt wird, überprüft. Anfangs war es an dieser Stelle nur möglich den Bytecode zu interpretieren, was nicht besonders gut für die Performance ist. Just-In-Time Compiler, die direkt nativen Code erzeugen sind in der Regel schneller. Es gibt Implementierungen der JVM, die hier eine Entscheidung treffen, ob kompiliert wird,

mit oder ohne Speicherung des Maschinen-Codes in einem Cache und einer Codeoptimierung, oder ob der zusätzliche Aufwand sich nicht lohnt und direkt interpretiert wird. Es gibt zusätzlich noch Varianten, wo der Code zuerst eine gewisse Zeit beobachtet wird und dann entschieden wird, ob Maschinen-Code erzeugt und optimiert wird.

Die CLR des .NET Frameworks bietet, wie oben beschrieben, keine so optimierte Variante an. Dort wird immer Just-In-Time kompiliert. Ob daraus Geschwindigkeitsunterschiede entstehen ist im Augenblick schwierig zu bestimmen, weil Microsoft Veröffentlichungen zu Geschwindigkeitsanalysen nur nach vorheriger Absprache erlaubt. Hier muss man sich also selbst ein Bild machen.

2.3 Klassenbibliothek des .NET Frameworks

Das besondere an der Klassenbibliothek des .NET Frameworks ist, dass sie von allen CLR-konformen Sprachen benutzt wird. Dies ist wichtig für die Interoperabilität der Sprachen. Dies hat außerdem dazu geführt, dass die Sprachen selbst, fast keinerlei eigene Funktionen mitbringen. Es ist zwar schon seid einiger Zeit üblich, dass Sprachen z.B. für IO Zugriffe auf mitgelieferte Bibliotheken zurückgreifen, hier allerdings ist dieser Trend sehr konsequent fortgeführt worden, und es ist eine einzige Bibliothek für alle Sprachen übrig geblieben.

Die Abbildung 3 stellt eine Übersicht über die Klassen dar. Diese sind in sechs Gruppen unterteilt: Die Klassen im `System.Web`-Namespace bieten Funktionen für Webzugriffe an; sowohl für die Web Services (`System.Web.Services`), wie auch für die Web Server (`Web.Ui`). Der Namespace `Windows.Forms` wird für Windows-GUI-Anwendungen benutzt und unter dem Namespace `System` gibt es unter `System.ServiceProcess` Klassen für Windows Dienste. (vergl. Abbildung 1) Drei weitere Gruppen gibt es für Datenbankzugriffe (`System.Data`), sowie für XML-Anwendungen (`System.XML`) und für 2D-Grafik (`System.Drawing`).

System.Web		System.Windows.Forms	
Services	UI	Design	ComponentModel
Description	HtmlControls		
Discovery	WebControls	System.Drawing	
Protocols			
Caching	Security	Drawing2D	Printing
Configuration	SessionState	Imaging	Text
System.Data		System.XML	
ADO	SQL	XSLT	Serialization
Design	SQLTypes	Xpath	
System			
Collections	IO	Security	Runtime
Configuration	Net	ServiceProcess	InteropServices
Diagnostics	Reflection	Text	Remoting
Globalization	Resources	Threading	Serialization

Abbildung 5: Überblick über die .NET Framework Class Library vergl. [Archer01] Seite 32

Insgesamt lässt sich erkennen, dass es eine vollständige Klassenbibliothek gibt, die die Standardaufgaben, wie Datenbankzugriff, GUI, Netzwerkzugriffe usw. von Haus aus mitbringt. Dies ist für den Erfolg und die Akzeptanz des .NET Frameworks sicherlich eine wichtige Vorraussetzung.

2.3.1 Vergleich zur Java Klassenbibliothek

Die Java Klassenbibliothek besteht aus über 1500 Klassen, die ebenfalls wie die Framework Class Library des .NET Frameworks, sämtliche Standardaufgaben abdecken. Java bietet seit kurzer Zeit zusätzlich ein Zusatzpack, das in zukünftige Versionen integriert sein wird, an, das Unterstützung für XML-Parser, XML-basierte Web Services mit SOAP, WSDL und UDDI liefert.

Hier ein Ausschnitt aus der Java Klassenbibliothek:

java.applet	Klassen für die Entwicklung von Applets
java.awt	**A**bstract **W**indowing **T**oolkit enthält: • Grafik: Farben, Schriften, Figuren • Komponenten: Buttons, Menüs, Listen, Dialogfenster • Layout Manager: Anordnung von Komponenten und Verhalten zueinander • Subpackages für Farbmanagement, Drag & Drop, Fonts, Eventhandling etc.
java.beans	Klassen für die Entwicklung von Java Beans
java.io	Ein- und Ausgabe über Streams
java.lang	Sprachnahe Funktionalität (z.B. Metainformation)
java.math	Mathematische Funktionen und zusätzliche mathematische Datentypen
java.net	Klassen für die Entwicklung von verteilten Applikationen
java.rmi	RMI – **R**emote **M**ethod **I**nvocation: Methodenaufrufe über VM-Grenzen hinweg
java.security	Klassen des Security Frameworks: Verschlüsselung, Authentisierung, Zertifikate
java.sql	JDBC – **J**ava **D**ata **B**ase **C**onnectivity: Zugriff auf SQL Datenbanken
java.text	Sprach- und länderspezifische Behandlung von Datentypen
java.util	Collection-Klassen, Datum, Uhrzeit, Zugriff auf jar- und zip-Files
javax.accessibility	Zugriff auf UI mit alternativen Mitteln (z.B. Braille-Terminal)
javax.swing	Klassen für GUI Entwicklung, grosse Menge von Subpackages
org.omg	CORBA APIs (verteilte Objekte, Broker, Naming Service)

Es gibt im Augenblick keine nennenswerten Unterschiede, was die Klassenbibliothek betrifft. Interessant ist hier sicherlich die zukünftige Entwicklung, den beide bieten ein vollständiges Konzept aus einer Hand an: Auf der einen Seite Microsoft mit dem .NET Framework und auf der anderen Sun mit seinem Sun Open Net Enviroment (SunONE). Welche Web Services, Standards für verteilte Anwendungen usw. sich nun wirklich durchsetzen und in weit sie kompatibel zueinander sein werden, bleibt abzuwarten.

2.4 Assemblies

Assemblies sind die kleinsten Einheiten für Wiederverwendung, Versionierung und Sicherheit. Sie bestehen aus einem **Manifest**, aus einem oder mehreren verwalteten Modulen und können zusätzlich Resourcendateien, wie Bilder, Dokumente usw. enthalten. Das Manifest ist eine Gruppe von Metadatentabellen, die Informationen über Dateien im Assembly, wie öffentlich exportierte Typen, die Resourcen- und Datendateien, die im Assembly enthalten sind, die Version, Herausgeber und einige weitere enthalten. Abbildung 6 stellt ein Assembly namens App.exe dar, das aus nur einem verwalteten Modul besteht und keinerlei Resource- oder Datendateien enthält.

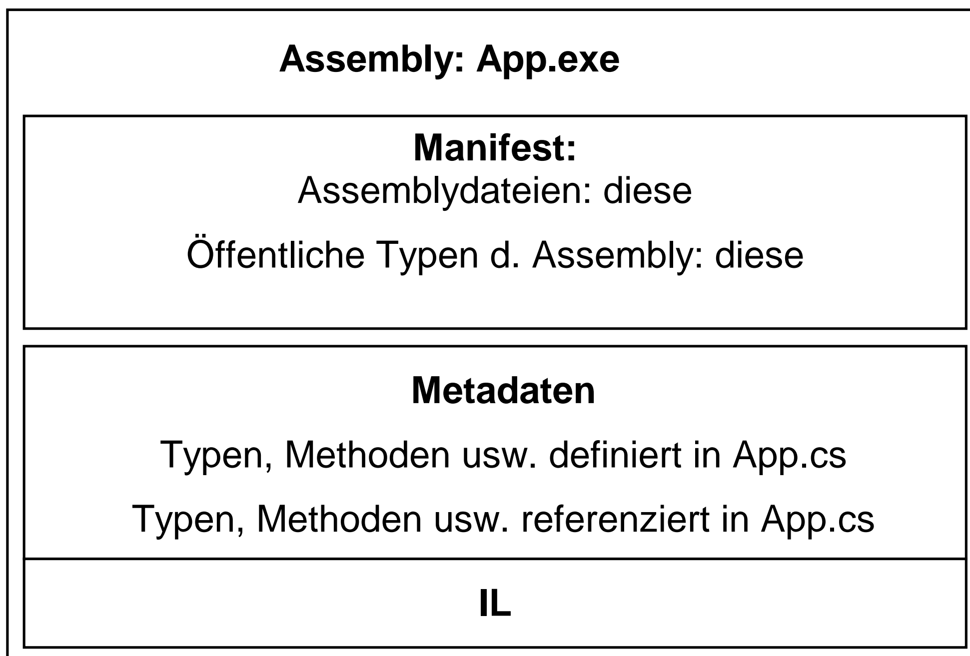

Abbildung 6: Aufbau eines Assemblies bestehend aus einem verwalteten Modul, vergl. [Archer01] S.32

Durch die Hilfe des Manifestes ist eine zentrale Registrierung eines Assemblies nicht notwendig; es tritt dadurch nicht das Problem auf, dass sich gleichbenannte Bibliotheken überschreiben oder bei der Deinstallation eines Programms die gemeinsam mit einem anderen Programm benutzten Bibliotheken entfernt werden. Assemblies verlinken selbstständig auf andere und es ist so keine komplizierte Installation notwendig. Möchte man Assemblies auch für andere freigeben, meldet man sie beim GAC (global assembly cache) mit Hilfe des Tools *gacutil.exe* an. Da ein Name nicht eindeutig genug ist, bekommt das Assembly einen Private und einen Public Key, der ähnlich wie die asynchrone Verschlüsselung arbeitet. Es bekommt einen eindeutigen Hashwert, den jeder mit Hilfe des Public Keys abfragen kann. So können Assemblies nebeneinander in verschiedenen Versionen oder Sprachen usw. existieren, jedes Programm kann sich aber sicher sein, dass es das richtige zur Verfügung hat.

2.5 Vergleich Java: Classfiles

Die Java Classfiles enthalten ähnliche Informationen wie die Assemblies: Ein Magic zur Identifizierung, die Version, die Konstantenmenge, ein Accesflag (class oder interface, public oder abstract usw.), einen Zeiger auf die Klasse selbst und einen auf die Elternklasse, Interfaces, Felder vor allem aber Methoden und Attribute. Zu den Unterschieden gehört, dass für jede Klasse eine einzelne Datei angelegt wird. Es gibt hier dann zwar die Möglichkeit die Classfiles zu einem .jar – Archive zusammenzupacken, was dann aber doch kein so enges Konstrukt wie ein Assembly ist. Die Unterschiede zwischen den Java Classfiles und den Assemblies sind trotzdem eher gering. Microsoft ist hier allerdings eine deutliche Verbesserung zu seinem eigenen COM Model gelungen.

2.6 Fazit

Microsoft hat mit seinem neuen .NET Framework sicherlich das Rad nicht neu erfunden, hat aber viele gute Ideen aus der Javawelt übernommen und ein paar Kleinigkeiten dazu entwickelt. Dinge wir die Garbage Collection usw. bringen sicherlich mehr Stabilität, als man es bisher z.B. von C++ kannte, was andererseits auf Kosten der Geschwindigkeit geht. Durch die Vielfalt der unterstützten Sprachen, es gibt sogar einen J# .NET Compiler, der zu Java 1.1.x Code kompatibel ist, soll es Entwicklern vereinfacht werden umzusteigen oder sogar ihre aktuellen Produkte zu übernehmen. Dies ist auch schrittweise möglich, da alte Programme auf herkömmliche Weise, also ohne CLR, von .NET Programmen ausgeführt werden können.

Ob es das .NET Framework wirklich einmal auch für Rechner und Systeme mit denen Microsoft nichts zu tun hat, wie Linux, geben wird, ist im Augenblick nur schwer abzuschätzen, denn es gibt einige lizenzrechtliche Vorbehalte, die noch zu klären sind. Außerdem halten einige Experten die CLI für nicht aussreichend.

Grundsätzlich ist das .NET Framework sicherlich eine ernst zu nehmende Entwicklung, die in Zukunft aus der Windows Welt nicht wegzudenken sein wird, denn die nächsten Windows Versionen werden das .NET Framework standardmäßig mitbringen und Microsoft Produkte, wie das Officepaket, wurden zum Teil schon jetzt umgestellt.

3 Literaturverzeichnis

- **[Richter02] Richter, Jeffrey: Microsoft .NET Framework Programmierung .net Fachbibliothek, Microsoft Press, 2002**
- **[Archer01] Archer, Tom: Inside C#, Microsoft Press, 2001**
- **[Thai & Lam01] Thai, Thuan und Lam, Hoang Q.: .NET Framework Essentials, O`Reilly & Associates, Inc., 2001**
- **[CT4/02] c't 04/2002: Das Microsoft-Internet, Sunspiration, Die neue C-Klasse, Verlag Heinz Heise, 2002**
- **[wrox00] Conard, J., Dengler, P., Francis, B., Glynn, J., Harvey, B., Holis, B., Ramachandran, R., Schenken, J., Short, S., Ullman, C.: Introducing .NET, Wrox Press Ltd., 2000**
- **[Bangay02] Shaun Bangay: Rotor comes to Linux, http://www.oreillynet.com/pub/a/dotnet/2002/07/01/rotorlinux.html**
- **[Mono] Open Source development of a Unix version of the Microsoft .NET development platform: http://www.go-mono.com/**