.NET Framework Kompakt

(Version 0.3)

Dr. Wolfgang Thronicke

C-LAB Report

Vol. 1 (2002) No. 1

Cooperative Computing & Communication Laboratory

ISSN 1619-7879

C-LAB ist eine Kooperation
der Universität Paderborn und der Siemens Business Services GmbH & Co OHG
www.c-lab.de

C-LAB Report

Herausgegeben von
Published by

Dr. Wolfgang Kern, Siemens Business Services GmbH & Co OHG
Prof. Dr. Franz-Josef Rammig, Universität GH Paderborn

Das C-LAB - Cooperative Computing & Communication Laboratory - leistet Forschungs- und Entwicklungsarbeiten und gewährleistet deren Transfer an den Markt. Es wurde 1985 von den Partnern Nixdorf Computer AG (nun Siemens Business Services GmbH & Co OHG) und der Universität Paderborn im Einvernehmen mit dem Land Nordrhein-Westfalen gegründet.
Die Vision, die dem C-LAB zugrunde liegt, geht davon aus, dass die gewaltigen Herausforderungen beim Übergang in die kommende Informationsgesellschaft nur durch globale Kooperation und in tiefer Verzahnung von Theorie und Praxis gelöst werden können. Im C-LAB arbeiten deshalb Mitarbeiter von Hochschule und Industrie unter einem Dach in einer gemeinsamen Organisation an gemeinsamen Projekten mit internationalen Partnern eng zusammen.

C-LAB - the Cooperative Computing & Cooperation Laboratory - works in the area of research and development and safeguards its transfer into the market. It was founded in 1985 by Nixdorf Computer AG (now Siemens Business Services GmbH & Co OHG) and the University of Paderborn under the auspices of the State of North-Rhine Westphalia.
C-LAB's vision is based on the fundamental premise that the gargantuan challenges thrown up by the transition to a future information society can only be met through global cooperation and deep interworking of theory and practice. This is why, under one roof, staff from the university and from industry cooperate closely on joint projects within a common research and development organization together with international partners. In doing so, C-LAB concentrates on those innovative subject areas in which cooperation is expected to bear particular fruit for the partners and their general well-being.

ISSN 1619-7879

C-LAB
Fürstenallee 11
33102 Paderborn
fon: +49 5251 60 60 60
fax: +49 5251 60 60 60
www.c-lab.de

.NET Framework Kompakt

(Version 0.3)

W.Thronicke

Vorwort

Dieser Report entstand als Zusammenfassung verschiedener Literatur- und Internetquellen, die im Kapitel 10 angegeben sind. Ich danke meinen Kollegen für die freundliche Durchsicht und Kommentierung der Version.

1. Einleitung

His words are a very fantastical banquet,
just so many strange dishes.
(Much ado about nothing, 2.Akt, 3. Szene)

.NET beschreibt nicht nur eine Marketingstrategie von Microsoft oder ein neues Produkt. Es ist ein (für Microsoft) neues Konzept zur Entwicklung, Verteilung, Ausführung und Verwaltung von Applikationen und Komponenten und deren Entwicklung mit einer Ausrichtung auf standardisierte, plattform-unabhängige Webtechnologien. Dabei löst die .NET Technologie die vorhandenen (Windows-) Architekturen ab.

Diese Darstellung fasst die *technischen* Aspekte des .NET Framework zusammen.

2. Technische Motivation für .NET

Engel und Boten Gottes steht mir bei
(Hamlet, 1. Akt, 4. Szene)

Die typische Microsoft-Technologie ist - nicht zuletzt aus Kompatibilitätsgründen - durch gewachsene Strukturen und schrittweise Weiterentwicklung gekennzeichnet. Eine Folge der so entstandenen Altlasten ist, dass aktuelle Anforderungen an Software nur schwer integriert werden können und Erweiterungen der vorhandenen Architektur keine optimalen Lösungen bieten.

Ein anderer Punkt ist die Sicht der Entwickler: Aktuelle Technologien insbesondere im Internetbereich müssen sehr schnell umgesetzt werden, um am Markt zu bestehen. Das bedeutet, die Entwicklung selbst muss effizient, verständlich und gemäss aktuellen Vorgehensweisen möglich sein.

Beispiel 1: COM / DCOM

COM/DCOM ist die (verteilte) Microsoft Komponentenimplementierung. ActiveX und alle Scripting, Server Technologien der Pre-.NET Ära basieren hierauf.

Für Internetszenarien ist COM bzw. DCOM nicht gut einsetzbar, weil konzeptionell eine permanente Verbindung zwischen COM Server und Client bestehen muss. Zudem ist das eingesetzte Binärprotokoll problematisch, sobald unterschiedliche Plattformen mit abweichenden Darstellungsformaten ins Spiel kommen. Die Komponenten werden üblicherweise in DLL[1]-Dateien abgelegt. Der Mechanismus der Identifikation dieser Dateien und damit Installation und Verwaltung dieser Dateien ist nicht optimal gelöst und führt nach Aussage vieler Entwickler in die gefürchtete „DLL-Hölle", d.h. unterschiedliche Versionen von DLLs mit unterschiedlichen Schnittstellen sorgen für instabile Programmsysteme oder schlimmstenfalls für ein nicht funktionierendes System.

Aus Programmierersicht stellt die Entwicklung und das Debugging solcher Komponenten hohe Anforderungen. Teilweise ist die Programmierung durch die notwendigen Makros[2] sehr kryptisch und verdeckt das eigentlich zu behandelnde Problem.

Beispiel 2: Multi-Language Programmierung

Microsofts Entwicklungsumgebung[3] unterstützt eine Reihe populärer Programmiersprachen: C, C++, J++ (eine JAVA Variante) und Visual Basic. Da diese Sprachen unterschiedliche interne Datentypen verwenden, ist die Verwendung einer Visual-Basic Komponente aus einem C++ Programm heraus und umgekehrt nicht trivial. Außerdem folgt die Programmierung unterschiedliche Konzepten, was sich in unterschiedlichen Bibliotheken für Oberflächen und Hilfsfunktionen und -klassen ausdrückt.

3. .NET Übersicht

Stand! Who is that?
(Hamlet, 1. Akt, 1.Szene)

Die .NET Plattform kann aus verschieden Blickwinkeln betrachtet werden: Der Produktsicht, der Funktionssicht und der Architektursicht.

Aus *Produktsicht* präsentiert sich .NET durch folgende Produktgruppen:

- Entwicklungswerkzeuge:
  Programmiersprachen und entsprechende Tools (Visual Studio .NET), Klassenbibliotheken zur Erstellung von Webservices und Windowsprogrammen und die CLR (Common Language Runtime)
- Server:
  gehen aus den bekannten SQL 2000, Exchange 2000, BizTalk 2000 ... hervor, z.B. Internet Information Server 6.0

[1] Dynamic link library

[2] Dies gilt für die Entwicklung in C und C++.

[3] Visual Studio 6.0

- Web Services:
  Hailstorm (z.B.: Passport zur Benutzerauthentifizierung)
- Geräte (Devices):
  PCs, PDAs, WebPads, Handys usw. als Plattformen für die Ausführung von .NET Programmen.

Die Betriebssysteme werden noch nicht als Teil der .NET Landschaft angesehen. Allerdings unterstützen die modernen Varianten die Ausführung von .NET Komponenten, ab Windows 2000 bzw. Windows Pocket CE 2002.

Von der *Funktionalität* her reflektiert .NET die neuen Entwicklungen der IT-Landschaft:

- Komponenten zur Gestaltung verteilter Softwaresysteme
- Komponentenarchitektur

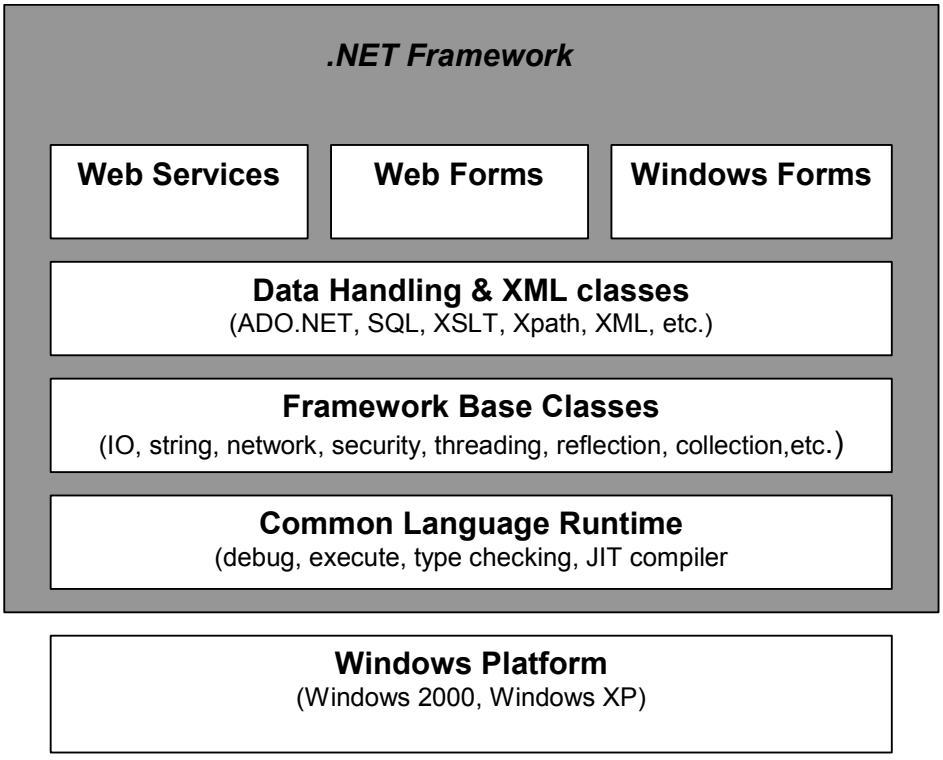

Abbildung 1: Das .NET Framework

- Enterprise Applikationen (skalierbare, unternehmensweit einsetzbare Anwendungen)
- Skalierbarkeit und Konfigurierbarkeit der Komponenten
- Web Anwendungen (Webservices)

Ein wichtiger Aspekt dieser (nicht neuen) Funktionalitäten ist ein Re-Design in Richtung Beherrschbarkeit und Klarheit der Plattform für die Softwareentwick-

lung und Anwendung. Die zentrale Maßnahme ist die Vereinheitlichung der Komponenten der .NET-Plattform. So werden zur Kommunikation und Identifikation der Komponenten die neuen XML-basierten Standards SOAP, WSDL und Mechanismen wie UDDI favorisiert, welche die bislang Microsoft eigenen Mechanismen COM, COM+ und DCOM ergänzen. Die Anwendung der standardisierten Internettechnologien und -formate ermöglichen die plattformübergreifende Kommunikation und ebenfalls die aus dem JAVA Umfeld bekannte Serialisierung von Objekten (hier: .NET Objekte)[4]. Diese können per Internet verteilt und eingesetzt werden.

Die *Architektur* des .NET Frameworks zeigt Abbildung 1. Dabei kennzeichnet das .NET Framework die Laufzeitumgebung der entwickelten .NET Komponenten. Genau genommen besteht das .NET Framework selbst auch bereits aus .NET Komponenten, die auf diese Weise erweitert werden könnten.

Dieses Framework ist hierarchisch aufgebaut. Auf unterster Ebene sind in der Common Language Runtime (CLR) alle Komponenten angesiedelt, welche andere .NET Komponenten ausführen, debuggen und überprüfen. Die Basisdienste realisieren wichtige, oft gebrauchte Dienste und Bibliotheken, auf denen die weiteren Schichten aufbauen. Dazu zählt die Datenzugriffschicht, welche das Arbeiten mit Datenbanken u.ä. regelt und zusätzlich die XML Funktionsbibliotheken enthält für die Erzeugung und Behandlung von XML Dokumenten. Zur Gestaltung der Schnittstellen neuer .NET Anwendungen stehen grafische Elemente durch die Windows und Web Forms Komponenten zur Verfügung, sowie die Webservice Komponente für Internet zentrierte Komponentenentwicklung.

Die Struktur der .NET Executables und Assemblies

.NET bringt einen wichtigen Unterschied in die Windowswelt: Die ausführbaren Programme enthalten keinen für eine Plattform übersetzten Maschinen-Code sondern eine Zwischensprache (Intermediate Language (IL)). Diese wird vor der Ausführung auf der Zielplattform übersetzt. Deshalb sind die ausführbaren Programme portabel und heißen PEs (= Portable Executables). Ein solches PE besteht aber nicht nur aus dem Zwischencode sondern enthält folgende Daten:

- Der Zwischencode des Executables
- Die Metadaten (Informationen über das PE selbst)

Diese Metadaten enthalten unter anderem:

- Versionsinformationen
- Typ Bibliotheken (type libraries) zur Beschreibung der die Schnittstellen und Methodensignaturen der Komponente.
- Spezifikation der notwendigen externen Bezüge.

Im .NET Umfeld gelten Komponenten dieser Art, wenn sie wiederverwendet werden als sog. „Bausteine“ oder „Assemblies“. Eine Assembly ist die kleinste wie-

[4] hier: in ein XML Dokument umwandeln.

derverwendbare Einheit in der .NET Architektur. Eine „Assembly“ kann eine oder mehrere PEs und oder Dateien enthalten. Damit ist sie ungefähr analog zu einer COM Komponente, die üblicherweise in einer oder mehreren DLL Dateien manifestiert ist.

Eine Assembly ist auch die Installationseinheit, die als .dll oder exe Datei installiert wird. Das .NET Laufzeitsystem unterscheidet sie von „gewöhnlichen“ ausführbaren Windowsprogrammen.

Der Zugriff auf die Daten innerhalb einer PE bzw. Assembly realisiert .NET über das „Reflection Interface“. Damit können andere Komponenten, beispielsweise die CLR oder der Just-In-Time Übersetzer, diese Informationen nutzen oder neue Daten in die PE integrieren. Diese Schnittstelle können auch Drittanbieter für die Entwicklung eigener Compiler und Werkzeuge nutzen.

Der Inhalt der PEs kann mit dem Programm „dumpbin.exe“ als Text ausgegeben werden. Mit dem Tool „ildasm.exe“ werden die Metainformationen grafisch aufbereitet. „xsd.exe“ kann aus Assemblies die Typinformationen als XML Schema gewinnen und umgekehrt aus einer Schemadefinition eine C# Klasse erzeugen, die diesem Datentyp entspricht.

Wiederverwendung existierender COM-Komponenten

Da COM Objekte anders strukturiert sind als .NET PEs, muss mit einem Wrapper gearbeitet werden, der ein .NET konformes PE für die COM Komponente erzeugt. Dieser Wrapper baut aus dem COM Interface die Metadaten des PEs auf. Der umgekehrte Weg wird ebenfalls unterstützt, so dass auch „alte“ Anwendungen auf .NET Komponenten zugreifen können (d.h. man bekommt einen COM Server in Form einer DLL).

Anmerkung: Hierbei besteht natürlich die Gefahr, dass wieder sehr unübersichtliche Systeme entstehen können, die die Nachteile der COM Technologie in sich tragen. Andererseits bleibt die Menge an nutzbaren COM Elementen für Programmierprojekte und Anwender erhalten, bis äquivalente .NET Komponenten verfügbar sind.

Die Common Language Runtime

For them all together; which maintained so politic a state of evil that they will not admit any good part to intermingle with them.
Much ado about nothing, 5,Akt, 2. Szene

Die CLR führt die PEs aus. Dazu werden folgende Aktionen durchgeführt:

- Überprüfen der Sicherheitskriterien (Zugriffsrechte etc.)
- Laden der benötigten abhängigen Assemblies
- Übersetzen des Zwischencodes in plattformspezifischen Code
- Ausführung des kompilierten Codes

Das bedeutet: Ein Programm wird vor seiner Ausführung übersetzt und geprüft. Der Just-In-Time Compiler führt dazu die Phasen „Code Verifier“ und den ei-

gentlichen Übersetzungsschritt aus. Der Verifier überprüft, ob die Metadaten gültig sind nach Form und Bedeutung und ob der Zwischencode typsicher ist und die Typ-Signaturen übereinstimmen.

Anmerkung: Die Kosten für die jedes Mal stattfindende Übersetzung[5] können durch ein sogenanntes „pre-JITting" vermieden werden. Bei diesem Vorgehen kann eine Assembly beispielsweise bei der Installation kompiliert werden, so dass der Aufwand minimiert wird. Allerdings kann die Konfiguration (d.h. mit welchen Versionen anderer Assemblies gearbeitet wird) nicht mehr so flexibel eingestellt werden.

Die Identität von Assemblies

Die CLR kontrolliert das Einbinden externer Komponenten mit Hilfe der Metainformationen. Durch die integrierte Versionsnummer kann die passende Variante einer Assembly identifiziert werden[6]. Für gemeinsam genutzte („shared") Assemblies wird die Integrität durch kryptografische Verfahren[7] gesichert: .NET erzeugt einen Hashcode über die Dateien der Assembly und speichert diesen in den Metainformationen ab. Zum Schutz vor Veränderungen wird dieser Hash signiert und der passende Prüfschlüssel ebenfalls in der Assembly codiert. Beim Linken einer nutzenden Assembly wird dieser Schlüssel in die neue Assembly mit eingebaut. Die CLR prüft dann mit diesem Schlüssel beim Ladevorgang, ob die gefundene „shared" Assembly die passende ist.

Der Zwischencode (IL = intermediate language)

Der Programmcode einer Komponente liegt in .NET als portabler Zwischencode vor. Da dieser von der CLR geladen und in endgültigen Maschinencode übersetzt wird, erzeugen die Compiler der Programmiersprachen diesen Zwischencode als Ergebnis. Der Zwischencode von .NET unterstützt die bekannten Eigenschaften objektorientierter Programmiersprachen, d.h. Klassen- und Vererbungskonzepte sind enthalten. Die Beschreibung der Zwischensprache ist ebenfalls offengelegt, um die Compilerentwicklung von Drittanbietern zu unterstützen.

Mit „ildasm.exe" kann eine Textausgabe der Zwischensprache erzeugt werden. Analog kann mit „ilasm.exe" Zwischensprache-Code in Bytecode umgesetzt werden.

---

[5] Bereits übersetzte PEs bleiben während ihrer Laufzeit in einem Systemcache, d.h. andere sie nutzende Assemblies erfordern nicht ihre Neuübersetzung.

[6] In der COM Architektur war das problematisch, da die DLLs an wenigen Stellen mit einem festen Namen installiert werden mussten. Somit war es problematisch und fehlerträchtig, wenn mehrere (evtl. inkompatible) Versionen installiert wurden („DLL-Hell").

[7] .NET setzt ein public-key Verfahren ein. Für das Verständnis genügt zu wissen, dass .NET mit diesen Schlüsseln die Integrität überprüfen kann.

Das Common Type System (CTS) und die Common Language Specification (CLS)

Das gemeinsame Typsystem ist der Schlüssel für die Interoperablität von Komponenten, die in unterschiedlichen Programmiersprachen (z.B. Visual Basic, C#, C++ etc.) realisiert werden können.

Das CTS definiert zwei unterschiedliche Typ-Klassen: *Value-Typen* und *Referenz-Typen*. Variablen der Valuetypen werden auf dem Stack abgelegt und beim Verlassen des Gültigkeitsbereichs zerstört. Dies entspricht dem üblichen Mechanismus lokaler Variablen in Programmiersprachen. Referenzen werden auf dem Heap erzeugt und sind Objekte, die von der Objekt-Basisklasse abgeleitet sind. Valuetypen sind beispielsweise integer, float, Strukturen oder Aufzählungen (siehe auch [ProfC#])[8].

Die Konvertierung zwischen beiden Typ-Ausprägungen wird „*boxing*“ bzw. „*unboxing*“ genannt. In Programmiersprachen sieht dies syntaktisch wie ein *typecast* aus. Das Typsystem unterstützt alle objektorientierten Konzepte: Klassen (einfache (!) Vererbung)[9], Interfaces, u.s.w. Zusätzlich sind weitere Elemente vorhanden: *Properties, Indizierungen* und *Delegates*.

Properties implementieren eine *get* und *set* Methode auf Variablen (ähnlich wie die Java Beans Konvention). Indizierungen können auf Arrays definiert werden. Sie sind Operatorüberladungen mit denen Iterationen über solche Felder programmiert werden können. Delegates entsprechen typsicheren Funktionspointern, wie sie aus C bekannt sind. Mit einem Delegate kann beispielsweise typsicher eine Methode einer Klasse einem entsprechenden Eventhandler usw. zugeordnet werden.

Die offengelegte *Common Language Specification* (CLS) definiert die Eigenschaften der Programmiersprachen, welche die Interoperabilität betreffen. Jeder Entwickler einer .NET konformen Programmiersprache muss sich danach richten. Alle neuen Programmiersprachen der .NET Plattformen orientieren sich hieran - auch die IL.

Die Ausführung von Programmen durch die CLR

Jedes .NET Executable durchläuft bei der Ausführung folgenden Schritte:

1. Den „Loader“
2. Den JIT Compiler bestehend aus den Phasen „Code Verifier“ und dem eigentlichen Just-In-Time Compiler. Der Verifier überprüft, ob die Metada-

[8] Valuetypen werden immer aus Effizienzgründen für temporäre, kurzlebige Objekte gewählt. Jeder dieser Typen besitzt eine Reihe von Methoden, wie echte Klassen.

[9] Aus diesem Grund ist Visual .NET C++ bei .NET konformer Übersetzung auch eingeschränkt und entspricht nicht dem C++ Standard. C# unterstützt (wie JAVA) nur einfache Vererbung und Schnittstellenklassen (Interfaces).

ten gültig sind nach Form und Bedeutung und ob der Zwischencode typsicher ist, d.h. das die Typ-Signaturen korrekt sind.

3. Die Execution Engine.

Die JIT-Übersetzung wird beim ersten Aufruf einer Methode durchgeführt. D.h. Methoden, die während der Laufzeit nicht benötigt werden, werden auch nicht übersetzt. Der native Code wird „managed code“ genannt. Durch sog. Pre-Jitting kann bei der Installation eines PEs bzw. der Assembly der native Code direkt erzeugt werden (Tool *„ngen.exe“*).

Die Ausführung geschieht kontrolliert durch den Code-Manager (d.h. es existieren durch den JIT erzeugte Sprünge in den Manager). Dieser kümmert sich um Funktionalitäten wie Garbage collection, Exeception Handling und Sicherheitsaspekte, sowie Debugging und Interoperabilitätsunterstützung. Aus diesem Grund wird auch von „managed code“ gesprochen. Konventionelle Programme und COM Objekte repräsentieren hingegen „unmanaged code“. Die Ausführung von „managed code“ ist langsamer, soll aber wesentlich schneller als äquivalenter JAVA-Code sein.

4. Arbeiten mit .NET Komponenten

Round about the cauldron go;
In the poison'd entrails throw.
Macbeth, 4. Akt, 1. Szene

Die Installation und Benutzung von .NET Komponenten hat sich im Vergleich mit COM/DCOM stark vereinfacht. Im folgenden wird die Verteilung und die Funktionalität dieser „Assemblies“ betrachtet.

Verteilung und Installation von .NET Komponenten und Programmen

Stand-Alone Programme

Die Installation solcher Programme ist besonders einfach, da sie keine weiteren Komponenten (ausser den Systemkomponenten von .NET) benötigen. Einfaches Kopieren in ein Verzeichnis genügt[11].

Nutzung anderer externer .NET Assemblies (Bibliotheken)

Werden externe Komponenten genutzt (also „DLLs“ eingebunden) können diese an zwei Stellen installiert sein: Einmal im Dateisystem (z.B. im gleichen Verzeichnis wie die Applikationen) oder systemweit:

[11] Vermutlich wird es Installationsprogramme weiterhin geben, um beispielsweise Menüeinträge zu erstellen oder Registrierungen für Filetypen anzupassen.

- *Private Assemblies:* Private Assemblies befinden sich im gleichen oder einem vom Client erreichbaren Verzeichnis. Man kann sich solche Assemblies wie lokale Bibliotheken vorstellen. Die CLR führt bei privaten Assemblies keinen Versionstest durch (da angenommen wird, dass sie nur mit einer Applikation genutzt und in der Regel auch gemeinsam installiert werden). Der Ort der Komponenten muss bei der Übersetzung angegeben werden. Diese Pfadangaben werden bei der Erzeugung der Assembly eingetragen.
  `csc /r:<path>\comp1.dll;<path>\comp2.dll ...`
- *Shared Assemblies:* Diese müssen im globalen Assembly-Cache (GAC) registriert werden, bevor sie genutzt werden können[12]. Sie sind damit Systemkomponenten und tragen Versions- und Herkunftsinformationen. Aus diesem Grund enthalten diese Assemblies auch Authentifizierungsinformationen, die genutzt werden um die Integrietät zu überprüfen.

  Ablauf der Registrierung:
  - Mit „`sn.exe`“ wird ein public/private Schlüsselpaar erzeugt, welches als „originator key“ weiterbenutzt wird.
  - Die Versions- und Originator-Key Information wird *in den Programmtext* durch spezielle Anweisungen (Pragmas) angegeben. Bei der Übesetzung werden diese dann in die Assembly integriert. Beim Übersetzen werden spezielle Optionen benutzt. Es entsteht eine DLL als Übersetzungsergebnis.
  - Die Assembly wird mit „`gactutil.exe`“ in der GAC registriert.

Die Applikationen finden aufgrund der Versionierungsinformationen immer ihre passende shared assembly. Deshalb können auch unterschiedliche Versionen einer Assembly nebeneinander installiert werden. Es gibt auch die Möglichkeit anzugeben, dass eine aktuellere Version einer Assembly genutzt werden soll. Dazu wird die neue Konfiguration der Anwendung in einem `<application>.exe.config` File abgelegt, welches im gleichen Verzeichnis wie die exe-Datei liegen muss.

Verteilte .NET Komponenten

.NET unterstützt eine besondere API zur Verteilung von Komponenten, um „remote services“ bereitzustellen und anzusprechen. Im Vergleich zu DCOM vereinfacht die „Remoting API“ die Entwicklung und Pflege. Konzeptionell wird die Kommunikation zwischen Komponenten durch sog. „Channels“ abgewickelt, welche eine Abstraktion des eigentlichen Kommunikationsprotokolls implemen-

[12] Dies entspricht den System-DLL-Verzeichnissen der alten Technologie, ist aber wesentlich strukturierter organisiert (ähnelt einer Komponenten-Datenbank).

tieren. Durch die Verwendung eines http-Channels wird beispielsweise mit dem SOAP Protokoll kommuniziert. (Achtung! Das ist kein Webservice!).

COM+ und .NET

Auch vor .NET war es mit COM/DCOM möglich unternehmensweite Applikationen zu realisieren. Eine Reihe von Diensten ist so entstanden, die auch in .NET verfügbar sind bzw. nach und nach portiert werden. Die nachfolgend vorgestellten Services sind also auch .NET nutzbar.

Die COM+ Services fassen die COM und MTS (Microsoft Transaction Server) Dienste zusammen. Zu den wichtigsten zählen:

- *Objekt-Pooling*: Wenn es sehr aufwendig ist, ein Objekt immer wieder zu erzeugen und zu zerstören, kann es durch Pooling verwaltet werden. Das Objekt wird dann nur einmal erzeugt und bei der nächsten Benutzung wiederverwendet.
- *Transaktionen*: Unterstützung für das Zurückfahren von Operationen im Falle eines Fehlers in der Bearbeitung, besonders wichtig bei Datenbankanwendungen.
- *Rollenbasiertes Programmieren*: Durch diesen Dienst wird die Erstellung von Sicherheitsmodellen in Geschäftsanwendungen vereinfacht. Methoden von Klassen können über Pragmas[13] Rollen zugewiesen werden, die entsprechend zur Laufzeit abgefragt werden können. Die Zuweisung von Benutzern zu Rollen geschieht über das „Component Services“ Werkzeug zur Laufzeit.

Ein Prinzip neben der Vererbung von entsprechenden .NET-Klassen ist die „attributierende Programmierung“ durch Pragma-Anweisungen. Diese erzeugen bei der Übersetzung entsprechenden Code.

Nachrichten-Queuing

Message Queuing erlaub es .NET Objekte in lokale oder netzwerkweite Warteschlangen einzufügen und weiterzuverarbeiten. Die entsprechenden Methoden sind in .NET im `System.Messaging` Namensraum verfügbar.

Zusammenfassung

Die Entwicklung von Anwendungen insbesondere solcher mit COM / COM+ Diensten ist mit dem .NET Framework standardisiert und durch ein relativ einheitliches Klassen und Pragmaschema unterstützt. Die vorhandenen Dienste der Vorgängerarchitekturen sind übernommen worden.

[13] Compileranweisungen.

5. ADO.NET, XML und Datenrepräsentation

ADO.NET ist die Weiterentwicklung von ADO (ActiveX Data Object). ADO.NET integriert XML als portables Austauschformat und Repräsentationsformat. Die Architektur in Abbildung 2 zeigt die wichtigsten Klassen von ADO.NET. Die *DataSet* Klasse repräsentiert die Inhaltsobjekte (content objects). Die sogenannten „managed provider" Klassen sind für den Datenzugriff, -suche und ähnliche Funktionalitäten zuständig.

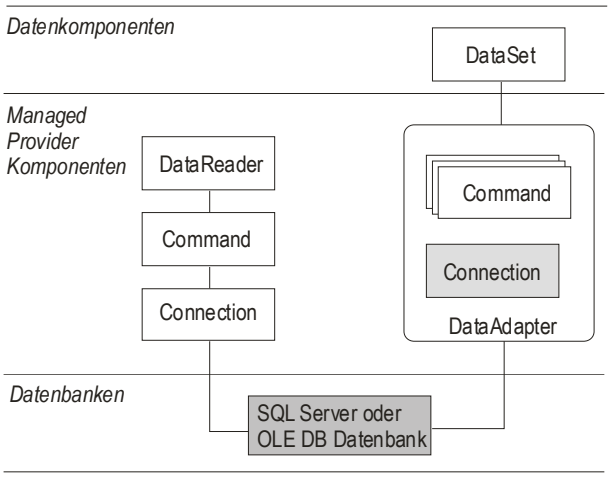

Abbildung 2:ADO.NET Architektur

ADO.NET verbessert den Vorgänger um Eigenschaften in den Bereichen Interoperabilität, Skalierbarkeit, Produktivität und Performanz. ADO.NET benutzt beispielsweise XML als Datenformat. Dies bedeutet, dass nicht mehr beide Enden einer Datenbankanwendung die COM Technologie verwenden müssen (aber deshalb wohl Microsoft Technologie?). Insbesondere auf Client Seite ist dies ein Vorteil (thin clients, PDAs etc.). Die Skalierbarkeit verbessert sich durch die sogenannten *disconnected data-sets*. Hierbei wird nicht mehr der Client Server Ansatz mit dauernd bestehenden Verbindungen genutzt, sondern diese nach Übertragung eines Datensatzes wieder getrennt. Dadurch wird unter anderem das Problem der maximal notwendigen Verbindungen zu einem Datenbankserver gelöst. Der Entwurf von ADO.NET Anwendungen wird vollständig im Visual Studio.NET unterstützt. Der Umgang mit den Inhalten und den Datenbanken wird durch verschiedene Bibliotheken erleichtert:

Inhaltskomponenten und -klassen

Die DataSet Komponente kapselt den Inhalt wie eine relationale Datenbank, d.h. das DataSet enthält Tabellen, Relationen etc. Allerdings werden die Daten im Hauptspeicher gehalten. Zur Synchronisation der Daten mit der Datenbank - z.B. bei Änderungen gibt es spezielle Methoden. Von der Semantik ist die DataSet Komponente also auf Web-Applikationen ausgerichtet.

DataSets unterstützen die Umwandlung in das XML-Format durch drei spezielle Methoden: `WriteXml`, `WriteXmlSchema` und `ReadXml`. Auf diese Wiese kann ein DataSet Objekt in XML serialisiert und wieder rekonstruiert werden.

Weitere Teile eines DataSets sind die Tabellen und Relationen. Durch Sichten (DataViews) können Filter und Sortierungen auf die Datensätze vorgegeben werden.

Managed Provider Komponenten

Die „managed provider“ Komponenten bilden das abstrakte Interface zu den Datenbanken. Standardmäßig sind bereits Komponenten für SQL und OLE Datenbankzugriff vorhanden. Diese bilden die Aufrufe an das abstrakte Interface effizient (?) auf die Datenbank durch Ausnutzung deren proprietären Protokolls ab. Diese Komponenten bilden also die Schicht zwischen den XML-basierten portablen DataSet Komponenten und der Datenbank selbst.

Für weitere Datenbanken können durch das vorgegebene Interface zusätzliche Implementierungen entwickelt werden. Anscheinend kann der Zugriff durch dieses Interface transparent erfolgen, also ohne besondere Eigenschaften der Datenbank dann berücksichtigen zu müssen[14].

XML Komponenten

In den ADO.NET Komponenten sind auch XML Parser und Writer integriert. Dabei steht sowohl die DOM Schnittstelle, als auch eine SAX ähnliches, ereignisorientiertes Parserinterface zur Verfügung. Die Transformation der Daten durch XST Dokumente wird ebenfalls unterstützt.

6. Webservices

Die .NET Plattform unterstützt das Erstellen von Webservices. Dabei kommt das Übertragungsprotokoll http und das SOAP Format zum Einsatz. Ein Webservice ist dabei eine Komponente, die auf einem Webserver ausgeführt wird und über diese Protokolle angesprochen werden kann[15].

> *Der Vorteil in der Verwendung von SOAP anstelle von HTTP POST und GET Methoden liegt darin, dass strukturierte Daten dargestellt werden können. Bei POST und GET (CGI Anwendungen) werden nur Paare der Form „name=value“ übertragen, ohne Typinformationen.*

Als Protokoll(!) kann allerdings immer HTTP POST und GET zum Einsatz kommen.

---

[14] Dies muss verifiziert werden. Der Text ist an der Stelle nicht ganz klar. Auf jeden Fall ist die Entwicklung einer solchen Schnittstelle dann sehr anspruchsvoll.

[15] Auf dem Microsoft IIS! Ein Webservice kann bei anderen Architekturen durchaus ohne einen Webserver als Host auskommen.

Namenskonventionen

In .NET wird bei der Entwicklung von Webservices ein Großteil automatisch generiert. Dabei werden Methoden und Beschreibungen nach einer Namenskonvention erzeugt. Zum Beispiel wird der Name einer SOAP Nachricht aus *Methodenname+Protokoll+(In|Out)* erzeugt. Eine Funktion „GetBooks“ kann also durch folgende SOAP Nachrichten repräsentiert sein:

<message name=“GetBooksSoapIn“> ….

<message name=”GetBooksSoapOut”>…

<message name=”GetBooksHttpGetIn”> …

Dieses Prinzip spiegelt sich in der gesamten WSDL Beschreibung eines Webservice wider. Die Kommunikationsendpunkte (SOAP Terminologie!) eines Webservice sind bei Microsoft durch „`.asmx`“ erkennbar. (D.h. die URL endet auf .asmx)

Webservices finden

Neben dem UDDI Standard zur Suche und Findung von Webservices unterstützt Microsoft einen eigenen Mechanismus zum Auffinden solcher Dienste. Dazu wird für eine Webservicebeschreibung (WSDL) eine XML Datei „.disco“ angelegt, die auf die WSDL zeigt und gleichzeitig auf eine weitere „.disco“ Datei. Damit kann sich ein Suchprozess durch die einzelnen Webservices eines oder mehrerer Server hangeln[16]. Ebenfalls kann eine dynamische Suche nach Webservices angegeben werden, die alle Webservices unterhalb einer bestimmten URL bestimmt.

Dieser Mechanismus entspricht leider nicht dem UDDI Standard, der ja einen speziellen Server vorschreibt. Andererseits ist so auf dem Server eine einfache Suchmöglichkeit realisierbar.

Webservices entwickeln

Die Entwicklung von Webservices wird durch eine Reihe von Klassen im .NET Framework unterstützt. Dabei kann der Webservice durch die Eigenschaften der .NET Architektur in jeder Programmiersprache entwickelt werden. Praktisch erbt jeder Webservice von der Basisklasse `Webservice`. Der Entwicklungsprozess gliedert sich in drei Schritte:

1. Erstellen einer `.asmx` Datei.
   Diese enthält eine Direktive `<% webservice ....%>`. Diese Datei ist der Startpunkt für Anwender des Dienstes.
2. Die neue Klasse von der Webservice Klasse `System.Web.Service` ableiten.

[16] Effizient ist das nicht (Anm d. Autors)

Auf diese Weise erhält die abgeleitete Klasse Zugriff auf alle ASP Objekte der Basisklasse

3. Alle Methoden, die als Webservice zur Verfügung gestellt werden sollen, erhalten eine `[WebMethod]` Direktive (Pragma) vor der Methodendefinition. Diese enthält bestimmte Attribute, die das Verhalten der Methoden bestimmen, oder zusätzliche Beschreibungen enthalten. Diese Methoden können dann als Funktionen eines Webservice angesprochen werden.

Webservices nutzen

Prinzipiell können .NET Webservices von beliebigen Konsumenten genutzt werden. Der Microsoft Webserver unterstützt praktischerweise einen web-basierten Konsumenten, der durch Zugriff auf das .asmx File zur Verfügung steht[17].

Die einfachste Form eines Konsumenten sind HTTP GET und POST Typen. Dabei wird der Webservice wie ein CGI Skript angesprochen und liefert als Antwort das Bearbeitungsergebnis (XML Schema und Daten) zurück. Das `wsdl` Programm generiert eine Proxy-Klasse mit dem ein Nutzer den Webservice bequem ansprechen kann, wahlweise in C#. Dabei kann das Protokoll angegeben werden, z.B. SOAP oder HTTP/GET.

Für den Umgang mit den XML Daten stehen Serialisierungsklassen und Transformationsmechanismen (über XSTL) bereit, die den Zugriff programmtechnisch sehr vereinfachen[18].

Sicherheitsaspekte von Webservices

Die .NET Webservice Implementierungen sind serverbasiert, d.h. die Sicherheitsaspekte der Benutzung von Webservices werden analog zu anderen Inhalten auf dem Webserver verwaltet.

Systemsicherheit

Zur Verifikation der Netzwerkadressen kann beispielsweise IPSec benutzt werden, das überprüft, ob die Netzwerkadresse im TCP/IP Header korrekt ist. Ebenso können Systeme wie Firewalls o.ä. eingesetzt werden.

Zur Zugangskontrolle der Benutzer bieten sich verschiedene Authentisierungsmethoden an[19]:

- *Basic Authentication:*
  Benutzername und Password werden im Klartext verschickt. Deshalb ist diese Methode nicht anwendbar für ernsthafte Anwendungen.

---

[17] Gut zum Testen des Webservice geeignet.

[18] In [Thai01] sind dazu Beispiellistings. Ebenfalls ist beschrieben, wie man nicht-.NET Konsumenten entwickelt.

[19] Die Authentifizierungsmethoden stützen sich auf die Funktionalität des Servers der die Webservices anbietet ab.

- *Basic over SSL Authentication:*
  wie Basic Authentication nur in diesem Fall werden Benutzername und Passwort verschlüsselt übertragen.
- *Digest Authentication:*
  Benutzt Hashalgorithmen zur Übertragung der Beglaubigung an den Server.
- *Integrated Windows Authentication:*
  Benutzt Windows Login Informationen des Clients. Ist nur in Intranetszenarien sinnvoll.
- *Client Certificates Authentication:*
  Jede Client-Anwendung erhält ein Zertifikat, welches einem Benutzer zugeordnet ist. (Wenig verbreitet)

Sicherheit in der Webservice-Anwendung

Alternativ kann eine eigene Sicherheitsstrategie in den Funktionen des Webservice selbst implementiert werden - zum Beispiel die Verwaltung eigener Access-Tokens. Dabei muss dieses Token dann beim Aufruf eines Webservice mit übergeben werden. Die Anwendung verschiedener Verschlüsselungstechnologien wie DES, RC2, 3DES und RSA sind durch .NET Framework Klassen unterstützt.

7. Web Formulare in .NET (Web Oberflächen)

.NET unterstützt nicht nur die Erstellung von Windows Applikationen sondern auch die Erstellung von Web-Applikationen:

Die .NET Technologie zur Erstellung dynamischer, interaktiver Webseiten heißt ASP.NET, die Weiterentwicklung von ASP. ASP ist Microsofts serverseitige Skripting Technik[20] mit einem eigenen Objektmodell für Sessions, Responses und Requests. Zusätzlich kann auf die COM Komponenten auf dem Server zugegriffen werden. ASP.NET adressiert die Problemfelder von ASP: unstrukturiert, keine Typsicherheit.

Unter ASP.NET können Webseiten wesentlich einfacher entwickelt werden, ähnlich einer „normalen“ Applikation. Die Formulare „Web-Forms“ bilden ein neues Programmiermodell:

- Trennung der Anwendungslogik von der Präsentationsschicht
- Server-Controls (UI-Elemente), die ihre HTML Darstellung rendern (damit für alle Browser nutzbar, andere Darstellungen sind ebenfalls denkbar, z.B. durch einen XML Renderer) und den eigenen Zustand verwalten[21]. Im einfachsten Fall von Standard-Webseiten wird für das Eingabeele-

[20] Analog zu Java Server Pages.

[21] Die einzelnen Controls und deren Programmierung ist im Buch ausführlich erläutert und wird hier nicht vertieft.

ment beim Aufruf vom Server-Control der entsprechenden HTML Code erzeugt.

- Ereignisbasiertes Programmiermodell auf der Serverseite (Events, ähnlich zu „normalen“ grafischen Applikationen)
- Die Anwendungslogik im Server kann in einer beliebigen .NET Programmiersprache implementiert werden.
- Rapid Development mit dem Visual Studio .NET.

Die Entwicklung von Web Form Komponenten erstreckt sich über zwei Teile: Dem eigentlichen Formularobjekt (Control) und dessen Programmcode zur Ereignisbehandlung. Dies trennt die Benutzungsschnittstelle von der Anwendungs- bzw. Geschäftslogik.

Die vorhandenen ASP.NET Kontrollelemente können durch Vererbung und Aggregation erweitert werden, beispielsweise um komplexere Eingabemechanismen oder komplexe Daten zu unterstützen.

Wie bereits in ASP unterstützt ASP.NET eine einfache Programmierung von Datenzugriffen auf Datenbanken[22].

8. Windows Formulare (Windows Oberflächen)

Für die Oberflächengestaltung steht ein einheitliches Programmiermodell mit einer neuen Komponentenbibliothek zur Verfügung: Den „Windows Forms“. Alle Programmiersprachen nutzen dabei die gleichen Komponenten. Damit wird die bekannte MFC Bibliothek ersetzt und durch ein einheitliches Objektmodell abgelöst[23]. Die Zusammenarbeit mit anderen Komponenten des Frameworks wie Webservices, ADO.NET usw. funktioniert ohne Einschränkungen.

Programmierung von Windows Oberflächen unter .NET

Alle grafischen Elemente werden von der Klasse `System.Windows.Forms` abgeleitet[24]. Dabei ist die Architektur einfach gehalten. Es gibt zwei zentrale Basisklassen: *Controls* und *Container.* Die grafischen Elemente leiten sich alle von den Controls ab, während Fenster, Paneele etc. typische Container sind. Die Anbindung von Event-Prozeduren an Controls ist unter .NET wesentlich vereinfacht. Die Handler können sowohl statisch als auch dynamisch zur Laufzeit den GUI Elementen zugeordnet werden.

---

[22] Wird an dieser Stelle nicht weiter vertieft.

[23] Die Windows Programmierung - insbesondere unter C# - ähnelt von der Logik stark dem JAVA Programmiermodell.

[24] Die Struktur ist nahezu identisch zu den Web-Forms!

9. Zusammenfassung

Das .NET Framework ist ein großer Schritt in Richtung eines flexiblen Komponentensystems von Microsoft. Die Ecken und Kanten in den alten Ansätzen wurden systematisch entfernt und wirklich eine neue einheitliche Architektur geschaffen, die auch die Entwicklung wesentlich transparenter macht. Webzentrierte und normale Applikationsentwicklung konvergieren durch gemeinsame Bibliotheken und die Zuwendung zu standardisierten Protokollen.

10. Literatur

[Thai01] Thuan Thai, Hoan Q. Lam; *.NET Framework Essentials*. O'Reilly & Associates, 2001.

[ProfC#] *Professional C#*. Wrox Press. 2001

Cooperative Computing &
Communication Laboratory

C-LAB
Marketing
Fürstenallee 11
D-33102 Paderborn

Telephone +49-5251-60-6060
Telefax +49-5251-60-6066
E-Mail marketing@c-lab.de
URL http://www.c-lab.de

Befragung über Ihre Zufriedenheit mit dem Report *„.NET Framework Kompakt“*

Wir bitten Sie, sich einen kurzen Moment Zeit zu nehmen, um uns ein paar Fragen über Ihre Einschätzung dieses Reports zu beantworten. Damit helfen Sie uns, Ihre Bedürfnisse besser zu verstehen. Wir möchten unsere Reports stärker nach Ihrem Interesse ausrichten, um so einen größeren Mehrwert bieten zu können. Vielen Dank für Ihre Mitarbeit.

Wie bewerten Sie das Thema dieses Reports?

	trifft voll zu			trifft überhaupt nicht zu
Aktuell	☐	☐	☐	☐
Interessant	☐	☐	☐	☐

Wie bewerten Sie den Inhalt dieses Reports?

	trifft voll zu			trifft überhaupt nicht zu
Aktuell	☐	☐	☐	☐
Interessant	☐	☐	☐	☐
Verständlich	☐	☐	☐	☐
Praxisrelevant	☐	☐	☐	☐
Informativ	☐	☐	☐	☐
Innovativ	☐	☐	☐	☐

Weitere Kommentare:

_____
_____
_____
_____

Freiwillige Angaben:
Name, Vorname: _____
Telefon: _____
E-Mail: _____

Bitte senden Sie das ausgefüllte Formular per Post, Fax oder E-Mail an die jeweilige Adresse (s. o.).

Kooperation
Siemens Business Services GmbH & Co. OHG
Universität Paderborn

Universität Paderborn
Prof. Dr. Franz-Josef Rammig
Telephone ++49-5251-606062

Siemens Business Services GmbH & Co. OHG
Dr. Wolfgang Kern
Telephone ++49-5251-606061