Praktische Einführung in *.NET*

Hauptseminar Tel SS2002

Kai Stammerjohann

1 Common Language Runtime Eigenschaften

Die Common Language Runtime[1] ist ein wesentlicher Bestandteil des .NET Frameworks. Jeglicher Code der Teil einer .NET Anwendung ist wird ausschließlich in ihr ausgeführt. Aus diesem Grund nimmt sie eine zentrale Rolle im gesamten Framework ein.
Dieses Kapitel soll einen Überblick über einige erwähnenswerte Common Language Runtime Eigenschaften geben. Mit erwähnenswert ist gemeint, dass weder auf die wichtigsten, noch auf alle Eigenschaften eingegangen wird. Vielmehr geht es um einige Eigenschaften, die das .NET Framework von ähnlichen Frameworks[2] unterscheidet.

1.1 CLR Struktur

Zuerst wird der Anwendungsquellcode von einem .NET Compiler in eine Zwischensprache, der so genannten Microsoft Intermediate Language[3] übersetzt. Da diese Zwischensprache nicht sofort ausführbar ist, muss sie zum Ausführen noch in ausführbaren Maschinencode übersetzt werden. Diese Aufgabe wird von der CLR während der Laufzeit übernommen. Das Übersetzen des MSIL Codes geschieht Just In Time. Just In Time bedeutet, dass nur MSIL Code übersetzt wird, der unmittelbar zum Ausführen der Anwendung benötigt wird. Es kann also durchaus vorkommen, dass nicht benutzter MSIL Code niemals übersetzt werden muss. Diese Vorgehensweise ermöglicht eine vollständige platformunabhängigkeit da im MSIL Code keine platformspezifischen Teile vorkommen können[4].
Es gibt bereits einige Programmiersprachen mit einem MSIL fähigen Compiler. Neben den .NET Standartsprachen C#, Visual Basic und C++ gibt es von Drittanbietern noch Java ([VJAVA]) und Forth ([DFORTH]), wobei die Anzahl zurzeit immer noch relativ schnell zunimmt.
.NET Compiler erzeugen beim Übersetzen des Quellcodes so genannte Assemblies. Assemblies sind kleine Archive die neben dem zugehörigen MSIL Code noch Funktionalität zur Wahrung der Sicherheit und der Konsistenz bieten. Zum einen werden Assemblies signiert und sind dadurch gegen Veränderung geschützt, zum anderen enthalten sie

[1] Common Language Runtime = CLR

[2] .NET ähnliches Framework ist z.B. Java ONE ([ONE])

[3] Microsoft Intermediate Language = MSIL

[4] platformspezifisch wird es erst zur Laufzeit, wenn der MSIL Code von der CLR in Maschinencode übersetzt wird

Versionisierungsinformationen, die verhindern, dass Probleme bei der Zusammenarbeit von Komponenten mit verschiedenen Versionen auftreten.

Abbildung 1.1 stellt das Zusammenspiel zwischen Quellcode, MSIL und der Ausführung in der CLR dar.

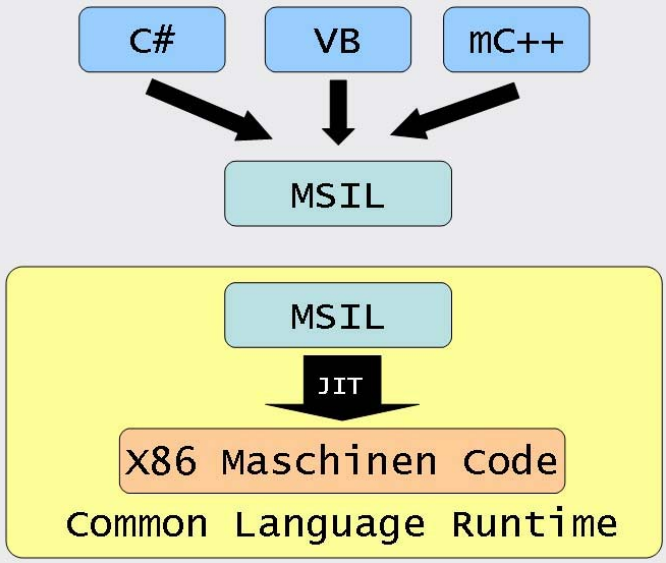

Abbildung 1.1

1.2 Sprachübergreifende Programmierung

Wie bereits im vergangenen Kapitel erwähnt wurde, wird aus allen Quellsprachen durch den passenden .NET Compiler MSIL Code erzeugt. Anders Ausgedrückt werden die verschiedenen Sprachen mit ihren individuellen Sprachmerkmalen auf den gemeinsamen MSIL Code[5] abgebildet. Auf diese Weise ist es möglich, dass Komponenten aus verschiedenen Sprachen auf Quellcodeebene miteinander interagieren.

Listing 1.1 zeigt wie eine C# Klasse in einer C++ Anwendung benutzt werden kann. Die Verbindung zwischen C# Klasse und C++ Anwendung geschieht durch die

#using "CSKlasse.dll"

Direktive im C++ Quellcode. Dadurch erhält der Compiler Informationen über den Aufbau der C# Klasse und kann sie so benutzen, als wäre es eine herkömmliche C++ Klasse. Daraus lässt sich folgern, dass in der C# Klassen Library die dazu notwendigen Informationen über den Aufbau enthalten sein müssen.

[5] ermöglicht durch gemeinsames Typsystem aller .NET Sprachen

Ein kleiner aber interessanter Nachteil dieser Vorgehensweise ist, dass sich .NET Assemblies nahezu vollständig in ihren zugehörigen Quellcode zurückübersetzen ([AKRNO]) lassen[6]. Dieser Nachteil tritt aber nicht erst mit .NET Sprachen zum Vorschein, sondern ist auch ein Problem bei Java ([JAD]) oder Visual Basic.
Lösung für dieses Problem sind so genannte Obfuscation Engines, die versuchen, den Sourcecode so schwer verständlich zu machen wie möglich[7].

[6] es ist sogar möglich eine C# Assembly in C++ Quellcode zurückzuübersetzen

[7] durch Umbenennung von Variablen, Funktionen, …

2 Datenbankzugriff

Einer der wichtigsten Prozesse bei Anwendungen, speziell für das Internet, ist der Umgang mit Datenbanken. .NET bietet viele verschiedene Mittel um einen passenden Datenbankzugriff zu gewährleisten.
Dieses Kapitel beschreibt verschiedene Zugriffsmöglichkeiten. Es sind nicht immer alle Verfahren in allen Situationen gleich geeignet. Aus diesem Grund wird bei den betrachteten Möglichkeiten auf Vor- und Nachteile eingegangen.

2.1 Herkömmlicher Datenbankzugriff

Ein Nachteil beim herkömmlichen Zugriff auf die Daten einer Datenbank ist, dass man sich relativ ‚freiwillig' entscheidet, als was für einen Typ man die Daten behandelt. Der Zugriff auf die Daten erfolgt also untypisiert[8]. Typischer Code für das Auslesen einer Spalte/Zeile sieht so aus:

reader.GetString(0)

Ein anderer (nicht weiter problematischer) Nachteil ist, dass der Zugriff auf die Daten nur per Spaltenindex möglich ist, was den Code nicht unbedingt verständlicher macht.
Diese Art des Datenbankzugriffs hat aber nicht nur Nachteile. Ein wichtiger Vorteil ist die Performance. Es werden weder aufwendige serverseitige Cursor gebraucht, noch werden Transaktionen[9] benutzt. Des weiteren werden immer nur die Daten aus der Datenbank gelesen, die aktuell gebraucht werden. Diese Zugrifftechnik eignet sich überall da, wo schneller, nur-lesender Datenbankzugriff notwendig ist. Ein typisches Anwendungsbeispiel sind Internetanwendungen, wo ständig auf viele kleine Datensätze lesend zugegriffen werden muss.

2.2 ADO[10].NET Datenbankzugriff

Einige Nachteile des herkömmlichen Datenbankzugriffs werden durch ADO.NET behoben, allerdings eignet es sich nicht in jeder Situation.

---

[8] bezogen auf die freiwillige Entscheidung für den Datentyp

[9] Zugriff ist nur-lesend

[10] ActiveX Data Object

Der Grundgedanke von ADO.NET ist, dass zuerst das Datenbankschema beschrieben wird um daraus eine auf die Datenbank spezialisierte Zugriffsklasse[11] zu erzeugen.
Die Datenbankschemabeschreibung erfolgt mittels XML[12]. Sie enthält alle benutzten Tabellen, die Spalten der Tabellen und die Typen der Spalten.
Zusätzlich können noch Relationen zwischen Tabellen in diesem XML Schema untergebracht werden.
Die so erzeugte Datenbankbeschreibung dient als Vorlage für das DataSet. Dieses DataSet bietet somit eine abstraktere Zugriffsmöglichkeit auf die Daten. Sie enthält für jede Tabelle einen zur jeweiligen Tabellensignatur passenden Datentyp[13]. Mit Hilfe dieser Datentypen ist ein typisierter Zugriff möglich. Für jede Spalte eines Datenbankrecords enthält der spezialisierte Datentyp eine Variable, die den Namen und den Datentyp der Spalte hat. Ein typischer Zugriff auf eine solche Datenstruktur wird in **Listing 2.1** gezeigt.
Die Daten müssen vor dem Zugriff vom Datenbankserver lokal in der spezialisierten Datenstruktur abgespeichert werden. Danach wird die Verbindung zum Server nicht mehr benötigt, die Daten können offline bearbeitet werden. Für schreibenden Zugriff auf die lokalen Daten unterstützt das DataSet Updates, allerdings muss vorher eine Verbindung zur Datenbank hergestellt werden.
Der Nachteil von DataSets ist die Tatsache, dass vor dem Zugriff erst alle betreffenden Records aus der Datenbank in das DataSet kopiert werden müssen. Aus diesem Grund ist der Zugriff nicht besonders schnell[14]. Es gibt allerdings auch Szenarien wo gerade dieser Offlinezugriff Vorteile hat: Windowsanwendungen, die dem Anwender accessähnlichen (lesenden und schreibenenden) Zugriff auf die Daten bieten.

2.3 XML / XSL

Die letzte vorgestellte Zugriffsweise nutzt die Vorteile einer XSL Transformation ([XSLT]). XSL Transformationen werden überall da eingesetzt, wo XML Bäume in andere XML Bäume überführt (transformiert) werden müssen.

[11] DataSet, ähnlich zu Enterprise Java Beans ([EJB]) oder MFC RecordSets ([MFCRS])

[12] typischerweise in Form einer .XSD Datei

[13] Klasse, die Tabellendeklaration repräsentiert (= Row)

[14] es gibt schnellere Verfahren, siehe 2.1

Grundlage für diese Transformation ist eine XSL Datei, die beschreibt wie die einzelnen Blättern und Knoten des Quell-XML-Baumes in entsprechende Blätter bzw. Knoten des Ziel-XML-Baumes überführt werden sollen. Typischerweise enthält der Ziel-XML-Baum nur HTML Elemente als Knoten bzw. Blätter. Auf diese Weise sind die Zieldaten ohne Umwege sofort in clientseitigen Browsern darstellbar. Der Vorteil einer XSL Transformation ist die nahezu perfekte Trennung von Daten und Darstellung[15].
.NET bietet für diesen Zweck Zugriffsklassen, die Datensätze aus Datenbanken in XML Dokumente überführen können. Neben diesen XML Zugriffsklassen existiert auch ein XSL Prozessor, der die XSL Transformation ausführen kann. **Abbildung 2.1** stellt diesen Prozess dar und in **Listing 2.2** wird gezeigt, wie eine solche Transformation in C# realisiert werden kann.

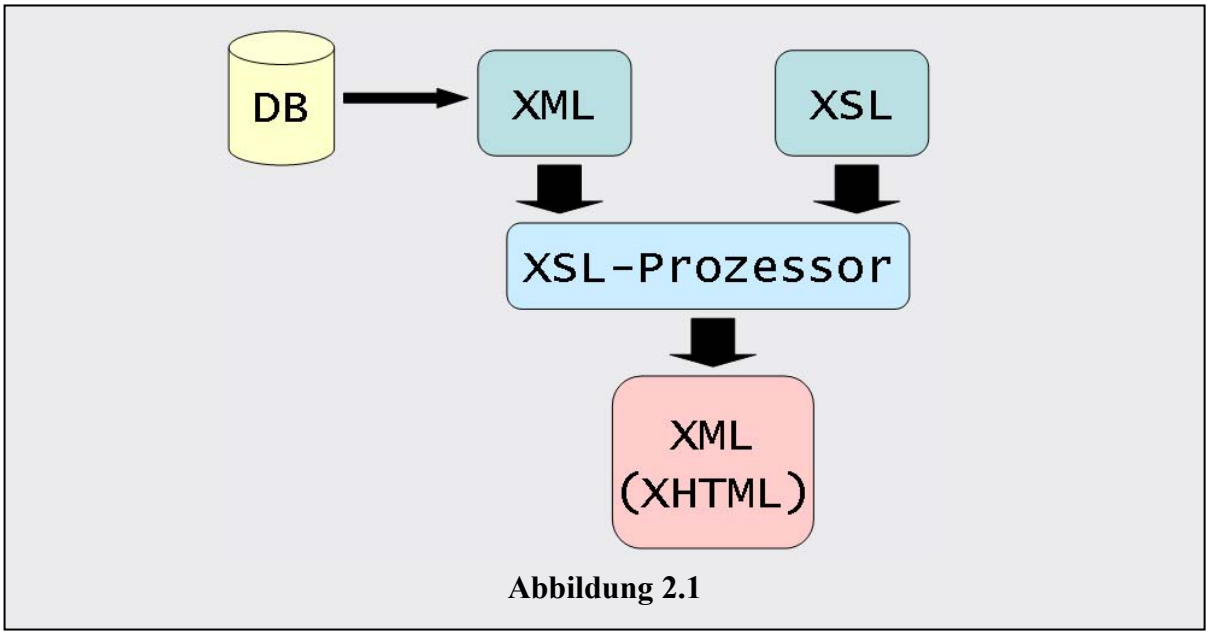

Abbildung 2.1

[15] siehe Document / View Modell

3 Web Service

Dieses Kapitel beschreibt wie Web Service Server programmiert werden, wie sie nach außen aussehen, wie mit ihnen Kommuniziert wird und wie sie in Anwendungen benutzt werden können.

3.1 Definition

Der Begriff Webservice ist keine Microsoft Erfindung, vielmehr wurde er zu einem früheren Zeitpunkt von SUN festgelegt.
SUN's Definition von Webservice besagt, dass ein Webservice ein Dienst sein muss, der über eine XML Schnittstelle benutzt werden kann. Durch wen[16] der Dienst benutzt wird soll dabei keine Rolle spielen.
Als Kommunikationsprotokoll soll dabei SOAP ([SOAP]) benutzt werden. Als Transportprotokoll können verschiedene Protokolle benutzt werden. .NET unterstützt unter anderem HTTP als Transportprotokoll für die SOAP Kommunikation.
Gewöhnlicherweise weiß ein Web Service Anwender nicht, wo (physikalisch) der Service läuft. Damit er ihn trotzdem finden kann, gibt es die so genannte UDDI. Die UDDI ist ein zentrales und standardisiertes Web Service Verzeichnis. Web Services können sich dort registrieren und Anwender können dort Dienste finden. Die UDDI stellt also einen Vermittler[17] zwischen Diensterbringer und Dienstnutzer dar.

3.2 Server

Einen Web Service Server zu schreiben ist denkbar einfach. Dazu muss eine Klasse von der Web Service Basisklasse abgeleitet werden. Außerdem muss jeder Methode die veröffentlicht[18] werden soll noch ein spezielles *[WebMethod]* Attribut[19] gegeben werden.
Listing 3.1 enthält eine komplette Web Service Server Klasse.

[16] Benutzung durch Programm oder Anwender

[17] Vgl. RMI Registry, CORBA ORB

[18] Methoden, auf die per SOAP vom Client zugegriffen werden kann

[19] Attribute sind spezielle Eigenschaften (vgl. public, protected, …), die Codefragmente(Variablen, Funktionen) bekommen können, z.B. Obsolete Attribut

Web Services können als Parameter oder Rückgabetypen neben Standardtypen wie int, string und float auch eigene komplexe Datentypen besitzen.
Das .NET Framework stellt zu jedem Dienst automatisch eine Art Testfrontend in Form einer Webseite zur Verfügung, über die der Dienst getestet werden kann. Über diese Seite können die verschiedenen Funktionen (inklusive ihrer Parameter) des Dienstes aufgerufen werden. Das Ergebnis eines Aufrufs wird ebenfalls durch die Seite dargestellt.
Damit Anwender einen Dienst finden können, kann er sich in einem zentralen Dienstverzeichnis, der UDDI ([UDDI]) registrieren.

3.3 Dienstbeschreibung

Damit verschiedene Clients einen Dienst nutzen können, benötigen sie genaue Informationen über seinen Aufbau. Das wird mit dem Dienstbeschreibungsformat WSDL[20] realisiert. Zu jedem Dienst existiert ein solches Beschreibungsdokument. Beschrieben werden die Funktionen, die der Dienst bereitstellt, wie viele und welche Parameter sie entgegennehmen und welchen Datentypen für Rückgabe oder Parameter benutzt werden.
Da auch eigene komplexe Datentypen für die Kommunikation eingesetzt werden können, müssen diese Datentypen auch explizit beschrieben werden. Dies geschieht, indem alle öffentlichen Membervariablen der komplexen Datentypen aufgezählt werden.

3.4 Client

Basierend auf der Dienstbeschreibung[21] ist es möglich (vollautomatisch) eine Dienstproxyklasse zu generieren. Mit Hilfe dieser Proxyklasse ist es möglich auf den Dienst zuzugreifen und seine Funktionen zu benutzen. Die Proxyklasse verpackt dazu die Dienstaufrufe des Clients in SOAP Pakete und übergibt sie dem Server. Der Server wiederum verpackt die Antwort in SOAP und sendet sie zu dem Client zurück.
Da die komplette SOAP Kommunikation nur zwischen Proxyklasse und Web Service stattfindet bekommt der eigentliche Dienstclient davon nichts mit. **Abbildung 3.1** illustriert diesen Mechanismus. **Listing 3.1** zeigt den entsprechenden Code dazu.

[20] Web Service Description Language (XML)

[21] sieht Kap. 3.3

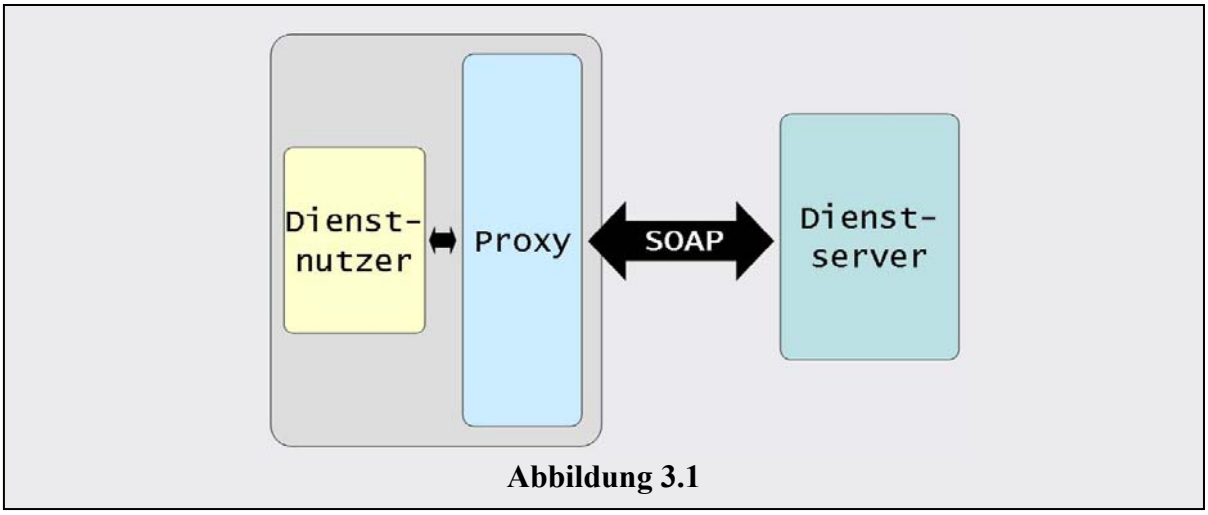

Abbildung 3.1

4 Webforms

Webforms sind für Clients normale Webseiten. Sie können alle möglichen HTML Steuerelemente wie Buttons, Links oder Inputs enthalten. Dieses Kapitel beschäftigt sich mit dem Aufbau von Webforms. Dabei wird auf deren Programmierung und Verwendung eingegangen. Zusätzlich wird noch kurz auf Validatoren eingegangen.

4.1 Webcontrol

Webcontrols sind normale HTML Steuerelemente. Beispiele für solche Steuerelemente wären <INPUT> oder <BUTTON>. Jedes Steuerelement wird durch eine .NET Klasse repräsentiert[22]. Zum <INPUT> HTML Control gehört beispielsweise die *System.Web.UI.WebControls.TextBox* Klasse.

4.2 Aufbau

Zu jedem Webform gibt es eine zugehörige Webform Klasse. Diese Klasse wird parallel zum (WYSIWYG[23]) Entwurf des Webforms erzeugt. Für jedes Control, was in der Seite enthalten ist bekommt die zugehörige Webform Klasse eine Membervariable mit dem entsprechenden Controlklassentyp zugeordnet. Somit ist auf die komplette Seite objektorientiert zugreifbar, alle Eigenschaften[24] aller enthaltenen Steuerelemente sind programmierbar. **Listing 4.1** enthält den Webform Klassen Code einer einfachen Seite, die die aktuelle Zeit innerhalb eines <INPUT> Controls anzeigt.

4.3 Benutzerinteraktion

Jedes Control kennt verschiedene Ereignisse. Webforms können Ereignisbehandlungsmethoden für diese Ereignisse registrieren. Auf diese Weise ist es für Webforms möglich auf Benutzereingaben zu reagieren. Beispiel wäre ein OnClick Ereignis von einem <BUTTON> Control. Im OnClick - Ereignisbehandlungscode kann dann auf verschiedene Eingaben reagiert werden.

---

[22] genau genommen rendert die Klasse das Control

[23] What You See Is What You Get – grafischer Entwurf, direktes visuelles Feedback

[24] Beispielsweise Texte, Positionen, Sichtbarkeit, Aussehen

4.4 Validatoren

Überall wo Benutzer Daten eingeben müssen können falsche Eingaben entstehen. .NET bietet aus diesem Grund so genannte Validatoren. Diese Validatoren können prüfen, ob sich die Eingaben an spezielle Regeln hält. Dazu werden unter anderem reguläre Ausdrücke benutzt: matched[25] der reguläre Ausdruck die Eingabe eines Controls sind die Daten korrekt, andernfalls wird ein vorher festgelegter Fehlertext bei dem Eingabecontrol angezeigt. Auf diese Weise ist es sehr einfach möglich, falsche Eingaben zu verhindern. Die Plausibilitätsprüfung ist somit nicht explizit vom Webform zu erledigen.
Grundsätzlich gibt es 2 verschiedene Möglichkeiten die Eingabedaten zu prüfen. Die erste Variante ist Clientseitig. Dabei wird eine relativ komplexe JavaScript Funktion erzeugt, die im Fehlerfall die Eingaben nicht zum Server schickt. Die zweite Variante ist Serverseitig. Hier wird die Validation der Eingaben vom Server übernommen.

[25] Aufbau des zu prüfenden Eingabestrings muss den Regeln des reguläre Ausdrucks entsprechen

5 Schlussfolgerung

Mit dem .NET Framework in Verbindung mit dem Visual Studio 7 ist eine sehr schnelle Entwicklung von komplexen Lösungen möglich. Diese Tatsache wurde auch von einem unabhängigen Testlabor bestätigt ([NETPS]). Dazu wurde SUN's J2EE Referenzanwendung „PetStore" ([JPS]) mit .NET nachgebaut. Ergebnis des Tests war, dass für die J2EE Lösung ein Viertel mehr Code nötig war. Zusätzlich war die Laufzeitperformance der J2EE Lösung dem .NET Nachbau unterlegen.

6 Quellcode

C#:

```
public class CSKlasse
{
        public void hallo(string text)
        {
                System.Console.WriteLine(text);
        }
}
```

mC++:

```
#using "CSKlasse.dll"                       //einbinden der C# Bibliothek
int main(void)
{
        CSKlasse *obj = new CSKlasse();     //Instanziieren der C# Klasse

        obj->hallo("test");                 //aufrufen einer C# Memberfunktion

        return 0;      //kein delete notwendig, wird vom C++ GarbageCollector gemacht
}
```

Listing 1.1

Benutzung einer C# Klasse in C++ Code

```
SqlConnection conn = new SqlConnection(" … ");      //Verbindung zur Datenbank
SqlDataAdapter da= new SqlDataAdapter("SELECT * FROM hauptseminare", conn);
EinschreiberDataSet ds = new EinschreiberDataSet(); //auf DB spezialisierte Zugriffsklasse
                                                    //erzeugt aus Datenbankschema

conn.Open();
da.Fill(ds, "hauptseminare");            //kopieren der Daten aus DB in lokales DataSet
conn.Close();


foreach(EinschreiberDataSet.hauptseminareRow row in ds.hauptseminare)
{
       Console.WriteLine("{0}, {1}", row.titel, row.inhalt);      //direkter Zugriff auf
                                                                  //die Spalten
}
```

Listing 2.1

typisierter ADO.NET Datenbankzugriff

```
SqlConnection conn = new SqlConnection(" … ");
SqlDataAdapter da= new SqlDataAdapter("SELECT * FROM hauptseminare", conn);
EinschreiberDataSet ds = new EinschreiberDataSet();

conn.Open();
da.Fill(ds, "hauptseminare");
conn.Close();

XmlDataDocument xml = new XmlDataDocument(ds);      //XML Dokument aus DataSet erzeugen
XslTransform xsl = new XslTransform();              //XSL Prozessor erzeugen
XmlTextWriter writer = new XmlTextWriter("output.html");   //Ergebnis von XSLT in Datei


xsl.Load("format.xsl");
xsl.Transform(xml, null, writer);
writer.Close();
```

Listing 2.2

XML Datenbankzugriff und XSL Transformation

```
public class Service1 : System.Web.Services.WebService     //Web Service Basisklasse
{
        [WebMethod]    //markiert Methode für öffentlichen Zugriff
        public string hallo(string t1, string t2)
        {
                return "server meint zu " + t1 + " das da " + t2;
        }
}
```

Listing 3.1

Web Service Server

```
localhost.Service1 svc = new localhost.Service1();  //neue Service Proxy Instanz erzeugen
Console.WriteLine(svc.hallo("1", "2"));             //Service Funktion auf Proxy ausführen
```

Listing 3.2

Web Service Client

```
public class Index : System.Web.UI.Page     //Basisklasse für Webforms
{
        protected System.Web.UI.WebControls.TextBox TextBox1;      //Variable für <INPUT>

        private void Page_Load(object sender, System.EventArgs e)  //wird beim laden der
                                                                   //Seite aufgerufen
        {
                TextBox1.Text = System.DateTime.Now.ToString();    //Text Eigenschaft des
                                                                   //Controls mit aktuelle
                                                                   //Zeit String
                                                                   //überschreiben

        }
}
```

Listing 4.1

Webform Klasse

7 Ressourcen

[VJAVA] Microsoft Visual J# Beta 2
http://search.microsoft.com/gomsuri.asp?n=1&c=rp_Results&siteid=us/dev&target=http://msdn.microsoft.com/msdn-files/027/001/898/Search.asp

[DFORTH] Delta Forth .NET - World's first Forth compiler for the .NET platform
http://www.codeproject.com/dotnet/dforthnet.asp

[ONE] Sun[tm] Open Net Environment
http://wwws.sun.com/software/sunone/

[AKRNO] Anakrino
http://www.saurik.com/net/exemplar/

[JAD] Jad - the fast JAva Decompiler
http://kpdus.tripod.com/jad.html

[EJB] Enterprise Java Beans
http://java.sun.com/products/ejb/index.html

[MFCRS] Record Sets
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_MFC_CRecordset.asp

[XSLT] *XML Bible, Second Edition* : XSL Transformations
http://www.ibiblio.org/xml/books/bible2/chapters/ch17.html

[SOAP] Simple Object Access Protocol (SOAP) 1.1
http://www.w3.org/TR/SOAP/

[UDDI] Universal Descritption, Discovery and Integration
http://www.uddi.org/

[NETPS] Microsoft .NET Pet Shop
http://www.gotdotnet.com/team/compare/petshop.aspx

[JPS] Java Pet Store Sample Application
http://java.sun.com/blueprints/code/index.html#java_pet_store_demo