

1 Was ist .NET?

Im Jahr 2002 brachte Microsoft nach mehrjähriger Forschung und Entwicklung die .NET-Technologie auf den Markt. .NET ist kein Betriebssystem im engeren Sinne und somit auch kein Nachfolger von Windows. Es handelt sich vielmehr um eine Schicht, die auf Windows (und später vielleicht auch auf anderen Betriebssystemen) aufsetzt und vor allem folgende zwei Dinge hinzufügt:

- Eine **Laufzeitumgebung**, die automatische Speicherbereinigung (*garbage collection*), Sicherheitsmechanismen, Versionierung und vor allem Interoperabilität zwischen verschiedenen Programmiersprachen bietet.
- Eine **objektorientierte Klassenbibliothek** mit umfangreichen Funktionen für grafische Benutzeroberflächen (*Windows Forms*), Web-Oberflächen (*Web Forms*), Datenbankanschluss (*ADO.NET*), Collection-Klassen, Threads, Reflection und vieles mehr. Sie ersetzt in vielen Fällen das bisherige Windows-API und geht weit über dieses hinaus.

.NET ist aber mehr als das. Es ist eine offene Plattform, mit der Microsoft versucht, die verschiedenen Strömungen der Softwareentwicklung, die sich in den letzten Jahren voneinander entfernt haben, wieder zusammenzuführen und damit ihre Kunden wieder mit einer einheitlichen Technologie zu bedienen.

- Internetanwendungen (z.B. Web-Shops) wurden bisher mit anderen Techniken entwickelt als lokale PC-Anwendungen. Während man am PC mit kompilierten Sprachen wie C++ oder Pascal programmierte und objektorientierte Klassenbibliotheken und Frameworks verwendete, arbeitete man in der Web-Programmierung mit HTML, ASP, CGI sowie mit interpretierten Sprachen wie JavaScript oder PHP. Unter .NET können beide Arten von Anwendungen mit denselben Techniken programmiert werden, z.B. mit kompilierten Sprachen wie C# oder Visual Basic .NET sowie mit einer reichhaltigen objektorientierten Klassenbibliothek.
- Unternehmen haben in den letzten Jahren viel Geld in Software investiert, die in unterschiedlichen Sprachen wie C++, Visual Basic oder Fortran entwickelt wurde. Sie haben wenig Interesse, diese Investitionen in den Wind zu schreiben. Vielmehr möchten sie, dass Programme, die in verschiedenen Sprachen geschrieben wurden, reibungslos zusammenarbeiten können.

.NET ermöglicht das durch eine bisher beispiellose Art der Interoperabilität.

- In letzter Zeit sind Kleinstcomputer wie Handhelds, Palmtops oder Mikrocontroller (*embedded systems*) im Vormarsch. Auch für sie gab es bisher spezielle Sprachen und Betriebssysteme. Unter .NET können sie mit denselben Sprachen und Bibliotheken programmiert werden wie PCs oder Webserver. Dadurch rückt die Softwareentwicklung für mobile und eingebettete Anwendungen näher zur konventionellen Programmierung.

Ein Bereich, der in der bisherigen Softwareentwicklung eine eher untergeordnete Rolle spielte, aber in Zukunft große Bedeutung erlangen wird, sind verteilte Systeme, die über das Internet zusammenarbeiten, um Aufgaben zu erfüllen, die lokal nicht zu lösen sind. Solche Systeme werden heute immer häufiger durch so genannte *Web-Services* implementiert, eine durch Nachrichtenaustausch realisierte Art von Diensten, die über XML und Protokolle wie HTTP zusammenarbeiten. Web-Services werden in .NET besonders gut unterstützt.

.NET stellt auch eine Reihe von Werkzeugen zur Verfügung, allen voran Visual Studio .NET, eine mehrsprachenfähige Softwareentwicklungsumgebung mit Debugger und GUI-Designer, die optimal auf .NET-Aufgaben wie das Erstellen von Web Forms oder Web-Services ausgerichtet ist.

Zu .NET im weiteren Sinne zählt Microsoft schließlich auch verschiedene Server, wie den *Windows Server 2003*, den Microsoft *SQL Server*TM [MSSQL] oder den Microsoft *BizTalk*TM Server [MSBiz].

Was ist nun also .NET? Es ist ein aufeinander abgestimmtes Ensemble aus Betriebssystemkomponenten, Bibliotheken, Werkzeugen, Web-Services und Servern, das schwer in einem einzigen Satz zu definieren ist. Alles zusammen hat zum Ziel, die Programmierung von Windows- und Web-Anwendungen bequemer und einheitlicher zu gestalten. Eines ist jedoch klar: Die Softwareentwicklung unter .NET unterscheidet sich deutlich von der bisherigen Windows- und Web-Programmierung. Sie wird einfacher, eleganter und sicherer. Allerdings muss man sich dafür auch mit neuen APIs und neuen Konzepten vertraut machen.

1.1 Das .NET-Framework

Das .NET-Framework bildet den Kern der .NET-Technologie (siehe Abb. 1.1). Es besteht aus einer Laufzeitumgebung und einer objektorientierten Klassenbibliothek, die alle Bereiche der Windows- und Web-Programmierung abdeckt. Dazu kommt noch die neue Programmiersprache C#, die auf .NET abgestimmt ist. In diesem Kapitel werden die einzelnen Teile des Frameworks kurz vorgestellt. Die übrigen Kapitel des Buches gehen dann darauf näher ein.

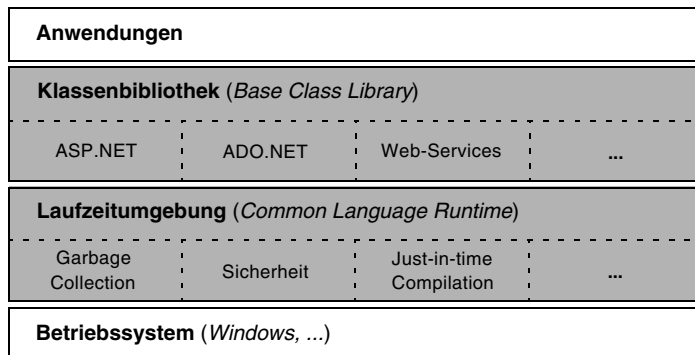


Abb. 1.1 Grobarchitektur des .NET-Frameworks

Common Language Runtime

Die *Common Language Runtime* (CLR) ist die Laufzeitumgebung, unter der .NET-Programme ausgeführt werden und die unter anderem Garbage Collection, Sicherheit und Interoperabilität unterstützt.

Ähnlich wie die Java-Umgebung basiert die CLR auf einer *virtuellen Maschine* mit einem eigenen Befehlssatz (CIL – *Common Intermediate Language*), in den die Programme aller .NET-Sprachen übersetzt werden. Unmittelbar vor der Ausführung (*just in time*) werden CIL-Programme dann in den Code der Zielmaschine (z.B. in Intel-Code) umgewandelt. Der CIL-Code garantiert die Interoperabilität zwischen den verschiedenen Sprachen und die Portabilität des Codes, die JIT-Compilation (*just in time compilation*) stellt sicher, dass die Programme trotzdem effizient ausgeführt werden.

Damit verschiedene Sprachen zusammenarbeiten können, genügt es aber nicht, sie in CIL-Code zu übersetzen. Es muss auch gewährleistet sein, dass sie die gleiche Art von Datentypen benutzen. Die CLR definiert daher auch ein gemeinsames Typsystem – das *Common Type System* (CTS), das beschreibt, wie Klassen, Interfaces und elementare Typen auszusehen haben. Das CTS erlaubt nicht nur, dass eine Klasse, die zum Beispiel in C# implementiert wurde, von einem Visual Basic .NET-Programm benutzt werden kann; es ist sogar möglich, diese C#-Klasse in Visual Basic .NET durch eine Unterklasse zu erweitern oder eine Ausnahme (*exception*), die in C# ausgelöst wurde, von einem Programm in einer anderen Sprache behandeln zu lassen.

Die *Common Language Specification* (CLS) ist jene minimale Teilmenge des CTS, die von allen Sprachen unterstützt werden muss, die von der Sprachinteroperabilität von .NET Gebrauch machen wollen. Zurzeit gibt es mehr als 20 solcher Sprachen, sowohl von kommerziellen Anbietern als auch von Universitäten. Neben den Microsoft-Sprachen C#, Visual Basic .NET und Managed C++ gehören unter anderem auch Fortran, Eiffel, Java, ML, Oberon, Pascal, Perl, Python und

Smalltalk dazu. Managed C++ ist eine Variante von C++, die in *Managed Code* übersetzt wird, der unter der Kontrolle der CLR läuft.

Die CLR stellt Mechanismen zur Verfügung, die .NET-Programme sicherer und robuster machen. Dazu gehört zum Beispiel der *Garbage Collector*, der dafür zuständig ist, den Speicherplatz von Objekten freizugeben, sobald diese nicht mehr benutzt werden. In älteren Sprachen wie C oder C++ ist der Programmierer für die Freigabe von Objekten selbst verantwortlich. Dabei kann es vorkommen, dass er ein Objekt freigibt, das noch von anderen Objekten benutzt wird. Diese Objekte greifen dann »ins Leere« und zerstören fremde Speicherbereiche. Umgekehrt kann es vorkommen, dass ein Programmierer vergisst, Objekte freizugeben. Diese bleiben dann als Speicherleichen (*memory leaks*) zurück und verschwenden Platz. Solche Fehler sind schwer zu finden, können aber dank Garbage Collector unter .NET nicht vorkommen.

Wenn ein CIL-Programm geladen und in Maschinencode übersetzt wird, prüft die CLR mittels eines *Verifiers*, dass die Typregeln des CTS nicht verletzt werden. Es ist zum Beispiel verboten, eine Zahl als Adresse zu interpretieren und damit auf fremde Speicherbereiche zuzugreifen.

Die CLR stellt also eine gemeinsame Plattform für alle .NET-Programme dar, egal in welcher Sprache sie geschrieben sind und auf welcher Maschine sie laufen. Durch ein System wohl durchdachter Regeln wird sichergestellt, dass alle Programme dieselben Vorstellungen über das Aussehen von Typen haben und Mechanismen wie Methodenaufrufe, Ausnahmen und Threads auf die gleiche Art behandeln. Die CLR als Teil der .NET-Architektur wird in Kapitel 3 beschrieben.

Assemblies

.NET unterstützt komponentenorientierte Softwareentwicklung. Die Komponenten heißen *Assemblies* und sind die kleinsten Programmbausteine, die separat ausgeliefert werden können. Ein Assembly ist eine Sammlung von Klassen und anderen Ressourcen (z.B. Bildern) und wird entweder als ausführbare EXE-Datei oder als Bibliotheksbaustein in Form einer DLL-Datei (*dynamic link library*) gespeichert. In manchen Fällen kann ein Assembly auch aus mehreren Dateien bestehen.

Jedes Assembly enthält neben Code auch *Metadaten*, also die vollständige Schnittstellenbeschreibung seiner Klassen, Felder, Methoden und sonstigen Programmelemente. Zusätzlich enthält es ein *Manifest*, das man sich als Inhaltsverzeichnis vorstellen kann. Assemblies sind also selbstbeschreibend und können mittels *Reflection* vom Lader, Compiler und diversen Werkzeugen analysiert und benutzt werden.

Assemblies dienen auch der Versionierung, d.h., sie haben eine mehrstufige Versionsnummer, die für alle in ihnen enthaltenen Klassen gilt. Wenn eine Klasse übersetzt wird, werden in ihrem Objektcode die Versionsnummern der aus anderen Assemblies benutzten Klassen vermerkt. Der Lader lädt dann jene Assemblies,

die der erwarteten Versionsnummer entsprechen. Unter .NET können also mehrere gleichnamige DLLs mit unterschiedlichen Versionsnummern nebeneinander existieren, ohne sich in die Quere zu kommen. Das bedeutet das Ende der »DLL Hell« unter Windows, bei der durch die Installation neuer Software alte DLLs durch gleichnamige neue überschrieben werden konnten und dadurch existierende Software plötzlich nicht mehr funktionierte.

Assemblies müssen auch nicht mehr in die Windows-Registry eingetragen werden. Man kopiert sie einfach ins Applikationsverzeichnis oder in den so genannten *Global Assembly Cache* und kann sie ebenso einfach wieder entfernen.

Assemblies sind gewissermaßen die Nachfolger von COM-Komponenten. Anders als unter COM (*Component Object Model*) braucht man Assemblies aber nicht mehr durch eine IDL (*Interface Definition Language*) beschreiben, da sie ja die vollständigen Metadaten enthalten, die der Compiler aus ihrem Quellcode gewonnen hat. Aufgrund des Common Type Systems wird sichergestellt, dass Software, die in unterschiedlichen Sprachen geschrieben wurde, die gleiche Art von Metadaten benutzt und somit binärkompatibel ist. Investitionen in die COM-Technologie sind aber nicht verloren. Es ist möglich, COM-Komponenten von .NET-Klassen aus zu verwenden und umgekehrt.

Die Sprache C#

Obwohl .NET-Programme in ganz verschiedenen Sprachen geschrieben werden können, hat Microsoft für .NET eine neue »Hausprache« entwickelt, die die Mächtigkeit des CTS voll ausnutzt. C# (sprich: *see sharp*) ist eine objektorientierte Sprache, die sich äußerlich stark an Java anlehnt, aber in ihrer Mächtigkeit darüber hinausgeht.

C# unterstützt einfache Codevererbung und mehrfache Schnittstellenvererbung über *Interfaces*. Sie erlaubt die Entwicklung von Komponenten im Sinne der komponentenorientierten Programmierung, indem sie *Properties*, *Events* und *Delegates* zur Verfügung stellt. Alle Typen bilden in C# (wie unter .NET üblich) ein einheitliches Typsystem, das es erlaubt, auch elementare Typen wie Zahlen und Zeichen als Objekte zu behandeln. Neben Referenztypen wie Klassen und Arrays gibt es auch Werttypen wie Structs und Enumerationen, die nicht auf dem Heap angelegt werden, sondern im normalen Variablenbereich. Dadurch wird der Garbage Collector entlastet und die Effizienz der Programme erhöht.

Numerische Anwendungen arbeiten häufig mit mehrdimensionalen Arrays. Im Gegensatz zu Java können solche Arrays in C# in einem zusammenhängenden Speicherbereich angelegt werden, was die Effizienz numerischer Anwendungen erhöht. Beim Iterieren über Objektsammlungen (*collections*), Arrays und Strings können Index-Operatoren (so genannte *Indexer*) verwendet werden. Eine neue Schleifenform – die *foreach*-Schleife – macht das Arbeiten mit solchen Objektsammlungen besonders einfach und lesbar.

C# ist eine der ersten Sprachen, die vom Programmierer mit Hilfe so genannter *Attribute* erweitert werden können. Attribute sind Metainformationen, die an fast alle Programmelemente (Klassen, Felder, Methoden, Parameter etc.) angehängt und von anderen Programmen zur Laufzeit mittels Reflection ausgewertet werden können. Damit lassen sich bedingte Compilation, Serialisierung von Objekten, COM-Interoperabilität und andere nützliche Mechanismen implementieren.

Reflection erlaubt es (wie in allen .NET-Sprachen), nicht nur auf Attribute zuzugreifen, sondern auch auf andere Metainformationen. So kann man zum Beispiel zur Laufzeit herausfinden, welche Methoden eine Klasse hat. Ein Aufruf einer dieser Methoden kann dann samt Parametern aus Laufzeitdaten zusammengestellt und ausgeführt werden. Das ermöglicht die bequeme Realisierung von Debuggern, Analysewerkzeugen und Testumgebungen.

Weitere Merkmale von C# sind die Fehlerbehandlung mittels Ausnahmen (*exceptions*), das Überladen von Operatoren und die Deklaration eigener Konversionsoperationen zwischen verschiedenen Typen.

Seit Version 2.0 bietet C# auch *generische Typen und Methoden*. Damit lassen sich zum Beispiel Listen oder Tabellen herstellen, bei denen der Typ ihrer Elemente als Parameter angegeben werden kann. Generische Bausteine erhöhen die Typsicherheit, Lesbarkeit und Effizienz von Programmen. Momentan arbeitet Microsoft an Version 3.0 von C#, in der vor allem SQL-artige Abfragen auf Hauptspeicherdaten eingeführt werden sollen.

Viele der Eigenschaften von C# finden sich auch in anderen Sprachen. C# ist also nicht revolutionär, aber eine gelungene Mischung aus den besten Eigenschaften moderner Programmiersprachen, eine an praktischen Bedürfnissen orientierte Auswahl moderner Konzepte des Software Engineering, an denen man als .NET-Entwickler kaum vorbeikommt. Deshalb ist der Sprache C# auch ein eigenes Kapitel dieses Buches gewidmet.

Die Base Class Library

Die *Base Class Library* (BCL) ist die Klassenbibliothek von .NET, die von allen .NET-Sprachen gleichermaßen benutzt werden kann und Funktionen für alle erdenklichen Zwecke zur Verfügung stellt. Sie ersetzt in den meisten Fällen die bisherigen Windows-APIs, die für ihre Komplexität und Unhandlichkeit berüchtigt waren. Es ist aber auch unter .NET noch möglich, klassische Windows-Funktionen aufzurufen. Die BCL ist in Namensräume gegliedert, die jeweils eine bestimmte Funktionalität abdecken. Zu den wichtigsten Namensräumen gehören die folgenden:

- `System.Collections` und `System.Collections.Generic` enthalten Klassen, mit denen man Sammlungen von Objekten verwalten kann. Dazu gehören Listen, Mengen, Bäume, dynamische und assoziative Arrays sowie Hashtabellen.
- `System.IO` enthält Klassen für die Ein-/Ausgabe. Hierzu zählen allgemeine Datenströme, Dateien, Verzeichnisse sowie Mechanismen zum Lesen und Schreiben von Daten in verschiedenen Formaten.
- `System.Threading` stellt Mechanismen für die parallele Programmierung zur Verfügung. Dazu gehören Threads und Thread-Pools sowie Synchronisationsmechanismen wie Monitore und Semaphore.
- `System.Net` ist jener Teil der Bibliothek, der sich mit der Netzwerkprogrammierung beschäftigt. Dazu gehören Sockets und Netzwerkströme, Protokolle wie HTTP samt den entsprechenden Request- und Response-Klassen sowie Cookies.
- `System.Reflection` erlaubt den Zugriff auf Metadaten, also auf Typinformationen von Programmen. Dazu gehören Klassen wie `Assembly`, `Type`, `MemberInfo` oder `MethodInfo`, mit denen man nicht nur Informationen über Programme abfragen, sondern auch dynamisch Daten manipulieren und Methoden aufrufen kann. Es ist sogar möglich, Programme zur Laufzeit zu erzeugen und auszuführen.
- `System.Windows.Forms` befasst sich mit grafischen Benutzeroberflächen. Hier sind Klassen für Fenster, Dialoge und GUI-Elemente enthalten. Dieser Namensraum gehört zu den umfangreichsten und komplexesten Teilen der BCL und ersetzt die bisher unter Windows üblichen *Microsoft Foundation Classes* (MFC). Visual Studio .NET ermöglicht es übrigens, grafische Benutzeroberflächen interaktiv mittels Drag-and-Drop zusammenzustellen und Methoden anzugeben, mit denen auf Benutzereingaben reagiert werden kann.
- `System.Xml` enthält Klassen zum Erzeugen und Lesen von Daten im XML-Format (*Extensible Markup Language* [XML]). XML spielt in Zusammenhang mit Web-Services eine wichtige Rolle.

Dieser Kern der BCL wird in Kapitel 4 behandelt. Weitere wichtige Namensräume sind `System.Web`, der für Webseitenprogrammierung unter ASP.NET benötigt wird, und `System.Data`, in dem man Klassen für Datenbankzugriffe unter ADO.NET findet. Beide werden in eigenen Kapiteln dieses Buches beschrieben.

ADO.NET

ADO.NET umfasst alle Teile der .NET-Bibliothek, die für den Zugriff auf Datenbanken und andere Datenquellen (z.B. XML-Dateien) zuständig sind. Es gab bereits eine Vorgängertechnologie namens ADO (*ActiveX Data Objects*), die jedoch

mit ADO.NET kaum mehr als den Namen gemeinsam hat. ADO.NET ist objektorientiert und somit strukturierter und einfacher zu benutzen.

ADO.NET unterstützt das relationale Datenmodell mit Transaktionen und Sperrmechanismen. Dabei ist es unabhängig von verschiedenen Anbietern und Datenbankarchitekturen. Implementierungen konkreter Datenbankanbindungen an MS SQL Server, OLE DB (*Object Linking and Embedding Database*) und ODBC (*Open Database Connectivity*), Oracle und andere werden durch gemeinsame Interfaces abstrahiert.

Der Zugriff auf Datenquellen kann entweder verbindungsorientiert oder verbindungslos erfolgen. Im ersten Fall wird eine ständige Verbindung zur Datenquelle aufrechterhalten. Der Datenzugriff erfolgt direkt über SQL-Kommandos, und Änderungen sind unmittelbar in der Datenbank sichtbar. Im zweiten Fall wird ein Schnappschuss eines Teils der Datenbank in Form eines DataSet-Objekts in den Hauptspeicher geholt und dann dort weiterverarbeitet. Nachdem über SQL die Daten in den Hauptspeicher geladen wurden, wird die Verbindung zur Datenquelle getrennt. Änderungen erfolgen vorerst unabhängig von der Datenquelle und müssen später abgeglichen werden. Der verbindungslose Zugriff eignet sich besonders für verteilte, von der Datenquelle entkoppelte Szenarien, wie sie bei heutigen verteilten Anwendungen mit einer Multi-Tier-Architektur typisch sind. ADO.NET wird in Kapitel 5 dieses Buches beschrieben.

ASP.NET

ASP.NET ist jener Teil der .NET-Technologie, der die Programmierung dynamischer Webseiten abdeckt. Mit der Vorgängertechnologie ASP (*Active Server Pages*) hat auch ASP.NET nur den Namen gemein. Das Programmiermodell hat sich grundlegend geändert.

Mit ASP.NET werden Webseiten am Server dynamisch aus aktuellen Daten zusammengestellt und in Form von reinem HTML an Klienten geschickt, wo sie von jedem Web-Browser angezeigt werden können. Im Gegensatz zu ASP wird in ASP.NET ein objektorientiertes Programmiermodell verwendet. Sowohl die Webseite als auch die in ihr vorkommenden GUI-Elemente sind Objekte, die man über einen Namen ansprechen und auf deren Felder und Methoden man in Programmen zugreifen kann. All das geschieht in einer kompilierten Sprache wie C# oder Visual Basic .NET und nicht wie in ASP in einer interpretierten Sprache wie JavaScript oder VBScript. Daher hat man auch Zugriff auf die gesamte Klassenbibliothek von .NET.

Die Verarbeitung von Benutzereingaben folgt einem ereignisgesteuerten Modell. Wenn der Benutzer ein Textfeld ausfüllt, einen Button anklickt oder einen Eintrag aus einer Liste wählt, wird ein Ereignis ausgelöst, das dann durch serverseitigen Code behandelt werden kann. Obwohl der Server – wie am Internet üblich – zustandslos ist, wird der Zustand einer Webseite zwischen den einzelnen Be-

nutzeraktionen aufbewahrt, und zwar in der Seite selbst. Das stellt eine wesentliche Erleichterung gegenüber älteren Programmiermodellen dar, bei denen der Programmierer für die Zustandsverwaltung selbst verantwortlich war.

ASP.NET bietet eine reichhaltige Bibliothek von GUI-Elementen, die weit über das hinausgeht, was unter HTML verfügbar ist, obwohl alle GUI-Elemente letztendlich auf HTML abgebildet werden. Der Programmierer hat sogar die Möglichkeit, eigene GUI-Elemente zu implementieren und somit die Benutzeroberfläche von Webseiten seinen speziellen Bedürfnissen anzupassen. Besonders einfach ist die Darstellung von Datenbankabfrageergebnissen in Form von Listen und Tabellen, was von ASP.NET weitgehend automatisiert wird. Eine weitere Neuheit von ASP.NET sind Validatoren, mit denen Benutzereingaben auf ihre Gültigkeit überprüft werden können.

Auch die Authentifizierung von Benutzern beim Zugriff auf geschützte Webseiten wird auf unterschiedliche Weise unterstützt, von der Standardauthentifizierung unter Windows über Cookie-basierte Authentifizierung bis zum externen Passport-Authentifizierungsdienst von Microsoft.

Seit .NET 2.0 kann man mit Hilfe so genannter Master-Seiten ein einheitliches Layout für alle Webseiten einer Anwendung festlegen. Verschiedene Navigationshilfen unterstützen einen, sich auf diesen Webseiten zurechtzufinden. Ferner gibt es Klassen zur Personalisierung von Webseiten und zur Verwaltung von Benutzerrechten sowie zahlreiche neue GUI-Elemente, die das Arbeiten mit .NET 2.0 weiter vereinfachen.

Mit Visual Studio .NET kann man die Benutzeroberfläche von Webseiten interaktiv erstellen, wie man das bei Benutzeroberflächen von Desktop-Anwendungen gewohnt ist. GUI-Elemente können mit der Maus in einem Fenster positioniert werden. Über Menüs und Property-Fenster kann man Attribute setzen und Methoden spezifizieren, die als Reaktion auf Benutzereingaben aufgerufen werden sollen. All das verwischt die Unterschiede zwischen der Programmierung lokaler Desktop-Anwendungen und Internetanwendungen und erleichtert das Erstellen von Web-Shops und tagesaktuellen Informationsseiten (z.B. Börseninformatio-
nen). ASP.NET wird in Kapitel 6 dieses Buches eingehend behandelt.

Web-Services

Web-Services werden von Microsoft als einer der Kernpunkte der .NET-Technologie bezeichnet, obwohl es sie auch außerhalb von .NET gibt. Es handelt sich um Prozedurfernaufrufe (*remote procedure calls*), die als Protokolle meist HTTP und SOAP (eine Anwendung von XML) benutzen.

Das Internet hat sich als äußerst leistungsfähig und geeignet erwiesen, um auf weltweit verstreute Informationen und Dienste zuzugreifen. Bisher erfolgte dieser Zugriff jedoch meist über Web-Browser wie den Internet Explorer oder den Netscape Navigator. Web-Services sollen nun eine neue Art des Zusammenspiels zwi-

schen verteilten Applikationen ermöglichen, bei denen die Kommunikation ohne Web-Browser abläuft. Normale Desktop-Anwendungen können sich Informationen wie aktuelle Wechselkurse oder Buchungsdaten über ein oder mehrere Web-Services holen, die als Prozeduren auf anderen Rechnern laufen und über das Internet angesprochen werden.

Die Aufrufe und Parameter werden dabei in der Regel mittels SOAP [SOAP] codiert, einem auf XML basierenden Standard, der von den meisten großen Firmen unterstützt wird. Der Programmierer merkt jedoch von all dem nichts. Er ruft einen Web-Service wie eine normale Methode auf, und .NET sorgt dafür, dass der Aufruf nach SOAP umgewandelt, über das Internet verschickt und auf dem Zielrechner wieder decodiert wird. Am Zielrechner wird die gewünschte Methode aufgerufen, die ihre Ergebnisse wieder transparent über SOAP an den Rufer zurückschickt. Der Rufer und die gerufene Methode können dabei in ganz verschiedenen Sprachen geschrieben sein und auf unterschiedlichen Betriebssystemen laufen.

Damit .NET die SOAP-Codierung und Decodierung korrekt durchführen kann, werden Web-Services samt ihren Parametern mittels WSDL (*Web Services Description Language* [WSDL]) beschrieben. Auch das erledigt .NET automatisch.

Microsoft erwartet, dass es weltweit unzählige Web-Services geben wird, die nützliche Dienste anbieten. Um den richtigen Web-Service zu finden, wurde daher UDDI (*Universal Description, Discovery and Integration* [UDDI]) entwickelt, das man sich als Adressbuch vorstellen kann, welches einem hilft, den für einen bestimmten Zweck passenden Web-Service zu finden. UDDI übernimmt also die Funktion einer Suchmaschine für Web-Services. Web-Services werden in Kapitel 7 dieses Buches näher beschrieben.

1.2 Was bringt .NET?

Wir haben gesehen, dass .NET gegenüber der bisherigen Windows- und Web-Programmierung zahlreiche Neuerungen enthält. Was bringen sie aber konkret? Welchen Nutzen haben Programmierer und Anwender von .NET?

Robustheit und Sicherheit

Durch Typprüfung und Verifikation des CIL-Codes wird sichergestellt, dass Programme keine unerlaubten Operationen durchführen, also zum Beispiel nicht auf fremde Speicherbereiche zugreifen oder Zeiger manipulieren. Der Garbage Collector garantiert, dass keine Fehler bei der Speicherfreigabe auftreten können. All das beseitigt einen Großteil der Probleme, die Programmierer bisher an den Rand der Verzweiflung brachten.

Die Versionierung von Assemblies erlaubt die Koexistenz gleichnamiger DLLs mit unterschiedlichen Versionen und verhindert das ungewollte Überschreiben bestehender DLLs durch neue. Man spricht vom Ende der »DLL Hell«.

Systemadministratoren können unter .NET nicht nur Zugriffsrechte für einzelne Personengruppen definieren (*rollenbasierte Rechte*), sondern auch Rechte für bestimmte Codeteile (*codebasierte Rechte*), die geprüft werden, egal von welcher Person dieser Code ausgeführt wird. Assemblies können nach dem *Public-Key-Verfahren* signiert werden, damit man sicher sein kann, dass sie in der vorliegenden Form vom Hersteller stammen und nicht nachträglich manipuliert oder erweitert wurden. Das schränkt die Möglichkeiten von Viren deutlich ein.

Einfachere Installation und Deinstallation von Software

Software kann unter .NET installiert werden, indem man einfach ein Verzeichnis anlegt, in das man alle Programmdateien kopiert. DLLs müssen nicht mehr in einem globalen Systemverzeichnis stehen und auch nicht in die Windows-Registry eingetragen werden. Assemblies, die von mehreren Programmen benutzt werden, kann man im so genannten *Global Assembly Cache* speichern, in dem auch gleichnamige DLLs mit unterschiedlichen Versionsnummern stehen können.

Genauso leicht kann man Programme auch wieder deinstallieren, indem man einfach die Dateien von der Platte löscht. Es bleiben keine Registry-Einträge oder sonstigen Rückstände zurück.

Interoperabilität

Unter .NET müssen nicht alle Softwareteile in der gleichen Sprache geschrieben sein, sondern man kann für jeden Teil die am besten geeignete Sprache wählen, z.B. Managed C++ für systemnahe Teile, C# oder Visual Basic .NET für die Benutzeroberfläche und ML für Dinge, die sich in einer funktionalen Sprache am besten ausdrücken lassen.

Aufgrund der Common Language Runtime und ihres Common Type Systems können diese Programmteile nahtlos zusammenarbeiten. Es ist nicht nur der Aufruf von Methoden aus anderen Sprachen erlaubt, sondern es ist zum Beispiel auch möglich, in Visual Basic .NET ein Objekt einer Klasse zu erzeugen, die in Eiffel deklariert wurde, oder in einem C#-Programm eine Ausnahme zu behandeln, die in Managed C++ ausgelöst wurde.

Das vom Common Type System vorgegebene objektorientierte Programmiermodell fördert die Entwicklung von Software, die objektorientiert und somit modular, erweiterbar und einfacher zu warten ist.

Einheitlichere Software für Desktop und Web

Mit .NET ist auch für die Web-Programmierung das objektorientierte Programmiermodell eingeführt worden, das sich in der Desktop-Programmierung schon seit Jahren bewährt. Webseiten und ihre Inhalte sind Objekte mit Daten und Methoden, die man in serverseitigem Code benutzen kann. Kryptische Mischungen aus HTML und Skriptcode, wie man es von ASP gewohnt war, gehören der Vergangenheit an.

Auch der Zugriff auf Dienste, die auf anderen Rechnern laufen, wird mit Web-Services zum gewöhnlichen Methodenaufruf. Die Softwareentwicklung für Desktop- und Web-Applikationen, die in den vergangenen Jahren divergierte, wird durch .NET wieder enger zusammengeführt.

Standards

Obwohl die .NET-Technologie von Microsoft stammt, besteht ihr Kern aus mehreren offenen Standards. Der ECMA-Standard 335 [CLI] definiert zum Beispiel die *Common Language Infrastructure* (CLI), zu der die CLR und Teile der BCL gehören. Der ECMA-Standard 334 [C#Std] definiert die Sprache C#. SOAP basiert auf den W3C-Standards für HTML und XML und ist selbst als IETF-Standard (*Internet Engineering Task Force*) eingereicht [SOAP]. Auch WSDL soll ein W3C-Standard werden [WSDL]. UDDI schließlich ist ein De-facto-Standard, der von mehr als 200 Firmen unterstützt wird, unter anderem von Boeing, Cisco, Fujitsu, Hitachi, HP, IBM, Intel, Microsoft, Oracle, SAP und Sun [UDDI].

1.3 Unterschiede zu Java

.NET weist viele Ähnlichkeiten mit Java auf und wird daher auch oft mit der Java-Technologie verglichen. Tatsächlich basieren sowohl Java als auch .NET auf einer virtuellen Maschine, die die Laufzeitumgebung darstellt und in deren Code alle Programme übersetzt werden.

Während unter .NET jedoch CIL-Programme vor der Ausführung *immer* in Maschinencode übersetzt werden, werden Programme im Java-Bytecode anfänglich interpretiert. Erst wenn eine Methode eine gewisse Anzahl von Malen aufgerufen oder eine Schleife genügend oft ausgeführt wurde, wird die entsprechende Methode im Hintergrund in Maschinencode umgewandelt. Das hat den Vorteil, dass Java-Programme sofort im interpretierten Modus startbereit sind, während unter .NET erst der JIT-Compiler laufen muss und Programme daher beim ersten Aufruf mit einigen Zehntelsekunden Verzögerung beginnen. Dafür braucht man unter .NET keinen Interpreter und hatte auch mehr Freiheiten beim Entwurf des CIL-Codes, da er nicht auf Interpretation ausgelegt sein musste.

Die Sprachen Java und C# sind einander auf den ersten Blick sehr ähnlich. Es gibt sogar viele Pascal-Dialekte, die sich stärker voneinander unterscheiden als Java und C#. Bei genauerer Betrachtung ist C# jedoch mächtiger als Java, auch wenn viele Spracheigenschaften nur »syntactic sugar« sind und sich auch in Java auf die eine oder andere Art bewerkstelligen lassen. C# hat zum Beispiel ein einheitliches Typsystem, kennt Referenzparameter und hat viele nützliche Details wie Properties, Indexer, Delegates und Iteratoren. Dafür ist Java einfacher und strikter. Es kennt zum Beispiel keine Programmteile, die als »unsafe« gekennzeichnet sind und in denen manche Typregeln außer Kraft gesetzt werden. Auch bei Ausnahmen ist Java strenger und fordert, dass der Programmierer sie immer behandeln muss, was in C# nicht der Fall ist.

Die Klassenbibliothek von Java und .NET ist stellenweise sehr ähnlich, was sogar so weit geht, dass die Namen vieler Klassen und Methoden in beiden Systemen gleich sind.

An Stelle von ASP.NET tritt in Java eine Technologie namens JSP (*Java Server Pages*), die wiederum aus der älteren ASP-Technologie abgeleitet wurde. JSP-Seiten werden wie in .NET in Klassen (*Servlets*) übersetzt, die den für den Klienten bestimmten HTML-Strom erzeugen. Der Hauptunterschied zwischen JSP und ASP.NET liegt darin, dass unter .NET die HTML-Beschreibung einer Webseite und ihr Programmcode sauberer getrennt werden. Sie können in .NET in unterschiedlichen Dateien stehen, während in JSP der HTML-Code mit Java-Codestücken gemischt wird. Auch die Zustandsverwaltung einer Seite, der objektorientierte Zugriff auf GUI-Elemente und die ereignisgesteuerte Art, wie auf Benutzereingaben reagiert wird, ist unter ASP.NET ausgereifter.

Auch für Web-Services gibt es eine entsprechende Java-Technologie [JavaWS], die wie unter .NET auf SOAP und WSDL basiert. Unter .NET sind Web-Services jedoch stärker ins System integriert. Microsoft hat hier einen Entwicklungsvorsprung.

Der Hauptunterschied zwischen Java und .NET liegt in den verschiedenen Zielsetzungen dieser Systeme. Während Java das Ziel verfolgt, eine einzige Sprache auf vielen verschiedenen Rechnern und Betriebssystemen zu unterstützen, hat .NET das genau entgegengesetzte Ziel, nämlich viele verschiedene Sprachen auf einer einzigen Plattform zu unterstützen. Diese Plattform heißt momentan Windows, was aber nicht bedeutet, dass das immer so bleiben muss. Es gibt bereits Implementierungen von .NET auf anderen Betriebssystemen (z.B. Linux, FreeBSD) und anderen Prozessoren (z.B. SPARC, PowerPC) [Mono]. Die beste Unterstützung wird .NET aber wahrscheinlich immer unter Windows haben.

1.4 Weiterführende Literatur

Dieses Buch gibt einen Überblick und eine Einführung in die .NET-Technologie. Es entstehen zurzeit viele Bücher, die sich mit einzelnen Aspekten von .NET (z.B.

CLR, C#, ASP.NET, Web-Services) wesentlich ausführlicher beschäftigen, als es hier möglich ist. Einige dieser Bücher sind im Literaturverzeichnis aufgeführt.

Zu .NET gibt es aber auch zahlreiche Onlinequellen, von Einführungen über Tutorials bis zu detaillierten Spezifikationen. Neue Entwicklungen werden am Internet rascher aufgegriffen und beschrieben, als das in gedruckter Form möglich ist. Hier sind einige der wichtigsten .NET-Portale aufgelistet:

- ❑ www.microsoft.com/net/
Dies ist die offizielle .NET-Seite von Microsoft mit allgemeinen Informationen zu .NET, Spezifikationen und Beispielprogrammen. Von dieser Seite kann auch die jeweils neueste Version des *.NET-Framework-SDK* geladen werden, das die CLR, die Klassenbibliothek, Compiler für C# und andere Sprachen, nicht jedoch Visual Studio .NET enthält.
- ❑ msdn.microsoft.com/net/
Dies ist die .NET-Seite des Microsoft Developer Networks mit allerlei technischen Informationen zu Aspekten von .NET.
- ❑ www.gotdotnet.com
Eine Fundgrube von Beispielen, Artikeln und anderen nützlichen Informationen rund um .NET.
- ❑ www.devhood.com
Eine Seite mit vielen Beispielen, Tutorials und Trainingsmodulen zu .NET.
- ❑ www.mono-project.com
Mono ist ein Open-Source-Projekt, in dem .NET auf Linux, Unix, Solaris und Mac OS X portiert wurde.
- ❑ dotnet.jku.at
Die Webseite zu diesem Buch mit dem Quellcode aller Beispiele sowie Links, Werkzeugen und diversen .NET-Informationen.

Eine der nützlichsten Quellen für .NET ist die Onlinedokumentation [SDKDoc], die mit dem .NET-Framework-SDK ausgeliefert wird (siehe CD zu diesem Buch). Man findet dort einführende Texte, Tutorials, Beispiele und detaillierte Spezifikationen. Insbesondere die Referenzdokumentation zur .NET-Klassenbibliothek ist ein unverzichtbares Nachschlagewerk für alle .NET-Entwickler. Die Dokumentation ist gründlich und gut zu lesen. Mit diesem Buch als Leitfaden und der Onlinedokumentation des .NET-Framework-SDK ist es möglich, alle Aspekte des .NET-Frameworks bis ins Detail auszuloten.