Zeitschrift: Bulletin Electrosuisse
Herausgeber: Electrosuisse, Verband für Elektro-, Energie- und Informationstechnik
Band: 94 (2003)
Heft: 1

Artikel: .NET von Microsoft : alle Probleme gelöst?
Autor: Willers, Michael
DOI: https://doi.org/10.5169/seals-857510

.NET von Microsoft – alle Probleme gelöst?

Im Juli 2000 hat Microsoft auf der Professional Developers Conference in Orlando einem breiten Entwicklerpublikum seine Vision und Strategien für die nächsten Jahre vorgestellt: Microsoft .NET. Mit der Freigabe von Visual Studio .NET im Februar 2002 nimmt diese Vision immer konkretere Formen an. Dabei wird nicht nur in der Entwicklergemeinde heiss diskutiert. Die Spannweite reicht dabei von «hellauf begeistert» bis «was soll das?». Neben rein technischen Aspekten wird immer öfter und lauter die Frage gestellt: «Was genau ist eigentlich .NET?» Und: «Wie passt es zu bisherigen Technologien von Microsoft?». Dieser Artikel versucht, eine Antwort auf diese und ähnliche Fragen zu geben; er spannt den Bogen von COM über COM+ und Windows DNA bis hin zu .NET.

Michael Willers

Blicken wir ein wenig zurück. Anfang der 90er-Jahre begann die Komponentenidee sich neben der reinen Objektorientierung mehr und mehr durchzusetzen: Ein Programm besteht nicht aus einem einzelnen grossen Block, sondern wird aus Komponenten zusammengesetzt.

Diese Idee hat sich auch deshalb durchgesetzt, weil im Falle eines Fehlers die Suche deutlich systematischer erfolgt: Man kann sämtliche Komponenten nacheinander durchleuchten und muss nicht nach der Stecknadel im Heuhaufen suchen. Wer schon mal eine intensive Fehlersuche hat durchmachen müssen, der weiss, wovon die Rede ist: Das Bauchgefühl ist entscheidend – an der falschen Stelle angefangen – und schon sind ein paar Stunden dahin und die Überstunden vorprogrammiert.

Microsoft hat diese Idee für die Windows-Plattform aufgegriffen und weitergeführt: Ein Programm sollte nicht nur aus Komponenten aufgebaut sein, vielmehr sollten sämtliche Komponenten auf ein und dieselbe Weise miteinander kommunizieren – und zwar unabhängig von der verwendeten Programmiersprache.

Lokale und verteilte Komponenten – COM und DCOM

Das Component Object Model (COM) beschreibt, wie diese Kommunikation aussieht. Jedes Windows-System enthält eine Implementation dieser Spezifikation – die COM-Runtime. Sie besteht im Wesentlichen aus der Datei OLE32.DLL. Der Vorteil: Komponenten werden stets auf die gleiche Art und Weise angesprochen und können als Blackbox eingesetzt werden. Dieses Konzept (Bild 1) hat sich durchgesetzt. Das belegen die zahlreichen Lösungen, die am Markt vorhanden sind. Mittlerweile existiert für nahezu jede Problemstellung eine Komponente.

Parallel zur Komponentenorientierung hat sich die Vernetzung von PCs durchgesetzt und zu einem neuen Typ von Anwendungen geführt: Programme laufen nicht mehr isoliert auf einem PC, sondern bedienen sich Komponenten, die über mehrere Rechner im Netz verteilt sein können. Damit gewinnt das Thema Sicherheit an Bedeutung. Microsoft hat COM für Fernaufrufe über das Netz erweitert und das Konzept der deklarativen Sicherheit eingeführt: Rechte sind nicht innerhalb einer Komponente «hart» kodiert, sondern werden administrativ festgelegt und beim Aufruf der Komponente überprüft (Bild 2).

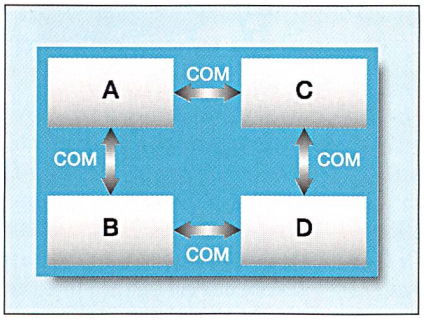

Bild 1 Das Component Object Model (COM) – Komponenten sprechen die gleiche Sprache

Diese Erweiterungen und Neuerungen werden unter der Bezeichnung Distributed COM (DCOM) zusammengefasst. Der entscheidende Punkt dabei: Aus der Sicht des Programmierers ist es völlig egal, ob sich eine Komponente auf dem lokalen oder auf einem anderen Rechner befindet. Sie wird vom Programm aus immer auf die gleiche Art und Weise angesprochen – die COM-Runtime führt die Sicherheitsüberprüfungen aus und sorgt für einen Fernaufruf über das Netz, wenn sich die Komponente auf einem anderen Rechner befindet (Bild 3).

Dienste für verteilte Anwendungen – MTS und COM+

Der Trend zu verteilten Anwendungen hat sich seit Mitte der 90er-Jahre eher noch verstärkt und die Komplexität bei der Softwareentwicklung weiter ansteigen lassen. Man denke beispielsweise an das Buchungssystem eines Reisebüros: Das Hotel muss gebucht werden, Flüge müssen bestätigt werden, eventuell wird ein Mietwagen reserviert und schliesslich muss vermerkt werden, wie und wann der Urlaub bezahlt wird. Auf weitere Einzelheiten soll an dieser Stelle verzichtet werden; das Problem dürfte klar geworden sein. Nicht selten besteht der Löwenanteil bei der Entwicklung verteilter Anwendungen aus der Lösung von Infrastrukturproblemen. Im Kern zählen dazu folgende Dienste:

- ein Sicherheitsmodell (wer darf worauf zugreifen);
- die Koordination gleichzeitiger Zugriffe durch mehrere Benutzer;
- eine Ressourcenverwaltung (Connection- und Thread-Pooling);
- verteilte Transaktionen.

Unter COM bzw. DCOM sind diese Dienste nicht automatisch vorhanden.

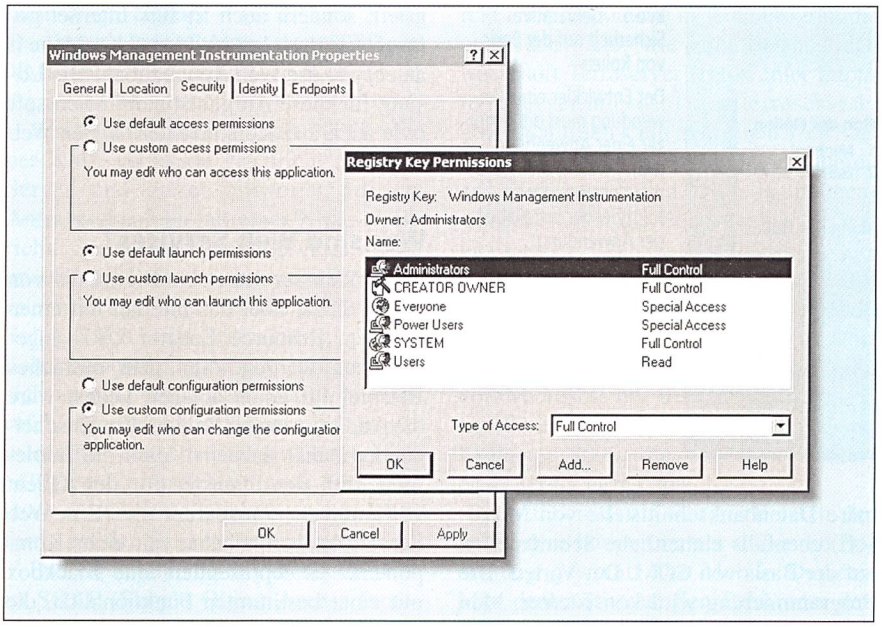

Bild 2 Distributed COM (DCOM)

Rechte für Komponentenaufrufe werden mit dem Programm DCOMCNFG.EXE definiert und in der Registry abgelegt

COM bietet zwar ein umfassendes Threading-Modell und ein Security-API – nur muss die anwendungsspezifische Logik komplett von Hand programmiert werden. (Für verteilte Transaktionen bietet der Microsoft Distributed Transaction Coordinator MSDTC entsprechende Schnittstellen.) Und wer schon mal vor dem Dilemma gestanden hat, seinem Chef zu erklären, warum ein Projekt vier statt der geplanten zwei Wochen dauert, der weiss, wo das eigentliche Problem

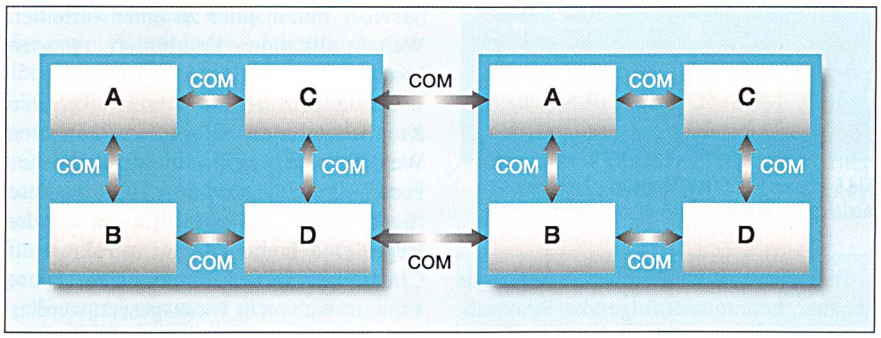

Bild 3 Distributed COM

Distributed COM ermöglicht einheitliche Kommunikation auch über Rechnergrenzen hinweg

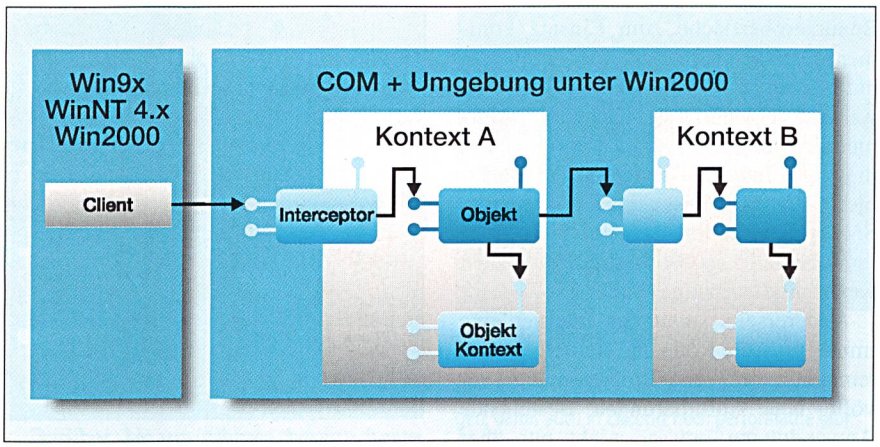

Bild 4 Der Kontext – ein Objekt erkennt jederzeit seinen Aufrufer und welche Dienste es nutzen kann

liegt – mehr Zeit und somit zusätzliche Kosten.

Um diesem Problem zu begegnen, hat Microsoft bereits 1996 mit dem Microsoft Transaction Server (MTS) eine Laufzeitumgebung für Komponenten vorgestellt, die genau diese Dienste bietet. Der MTS beruht auf dem Prinzip der attributbasierten Programmierung.

- Jede Klasse einer Komponente definiert über Attribute, welche Dienste sie benötigt.
- Beim Erstellen einer Instanz dieser Klasse (Objekt) wird eine auf deren Attributen basierende Laufzeitumgebung erstellt, die ihr die gewünschten Systemdienste bereitstellt.
- Diese Laufzeitumgebung wird Kontext genannt.

Das ist die zweite wesentliche Neuerung: Jedes Objekt verfügt über einen Kontext (Bild 4), über den es zusätzlich Informationen erfragen kann. Zu diesen Informationen zählen unter anderem:

- der Benutzer, der das Objekt erstellt hat (Direct Caller);
- der Benutzer, der die Aufrufkette angestossen hat (Original Caller);
- welche Dienste das Objekt benutzen kann (Transaktionen, Sicherheit).

Darüber hinaus ist das Konzept der deklarativen Sicherheit verfeinert und vereinfacht worden: Die Vergabe von Rechten erfolgt auf der Basis von Rollen. Der Vorteil: Die Zuordnung von Benutzern zu einer Komponente muss während der Entwicklungszeit nicht bekannt sein. Benutzer können einer Rolle bei der Installation dynamisch zugeordnet werden (Bild 5).

Der wichtigste Punkt auch hier: Für den Programmierer ist völlig transparent, ob eine Komponente unter der Regie des MTS läuft oder nicht. Sie wird vom Programm aus immer auf die gleiche Art und Weise angesprochen. Damit das funktioniert, sind unter der Haube zwei unterschiedliche Laufzeitsysteme notwendig – die COM-Runtime und die MTS-Runtime (Letztere besteht im Wesentlichen aus den Daten MTXEX.DLL und MTX.EXE). Unter Windows 2000 sind beide Umgebungen zu einer einheitlichen Laufzeitumgebung zusammengefasst worden: COM+.

Neben vielen Detail- und Performanceverbesserungen sind in COM+ auch neue Dienste integriert worden. Dazu gehören ein Event-Service und die Möglichkeit, Komponentenaufrufe über Message Queues zu verarbeiten (Queued Components). Die häufig gestellte Frage «Was ist COM+?» kann nun einfach beantwortet werden: «COM+ ist die Weiterentwicklung von COM und integriert

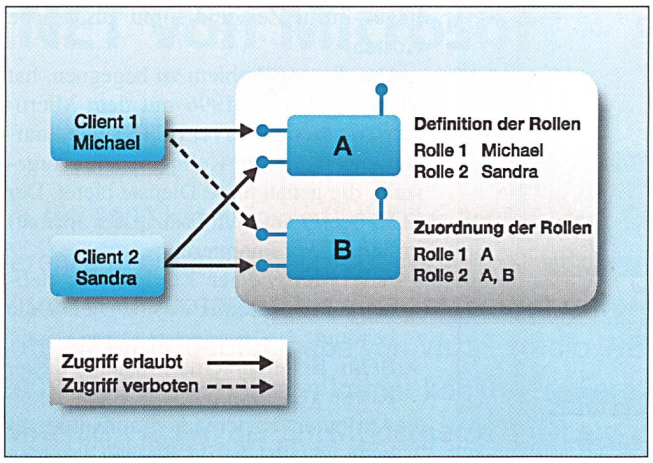

Bild 5 Deklarative Sicherheit auf der Basis von Rollen

Der Entwickler einer Anwendung muss die Benutzer einer Anwendung nicht kennen. Die Zuordnung der Benutzer erfolgt bei der Installation der Anwendung.

Dienste für die Entwicklung verteilter Anwendungen.»

Ein Modell für verteilte Anwendungen – Windows DNA

Soweit zu den Technologien. Verteilte Anwendungen baut man aber nicht nur mit Technologien allein. Dazu bedarf es auch einer geeigneten Architektur und entsprechender Produkte, auf denen die Anwendung aufsetzt. Diesen ganzheitlichen Ansatz hat Microsoft unter dem Begriff Windows DNA (Distributed Internet Architecture) zusammengefasst.

Fangen wir mit den Produkten an: Dazu zählen Windows 2000 mit COM+ als Application Server, Visual Studio als Entwicklungsumgebung sowie die heutigen Serverprodukte von Microsoft mit SQL Server und Exchange als Basis.

DNA-Anwendungen sind grundsätzlich mehrschichtig aufgebaut und folgen dem Gesetz der Trennung von Logik und Darstellung innerhalb einer Anwendung. Die meisten Anwendungen bestehen im Wesentlichen aus drei Schichten:

- einer Schicht mit Komponenten, die interne Dienste implementiert (auch eine SQL-Abfrage an eine Datenbank ist ein Dienst);
- einer Schicht mit Komponenten, die einen Prozess abbildet (Geschäftslogik) und dabei Plattformdienste nutzt (eine COM+-Transaktion ist ein Plattformdienst);
- einer Schicht, welche die Benutzeroberfläche implementiert.

Die Kommunikation zwischen allen Schichten beruht auf einem einzigen Modell – dem Component Object Model – und ist somit über alle Schichten hinweg einheitlich. Das Gleiche gilt für den Zugriff auf Systemdienste (das Anlegen einer Benutzergruppe ist zum Beispiel ein Systemdienst) und Datenquellen. Hier gibt es mit den Active Directory Service Interfaces (ADSI) und OLE DB (primäre Datenbankschnittstelle von Microsoft) ebenfalls einheitliche Schnittstellen auf der Basis von COM. Der Vorteil: Die Programmierung wird konsistenter. Man benutzt nicht mehr zig API-Funktionen, sondern programmiert durchgehend auf der Basis eines einzigen Komponentenmodells (Bild 6).

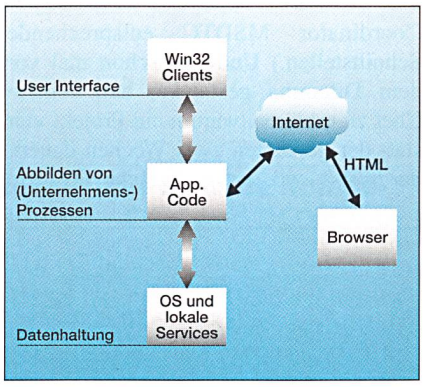

Bild 6 Components for Windows – Architekturmodell einer Windows DNA-Anwendung

Betrachtet man diesen Anwendungstyp genauer, könnte sich folgender Punkt als kritisch erweisen: Eine Internetanbindung ist nur über die Darstellungsschicht möglich. Hier kann alternativ zu einer Windows-Anwendung ein Browser als Benutzeroberfläche zum Einsatz kommen. Die Geschäftslogik kann hingegen nicht über das Internet angesprochen werden, der Zugriff ist nur intern im Firmennetz möglich. Selbst wenn es über die COM Internet Services (CIS) klappt, bleibt die Port-Diskussion ein Thema. Sofern dies für Ihre Projekte kein Problem darstellt, ist Windows DNA auch weiterhin die richtige Wahl.

Andererseits spielt das Internet eine immer grössere Rolle und stellt viele Systemintegratoren und Programmierer vor völlig neue Anforderungen. Bestehende Anwendungen müssen nicht nur über verschiedene Systemplattformen integriert, sondern auch fit fürs Internet gemacht werden. Auf Neudeutsch: «Make it an easy to use Web Application». Die Lösung für diese Aufgaben sieht Microsoft (wie auch ihre Konkurrenten) in den Web Services.

Was sind Web Services?

Ein Web Service ist ein Dienst, der von einem Client über das Internet mit einen Uniform Resource Locator URL angesprochen werden kann. Ein einfaches Beispiel für einen solchen Dienst wäre die Addition zweier Zahlen. Ein entscheidender Punkt dabei ist, dass die Implementation des Dienstes für den Client vollkommen transparent ist. Ein Web Service ist vergleichbar mit einer Komponente: Er repräsentiert eine Blackbox mit einer bestimmten Funktionalität, die man flexibel einsetzen kann, ohne deren Implementationsdetails zu kennen.

Nehmen wir das Beispiel Reisebüro: Im Idealfall spricht ein Buchungssystem per URL einen Service für Flugbuchungen an, der Flugpläne verschiedener Airlines abfragt und als einzelnes Dokument zur Verfügung stellt. Eine weitere URL liefert einen Dienst, mit dem ein bestimmter Flug gebucht werden kann. Zuletzt wird dann – ebenfalls mittels eines Web Services – die Hotelbuchung durchgeführt. Bild 7 zeigt modellhaft, wie Web Services miteinander zu einer verteilten Web-Applikation kombiniert werden können.

Im Gegensatz zu derzeit aktuellen Komponententechnologien benutzen Web Services kein objektspezifisches Protokoll wie DCOM oder IIOP, da diese für den reibungslosen Einsatz in der Regel eine homogene Infrastruktur auf Client und Server voraussetzen. Diese kann im Web nicht vorausgesetzt werden. Web Services folgen deshalb einem anderen Ansatz. Sie bauen auf Internetstandards auf und benutzen – als kleinsten ge-

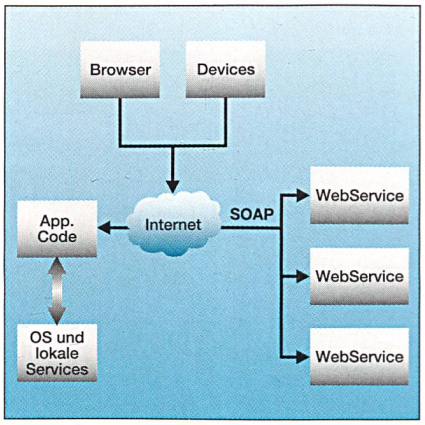

Bild 7 Components for the Web – Web Services werden zu einer Anwendung kombiniert

meinsamen Nenner – HTTP und XML. Das heisst: Jedes System, das HTTP und XML unterstützt, kann Web Services integrieren und nutzen.

Ein Client schickt mittels HTTP eine per XML verpackte Nachricht an einen Server und dieser antwortet auf die Anfrage ebenfalls mit einer XML-Nachricht. Somit sind Web Services völlig unabhängig von bestimmten Programmiersprachen und Systemplattformen. Solange sich beide Seiten auf ein einheitliches Nachrichtenformat einigen und sich an eine gemeinsam definierte Aufrufabfolge halten, ist die Art der Implementation des Dienstes (Web Service) völlig egal; er und damit auch der Client kann sämtliche Möglichkeiten der Plattform, auf der ersterer läuft, ausschöpfen.

Die Verallgemeinerung dieses Prinzips ist SOAP (Simple Object Access Protokoll). Dieses definiert, wie die XML-Nachrichten aufgebaut sein müssen und wie die Aufruffolge auszusehen hat. Damit können unterschiedlichste Anwendungen, die auf verschiedenen Plattformen laufen, über das Internet miteinander kombiniert und in bestehende Lösungen integriert werden. Einzige Voraussetzung ist, dass die Anwendungen über SOAP miteinander kommunizieren.

Die Microsoft .NET-Plattform

Es entsteht ein völlig neuer Anwendungstyp: Verschiedene Dienste werden über das Internet abgerufen und zu einer Lösung integriert. Solche Anwendungen bringen eine Reihe neuer Anforderungen mit sich, die mit den bisherigen Werkzeugen und Programmiermodellen nur schwer bewerkstelligt werden können. Dazu zählen u.a. folgende Fragestellungen:

- Wie programmiert man einen Web Service?
- Wie kann man einen Web Service debuggen?
- Wie installiert man einen Web Service?

Aus der Sicht des Programmierers ist es allerdings noch viel wichtiger, solche Web-Anwendungen auf einfache Weise zu entwickeln. Dazu benötigt man eine entsprechende Entwicklungsumgebung und eine moderne Klassenbibliothek für die Programmierung. Diese Gründe haben zu Microsofts Entschluss geführt, neue Werkzeuge und ein Framework zu entwickeln, das diesen Anforderungen gerecht wird. Fassen wir beides zunächst unter dem Begriff Framework und Tools zusammen.

Darüber hinaus gibt es bereits heute vorgefertigte Web Services, die man direkt als Komponente in seine Programme einbinden kann, wie zum Beispiel den Microsoft Terraserver (Infos unter http://terraserver.microsoft.net/terraservice.htm). Natürlich kann man einen Web Service auch selbst entwickeln und anderen zur Verfügung stellen. Diese Komponenten bekommen die Bezeichnung MyServices.

Ausserdem benötigt man für den Betrieb von Web Services eine entsprechende Infrastruktur. Diese Infrastruktur bildet die heutige 2000-Produktfamilie von Microsoft mit den Basiskomponenten Windows 2000, SQL Server 2000 und Exchange 2000. Sie werden als Enterprise Server bezeichnet.

Hinzu kommt ein weiterer Bereich, der neben dem Internet immer stärker an Bedeutung gewinnt: Die Mobile Devices. Man denke etwa an den Palm oder den Compaq IPAQ. Solche Geräte haben sich neben dem klassischen PC als Alternative etabliert. Auch auf diesen Geräten sollen zukünftig Anwendungen laufen, die mit dem Framework programmiert werden. Somit bilden die folgenden vier Bestandteile die Microsoft .NET-Plattform:

- Framework und Tools, welche die Common Language Runtime (CLR), eine einheitliche Klassenbibliothek und Studio.NET beinhalten;
- MyServices, welche die ständig verfügbaren Internet-Dienste (Code-Updates, Suchdienste, Messengers) enthalten;
- Enterprise Servers, welche heute die 2000-Produktfamilie, in Zukunft die .NET-Enterprise Servers bezeichnen;
- Devices, mobile Geräte, auf denen .NET-Anwendungen laufen (Handy, Handheld).

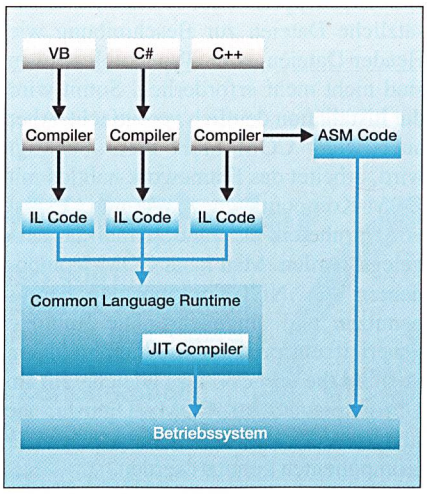

Bild 8 Sprachintegration erfolgt zukünftig auf Code-Ebene

Nur Visual C++ kann weiterhin native Code erzeugen, damit auch in Zukunft noch performante (auf den Zweck zugeschnittene, schnelle) Treibersoftware entwickelt werden kann.

Die .NET-Plattform wird nicht über Nacht da sein. .NET stellt Microsofts Strategie der nächsten Jahre dar. Einige Bereiche davon sind schon sehr weit fortgeschritten, während in anderen Bereichen wie den MyServices die Aufbauarbeit gerade erst abgeschlossen wird.

Wir werden also noch auf lange Sicht hin zwei Typen von Anwendungen haben: Einerseits Windows-DNA-Anwendungen, bei denen die Internet-Anbindung über den Browser völlig ausreichend ist, und andererseits Anwendungen, die intensiven Gebrauch vom Internet machen, um Web Services zu integrieren. Das mag sich alles nach zuviel Zukunftsmusik anhören. Aber wer hätte Mitte der 80er-Jahre jemandem geglaubt, der den heutigen Stellenwert des Internets vorausgesagt hätte?

Das .NET-Framework

Zum Abschluss soll der Bereich *Framework und Tools* noch ein wenig unter die Lupe genommen werden. Das .NET-Framework ist die neue Entwicklungsplattform für Anwendungen. Das Fundament bildet die Common Language Runtime. Code, der unter der Regie der Runtime ausgeführt wird, wird als Managed Code bezeichnet. Das bedeutet, dass Aktionen wie das Anlegen eines Objekts oder das Ausführen eines Methodenaufrufs nicht direkt ausgeführt, sondern an die Runtime delegiert werden. Sie kann dann zusätzliche Dienste, wie beispielsweise Versions- und Sicherheitsüberprüfungen, durchführen. Die Runtime ist also quasi ein Manager für den Code, der ausgeführt werden soll.

Die Compiler des Frameworks (derzeit Visual Basic.NET, Visual C++ und C#) erzeugen daher keinen native Code mehr. Vielmehr wird aus dem Quelltext eine Zwischensprache erzeugt, die dann unter Aufsicht der Runtime bei Bedarf zu native Code kompiliert und ausgeführt wird (Just in time Compiler).

Diese Zwischensprache wird mit Common Intermediate Language (CIL) bezeichnet und ist von der europäischen Standardisierungsbehörde ECMA im Dezember 2001 als Standard verabschiedet worden (weitere Informationen findet man unter http://msdn.microsoft.com/net/ecma). Somit kann jeder Compiler, der CIL erzeugt, Code unter Aufsicht der Runtime ausführen lassen. Oder anders gesagt: Dreh- und Angelpunkt der Runtime ist Sprachintegration. Ob man nun Cobol, Pascal, C# oder Visual Basic benutzt ist egal – solange der Compiler CIL-Code erzeugt (Bild 8).

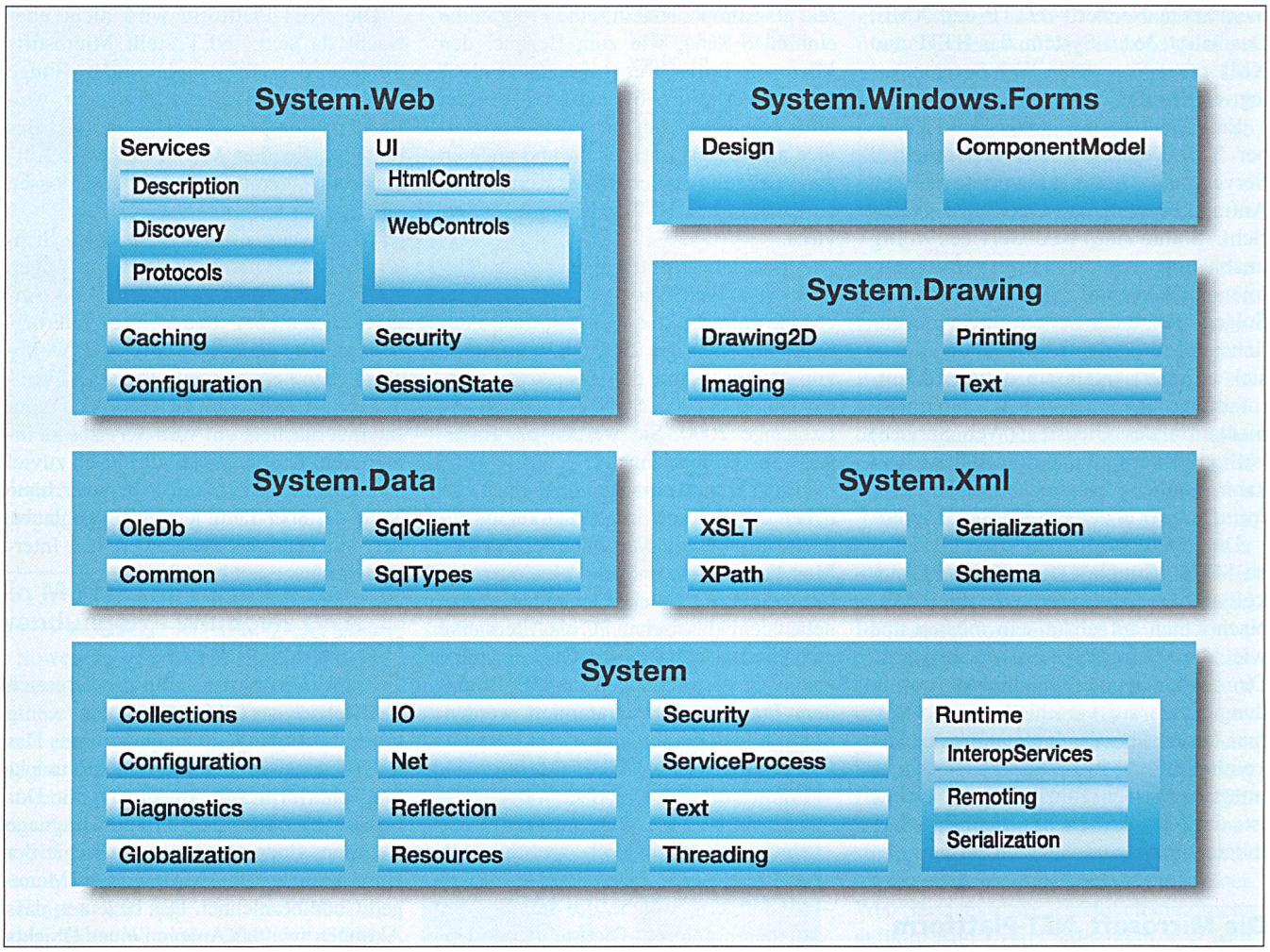

Bild 9 Die Klassenbibliothek im .NET-Framework – Klassen für fast jede Lebenslage

Da jeder .NET Compiler CIL-Code erzeugt, findet die Sprachintegration auf Codeebene und nicht wie bei COM auf binärer Ebene statt. Es kann nun beispielsweise eine Klasse in einer Sprache erstellt und mittels einer anderen Sprache eine weitere Klasse davon abgeleitet werden. Die Bedeutung, welche Sprache man zur Entwicklung von Anwendungen benutzt, rückt damit in den Hintergrund. Man arbeitet einfach mit der Sprache, die einem am ehesten liegt.

So braucht man beispielsweise eine Klassenbibliothek nur noch ein einziges Mal zu programmieren. Mithilfe der Runtime kann sie von jeder Sprache aus benutzt werden. Dieses Prinzip macht sich natürlich auch die Klassenbibliothek des Frameworks zunutze (Bild 9). Positiver Nebeneffekt: Die Programmierung wird konsistenter. Man benutzt eben nicht mehr zig API-Funktionen oder diverse Klassenbibliotheken, sondern genau eine Einzige – diejenige, die man für die Runtime erstellt hat.

Das .NET-Framework basiert nicht auf COM. Komponenten, die mit der Runtime programmiert sind, beschreiben sich selbst. Entsprechende Metadaten werden beim Kompilieren in die Komponente geschrieben. Die Registry wird nicht mehr benötigt. Der Vorteil: Das Installieren einer Komponente beschränkt sich auf einfaches Kopieren der Komponente; zusätzliche Dateien zur Beschreibung wie Header-Dateien oder Typenbibliotheken sind nicht mehr erforderlich. Somit wird die Installation deutlich vereinfacht. Aber auch wenn COM nicht mehr benötigt wird, arbeitet das Framework nahtlos mit COM-Komponenten zusammen. Es ist von vornherein auf Interoperabilität ausgelegt worden. Man kann COM-Komponenten aus .NET-Komponenten heraus benutzen und umgekehrt. Die Runtime generiert entsprechende Wrapperklassen «behind the scenes». Das Gleiche gilt im Übrigen auch für COM+-Dienste; sie können ebenfalls innerhalb von .NET-Komponenten genutzt werden.

Fazit

Das Internet und die stark zunehmende Zahl von Breitbandzugängen führt zu einem Paradigmenwechsel in der Softwareentwicklung: An die Stelle des Komplettpakets auf CD treten immer öfter auch Dienste, die bei Bedarf über das Internet abgerufen und zu einer Anwendung kombiniert werden. Wie dies vonstatten gehen kann, hat uns Napster kürzlich eindrucksvoll gezeigt.

Das alles passiert natürlich nicht über Nacht, denn neben einer geeigneten Infrastruktur sind Werkzeuge notwendig, die das Entwickeln von Internet-Anwendungen so einfach machen wie die Entwicklung von Desktop-Anwendungen. Diese Werkzeuge sind heute noch dünn gesät und oftmals ist das Lösen von Infrastruktur-Problemen nach wie vor der Löwenanteil bei Internet-Projekten.

Microsoft bietet mit der .NET-Plattform eine vielversprechende Lösung, und seit dem Erscheinen von Visual Studio .NET im Februar 2002 wird die Vision *Programming the Web* langsam aber sicher Realität. Sieht man einmal vom Thema Internet ab, bringt die .NET-Plattform auch für reine Windows-Anwendungen Verbesserungen mit sich. Es gibt ein einheitliches Integrationsmodell; die Runtime-Systeme verschiedener Spra-

chen fallen weg und werden durch eine einzige Runtime ersetzt. Die Installation von Anwendungen wird ebenfalls einfacher, da diese Runtime nicht auf der Registry aufsetzt. Und nicht zuletzt gibt es endlich eine einheitliche Klassenbibliothek, die mit den Inkonsistenzen des Win32-API aufräumt.

Angaben zum Autor

Michael Willers, Initiator und Gründer des Entwicklerforums msdn TechTalk von Microsoft, war lange Jahre als Entwickler und Projektleiter tätig. Heute liegt der Schwerpunkt seiner Arbeit in der Vermittlung und Anwendung moderner Softwaretechnologien und -architekturen, insbesondere dem .NET-Framework. Er ist Mitglied in verschiedenen Fachbeiräten und Lehrbeauftragter an Universitäten und Fachhochschulen in Deutschland. *Kontakt: michael.willers@devcoach.de, www.devcoach.de*

.NET de Microsoft – tous les problèmes sont-ils résolus?

En juillet 2000, Microsoft a présenté à l'occasion de la Professional Developers Conference à Orlando, à un vaste public de concepteurs, sa vision et ses stratégies pour les prochaines années: Microsoft .NET. Après l'homologation à la vente de Visual Studio .NET en février 2002, cette vision prend corps peu à peu. Et la discussion animée n'a pas lieu que parmi les développeurs. La gamme des réactions va de «l'enthousiasme effréné» au «ça sert à quoi?». Outre les aspects purement techniques, on pose de plus en plus souvent et avec de plus en plus d'insistance la question: «Qu'est-ce que c'est, au fond, .NET?» Et encore: «Dans quelle mesure cela cadre-t-il avec les technologies Microsoft actuelles?». L'article tente de répondre à ces questions et à d'autres du même genre; il présente toute la gamme de COM à .NET en passant par COM+ et Windows DNA.