

„Microsoft .NET“ – Evolution oder Revolution?



Michael Stal

Über kurz oder lang dürfte Microsoft .NET alle Softwareentwickler betreffen, deren Lösungen auf Microsoft-Technologien beruhen. Der Artikel führt in die Bestandteile von Microsoft .NET ein. Anschließend stehen die Interoperabilität von .NET mit existierender Software sowie mögliche Auswirkungen auf zukünftige Software und existierende Produkte im Vordergrund.

Vor kurzer Zeit hat Microsoft Einblick in die zukünftige Evolution seiner Plattformstrategie gewährt. Ob die innovative Plattform mit dem einprägsamen Namen „Microsoft .NET“ als Antwort oder Kampfansage an Sun Microsystems DOT-COM Initiative zu werten ist, sei an dieser Stelle dahingestellt. Jedenfalls stellt Microsoft .NET eine radikale Weiterentwicklung von Windows DNA 2000 („Distributed interNet Applications Architecture“) dar. Im Gegensatz zu DNA liefert .NET eine vollständige Plattform zur Entwicklung vernetzter (web-basierter) Anwendungen unter Nutzung beliebiger Programmiersprachen. Die .NET-Strategie hält dabei Einzug in die komplette Produktlandschaft von Microsoft, angefangen von Betriebssystemen bis hin zu Backoffice-Lösungen. Deshalb dürfte Microsoft .NET über kurz oder lang alle Softwareentwickler betreffen, deren Lösungen auf Microsoft-Technologien beruhen. Der Artikel führt in die Bestandteile von Microsoft .NET ein. Anschließend stehen die Interoperabilität von .NET mit existierender Software sowie mögliche Auswirkungen auf zukünftige Software und existierende Produkte im Vordergrund.

Michael Stal ist Mitarbeiter der Zentralabteilung Technik der Siemens AG in München. Er ist Mitglied der OMG, Chefredakteur von JavaSpektrum, Präsident des „International Java Club“ und Koautor der Bücher „Pattern-Oriented Software Architecture – A System of Patterns“ sowie „Pattern-Oriented Software Architecture, Volume II, Patterns for Concurrent and Networked Objects“.

Microsofts Vision

Bill Gates hat in seinem Vortrag auf dem Forum 2000 die Motivation für .NET sehr anschaulich erläutert: Ausgangspunkt für die neue Plattform sei die augenblickliche Entwicklung des Internet, dass sich mehr und mehr zur Basis aller Geschäftsaktivitäten entwickle. In der digitalen Welt nutzen physikalisch verteilte Personen unterschiedliche Medien, um Informationen zu transportieren, abzugeben oder abzufragen. Auf der Anwenderseite spielt der Web-Browser als Instrument für die Informationsverarbeitung die zentrale Rolle. Er zeichnet sich durch einfache Bedienbarkeit und Konnektivität aus, stellt auf der anderen Seite aber ein tastaturzentriertes, reines Lese-Werkzeug dar, mit dem der Anwender überwiegend passiv interagiert. Wünschenswert wäre hingegen ein Instrumentarium, mit dem sich personalisierte Informationen mit jedem beliebigen Endgerät und von jedem Ort über natürliche Bedien-

erschnittstellen darstellen und manipulieren lassen. Mit den Worten von Bill Gates: „Empower People through great Software anytime, any place, and on any device“.

Genau an dieser Stelle kommt die *extendible Markup Language (XML)* ins Spiel, die nicht nur eine adäquate Lösung für den Dokumentenaustausch, sondern ebenso für die Entwicklung verteilter Applikationen bietet. XML hat heute entscheidenden Einfluss auf nahezu alle Softwaretechnologien. Mit .NET möchte Microsoft nun die Plattform für das Internet der nächsten Generation zur Verfügung stellen. Die Plattform (vgl. Abb. 1) umfasst Technologien für Clients, Server und Dienste. Sie unterstützt unterschiedliche Endgeräte, erlaubt Datenzugriff über Informationssysteme und schützt die Privatsphäre durch entsprechende Sicherheitsmechanismen.

Basiskomponenten

Die Evolution von Microsoft Windows auf .NET gestaltet sich nach der Vorstellung von Microsoft ebenso dramatisch wie einst der Übergang von MS-DOS auf Windows. Die grundlegenden Komponenten der .NET-Infrastruktur präsentieren sich dabei wie folgt:

- Basis der Infrastruktur ist die sogenannte .NET-Plattform, die fundamentale Systemdienste bereitstellt. Deren plattformneutrale Schnittstellen entkoppeln

Abb. 1:
In der .NET-Plattform bilden die offenen XML-Standards eine tragende Säule, auf der die komplette .NET-Infrastruktur ruht. Mit Hilfe dieser Infrastruktur ist die Entwicklung erweiterter Dienste möglich. Der Zugriff auf diese Dienste wiederum soll über beliebige Endgeräte möglich sein.



die Anwendungen vom zu Grunde liegenden Betriebssystem.

- Direkt auf der Plattform setzen erweiterte Dienste (*Building Blocks*) auf; im Detail sind das Dienste für Personalisierung, Authentisierung, Ereignismeldungen, Verzeichnis- und Suchdienste, Softwareverteilung, Kalenderdienste und Speicherung. Diese Dienste nutzen XML und stehen sowohl im Intranet als auch offline oder über das Internet zur Verfügung.
- Die Speicherung und Verwaltung von Daten erfolgt systemweit über eine XML-basierte Datenhaltung, den sogenannten *XML Store*. In der Praxis handelt es sich um ein XML-basiertes Frontend für Datenbankprodukte wie DB2, SQL Server oder Oracle.
- Auf Präsentationsebene bietet der sogenannte „Universal Canvas“ ein Medium, auf dem entweder web-basierte Oberflächen (*WebForms*) oder Windows-basierte Oberflächen (*WinForms*) aufsetzen. Statt „Benutzeroberfläche“ führt Microsoft in diesem Kontext den Begriff „User Experience“ ein. Die Bedienoberflächen sind kommunikationsorientiert, erlauben den personalisierten Informationszugriff von jedem Ort und integrieren innovative Interaktionstechnologien, wie z. B. Handschrifterkennung, Sprache oder Bilder. Speziell wegen neuer Hardwareentwicklungen, wie beispielsweise Breitbandkommunikation, Mobilgeräte, Smart-Cards, Pocket-PCs oder Bildtelefone, bedarf es angepasster und flexibler Bedienparadigmen.

.NET Server

Durch die .NET-Plattform strebt Microsoft eine Symmetrie zwischen Clients, Servern und Diensten an. Anders formuliert, können Cli-

ents flexibel als Server fungieren oder Server andererseits eine Client-Rolle übernehmen, ganz abhängig von der jeweiligen Anwendungssituation. Gerade in vernetzten Umgebungen muss dabei der Faktor Skalierbarkeit besondere Berücksichtigung finden. Zu diesem Zweck lassen sich dieselben Dienste auf unterschiedlichen Servern zur Ausführung bringen. Speziell *Application Service Provider (ASP)* finden dadurch Unterstützung.

Microsoft richtet auch seine eigene „BackOffice“-Produktlinie komplett an der .NET-Strategie aus. Dazu bringt das Unternehmen bestehende Server-Produkte in neuer Version heraus, fügt aber auch einige völlig neue Produkte hinzu. Zu den .NET-Enterprise-Servern, die alle mit dem Suffix „Server 2000“ firmieren, zählen unter anderem „BizTalk Server“, „SQL Server“, „Exchange Server“, „Host Integration Server“, „Commerce Server“, „Application Center“ sowie „Internet Security and Acceleration (ISA) Server“.

.NET Services

Die eigentliche Innovation von Microsofts neuem Ansatz besteht darin, dass Anwender beliebige Dienste netzweit von beliebigen Geräten und Standorten über Informationsagenten finden und nutzen können. Ein Offline-Arbeiten ohne Netzverbindung ist ebenfalls möglich. Dabei lassen sich die Informationen und deren Präsentation mit netzweit abrufbaren Profilen personalisieren. Dem Benutzer stehen intelligente innovative Browser mit natürlicher Bedienschnittstelle zur Verfügung.

Dabei ist jedoch zu beachten, dass Microsoft als Betriebssystem auf herkömmlichen Computern wie PCs und Servern ausschließlich Windows in allen seinen Varianten (NT, 2000, 9x, ME) vorsieht und sich lediglich bei den Kleingeräten, wie PDAs, Set-Top-Boxen, Handys usw., öffnet.

Softwareentwicklung

Die Hauptaufgabe des Softwareentwicklers in der .NET-Ära besteht darin, existierende Dienste über Workflows zu verbinden und zu integrieren (*Orchestration*), wobei die Beschreibung der Workflows über XML-Dokumente erfolgt (siehe das Produkt BizTalk Server). Fertige Applikationen sollen sehr einfach auf einer vorliegenden Infrastruktur installiert werden können, was auch in Hinblick auf ASP-Lösungen enorme Vorteile bietet. Eine dynamische Geräteintegration kann dabei über das ebenfalls XML-basierte *Universal Plug and Play (UPnP)* erfolgen.

Ein besonders wichtiges Konzept von .NET ist darüber hinaus der Webservice. Damit soll es möglich sein, dass die Informationen, die ein Webserver im Internet oder Intranet anbietet, nicht nur einem menschlichen Benutzer in Form einer textuellen oder grafischen Darstellung zur Verfügung stehen, sondern auch über definierte Schnittstellen aus einem Programm abfragbar sind (mittels des SOAP-Protokolls, siehe unten). Damit könnten der Informationsaustausch im Internet und die Möglichkeiten des Umgangs mit Informationen auf eine ganz neue qualitative Stufe gestellt werden.

Roadmap

Abbildung 2 veranschaulicht die *Roadmap* für alle unter .NET subsummierten Technologien.

Schon heute liegen Evaluierungsversionen wichtiger Bestandteile von .NET vor. Der „BizTalk Server 2000“ repräsentiert eine Integrationsplattform auf XML-Basis zur Entwicklung integrierter B2B-Lösungen. Der *Passport*-Dienst erlaubt eine netzweite Authentisierung von Benutzern. Im Jahr 2001 folgt die nächste Windows-Version (Codename „Whistler“), die sowohl Windows *Millenium Edition (ME)* als auch Windows 2000 zur Konvergenz führen soll. Die Programmierumgebung „Visual Studio“ folgt ebenfalls dem .NET-Konzept und umfasst die neue Programmiersprache C# (gesprochen: „C Sharp“)*. Schließlich soll ab dem Jahre 2002 eine komplette Realisierung der .NET-Infrastruktur vorliegen. Letztendlich werden alle Microsoft-Produkte in die .NET-Infrastruktur integriert.

	Today	2001	2002+
User Experience	Tech Preview	Windows .NET 1.0	Full .NET UE
Infrastructure and Tools	XML in servers BizTalk Server	Visual Studio 7	Windows .NET Server
Building Blocks	Passport	3 or 4 key services	Full Offer, Corporate Federation
User Services	Service experience	hCentral, MSN, Personal Sub	Office .NET, Visual Studio.NET

Abb. 2:
Aus der Roadmap von Microsoft .NET ist ersichtlich, dass bis zur Entwicklung aller Bestandteile noch mindestens zwei Jahre ins Land gehen werden.

*also einen Halbton höher als „C“; auf Deutsch: „Cis“.



Abb. 3: In SOAP werden Methodenaufrufe als Datenpakete in XML kodiert und als HTTP-Pakete über die Leitung verschickt. Neben HTTP sind beliebige Transportprotokolle als Transportmedium nutzbar.

Komponenten, die in MSIL-Code vorliegen, heißen *Managed Components*. Durch spezielle Methoden (`Platform/Invoke`) ist aber auch ein direkter Zugriff auf die Plattform möglich, zum Beispiel auf Code, der nicht mit der CLR erzeugt wurde. Ebenso sind Zugriffe existierender Applikationen auf *Managed Components* über COM-Wrapper vorgesehen. Damit ist die .NET-Welt interoperabel zur existierenden Windows-DNA-Landschaft.

SOAP

Das *Simple Object Access Protocol (SOAP)* haben die Firmen Userland, DevelopMentor, IBM und Microsoft beim „World Wide Web Consortium“ (W3C) zur Standardisierung eingereicht. Da in dieser Ausgabe ein eigener Artikel zu diesem Thema zu finden ist (siehe Seite 26), sollen hier ein paar grundsätzliche Informationen ausreichen.

Motivation für das Protokoll war ursprünglich folgendes Problem: Sobald Clients und Serverobjekte über Kommunikationsstandards wie CORBA oder DCOM interagieren wollen, benötigen sie dafür spezielle, dynamisch zugewiesene Kommunikationsendpunkte. Im Internet blockieren allerdings Firewalls aus Sicherheitsgründen die meisten Ports bis auf wenige internet-relevante Zugänge. Für das Webprotokoll HTTP (z.B. Port 80) sind diese Schutzmauern durchlässig. Um dieses Problem zu umgehen, haben Middleware-Hersteller proprietäre Lösungen entwickelt, um Methodenaufrufe und deren Ergebnisse huckepack über HTTP-Pakete zu verschicken. Diese Lösungen sind allerdings nicht interoperabel und bedürfen eines großen Konfigurations- und Administrationsaufwands. Ziel von SOAP ist es nun, auf Basis von XML ein allgemeines – also generisches – Datenaustauschprotokoll für entfernte Methodenaufrufe zu definieren (vgl. Abb. 3). Transportmedium für den Datentransport kann dabei HTTP oder jedes andere Transportprotokoll sein (beispielsweise sind auch Methodenaufrufe über E-Mail denkbar).

Die Nutzbarkeit von SOAP beschränkt sich übrigens nicht auf synchrone Kommunikation über Methodenaufrufe. Die Einbindung von MOM-Produkten (*Message-Oriented Middleware*) ist ebenfalls möglich. Mit SOAP ist somit die Firewall-Problematik gelöst. Darüber hinaus ist eine Interoperabilität zwischen verschiedenen Middleware-Herstellern möglich. So haben neben Microsoft zahlreiche CORBA- und Java-Hersteller ihre Unterstützung angekündigt bzw. zum Teil bereits im Angebot.

Die Technologien im Detail

Nachdem bisher die Vision und die Ziele von Microsoft .NET im Mittelpunkt gestanden haben, soll nun ein Blick auf die einzelnen technologischen Komponenten folgen, mit denen sich die Vision in die Praxis umsetzen lässt.

Common Language Runtime

Das Herz der .NET-Plattform besteht

- zum einen aus einem aus der *Common Language Runtime (CLR)* und
- zum anderen aus einem Basisframework mit grundlegender Funktionalität, wie etwa Ausgabefunktionen oder Ein-/Ausgabe-Zugriffe.

Unter CLR verbirgt sich eine objektorientierte Laufzeitumgebung, mit der sich Code aus beliebigen Programmiersprachen integrieren lässt. Zu diesem Zweck übersetzen CLR-kompatible Programmierumgebungen die Programme zunächst in einen CPU-unabhängigen Zwischencode *MSIL (Microsoft Intermediate Language)* für eine theoretische virtuelle Maschine (Java lässt grüßen). Erst zur Laufzeit erfolgt mit Hilfe eines *Just-in-Time*-Compilers die Übersetzung in den jeweiligen nativen maschinenabhängigen Code. Dadurch erreicht .NET eine Entkopplung zwischen Anwendungssoftware und der darunter liegenden Systemumgebung. Ebenso lassen sich MSIL-Programmfragmente unabhängig von ihren Implementierungssprachen koppeln. Beispielsweise ist die Ableitung einer Visual-Basic-Klasse aus einer C++-Klasse möglich, die dann wiederum ein Java-Programmierer nutzen kann. CLR-Unterstützung finden momentan die Programmiersprachen Visual Basic, C++ und C# (siehe nächster Unterabschnitt). Weitere Hersteller haben die Integration ihrer Programmiersprachen bereits angekündigt.

Klassen, die auf der CLR aufbauen, sind per se fähig, über Prozessgrenzen hinweg miteinander in Verbindung zu treten. Für den COM-Entwickler bedeutet das, dass jede Klasse in Visual Basic oder C# automatisch eine COM-Klasse ist. Jegliche Umwandlung oder Methodenaufrufe werden nur noch mit den Mitteln der Programmiersprache bewerkstelligt. Besonderer COM-Code (wie `CreateInstance`, `QueryInterface`) und universelle Austauschschnittstellen (beispielsweise `IUnknown` oder `IDispatch`) sind damit überflüssig.

Die Erzeugung von Metainformation erfolgt automatisch bei der Programmübersetzung. Anwendungen, wie z. B. Programmierumgebungen, sind damit in der Lage, „fremde“ Objekte über eine sogenannte *Reflection-API* dynamisch kennen zu lernen und einzubinden. Die Auslieferung der Software erfolgt in *Assemblies* (nach Microsoft eine „group of resources and types“), die als Einheiten der Verwaltung und Wiederverwendung fungieren. Ähnlich wie Java-Archive erhalten *Assemblies* eine Manifestdatei mit der Beschreibung aller Bestandteile. Für die Installation von Paketen ist im Gegensatz zu COM/COM+ kein Eintrag in der *Registry* notwendig. Dadurch lässt sich die gefürchtete „DLL Hell“ (DLL-Hölle) umgehen, bei der unterschiedliche Applikationen unterschiedliche Versionen desselben Pakets unsichtbar für den Anwender installieren und damit massive Probleme verursachen können. Bei .NET genügt es, die Pakete in einem Verzeichnis zu installieren oder sie zu löschen.

Der CLR liegt ein eigenes objektorientiertes Typsystem mit dem Namen *Common Type System* zu Grunde, das die Sicht auf Objekte beschreibt, also zum Beispiel Basistypen und objektorientierte Konzepte, wie etwa Klassen oder Instanzen. Neben den üblichen Feldern (*Fields*) und Methoden (*Methods*) können Klassen auch Eigenschaften (*Properties*) und Ereignisschnittstellen (*Events*) definieren.

„Visual Studio .NET“ und C#

Mit C# stellt Microsoft eine objektorientierte Programmiersprache vor, deren Ursprünge offensichtlich in C++, Java, Delphi und C liegen. Während Visual Basic als Systemsprache für COM und COM+ gedient hat, präsentiert sich C# als die Systemsprache für Microsoft .NET. C# offeriert Typsicherheit, integrierte automatische Speichereinigung sowie Unterstützung für Versionierung. Ein einfaches C#-Beispiel soll die Verwandtschaft zu C++, Delphi und Java verdeutlichen:

```
using System;
class HelloWorld {
    public static int Main(String[] args)
    {
        Console.WriteLine("Hello, World!");
        return 0;
    }
}
```

C# ist Bestandteil der Programmierumgebung Visual Studio .NET, in der übrigens keine Unterstützung für Java mehr vorgesehen ist. Dafür erfährt Visual Basic zahlreiche Neuerungen, darunter die ersehnte Erweiterung um zusätzliche objektorientierte Konzepte, wie beispielsweise Implementierungsvererbung.

Die neue Programmierumgebung stellt eine radikale Abkehr von den Vorgängerversionen dar. In der neuen Version finden sich ausgefeilte visuelle Werkzeuge zur Entwicklung Web- oder Windows-basierter Oberflächen. Mittels architektonischer Schablonen lassen sich Programmierkonventionen in Projekten durchsetzen (*Visual Studio Extension Frameworks*). Basisbibliothek in Visual Studio ist das .NET-Framework, als integriertes Laufzeitsystem dient CLR. Dienste lassen sich sehr einfach als Web-Dienste zur Verfügung stellen. Dazu kann der Ent-

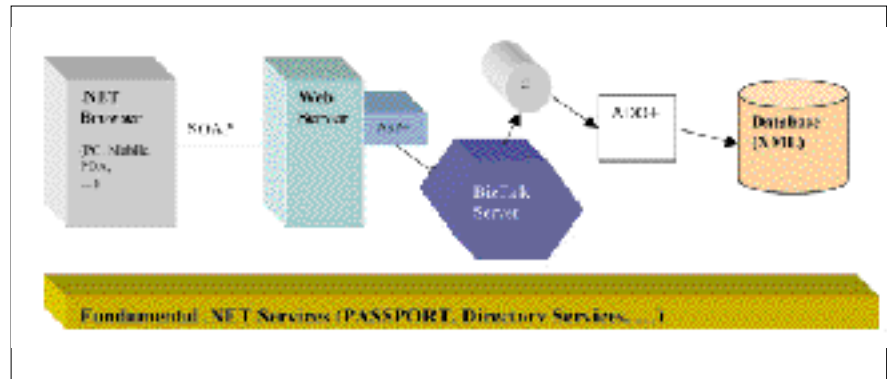


Abb. 4: .NET erlaubt die Entwicklung web-basierter Anwendungen als Multitier-Architekturen. Auf jeder Ebene der Architekturen finden Technologien aus dem .NET-Portfolio Anwendung. Globale Dienste lösen allgemeine Probleme, wie Personalisierung oder Sicherheitsaspekte.

wickler über einen speziellen *Wizard* sogenannte ASP+-Web-Seiten (siehe Abschnitt über ASP+) generieren, die sich genauso einfach editieren lassen wie Visual-Basic-Forms. In der neuesten Version der *Active Template Library (ATL)* ist die Kreation sogenannter ATL-Server möglich, die sich über *ISAPI-Schnittstellen (Internet Information Server API)* an den Microsoft-Web-Server anschließen lassen.

Insgesamt betrachtet entwickelt sich Visual Studio .NET somit zu einem leistungsfähigen Werkzeug für Geschäfts- und Web-Integration.

ADO+

ADO+ dient zum Zugriff auf Datenbanken und ergänzt die Vorgängerversion *ActiveX Data Objects (ADO)* um Plattforminteroperabilität und skalierbaren Datenzugriff. Der Datenaustausch zwischen Datenbank, *Middle-Tier* und Benutzerschnittstelle erfolgt mittels eines XML-basierten Persistenz- und Datenübertragungsformats. Innerhalb des *Middle-Tiers*, der die Geschäftslogik enthält, lassen sich sogenannte *Datasets* nutzen, um

einen Abzug der Datenbank im Hauptspeicher zwischenspeichern.

ASP+

Mit *Active Server Pages (ASP)* lassen sich schon jetzt sehr flexible Web-Anwendungen entwickeln. ASP-Seiten enthalten neben HTML-Anweisungen auch Code, der zur Laufzeit am Server ausgeführt wird. Dieser Code kann zum Beispiel Anweisungen enthalten, um auf COM-Objekte oder Datenbanken zuzugreifen. Als Ergebnis bekommt der Benutzer im Web-Browser eine dynamisch aktualisierte Web-Seite zu sehen. Im Gegensatz zu ASP bietet ASP+ die Möglichkeit, vorkompilierte ASP-Seiten abzulegen. Dadurch entfällt die zeitaufwendige Interpretation zur Laufzeit. Neben Visual Basic und JavaScript, lassen sich dadurch auch C++ oder C# einsetzen. ASP+-basierte *WebForms* erlauben die Integration von Benutzer-Kontrollelementen auf Webseiten und deren Verbindung zu server-basiertem Code. Dadurch ist es dem Entwickler möglich, Web-Seiten visuell so (bequem) zu konstruieren, wie dies bisher nur bei Visual Basic möglich war.

OBJEKTspektrum ist eine Fachpublikation des Verlags:

SIGS Conferences GmbH

Odenthaler Str. 19 · D-51465 Bergisch Gladbach

Tel. 02202/9372-0 · Fax: 02202/9372-2

E-mail: infogmbh@sigs.de · <http://www.sigs.de> und www.sigs.com



Ein weiterer damit verbundener Vorteil ist, dass ASP+ die Trennung von Programm- und Grafikinhalte (Layout) erlaubt. Damit können beide Teile von den jeweiligen Experten getrennt entwickelt und gepflegt werden.

BackOffice- und Integrationslösungen

Serverprodukte, wie z. B. Host Integration Server, BizTalk Server, Exchange Server und die Office-Produkte, werden in die .NET-Strategie integriert. Speziell der BizTalk Server bietet eine Plattform, um B2B-Integration über XML zu betreiben und dabei über Konnektoren auf existierende *Legacy*-Applikationen – wie z. B. ERP-Produkte á la SAP R/3, Baan, Peoplesoft oder CRM-Produkte (*Customer Relationship Management*) und SCM-Produkte (*Supply Chain Management*) – zuzugreifen.

Gesamtbild

Aus architektonischer Sicht lassen sich verteilte Anwendungen am geeignetsten anhand eines *Multi-Tier*-Ansatzes veranschaulichen (vgl. Abb. 4).

Der Browser vermittelt auf einem .NET-fähigen Endgerät die „User Experience“, ermöglicht also die Interaktion zwischen Benutzer und Web-Dienst. Zur Kommunikation von Anwendungen auf dem Client mit den Web-Diensten auf dem Server ist SOAP als XML-basiertes Kommunikationsprotokoll verfügbar. Im Server erfolgt die Verarbeitung über ASP+-Web-Seiten. Zur Integration von Applikationen dient die *Orchestration Engine* des BizTalk Servers, der zum Beispiel Konnektoren zu *Legacy*-Applikationen und zu COM-Funktionalität bietet. Die Verwendung von COM-Funktionalität ist aber auch direkt möglich, beispielsweise über ASP+-Seiten. Systemweit sind die fundamentalen Dienste, wie zum Beispiel *Passport*, auf allen Ebenen der Architektur nutzbar. Durch den .NET-Ansatz lassen sich bisher passive Web-Seiten miteinander zu komplexen Web-Diensten vereinigen.

Auf allen Ebenen spielt XML die entscheidende Rolle. So werden zum Beispiel Web-Dienste über XML-Dokumente beschrieben. Clients sind dadurch in der Lage, diese Dienstbeschreibungen zu finden und daraus dynamisch Methodenaufrufe zu konstruieren. Zur dynamischen Konfiguration der Dienstlandschaft dient UPnP, das ebenfalls auf XML und IP beruht.

Zur Entwicklung einer .NET-Anwendung bedarf es einer entsprechenden Werkzeugkette (Visual Studio .NET) und eines geeigneten Entwicklungsprozesses.

Interoperabilitätsfragen

Durch die Interoperabilität mit existierenden Technologien, beispielsweise COM und MFC, ergeben sich für Softwareentwickler zunächst keine Probleme. Bereits getätigte Investitionen erscheinen mittelfristig gesichert, zumal Microsoft seine Kompatibilitätsversprechen bisher immer eingehalten hat. Auf der anderen Seite führt die Nutzung der Microsoft .NET-Infrastruktur zu gravierenden Änderungen. Das ist für Neuentwicklungen (bis auf den erforderlichen Einarbeitungsaufwand) relativ unproblematisch. Sollen allerdings existierende Anwendungen mit Microsoft .NET web-tauglich gemacht werden, sind größere Aufwände an Ressourcen erforderlich. Das gilt aber generell für alle *EAI (Enterprise Application Integration)*-Ansätze.

Alternativen

Mit Microsoft .NET tritt Microsoft in direkte Konkurrenz zur *Java 2 Enterprise Edition (J2EE)*. Während J2EE seinen großen Erfolg auf Grund der Plattformunabhängigkeit erzielt, bleibt Microsoft .NET trotz Abkapselung von Windows an eine proprietäre Plattform gebunden. Auf der anderen Seite ermöglicht .NET die Integration beliebiger Programmiersprachen, wobei C# als optimale Systemsprache für .NET-Anwendungen ausgelegt scheint und sich dort als Implementierungssprache durchsetzen dürfte. Dahingegen beschränkt sich J2EE ausschließlich auf Java, bietet aber eine Interoperabilität mit anderen Programmiersprachen über CORBA. Beide Ansätze liefern ein Basisframework zum Zugriff auf die zu Grunde liegende Betriebssystemplattform. In Java sind dies die Basisbibliotheken, in .NET entsprechende Kernkomponenten. Den *Active Server Pages (ASP+)* auf Microsoft-Seite stehen *Java Server Pages (JSPs)* und Servlets gegenüber.

Vorteil von Microsofts Ansatz ist die Durchgängigkeit von .NET. Vom Betriebssystem bis zu BackOffice-Lösungen erfolgt eine komplette .NET-Integration. Vorteil von Java ist die Verfügbarkeit unterschiedlicher Implementierungen für un-

terschiedliche Plattformen, ganz im Gegensatz zur engen Herstellerbindung durch die .NET-Infrastruktur. Alles in allem gelten auch für die Entscheidung „.NET versus Java“ die gleichen Fakten und Kriterien wie bisher.

Zusammenfassung

Microsoft .NET impliziert einen kompletten technologischen Umbruch in der gesamten Welt der Microsoft-Technologien. Es handelt sich aber nicht um eine einzelne isolierte Technologie, sondern um eine Palette unterschiedlicher Technologien, deren Fundament primär auf Web-Techniken und XML beruht. Ziel von .NET ist die Bereitstellung einer komplett web-basierten Produktlandschaft, angefangen vom Betriebssystem bis zum BackOffice-Bereich. Stimmen in der Fachpresse, die bereits den Tod von COM/COM+ prognostiziert hatten, waren zu voreilig, zumal die Interoperabilität der .NET-Plattformen mit existierenden Lösungen gewährleistet ist. Microsoft möchte mit seinem Ansatz auf die zunehmende Bedeutung des Internet und die Integration mobiler und eingebetteter Systeme reagieren. Informationen sollen sich überall und mit jedem beliebigen Endgerät verarbeiten und darstellen lassen. Wer heute schon Microsoft-Technologien im großen Umfang nutzt, sollte sich deshalb frühzeitig mit .NET beschäftigen. Für eine genaue Bewertung der Tragfähigkeit und Durchgängigkeit des Ansatzes ist es indes momentan noch zu früh. Erste konkrete Produkte sind nicht vor Sommer 2001 zu erwarten. Nutzer alternativer Technologien (Java, CORBA) sollten daher abwarten, ob sich ein Umstieg wirklich lohnt.

Literatur

[Gru00] D. Grzuntz, C# – Microsoft schlägt neue Töne an, in: OBJEKTSpektrum 5/2000

[Mic00a] Microsoft, MSDN Magazin, Vol. 15, No. 9, Oktober 2000

[Mic00b] Microsoft, MSDN Magazin, Vol. 15, No. 10, Oktober 2000

[Mic00c] Microsofts Website zum Thema .NET, siehe <http://www.microsoft.com/net>

[O'Re] Verlagshaus O'Reilly, Vergleich J2EE mit .NET, siehe

http://java.oreilly.com/news/farley_0800.html

[Sta00] M. Stal, Pränante Post (SOAP), in: iX 5/2000