

Studiengang: Softwaretechnik
Prüfer: Prof. Dr. rer. nat. K. Rothermel
Betreuer: Dr. phil. nat. C. Becker

begonnen am: 01. Februar 2002
beendet am: 31. Juli 2002
CR-Klassifikation: D.2.6, D.2.11

Fachstudie Nr. 6

Bewertung der Microsoft .NET-Strategie

Daniel Friedrich Sven Stitzelberger

Institut für Parallele und
Verteilte Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20-22
D-70565 Stuttgart

Inhaltsverzeichnis

I Grundlagen	4
1 Einleitung	4
2 Die Architektur von .NET	5
2.1 Common Language Runtime (CLR)	5
2.1.1 Laufzeitumgebung	6
2.1.2 CTS	7
2.1.3 Performance	7
2.1.4 Sicherheitsmodell	8
2.2 Framework	9
2.2.1 Grundframework	9
2.2.2 Framework-Erweiterungen	10
2.3 Assemblies	10
2.3.1 Konzept	11
2.3.2 Komponentenmodell	12
2.4 Programmiersprachen und Compiler	13
2.5 Anwendungsszenarien	13
3 .NET My Services	15
4 C#	15
5 Vor- und Nachteile der .NET Strategie	16
6 Einordnung von .NET	16
6.1 .NET als Komponentenmodell	16
6.2 .NET für GUI-Anwendungen	17
6.3 .NET als Web-Services Plattform	17
6.4 .NET als IT-Infrastruktur	17
II Einsatzfähigkeit	18
7 Einsatzfähigkeit von .NET bei ARS NOVA	18
7.1 Klassifikation von Anwendungen	18
7.1.1 Stand-alone-Anwendungen	18
7.1.2 FAT-Clients	18
7.1.3 THIN-Clients	19
7.2 Zusätzliche Aspekte der Server-Seite	19
7.2.1 Geschäftsobjekte	19
7.2.2 BackOffice-Anwendungen	20

7.2.3	Web-Services	20
7.3	.NET-Unterstützung der Anwendungs-Klassen	20
7.4	Evaluation der .NET-Komponentenverfügbarkeit	20
7.5	Ableitung der Einsatzfähigkeit bei ARS NOVA	21
8	Vergleich der Entwicklungsumgebung	22
8.1	IBM Websphere Studio Application Developer	22
8.2	Microsoft Visual Studio .NET	23
8.3	Vergleich beider Entwicklungsumgebungen	24
9	Anhang	25
9.1	„ADIS“ — FAT-Client-/Server-Anwendung	25
9.1.1	Beschreibung und Klassifikation	25
9.1.2	Komponentenvergleich	26
9.2	„KoMo“ — THIN-Client-/Server-Anwendung	28
9.2.1	Beschreibung und Klassifikation	28
9.2.2	Komponentenvergleich	28
III	Einführungsstrategien	31
10	.NET-Einführungsstrategien	31
10.1	Beschreibung der Einführungsprojekte	31
10.2	Merkmale der Einführungsprojekte	32
11	Voraussetzung zur Einführung bei ARS NOVA	32
11.1	Werkzeuge	33
11.2	Sprache(n)	33
11.3	Technologie-Umfeld	34
12	Vorschlag für die Einführung bei ARS NOVA	34
13	Zusammenfassung	34

Zusammenfassung

Dieses Dokument stellt das schriftliche Ergebnis der Einarbeitung in die Grundlagen der .NET-Strategie im Rahmen der Fachstudie zum Thema Bewertung der .NET-Strategie dar. Ziel der Fachstudie ist die Erstellung eines Überblicks über die Gesamtstrategie Microsofts, die Feststellung der Tauglichkeit für bei ARS NOVA durchgeführte Projekte, sowie schließlich die Erarbeitung einer möglichen Einführungsstrategie.

Teil I

Grundlagen

1 Einleitung

Microsoft hat Mitte des Jahres 2000 seine .NET-Strategie bekannt gegeben. Inzwischen sind erste Endprodukte im Handel erhältlich. Ziel dieses Dokumentes ist es, den derzeitigen Stand der .NET-Strategie und diejenigen Teilaspekte, die für das Unternehmen ARS NOVA Software GmbH, interessant sind, genauer zu evaluieren und zu bewerten. Die .NET-Strategie besteht aus drei Eckpfeilern:

- .NET My Services:
Die .NET My Services stellen Web-Services im B2C (Business-to-Customer) Bereich dar. Sie zielen somit direkt auf potentielle Endnutzer ab und sind dadurch für die ARS NOVA in ihrer derzeitigen Form wenig interessant. .NET bietet zwar auch sehr gute Unterstützung für die Entwicklung von Web-Services mit anderer Zielsetzung (z.B. B2B) aber diese werden von der Strategie quasi als Anwendungen betrachtet und fallen nicht in den Bereich .NET My Service.
- .NET Enterprise Server Family:
Unter den .NET Enterprise Servern versteht Microsoft die neueste Generation ihrer Serverprodukte, z.B. den SQL-Server oder der BizTalk-Server. Die .NET Enterprise Server wurden im Gegensatz zu ihren Vorgängern um spezielle Funktionen erweitert, die das Zusammenspiel mit .NET-Anwendungen erleichtern sollen. Ein interessantes Produktmerkmal ist die Tatsache, dass für den Datenaustausch bei allen .NET Servern der XML-Standard SOAP verwendet werden kann.
- .NET-Plattform:
Mit der .NET-Plattform wird die Ausführungsumgebung bezeichnet. Diese besteht aus der Common Language Runtime (CLR), die ein gemeinsames Typsystem für alle .NET-Sprachen bereitstellt, und einem gemeinsamen Framework. Anwendungen können in (nahezu) beliebigen Programmiersprachen implementiert werden. Die Verwendung einer Zwischensprache, im Zusammenspiel mit der CLR, ermöglicht die Interoperabilität aller unterstützten Sprachen. Außerdem wird mit der .NET-Plattform ein neues Komponentenmodell eingeführt, das (D)COM auf lange Sicht ablösen soll.

Im Folgenden werden die .NET Enterprise Server und die .NET-Plattform zusammenhängend betrachtet, während die .NET My Services in einem eigenen Kapitel lediglich kurz vorgestellt werden. Auch die neue Sprache C# wird in einem eigenen Kapitel kurz erläutert. Da die Plattform aber sprachunabhängig ist, ist C# weder zum Verständnis noch zur Verwendung zwingend notwendig. Unabhängig von dieser

Einschätzung muss man derzeit aber anmerken, dass C# aufgrund seiner Entstehungsgeschichte die Eigenschaften der .NET-Plattform am besten unterstützt [27].

2 Die Architektur von .NET

Wie in der Einleitung erwähnt wurde, ermöglicht es die .NET-Plattform Anwendungen in nahezu beliebigen Sprachen zu entwickeln. .NET sieht hierfür eine Zwischensprache, Intermediate Language (IL), vor. Um eine Sprache auf die .NET-Plattform zu portieren muss lediglich ein Compiler verwendet werden, der aus der Ausgangssprache IL-Code und bestimmte Metadaten erzeugt. Diese Daten werden in sogenannte Assemblies abgelegt, die von der Laufzeitumgebung zur Ausführung gebracht werden. Die Betriebssystem-Funktionalität wird von der CLR auf die spezifische Implementierung des zugrundeliegenden Betriebssystems abgebildet. Derzeit unterstützt werden von Microsoft alle Windows-Versionen sowie eine auf dem ECMA-Standard [9] basierende FreeBSD-Version. Für Linux wird im Rahmen des Mono-Projekts [19] eine Laufzeitumgebung entwickelt.

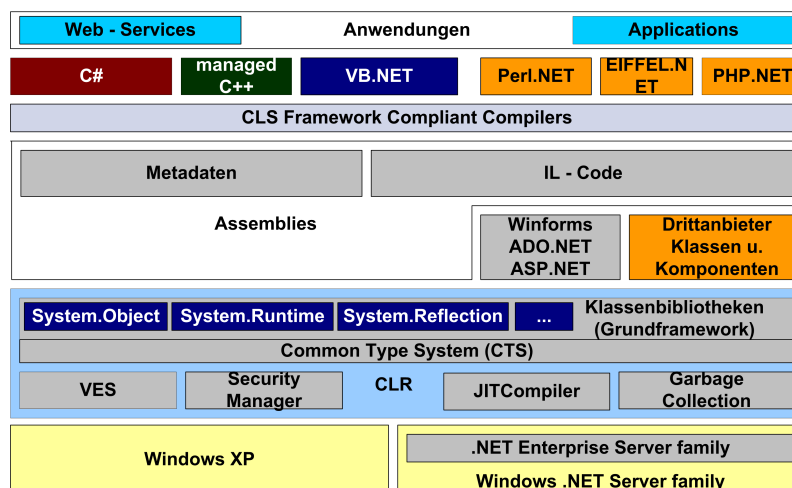


Abbildung 1: Die .NET-Architektur

Die einzelnen Bestandteile der in Abbildung 1 skizzierten .NET-Architektur werden im folgenden vorgestellt.

2.1 Common Language Runtime (CLR)

Alle .NET-Anwendungen werden von der CLR ausgeführt. Die auf der CLR ausführbaren Programme bestehen aus IL-Code, den die Compiler der einzelnen .NET-Sprachen erzeugt haben. Die CLR besteht aus den Bestandteilen Common Type System, Virtual Execution System, Security Manager, Just-In-Time-Compiler und Garbage Collection.

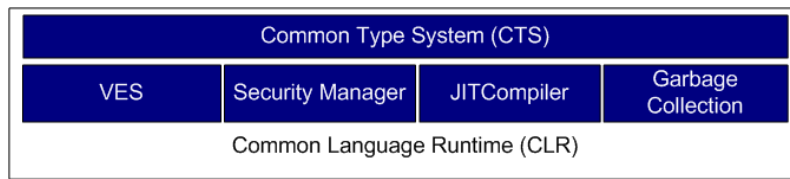


Abbildung 2: Die Common Language Runtime

Die Integration des gemeinsamen Typsystems in die CLR ermöglicht das Arbeiten ohne Typkonvertierung über die Sprachgrenzen der .NET-Sprachen hinweg. Ein weiterer interessanter Punkt ist die Integration der Garbage Collection in die CLR. Diese steht allen Sprachen, auch jenen, die bisher nur eine manuelle Speicherverwaltung unterstützten, zur Verfügung. Im folgenden wird die CLR, in die beiden Aspekte Laufzeitumgebung und Typsystem getrennt, vorgestellt.

2.1.1 Laufzeitumgebung

Die Laufzeitumgebung umfasst die in Abbildung 2 auf einer Ebene angesiedelten Komponenten Virtual Execution System (VES), Security Manager, JIT-Compiler und Garbage Collection. Diese kooperieren während der Ausführung einer Anwendung. Das VES führt, unter Berücksichtigung der Sicherheitsrichtlinien des Security Managers, nativen Maschinencode aus. Die Übersetzung von IL-Code in Maschinencode wird vom JIT-Compiler durchgeführt, auch Jitting genannt. Standardmäßig geschieht das Übersetzen beim ersten Aufruf einer Routine. Der Maschinencode wird für weitere Aufrufe gespeichert. Es steht aber auch die Möglichkeit offen, bereits bei der Installation einer Anwendung den kompletten Code zu übersetzen. (Maschinen)Code der in der VES abläuft wird „Managed Code“ genannt. „Managed Code“ deshalb, da die Laufzeitumgebung neben Ausnahmenbehandlung auch automatische Speicherverwaltung durch eine Garbage Collection bietet. Des weiteren greifen nur bei „Managed Code“ die Sicherheitsrichtlinien. Die Garbage Collection erfordert von einigen Sprachen wie zum Beispiel C/C++ eine Einschränkung ihrer Fähigkeiten. Man nennt die .NET Version von C++ deshalb auch managed C++. Aufgrund der Garbage Collection muss man hier zum Beispiel auf Zeigerarithmetik verzichten. Werden für die Entwicklung einer Anwendung diese Eigenschaften trotzdem benötigt, kann die CLR auch sogenannten unmanaged Code ausführen. Dieser läuft dann nicht mehr unter der Oberaufsicht der Garbage Collection ab. Innerhalb einer Anwendung kann, wenn die Sicherheitsrichtlinien dies gestatten, zwischen beiden Modi gewechselt werden. Im ECMA-Standard [9] wird in Partition I der genaue Aufbau der VES spezifiziert, so dass diese für beliebige Plattformen implementiert werden kann.

2.1.2 Common Type System (CTS)

Auf die im vorherigen Kapitel beschriebene Laufzeitumgebung setzt das gemeinsame Typsystem (CTS) auf. Dieses legt den Grundstein für die Sprachinteroperabilität zwischen den verschiedenen Programmiersprachen in .NET. Der Compiler für die jeweilige Programmiersprache unter .NET übernimmt das Abbilden der spracheigenen Typen auf die des CTSs, so dass die Verwendung für Entwickler völlig transparent ist. Im optimalen Fall können in jeder Sprache die Typen unter ein und dem selben Namen erreicht werden. In .NET stammen alle Typen von einer Wurzel, System.Object, ab. Alle Typen die man über das CTS definiert und damit von System.Object ableitet, sind Objekte und heißen Managed Types, da sie in der CLR ablaufen. Man unterscheidet in .NET zwei Arten von Typen, den Referenztypen, und den Wertetypen. Neben den Typen definiert die CTS auch einige Regeln wie z.B. Sichtbarkeitsbereiche von Klassen oder die Vererbung. In allen .NET-Sprachen ist man an die Einfachvererbung gebunden. Das gemeinsame Typsystem erlaubt auch Vererbung über die Programmiersprachengrenze hinweg. Von einer in C# geschriebene Klasse kann also eine in VB.NET geschriebene Klasse abgeleitet werden. Programmiersprachenspezifische Vorteile, wie z.B. die besonders performante String-Verarbeitung bei Perl, fallen durch die Transformation von Perl auf den IL-Code weg.

2.1.3 Performance

Die Performance einer .NET-Anwendung hängt bei einer optimalen Implementierung in großem Umfang direkt von der Performance der CLR ab. Die Architektur der CLR bietet mindestens zwei potentielle Angriffspunkte für Performacekritiker. Diese sind die Verwendung einer Zwischensprache und eines Garbage Collectors. Die Verwendung der Zwischensprache bietet nur auf den ersten Blick einen Angriffspunkt. Wie bereits erwähnt wird jede Methode bei ihrem ersten Aufruf in nativen Maschinencode übersetzt und anschließend für weitere Aufrufe zwischengespeichert. Das heißt, dass eine Anwendung zu Beginn langsamer laufen wird als wenn sie direkt in Maschinencode vorhanden wäre. Im Laufe der Zeit (im Sinne von Programmlaufzeit) aber wird die Anwendung immer schneller werden, da ja immer mehr Teile des Codes bereits in Maschinencode vorliegen. Darüber hinaus könnte die CLR weitere Optimierungen, abhängig von der Benutzungshäufigkeit einzelner Funktionen einer Anwendung, durchführen, so dass die Anwendung für den einzelnen Endanwender im Endeffekt schneller ist als wenn sie übersetzt ausgeliefert werden würde. Außerdem wäre es für die Zukunft darüber hinaus denkbar, dass der JIT-Compiler der CLR auf die konkrete Prozessor-Familie und sogar -Version optimieren könnte. Alternativ bietet das Tool NGEN (Native Image Generator) die Möglichkeit den gesamten IL-Code während der Installation in Maschinencode zu übersetzen. Dadurch wird dann zwar der Installationsvorgang gestreckt, die Anwendung ist aber von ihrem ersten Einsatz an in Maschinencode-Form vorhanden. Da noch keine gegenteiligen Berichte verfügbar sind, kann man davon ausgehen, dass die

Performance der Garbage Collection mit der von Java vergleichbar ist. Das Thema Performance kann in diesem Artikel in keiner Weise Vollständigkeit beanspruchen. Auch ist es im derzeitigen Entwicklungsstadium von .NET sehr schwer konkrete Aussagen über die Performance zu bekommen. Es empfiehlt sich also vor einem geplanten Einsatz von .NET-Performance-Messungen für den konkreten Anwendungsfall durchzuführen.

2.1.4 Sicherheitsmodell

Das Sicherheitsmodell von .NET gliedert sich in die Bereiche Verifikation und Zugriffs-Sicherheit. Unter Verifikation versteht man hierbei die Verifikation des IL-Codes sowie der Metadaten. Ein Stück IL-Code kann immer in eine von vier Klassen (ungültig, gültig, typsicher und verifizierbar) eingeteilt werden. Ungültig bzw. gültig beschreibt lediglich ob der Code der Spezifikation genügt und somit also in nativen Code übersetzbar ist. Typsicherer Code hält sich an den Schnittstellen-Vertrag zu fremdem Code, ruft also insbesondere nur „public“ Methoden auf. Verifizierbarer Code ist typsicherer Code, der über einen Algorithmus als solcher erkannt werden kann (nicht jeder typsichere Code kann per Algorithmus verifiziert werden). Interessant an dieser Aufteilung ist eigentlich nur, dass man einstellen kann ob der Code wirklich verifizierbar sein muss oder ob etwa die Typsicherheit genügt. Die Überprüfung erfolgt zur Übersetzungs-Zeit. Muss der Code verifizierbar sein, ist es aber nicht, wird bei Aufruf des entsprechenden Codes eine Exception ausgelöst. Die Zugriffs-Sicherheit selbst wird wiederum aufgeteilt in die nachweisbasierte (evidence-based) und die rollenbasierte Zugriffs-Sicherheit. Der nachweisbasierte Ansatz regelt, welcher Code ausgeführt werden darf, während der rollenbasierte Ansatz regelt, wer den Code ausführen darf. Der nachweisbasierte Ansatz berechnet beim Laden einer Assembly ein sogenanntes PermissionSet, also eine Menge von Berechtigungen. Diese Berechnung benötigt zum Einen eine Liste von Nachweisen (pieces of evidence) und zum Anderen die aktuellen Sicherheits-Einstellungen. Als Nachweise werden zum Beispiel die Zone (siehe Internet Explorer), die Site oder die URL von der ein Assembly kommt übergeben. Auch der Name, das Anwendungsverzeichnis oder etwa Zertifikate können als Nachweise dienen. Die Liste der möglichen Nachweise ist darüber hinaus beliebig erweiterbar. Die Sicherheits-Einstellungen definieren vier Policy Levels (diese sind ebenfalls erweiterbar), welche über die Management-Konsole einstellbar sind. Für weitere Vertiefungen sei auf [22] verwiesen.

Während der nachweisbasierte Ansatz dem Schutz von Ressourcen vor unberechtigtem Zugriff dient, verhindert der rollenbasierte Ansatz das Ausführen von bestimmtem Code durch einen unberechtigten Benutzer. Der hier verwendete rollenbasierte Ansatz entspricht dem in der Informatik bekannten Role-Principal-Ansatz [16].

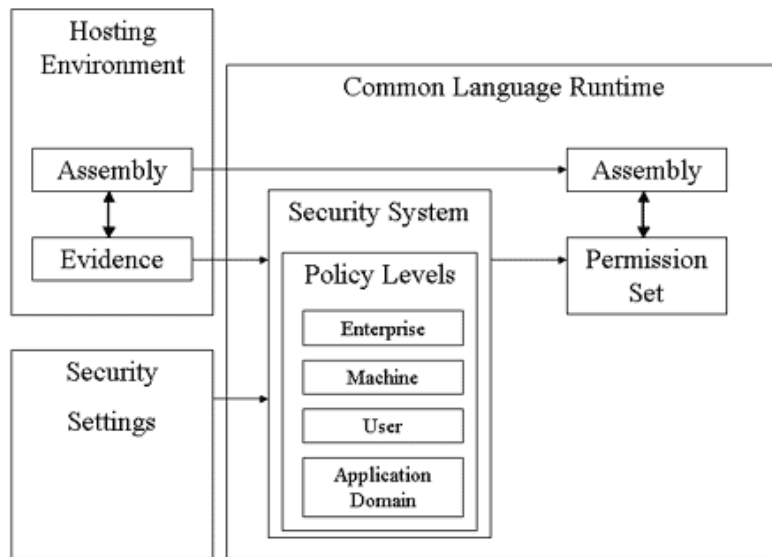


Abbildung 3: Berechnung der PermissionSets

2.2 Framework

Microsoft hat das Framework der .NET Architektur in zwei Teile aufgeteilt. Der Basis-Teil (Grundframework) ist bei der ECMA standardisiert worden, während die Framework-Erweiterung nur in der Microsoft-Version für die Windows-Plattformen enthalten sind.

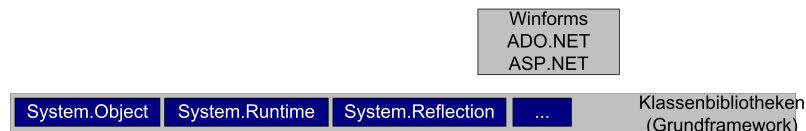


Abbildung 4: Framework

2.2.1 Grundframework

Das Grundframework stellt, neben dem Zugriff auf das CTS, Datenstrukturen wie Collection und Arrays, die Unterstützung von IO-Zugriffen, Serialisierung von Daten, Threads, Reflection, Kryptographie, die Security-Klassen und den Zugriff auf die Laufzeitumgebung bereit. Sieht man von den fehlenden Oberflächen-Klassen ab, dann umfasst der Standard in etwa den funktionalen Umfang von Java 1.0. Die Standardisierung der Klassenbibliotheken bedeutet nicht, dass sich Microsoft nun ganz aus der Entwicklung der grundlegenden Klassen verabschiedet hat. Im Rahmen der Evolution von .NET wird Microsoft auch hier weiter entwickeln.

2.2.2 Framework-Erweiterungen

Microsoft erweitert das standardisierte Framework, das nur die oben beschriebene rudimentäre Funktionalität bereitstellt, in seiner Implementierung um einige zusätzliche Bibliotheken. Mit diesen Bibliotheken wird das Framework zu einer umfangreichen und vor allem komfortablen Klassenbibliothek für die Windows-Plattform. Die wohl wichtigsten Erweiterungen dürften die drei Bibliotheken WinForms, ADO.NET und ASP.NET sein, welche architektonisch oberhalb des Grundframeworks einzuordnen sind (s. Abbildung 1). Zusätzlich zu diesen drei Klassenbibliotheken, deren Funktionalität im Anschluss näher erläutert wird, da sie wichtige Bausteine mitbringen um z.B. Client-Server-Anwendungen zu schreiben, werden von Microsoft noch die Bibliothek System.Drawing (für 2D-Graphik) und Erweiterungen zur Bibliothek System.XML für die Unterstützung von XPath und XSLT bereitgestellt. Die auf .NET abgestimmte Oberflächen-Bibliothek WinForms löst die bisherigen, auf die einzelnen Sprachen bezogenen, Bibliotheken, wie z.B. die MFC, ab. Die WinForms orientieren sich konzeptionell an Qt oder Java Swing. Die Verwendung der bisherigen Klassenbibliotheken ist in .NET-Anwendungen jedoch möglich. Mit ADO.NET kommt eine mächtige Bibliothek für den Datenzugriff hinzu. ADO.NET unterscheidet sich in wichtigen Punkten von seinem Vorgänger ADO-DB. Die Klasse DataSet ist der wichtigste Baustein von ADO.NET. DataSets sind Daten-Container in denen komplette relationale Datenbankstrukturen mit ihren Tabellen, Relationen und Constraints abgebildet werden können. Für die Nutzer eines DataSets ist es völlig transparent, ob sie nun aus einer Datenbank, über die .NET-Datenbankschnittstellen die sogenannten Managed-Provider oder aus einer XML-Datei gespeist werden. Die Strukturen eines DataSet lassen sich entweder von Hand programmieren oder automatisch erstellen, indem man ein XML-Schema einliest. XML-Schemas lassen sich auch zum Speichern der Daten in XML verwenden. ASP.NET bündelt die Web-Aktivitäten von .NET, angefangen mit den Web-Controls (kleine Oberflächenbausteine für Webseiten) über einfache Mechanismen für Dateizugriffe im Internet, bis zur Unterstützung der Web-Service Standards SOAP und WSDL. Setzt man die Entwicklung der Java-Klassenbibliotheken voraus, wird sich die Anzahl der zusätzlichen Klassen-Bibliotheken in Zukunft noch weiter erhöhen.

2.3 Assemblies

Komponentenmodelle sind in der heutigen Softwareentwicklung nicht mehr wegzudenken. In Anlehnung an die Objektorientierung ermöglichen sie Wiederverwendung, z.B. durch Kapselung, genaue Schnittstellendefinitionen und ihr binäres Format. Microsoft hat bisher auf diesem Gebiet den Zweig der COM-Technologien angeboten. Dieser war in alle Windows-Versionen integriert und führte dazu, dass viele Anwendungen auf COM oder dessen Nachfolgern aufbaut (unter COM werden im folgenden alle auf COM basierenden Komponenten-Technologien verstanden). COM hat aber gewisse Nachteile. Die umständliche In-

stallation inkl. der Registrierung einer COM-Komponente in der Registry dürfte dabei zu den kleineren Übeln gehören. Ein weiterer Punkt sind die Typkonversionen zwischen verschiedenen COM-Komponenten, die in unterschiedlichen Sprachen entwickelt wurden, auch wenn diese automatisch durchgeführt werden. Dieser Aspekt wird aber durch das Common Type System entschärft. Als sehr großes Problem hat sich die, unter dem Schlagwort DLL-Hölle bekannt gewordene, fehlende Versionierung von COM-Komponenten erwiesen. Mit COM war der Entwickler darüber hinaus gezwungen, seine Komponenten durch eine IDL (Interface Definition Language, Schnittstellenbeschreibung) zu definieren. Komponenten können nur mit Hilfe einer solchen Definition auf andere Komponenten zugreifen. Die Problematik der IDL-Definition wurde im Artikel [33] ausführlich behandelt. Ziel der Entwicklung von .NET war deshalb die Einführung eines geeigneten Konzepts zur Versionierung sowie einer einfacheren Installation von Komponenten. Beide Ziele werden durch die, jetzt Assembly genannten, Komponenten erreicht. Assemblies stellen somit den Nachfolger von COM dar. Die von den Compilern erzeugten und vom Entwickler erweiterbaren Metadaten, die die Schnittstelle einer Komponente beschreiben, sind fester Bestandteil einer jeden Assembly und lösen damit die IDL-Definitionen ab.

2.3.1 Konzept

Eine Assembly besteht aus mindestens einer Datei im Portable Executable (PE) Format [25]. Genau eine PE-Datei eines Assemblies enthält außer dem IL-Code und den Metadaten ein Manifest (wird im Unterschied zu den Typ Metadaten oft auch Assembly Metadaten genannt) und evtl. Ressourcen wie z.B. Grafiken. Den genauen Aufbau einer Assembly in UML-Notation enthält das Dokument unter [2]. Das Manifest enthält den Namen, die Versionsnummer, den Public-Key des Autors und eine Locale-Bezeichnung. Außerdem verweist das Manifest im Falle einer sogenannten Multifile-Assembly auf die dazugehörigen Dateien. Im Falle sogenannter Shared Assemblies, also Assemblies die von mehreren Anwendungen genutzt werden können, wird aus diesen Versionsinformationen bei der Installation ein eindeutiger Dateiname ermittelt. Es können also verschiedene Versionen der selben Komponente installiert sein. Obwohl hier von einer Installation die Rede ist, handelt es sich dabei um ein mit dem DOS-Befehl Xcopy vergleichbaren Vorgang. Denn um ein Assembly zu installieren muss dieses nur in den Global Assembly Cache kopiert werden. Eine aufwändige Installation mit Registrierung der Komponente entfällt somit. Dementsprechend ist das Deinstallieren mit einem Löschvorgang gleichzusetzen. Natürlich gilt dies nur für die shared Assemblies. Die sogenannten private Assemblies, die nur einer Anwendung zur Verfügung stehen, liegen in einem Unterverzeichnis des Installationspfades der Anwendung und müssen nicht dem strengen Versionierungsschema genügen. Will ein Anwendungsentwickler aus seiner Anwendung heraus eine shared Assembly einbinden, so muss er genau die oben genannten Informatio-

nen, also Name, Public-Key, Version und Locale, kennen. Somit kann er genau steuern, welche Version von welcher Komponente verwendet werden soll. Somit kann zum Beispiel auf Abwärtskompatibilität in Bibliotheken theoretisch verzichtet werden. Obwohl das genaue Versionierungsschema hier nicht erläutert werden soll, wird an dieser Stelle dennoch darauf hingewiesen, dass der viergliedrige Versionsnummer eine eindeutige Semantik zugrunde liegt [3]. Außerdem können durch Policies die festkodierte Versionsangaben nachträglich angepasst werden. Auch hierfür sei auf [3] verwiesen. Die Problematik der „DLL-Hölle“, eine einzige systemweite Instanz einer Komponente für alle Versionen, wird durch die Möglichkeit mehrere Versionen parallel vorzuhalten, beseitigt.

2.3.2 Komponentenmodell

Wie in den beiden vorherigen Abschnitten beschrieben wurde, wird mit den Assemblies ein neues Komponentenmodell eingeführt. .NET bietet die Integration bestehender Komponenten aus COM an. Die Interoperabilität zwischen .NET- und COM-Komponenten besteht in beide Richtungen. Dadurch ist es möglich, ein .NET-Assembly aus einer COM-Anwendung heraus zu verwenden und umgekehrt.

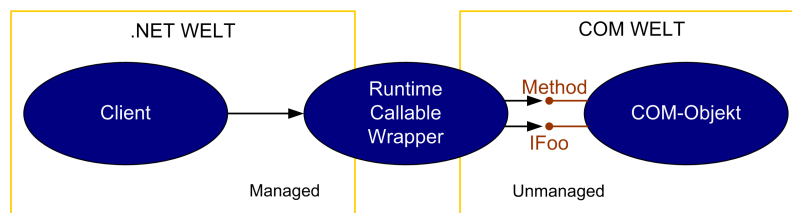


Abbildung 5: Interoperabilität zwischen .NET und COM

Die Abbildung 5 zeigt wie eine Anwendung aus der .NET-Welt heraus auf die beiden Methoden des COM-Objekts zugreifen kann. Hierzu benötigt man für das COM-Objekt entweder ein direkt vom Hersteller erstelltes Interop-Assembly oder man erstellt selbst ein solches. Dieses wird dann in das Assembly-Verzeichnis kopiert. Die Laufzeitumgebung erstellt dann einen Runtime Callable Wrapper der als eine Art Proxy-Klasse den Zugriff des .NET Clients auf das COM-Objekt übersetzt, so dass beide Seiten den Eindruck haben, dass sie mit einer Komponente aus ihrer Welt zusammen arbeiten.

Der umgekehrte Fall (Abbildung 6), d.h. der Zugriff von einer COM-Anwendung auf ein .NET-Objekt, funktioniert ähnlich. Auch hier wird das Interop-Assembly automatisch erzeugt. Der COM Callable Wrapper stellt dann der COM-Anwendung zur Laufzeit die öffentlichen Methoden und Attribute des .NET-Objekts zur Verfügung. über einen ähnlich aufgebauten Mechanismus besteht theoretisch auch die Möglichkeit neben DLLs auch andere Programme z.B. auch ERP-Anwendungen als Assembly zu kapseln [24]. SAP arbeitet aber bereits an einem Toolkit, das den Zugriff auf SAP-Systeme noch einfacher gestalten soll,

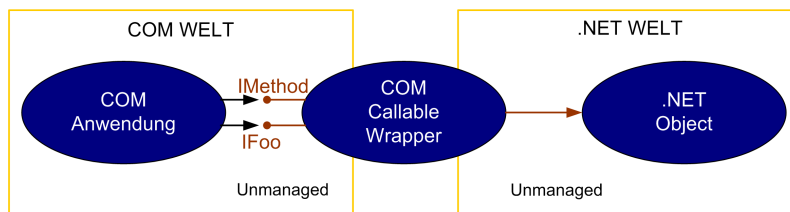


Abbildung 6: Interoperabilität zwischen COM und .NET

als dies bereits durch den vorgestellten Ansatz möglich ist.

2.4 Programmiersprachen und Compiler

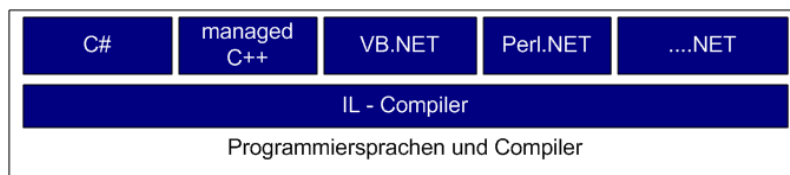


Abbildung 7: .NET-Programmiersprachen

Abbildung 7 verdeutlicht noch einmal, dass in .NET mit mehreren Programmiersprachen programmiert werden kann. Microsoft liefert hierzu die Programmiersprachen C#, VB.NET, managed C++, J# und Jscript mit. Von anderen Anbietern sind unter anderem die Sprachen Perl.NET, Python.NET, Eiffel.NET, COBOL.NET und Smalltalk verfügbar. Für diese Programmiersprachen gibt es einen Compiler, der die Sprache auf das .NET-Typsystem (CTS) und die .NET-Klassenbibliothek umsetzt. Will man ein besonders performantes Sprachkonstrukt aus der Sprache seiner Wahl verwenden, das aber keinen Pendant in der .NET-Klassenbibliothek besitzt, muss unmanaged Code eingesetzt werden.

2.5 Anwendungsszenarien

In diesem Abschnitt soll schematisch dargestellt werden, für welche Anwendungsszenarien die .NET-Plattform geeignet ist. Unterschieden wird zwischen Client- und Server-Anwendungen.

Der Bereich der Client-Anwendungen lässt sich in drei Klassen einteilen: Als erstes gibt es Stand-alone-Anwendung wie zum Beispiel Texteditoren o.Ä. Als zweites kann man sogenannte FAT-Client-Anwendungen ausmachen. Anwendungen also, die einen großen Teil ihrer Geschäftslogik auf dem Arbeitsplatz-Rechner ablaufen lassen und nur für spezielle Aufgaben, meistens Datenbank-Zugriffe, entfernte Dienste in Anspruch nehmen. Eine dritte Anwendungsklasse wären im Gegensatz dazu sogenannte Thin-Clients, die im Wesentlichen nur die Präsentationslogik auf dem Arbeitsplatzrechner ablaufen lassen und

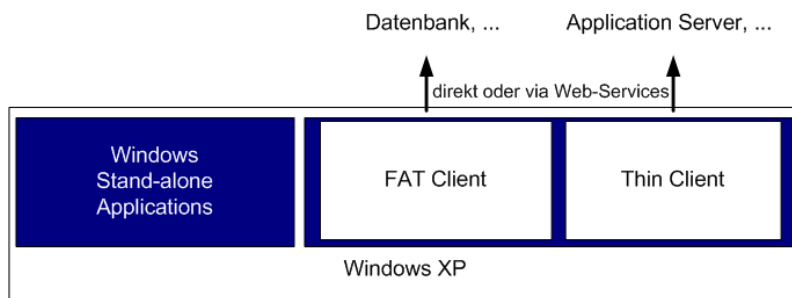


Abbildung 8: Client-seitige Anwendungsszenarien

die einzelnen Geschäftsvorfälle auf einem Anwendungsserver, der wiederum Server-Dienste verwenden kann, aufrufen. Alle diese Anwendungsklassen werden durch die .NET-Plattform unterstützt. Der zweite Teil des Kapitels betrachtet die serverseitigen Anwendungsklassen. Als Betriebssystem-Plattform wird beispielhaft aus der „Windows .NET Server Family“ der „Windows .NET Enterprise Server“ ausgewählt [38].

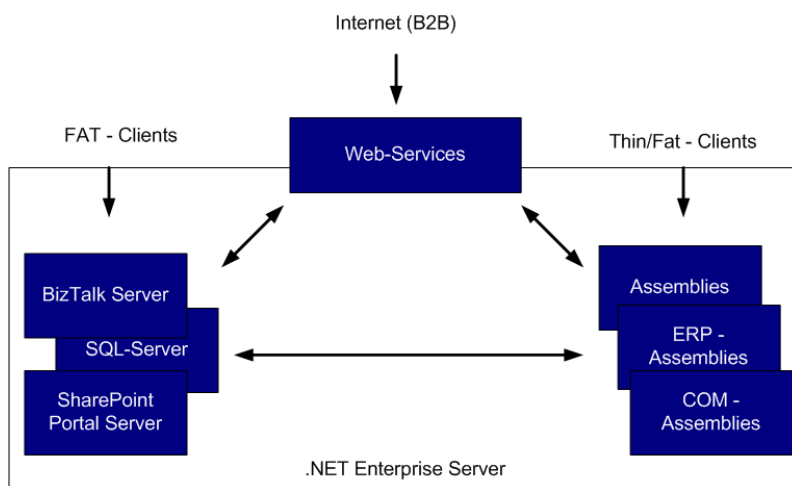


Abbildung 9: Serverseitige Anwendungsszenarien

Ein solches Betriebssystem kann als Plattform für Server-Programme wie die komplette .NET Server Familie [20] oder auch zum Beispiel ein Oracle-DBMS dienen. Darüber hinaus kann das Betriebssystem durch die integrierte .NET-Unterstützung quasi als Application Server dienen. Des weiteren können alle direkt verfügbaren Dienste auch über eine Web-Services-Schnittstelle verfügbar gemacht werden. Hier nicht dargestellt wird das sogenannte Compact Framework [32]. Damit werden Pocket PCs um eine .NET-Plattform ergänzt. Das Compact Framework enthält einen Großteil der Klassenbibliotheken und erlaubt somit die Implementierung von mobilen Anwendungen. Bis auf bekann-

te Einschränkungen dieser Handheld Devices hinsichtlich zum Beispiel der Display-Größe können damit ähnliche Anwendungsszenarien wie auf Client-Seite verwirklicht werden.

Durch Kombinationen der vorgestellten Szenarien kann man fast beliebige Modelle verteilter Systeme abdecken.

3 .NET My Services

Ein Web-Service ist ein Stück Code das mit einem anderen Stück Code über grundlegende Internet-Technologie kommuniziert. Ihnen wird für den B2B-Bereich (Business-To-Business, also Geschäfte zwischen Unternehmen) zukünftig eine große Rolle vorhergesagt [17]. Web-Services können mit dem .NET-Framework bzw. mit Visual Studio .NET sehr leicht erstellt und auch genutzt werden. Für den Anwendungsentwickler gibt es keinen Unterschied zwischen einem lokalen oder einem entfernten Routinen-Aufruf. Die .NET My Services stellen eine Sammlung von Web Services für den B2C-Bereich (Business-To-Consumer), basierend auf dem Microsoft'schen Authentifizierungs- und Personalisierungsdienst .NET Passport, dar. Letzterer existiert unter dem Namen Passport bereits seit einiger Zeit und dient der globalen Verwaltung von Benutzerdaten und der Verwendung dieser zur Authentifizierung bei Web-Diensten, wie z.B. bei EBay. Basierend auf .NET Passport wollte Microsoft eine Reihe von Diensten anbieten gegenwärtig ist allerdings nur .NET Alerts, ein Dienst zur Erinnerung an Termine, verfügbar. Inzwischen hat Microsoft seine Strategie für .NET My Services geändert. In Zukunft sollen diese Technologien in Form von Produkten an Kunden weitergegeben werden. Grund für diesen Wandel sind erhebliche Akzeptanz-Probleme auf Seiten potentieller Anbieter (zum Beispiel American Express) [33]. Da der B2C-Bereich für die Kunden der ARS NOVA Software GmbH auf absehbare Zeit keine Rolle spielen wird, soll hier auf eine genauere Untersuchung dieses Teils der .NET-Strategie verzichtet werden. Es soll aber darauf hingewiesen werden, dass der bereits existierende Dienst .NET Passport inzwischen von der US-Regierung als potentielles Authentifizierungs-System für US-Regierungs-Webseiten im Gespräch ist [21].

4 C#

Mit C# wurde eine neue Sprache der C/C++-Familie hinzugefügt. Grund war die Integration der von .NET eingeführten Konzepte in eine darauf abgestimmte Programmiersprache. Genauso wie das Grundframework wurde C# bei der ECMA [9] standardisiert und dementsprechend gibt es im Linux-Umfeld mit dem dotGnu-Projekt [7] und dem oben bereits erwähnten Mono-Projekt [19] bereits Bestrebungen C# auf andere Plattformen zu portieren. In wie weit C# Marktanteile auf der Linux-Plattform gewinnen kann bleibt abzuwarten. Auf der Windows-Plattform wird es sich im .NET-Umfeld seinen Platz zwischen VB.NET und managed C++ erobern [35]. Mehr Informationen zum

Vergleich der Sprachen C# und C/C++ sowie Java findet man in den Artikeln [36], [37], [28], [29], [30].

5 Vor- und Nachteile der .NET Strategie

Vorteile:

- Die CLR ermöglicht es Programme mit theoretisch jeder beliebigen Programmiersprache zu entwickeln.
- Durch das Jitting auf der Zielplattform wären für die Zukunft auch prozessorspezifische CLR's oder zumindest JIT-Compiler denkbar, so dass jeder Zielprozessor optimal ohne Programmanpassung ausgenutzt werden kann.
- Durch das Versionieren von shared Libraries wird die Verwaltung von Software-Installationen stark vereinfacht.
- Ein einheitliches, objektorientiertes und in die Laufzeitumgebung integriertes Typsystem macht Typkonvertierung zwischen verschiedenen Sprachen und Typverpackung innerhalb einer Sprache überflüssig.
- Die Garbage Collection vermeidet das Entstehen des wohl häufigsten Programmierfehlers, den Memory Leaks, bei Programmiersprachen wie z.B. C++.

Nachteile:

- Der wohl gravierendste Nachteil der .NET-Strategie ist die, zumindest zum jetzigen Zeitpunkt, fehlende Plattformunabhängigkeit. Diese entsteht dadurch, dass wichtige Klassenbibliotheken wie z.B. die WinForms oder ADO.NET nur für die Windows-Plattformen verfügbar sind. Ob aus technischer Sicht z.B. eine Portierung der WinForms überhaupt Sinn macht sei einmal dahingestellt. Hier muss abgewartet werden, wie sich Microsoft gegenüber anderen Plattformen in Zukunft verhält.

6 Einordnung von .NET

.NET lässt sich auf mindestens vier Ebenen mit anderen Ansätzen vergleichen. Wir wollen damit aufzeigen, dass man auf den konkreten Anwendungsfall schauen muss, wenn man sich für oder gegen .NET entscheidet.

6.1 .NET als Komponentenmodell

Auf dieser Ebene muss sich .NET mit J2EE (EJB) und CORBA messen lassen. (D)COM wird dagegen nicht als Konkurrent sondern nur als

Vorgänger betrachtet. Sowohl für J2EE als auch für CORBA fallen deren inhärente Plattformunabhängigkeit stark ins Gewicht. Wirklich kritische Anwendungen laufen heute selten auf Microsoft-Betriebssystemen, so dass für diese .NET quasi keine Alternative darstellt. Natürlich deckt der Markt der Komponententechnologie in großem Umfang auch das Umfeld der Client-Betriebssysteme ab. Viele Windows-Anwendungen sind heute mittels der COM-Technologie implementiert. Für diese wird wohl J2EE oder CORBA keine echte Alternative sein, da die dafür notwendige Middleware auf den Windows-Plattformen im Gegensatz zur .NET-Plattform nicht a priori vorhanden ist. Auf diesem Gebiet werden .NET und J2EE wohl sehr gut parallel existieren können.

6.2 .NET für GUI-Anwendungen

Auf dieser Ebene muss sich .NET mindestens mit Java vergleichen lassen. Denkbar wäre aber auch ein Vergleich zum Beispiel mit Delphi o.Ä. Wenn man davon ausgeht, dass ein sehr großer Teil aller GUI-Anwendungen auf Windows-Rechnern laufen, ist einzusehen, dass .NET mit seinen für diese Plattform optimierten Oberflächen-Bibliotheken am besten für GUI-Anwendungen geeignet sein wird.

6.3 .NET als Web-Services Plattform

Auf Ebene der Web-Services bietet sich nur der Vergleich zu J2EE an. Weitere nennenswerte Plattformen für diese Anwendungsform sind uns nicht bekannt. Hier spricht für .NET, dass es in Gestalt des Visual Studio .NET eine Entwicklungsumgebung gibt, die es einem sehr leicht macht Web-Services sowohl zu entwickeln als auch zu verwenden. Vergleichbar hierzu ist der WebSphere Studio Application Developer (WSAD). Da Web-Services vom Prinzip her plattform- und sprachunabhängig sind, wird es hier wohl keinen Gewinner oder Verlierer geben.

6.4 .NET als IT-Infrastruktur

Hier muss sich .NET insofern einem Vergleich stellen, dass mit .NET und den dazugehörigen Servern und Betriebssystemen eine homogene IT-Infrastruktur aus einer (Hersteller-)Hand existiert. Andere Infrastrukturen wie zum Beispiel J2EE setzen dagegen auf standardisierte APIs um beliebigen Anbietern die Möglichkeit zu bieten entsprechende Produkte auf den Markt zu bringen. Dadurch ist auf J2EE-Seite eine Auswahl verschiedener Hersteller bezuglich einer bestimmten API, und somit auch ein problemloser Austausch von Produkten möglich. Auf eine Betrachtung des Lizenzmodells wurde verzichtet, da dies den Rahmen der Fachstudie sprengen würde.

Teil II

Einsatzfähigkeit

7 Einsatzfähigkeit von .NET bei ARS NOVA

Dieses Kapitel beleuchtet die technologische Einsatzfähigkeit von .NET bei ARS NOVA. Es geht dabei letztlich um die Frage, ob die bei ARS NOVA typisch durchgeführten Projekte mit Hilfe der .NET-Technologie realisierbar wären.

7.1 Klassifikation von Anwendungen

Eine Anwendung dient typischerweise der rechnergestützten Abwicklung eines Geschäftsvorfalles. Die Klassifikation geht dabei von einem Arbeitsplatz-Rechner aus, auf dem ein Anwender die Anwendung bedient. Anwendungen werden oft in einem 3-Stufenmodell (s. Abbildung 10) entwickelt, deren einzelnen Schichten variabel auf Client (dem Arbeitsplatz-Rechner) und Server aufgeteilt werden können. Aus dieser Aufteilung heraus ergeben sich die einzelnen Klassen von Anwendungen. Für die Implementierung jeder dieser Schichten werden bestimmte Komponenten benötigt. Die Anforderungen die eine Schicht an eine Technologie stellt kann sich auf dem Server durchaus von der auf dem Client unterscheiden, weshalb die geforderte Unterstützung derselben Schicht in verschiedenen Klassen durchaus unterschiedlich sein kann. Besondere Eigenschaften einer Technologie wie zum Beispiel eine integrierte Garbage Collection o.ä. sind zumeist wünschenswert aber im Normalfall nicht erforderlich. Solche Eigenschaften werden in dieser Klassifikation nicht berücksichtigt, obwohl sie von Fall zu Fall sehr wohl zu Entscheidungskriterien werden können.

7.1.1 Stand-alone-Anwendungen

Die Klasse der Stand-alone-Anwendungen fasst diejenigen Anwendungen zusammen, die alle drei Schichten auf einem einzelnen Rechner ausführen, wie Textverarbeitung oder Bildverarbeitung.

7.1.2 FAT-Clients

FAT-Clients sind Client-seitige Anwendungsteile einer Client-/Server-Architektur, die zumindest einen Teil der Anwendungslogik auf dem Client ausführen. Die Grenze zwischen Client und Server kann also innerhalb oder unterhalb der Schicht 2 verlaufen. Beispiele: Groupware-Lösungen, Management-Anwendungen (z.B. verteilte Ressourcenplanung)

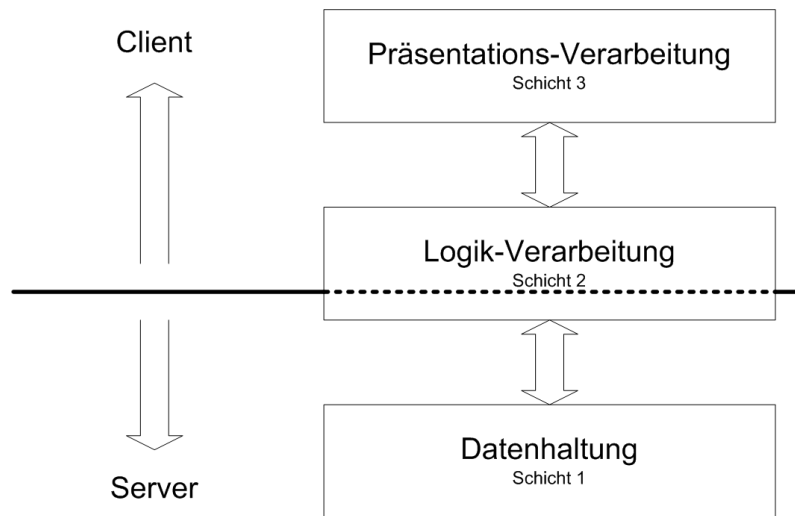


Abbildung 10: 3-Stufenmodell

7.1.3 THIN-Clients

THIN-Clients sind Client-seitige Anwendungsteile einer Client-/Server-Architektur, die keine Anwendungslogik auf dem Client ausführen. Oft wird selbst die Präsentationslogik bzw. auch die Präsentationserzeugung (Serverside Rendering), zumindest teilweise auf den Server verlagert. Die Grenze zwischen Client und Server kann also oberhalb, innerhalb oder unterhalb der Schicht 3 verlaufen. Beispiele: Webbasierete Anwendungen

7.2 Zusätzliche Aspekte der Server-Seite

Server-basierte Anwendungen stellen meist eine geschäftskritische Funktionalität zur Verfügung. Aspekte wie Verfügbarkeit, Lastausgleich und Ähnliches bzw. die Dienste, die diese zur Verfügung stellen, sind somit für alle Anwendungen auf einem Server von großer Bedeutung.

7.2.1 Geschäftsobjekte

Geschäftsobjekte bilden die Geschäftsvorfälle in Software ab und bilden somit die Logik-Schicht auf einem Server ab. Die für die Verwendung durch andere Anwendungen benötigten Funktionalitäten, wie z.B. Verbindungsaufbau, werden typischerweise von einer technologieabhängigen Ausführungsumgebung (Application Server) bereitgestellt. Dieser Application Server übernimmt oft auch Aspekte, die im vorigen Abschnitt zu Server-Anwendungen im Allgemeinen genannt wurden (z.B. Lastausgleich). Ein solcher Application Server stellt in der .NET-Technologie das Server-Betriebssystem selbst dar. Man benötigt dazu keinerlei zusätzliche Software. Die Unterstützung für Geschäftsobjekte

ist also gewährleistet.

7.2.2 BackOffice-Anwendungen

Dieser Klasse gehören Anwendungen wie zum Beispiel Datenbank-Server und ähnliches an. Während Geschäftsobjekte meist unternehmensspezifische Funktionalitäten anbieten, stellen BackOffice-Anwendungen allgemeine Funktionen, wie zum Beispiel die Datenverwaltung, zur Verfügung. Entscheidend für den Einsatz einer Technologie ist hier vor allem, dass existierende BackOffice-Anwendungen möglichst gut angebunden sind. Hier zeigt sich auch das derzeit größte Manko der .NET-Initiative: Die Liste der direkt unterstützenden BackOffice-Anwendungen ist sehr kurz. Bei der Entscheidung für oder gegen .NET muss also gerade dieser Bereich individuell und sehr sorgfältig geprüft werden. Insbesondere kann sich die Verfügbarkeit mit zunehmender Verbreitung von .NET erhöhen.

7.2.3 Web-Services

Die Klasse der Web-Services-basierten Anwendungen stellt eine Kapselung der beiden bisher genannten Klassen dar. Sowohl Geschäftsobjekte als auch die BackOffice-Server sind über eine definierte Schnittstelle basierend auf Web-Protokollen (TCP/IP, HTTP, XML, SOAP, ...) ansprechbar und somit für andere Software (auch) über diese Schnittstellen nutzbar. Mit ASP.NET und dem Internet Information Server bietet .NET auch hierfür eine sehr gute Unterstützung.

7.3 .NET-Unterstützung der Anwendungs-Klassen

In diesem Abschnitt wird tabellarisch dargestellt, welche Komponenten die einzelnen Schichten des 3-Stufenmodells benötigen und wie ihre Umsetzung in .NET heißen (In Klammer: Der dazugehörige Namespace der .NET-Klassen-Bibliothek).

7.4 Evaluation der .NET-Komponentenverfügbarkeit

Die meisten Projekte bei ARS NOVA basieren auf der J2EE-Architektur und zur Erstellung der Anwendungen werden oft Komponenten von Drittanbietern oder aus dem Open-Source-Umfeld eingesetzt. Da ARS NOVA kundenspezifische Anwendungen in einem breiten Anwendungsspektrum entwickelt, ist es nicht möglich eine Liste von typischen Standardkomponenten aufzustellen und diese dann direkt mit den äquivalenten Komponenten für die .NET-Architektur zu vergleichen. An dieser Stelle wird deshalb nur eine Betrachtung des Gesamtumfelds durchgeführt. Im Anhang befindet sich aber die Dokumentation eines im kleinen Rahmen durchgeführten Versuchs, die in zwei Beispielprojekten bei ARS NOVA verwendeten Komponenten aus dem J2EE-Umfeld durch .NET-fähige Komponenten zu ersetzen. Ein entscheidendes Kriterium für den Einsatz von .NET ist nach dem oben

		Stand-Alone	FAT-Client	THIN-Client	.NET
Allg.	Kommunikation	Intra-/Interprozess	Netzwerk		- .NET Remoting (System.Net) - ASP.NET (s. Web-Services)
Client	UI				- GDI+ (System.Drawing) - WinForms (System.Windows.Forms)
	Logik				
	Daten				- IO (System.IO) - ADO.NET/DataProvider (System.Data)
Server	UI				- WebForms (System.Web.UI) - ASP.NET (System.Web)
	Logik				s. Geschäftsobjekte
	Daten				s. BackOffice-Anwendungen

Abbildung 11: .NET-Unterstützung der Anwendungs-Klassen

aufgezeigten Bild ein ausreichend großes Angebot an Komponenten. Da innerhalb von .NET-Anwendungen COM-Komponenten verwendet werden können, gibt es eine Vielzahl von Anbietern, die mit ihren Produkten das komplette Anwendungsspektrum abdecken. Für einen Teil der COM-Komponenten gibt es bereits neue oder Nachfolgeprodukte auf .NET-Basis oder es sind welche in Planung. Nachholbedarf besteht eigentlich nur in der Anbindung an die am meisten verkauften Datenbanksysteme, da bisher nur für den SQL-Server von Microsoft und für Oracle-Datenbanken die spezifischen Managed DataProvider (vergl. Kapitel Teil 1) vorliegen. Der Zugriff ist zwar auch über die mitgelieferten Standard DataProvider möglich, aber noch ausbaufähig.

7.5 Ableitung der Einsatzfähigkeit bei ARS NOVA

Die bei ARS NOVA entwickelten Anwendungen folgen der hier vorgestellten Klassifikation von Anwendungen. Wie gezeigt werden konnte, werden diese Klassen durch .NET unterstützt. Auch im Bereich der allgemeinen Komponenten lässt sich festhalten, dass .NET nur bei den Datenbankanbindungen gegenüber der J2EE-Architektur im Nachteil ist, ansonsten aber eine vergleichbare Komponentensituation existiert. Es soll auch nicht versäumt werden die hervorragende Unterstützung der Oberflächen-Entwicklung herauszustellen. Die mit .NET eingeführten WinForms sollen auf lange Sicht die GDI-Schnittstelle der Windows-Betriebssysteme ablösen und bieten eine durchgängig objekt-

orientierte Oberflächen-Bibliothek. Zur Ableitung der Einsatzfähigkeit kann man abschließend sagen, dass .NET prinzipiell einsetzbar ist. Eine genaue Evaluation der einzusetzenden Fremdprodukte und die entsprechende Anbindung wird aber empfohlen.

8 Vergleich der Entwicklungsumgebung

Im folgenden Kapitel soll die Frage geklärt werden, inwieweit sich die beiden Entwicklungsumgebungen IBM Websphere Application Developer und Microsoft Visual Studio .NET in ihrem Unterstützungsniveau bei der Entwicklung von Anwendungen unterscheiden. Hierzu werden zuerst die beiden Entwicklungsumgebungen nacheinander genauer betrachtet und dann werden beide miteinander verglichen. Die Betrachtung beginnt in Kapitel 8.1 mit dem IBM Websphere Studio Application Developer, dann folgt in Kapitel 8.2 das Microsoft Visual Studio .NET und zum Abschluss der Vergleich der beiden.

8.1 IBM Websphere Studio Application Developer

Der Websphere Studio Application Developer (WSAD) [14] wurde für den Vergleich ausgewählt, da er bei ARS NOVA die derzeitige Standard-Entwicklungsumgebung ist. Er wurde bereits in einigen Projekten eingesetzt und löst nun nach und nach die beiden IBM-Werkzeuge [12] Visual Age for Java und Websphere Studio ab, deren offizieller Nachfolger er ist. Der WSAD basiert auf der Plattform-Technologie des Open-Source-Projektes Eclipse [8]. Hinter dem Eclipse-Projekt steht die Idee eine Entwicklungsplattform zu entwickeln, in die alle benötigten Entwicklungstools per Plug-In integriert werden können. Der WSAD ist optimiert für das Entwickeln von J2EE-Applikationen und deckt damit das komplette Projektspektrum von der Fat-Client-/Server-Lösung bis zu ausgereiften, webbasierten Enterprise-Applikationen auf Basis von Portlets, Servlets oder Enterprise Java Beans ab. Im Lieferumfang des WSAD befinden sich zusätzlich noch der IBM Websphere Application Server [15], die IBM DB2 UDB [11] sowie die beiden Versionierungswerkzeuge ClearCase LT. von Rational [26] und CVS [5]. Durch das Plug-In-Konzept von Eclipse ist neben dem Austausch des zur Entwicklung benötigten JDKs, auch der Austausch oder die Erweiterung der Entwicklungsplattform durch Plug-Ins kein Problem. Da die Schnittstellen zur Entwicklungsplattform offen gelegt sind, gibt es bereits eine Vielzahl von kommerziellen oder Open-Source-Erweiterungen für Eclipse und den WSAD. Die oben erwähnten Versionierungssysteme, CVS und ClearCase Lt., erlauben entweder eine dateibasierte (CVS) oder eine repository-basierte (ClearCase) Quellcode-Verwaltung. Bei ARS NOVA wird die dateibasierte Version verwendet. Zusätzlich zu der Funktionalität der beiden Tools, stellt der WSAD noch ein Konzept bereit, das es ermöglicht die in den Projekten anfallenden Dateien in Projekten und Solutions geordnet abzuspeichern. Spezielle Beziehungen zwischen diesen werden in dafür erstellte Verwaltungsdateien abgelegt. Neben den reinen Editorfähigkeiten, wie z.B. Syntaxhighlighting

und der Referenzsuche, erlaubt der WSAD auch das integrierte Debugging von EJBs, Servlets und JSPs (lokal oder remote). Der Tomcat Webserver [1] lässt sich dafür genauso wie der mitgelieferte Websphere Application Server in die Umgebung integrieren. Mitgelieferte Werkzeuge erlauben die Analyse der ablaufenden Applikationen, so dass z.B. Performance-Engpässe besser erkannt werden können. Der WSAD unterstützt die Entwicklung von Web-Services komplett mit Werkzeugen die z.B. automatisiert aus WSDL-Code Stellvertreterobjekte erzeugen oder umgekehrt. Da Web-Services bei ARS NOVA noch keine Rolle spielen, wurden die Werkzeuge nicht näher betrachtet. Dem WSAD fehlt bis zur Fertigstellung des Plug-In für den WSAD durch die IBM im Oktober 2002 im Gegensatz zu seinem Vorgänger, dem Visual Age für Java, allerdings ein Oberflächendesigner.

8.2 Microsoft Visual Studio .NET

Das Visual Studio .NET [18] ist die bisher einzige Entwicklungsumgebung für .NET, die über die reinen Editorfunktionalität hinaus geht. Deshalb wird es in diesem Abschnitt näher betrachtet, um im nachfolgenden Kapitel den Vergleich mit dem Websphere Studio Application Developer durchzuführen. Das Studio wird in drei vom Funktionalitäts-Umfang her unterschiedlichen Versionen angeboten. Bei der Evaluation wird nur die „Enterprise Developer“-Version betrachtet, da die „Professional“ Version zuwenig Funktionalität für die unternehmensweite Software-Entwicklung bietet. Die „Enterprise Architect“-Version bietet zusätzlich zur in der Developer-Version enthaltenen Funktionalität noch interessante Möglichkeiten zur Modellierung von team- oder gar unternehmensweiten Entwicklungsprozessen und zum Visio-gestützten UML- und Datenbankdesign. Ihr Einsatz erfordert aber wahrscheinlich Eingriffe in den Entwicklungsprozess weshalb eine Betrachtung über den Rahmen der Fachstudie hinaus geht. Mit dem Visual Studio .NET bekommt man in der „Enterprise Developer“-Version eine visuelle IDE (Integrated Development Environment) zum Entwickeln von Software. Unterstützt werden mehrere .NET-Sprachen, wie z.B. C#, managed C++, VB.NET oder J#. Für diese werden neben den schon standardmäßigen Funktionen, wie Syntaxhighlighting und Befehlsweiterung, auch die benötigten Debugger, Projekt-schablonen sowie entsprechende Dokumentation der Klassenbibliotheken mitgeliefert. Ergänzt wird die Entwicklungsumgebung durch eine Vielzahl von Editoren, Generatoren und Wizards für z.B. XML- oder HTML-Dateien, die Entwicklung von Web-Services, der Verknüpfung von Datenquellen oder zum Analysieren von Programmabläufen. Das integrierte Debuggen, sowie das Deployment von Web-Applikationen wird in Verbindung mit dem Internet Information Server (IIS), der die Laufzeitumgebung für die .NET-Web-Applikationen ist, direkt aus der Umgebung ermöglicht. Ähnlich wie beim WSAD werden auch beim Studio mit dem SQL-Server eine Datenbank und verschiedene Serverprodukte zum Testen und Entwickeln von entsprechenden Anwendungen mitgeliefert. Für die Abbildung der Quellcodeorgani-

sation und der Projektstruktur wird ein auf Projekten und Solutions, dem WSAD ähnlichen, basierendes Konzept zur Verfügung gestellt. Mit dem vom WSAD bekannten Open-Source-Produkt CVS und dem Microsoft-Produkt Visual Source Safe werden zwei dateibasierte Versions- und Quellcode-Verwaltungsprogramme mitgeliefert. Die beiden interessantesten Werkzeuge, die in die IDE integriert sind, sind der Oberflächendesigner für Web-Applikationen und der GUI-Builder für WinForms-Oberflächen. Der Oberflächendesigner für Web-Applikationen ermöglicht das Erstellen von dynamischen Applikationen auf Basis der ASP-Webcontrols in der Art, dass für den Entwickler bei der Programmierung der Ablaufsteuerung kein Unterschied zur Programmierung von GUI-Applikationen bemerkbar ist. Diese Transparenz wird durch die Trennung des Source-Codes von der eigentlichen Webseite und die Nähe der Webcontrols-Architektur zu den WinForms bewusst herbeigeführt. Einen ähnlich guten Eindruck hinterlässt der GUI-Builder für die WinForms-Oberflächen. Bei der Erstellung einer Testapplikation durch die Autoren konnte festgestellt werden, dass sich der generierte Programmcode nicht von einer von Hand programmierten Version unterscheidet. Es wurden also keine überflüssigen Codezeilen vom Generator hinzugefügt. Als subjektiver Eindruck bleibt festzuhalten, dass die Erstellung der Oberflächen mit beiden Werkzeugen sehr komfortabel und intuitiv geschieht. Vor allem der GUI-Builder ist einer der zur Zeit besten Oberflächendesigner die es am Markt gibt.

8.3 Vergleich beider Entwicklungsumgebungen

Beim Vergleich der beiden Entwicklungsumgebungen Websphere Studio Application Developer und Visual Studio .NET richtete sich der Hauptaugenmerk neben dem Vergleich des Funktionalitäts-Umfangs, vor allem auf die Einsatzfähigkeit der .NET-Entwicklungsumgebung für die in Teil 1 und 2 der Fachstudie näher beschriebenen typischen Projekttypen bei ARS NOVA, in denen sich der WSAD schon bewährt hat. Betrachtet man den Funktionalitäts-Umfang, den beide Entwicklungsumgebungen umfassen, dann lässt sich feststellen, dass beide Produkte in etwa das gleiche Unterstützungsniveau für die Entwickler bieten. Den größten Unterschied den es bei der Funktionalität zwischen beiden gibt, ist der fehlende Oberflächendesigner für Swing-Oberflächen beim WSAD. Hier bietet das Visual Studio eindeutig mehr Funktionalität. Das Fehlen eines Oberflächendesigners für Web-Applikationen ist eher ein Unterschied in der Architektur von J2EE zu .NET. J2EE verfügt standardmäßig nicht über ein Webcontrols-Äquivalent, daher fehlt auch der Designer. Das Visual Studio .NET bietet Entwickler im Gegensatz zum WSAD, dessen Angebot überschaubar bleibt, eine riesige Anzahl an Wizards an. Dort bietet der WSAD nicht soviel Auswahlmöglichkeit. Da der Einsatz von Wizards in der Entwicklung bei ARS NOVA eher keine Rolle spielt, existiert hier auch kein wirklicher Vorteil für das Studio. Der WSAD besitzt dafür den Vorteil, dass er einfacher erweiterbar ist. Beim Visual Studio existieren zwar auch Schnittstellen für Plug-Ins, trotzdem ist der WSAD

allein vom Konzept her, durch die Eclipse-Plattform besser aufgestellt. Der fehlende GUI-Builder macht sich bei der Entwicklung von Stand-alone-Anwendungen bemerkbar, damit besitzt das Visual Studio im Gegensatz zum WSAD den klaren Vorteil, dass es damit ein schnelles Prototyping unterstützt. Selbst wenn die endgültige Entwicklung der Oberfläche doch von Hand geschieht, kann damit der Entwicklungsprozess beschleunigt werden, da dem Kunden bereits sehr schnell ein Prototyp der Oberfläche präsentiert werden kann, so dass dieser sich den Ablauf der Anwendung visuell betrachten und auf eventuelle Fehler oder Probleme früher reagieren kann. Bei den Thin-Clients bietet der Oberflächendesigner für die Web-Applikationen im Visual Studio einen ähnlichen Vorteil wie bei den Stand-alone-Anwendungen mit ein. Auf der Serverseite bieten beide Produkte für die Entwicklung ihrer Komponenten eine ähnliche gute Unterstützung. Vorteile besitzt der WSAD bei der Anbindung von Datenbanken, da hier spezielle Zugriffskomponenten verfügbar sind, außerdem stehen vor allem im Bereich der EJB-Entwicklung besonders viele Erweiterungen für den WSAD zur Verfügung. Auch bei den FAT-Client-/Server-Systemen besitzt das Visual Studio den Vorteil des Oberflächendesigners. Ansonsten sind bei beiden aber keine Unterschiede festzustellen. Im Bereich der Teamentwicklung weisen beide Umgebungen keine Unterschiede auf. Bei beiden sind die selben Standardtools im Lieferumfang enthalten und die Integration von Speziallösungen ist möglich. Als Fazit lässt sich damit festhalten, dass sich mit dem Websphere Studio Application Developer und dem Visual Studio .NET zwei vergleichbare Entwicklungsumgebungen gegenüberstehen. Bei einem Einsatz von .NET in Projekten bei ARS NOVA steht mit dem Visual Studio .NET eine Entwicklungsumgebung bereit, die eine vergleichbar gut unterstützte Entwicklung von Anwendungen erlaubt, wie es bei ARS NOVA mit dem Websphere Studio Application Developer zur Zeit möglich ist [10], [4], [6], [31].

9 Anhang

9.1 „ADIS“ — FAT-Client-/Server-Anwendung

9.1.1 Beschreibung und Klassifikation

Im bereits abgeschlossenen Projekt „ADIS“ wurde eine den Außendienst unterstützende Kunden-Verwaltungs-Software entwickelt. Es handelt sich um eine Client-Anwendung für Microsoft Windows-Plattformen, die auf persistente Daten in einer Datenbank zugreift. Diese Datenbank kann je nach Installationsort auf demselben oder einem anderen Rechner installiert sein. Die Oberfläche ist mit dem Java Swing-Framework realisiert. Als zusätzliche Komponenten werden diverse DLLs zum Beispiel für die Hilfeanbindung sowie ein Objekt-Relationales Mapping-Tool, das die Umwandlung zwischen Fachobjekt und relationalem Tupel durchführt, verwendet. Außerdem werden in manchen Situationen weitere Anwendungen, wie zum Beispiel Microsoft Word gestartet.

Einordnung in Anwendungs-Klassifikation:

ADIS ist eine reine FAT-Client-/Server-Anwendung, die keine zusätzlichen Anforderungen stellt. ADIS könnte somit mit jeder die FAT-Client-Klasse unterstützenden Technologie realisiert werden. Allerdings ist die Verwendung eines Oracle-DBMS vom Kunden vorgegeben worden. Da hierfür derzeit nur eine Beta-Version des notwendigen DataProviders in .NET verfügbar ist, wären hier evtl. Performance-Einbußen zu erwarten.

9.1.2 Komponentenvergleich

Eingesetzte Komponenten	Beschreibung	Vergleichbare Lösung in .NET
BissPress	BissPress erzeugt PDF-Dokumente und startet den Adobe Acrobat Reader. Es liegt in Form einer DLL für die Windows-Plattform vor. Der Aufbau der Seite geschieht über Primitive wie z.B. Dokument, Seite, Teil, Linie, Grafik, Text usw.	Unter .NET kann die Komponente, da sie als DLL vorliegt, als unmanaged Code verwendet werden.
Word nur Start	Word wird mit einem speziellen Dokument und einer Steuerdatei für Serienbriefe aufgerufen	Der Zugriff auf Office-Applikationen ist über existierende COM-Komponenten möglich

Eingesetzte Komponenten	Beschreibung	Vergleichbare Lösung in .NET
Produktsystem	Stellt eine „Laufzeitumgebung“ für Rechenkerne ¹ (quasi-Plug-In) zur Verfügung. Es kann innerhalb derselben JVM oder auch außerhalb via TCP/IP angesprochen werden. Es stellt auch die definierte Kommunikation bereit.	Da .NET nicht auf der JVM abläuft wird das Produkt nicht benötigt.
Webgain TopLink for Java [34]	ermöglicht das Mapping von Objekten auf Relationen, ist anpassbar auf diverse DBMS (Oracle: Normalversion ADIS, Access: Maklerversion ADIS)	Für das Produkt gibt es kein äquivalentes Produkt in .NET. Es ist auch nicht ersichtlich ob der Hersteller eines plant.
WinHelp-DLL	DLL zur Verwendung des Standard Windows-Hilfesystems	Die Erstellung von Windows-Hilfedateien wird von Visual Studio direkt unterstützt.

¹Ein Rechenkern ist ein Programm das den Berechnungsalgorithmus, die Standard-Vorgaben, uvm. einer konkreten Versicherungsart zum Beispiel einer Haftpflicht-Versicherung kapselt

Eingesetzte Komponenten	Beschreibung	Vergleichbare Lösung in .NET
CoRoutine	Ein Tool zum einfachen Verwenden von DLLs aus Java heraus	Die Verwendung dieses Tools entfällt, da .NET nativ auf DLLs zugreifen kann.
Oracle 9i [23]	Datenbanklösung von Oracle	Im Moment gibt es nur eine Vorabversion des Datenbank-Treibers.

9.2 „KoMo“ — THIN-Client-/Server-Anwendung

9.2.1 Beschreibung und Klassifikation

Die Anwendung KoMo ist eine typische THIN-Client-/Server-Anwendung, da die gesamte Geschäftslogik auf einem Server abläuft. Der Client stellt lediglich die Bildschirmmasken in Form von HTML-Seiten dar. Die Anwendung implementiert einen genau abgegrenzten Bereich eines größeren Workflows. Die Anbindung an diesen Workflow erfolgt in beide Richtungen über das Messaging-System MQSeries von IBM. Die Geschäftslogik läuft, in EJB-Technologie implementiert, in einer WebSphere Application Server Installation und greift auf eine IBM DB2-Datenbank zurück. Für die Umsetzung des MVC-Frameworks wird das Struts-Framework der Apache Software Foundation eingesetzt.

Einordnung in Anwendungs-Klassifikation:

Wie oben schon erwähnt ist KoMo eine typischer THIN-Client-/Server-Anwendung. KoMo stellt keine zusätzlichen Anforderungen.

9.2.2 Komponentenvergleich

Eingesetzte Komponenten	Beschreibung	Vergleichbare Lösung in .NET
IBM Websphere Application Server [15]	Applicationserver für die Ausführung der Enterprise Java Beans.	Die Verwendung des Applicationservers entfällt bei einer reinen .NET-Lösung, da die Assemblies direkt auf dem Betriebssystem ausgeführt werden und keinen speziellen Applikationsserver für die Ausführung benötigt. Soll eine heterogene Lösung unterstützt wrden, muss als Schnittstelle der BiztalkServer eingesetzt werden.
IBM MQSeries [13]	ermöglicht die Kommunikation mit anderen angebundenen Systemen via Java Messaging Services	Als vergleichbares Produkt steht der Biztalk-Server als Ersatz zur Verfügung

Eingesetzte Komponenten	Beschreibung	Vergleichbare Lösung in .NET
IBM DB2 UDB [11]	Highlevel-Datenbank von der IBM	Prinzipiell ist der Zugriff von .NET aus über allgemeine Managed DataProvider möglich, allerdings sind DataProvider vom Hersteller, aus Performancegründen, immer vorzuziehen. Für die IBM DB2 gibt es zur Zeit noch keinen speziellen DataProvider.
Struts [1]	Ein Framework für Weboberflächen-Widgets, das von der Open-Source Community Apache.org im Rahmen des Jakarta Projektes entwickelt wurde.	Mit ASP.NET ist eine vergleichbare Bibliothek direkt in .NET integriert.

Teil III

Einführungsstrategien

10 .NET-Einführungsstrategien

Die Anzahl der dokumentierten Projekte und Produktentwicklungen mit .NET ist noch relativ überschaubar. Die meisten Dokumente stammen von Microsoft selbst und enthalten meist statt konkreten Strategien zur Einführung von .NET nur Firmen- und technische Lösungsbeschreibungen. Die Einführungsdokumente, die ansonsten existieren beschäftigen sich mit der Migration der Entwicklung mit den bisherigen MS-Programmiersprachen auf ihre Nachfolger. Diese Ansätze spielen für ARS NOVA nur eine untergeordnete Rolle, da dort zur Zeit hauptsächlich mit Java entwickelt wird, weshalb diese Ansätze nachfolgend auch nicht weiter betrachtet werden. Aus den wenigen Artikeln, Berichten und Reportagen die verfügbar sind, kristallisiert sich ein einziges Rahmenwerk als Einführungsstrategie heraus. Der nachfolgende Abschnitt beschreibt ein solches Einführungsprojekt und in 10.2 werden nochmals die wichtigsten Merkmale zum Gelingen eines Einführungsprojekts gesondert aufgelistet.

10.1 Beschreibung der Einführungsprojekte

Das typische Einführungsprojekt der Artikel ist ein webbasierter Client der mit einem SQL-Server als Datenbank auf der Serverseite zusammen arbeitet. Der Grund für diese statistische Häufung kann nicht genau belegt werden. Er ist wohl auf den überschaubaren Rahmen, was Umfang und Komplexität solcher Projekte betrifft, zurückzuführen. Auch schienen alle Projekte ohne Zeitdruck durchgeführt worden zu sein, da nirgends auf eine besonders schnelle Durchführung hingewiesen wird. Typischerweise startet das Einführungsprojekt mit der Schulung einer kleinen Anzahl von Mitarbeitern in die Grundlagen von .NET und die verwendete Programmiersprache, sofern diese nicht schon bekannt ist. Diese beginnen nach dem Ende der Schulung, dann mit der Analyse und dem Entwurf. Während der Analyse und Entwurfsphase werden, sofern dies für den weiteren Projektverlauf nötig ist noch weitere Entwickler geschult, um bei der Programmierung zu helfen. Eine Alternative wäre hier die Entwickler während der Codierung unter Anleitung des schon geschulten Kernteams mittels „training on the job“ anzulernen. Der weitere Verlauf unterscheidet sich meist nicht von den herkömmlich eingesetzten Entwicklungsprozessen. Der Hauptunterschied zu normalen Projekten wie sie oft in Unternehmen wie ARS NOVA durchgeführt werden, liegt somit in der sorgfältigeren Auswahl der Aufgabe, die das Projektteam lösen soll und der vorgeschalteten Schulungsphase für den Kern des Entwicklerteams sowie einzelnen weiteren Schulungen die bei Bedarf für die Mitarbeiter während des Projektes zusätzlich angeboten werden.

10.2 Merkmale der Einführungsprojekte

Aus der Beschreibung des Einführungsprojekts werden nachfolgend die Merkmale eines Einführungsprojekts herausgearbeitet.

Projektbezogene:

- Von Zeit und Umfang überschaubare Aufgaben für das Einführungsprojekt auswählen, um die Komplexität möglichst klein zu halten und um die benötigten Mitarbeiter gezielter auswählen zu können.
- Das Projekt mit großzügig bemessener Zeit ausstatten, um Zeit genug für die Einarbeitung in .NET zu lassen.
- Wenn es geht eine Aufgabe aus einem bekannten Umfeld auswählen, um einen zusätzlichen Bruch über Fachgebietsgrenzen zu vermeiden

Mitarbeiterbezogene:

- Das Projekt mit einer kleinen Anzahl an Mitarbeitern beginnen, die dann über die komplette Projektdauer den Kern des Projektteams bilden.
- Diese sollten vor dem Beginn des eigentlichen Projekts eine Schulung über die Grundlagen von .NET bekommen. Sie sollten die zusätzlich zum Projekt hinzustoßenden Mitarbeiter, oder nach Projektende die restlichen Mitarbeiter, schulen können.

11 Voraussetzung zur Einführung bei ARS NOVA

Das Unternehmen ARS NOVA hat bereits einen größeren Technologie-Wechsel erfolgreich bestritten. Aus diesem Grund werden in diesem Abschnitt die daraus gewonnenen Erfahrungen genannt.

Zu diesem Zweck wurde mit einem Repräsentanten der ARS NOVA ein Interview geführt. Für die Erarbeitung einer Einführungsstrategie müssen demnach drei Punkte mehr oder weniger separat betrachtet werden:

Werkzeuge Die Entwickler des Unternehmens müssen sich in mindestens ein neues Werkzeug einarbeiten. Im Falle von Java nach .NET ist auf jeden Fall die Entwicklungsumgebung betroffen. Hier kommt es darauf an, den Umstieg so einfach wie möglich zu gestalten. Die Werkzeuge sollten also so ähnlich wie möglich sein.

Sprache(n) Die Entwickler müssen eine neue Sprache erlernen. Auch im Falle von .NET ist dies ein wichtiger Punkt, da die tragende Rolle von C# bei ARS NOVA anerkannt wurde und somit kein „Java.NET“ angestrebt wird.

Technologie-Umfeld Hierzu gehört der Wechsel von der J2EE-Architektur zur .NET-Welt. Außerdem wird eine Vermischung von Java und .NET in konkreten Projekten als wahrscheinlich angesehen. In einem solchen Szenario, Java auf Server- und .NET auf Client-Seite, ist besonders das Zusammenspiel der beiden Technologien maßgebend.

Die Erfahrungen bei ARS NOVA haben für diese drei Bereiche bereits zu einem Vorgehens-Muster geführt.

11.1 Werkzeuge

Im Falle der Entwicklungsumgebung hat sich gezeigt, dass eine wesentliche Voraussetzung ist, dass sogenannte Schlüsseleigenschaften wie zum Beispiel Klassen-Browser, Debugger, u.ä. möglichst ähnlich zu bedienen sind und äquivalente Funktionalitäten anbieten.. Bei allen anderen Funktionalitäten sind die Entwickler offenbar eher bereit ihre Arbeitsweise anzupassen. Nicht so aber bei diesen für die Entwicklungsarbeit wesentlichen Funktionen. Die Auswahl der neuen Entwicklungsumgebung muss sich also möglichst nahe an der bisherigen orientieren. Für den konkreten Umstieg wird einem Entwickler zur Aufgabe gemacht sich kurz einzulesen (beispielsweise Online-Dokumentation) um anschließend eine konkrete Aufgabe zu lösen. Die Erfahrung hat gezeigt, dass ein „Rumspielen“ nicht zum Erfolg führt! Der minimale Zeitaufwand für einen erfolgreichen Werkzeug-Wechsel wird von Seiten ARS NOVAs mit ein bis zwei Wochen angegeben.

11.2 Sprache(n)

Für den Umstieg auf eine neue Sprache hat sich gezeigt, dass ein möglichst konstantes Umfeld empfehlenswert ist. Darüber hinaus kann es sehr sinnvoll sein, von wenigen Entwicklern Kern-Konstrukte des bisherigen Arbeitens auf die neue Sprache übertragen zu lassen. Diese Konstrukte (wie sie zum Beispiel in den ARS NOVA Infrastruktur-Komponenten vorkommen) haben oft eine tragende Rolle und sind oft auch nicht primitiv umgesetzt. Es wäre äußerst unvorteilhaft diese Migrationsarbeit auf diesem Gebiet jedem einzelnen Entwickler aufs neue zu übertragen. Als entscheidend wurde von ARS NOVA empfunden, dass möglichst nicht nur die Syntax sondern auch die neuen Konzepte erlernt werden. Beim Umstieg von Smalltalk auf Java zum Beispiel wurde lange Zeit zwar Java-Syntax verwendet, die Programm-Strukturen erinnerten aber eher an Smalltalk. Aus diesem Grund wird auch das Vorhandensein eines Buches der Art „Java in a nutshell“ als quasi Voraussetzung genannt. Wesentlich ist hier aber, dass es sich um ein qualitativ sehr hochwertiges Exemplar handelt. Für den Umstieg auf eine neue Sprache würde man bei ARS NOVA mindestens drei Wochen ansetzen. Besonders wichtig ist dabei, den Umstieg in einem Projekt anzugehen, das bezüglich der terminlichen Anforderungen sehr viel Spielraum bietet.

11.3 Technologie-Umfeld

Das Technologie-Umfeld wird als der heikelste Punkt bei einem geplanten Umstieg angesehen. Im Zuge des Einstieges in die J2EE-Technologie hat sich gezeigt, dass externe Schulungen nicht vermeidbar sind. In entsprechend dimensionierten Projekten wurde auch aktives Coaching der einzelnen Entwickler zugekauft. Außerdem wurde ein guter Support seitens der Hersteller-Firma als elementar eingestuft. Vor allem wenn die Technologie bereits einige Zeit auf dem Markt ist, bei .NET ist dieser Punkt bereits erreicht, ist auch das Überangebot an Informationen ein nicht zu unterschätzendes Risiko. Eine gute Filterung ist dabei unerlässlich. Bei ARS NOVA hat es sich bewährt mit einem einzelnen Projekt in eine neue Technologie zu starten. Ziel dieses Projektes ist dann zusätzlich diverse Richtlinien und Erfahrungsberichte zu erarbeiten, damit nachfolgende Projekte die Erfahrungen des Pionier-Projektes sinnvoll und größtmöglich einsetzen kann. Das Technologie-Umfeld bietet zu viele Randbedingungen, als dass eine realistische Zeitabschätzung angegeben werden kann.

12 Vorschlag für die Einführung bei ARS NOVA

Basierend auf der in 10 festgestellten Tatsache, dass eine konkrete Beispiel-Strategie seitens Microsofts nicht angeboten wird und auch dementsprechend wenig Referenz-Projekte verfügbar sind, lautet der Vorschlag an die ARS NOVA sich an den eigenen Erfahrungen aus dem Umstieg von Smalltalk auf Java zu orientieren. Aus diesem Grund wird in dieser Ausarbeitung auf eine Erarbeitung einer konkreten Strategie verzichtet. Eine solche Strategie wäre eineseits lediglich eine mehr oder weniger ersichtliche Verpackung der ohnehin vorhandenen Erfahrungen. Andererseits ist eine Strategie zur Einführung einer neuen Technologie von vielen Randbedingungen abhängig, die sich bis zu einer tatsächlichen Einführung bei ARS NOVA noch erheblich ändern könnten.

13 Zusammenfassung

In diesem Dokument wurden die grundlegenden Eigenschaften der .NET-Strategie und ihr Einsatz näher untersucht. Ziel der Untersuchung war es herauszufinden, ob für das Unternehmen ARS NOVA Software GmbH ein Umstieg von Java auf .NET als möglich, sinnvoll oder notwendig erachtet wird. Nachdem im ersten Teil die Architektur von .NET vor allem in Hinblick auf ihre Besonderheiten beschrieben wurde, konnte im zweiten Teil gezeigt werden, dass für die meisten bei ARS NOVA durchgeführten Projekte ein Wechsel zumindest möglich ist. Die Frage ob ein Wechsel sinnvoll oder notwendig ist konnte nicht eindeutig beantwortet werden. Es lässt sich aber zusammenfassen, dass .NET auf dem Client eine echte Alternative darstellt, während dies auf

dem Server derzeit noch eher verneint werden muss. Schließlich ergab der dritte Teil des Dokuments, dass die bei ARS NOVA vorhandenen Erfahrungen im Zusammenhang mit einem Technologie-Wechsel ausreichen sollten, um auch einen evtl. Umstieg auf .NET erfolgreich durchführen zu können.

Literatur

- [1] *Apache Software Foundation: Jakarta Projekt.*
URL: <http://www.apache.org/jakarta>.
- [2] *Assembly Objektmodell.*
URL: [http://www.craiglarman.com/articles/A Domain Model of .NET Assemblies.pdf](http://www.craiglarman.com/articles/A%20Domain%20Model%20of%20.NET%20Assemblies.pdf).
- [3] *Assembly Versionierung.*
URL: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconassemblyversioning.asp>.
- [4] *Building Web Services with .NET vs. IBM Websphere 4.0.*
URL: <http://www.microsoft.com/msdn>.
- [5] *CVS.*
URL: <http://www.cvshome.org>.
- [6] DEBOER STEPHENSON: *Getting know Websphere Studio Application Developer.*
URL: <http://www-106.ibm.com/developerworks/ibm/library/i-wsad>.
- [7] *dotGnu.*
URL: www.dotgnu.org.
- [8] *Eclipse.*
URL: <http://www.eclipse.org>.
- [9] *ECMA.*
URL: <http://www.ecma.ch>.
- [10] *How the IBM Web Services Development Environment and Toolkit Compares Microsoft Visual Studio .NET.*
URL: http://www7b.boulder.ibm.com/wsdd/techjournal/0106_kraft/%-kraft.html.
- [11] *IBM DB2 UDB.*
URL: <http://www-5.ibm.com/de/software/data/udb.html>.
- [12] *IBM Entwicklungsumgebungen.*
URL: www.ibm.de, IBM - Produkte - Entwicklungsumgebungen.
- [13] *IBM MQSeries.*
URL: <http://www-3.ibm.com/software/ts/mqseries>.
- [14] *IBM Websphere Application Developer.*
URL: <http://www.ibm.com/de/software/websphere>.
- [15] *IBM Websphere Application Server.*
URL: <http://www.ibm.com/de/software/websphere>.
- [16] KLEINOEDER, J. T. RIECHMANN: *Meta Objects for Access Control: Role-Based Principals.*
URL: <http://www4.informatik.uni-erlangen.de/riechman/papers/abstract-rbp.shtml>.
- [17] LEYMANN, FRANK: *Business in the Internet Age.* Vorlesung Universität Stuttgart.

- [18] *Microsoft Visual Studio .NET.*
URL: <http://www.microsoft.de>.
- [19] *Mono Projekt.*
URL: <http://www.go-mono.com>.
- [20] *.NET Enterprise Server.*
URL: <http://www.microsoft.com/net/products/servers.asp>.
- [21] *.NET Passport und E-Government.*
URL: <http://www.heise.de/newsticker/data/jo-19.04.02-000>.
- [22] *.NET Security.*
URL: <http://msdn.microsoft.com/library/en-us/dnnetsec/html/-netframesecover.asp?frame=true>.
- [23] *Oracle 9i Datenbank.*
URL: <http://www.oracle.com/de>.
- [24] PALYSZ, DARIUS: *Interoperability zwischen .NET und COM.*
MSDN TechTalk.
- [25] *PE Format Spezifikation.*
URL: <http://www.microsoft.com/hwdev/hardware/-PECOFF.asp>.
- [26] *Rational.*
URL: <http://www.rational.com>.
- [27] SOLLICH, PETER. 1. *.NET day Stuttgart.*
- [28] STAL, MICHAEL: *C# und .NET Tutorial - Teil I.* iX, 12, 2001.
- [29] STAL, MICHAEL: *C# und .NET Tutorial - Teil II.* iX, 01, 2002.
- [30] STAL, MICHAEL: *C# und .NET Tutorial - Teil III.* iX, 02, 2002.
- [31] *Teamdevelopment mit Visual Studio .NET und Visual Source Kapitel 1-7.*
URL: http://msdn.microsoft.com/library/en-us/dnbdta/html/-tdlg_ch1.asp.
- [32] *Visual Studio Compact Framework.*
URL: <http://msdn.microsoft.com/vstudio/device/compact.asp>.
- [33] WALKER, BANIASSAD MURPHY: *An Initial Assessment of Aspect-Oriented Programming. 21st International Conference on Software Engineering,* 120–130, 1999.
- [34] *Webgain Toplink.*
URL: <http://www.webgain.com/toplink>.
- [35] WIEDEKING, MICHAEL: *Die Chancen von .NET sind sehr hoch.*
dotnet-Magazin, 01, 2002.
- [36] WIEDEKING, MICHAEL: *Die neue Tonart - C# oder wie man auf Java und C++ noch eins drauf setzt Teil(1).*
dotnet-Magazin, 01, 2002.
- [37] WIEDEKING, MICHAEL: *Die neue Tonart - C# oder wie man auf Java und C++ noch eins drauf setzt Teil(2).*
dotnet-Magazin, 02, 2002.
- [38] *Windows .NET Server.*
URL: <http://www.microsoft.com/windows.NETserver>.

Erklärung

Hiermit versichern wir, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Daniel Friedrich / Sven Stitzelberger)