Seminar der Arbeitsgruppe

Datenbanken und Informationssysteme

im Wintersemester 2001/2002

DB-Aspekte des E-Commerce, Schwerpunkt: Anwendungen

Vergleich der E-Business Plattformen J2EE & .NET

Alexander Hilliger von Thile

Vergleich der E-Business Plattformen J2EE & .NET

Vergleich der E-Business Plattformen J2EE & .NET

Einführung

Wenn man Äpfel mit Birnen vergleicht und dabei der Apfel vorne liegt, so hat man den Vorteil, daß man deshalb von der Birne nicht gleich verklagt wird. Diesen Vorteil scheint man beim Vergleich von Multimillionen-Dollar-Technologien nicht zu haben, was sich in diesem Fall darauf auswirkt, daß sich nur umständlich verwertbare Informationen zum Vergleich finden lassen, die nicht vom Einfluß eines Herstellers geprägt sind. Leider sind aber in der Regel Produktbewertungen, sofern sie vom Hersteller selbst stammen, nicht repräsentativ. Eine klare Aussage, wie „unser Produkt kann das nicht“ wird sich kaum finden lassen und Aussagen der Art „das können die anderen nicht“ sind nicht immer glaubwürdig. Wenn es sich dann bei den Herstellern auch noch um die beiden größten IT-Dienstleister handelt, sollte die Problematik leicht nachvollziehbar sein.

Diese Auswertung beschränkt sich daher beim Vergleich nur auf die Angaben der Hersteller zum eigenen Produkt, die sich nachweisen lassen und bewertet die tatsächliche Funktionalität. Irrelevant sind insbesondere Erweiterungen und Pläne durch die marketingtechnisch geschickt versucht wird, von tatsächlichen Mängeln abzulenken.

Im Rahmen dieser Auswertung werden die E-Business Plattformen J2EE und .NET untersucht. Es werden die Technologien vorgestellt, Unterschiede erörtert und Stärken und Schwächen verdeutlicht. Um eine Bewertung vornehmen zu können, werden die Technologien anschließend in einem Lebenszyklusmodell exemplarisch untersucht. Dem Leser bleibt hierbei die Gewichtung der Ergebnisse selbst überlassen.

KAPITEL 1

1.1 Was sind E-Business Plattformen?

„Unsere Firma muss ins Internet“. Mit diesem Werbeslogan warb bis vor kurzem ein großer Hard- und Softwarehersteller. Trotz der einfachen Aussage des Satzes spiegelt er den Trend der letzten Jahre wider: electronic commerce. Die Möglichkeit von jedem Ort zu jeder Zeit Waren einzukaufen und Dienstleistungen in Anspruch nehmen zu können, war Grundlage vieler neuer Geschäftsideen.
Doch wie kam das Angebot damals ins Internet?
In den ersten Versionen bestanden die Systeme primär aus einer Datenhaltungskomponente, z.B. einer existierenden Artikel- und Kundendatenbank und einer Präsentationskomponente für die Ausgabe von html-basierten Webseiten. Während es für diese Komponenten bereits standardisierte Lösungen gab, stellte der Kontrollfluß, also die Business-Logic, die einen Benutzer von einer zur nächsten Seite führt, ein großes Problem dar. Bei kleinen Systemen wurde der Kontrollfluß über selbsterstellte Skripte gesteuert, große Systeme wurden auf Grundlage des common gateway interface (CGI) entworfen. Beide Realisierungsmöglichkeiten hatten jedoch entscheidende Nachteile. Skriptlösungen stellten sich bezüglich Wartung und Systemevolution als unbrauchbar heraus. CGI-basierte Lösungen waren aufwendig und teuer. In beiden Fällen wurden wichtige Punkte modernen Softwareengineerings, wie z.B. der Wiederbenutzung von Klassen und Komponenten unzureichend unterstützt. Ressourcen wurden durch Skriptinterpretation bei jedem Aufruf einer Seite verschwendet und Verbindungen zu externen Komponenten, wie z.B. Datenbanksystemen wurden pro Aufruf auf- und abgebaut und somit nicht optimal genutzt. Besondere Probleme bereitete die Heterogenität der Komponenten auf Grund fehlender Standards, weshalb extern bezogene Komponenten nur schwer in existierende Systeme eingebunden werden konnten und der Austausch von Daten zwischen den Systemen nur über Umwege realisiert werden konnte.

Aus dieser Misere entstand die Notwendigkeit nach offenen E-Business Architekturen. Es wird also deutlich, daß man bei .NET und J2EE keineswegs von einer Revolution ausgehen kann. Sie stellen vielmehr eine Evolution dar [VER01].

1.2 Anforderungen an E-Business-Architekturen

An E-Business Architekturen werden spezifische Anforderungen gestellt [SES01]. Auf diese soll im Folgenden weiter eingegangen werden:

Interoperabilität: Große E-Business-Systeme bestehen aus vielen Komponenten, die zum Teil fremd bezogen werden (Outsourcing). Diese Komponenten müssen Informationen untereinander austauschen können und sich in Systeme integrieren lassen. Des weiteren müssen E-Business-Systeme in der Lage sein, miteinander zu interagieren.

Verfügbarkeit: Verfügbarkeit ist eine sehr wichtige Anforderung an E-Business-Systeme. Ausfallzeiten, egal wie kurz, müssen vermieden werden. Das bedeutet, daß ein E-Business-System niemals unerreichbar sein darf, da dies mit hohen Kosten für den Betreiber verbunden ist. Auch Wartungsaufgaben und Evolution des Systems dürfen die Verfügbarkeit nicht beeinträchtigen.

Durchsatz: Performance ist eine weitere Anforderung an E-Business-Systeme. Die Zeit, die eine Transaktion in Anspruch nehmen darf, sollte minimal sein. Gerade bei dynamisch erzeugten Web-Seiten haben Marketingstudien gezeigt, daß Besucher eines Web-Shops bei Wartezeiten von mehr als fünf Sekunden leicht das Interesse verlieren. Gute E-Business-Systeme zeichnen sich daher auch durch eine hohe Anzahl von TPS (transactions per second) aus.

Entwicklungskosten: *Personalkosten* stellen bei der Softwareentwicklung den größten Posten dar. Daher liegt nahe, die Entwicklungszeit möglichst kurz zu halten. Um dies zu gewährleisten, sollten E-Business-Architekturen Bibliotheken mit den wichtigsten Funktionen für die unterschiedlichen Domänen zur Verfügung stellen.

Tools stellen einen weiteren wichtigen Faktor bei der Entwicklung dar, da sie komplizierte oder zeitaufwendige Arbeiten automatisieren, aber auch das beste Tool, kann Designschwächen der darunterliegenden Architektur nicht kompensieren.

Fehlersuche stellt sich bei großen Systemen, die aus mehreren Komponenten bestehen, besonders schwierig dar. Ausnahme-, Fehlerbehandlung und Debugging, auch über Komponentengrenzen hinweg, sind wichtig und sollten nicht nur auf Tool-, sondern auch auf Architekturebene vorgesehen sein.

Komponenten, aus denen große Systeme zusammengesetzt sind, sollten wiederverwendbar und leicht austauschbar sein. Klare Schnittstellenkonzepte sowie die Möglichkeit, Komponenten auch verteilt auszuführen sind eine wichtige Anforderung.

Kosten für Wartung: E-Business-Systeme sind einem schnellen Wachstum unterworfen. Anforderungen ändern sich schnell und müssen unkompliziert umsetzbar sein. Hierbei ist wichtig, daß die Verfügbarkeit durch Änderungen nicht beeinträchtigt wird. Komponenten stehen häufig über Workflows miteinander in Verbindung. Sinnvoll ist also, wenn die Architektur nicht nur einfache Komponentenänderungen, sondern auch Änderungen am Workflow unterstützt.

Arbeitsschrittkosten: Die Abarbeitung eines Workflows besteht aus (vielen) Transaktionen. Auf Grund hoher Benutzerzahlen und starken Antwortzeitbeschränkungen sollten einzelne Arbeitsschritte kostengünstig ausgeführt werden können, das bedeutet, daß zur Verfügung stehende Ressourcen sinnvoll (z.B. kooperativ) genutzt werden sollten, zeitkritische Operationen optimiert werden können und sich wiederholende Aufgaben (z.B. über Caching) vom System beschleunigt werden.

Skalierbarkeit: Die Architekturen müssen Skalierbarkeit unterstützen. Das gilt sowohl für Multiprozessorsysteme aber auch für verteile Systeme. Die Kommunikation zwischen den Systemen muß durch die Architektur unterstützt und nach Möglichkeit verwaltet werden.

Besondere Aufmerksamkeit gilt dem Punkt der Interoperabilität, denn wie bei der old economy werden in der new economy Waren und Dienstleistungen in den seltensten Fällen von einer Firma vollkommen allein erstellt. Es existiert vielmehr ein Netzwerk von Zulieferern und Zwischenprodukten. Im Falle von E-Business betrifft dies primär den Austausch von Information sowie die Möglichkeit eine Dienstleistung bei einem anderen System in Auftrag zu geben. Hierauf fixiert sich insbesondere Microsoft durch die Technologie der *webservices*, die bei J2EE in der Weise bei J2EE nicht behandelt werden.
Somit wird aus competition zunehmend collaboration [SES01], was eine neue Form von distributed computing darstellt [VER01].

Kapitel 2

2.1 Sun Microsystems und Microsoft: Die Kontrahenten im Vergleich

Die bedeutendsten Realisierungen der oben genannten Punkte stammen von Sun Microsystems mit J2EE und Microsoft mit .NET. Die Kontrahenten bedürfen keiner großen Vorstellung. Sun Microsystems gehört zu den weltweit größten Herstellern von Hard- und Software, vornehmlich im Serverbereich, und ist Erfinder der Java-Technologie. Sun definiert sich als „*we are the dot in* ***.****com*" als wichtige Kompetenz im E-Business.
J2EE stellt mehr als nur eine Weiterentwicklung der Java-SDKs (Software Development Kit) dar. Primäres Ziel ist es, den Grundstein der Java-Philosophie „*write once, run everywhere*" von der

Präsentationsebene, in Form von *Applets* und der Anwendungsebene als *Application* auf die Serverseite auszudehnen. Hierbei legt Sun besonderen Wert darauf, durch leistungsstarke Bibliotheken die Entwickler von häufig benutzten Funktionen von E-Business-Applikationen zu entlasten und somit die Entwicklung zu vereinfachen und Entwicklungszeit einzusparen. Java-Entwicklern stehen somit neue Möglichkeiten offen und Nicht-Java-Entwicklern wird ein Anreiz zum Umstieg gegeben.
Obwohl hier naheliegt, daß Sun als Hersteller von Servern ein Komplettsystem aus Hard- und Software ausliefert, ist dem in der Praxis nicht so. J2EE ist lediglich ein Standard und nicht mit einem Produkt zu verwechseln. Das bedeutet, daß man auf die Implementierung durch eine weitere Firma angewiesen ist. Diese können beispielsweise von IBM, Bea oder Borland stammen.

2.2 J2EE im Detail

Die J2EE Architektur lässt sich in vier Teile untergliedern. Auf Grund der teilweisen Überschneidung von Technologien bei J2EE und .NET werden hier die Pendants von Microsoft ebenfalls vorgestellt. Dies soll aber keines Falls einen der Hersteller des Nachmachens beschuldigen.

2.2.1 Java-language-Modell

Das Java-Language-Modell war ausschlaggebend für den Erfolg der Java-Technologie. Hierzu zählt, daß Java-Quelltexte nicht mehr direkt in ausführbare Programme übersetzt werden, die dann nur auf einer Plattform lauffähig sind, sondern daß eine Zwischenschicht erzeugt wird. Dieser *Bytecode* wird dann auf dem Zielsystem von einer *Virtual Machine* interpretiert oder mit Hilfe eines *Just-In-Time-Compilers* beim Start des Programmes übersetzt, um die Ausführungsgeschwindigkeit zu erhöhen.
Ein weiterer Erfolgsfaktor war die Sprache an sich, sie ist nahe an C++ angelehnt und kann somit von vielen Programmierern schnell erlernt werden. Durch das „*everything is a class*“-Konzept sind das API sowie mit Java erstellte Programme bezüglich der Schnittstellen leicht verständlich. Durch *Reflection* ist es möglich, jede Java-Klasse in einen String zu serialisieren und daraus wiederherzustellen, was die Entwicklung von verteilten komponentenbasierten Systemen vereinfacht.
Diese Konzepte finden sich in sehr ähnlicher Form bei .NET wieder (siehe Abschnitt 2.4.1).

2.2.2 Client-Programmiermodell

Das Client-Programmierungsmodell spezifiziert die Interaktion mit dem Browser. Bei Java gibt es hierfür drei Möglichkeiten:

Java-Applets: Hier wird der Java-Code direkt im Browser des Benutzers ausgeführt. Dies bietet Vorteile, falls Darstellungen oder Interaktionen mit dem Benutzer gewünscht sind, die sich mit HTML-Dateien nicht realisieren lassen, beispielsweise eine CAD-Komponente. In der Microsoftwelt wird dies durch ActiveX-Komponenten bewerkstelligt. Diese stoßen in der Praxis aber auf

heftigen Widerstand, da ActiveX-Komponenten im Gegensatz zu Applets vollen Zugriff auf das System des Clients haben. Eine Sandbox bei Applets verhindert den Zugriff bei Java. Microsoft begegnet diesem Risiko durch eine digitale Signatur. Der Benutzer kann somit vor Ausführung entscheiden, ob er dem Hersteller der Komponente vertraut, hat nach Bestätigung aber keinen Einfluß mehr auf die Aktionen der Komponente. Ein weiterer Vorteil von Applets liegt in der Plattformunabhängigkeit. Lediglich eine Virtual Machine muß vom Browser bereitgestellt werden.
Betrachtet man die E-Business-Portale im Internet, fällt allerdings auf, daß sowohl Applets, als auch ActiveX-Komponenten in der Praxis keine Rolle spielen. Dies soll sich aber nach dem Willen von Sun und Microsoft ändern.

JavaServlets und Java-Server-Pages Diese werden im Gegensatz zu Applets vom Server ausgeführt. Beide dienen im Wesentlichen dazu, HTML-Seiten dynamisch zu erzeugen. Sie sind vergleichbar mit ASP (Active Server Pages) und ASP+, siehe Abschnitt 3.1.1, bieten aber nur einen kleinen Teil der Funktionalität.

2.2.3 Middle-Tier Infrastruktur

Diese ist bei J2EE durch EJB (Enterprise Java Beans) repräsentiert und ist vergleichbar mit COM (Component Object Model) bzw. COM+ und dem Microsoft Transaction Server, der von Microsoft bereits 1996 veröffentlicht wurde.
Sun definiert EJBs folgendermaßen:

> *„The Enterprise JavaBeans architecture is a **component based architecture** for the **development** and **deployment** of **component-based distributed business applications**. Applications written using the EJB architecture are **scalable**, **transactional and multi-user secure**. These Applications may be **written once**, and then deployed on any server platform that supports the EJB specification.“*

Die Kernpunkte werden hier deutlich.

2.2.4 Enterprise API

Das Enterprise API lässt sich in zwei Teile untergliedern:

1. **das Sichtbare**: Hierzu zählen *JDBC (Java Database Connectivity)* für Datenbankzugriffe, was bei .NET durch *ADO.NET (ActiveX Data Objects)* realisiert wird, *JNDI (Java Naming and Directory Interface)*, das vergleichbar mit *ADSI (Active Directory Service Interface)* ist und *JMS (Java Message Service)* zur Steuerung von asynchronen Prozessen, das bei Microsoft durch das *Message Queue API* realisiert wird.

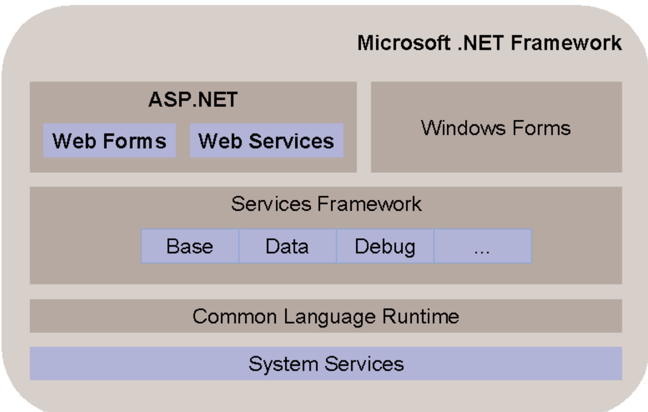
2. **das Unsichtbare**: Hierzu gehört das *Java Connector API*, das festlegt, wie andere Produkte in J2EE eingebunden werden können. Ein Beispiel stellt das *JTA (Java Transaction API)* dar.

2.3 Microsoft

Microsoft gehört zu den bekanntesten Marken der Welt. Besonders erfolgreich sind die Produkte der Windows- und Officereihe. Microsoft hat einen Marktanteil von über 90% der PC-Betriebssysteme. In den letzten Jahren hat Microsoft u. a. durch Marketing und eine international in die Kritik geratene Produktbündelung den Internet-Explorer zum weitverbreitetsten Web-Browser gemacht.

Im Serverbereich hingegen hat Microsoft in der Vergangenheit lediglich eine untergeordnete Rolle gespielt. Sowohl Serverbetriebssysteme als auch der sog. Microsoft Backoffice-Bereich, wozu z.B. Datenbankserver zählen, setzte sich mit weit weniger Erfolg durch. Größter Konkurrent ist hier das freie Betriebssystem Linux, das sich sowohl durch seinen Preis als auch durch seine Stabilität auszeichnet.

Doch glaubt man der Roadmap, so soll sich das in den nächsten Jahren ändern. .NET stellt nach Meinung von Bill Gates eine ähnliche Revolution dar, wie der Wechsel von DOS zu Windows. .NET ist eine Weiterentwicklung von *Windows DNA 2000 (Distributed Internet Application Architecture)* und bietet im Gegensatz zu J2EE eine Produktfamilie zur vollständigen Entwicklung von web-basierten Anwendungen, die in den kommenden Jahren Einzug in alle Produktlandschaften von Microsoft, sowohl im Betriebssystem-, als auch im Backofficebereich einhalten wird.
In den Fokus rückt Bill Gates den Web-Browser als Instrument der Informationsverarbeitung. Er zeichnet sich bereits heute durch die einfache Bedienbarkeit und Konnektivität aus, dient aber zur Zeit hauptsächlich als Lesewerkzeug. Dies soll sich hingegen ändern: „*empower people through great software anytime, any place and on any device*“.



2.4 .NET im Detail

.NET läßt sich in fünf Teile untergliedern:

2.4.1 .NET Infrastruktur und Werkzeuge

Hierzu zählen das .NET Framework, die .NET Enterprise Server und Visual Studio .NET:

Das .NET Framework stellt die Kernkomponente der .NET Plattform dar. Es ist stark mit dem Betriebssystem gekoppelt und beinhaltet die komponentenorientiere middle-tier Infrastruktur *COM+* sowie die CLR (Common Language Runtime). Die *CLR* hat viele Gemeinsamkeiten mit der *Virtual Machine* von Java (siehe 2.2.1). Programme, die im Gegensatz zu Java, in allen verbreiteten Programmiersprachen geschrieben sein können, werden in eine Zwischensprache *MSIL (Microsoft Intermediate Language)* übersetzt. Auf dem Zielsystem werden *Assemblies* (enthalten die *MSIL* Dateien), unter denen Microsoft eine „*group of resources and types*" versteht und die Java-Entwicklern als *JARs (Java Archive)* bekannt sind mit Hilfe *eines Just-In-Time-Compilers* ausgeführt. Eine zeitaufwendige Interpretation, wie bei Java, ist nicht vorgesehen. Obwohl man .NET Applikationen theoretisch in jeder Programmiersprache entwickeln können soll, favorisiert Microsoft die Sprache C# (gesprochen: C Sharp, was wohl als *Sehe schärfer*, *Halbton höher als C* oder *C++++* verstanden werden soll). C# erinnert sehr an Java, vereint aber nach Microsoft das Beste aus C, C++, Java und Delphi. Hauptvorteil von C# sollen, wie bei Java, Typsicherheit, Fehlerbehandlung und integrierte automatische Speicherbereinigung sein. Zusätzlich vereinfacht C# die Versionierung und bietet neben Methods, Fields und Properties auch Events, unterstützt Pointer, conditional compilation und Operatorüberladung (siehe Abbildung 2.4.1).

Code	Erläuterung
`namespace dbis.test`	Namespaces bieten Funkionalitäten wie Packages.
`attribute [Transaction(TransactionOption.Required)]`	Transaktionseigenschaften können direkt im Code angegeben werden. Externe Descriptoren sind nicht notwendig.
`public class Person: IPerson {`	Sichtbarkeiten und Vererbung erinnern an Java.
`[PersonFirstName] Vorname;` `private int dob;` `private int[,] twoDimArray;`	Persitenz direkt im Quelltext definierbar.
`struct Address {` `String Street;` `...` `void moveTo(String Street...){` `...` `}` `}`	Structs (wie bei C) bieten trotz Klassen besondere Vorteile: sie unterstützen keine Vererbung, werden aber wesentlich schneller verarbeitet als Klassen, sogar Methoden sind möglich.

Abbildung 2.4.1 C# im Vergleich zu Java

`///<summary>`
`///This does something`
`///</summary>`
	„Triple Slash“: das Pendant zu JavaDoc diese Kommentare werden exportiert.

`public Person() {`
`  try {`
`  } catch(...) {}`
`  finally {}`
`}`
	Exceptionhandling wird unterstützt.

`public DateOfBirth {`
`  get {return dob};`
`  set {dob = value};`
`}`
	Sehr praktisch: getter/setter Methoden.

`unsafe void f(byte[] b) {`
`  ...`
`}`
	„Unsafe code“ erlaubt Zugriff auf Pointer.

`#ifdef debug`
`  public void foo {`
`    Console.Write( 256.ToString() );`
`  }`
`#endif`
`}`
	„Conditional compiling“ optimiert den Code.
	
	Auch bekannt von Java: alles ist ein Objekt.

Fortsetzung

Trotz der Vorteile, die eine solche Sprache bietet, ist es durchaus in geschwindigkeitskritischen Anwendungen sinnvoll, auf gewisse Sicherheiten zu verzichten und (wie früher) z.B. direkt auf Speicherbereiche zugreifen zu können. Das ist mit Java undenkbar. .NET bietet hier die Vorzüge von beidem: Komponenten können als *managed* und *unmanaged* deklariert werden.

Stark vereinfacht hat sich die Kommunikation zwischen verschiedenen Komponenten. COM+ benötigt keine Einträge mehr in die zentrale Windows Registry, womit die sog. DLL-Hell (Bibliotheken verschiedener Hersteller unterschiedlicher Versionen haben alle den gleichen Namen), ein Hauptproblem der alten Windowsarchitektur, gelöst wird.

Theoretisch ist das CLR plattformunabhängig. Es ist sogar eine teilweise Portierung der Kernkomponenten auf andere Betriebssysteme geplant. Microsoft konzentriert aber seine Entwicklung auf die Windowsplattform.

Visual Studio ist die integrierte Entwicklungsumgebung für .NET Applikationen. Eine genauere Betrachtung der Möglichkeiten von Visual Studio findet in Abschnitt 3.2.1 statt.

2.4.2 .NET Foundation Services

Foundation Services sind eine Menge von Diensten, die über das Internet genutzt werden. Beispiele hierfür sind Passport.NET für zentrale Benutzerauthentifizierung, sowie Datei- und Terminmanagement. Auch wenn die Dienstangebote, wie Passport, in jüngerer Zeit in die Kritik geraten sind, da Datenschützer den gläsernen Surfer befürchten, so ist der Grundgedanke der Dienstleistung über das Internet in den Augen Microsofts sehr wichtig.

2.4.3 .NET User Experience

Hierzu zählen die Microsoft Windows XP Produkte: Windows XP, Office XP, etc.

2.4.4 .NET Geräte

Diese Geräte sind in der Lage die .NET Dienste zu nutzen. Zur Zeit sind das alle Windowsplattformen der XP-Generation. Ältere Windowsversionen können durch entsprechende Servicepacks nachgerüstet werden. Handheld-PCs mit Windows CE sollen ebenfalls .NET Dienste nutzen können.

2.4.5 .NET Enterprise Server

Die .NET Enterprise Server dienen als Erweiterung zu .NET und sind separat erhältlich. Zu den bekanntesten gehört der *Microsoft SQL Server,* der sich durch hohe Performance und Verfügbarkeit auszeichnen soll, Phrasen, die auch auf so mancher Windowspackung standen. Und auch wenn die Zweifel unbegründet sein mögen, so stehen dem Nutzer auch andere Datenbankserver, die die ADO.NET Schnittstelle unterstützen, zur Verfügung.
Weitere wichtige Server sind der *Application Center Server*, der sowohl *Cluster-Koordinator* und *Cluster-Manager* [STAL0] ist und somit unterbrechungsfreie Verfügbarkeit und einfache Skalierbarkeit des Systems garantiert. Mit Hilfe des *ISA Server (Internet Security and Acceleration Server)* wird durch Caching von HTML-Seiten die Performance erhöht und durch Firewallfunktionalität das System nach außen gesichert.

Innovativ ist der BizTalk Server, der zur Integration und Orchestration dient. Hierunter wird die Zusammenfassung von Diensten zu Workflows verstanden. Somit wird die Übersichtlichkeit des Gesamtsystems deutlich erhöht und Wartungsaufgaben vereinfacht.

KAPITEL 3

Problematik beim Vergleich

Es lässt sich klar feststellen, daß Microsoft und Sun mit ihren E-Business-Architekturen ganz unterschiedliche Ziele verfolgen. Die Roadmap [MSNET0] von Microsoft zeigt deutlich, daß .NET fundamentale Änderungen für die Microsoftwelt bedeutet, während sich für Javaentwickler die Änderungen in Grenzen halten. Diese unterschiedlichen Motivationen und Zielsetzungen werfen Probleme beim Vergleich der beiden Architekturen auf. Daher wird im Folgenden der Vergleich anhand von zwei Methoden erfolgen:

1. Anhand eines allgemeinen Modells werden die Architekturen eingeordnet, ihre Technologien vorgestellt, abgegrenzt und Unterschiede herausgearbeitet.
2. Durch ein Fallbeispiel soll von der Installation bis zur Wartung ein Beispielprojekt untersucht werden. Hierbei wird versucht, die oben vorgestellten Technologien bzgl. ihrer Alltagstauglichkeit zu bewerten.

Dieses Vorgehen soll die Hauptproblematik, daß es sich bei J2EE um einen Standard und bei .NET um ein Produkt handelt, lösen, indem im 1. Teil von .NET lediglich die Architektur betrachtet wird und im 2. Teil J2EE in Form von kommerziell erhältlichen Produkten mit .NET verglichen wird.

3.1 Vergleich der Technologien

Sowohl .NET als auch J2EE unterstützen sog. multi-tier Architekturen (siehe Abbildung 3.1). Sie bestehen hauptsächlich aus den Schichten Präsentation (Presentation Tier), Geschäftslogik (Business Logic Tier) und Datenhaltung (Database Tier).

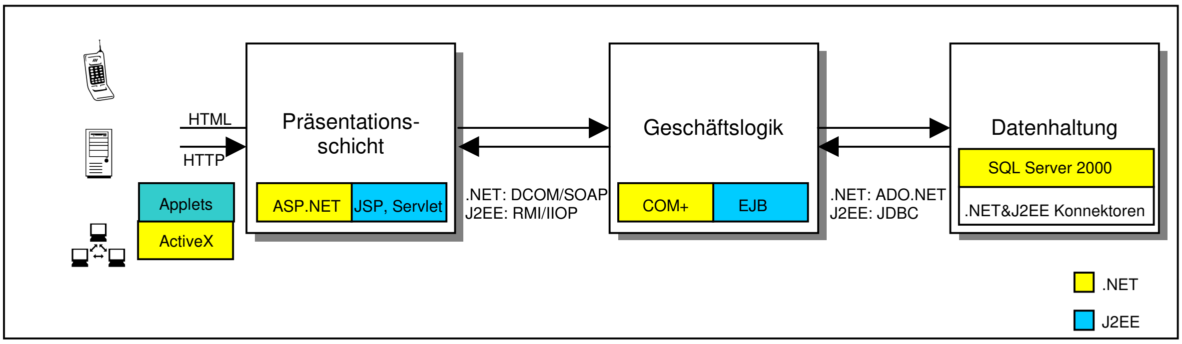


Abbildung 3.1 Übersicht über die drei wichtigsten *tiers* mit ihren Technologien sowie die Kommunikationsbeziehungen und Protokolle zwischen ihnen

3.1.1 Präsentationsschicht:

Die Präsentationsschicht steuert die Interaktion mit den Clients. Als Kommunikationsprotokoll hat sich HTTP etabliert, da es bereits weit verbreitet ist und der HTTP-Port von vielen Firewalls nicht blockiert wird. Die Präsentationsschicht empfängt vom Client (dies ist i. d. R. ein Web-Browser) einen HTTP-Request und antwortet mit einer HTML-Seite. Obwohl die Möglichkeiten von HTML bezüglich der Anzeige von Daten für die meisten E-Business-Lösungen ausreicht, so stellt die Eingabe von Daten, sofern sie von den Standard-Formularelementen abweicht, ein Problem dar. Durch Erweiterungen, bei J2EE als Applet und bei .NET durch ActiveX (siehe Abschnitt 2.2.2) repräsentiert, sind theoretisch beliebig komplexe Interaktionen mit dem Benutzer möglich. Beide Lösungen bieten jedoch, wie eingangs beschrieben (siehe Abschnitt 2.2.2), erhebliche Nachteile und haben in existierenden E-Business-Portalen bisher nur wenig Einzug gefunden. Dennoch werden die Entwicklungen sowohl von Sun als auch von Microsoft in dieser Richtung vorangetrieben. Wichtig ist hierbei, daß die Darstellung

vom Zielsystem gesteuert wird. Eine Applikation soll, wie auch eine HTML-Seite, sowohl auf Desktop- und Palmtopsystem korrekt angezeigt werden, ohne daß der Entwickler explizit auf jede Plattform eingeht.
Javaentwicklern ist dies bereits durch *Layouts* bekannt. Der Entwickler fügt einem *Layout* Komponenten hinzu, das *Layout* entscheidet aber, wo und wie diese angezeigt werden. Bei .NET findet sich ein ähnliches Vorgehen: Das *Universal Canvas* ist ein Medium, auf dem entweder web-basierte (*WebForms*) oder windows-basierte (*WinForms*) Komponenten aufsetzen. Applikation und Webseite fließen also bei .NET zusammen, was den Browser zur universalen Applikation macht. Benutzeroberfläche klingt in diesem revolutionären Zusammenhang etwas antiquiert und wird vom Microsoft in „*User Experience*" umbenannt.
Hauptunterschied zu Java ist, dass Win-und WebForms innovative Interaktionstechnologien, wie z.B. Handschrifterkennung (sinnvoll bei PDAs, Personal Digital Assistent) und Spracherkennung unterstützen.

Zur Zeit werden in E-Business-Portalen allerdings hauptsächlich pure HTML-Seiten verwendet. Diese werden größtenteils dynamisch erzeugt. Wichtig hierbei ist, daß diese Seiten einfach zu entwerfen und zu warten sind und daß sie vom Server schnell erzeugt werden können. Da das HTML-Layout von Designern und die Logik von Programmierern erstellt wird, ist eine Trennung von Präsentation und Logik sinnvoll. Die Realisierungen von Sun und Microsoft unterscheiden sich nur marginal. J2EE bietet *JSPs (JavaServerPages)*, die zu *Servlets* kompiliert werden. Somit entfällt die Interpretation bei jedem Seitenaufruf. *ASPs (Active Server Pages)*, das Pendant von Microsoft, boten diesen Vorteil in der Vergangenheit nicht. Dies wurde bei ASP+ (bzw. ASP.NET) hinzugefügt. Beide Technologien unterstützen die Trennung von Layout und Logik. Bei J2EE können über Tags JavaBean-Komponenten genutzt werden. Des weiteren können die existierenden Tags erweitert werden *(Custom Tag Library)*. Bei ASP.NET können COM+ Komponenten genutzt werden. Ein Vorteil gegenüber JSP ist, daß Skripte in mehreren Sprachen verfasst werden können (siehe Abb. 3.1.1):

```
<html>
  <script language=“VB” runat=server>
    Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
      Message.Text = “Hi ” & Name.Text
    End Sub
  </script>
  <body>
    <form action=“thisfile.aspx” method=post runat=server>
      <h3> Name:
      <asp:textbox id=“Name” runat=server/>
      <asp:button type=submit text=“LookUp”
                  OnClick=“SubmitBtn_Click” runat=server/>
      <p>
      <asp:label id=“Message” runat=server/>
    </form>
  </body>
</html>
```

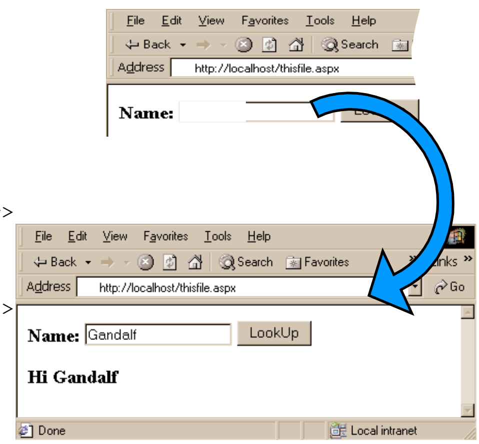


Abbildung 3.1.1a Beispiel einer Active Server Page (ASP)

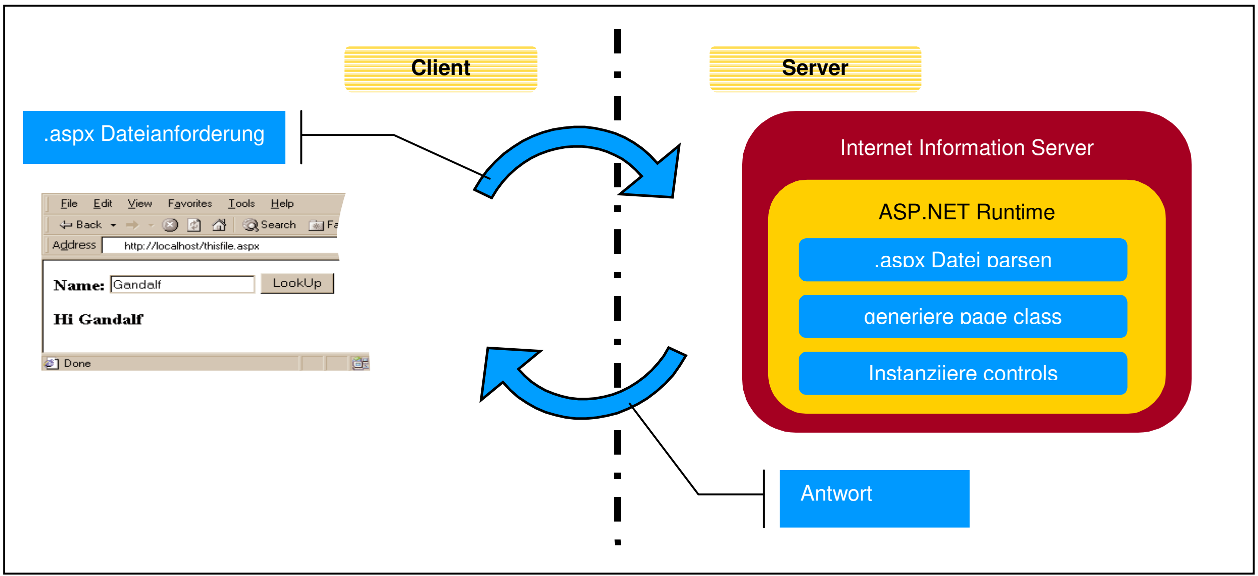


Abbildung 3.1.1b Verarbeitung einer Active Server Page (ASP)

Quelle: Microsoft .NET Developer Tools Readiness Kit

Übersicht über die Technologien

	ASP, ASP.NET Technologie	JSP Technologie
Web Server	Microsoft Internet Information Server (IIS), Personal Web Server	Apache, Netscape, IIS, ...
Plattform	Microsoft Windows	Microsoft Windows, Solaris, Mac OS, Linux, Unix (versch.)
Cross-Plattform-Komponenten	Nein (später evtl. COM+)	(Enterprise) Java Beans
Absturzsicherheit	Teilweise	Immer
Skriptsprachen	VBScript, JSkript, C#, ...	Nur Java
Datenbankanbindung	ODBC, ADO.NET	ODBC, JDBC
Legacy-Datenbankanbindung	Ja (COM, COM+)	Ja (JDBC API)

3.1.2 Geschäftslogik

Die Präsentations-Schicht kommuniziert mit der Geschäftslogik-Schicht über ein proprietäres Komponentenkommunikationsprotokoll. Im Falle von .NET sind dies DCOM (Distributed Component Object Model) und SOAP (Simple Object Access Protocol), das wie alle .NET Kommunikationstechniken auf XML basiert. Bei J2EE findet die Kommunikation über Java-RMI (Remote Methode Invocation) und IIOP (Internet Inter ORB Protocol) statt.
Sowohl bei .NET als auch bei Java ist die Geschäftslogik in Komponenten gekapselt. Eine Komponente bietet nach außen hin wohldefinierte Schnittstellen, modularisiert die Entwicklung und vereinfacht die Wiederbenutzung. Da diese Komponenten zahlreiche Ressourcen, wie etwa Datenbank- und TCP/IP-Verbindungen aber auch Threads benötigen, ist es sinnvoll, diese vom System verwalten zu lassen. Ein weiterer Aspekt ist, daß diese Komponenten teilweise Transaktionseigenschaften haben, die überwacht und koordiniert werden müssen. Komponenten werden daher bei J2EE in sog. Containern ausgeführt, bei .NET durch ein MTS-Executive (Microsoft Transaction Server) .

	.NET middle-tier Technologie	J2EE middle-tier Technologie
Komponenten	COM+	EJB
Infrastrukturumgebung	MTS-Executive	Container
Interface	IClassFactory Interface	Home-Interface
Sessionmanagement	IObjectContext	javax.ejb.SessionContext javax.ejb.EntityContext
Transaktionskoordinator	DTC (Distributed Transaction Coordinator)	(JTS) Java Transaction Server
Transaktionstyp/Persitenz	automatic transaction	Client/Container/Bean – managed persistence

Hauptunterschied zwischen MTS/COM+ und EJB ist, daß Sessionmanagement und Persistenz bei MTS automatisch gesteuert werden, was die Entwicklung vereinfachen soll.

3.1.3 Datenhaltung

Die Geschäftslogik-Schicht kommuniziert mit der Datenhaltungs-Schicht im Falle von J2EE per JDBC und bei .NET über ADO.NET. An die Datenhaltungsschicht werden hohe Anforderungen an Geschwindigkeit und Zuverlässigkeit gestellt. Auf Grund der Komplexität eines Datanbanksystems wird dieses bei J2EE in dieser Schicht nicht implementiert. Es werden lediglich Konnektoren vorgesehen, um Datenbanksysteme anzubinden. Microsoft bietet hier seinen SQL-Server und ermöglicht somit eine komplette E-Business-Architektur „aus einer Hand“, was für Kunden besondere Vorteile bzgl. Support hat.
Zu dieser Schicht gehören ebenfalls verteilte Namensdienste, wie beispielsweise Java Naming and Directory Interface (JNDI) und Active Directory Service Interface (ADSI).

	.NET database-tier	J2EE database-tier
Verteilte Transaktion	DTC	JTS
Relationales DB API	ADO.NET	JDBC
Datenbanksystem	SQL-Server (separat erhältlich)	-
Verteilte Namensdienste	ADSI	JNDI

3.2 Die Architekturen in der Praxis

Im zweiten Teil des Vergleiches wird .NET mit J2EE in Form von kommerziell erhältlichen Produkten anhand eines Lebenszyklusmodells verglichen. Dieses Modell umfaßt die Schritte von der Planung und Entwicklung des Produktes sowie über die Inbetriebnahme und Wartung.

3.2.1 Planung: time to market

Eine Hauptrestriktion bei Softwareentwicklung ist die Entwicklungszeit. Diese muß aus Kostengründen kurz gehalten werden, bzw. entscheidet, ob das eigene Produkt vor der Konkurrenz fertig ist. Hierauf haben die Architekturen nur wenig Einfluß, dies ist Aufgabe von Werkzeugen (Tools). Hierbei gibt es zwei wichtige Punkte zu beachten:

1. Die beste Architektur ist nutzlos, wenn sie nicht durch leistungsstarke Werkzeuge unterstützt wird.
2. Auch das beste Werkzeug kann Designschwächen der Architektur nicht vollständig kompensieren.

Microsoft bietet für die Entwicklung von .NET Software eine breite Palette an Werkzeugen aus einer Hand. Das Zugpferd ist Visual Studio.NET (siehe Abb. 3.2.1) das mit zahlreichen Erweiterungen (beispielsweise Visio) den gesamten Softwareentwicklungsprozess unterstützt. Praktische Funktionen, wie aus Diagrammen (UML, ER, ...) Quelltexte zu erzeugen und andersherum, erleichtern die Dokumentation.

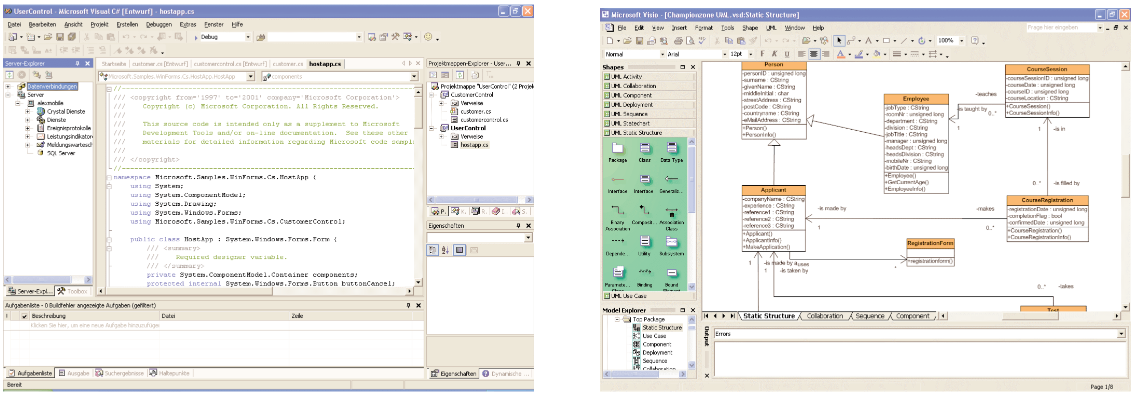


Abbildung 3.2.1 VisualStudio.NET und Microsoft Visio

Für Java existieren ebenfalls sehr durchdachte Lösungen, beispielsweise von Borland mit dem JBuilder, Together Soft mit Together sowie zahlreichen anderen und Sun selbst mit Forte.

<u>3.2.2 Entwicklung</u>

Software wird in den seltensten Fällen komplett neu entwickelt. Daher ist es sinnvoll, existierende Komponenten in ein neues System einbinden zu können. Sowohl bei J2EE als auch bei .NET kann man existierende Komponenten weiterverwenden (Abwärtskompatibilität). .NET hat aber hier klare Vorteile, da es von der Programmiersprache unabhängig ist. Cobol-Programmierer können ihre Komponenten ohne Mehraufwand weiterentwickeln während sie bei J2EE darauf angewiesen wären, diese erst in Java zu übersetzten (von Legacyanbindung über CORBA wird hier abstrahiert). Dies kann sehr teuer werden, da nicht nur die Entwicklungszeit, sondern auch die Umschulungen bezahlt werden müssen.

<u>3.2.3 Inbetriebnahme</u>

Nachdem die Entwicklung abgeschlossen ist, erfolgt die Installation auf dem Zielsystem. Vorteilhaft bei J2EE ist die Plattformunabhängigkeit. .NET ist auf die Windowsplattform beschränkt. Ein

Systemabsturz durch Fehler im Programm ist bei J2EE wegen der Interpretation prinzipiell ausgeschlossen, bei .NET theoretisch auch, wenn man nur managed comonents benutzt. Inwieweit Sun und insbesondere Microsoft ihre Stabilitäts- und Sicherheitsversprechen einhalten wird die Praxis zeigen. Tatsache ist aber, daß sich in letzter Zeit Microsoftprodukte allzu oft wegen kritischer Sicherheitslücken in den Schlagzeilen befinden (diese Aussage bezieht sich auf den Internet Information Server, sowie einen Wurm, der sich über den SQL Server verbreitet).

3.2.4 Wachstum/Skalierbarkeit/Wartung

Sowohl J2EE als auch .NET erfüllen die Skalierbarkeitsanforderungen problemlos. Erste Tests ergaben jedoch kleinere Schwierigkeiten, die sich auf Ungenauigkeiten in der Dokumentation und Implementierungsfehler zurückführen lassen. Diese werden jedoch in folgenden Versionen (Updates) behoben.

KAPITEL 4

Bewertung

Im Abschnitt 1.2 wurden fundamentale Anforderungen an E-Business-Architekturen vorgestellt und wie in den nachfolgenden Kapiteln deutlich wurde sowohl von .NET als auch von J2EE umgesetzt. Wie ebenfalls deutlich wurde, gibt es keinen klaren „Gewinner“. Beide Architekturen bieten Vor- und Nachteile. Bei der Entscheidungsfindung helfen aber derartige Aussagen nicht weiter. Deshalb wird in diesem Kapitel ein Ansatz vorgestellt, der es ermöglichen soll für die gegebene Problemstellung die bessere Architektur auszuwählen.

4.1 Vor- und Nachteile von J2EE

Vorteil	Nachteil
J2EE ist plattformunabhängig. „write once run everywhere“	J2EE ist nur eine Spezifikation. „J2EE“ alleine kann man nicht einsetzen.
Anwendungen sind nicht auf die Implementierung eines Herstellers beschränkt. (Flexibilität)	J2EE ist javazentriert. Andere Programmiersprachen werden nicht unterstützt
Java-Programme sind sehr robust. Es kann weder zu Speicherlecks noch zu Abstürzen kommen	Java bietet keine Unterstützung von hardwarenaher Programmierung. Java-Programme sind durch fehlende direkte Zugriffe beispielsweise in den Hauptspeicher ineffizienter
Die Java Connector Architecture bietet Anbindungsmöglichkeiten von Legacy-Systemen	
Java ist weit verbreitet und wird von vielen Programmierern beherrscht	

4.2 Vor- und Nachteile von .NET

Vorteil	Nachteil
.NET ist eine Produktfamilie: „alles aus einer Hand“. Dennoch gibt es nach außen hin dokumentierte und spezifizierte Schnittstellen zur Interoperabilität	.NET ist praktisch auf die Windowsplattform beschränkt
Alle verbreiteten Programmiersprachen werden unterstützt	C# ist sehr neu und muss im Gegensatz zu Java von den meisten Programmierern erst gelernt werden
.NET ist effizienter als J2EE da es keine Interpretation gibt und Komponenten hardwarenah (unsafe) entwickelt werden können	„User Experience“ ist noch nicht weit verbreitet

4.3 Wahl der Architektur

Die Wahl der Architektur ist von verschiedenen Faktoren anhängig. In diesem Abschnitt wird eine Übersicht über die wichtigsten gegeben, die endgültige Bewertung wird aber dem Leser überlassen.

1. Gibt es einen Vorgänger des zu entwickelnden Systems: Falls dieser in Java entwickelt wurde, ist der Einsatz von J2EE empfehlenswert. Bei anderen Programmiersprachen empfiehlt sich der Einsatz von .NET, sofern keine Neuentwicklung oder eine Legacyanbindung erwünscht sind.

2. Gibt es Vorgaben für die Plattform: Steht keine Windowsplattform zur Verfügung ist .NET nicht geeignet. J2EE unterstützt alle verbreiteten Plattformen

3. Welchen Kenntnisstand haben die Entwickler: Sowohl C# als auch Java sind einfach zu erlernen, falls Java oder C#-Kenntnisse vorhanden sind, eignet sich sowohl der Einsatz von .NET, als auch J2EE. Lediglich bei Basic, C und Cobol kann der Einsatz von .NET Vorteile bringen.

4. Ist Herstellerunabhängigkeit gefordert: .NET wird nur von Microsoft entwickelt. J2EE wird von Sun, IBM, Bea, Borland und vielen anderen Firmen unterstützt. Der Wechsel zwischen Applicationservern ist durch die Spezifikation von Sun unproblematisch.

5. Hardwarenahe Programmierung ist bei E-Commerceanwendungen in der Regel nicht notwendig. Wird aber der Einsatz von Pointern oder der direkte Zugriff auf Hardwareressourcen gefordert, so bietet .NET Vorteile gegenüber J2EE.

KAPITEL 5

Ausblick

.NET und J2EE waren weder die ersten noch werden sie die letzten Architekturen für E-Business-Systeme sein. Sun, die Firmen die auf J2EE aufsetzen und Microsoft haben viel Geld in die Entwicklung der Architekturen investiert und sind dementsprechend auch an einer hohen Marktdurchdringung interessiert.
Wie die Vergangenheit gezeigt hat, so hat es bei Sun und Microsoft bisher noch keinen klaren Gewinner gegeben. Vorteile des Einen wurden vom Anderen nach kurzer Zeit übernommen und somit ist auch nicht davon auszugehen, daß eine der Architekturen langfristig der anderen überlegen sein wird.

QUELLEN

[SES01] Roger Sessions, „Java 2 Enterprise Edition versus The .NET Platform. Two Visions for eBusiness", ObjectWatch Inc. 2001

[VER01] Chad Vawter, Ed Roman, „J2EE vs. Microsoft .NET. A comparison of building XML-based web services", TheServer-Side.com 2001

[RMH00] Richard Monson-Haefel, „Enterprise Java Beans 2nd Edition", O'Reilly 2000

[STAL0] Michael Stal, „Microsoft .NET Evolution oder Revolution", OBJEKTspektrum 6/2000

[FAR00] Jim Farley, „Microsoft .NET vs. J2EE: How do they stack up?", O'Reilly & Associates Inc. 2001

[FAR01] Jim Farley, „Picking a Winner: .NET vs. J2EE", CMP Media LLC 2001

[ALW1] Al Williams, „Choosing between the .NET and J2EE Frameworks", CMP Media LLC 2001

[SUN00] „Comparing JavaServer Pages and Microsoft Active Server Pages Technologies", Sun Microsystems 2001

[MS00] „Comparing Microsoft Transaction Server to Enterprise Java Beans", Microsoft Corporation 2000

Internetlinks:

[MSNET0] www.microsoft.com, Products, „.NET"

[SUNJ2EE0] www.java.sun.com , Products, „J2EE" oder Link „API & Downloads", J2EE

Stand der Internetlinks 1.1.2002