



# Strategievergleich: .NET versus Java

**Dr. Thomas Wieland**

(unter Verwendung von Material von M.Stal)

Siemens AG, Corporate Technology, München

[thomas@drwieland.de](mailto:thomas@drwieland.de)

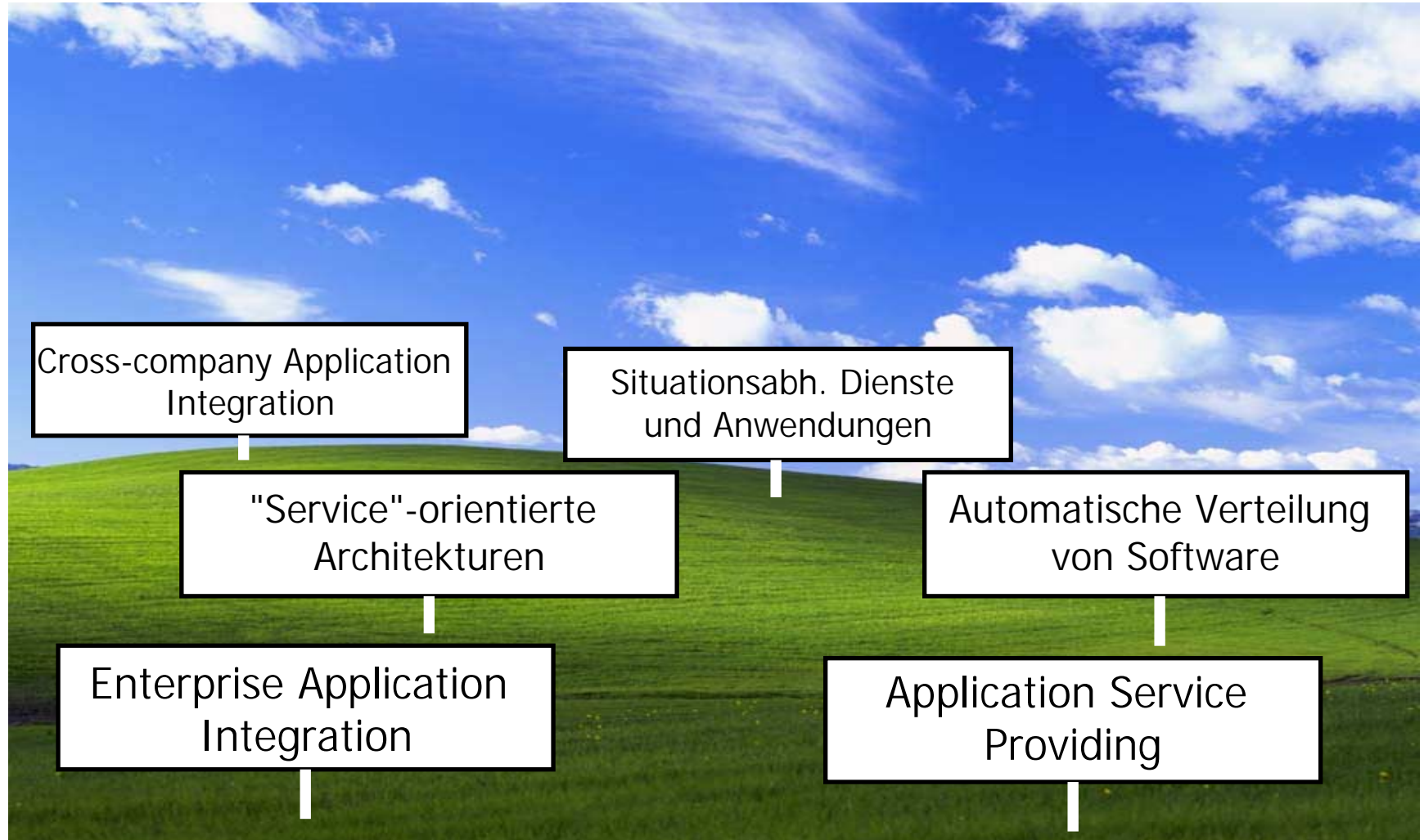
# Gliederung

- Herausforderungen für moderne Software-Architekturen
- Vergleich der Visionen:  
Sun ONE vs. Microsoft .NET
- Technologische Gegenüberstellung der Java-Plattform mit .NET
- Nicht-technischer Vergleich der Plattformen
- Fazit und Empfehlungen



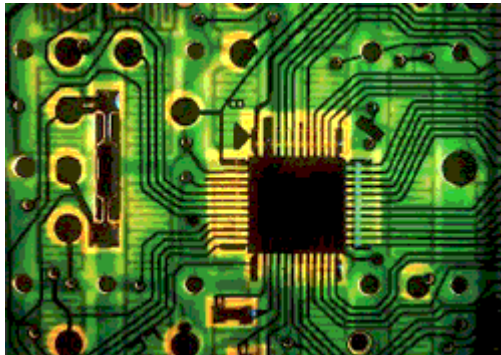
# Herausforderungen für moderne Software- Architekturen

# Anwendungslandschaft von morgen



# Agile Software

Heute: Software ebenso  
hart wie Hardware



Ziel: Software muss sich dynamisch  
an veränderte Anforderungen  
anpassen

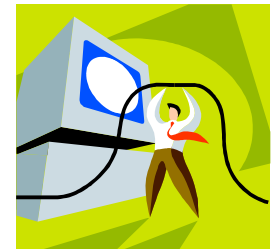
- Möglichst ohne Herunterfahren des Systems
- Veränderungen nicht nur durch Entwickler

# Heterogene Plattformen

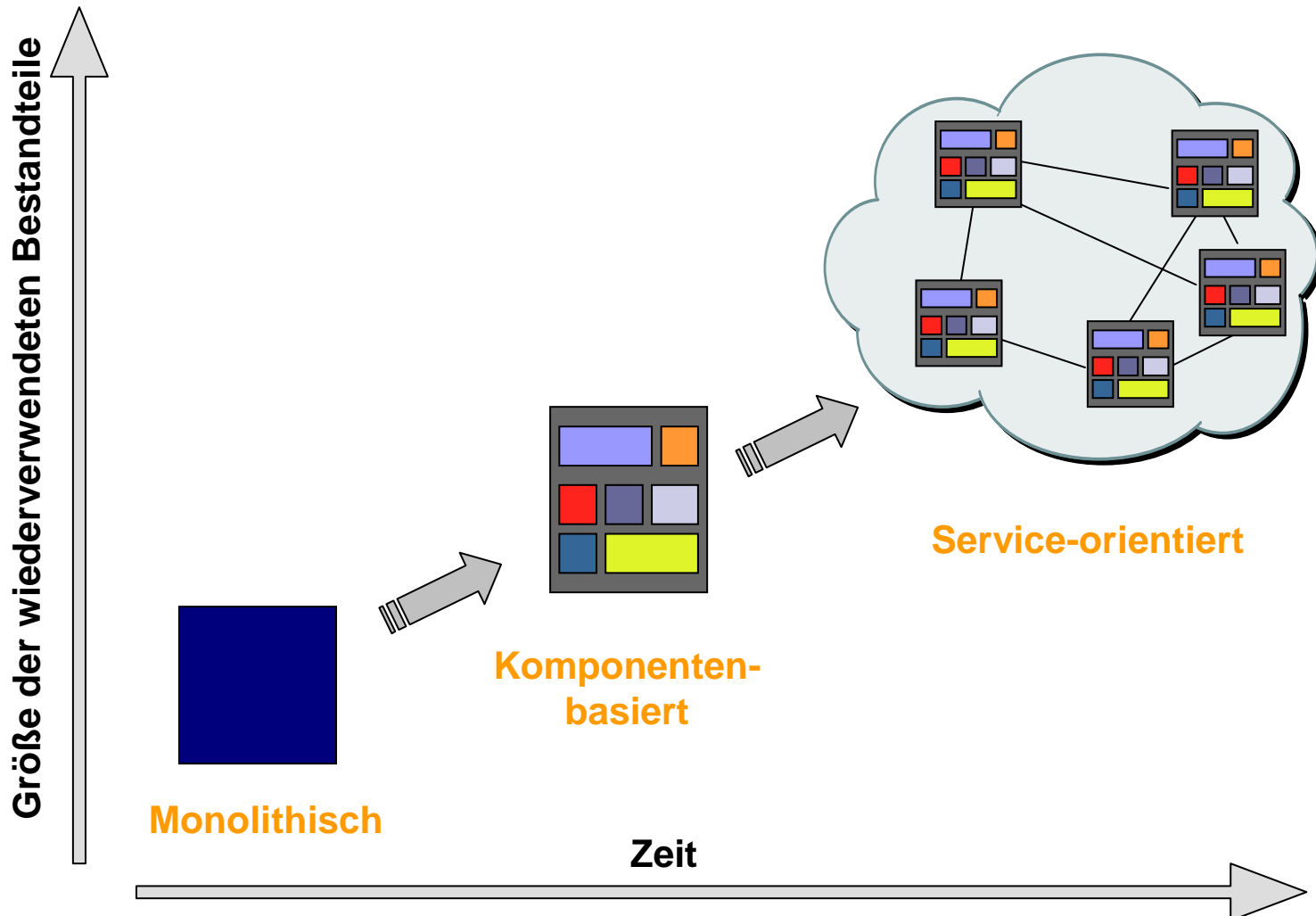


# Heutige Herausforderungen

- Web-Integration von Anwendungen und Diensten
  - "Fit for e-business"?
- "Just-in-time" Integration
- Zunehmende Bedeutung nicht-funktionaler Eigenschaften
  - Verfügbarkeit
  - Ressourcen-Beschränkungen
  - Time-to-market
  - Skalierbarkeit
- Kontextsensitivität
- Zugriff auf Basisdienste
  - Single Sign-On, Persistenz, Kalender, ...
- Abbildung von Geschäftsprozessen



# Evolution der Software-Systeme



# Ziele der Service-Orientierung



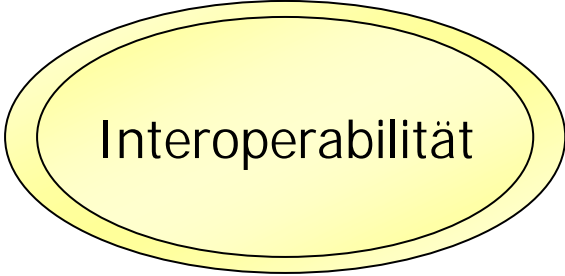
Lose Kopplung  
der Komponenten



Dezentralisierte  
Informationen

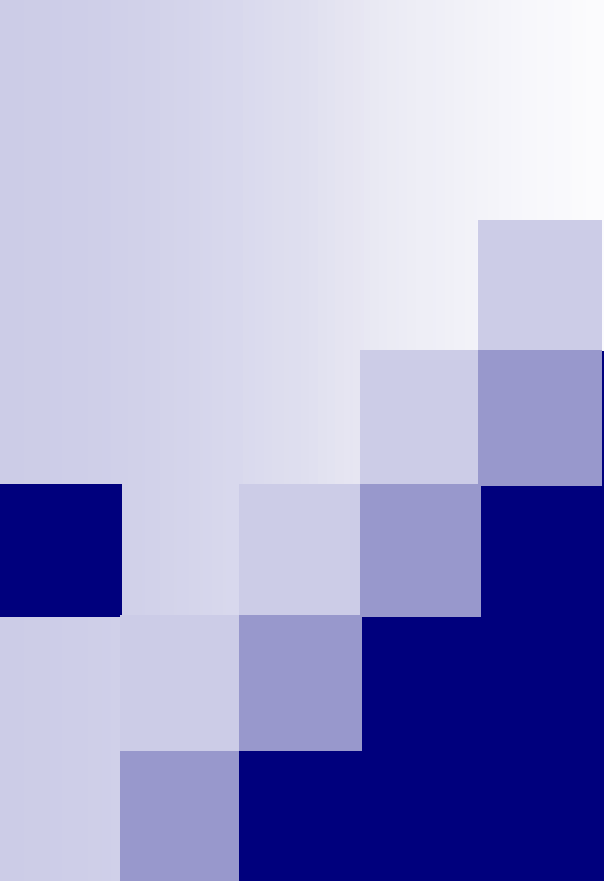


Geräteunabhän-  
gige Plattform



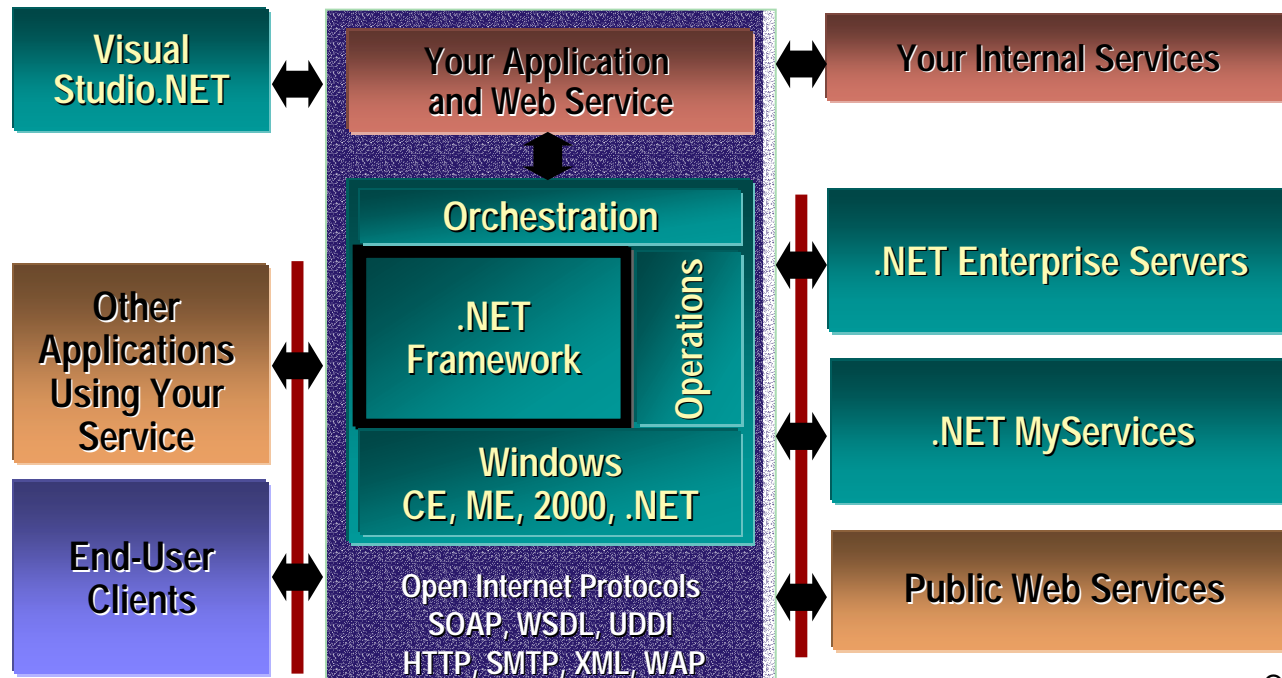
Interoperabilität

Zwischen Code und Daten als auch zwischen  
Legacy-Code und neuen Anwendungen



# Vergleich der Visionen: Sun ONE vs. Microsoft .NET

# Microsoft .NET Plattform



Quelle: Microsoft

- Vollständiges Framework
- Integriert Betriebssystem, Server-Produkte, Anwendungen und Dienste
- Setzt auf Standards (XML), zielt auch auf Kleingeräte
- Plattformabhängig, aber weitgehend programmiersprachenunabhängig
- Für Windows- und Web-Clients, für Web-Architekturen und komponentenorientierte Systeme geeignet

# GXA – Global XML Web Services Architecture

**Inter Application Protocols**

**Reliable Messaging**

**Eventing**

**Transactions**

**Directory**

**Inspection**

**Description**

**Building Block Modules**

**Referral**

**Security**

...

**Routing**

**License**

...

**SOAP**

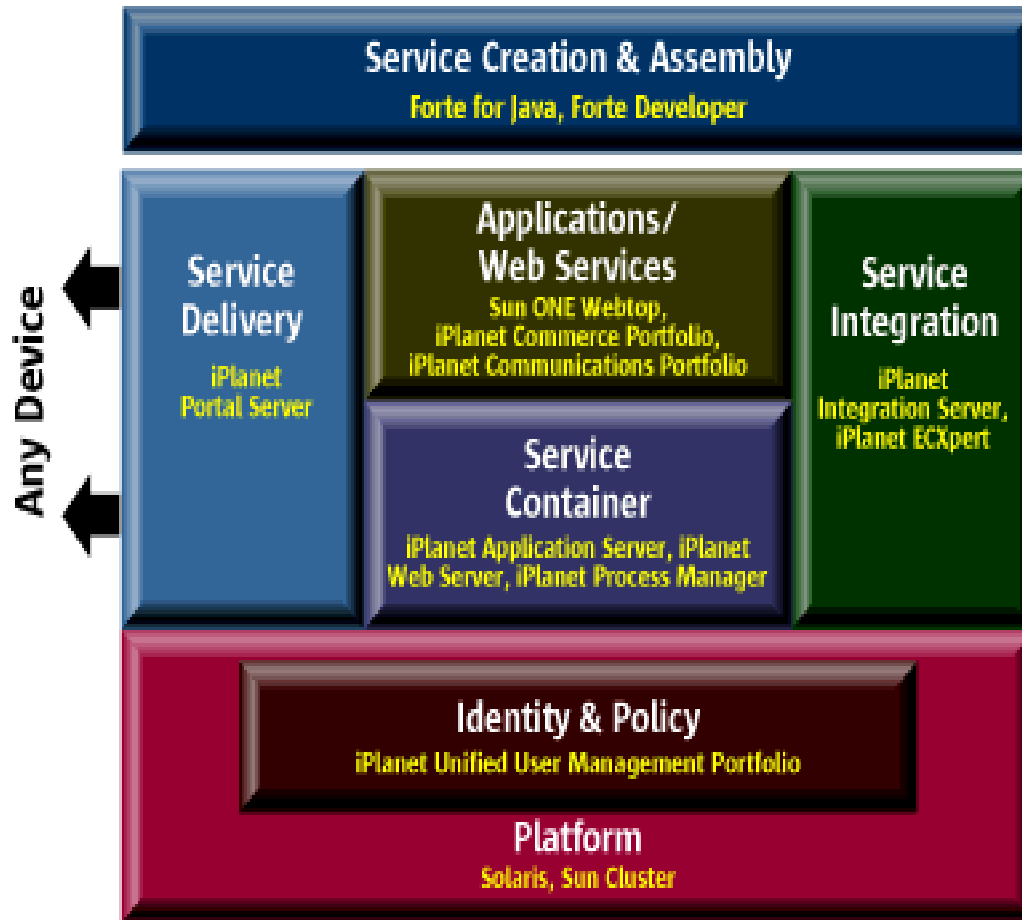
**Internet**

**TCP/IP**

**XML**

**HTTP/SMTP**

# Sun Open Network Environment (SunONE)



(c) Sun 2001

- Einzelkomponenten sind ausgereift, ergeben jedoch noch kein ausgereiftes Framework, dessen Komponenten kooperieren
- Sun ONE erscheint offener gegenüber anderen Betriebssystemen als .NET
- Prinzipiell können SunONE und .NET auf der Protokollschicht kooperieren (über XML, UDDI, WSDL)
- Vorwiegend für E-Business-Anwendungen geeignet

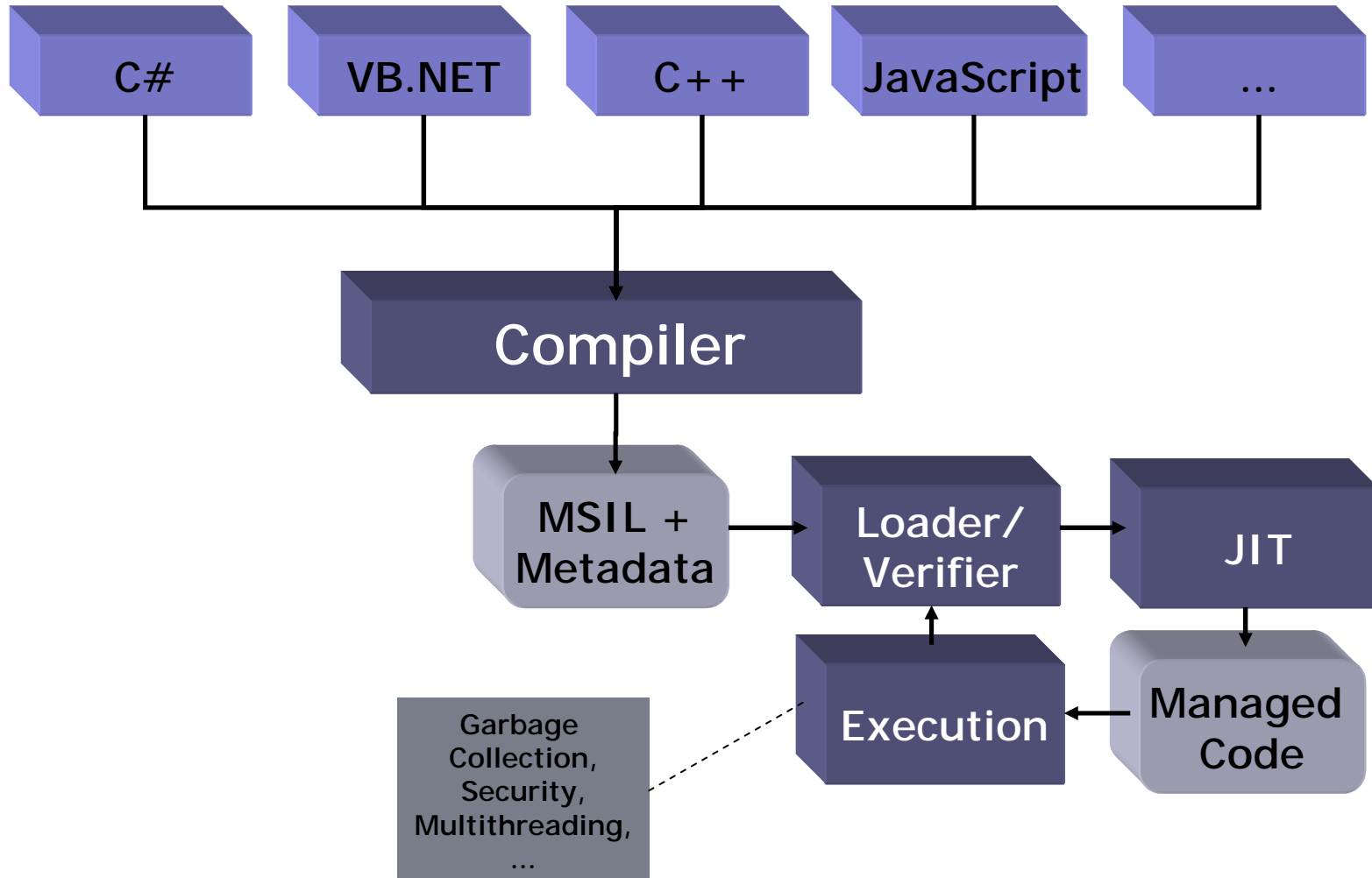


# Technologische Gegenüberstellung

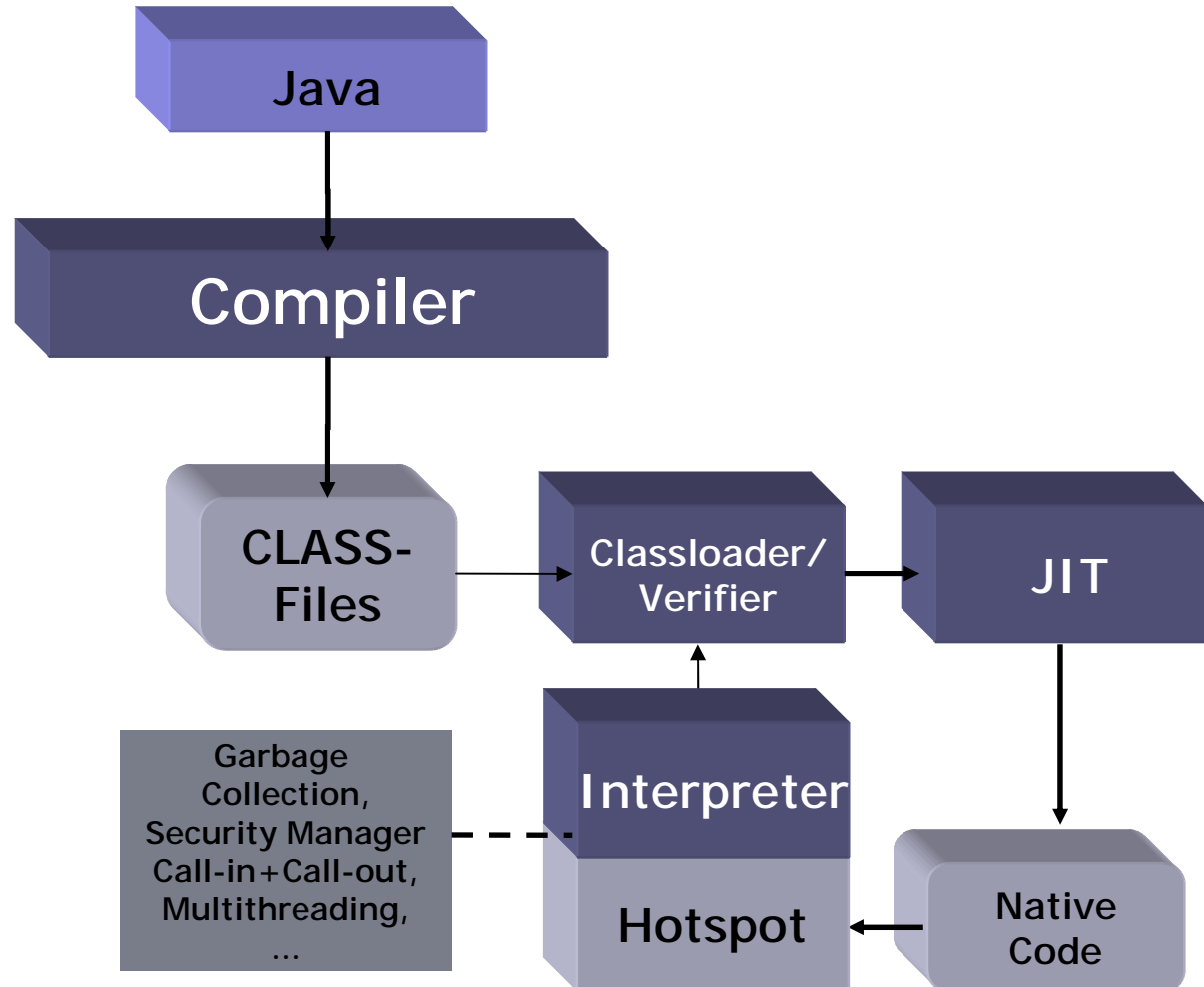
# Zu untersuchende Aspekte

- Laufzeitsystem
- Objektmodell
- Basisklassen
- Komponentenmodell
- Datenbankzugriff
- Verteilte Kommunikation
- Web Services-Architekturen

# .NET Laufzeitsystem



# Java Laufzeitsystem



# Vergleich: Laufzeitsysteme

## ■ Gemeinsamkeiten

- Grundkonzepte sind ähnlich
- .NET-Designer haben sich an Java orientiert

## ■ Unterschiede

- Java prinzipiell auf Interpretation ausgelegt
- Java erlaubt das Einbinden eigener Class-Loader und Security-Manager
- .NET CLR legt die Mächtigkeit der Programmiersprachen fest, erlaubt auch prozedurale Sprachen

# Objektmodell in .NET

- Unterscheidung zwischen Werttypen und Referenztypen:
  - Werttypen: einfachere Variablen, auf dem Stack verwaltet
  - Referenztypen: i.a. Objekte und Felder, auf dem Heap verwaltet
- In C# sind alle Variablen Objekte
  - Java unterscheidet zwischen einfachen Typen und Klassen
- Ein Werttyp kann durch "Boxing" zu einem Referenztyp werden (umgekehrt durch "Unboxing")
- Referenzen auf Funktionen und Methoden haben spezielle Typen (events und delegates)

# Objektmodell in Java

- Java kennt primitive Typen und Klassen
- Für die Umwandlung von primitiven Typen in Klassen sind Kapselungen nötig
- Statt Funktionszeiger Rückruffschnittstellen nach dem Publisher/Subscriber-Muster

# Vergleich: Objektmodell - Gemeinsamkeiten

- Schnittstellen sind rein abstrakte Klassen
- Nur einfache Implementierungsvererbung, aber mehrfache Schnittstellenvererbung erlaubt
- Namensräume (package in Java, verschachtelte Namespaces in .NET)
- Zugriffsbeschränkungen (public, ...)

# Vergleich: Objektmodell - Unterschiede

.NET	Java
Alle Datentypen sind Klassen	Unterscheidung zwischen primitiven Typen und Klassen
Referenzen auf Funktionen und Methoden haben spezielle Typen (events und delegates)	Rückruffschnittstellen nach dem Publisher/Subscriber-Muster
Zusätzliche Sprachelemente (sealed, enum, struct, etc.)	
Methoden müssen als virtual, override etc. gekennzeichnet werden	Alle Methoden virtuell

# Wichtige Basisklassen

	.NET	Java
<b>GUI</b>	Windows.Forms Web.Forms	SWING, AWT, (SWT)
<b>Kommunikation</b>	System.Net: Connection, HttpWebRequest, ...	Java.net: Sockets, URL, ...
<b>Container</b>	System.Collections: ArrayList, BitArray, Maps, Queue, List, Stack	java.util: Lists, Maps, Sets, Trees, Vectors

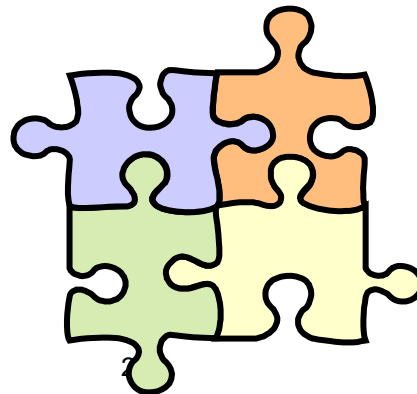
# Vergleich: Komponentenmodelle

## ■ Gemeinsamkeiten

- Modelle für Komponenten auf Client- und Serverseite vorhanden (Assemblies bzw. JavaBeans/EJB)
- Interoperabilität mit existierenden Komponenten möglich (über COM+ in .NET, über CORBA in Java)

## ■ Unterschiede

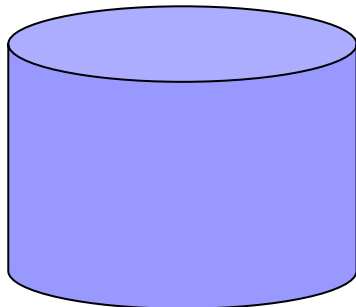
- Modell der JavaBeans und EJBs reifer, Container leistungsfähiger
- Explizites Versionierungskonzept in .NET (gleichzeitiger Zugriff auf verschiedene Versionen)



# Vergleich: Datenbankzugriff

## ■ Gemeinsamkeiten

- Abstrakte APIs
- Entkopplung zwischen Datenquelle und Nutzer



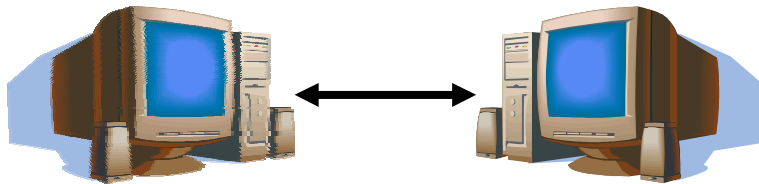
## ■ Unterschiede

- ADO.NET auf XML ausgerichtet, JDBC eher binär
- ADO.NET bevorzugt asynchronen Zugriff, JDBC eher synchronen

# Vergleich: Verteilte Kommunikation

## ■ Gemeinsamkeiten

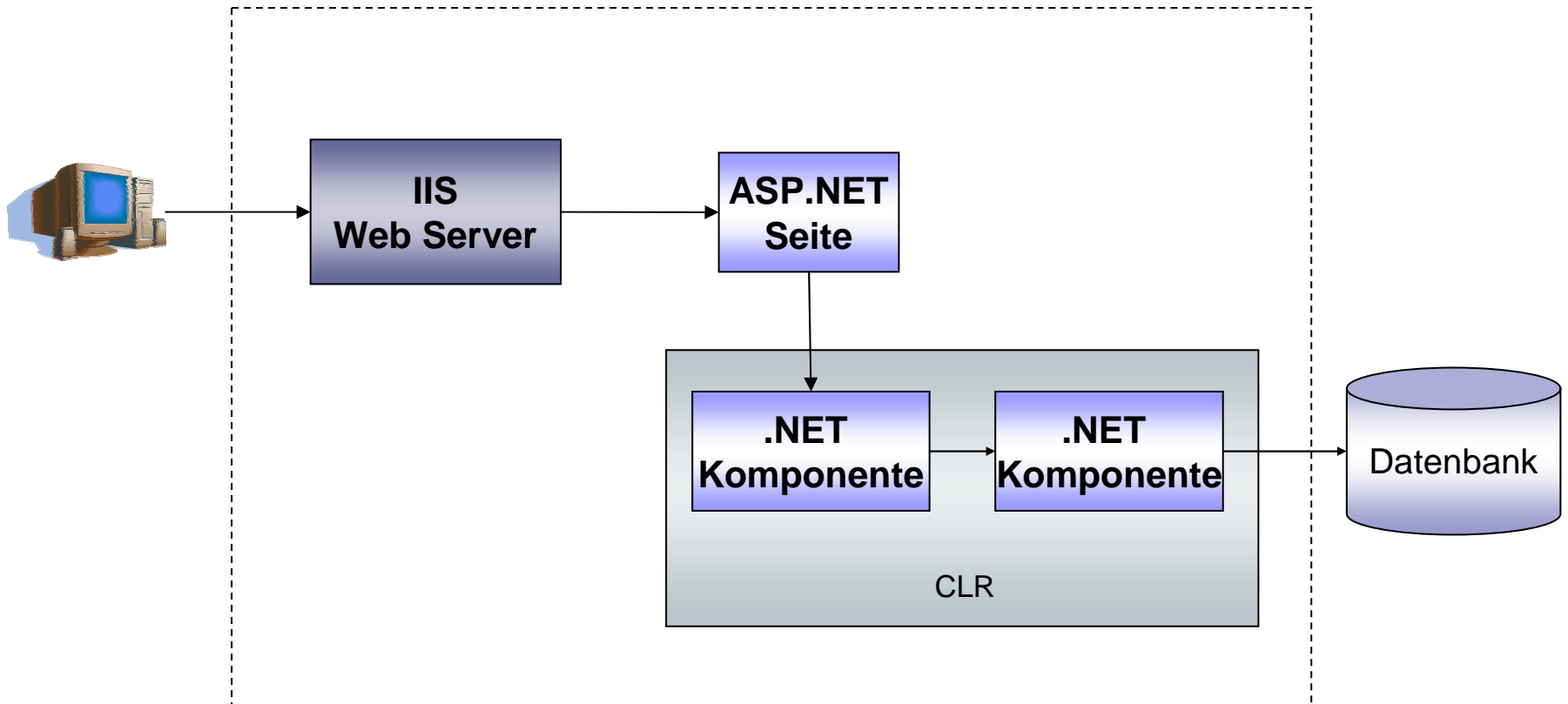
- Ähnliche Architektur
- Einfach nutzbar



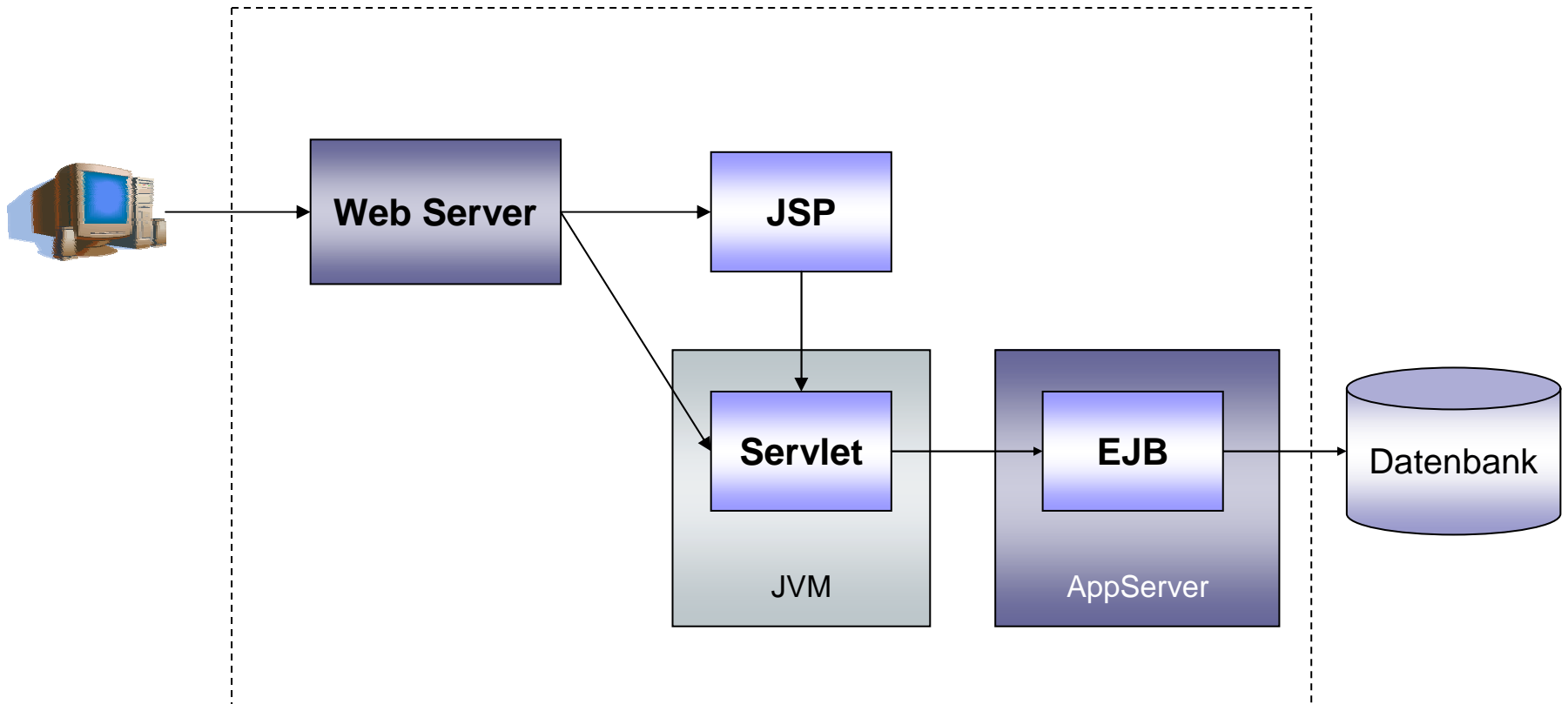
## ■ Unterschiede

- .NET-Remoting bietet offene Architektur
  - Transportkanal und Serialisierung unabhängig voneinander wählbar
  - unterstützt SOAP standardmäßig
- Java RMI hat eigenen Naming-Service und Interoperabilität mit CORBA über IIOP

# Web-Applikationen mit .NET



# Web-Applikationen mit Java



# Web Service-Architekturen mit .NET



- Web Services sind Kernbestandteil von .NET
- Verschiedene Zugriffswege
  - ASP.NET Web Services über .aspm-Seiten
  - .NET Remoting Web Services
- Sehr einfache Entwicklung
  - Umfangreiche Web Service-Unterstützung in Visual Studio .NET
  - Kommandozeilen-Tools bereits im kostenlosen .NET Framework
- .NET Web Services sind gegenwärtig an den Microsoft Internet Information Server und Windows-Plattform gebunden

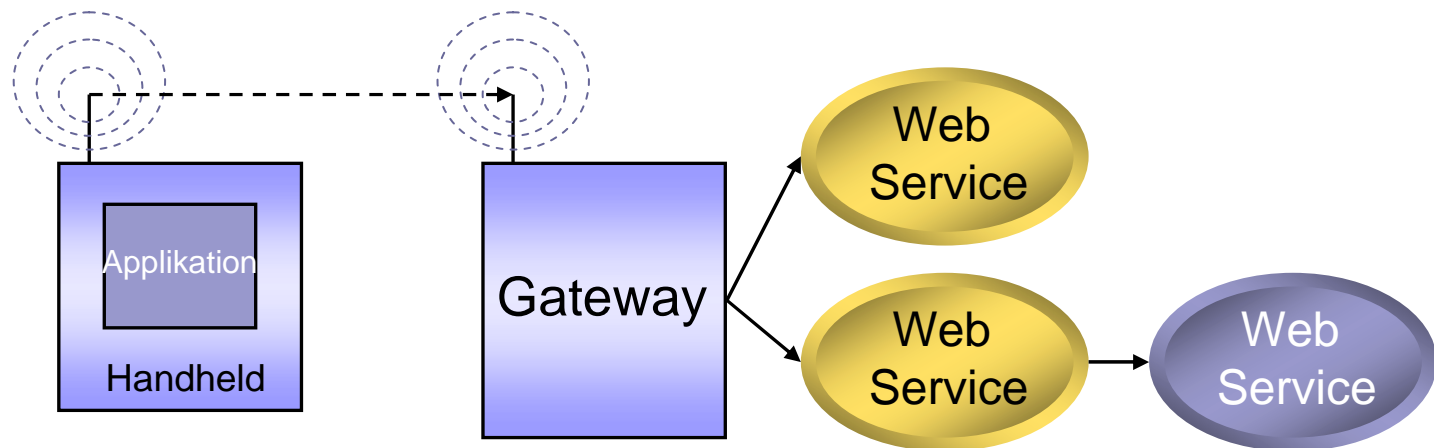
# Web Service-Architekturen mit Java



- Sun stellt ein Web Service Developer Pack bereit
  - Ist ebXML-konform
- Bislang wurden verschiedene Toolkits verwendet
  - Z.B. Apache SOAP, IBM Web Services Toolkit, GLUE
- Web Services oft als Servlets oder EJBs implementiert
- Keine so durchgängige Tool-Unterstützung
- Interoperabilität weitgehend gewährleistet
- Bei GXA enge Zusammenarbeit zwischen Microsoft (.NET) und IBM (Java)

# Nutzung von mobilen Web Services

- Anwendungen werden auf dem Endgerät installiert
  - Evtl. mittels automatischem Deployment
- Applikationen enthalten nur Koordinationslogik, um eine schnelle Reaktion auf Benutzerinteraktionen zu gewährleisten
- Business-Logik befindet sich in Form von Web Services im Netz verteilt



# Mobile Web Services in .NET

- .NET bietet "Smart Device Extensions" für Windows-CE-Geräte
  - Enthält Compact Framework (verschlanktes .NET Framework)
- Bindet Visual Studio .NET ein
  - Lässt "managed" Applikationen direkt ausführen
  - Debuggen mit Visual Studio .NET
  - Bietet Pocket-PC-Emulator
- Arbeitet komplementär zum Betriebssystem
  - Betriebssystem übernimmt Scheduling, Anzeige der Benutzeroberfläche, Eingabeverarbeitung, Ressourcenverwaltung etc.
- Unterstützt über ADO.NET und SQL Server Datenhaltung mit unterbrochener Verbindung auf dem Gerät
- SDE unterstützt Web Services nur als Client!
  - Auch mittelfristig keine Unterstützung mobiler Geräte als Web Service Provider vorgesehen!



# Mobile Web Services in Java

- Verschiedene Toolkits für Personal Java und J2ME verfügbar, z.B.
  - kSOAP (<http://www.ksoap.org>)
    - Open-Source-Version für die J2ME, z.B. auf Mobiltelefonen
  - GLUE (<http://www.themindelectric.com>)
    - Läuft auf Basis von JDK 1.1.x
    - Für PocketPCs geeignet
    - Gute Toolunterstützung
  - eSOAP (<http://www.embedding.net/esoap/>)
    - Embedded SOAP, kleiner Footprint, speziell für Embedded-Anwendungen
  - Wingfoot SOAP (<http://www.wingfoot.com>)
    - Java-Implementierung mit kleinem Footprint (35 kB) für CLCD/MIDP und CDC/Personal Java
    - Schnelle Verarbeitung
- Basieren entweder auf Insignias *Jeode VM* oder Suns *KVM*



# Vergleich: Web Services – Gemeinsamkeiten

- Ähnliche Konzepte
- Gemeinsame Standards bei Protokollen
  - heute: SOAP, WSDL, UDDI
  - Gemeinsame Arbeit an Weiterentwicklung (-> GXA)
  - Dadurch Interoperabilität weitgehend gewährleistet
- Effiziente Verarbeitung durch Vorkompilierung der Services

# Vergleich: Web Services – Unterschiede

- .NET Web Services können in allen .NET-Sprachen geschrieben werden
  - Aber: Microsoft-WS sind nicht ebXML-konform!
- Java Web Services sind nicht an einen Web Server oder einen Servlet- bzw. EJB-Container gebunden
  - Viele Open Source-Implementierungen verfügbar
  - Aber: Manche Toolkits arbeiten nur mit (teurem) EJB-Container
- Toolunterstützung in .NET durchgängiger, dadurch Entwicklung einfacher
  - Toolkits für Java stark unterschiedliche Konzepte
- Web Services-Unterstützung für mobile Geräte in Java besser



# Nicht-technischer Vergleich der Plattformen

# Herstellerabhängigkeit

- .NET ist derzeit klar an Microsoft gebunden
  - Aber: Projekte wie Mono versuchen, .NET auch auf andere Plattformen zu bringen
  - C# und Teile von .NET sind über ECMA standardisiert
- J2EE-AppServer folgen dem Standard
  - Aber: Hersteller haben Lücken im Standard unterschiedlich interpretiert
  - Folge: EJBs nicht ohne Weiteres von einem AppServer zu einem anderen übertragbar

# Technische Reife

- .NET kam 3 Jahre nach J2EE, 10 Jahre nach Java
  - .NET hat aus den Problemen von Java/J2EE gelernt
- J2EE-Produkte haben die Anfangsprobleme mit EJBs überwunden, nicht die mit Web Services!
- Aufbau einer komplexen J2EE-Site auch heute noch eine Herausforderung
  - Es gibt bereits Sites, die .NET produktiv einsetzen (MS, NASDAQ, Dell)

# Interoperabilität

- Lange Zeit war Interoperabilität für Sun nur Inter-AppServer und Inter-ORB
  - Selbst diese funktionieren heute nicht immer!
- Kein Standard-API für Web Services in Java
  - Viele Einzellösungen (Toolkits), die alle kreuzweise auf Interoperabilität getestet werden müssen
- .NET-Java Interoperabilität wird von Microsoft und IBM bestimmt
  - Gute Zusammenarbeit nach Anfangsschwierigkeiten

# Skalierbarkeit

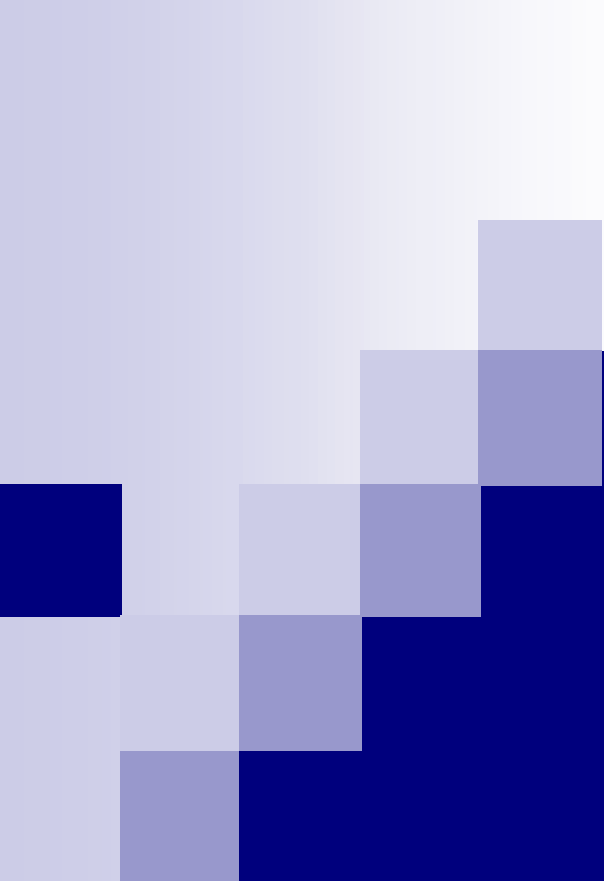
- Für Unternehmensanwendungen sind AppServer für nicht-triviale Web Services notwendig!
  - Aber: keine offiziellen Transaktions-Benchmarks von J2EE-Plattformen verfügbar
- Preise für J2EE-Lösungen höher
  - J2EE-Produkte teurer als .NET Framework
  - Unix-Hardware teurer als PC-Hardware
- Skalierbarkeit mit beiden Lösungen erreichbar
  - J2EE mit höheren Kosten
  - Aber: TCO-Vergleiche heute nur Vermutungen

# Programmiersprachen

- Java und C# sind sehr ähnlich
  - Java-Programmierer können schnell auf C# umsteigen
- .NET kann auch in C++ (unschön), Visual Basic und anderen Sprachen programmiert werden
  - Nutzt ggf. vorhandene Kenntnisse der Entwickler
- Beide Plattformen sind objekt-orientiert und komponentenbasiert
  - Beträchtlicher Aufwand, um COBOL/PL1/C/...-Entwickler darauf zu schulen

# Portabilität

- Nach ECMA-Standardisierung könnte .NET auch auf nicht-MS-Plattformen kommen
- J2EE-AppServer meist für mehrere Betriebssysteme verfügbar
  - Möglichkeit des Wechsels klarer Vorteil!
  - Aber: Wer braucht diese Möglichkeit wirklich?
- Echte Portabilität über Herstellergrenzen existiert kaum!



# Fazit und Empfehlungen

# Kernempfehlung

**Schützen Sie Ihre Investitionen  
in Know-How (Entwickler,  
Betrieb) und Applikationen!**

# Empfehlungen

- Jeder sollte seine Plattform beibehalten
  - Wer in Windows investiert hat, sollte nach .NET migrieren
  - Wer Java nutzt und/oder Unix, sollte eine Java-Plattform verwenden
- .NET bietet alles aus einer Hand
  - Inkl. integrierte Server-Produkte (SQL Server, Commerce Server, Host Integration Server)
- Java wird von verschiedenen Herstellern angeboten

# Technisches Fazit

- .NET stammt nur von Microsoft, Java von Sun und JCP-Partnern
- .NET ist eine Produktfamilie, J2xE eine Spezifikation
- Web Services sind mit beiden gut zu verwirklichen, die Probleme liegen woanders!
  - Standardisierung von XML-Formaten für Daten und Schnittstellen
  - UDDI als universelle Dienst-Suchmaschine?
  - Sicherheit und Transaktionsfähigkeit
  - ...

# Danke für Ihre Aufmerksamkeit

Fragen oder Anmerkungen?



[www.drwieland.de](http://www.drwieland.de)