Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
InIT Institut für angewandte
Informationstechnologie

Bachelor of Science (BSc) in Informatik

Modul Software-Entwicklung 1 (SWEN1)

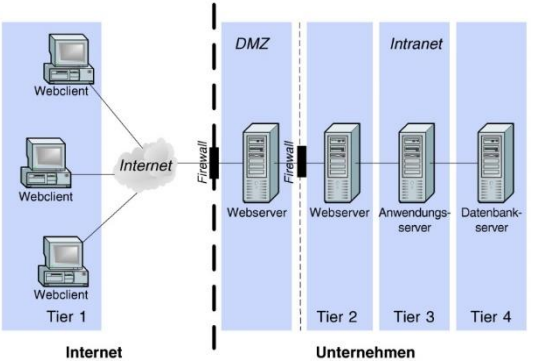
V1 – Verteilte Systeme

SWEN1/PM3 Team:
R. Ferri (feit), D. Liebhart (lieh), K. Bleisch (bles), G. Wyder (wydg)

Ausgabe: HS24

Um was geht es?

- Was sind verteilte Systeme?
- Wie ist der prinzipielle Aufbau eines Client-Server-Systems?
- Welche Phänomene und Probleme ergeben sich bei verteilten Systemen?
- Welche Aspekte sind zu berücksichtigen beim Design und der Implementierung eines Client-Server-Systems?
- Was sind gängige Technologien (Middleware) zur Entwicklung von verteilten Systemen?



Lernziele VT 01 – Verteilte Systeme

Sie sind in der Lage,

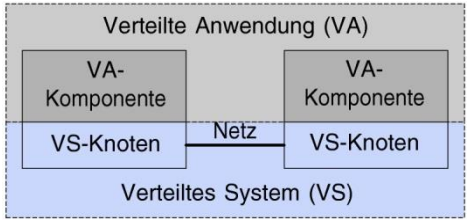
- zu erläutern, was ein verteiltes System ist und warum verteilte Systeme eingesetzt werden,
- die fundamentalen Konzepte eines verteilten Systems wie Architekturstil, Kommunikationsverfahren, Fehlertoleranz und Fehlersemantik zu erläutern,
- wichtige Design- und Implementierungsaspekte von Client-Server-Systemen zu diskutieren,
- für einen Entwurf eines verteilten Systems gängige Architektur und Design Patterns zu benutzen,
- gängige Technologien (Middleware) zur Entwicklung von verteilten betrieblichen Informationssystemen und Internet-basierten Systemen einzuordnen.

Agenda

1. Einführung in verteilte Systeme
2. Design- und Implementierungskonzepte von Client-Server-Systemen
3. Middleware für verteilte Systeme
4. Wrap-up und Ausblick

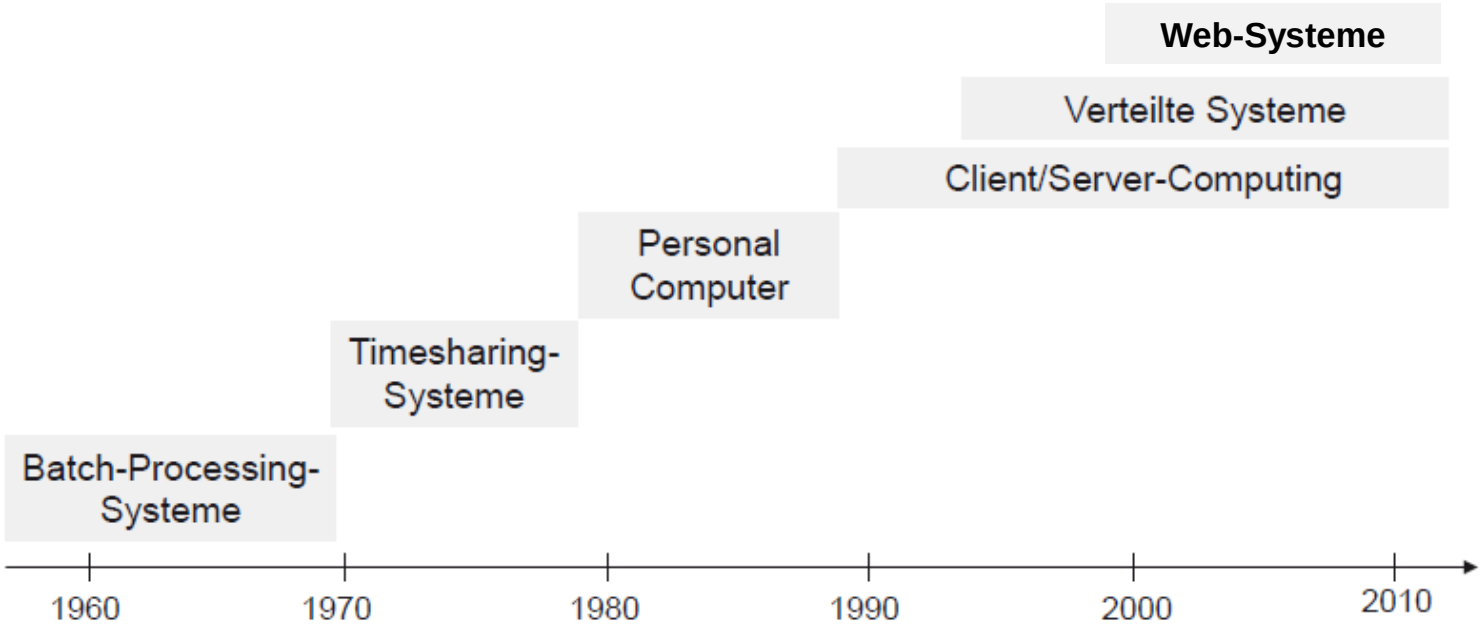
Was ist ein verteiltes System?

- Verteiltes System
  - Basiert auf einer Menge voneinander unabhängiger Rechnersysteme (Knoten) und Softwarebausteinen (Komponenten).
  - Erscheinen dem Benutzer wie ein einzelnes, kohärentes System bzw. Anwendungssystem.



- Verteilte Anwendung
  - Anwendung, die auf einem verteilten System läuft.
  - Jeder Softwarebaustein kann auf einem eigenen Rechner liegen.
  - Es können aber auch mehrere Softwarebausteine auf dem gleichen Rechner installiert sein.

Historische Entwicklung

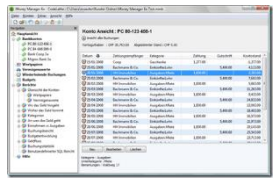


Die folgenden Faktoren haben die Entwicklung wesentlich beeinflusst:

- Leistungsexplosion in der Mikroprozessortechnik,
- schnelle lokale Netzwerke (LAN)
- Verbindung mehrerer physischer Netze zu einem einheitlichen Kommunikationssystem (WAN) und das Anbieten eines Universaldienstes für heterogene Netzwerke, dem Internet

Typische verteilte Systeme heute sind…

- Informationssysteme
- Mobile Systeme
- Eingebettete Systeme
- Cloudbasierte Systeme
- Hochleistungsrechnersysteme

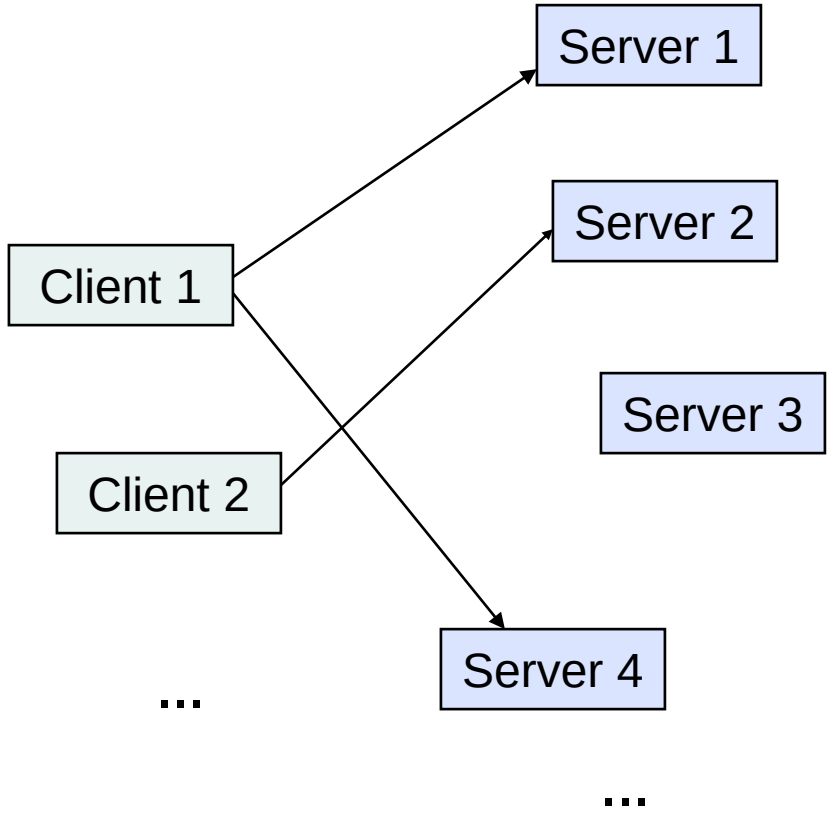


Was sind verteilte Informationssysteme?

- Verteilte Informationssysteme sind verteilte Systeme mit besonderen Merkmalen.
- Typische Merkmale:
  - Oft sehr gross
  - Sehr datenorientiert: Datenbanken im Zentrum der Anwendung
  - Extrem interaktiv: GUI, aber auch Batch
  - Sehr nebenläufig: Grosse Anzahl an parallel arbeitenden Benutzern
  - Oft hohe Konsistenzanforderungen

Warum setzt man auf verteilte Systeme?

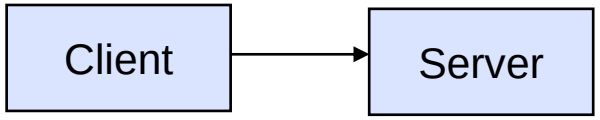
- Vorteile:
  - Gemeinsamer Ressourcenzugriff
  - Lastverteilung
  - Ausfallsicherheit, Verfügbarkeit
  - Skalierbarkeit
  - Flexibilität
  - Verteilungstransparenz (Ort, Fehler, Persistenz, ...)
- Nachteile:
  - Komplexität durch Verteilung, Netzinfrastruktur
  - Sicherheitsrisiken



Architekturmodelle verteilter Systeme

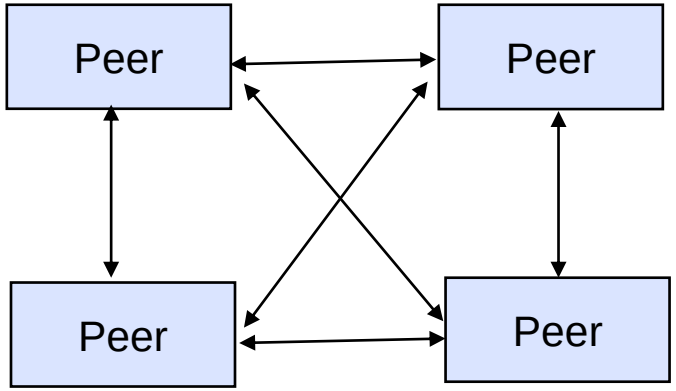
- Ein Architekturmodell beschreibt die Rollen der Komponenten innerhalb einer verteilten Anwendung sowie die Beziehungen zwischen ihnen.
- Heute finden vor allem folgende Architekturmodelle ihren Einsatz:

Client/Server



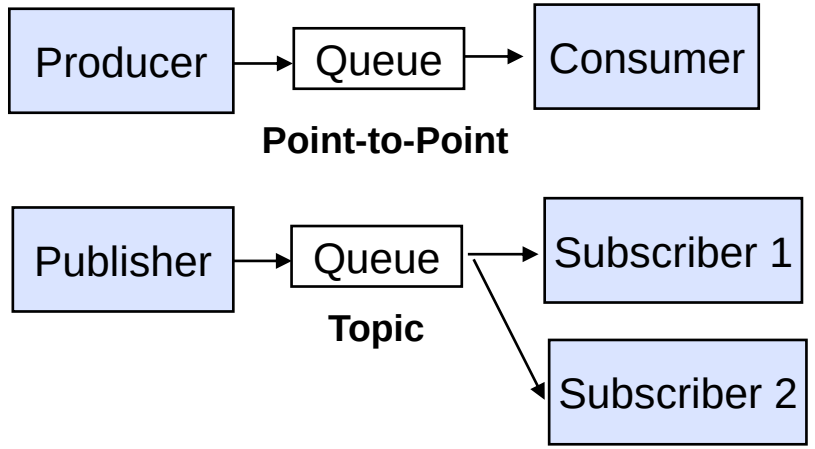
Kurzlebiger Client-Prozess, der mit einem langlebigen Server-Prozess kommuniziert (z.B. Web-Applikation)

Peer-to-Peer



Gleichberechtigte Peer-Prozesse, die nur bei Bedarf Informationen austauschen (z.B. Blockchain)

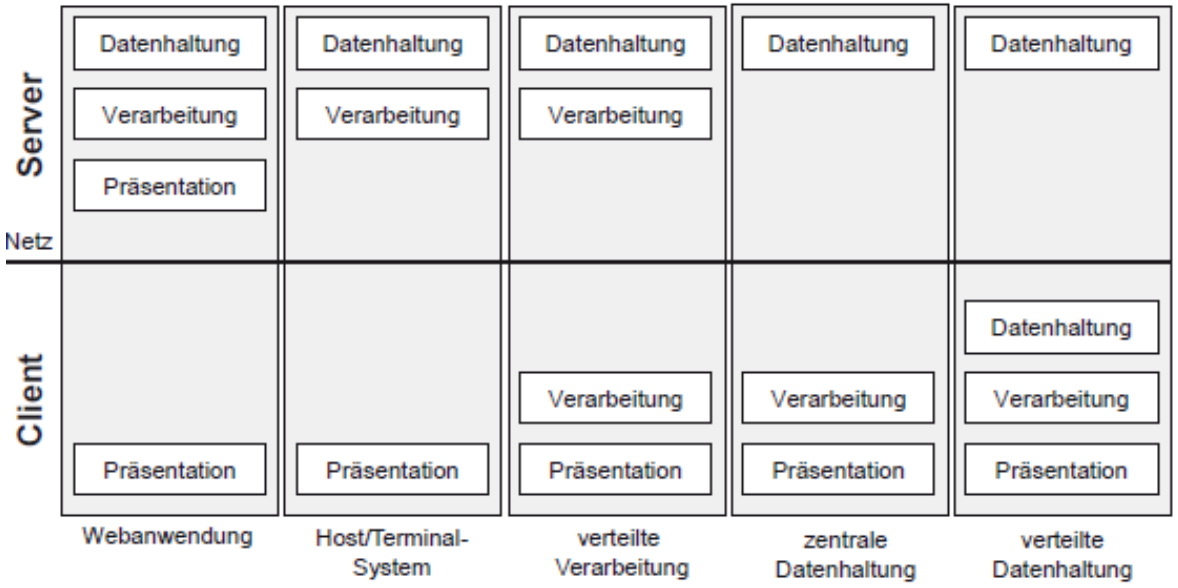
Event Systems



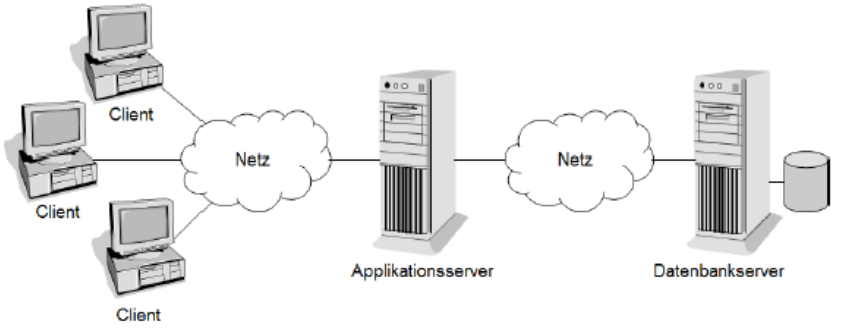
Event-Sources-Prozesse und Event-Sinks-Prozesse, die asynchron Informationen austauschen (z.B. E-Mail)

Mehrstufige Architekturen (Multi-Tier-Architekturen)

- Multi-Tier-Architekturen sind eine Ergänzung zum Client-Server-Architekturmodell und beschreiben Modelle zur Verteilung einer Anwendung auf den Rechnern (engl. tiers) des verteilten Systems.
- Für die Arbeitsteilung zwischen Client und Server existieren verschiedene Alternativen, je nachdem, wo die Schichten (Layer) Präsentation, Verarbeitung (Domänenlogik) und Datenhaltung angesiedelt sind.



Beispiel: 3-Tier-Architektur

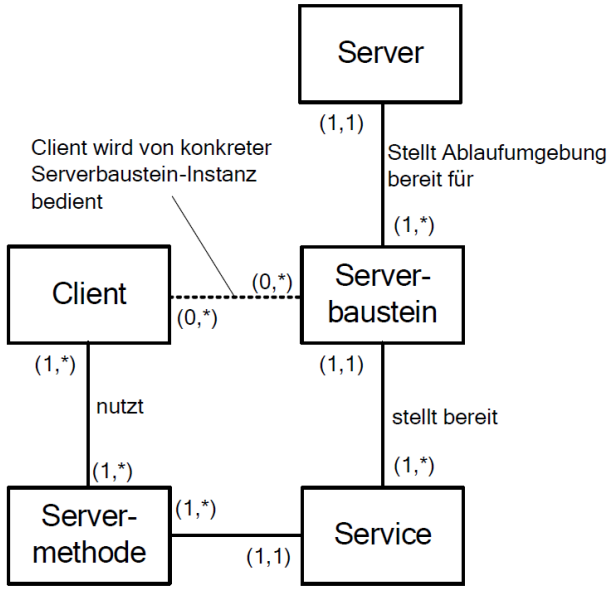


Agenda

1. Einführung in verteilte Systeme
2. **Design- und Implementierungskonzepte von Client-Server-Systemen**
3. Middleware für verteilte Systeme
4. Wrap-up und Ausblick

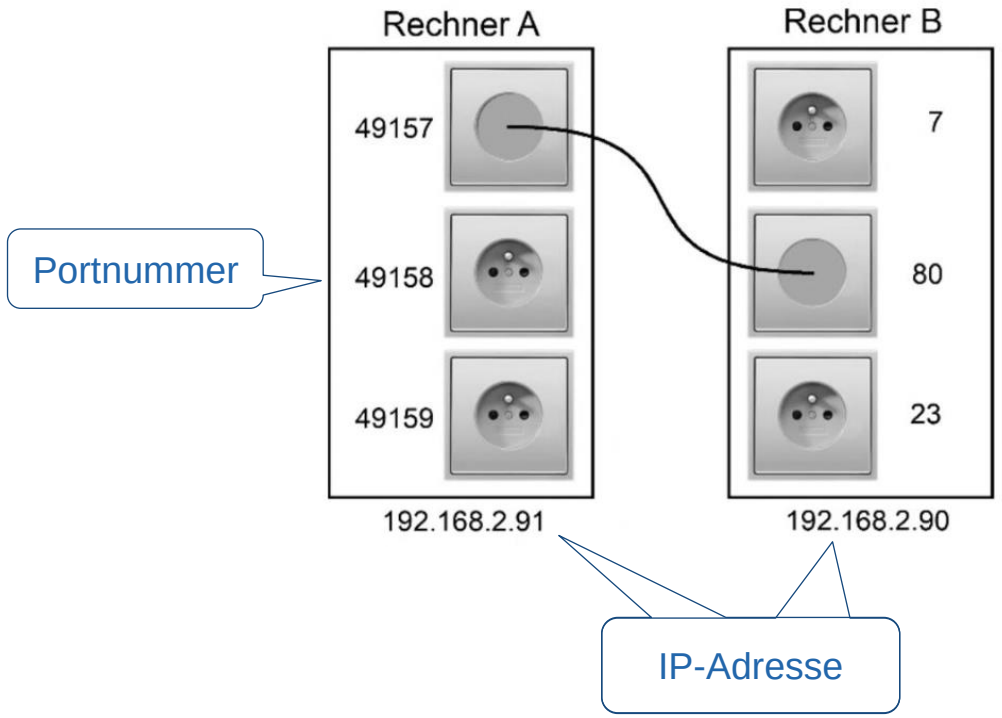
Terminologie bzw. Metamodell für die Diskussion von Client-Server-Systemen

- Ein Server stellt eine Ablaufumgebung für einen oder mehrere Serverbausteine bereit.
- Ein Applikationsserver ist auch ein Server, auf dem Serverbausteine ausgeführt werden, aber im engeren Sinne noch verschiedene Dienste den Serverbausteinen anbietet (z.B. Authentifizierung, Transaktionen etc.).
- Ein Serverbaustein ist ein Objekt, Modul oder Komponente (je nach verwendetem Programmiermodell), das zum Ablaufzeitpunkt instanziiert und bei Bedarf einem Client für die Abarbeitung einer Anforderung (eines Requests) zugeordnet wird.
- Ein Service oder Dienst wird durch einen Serverbaustein bereitgestellt und enthält eine oder mehrere Servermethoden oder Serverprozeduren.
- Eine Servermethode oder eine Serverprozedur ist Bestandteil eines Services, den ein Client durch das Senden eines entsprechenden Requests nutzen kann.



Kommunikation zwischen Client und Server (1/2)

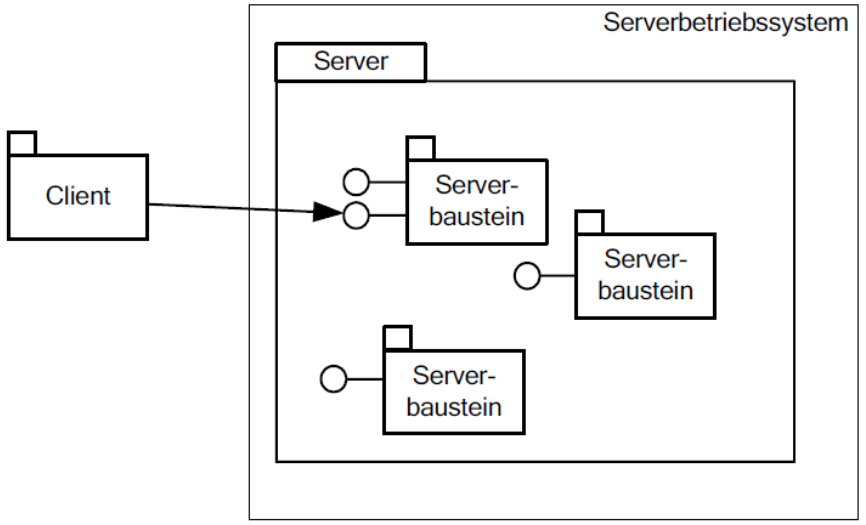
- Jeder Service ist über seine URL aufrufbar:
  - protokoll://<server>:<port>/<pfad_des_service>
- Kommunikation zwischen Client und Server
  - Über TCP oder UDP
  - Socket
    - Programmierschnittstelle zu Kommunikationskanal
    - IP-Socket-Adresse: IP-Adresse + Portnummer



Kommunikation zwischen Client und Server (2/2)

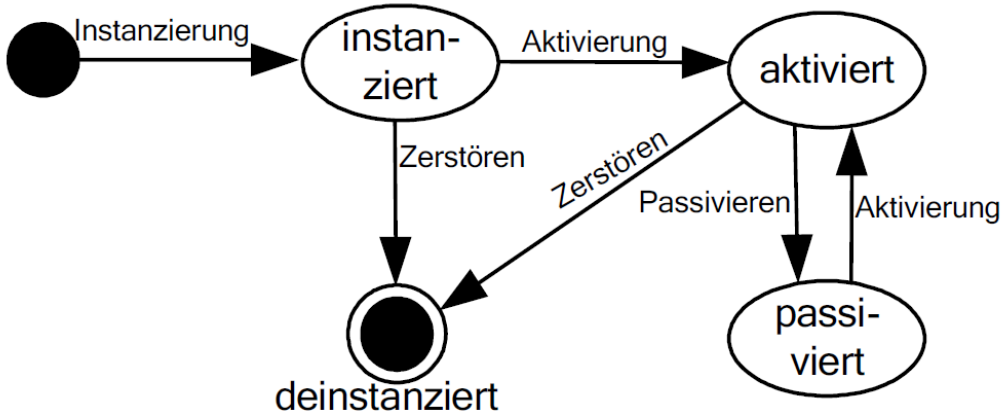
- Client sendet Request an Server
- Ein Server empfängt den Client-Request und leitet diesen zur Verarbeitung an den entsprechenden Service (des betreffenden Serverbausteins) weiter.
- Service bearbeitet Request und schickt Antwort (Response) zurück an den Client.

- Ein Server ist seinerseits in eine Ablaufumgebung (z.B. VM) innerhalb des Rechnerbetriebssystems eingebettet.
- Server und Serverbaustein müssen vor der Verwendung instanziiert werden.



Lebenszyklus von Serverbausteinen

- Ein Serverbaustein wird zur Laufzeit von einem Server instanziiert und durchläuft, je nach Bausteintyp und Implementierung verschiedene Zustände.
- Zustandsdiagramm für eine Serverbaustein-Instanz:

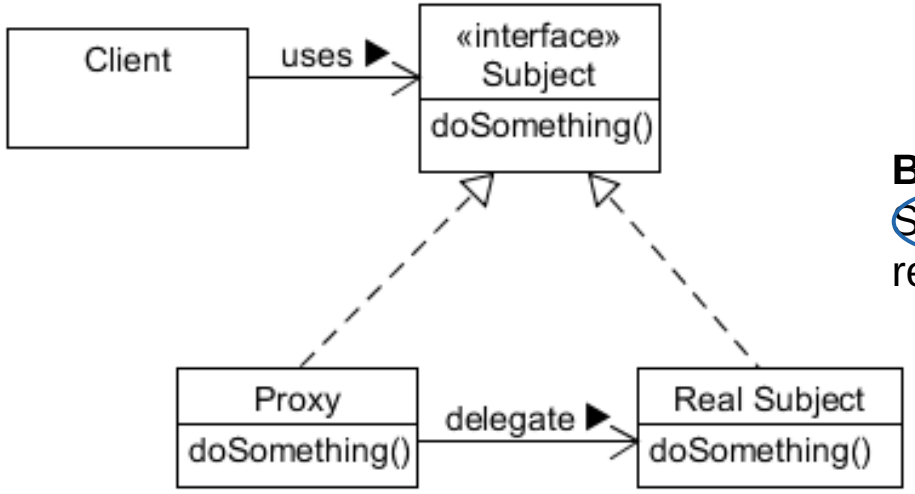


- Die Anzahl und Benennung der Zustände sind je nach Client-Server-Implementierung (Middleware) verschieden.

Verwendetes Design Pattern für den Zugriff auf Services in Serverbausteinen

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Grundlegendes Design Pattern für den Zugriff auf Serverbausteine ist das Remote Proxy.



Interface

Proxy:
Instanziierung in der Regel durch Dependency Injection (DI)

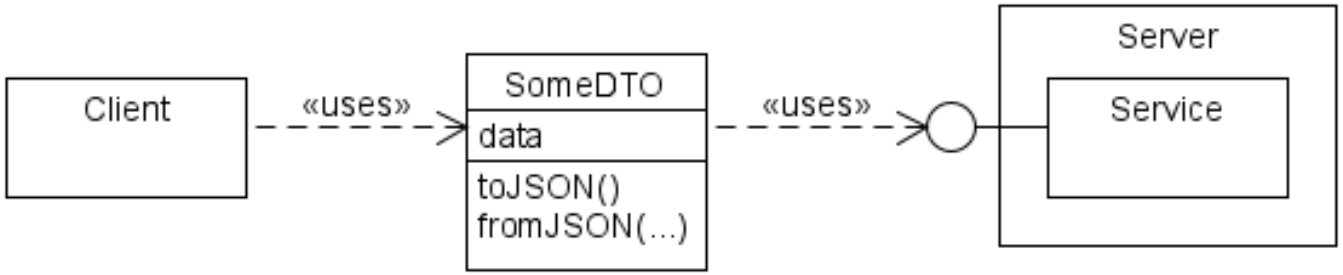
Beispiel:

```
Service service = new ServiceImpl();
result = service.doSomething();
```

Anmerkung: *In einer Client-Server-Implementierung heisst der (client- und serverseitige) Proxy auch Stub (von englisch stub, Stubben, Stummel, Stumpf). Ein server-seitig generierter Stub wird dabei Skeleton (engl. Skelett, Gerippe, Gerüst) genannt.*

Verwendetes Design Pattern für den Datenaustausch zwischen Client und Server

- Grundlegendes Design Pattern ist das Data Transfer Object (DTO). [3]



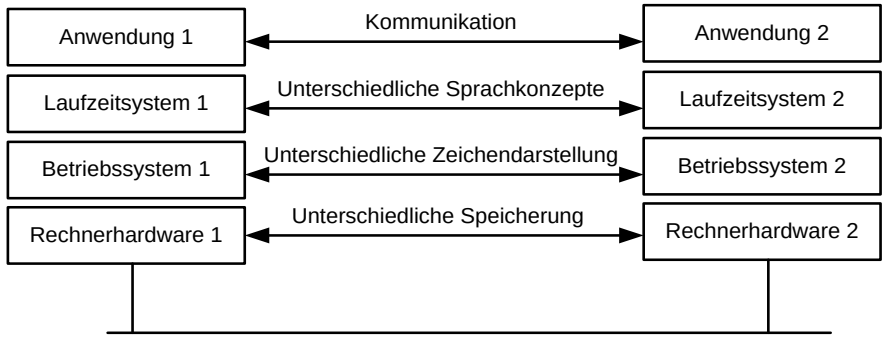
- Es bündelt mehrere Daten in einem Objekt, sodass sie durch einen einzigen Programmaufruf übertragen werden können.
- Der Zweck ist, mehrere zeitintensive Remotezugriffe durch einen einzigen zu ersetzen.
- Ein DTO ist in der Regel «immutable», d.h. enthält nur getter-Methoden.

Ausgewählte Implementierungsaspekte von Client-Server-Systemen

- Wir betrachten im Weiteren einige ausgewählte Aspekte:
  - Heterogenität
  - Serverarchitektur
  - Nebenläufigkeit im Server (Parallelität)
  - Serverseitige Service- bzw. Dienstschnittstellen
  - Fehlersituationen, Fehlerklassierung
  - Parameterübergabe zwischen Client und Server
  - Marshalling/Unmarshalling
  - Kommunikation
  - Zustandsverwaltung
  - Garbage Collection
  - Lastverteilung, Verfügbarkeit, Skalierbarkeit

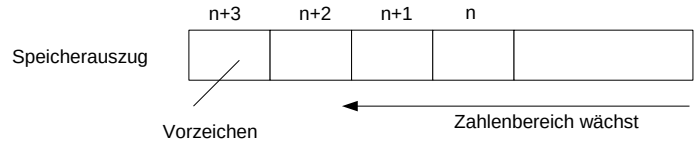
Heterogenität

- Mehrere Ebenen der Heterogenität
- Standardformate notwendig!

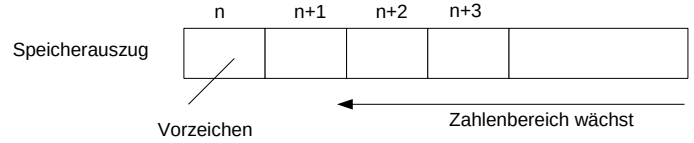


- **Rechnerhardware und Betriebssysteme**
  - Unterschiede bei der Speicherung der Daten
    - «Little Endian» versus «Big Endian»
  - Unterschiedliche Zeichensätze
    - ASCII - EBCDIC - Unicode

Darstellung: "little endian"

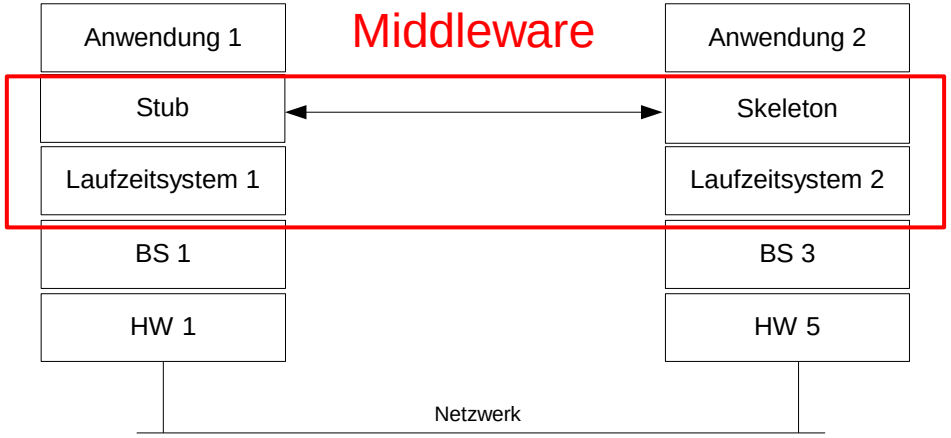


Darstellung: "big endian"



Überlegungen zur Überwindung von Heterogenität

- Was wir brauchen!
  - Einheitliche Transportsyntax (ASN.1, XDR, HTML, XML, JSON …) → Schicht 6 (ISO/OSI-Modell)
  - Middleware-Technologien bieten meist ähnliche Ansätze
  - Marshalling (Serialisierung) und Unmarshalling (Deserialisierung) der Nachrichten über generierten Code (Stubs und Skeletons)

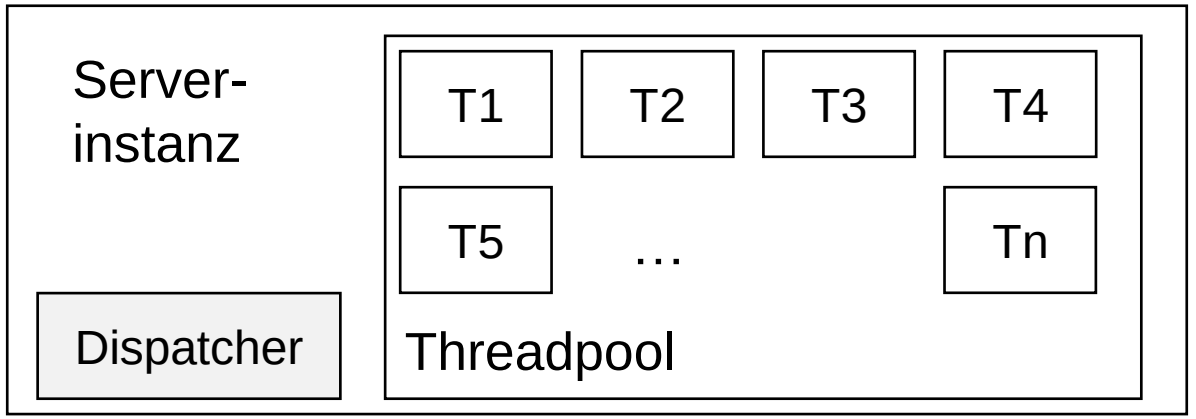


Nebenläufigkeit (Parallelität)

- Iterative (sequentielle) oder parallele Serverbausteine
- Threadpooling, Multithreading für die Bedienung mehrerer Clients gleichzeitig
- Ein Dispatcher ist ein Softwarebaustein im Server, der alle Requests der Clients entgegennimmt und sie auf Threads verteilt
- Einfaches sequentielles Programmiermodell für die Programmierer-Sicht
- Im JDK gibt es verschiedene Klassen für Thread-Pooling (s. java.util.concurrent)

Innenleben eines Servers

*Allg.: **Pooling** von Ressourcen = Vorbereiten zur schnelleren Nutzung*

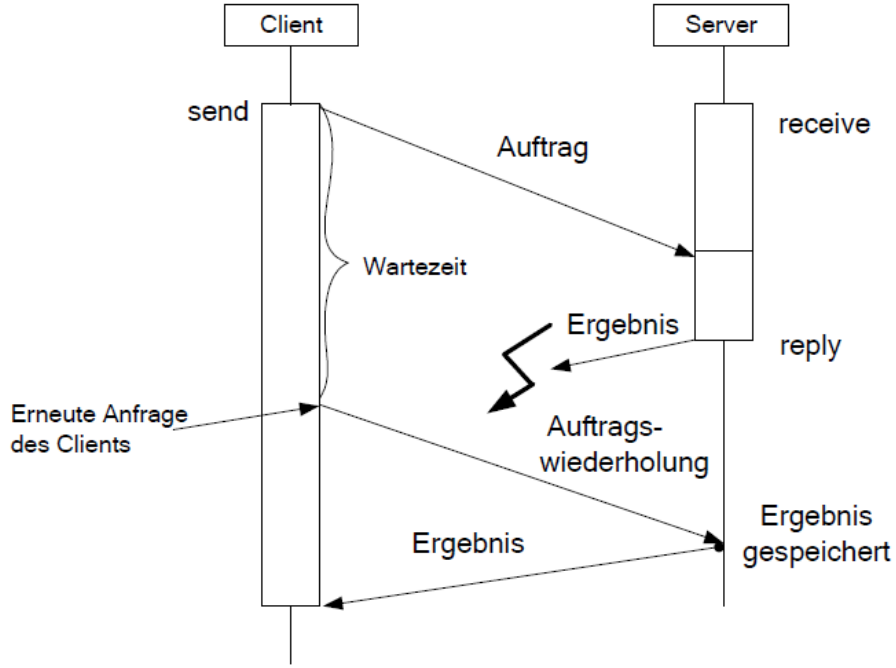


Dienst- bzw. Serviceschnittstellen

- Wie wird die Schnittstelle (Parameter- und Rückgabewertetypen) eines Serverbausteins beschrieben?
  - Neutrale Schnittstellenbeschreibungssprache oder eingebettet in Hostsprache (sprachabhängig)
  - Exception-Behandlung nicht immer gleich
- Diskussionsfrage:
  - Wie gut muss ein Server, der einen Service bereitstellt, prüfen, ob die empfangenen Parameter korrekt sind?

Fehlersituationen

- Es kann u.a. passieren, dass
  - ein Auftrag (engl. request) verloren geht,
  - das Ergebnis (engl. reply) des Servers verloren geht,
  - der Server während der Ausführung des Auftrags abstürzt,
  - der Server für die Bearbeitung des Auftrags zu lange braucht oder
  - der Client vor Ankunft des Ergebnisses abstürzt.



Parameterübergabe

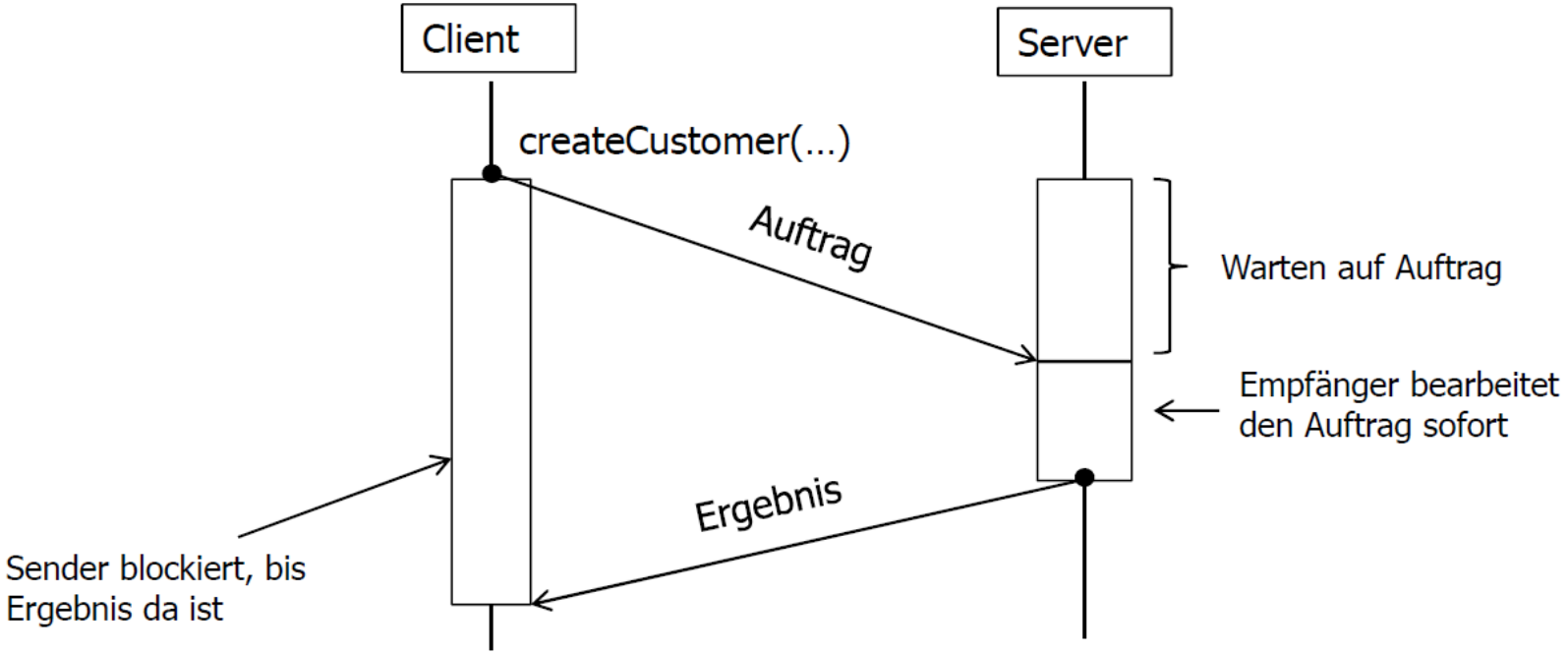
- Methodenaufruf und Parameterübergabe
  - ist lokal in demselben Prozess einfacher als bei entferntem (remote) Aufruf.
  - Entfernte Methodenaufrufe müssen für die Datenübertragung zwischen Rechnerknoten serialisiert (Marshalling) und deserialisiert (Unmarshalling) werden.
- Varianten für den entfernten Aufruf:
  - **Call-by-value:** Wert wird übergeben
    - Synonym: Call-by-copy
  - **Call-by-reference**: Verweis auf Variable wird übergeben
  - **Call-by-copy/copy-back**: Aufrufer arbeitet mit Kopie
    - Synonym: Call-by-restore = Call-by-value-result

Marshalling/Unmarshalling

- Marshalling/Unmarshalling ist das Umwandeln (Serialisierung/Deserialisierung) von strukturierten oder elementaren Daten für die Übermittlung an andere Prozesse.
- **Tag-basierte** Transfersyntax
  - Siehe ASN.1 mit BER (Basic Encoding Rules)
  - TLV-Kodierung (Type, Length, Value)
- **Tag-freie** Transfersyntax
  - Siehe Sun ONC XDR, CORBA CDR
  - Beschreibung der Daten aufgrund der Stellung in der Nachricht
  - Aufbau der Datenstrukturen ist dem Sender und dem Empfänger bekannt
- Meist automatische Erzeugung von Marshalling- und Unmarshalling-Routinen durch Compiler/Präcompiler
- Heute werden oft auch sprachunabhängige Notationen verwendet:
  - XML (Markup-Sprache), Tag-basiert
  - JSON (JavaScript Object Notation), Tag-basiert, sprachunabhängig?

Kommunikationsmodelle: Synchrone Kommunikation

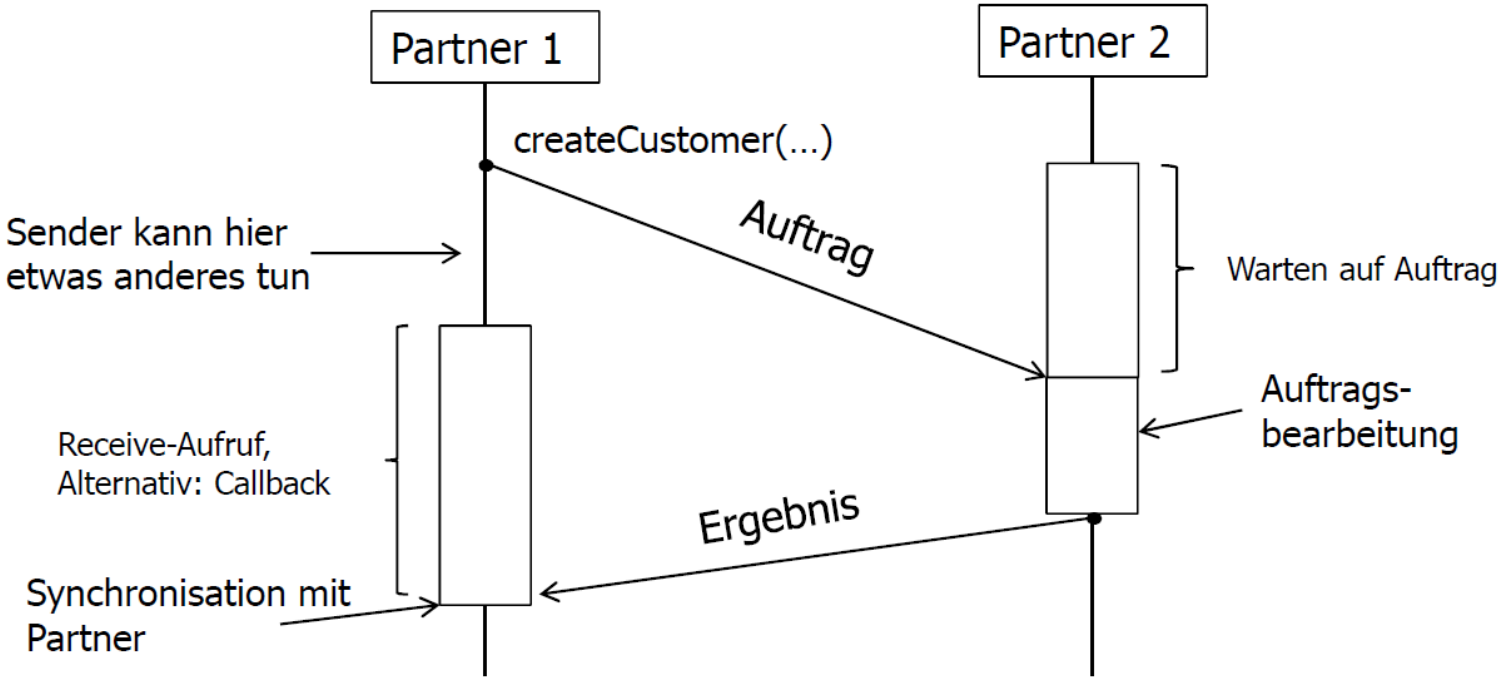
- Synchroner entfernter Dienstaufruf → blockierend
- Der Sender wartet, bis eine Methode send mit einem Ergebnis zurückkehrt



***Synchronisation** = **Synchronisierung (griech:** sýn = zusammen, chrónos = Zeit)**:** Aufeinander-Abstimmen von Vorgängen (zeitlich). Engere Bedeutung je nach Wissensgebiet: siehe Film, Informatik,…*

Kommunikationsmodelle: Asynchrone Kommunikation

- Asynchroner entfernter Serviceaufruf → Nicht blockierend, der Sender kann weiter machen



In der Datenkommunikation: asynchron = *Senden und Empfangen von Daten zeitlich versetzt und ohne Blockieren des Prozesses*

Kommunikation

- Namensauflösung und Adressierung auf der Anwendungsebene (entferntes Objekt oder Prozedur)
  - Naming- und Directory-Services notwendig
- Binding-Vorgang: Aufbau eines Verbindungskontextes zwischen Client und Server
  - Statisch zur Übersetzungszeit
  - Dynamisch zur Laufzeit
- Kommunikationsprotokoll für die Client-Server-Kommunikation
  - Nachrichtentypen (meist Request-Response-Protokolle)
  - Unterstützte Fehlersemantik
  - Unterstützung für verteiltes Garbage Collection

Zustandsverwaltung

- Server können zustandsinvariante und zustandsändernde Dienste bzw. Services anbieten
  - Zustandsändernde Dienste führen bei der Bearbeitung zu einer Änderung von Daten (z.B. in Datenbanken)
  - Zustandsinvariante Dienste verändern nichts
- Weiterer Aspekt: Server muss sich das Wissen über die Zustandsänderung über einen Aufruf hinweg merken
  - stateful und stateless Server
  - Stateless Server verwalten den aktuellen Zustand der Kommunikationsbeziehung zwischen Client und Server nicht
  - Wenn möglich: stateless!
- Zustandslose Kommunikationsprotokolle im Web: HTTP und REST für Webservices

Garbage Collection (GC)

- Verteiltes Reference-Counting
  - Server verwaltet eine Liste aller Clients (Proxies), die entfernte Referenzen nutzen
  - Server verwaltet Referenzzähler für alle benutzten Objekte
  - Client sendet spezielle Nachrichten an den Server, wenn Referenz benutzt bzw. gelöscht wird
- Leases
  - Referenz wird nur eine begrenzte Zeit für den Client freigegeben
  - Nach definierter Zeit löscht der Server die Referenz, wenn sich der Client nicht meldet
  - Ein Client kann sich somit problemlos beenden
- Zusammenarbeit mit lokalen GC-Mechanismen
  - Heap-Bereinigung

Lastverteilung, Hochverfügbarkeit, Skalierbarkeit

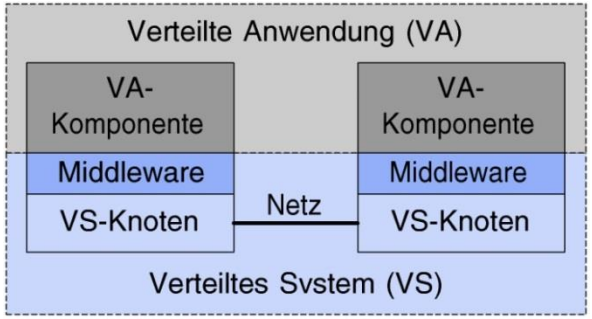
- Load Balancing (Lastverteilung)
  - Lastverteiler verteilen die Last auf mehrere Serverinstanzen
  - Dispatching z.B. über DNS-basiertes Request-Routing
- Hochverfügbarkeit
  - Server-Cluster, Beispiel: JBoss Cluster, Oracle Real Application Cluster
  - Failover
  - Session-Replikation
- Skalierbarkeit
  - Horizontal: Steigerung der Leistung durch Hinzunahme von Rechnern
  - Vertikal: Steigerung der Leistung durch Hinzufügen von Ressourcen zu einem Rechner (CPU, Speicher, …)

Agenda

1. Einführung in verteilte Systeme
2. Design- und Implementierungskonzepte von Client/Server-Systemen
3. **Middleware für verteilte Systeme**
4. Wrap-up und Ausblick

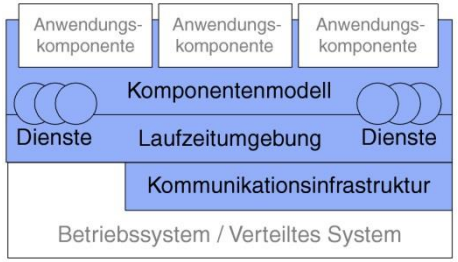
Middleware

- Middleware ist eine Softwareschicht, die den Anwendungen standardisierte, höhere Kommunikations- und sonstige Dienste über ein Application Programming Interface (API) bereitstellt und damit die transparente Kommunikation von Komponenten verteilter Systeme unterstützt.



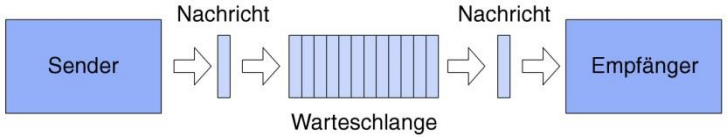
Middleware-Kategorien

- **Anwendungsorientierte Middleware**
  Java Enterprise Edition (EE) neu Jakarta EE
  Spring-Framework
  .NET Enterprise Services



- **Kommunikationsorientierte Middleware**
  Remote Procedure Call (RPC), Remote Method Invocation (RMI), REST, WebSocket …

- **Nachrichtenorientierte Middleware**
  Message Oriented Middleware (MOM),
  Java Messaging Service (JMS), MQTT …



Eine Middleware-Plattform vereinigt die verschiedenen Kategorien zu einer vollständigen verteilten Plattform für verteilte Anwendungen auf allen Tiers (Java EE, .NET)

Implementierungskonzepte: Konkrete Ansätze für das Client-Server-Modell

- **Remote Procedure Call (RPC)**
  - z.B. Sun ONC RPC, DCE RPC
- **Verteilte Objekte**
  - z.B. CORBA, Java RMI, .NET Remoting
- **Verteilte Services**
  - z.B. Webservices, SOAP, RESTful

Historische Entwicklung

Implementierungskonzepte: Konkrete Ansätze für das Client-Server-Modell

- Auf den folgenden Folien finden Sie verschieden Implementierungsansätze zu verschiedenen Ebenen der Programmierung.
  - Socket
  - RPC (remote procedure call)
  - RMI (remote method invocation) -> mit Fallstudie
  - Web-basierte Kommunikation, angewendete Protokolle
  - Websockets -> mit Fallstudie
  - Web-Services -> mit Fallstudie REST (Representational State Transfer)

Agenda

1. Einführung in verteilte Systeme
2. Design- und Implementierungskonzepte von Client-Server-Systemen
3. Middleware für verteilte Systeme
4. **Wrap-up und Ausblick**

Wrap-up

- Ein verteiltes System setzt sich aus einzelnen voneinander unabhängigen Bausteinen (Komponenten) zusammen, die dem Benutzer wie ein einzelnes kohärentes System erscheinen.
- Verteilte Systeme sind komplizierter als nicht verteilte Systeme und es müssen verschiedene praktische Probleme gelöst werden (Heterogenität, Fehlersituationen, Deployment etc.).
- Gängige Architekturstile verteilter Systeme sind Client-Server, Peer-to-Peer und Event Systems.
- Wichtige Design- und Implementierungsaspekte von Client-Server-Systemen sind Request-Handling (Threading), Design der serverseitigen Serviceschnittstellen, unterstützte Fehlersemantik, Parameter-Übergabe (Call-by-value, Call-by-reference), Kommunikationsstil (synchron, asynchron), Zustandsverwaltung und Garbage Collection.
- Grundlegende Architektur und Design Patterns für verteilte Systeme sind: Remote Proxy, Service Locator, Data Transfer Object und Remote Facade.
- Java RMI (Remote Method Invocation) ist die objektorientierte Umsetzung des RPCs (Remote Procedure Call) in Java und realisiert einen transparenten, entfernten Methodenaufruf.
- Web-basierte Applikationen verwenden das zustandslose Protokoll HTTP(S), Ajax und RESTful Webservices für die Kommunikation zwischen Browser und Webserver.

Ausblick

- In der nächsten Lerneinheit werden wir:
  - das Thema GUI-Architekturen vertiefen.

Quellenverzeichnis

[1] Mandl, P.: Masterkurs Verteilte betriebliche Informationssysteme, Springer-Vieweg, 2008

[2] Schill, A.; Springer, T.: Verteilte Systeme, 2. Auflage, Springer-Vieweg, 2012

[3] Fowler, M.: Patterns of Enterprise Application Architecture, Addison Wesley, 1. Auflage, 2002

[4] Martin, R. C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design, mitp Professional, 2018

[5] Abts D.: Masterkurs Client/Server Programmierung mit Java, 5. Auflage, Springer-Vieweg, 2019