

4 Architekturmodelle

- 4.1 Architekturelemente — Kommunikationsteilnehmer
- 4.2 — Kommunikationsparadigmen
- 4.3 — Rollen
- 4.4 — Orte
- 4.5 Architekturmuster — Ebenen
- 4.6 — Stufen
- 4.7 — Thin Clients



System-orientierte und Problem-orientierte Sichtweise

- **Prozesse:** intuitive Systemsicht, führt zu der weit verbreiteten Darstellung von verteilten Systemen als Menge von Prozessen, die über IPC-Mechanismen interagieren
 - ? sehr kleine Systemen ohne ein Prozess-Konzept
 - ? Threads — eigentlich sind nicht die Prozesse, sondern die Threads die Kommunikationsteilnehmer
- **Objekte:** Problem-orientierte Sicht — in einem verteilten Objekt-basierten System bestehen Anwendungen aus einer Menge verteilter, interagierender Objekte
 - ? Objekte mit mehreren Threads
- **Komponenten:** Weiterentwicklung des Objekt-Konzepts hin zu Software-Bausteinen
- **Web Services:** ähnlich zu Komponenten, aber explizit auf das WWW als Kommunikationsplattform ausgerichtet



direkte Kommunikation

- **Interprozesskommunikation**
 - *low-level*-Mechanismen für Nachrichtenübertragung
- **Fernaufruf**
 - *Request-Reply-Protokolle*
 - Prozedur-Fernaufruf
 - Methoden-Fernaufruf
- Details in Kapitel 5



indirekte Kommunikation

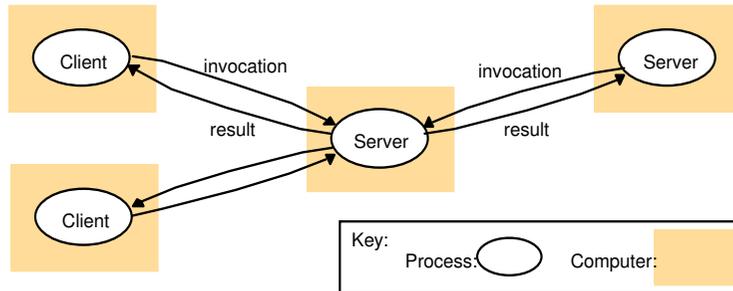
- **Gruppenkommunikation**
 - Abstraktionsmechanismus für mehrere Nachrichtenempfänger
 - Sender kennt nur die *Gruppe*, nicht aber die konkreten Empfänger
 - Basismechanismus für fehlertolerante Systeme
- **Publish-subscribe Mechanismen**
 - Viele Produzenten (*Publisher*) verteilen Informationseinheiten (*Events*) an eine große Menge von Empfängern (*Subscribers*).
- **Message Queues**
 - Indirekte 1:1-Kommunikation, entkoppelt über *Briefkasten*
- **Tuple Spaces**
 - Prozesse platzieren strukturierte Dateneinheiten (*Tuples*) in einem persistenten Speicherbereich (*Tuple Space*), andere Prozesse können die Daten dort lesen oder entnehmen.
 - Leser und Schreiben müssen nicht gleichzeitig existieren.
- **Distributed Shared Memory**
 - Abstraktion für gemeinsamen Speicher im Verteilten System.



Architekturelemente: Rollen

Client-Server

- üblichste Kommunikationsart in Verteilten Systemen



Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

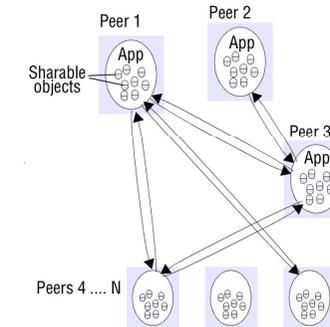
- Prozesse nehmen *Client*- oder *Server*-Rolle ein.
- Server können zur Erbringung ihres Dienstes auch die Dienste weiterer Server in Anspruch nehmen — und werden damit zu Klienten.
- Beispiele: Web-Server, Mail-Server, File-Server, ...



Architekturelemente: Rollen

Peer-to-Peer

- Interaktion gleichberechtigter Prozesse im Netzwerk



Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

- zentralisierte P2P-Systeme: Server zur Verwaltung vorhanden
- Beispiele: Datei-Tauschbörsen



Architekturelemente: Orte

Dienste auf mehrere Server abbilden

- Prozesse auf mehreren Rechnern erbringen gemeinsam einen Dienst
- Teile des Dienstes können auf verschiedene Server platziert werden
- Lastverteilung zwischen gleichartigen Servern

Caching

- kürzlich benötigte Objekte werden näher am Klienten gespeichert
- bei Anfragen wird die Aktualität des Objekts überprüft und nur bei Bedarf vom eigentlichen Server neu übertragen

Mobiler Code

- Code wird vom Server geladen und lokal ausgeführt
- Beispiele: *Applets*, Javascript

Mobile Agenten

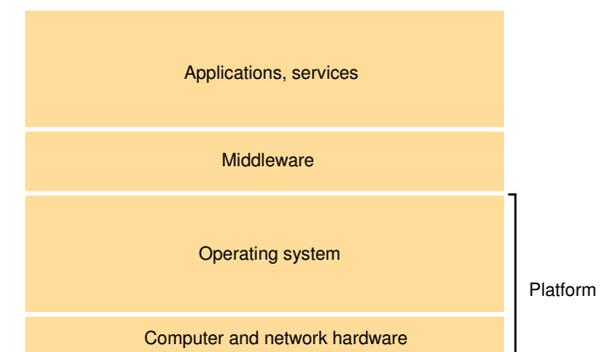
- Programme wandern zwischen Rechnern und führen Aufgaben aus
- Beispiel: Installation und Wartung von Software in einem Firmennetzwerk



Architekturmuster: Ebenen (*Layers*)

Grundlegendes Abstraktionskonzept

- jede Ebene nutzt die Dienste der darunterliegenden Ebene
- jede Ebene bietet nach oben eine Software-Abstraktion an (und verbirgt damit Implementierungsdetails)



Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012



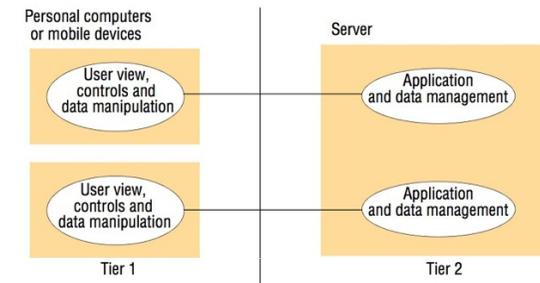
Architekturmuster: Ebenen (Layers) (2)

- Plattform für Verteilte Systeme:
 - umfasst die grundlegenden Hardware- und Software-Ebenen
z. B. Intel x86 / Windows, ARM / Linux
 - bietet einfache Mechanismen zur Kommunikation und Koordination von Prozessen im Verteilten System
z. B. TCP/IP-Implementierung mit Socket-Schnittstelle
- Middleware
 - verbirgt Heterogenität und bietet Programmierparadigma-gerechte Schnittstellen
 - Plattform für verteilte interagierende Prozesse oder Objekte
 - unterstützt verteilte Anwendungen durch geeignete Abstraktionen und Dienste
z.B. Nameservice, verteilte Transaktionen, Objektmigration, ...
 - Beispiel: CORBA



Architekturmuster: Stufen (Tiers)

- Komplementär zu dem Konzept der Ebenen
 - Ebenen dienen der vertikalen Strukturierung eines Dienstes in Abstraktionsebenen
 - Organisation in Stufen (Tiering) strukturiert die Funktionalität einer Ebene in verschiedene Server, ggf. auf verschiedenen Knoten im Netz

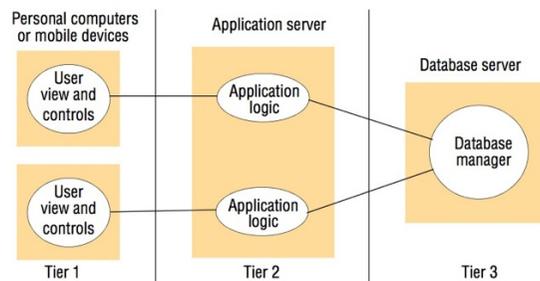


Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012



Architekturmuster: Stufen (Tiers) (2)

- Typische Struktur einer *three-tier*-Lösung:
 - *Presentation logic*: Benutzerinteraktion und Darstellung der Anwendung
 - *Application Logic (Business Logic)*: Anwendungsspezifische Verarbeitung der Daten (Berechnungen/Algorithmen, Strategien)
 - *Data logic*: Persistente Speicherung der Anwendungsdaten - z. B. in einer Datenbank

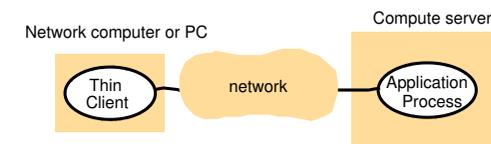


Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012



Architekturmuster: Thin Clients

- Verlagerung der Komplexität vom Endgerät des Benutzers auf Dienste im Internet
 - wird vor allem im Bereich *Cloud Computing* deutlich
 - auch in gestuften Architekturen erkennbar



Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012



Architekturmuster: *Thin Clients* (2)

- Vorteil: Einsatz sehr einfacher Nutzer-Endgeräte
 - kostengünstig
 - einfache Installation und Wartung der Software
 - mehrere Abstufungen möglich: kleiner PC, SunRay, VNC-Software
- Funktionalität des Endgeräts wird durch eine Vielzahl von Diensten im Netz erweitert
- Problem: rechenintensive oder graphisch aufwändige Anwendungen (z. B. CAD-Systeme)
- **Virtual Network Computing (VNC)**
 - lokales System wird nur zur Anzeige der Grafik (auf FrameBuffer-Ebene) genutzt, jede Mausbewegung und Tastatureingabe wird an den Server übertragen



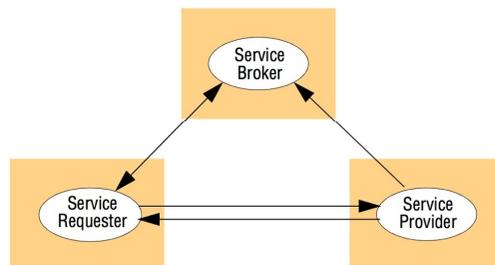
Architekturmuster: *Proxies*

- Grundlegendes Softwaremuster in verteilten Anwendungen, um **Ortstransparenz** zu erreichen.
 - lokaler Proxy agiert als Stellvertreter für das entfernte Objekt
 - Proxy bietet die gleiche Schnittstelle wie das entfernte Objekt
 - Verteiltheit wird vor dem Aufrufer verborgen (nicht ganz: anderes Zeit- und Fehlverhalten möglich!)
 - Details siehe Kapitel 5
- weitere Anwendungsmöglichkeiten für Proxies
 - Zwischenspeicherung (Caching) von entfernten Objekten (Reduzierung der Netzlast, schnellere Antwort)
 - Verbergen von replizierten Objekten
 - spezielle Zugriffskontrollmechanismen



Architekturmuster: *Broker*

- Unterstützung von Interoperabilität in komplexen verteilten Infrastrukturen
 - wie findet ein Dienst-Nutzer den Dienst-Anbieter?
- typische Beispiele:
 - Naming Service in der CORBA-Middleware
 - Registry in Java RMI



Architekturmuster: *Reflection*

Reflection unterstützt zwei Konzepte:

- **Introspection**
 - die Möglichkeit, die Eigenschaften (z.B. Schnittstellen) eines Systems dynamisch (zur Laufzeit) zu untersuchen
 - dynamische Erzeugung von Proxies aus den Schnittstelleninformationen
 - Vermittlung von Aufrufen durch einen generischen Server an die passenden Schnittstellen hintergelagerter Diensterbringer
- **Intercession / Interception**
 - die Möglichkeit, dynamisch Strukturen und Verhalten zu verändern
 - Abfangen von Aufrufen (für zusätzliche Sicherheitsprüfungen, Optimierungen, Replikation, ...)

