3.3 Architekturprinzipien

Architekturmittel

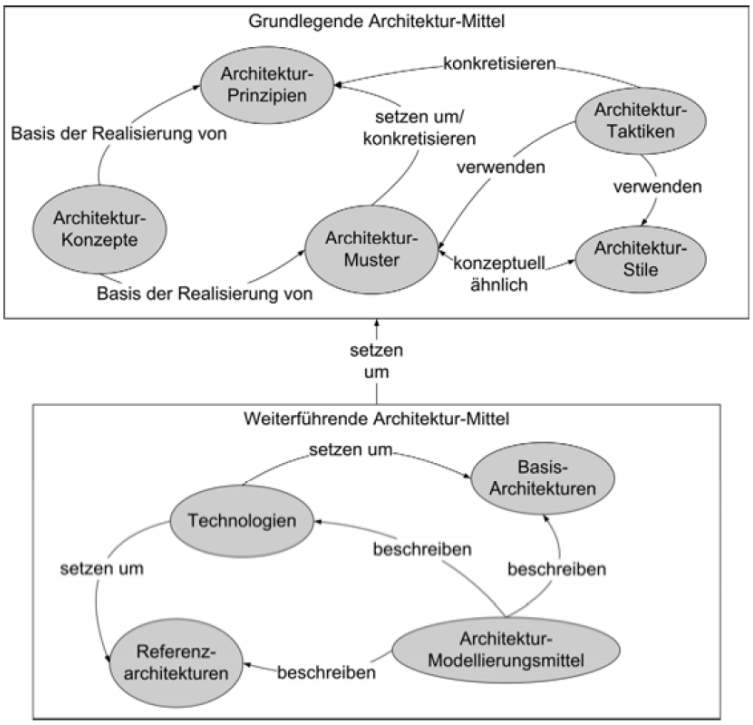


Bild 3.17: Grundlegende Konzepte für Architektur-Mittel, aus [VZM+09]

Wir konzentrieren uns auf Architekturpinzipien

Prinzipien

- z.B. lose Kopplung oder hohe Kohäsion
- sehr allgemeine Richtlinien

Grundlegende Konzepte

- z.B. Objektorientierung oder Aspektorientierung
- konkretisieren Prinzipien
- Basis zur Realisierung der Prinzipien

Taktiken

- konkretisieren allgemeine Prinzipien, indem sie Handlungsanweisungen auf Basis von Qualitätsattributen anbieten

Architektur-Muster und Architektur-Stile

- detaillierte Lösungsansätze für konkrete Entwurfsentscheidungen

Basis-Architekturen

- z.B. Schichtenarchitekturen, n-Tier-Architekturen
- konkrete Richtlinien, mit denen man Systeme ganzheitlich strukturieren kann

Referenzarchitekturen

- allgemeine Lösungsansätze für konkrete Domänen
- oft kombiniert mit bestimmten Technologieansätzen

Architektur-Modellierungsmittel

- z.B. UML, Domain Specific Languages oder Architecture Description Languages
- Ansätze zur Modellierung und Dokumentation von Architekturen

Architektur-Technologien

- z.B. Web-Services, Object-Relational-Mapping, XML
- z.B. Plattformen und Infrastrukturen, mit einer grundlegenden Bedeutung für die Architektur des Systems

---

Einfluss von Architektur-Mitteln

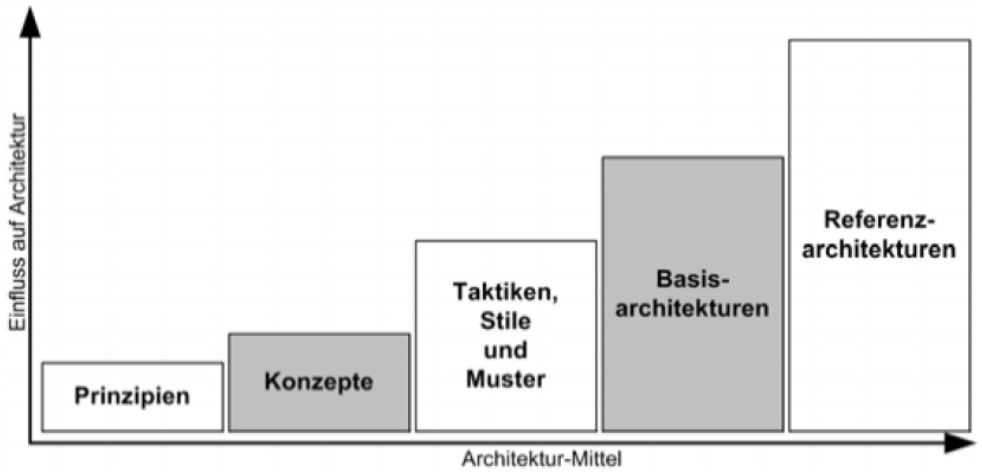


Bild 3.18: Einfluss von Architektur-Mitteln, aus [VZM+09]

- Referenzarchitekturen nehmen viele Entscheidungen vorweg
- Freiheitsgrad sinkt
- Wiederverwendungsgrad steigt

Kategorisierung der Architektur-Prinzipien

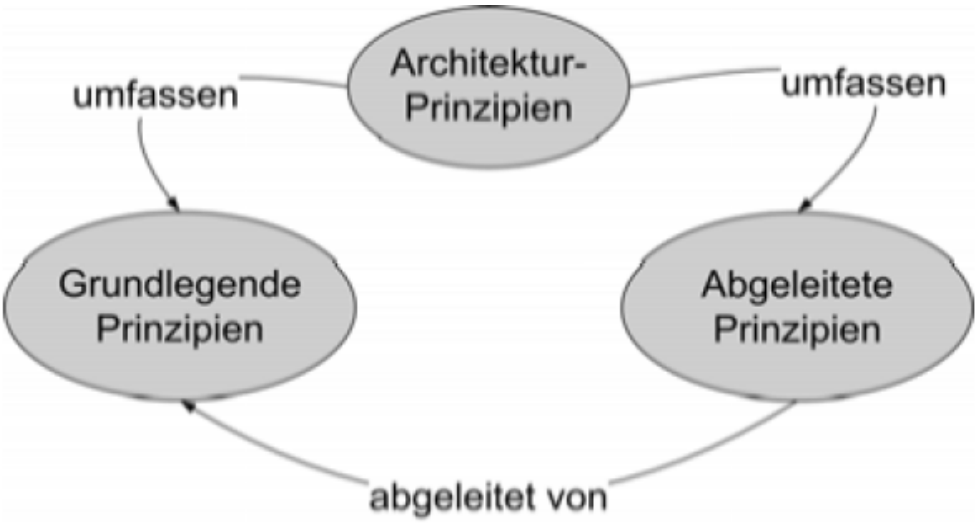


Bild 3.19: Übersicht zur Kategorisierung der Architektur-Prinzipien, aus [VZM+09]

Grundlegende Architektur-Prinzipien - Übersicht

Abgeleitete Architektur-Prinzipien - Übersicht

3.4 Basisarchitekturen

- grundlegende Basisarchitekturen, nach [VZM+09], Abschnitt 6.4, S. 216ff

---

Basisarchitekturen

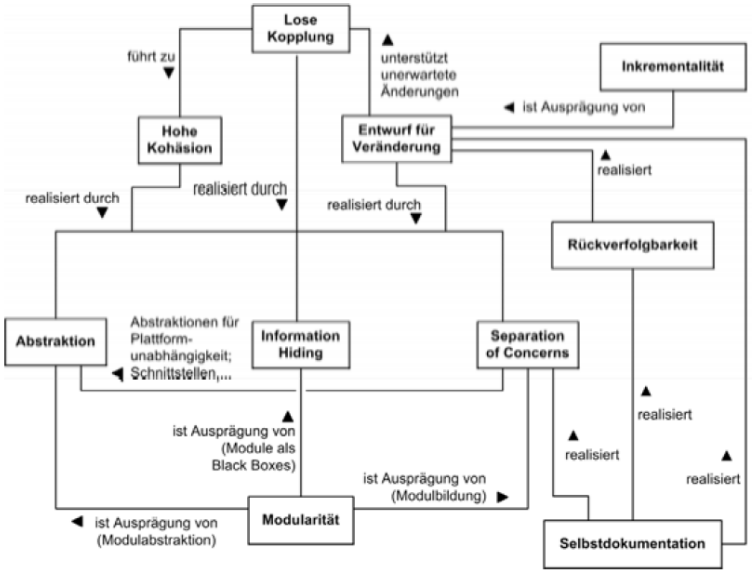


Bild 3.20: Übersicht zu den grundlegenden Architektur-Prinzipien, aus [VZM+09]

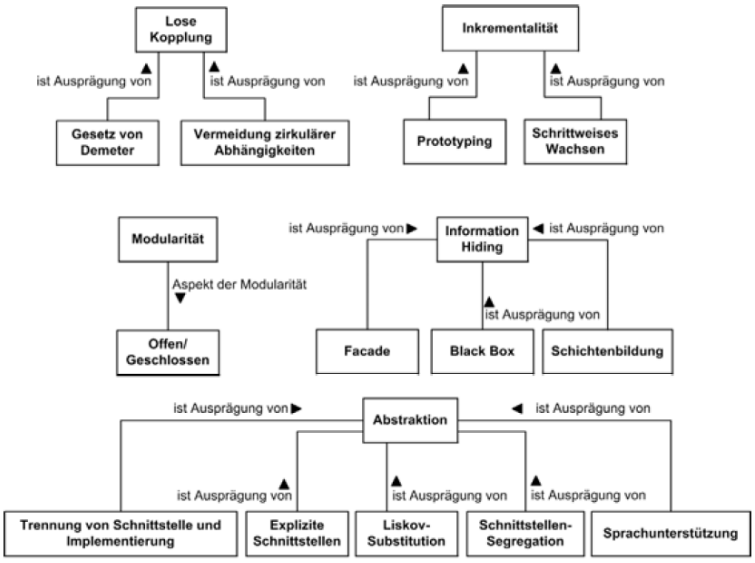


Bild 3.21: Übersicht zu den abgeleiteten Architektur-Prinzipien, aus [VZM+09]

Monolith

- komplette Architektur in einem einzelnen Systembaustein zusammengefasst
- Separation of Concerns nicht zu erreichen
- lose Kopplung nicht zu erreichen
- typisches Beispiel für Monolithen: Altsysteme

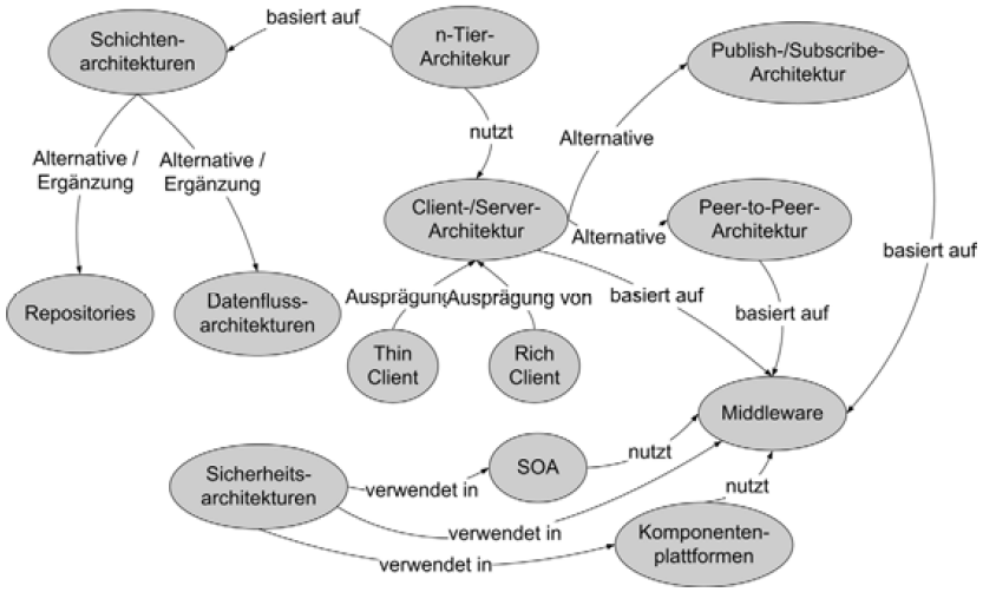


Bild 3.22: Überblick Basisarchitekturen, aus [VZM+09]

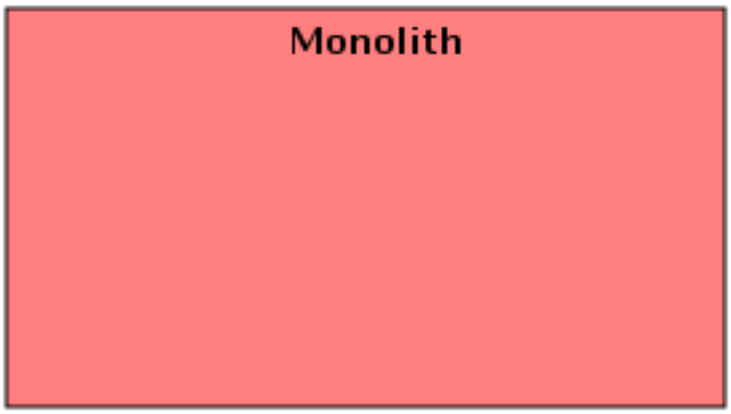


Bild 3.23: Monolith

Monolith, Symptome

- Anpassung der Geschäftslogik an neue Anforderungen schwierig
- Tests neuer Releases aufwendig und langwierig
- kleine Änderungen sehr aufwendig
- kleine Änderungen gefährden die Stabilität des Gesamtsystems
- wenige altgediente Mitarbeiter kennen sich mit der Software aus

3.4.1 Schichtenarchitekturen

- in Gruppen von ähnlichen Funktionen bzw. Verantwortlichkeiten unterteilen
- eine Gruppe von Bausteinen hängt von einer anderen Gruppe von Bausteinen ab
- siehe Entwurfsmuster Schichten (layers), Skript Softwarekonstruktion [See12]

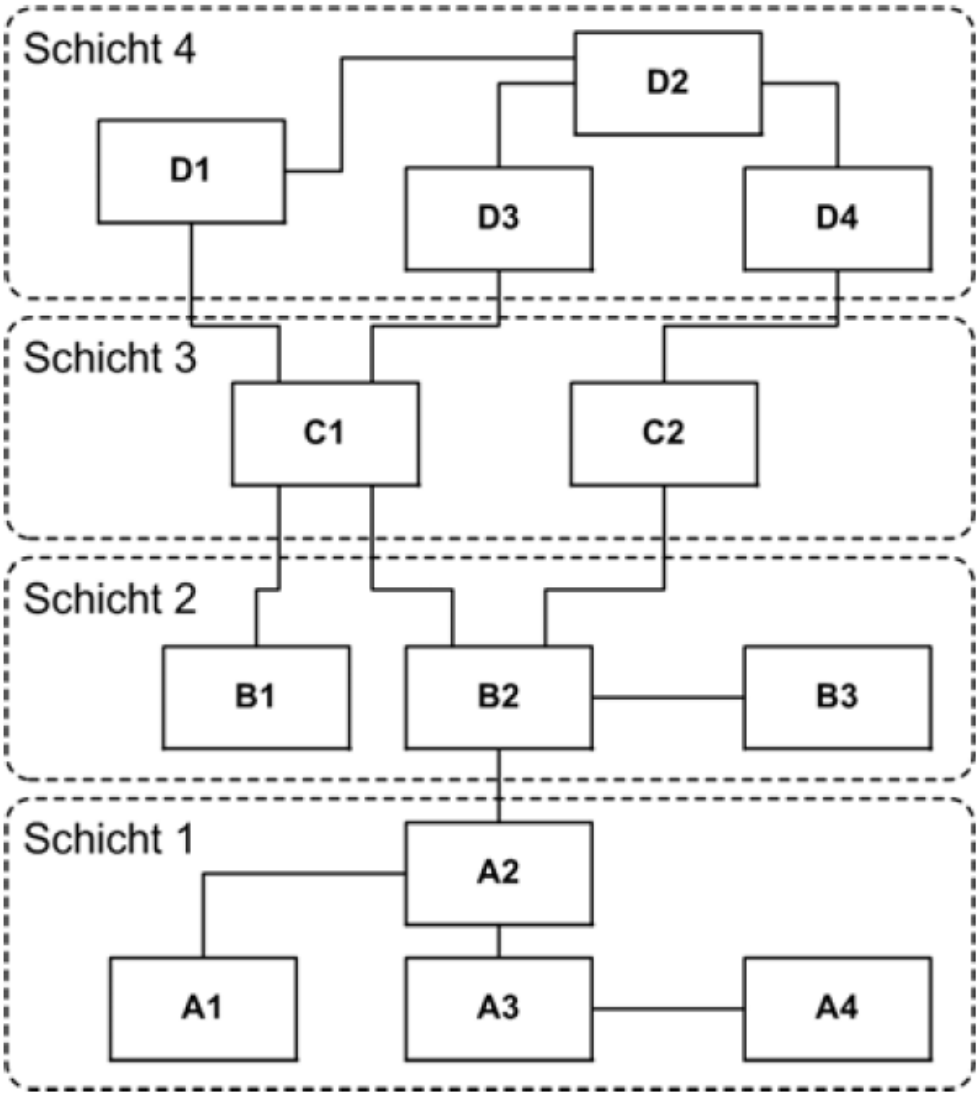


Bild 3.24: Beispiel Schichtenarchitektur, aus [VZM+09]

Schichtenarchitekturen

Hauptziele

- Veränderbarkeit des Systems erhöhen

- Portierbarkeit erhöhen
- Wiederverwendbarkeit der Schichten erhöhen

3.4.2 Datenflussarchitekturen

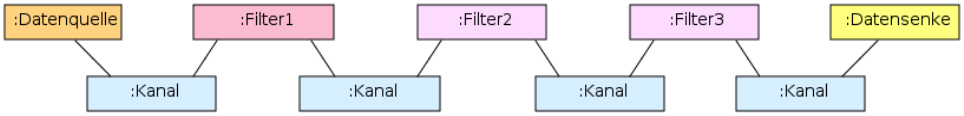


Bild 3.25: Beispiel Filter, aus [See12]

- Strukturierung entlang der Datenflüsse einer Architektur
- komplexe Aufgabe als Kombination unabhängiger Aufrufe einfacher Aufgaben

---

Batch-Sequential-Stil

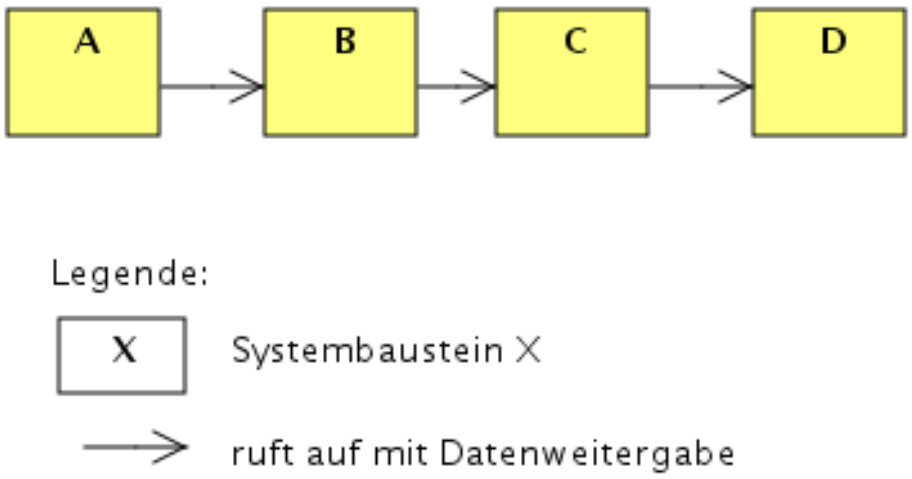


Bild 3.26: Schema Batch-Sequential

- Gesamtaufgabe in Teilschritte zerlegt und
- als separate und unabhängige Bausteine implementiert

- Jeder Teilschritt wird bis zur Vollendung durchlaufen und
- ruft dann den nächsten Schritt in einer Sequenz von Schritten auf
- In jedem Schritt werden die Daten zu Berechnungen herangezogen und
- die Ergebnisdaten werden als Ganzes an den nächsten Schritt weiter gegeben

```
S1(D1, D2); S2(D2, D3); S3(D3, D4); ... ; Sn(Dn, Dm);
```

Filter (Pipes-and-Filters)

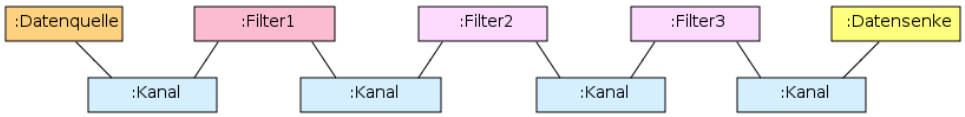


Bild 3.27: Beispiel Filter, aus [See12]

- Ströme von Daten werden verarbeitet
- verschiedene Kombinationen der Verarbeitungsschritte benötigt
- siehe [See12]

Beispiel: Java-Servlet-Filter

- auf die Anfragen und Antworten bei dem Zugriff auf eine Web-Ressource zugreifen
- mehrere Filter können in einer Kette flexibel jeweils für eine bestimmte Web-Ressource konfiguriert werden
- Beispiele:
  - Logging,
  - Verschlüsselung und Entschlüsselung,
  - Komprimierung und Dekomprimierung,
  - Transformation (z. B. von XML mittels XSLT).

3.4.3 Repositories

- Repository: verschiedenen Bausteinen den gleichzeitigen Zugriff auf Daten ermöglichen
- **Shared-Repository-Stil**:
- ein Baustein des Systems stellt zentralen Datenspeicher zur Verfügung
- mit effizientem Zugriff, Skalierbarkeit, Gewährleistung der Konsistenz der Daten, ...

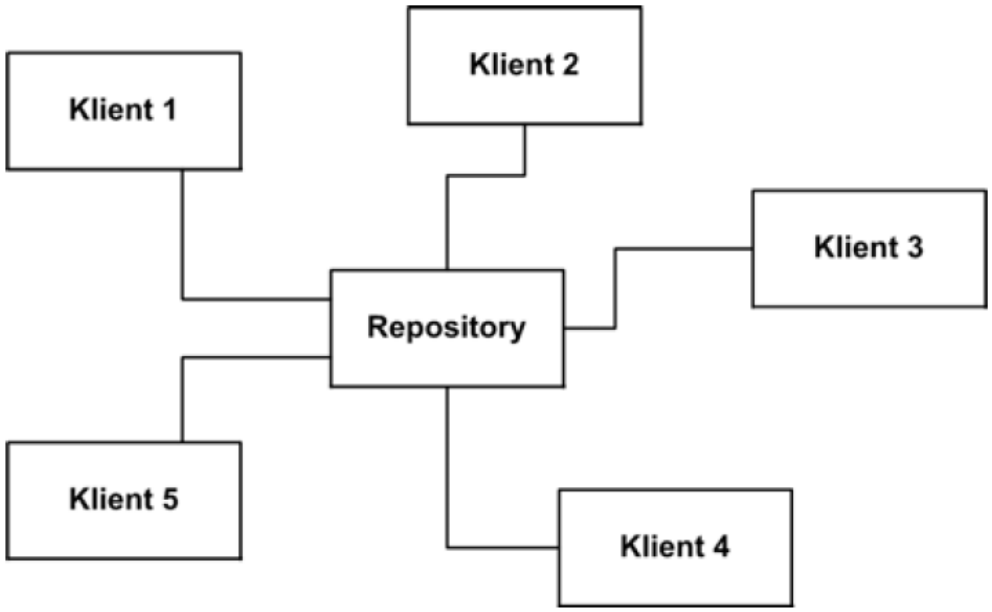


Bild 3.28: Beispiel Repository-Architektur, aus [VZM+09]

3.4.4 Zentralisierung - Dezentralisierung

- Kompromiss Zentralisierung versus Dezentralisierung
- Monolith: extreme Form der Zentralisierung
- alle anderen Architekturen: Kompromisse

Vorteile der Dezentralisierung

- günstigere Hardware
- flexibler gegenüber Veränderungen
- robuster bei Ausfällen einzelner Bausteine
- einfacher, die Architektur aufgabenorientiert zu strukturieren

---

Vorteile der Zentralisierung

- zentrale Aufgaben im Vordergrund
- z.B. bei IT-Sicherheit, Logging, Kontrolle, Monitoring
- Auslieferung neuer Software-Bausteine einfacher
- Vorteile im Bereich der Kosten für den personellen Betreuungsaufwand
- typisch: Verwaltung großer Datenbestände, Netzwerksteuerung, Steuerung der Transaktionsverarbeitung, laufende Prüfung der Hard- und Software im gesamten Netz
- teilweise niedrigere Hardware-Kosten
- teilweise niedrigerer Energieverbrauch

Mainframe-Architektur

- teurer und leistungsfähiger Großrechner (englisch: mainframe)
- gemeinsame Anwendungen
- mit **Terminals** auf Mainframe zugegriffen

File Sharing

- kostengünstiger Personal Computer (PC)
- PC-Rechnernetze basierten auf File Sharing
- Server stellt Dateien in einem gemeinsamen Speicher zur Verfügung
- Netzbelastung beim File Sharing relativ hoch
- Konsistenzprobleme

Client-/Server-Modell

- Der Anwender betreibt auf seinem Rechner Anwendungsprogramme (**Klienten**, englisch: clients),
  welche auf die Ressourcen des **Servers** zugreifen
- Ressourcen werden zentral verwaltet
- einfaches Anfrage-Antwort-Schema
- Dateien müssen nicht als Ganzes übertragen werden

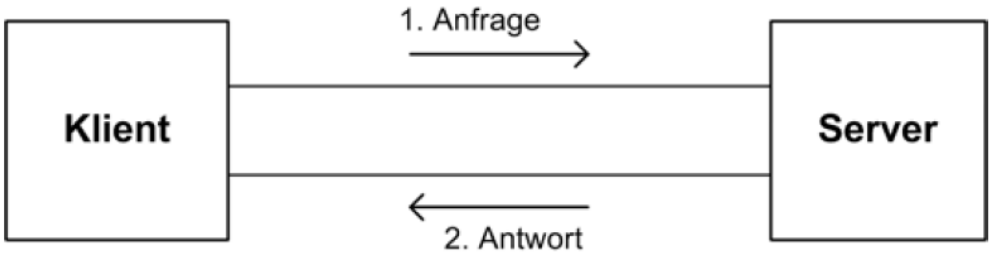


Bild 3.29: Client-/Server-Modell, aus [VZM+09]

- Z. B. greift der Server auf eine (relationale) Datenbank zu und führt eine Abfrage durch, die in der Anfrage des Klienten spezifiziert wurde

Mischformen

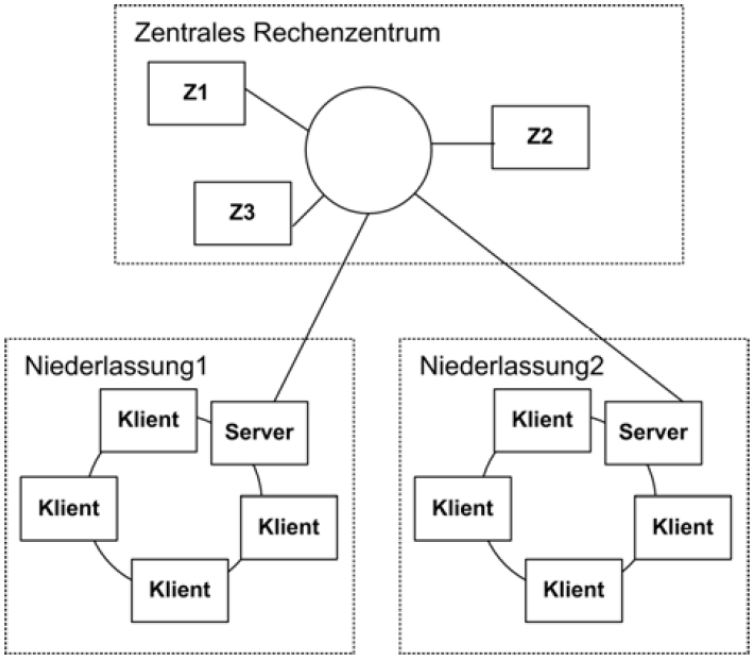


Bild 3.30: Beispiel eines Rechnernetzes, Mischform, aus [VZM+09]

- Zentralisierung gegenüber Dezentralisierung auch in „größerem“ oder „kleinerem“ Kontext
- Mischformen aus zentralen und dezentralen Elementen

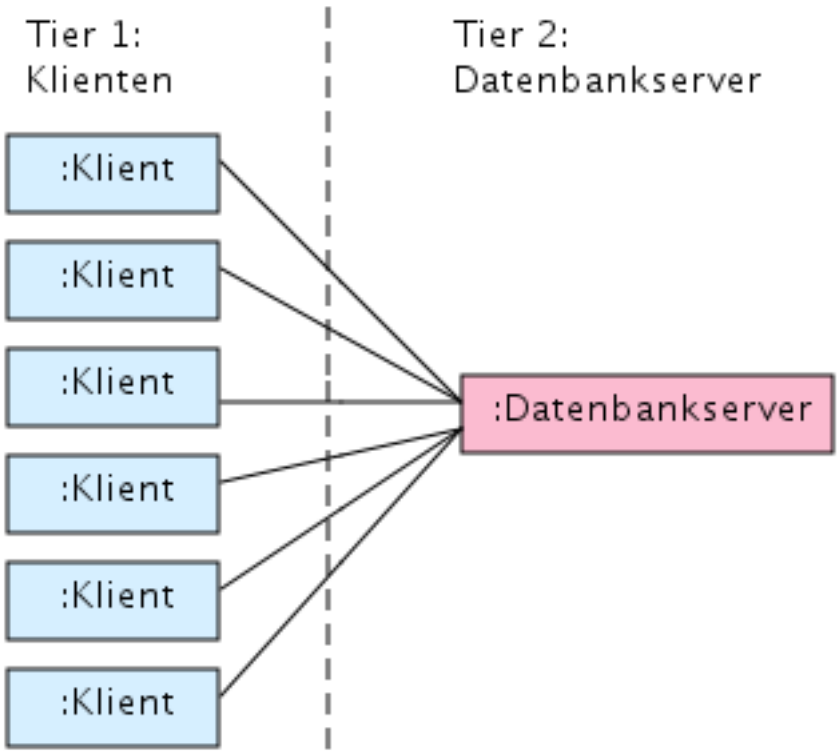


Bild 3.31: klassische 2-Tier-Architektur

3.4.5 n-Tier-Architektur

- klassische Client-/Server-Modell basiert auf einer sogenannten **2-Tier-Architektur**
- Benutzerschnittstelle ist für gewöhnlich auf dem PC des Anwenders
- Datenbankmanagement ist auf einem leistungsfähigerem Rechner
- 2-Tier-Architekturen funktionieren bis zu einer bestimmten Anzahl von Klienten gut
- Bei sehr hohen Nutzerzahlen nimmt die Performanz rapide ab
- Abhängigkeit vom Hersteller der Datenbank

3-Tier-Architektur

- Zwischenschicht zwischen Klient und den Datenbankserver
  - Queuing (Aufreihen) von Anfragen,

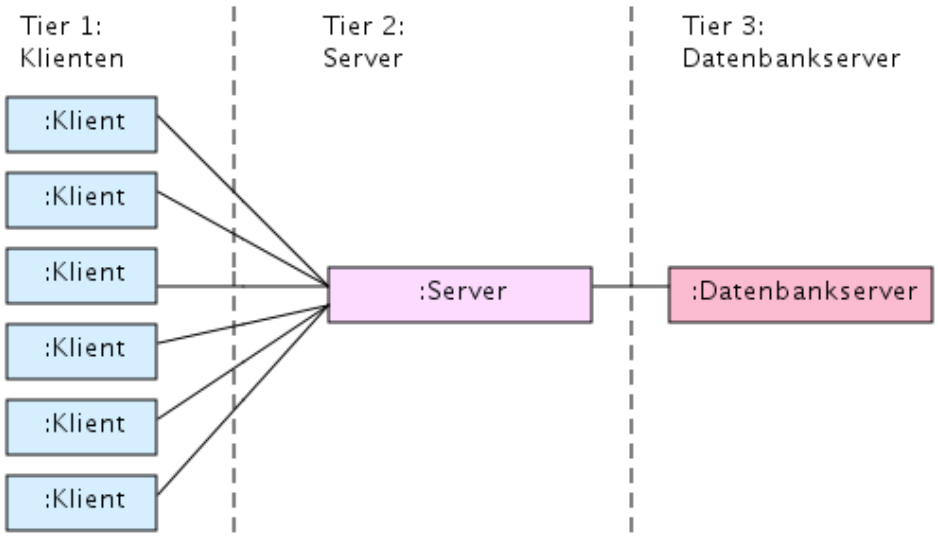


Bild 3.32: klassische 3-Tier-Architektur

  - Durchsetzen von Ablaufplänen für Anfragen (englisch: scheduling),
  - Behandeln von Anfragen gemäß Prioritäten ...
  - zentrale Aufgaben der Anwendungslogik
- Standardlösungen
  - Transaktionsmonitore
  - Messaging Server
  - Anwendungsserver
- Verbesserung der Performance bei großer Anzahl von Klienten und zu einer Steigerung der Flexibilität

---

n-Tier-Architekturen

- mehr Tier:
  - hohe Lastzahlen
  - Sicherheitsanforderungen
  - Ausfallsicherheitsanforderungen
  - weitere Qualitätsanforderungen
  - ...

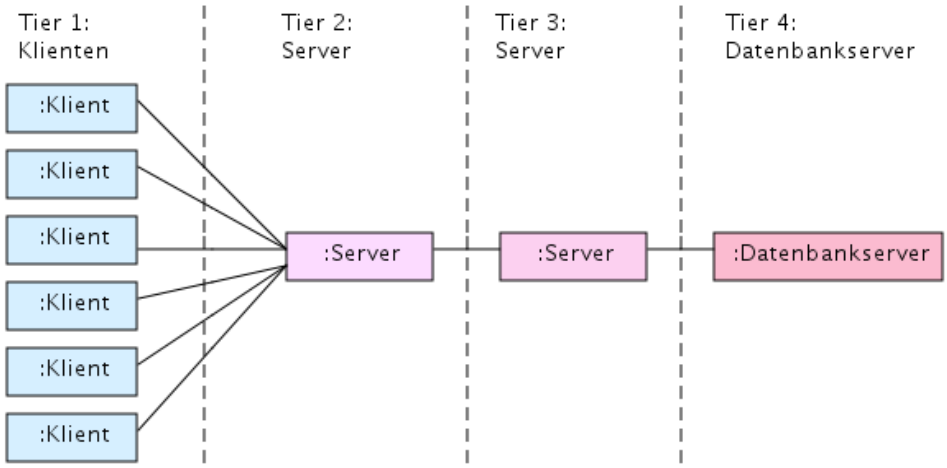


Bild 3.33: 4-Tier-Architektur

3.4.6 Rich Client - Thin Client - extrem

- Frage:
  - wie **Funktionalität zwischen Klienten und Server aufteilen**
- extremer Thin Client:
  - Klient hat so gut wie gar keine Funktionalität,
  - z.B. Terminal
- extremer Rich Client:
  - fast alle Berechnungen werden beim Klienten durchgeführt,
  - z.B. File Sharing

Rich Client

Thin Client

Ultra Light Client

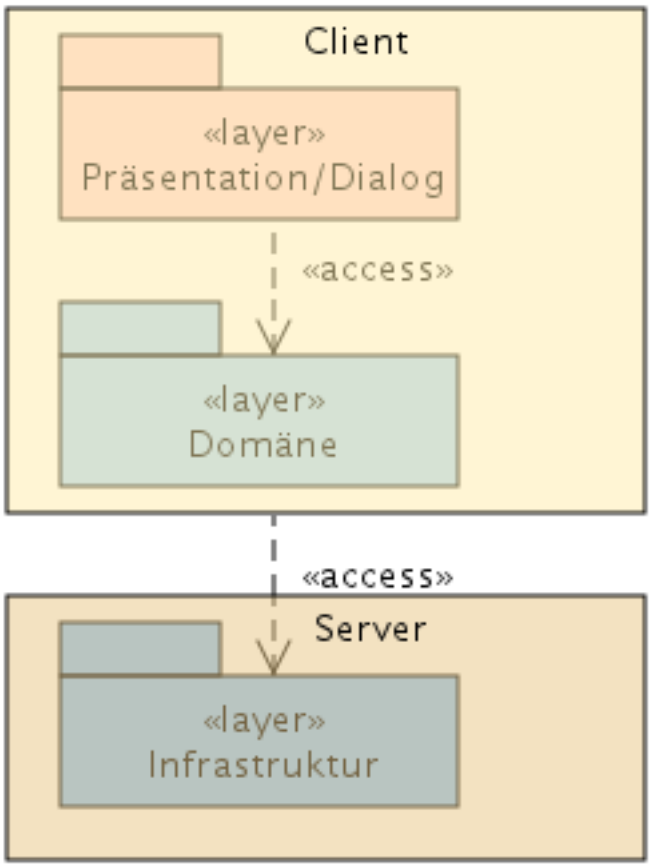


Bild 3.34: Rich Client

Rich Client - Thin Client

- **Server-Ressourcen**:
  Ein Thin Client belastet die Server-Ressourcen generell stärker,
  denn es müssen mehr Berechnungen am Server durchgeführt werden

- **Netzbelastung**:
  auf eine Netzanfrage muss man erheblich länger warten als auf eine lokale Anfrage
  Art und Menge der zu übertragenden Daten sind wichtig

- **Funktionsfähigkeit des Netzwerkes**:
  Thin Clients stark abhängig davon,
  daher z.B. Rich Clients für Laptops von Außendienstmitarbeitern

- **Wartung und Auslieferung der Klienten**:
  Zentrale Software-Bausteine können am Server direkt ausgetauscht werden.

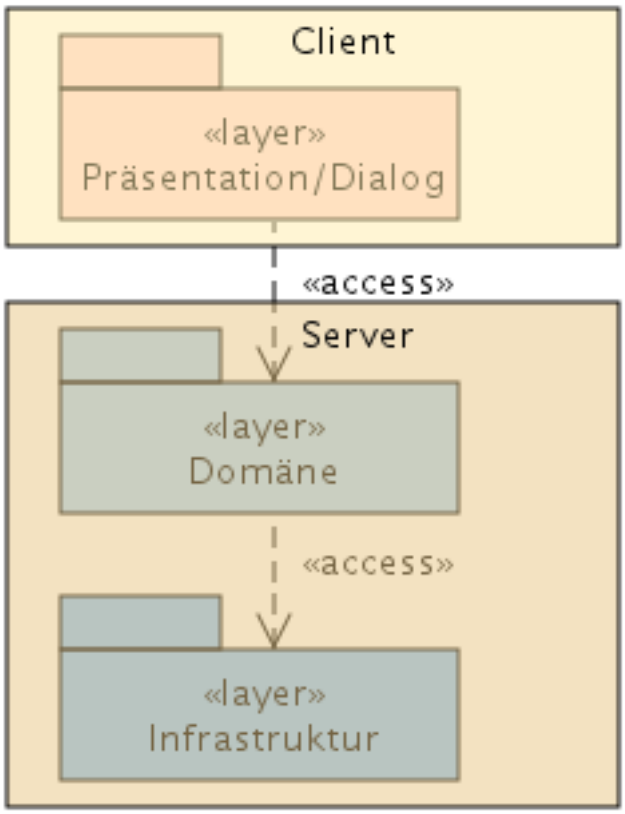


Bild 3.35: Thin Client

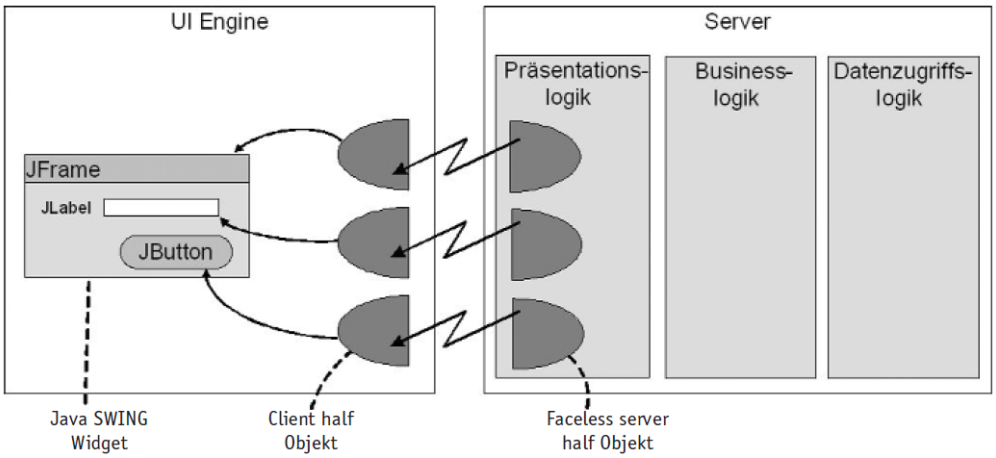


Bild 3.36: Darstellung einer beispielhaften UltraLightClient-Anwendung: Die gesamte Anwendung einschließlich der Präsentationsschicht läuft auf dem Server. Die UI-Engine übernimmt den Aufbau der Benutzungsoberfläche und kommuniziert mit dem Server über das Half-Object-Muster, aus [Neu05]

Typische Lösungen - Anwendungsserver und Web Browser

- **Anwendungsserver und Web Browser** als Thin Client

- Klienten kaum noch warten/aktualisieren
- eingeschränkte Benutzeroberfläche
- ohne Netzzugang ist die Arbeit nicht (ohne Weiteres) möglich
- Performanz der Anwendung eher gering

Typische Lösungen - Rich-Client-Plattformen

- **Rich Clients**
  - Rich-Client-Plattformen, wie Eclipse
  - automatisches Ausliefern und Updaten von Klienten in einer Standardumgebung
  - Benutzerumgebung steht einer klassischen Desktopanwendung in nichts nach

---

Typische Lösungen - Web 2.0 / Ajax

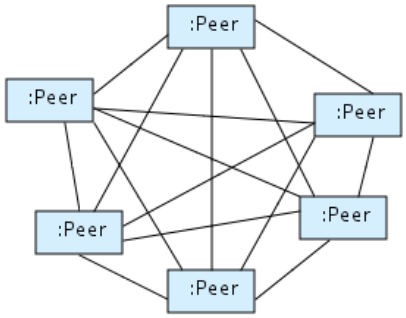
- **Web 2.0** bzw. **Ajax**
  - mit Javascript asynchrone Abfragen durchführen
  - Inhalte nachladen und verändern,
    ohne dass der Nutzer die Web-Seite wechseln muss
  - Teile der Aufgaben werden im Browser durchgeführt
  - Skripts, die diese Aufgaben implementieren, werden vom Server ausgeliefert
  - (Web 2.0 keine spezifische Technologie, sondern bestimmte Arten von Web-Anwendungen, in denen Nutzer Inhalte selbst erstellen und bearbeiten)

3.4.7 Peer-to-Peer-Architektur

Peer-to-Peer-Architektur

Peer-to-Peer-Stil

- direkte Kommunikation von Klienten



P2P-Modell

- Peer-to-Peer (P2P)
- keine zentralen Server
- Jeder Peer kann im Netzwerk Services anbieten und konsumieren
- Gesamtzustand des Systems ist über die Peers verteilt
- Dienst kann zu jeder Zeit hinzugefügt und wieder entfernt werden
- Klienten müssen herausfinden, welche Dienste gerade zur Verfügung stehen, bevor sie einen Dienst nutzen.
- Die meisten Systeme sind hybrid und benutzen zentrale Server für bestimmte Dienste, z.B. als Einstiegspunkte in das Netzwerk

- gut für neue, innovative Anwendungen

3.4.8 Publish/Subscribe-Architektur

Publish/Subscribe-Stil und -Muster

- Aufrufe werden nicht direkt unter den Kommunikationsteilnehmern versendet, sondern Ereignisse werden durch einen Vermittler weitergeleitet
- eine Reihe von Klienten muss über Laufzeitereignisse informiert werden
- Ereigniskonsumenten können sich für bestimmte Ereignistypen zu registrieren
- entkoppelt die Produzenten und Konsumenten von Ereignissen

3.4.9 Kommunikations-Middleware

Kommunikations-Middleware

- Plattform,
  die Anwendungen Dienste für alle Aspekte der Verteilung anbietet, wie verteilte Aufrufe, effizienten Zugriff auf das Netzwerk, Transaktionen und viele andere
- Anwendungsgebiete für verteilte Systeme divers:
  - Internet-Systeme
  - Telekommunikationsnetzwerke
  - Business-to-Business-Anwendungen (B2B)
  - internationale Finanztransaktionen
  - Kollaboration von räumlich verteilten Partnern
  - ...

Kommunikations-Middleware

- übernehmen Kommunikationsaufgaben transparent für die Entwicklerin
- Komplexität und Heterogenität der darunter liegenden Plattformen verbergen

---

3.4.10 Komponentenplattformen

Komponentenplattformen im Enterprise-Umfeld

- Beispiele für Komponentenplattformen im Enterprise-Umfeld:
  - JEE, CCM und .NET
- Trennung von technischen Belangen und den fachlichen Belangen an ein Informationssystem
- technische Belange im Enterprise-Umfeld: Verteilung, Sicherheit, Persistenz, Transaktionen, Nebenläufigkeit und Ressourcenmanagement.
- Black-Box-Wiederverwendung

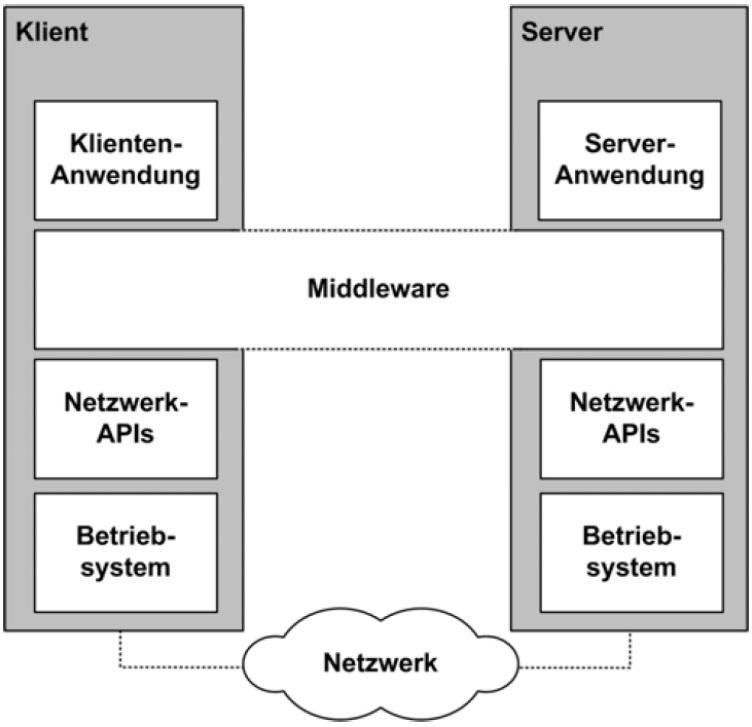


Bild 3.37: Schematische Darstellung einer Middleware-Architektur, aus [VZM+09]

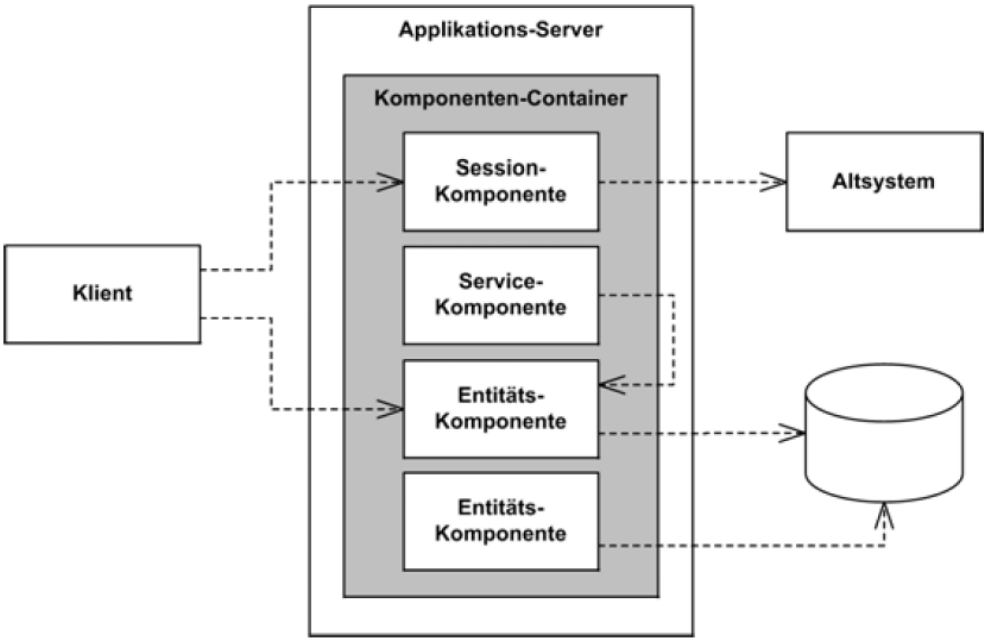


Bild 3.38: Komponenten-Container-Architektur, aus [VZM+09]

3.4.11 Plug-in-Architektur

- nach [LL06], S. 406

- System kann an dafür vorgesehenen Punkten erweitert werden, ohne es zu modifizieren
- Anwendung besteht aus:
  - **Kern** (auch *Host* = Wirt genannt),
  - der um sogenannte **Plug-ins** um neue Funktonen werweitert werden kann.
- Der Host definiert spezielle Schnittstelle, die sogenannten **Erweiterungspunkte**, auf die ein Plug-in Bezug nehmen kann

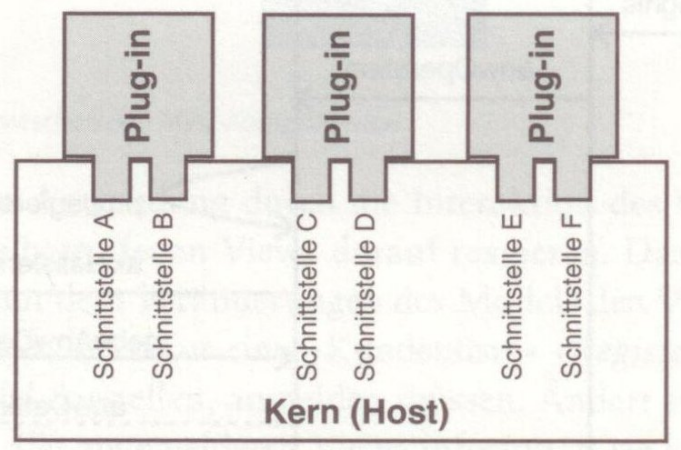


Bild 3.39: Schema einer Plug-in-Architektur, aus [LL06]

- Ein Plug-in muss gemäß den vom Host vorgegebenen technischen Konventionen realisert sein, der Code entsprechend abgelegt und das Plug-in beim Host registriert sein
- Bei Start des Hosts werden die aktuell vorhandenen Plug-ins identifiziert und entweder sofort oder nach Bedarf geladen.
- Ein Host kann selbst ein Plug-in sein.
- Beispiele für Architekturen für Plug-ins:
  - Eclipse (auf der Basis von OSGi)
  - Werkzeuge der Mozilla-Suite
  - Maven

Maven im Überblick

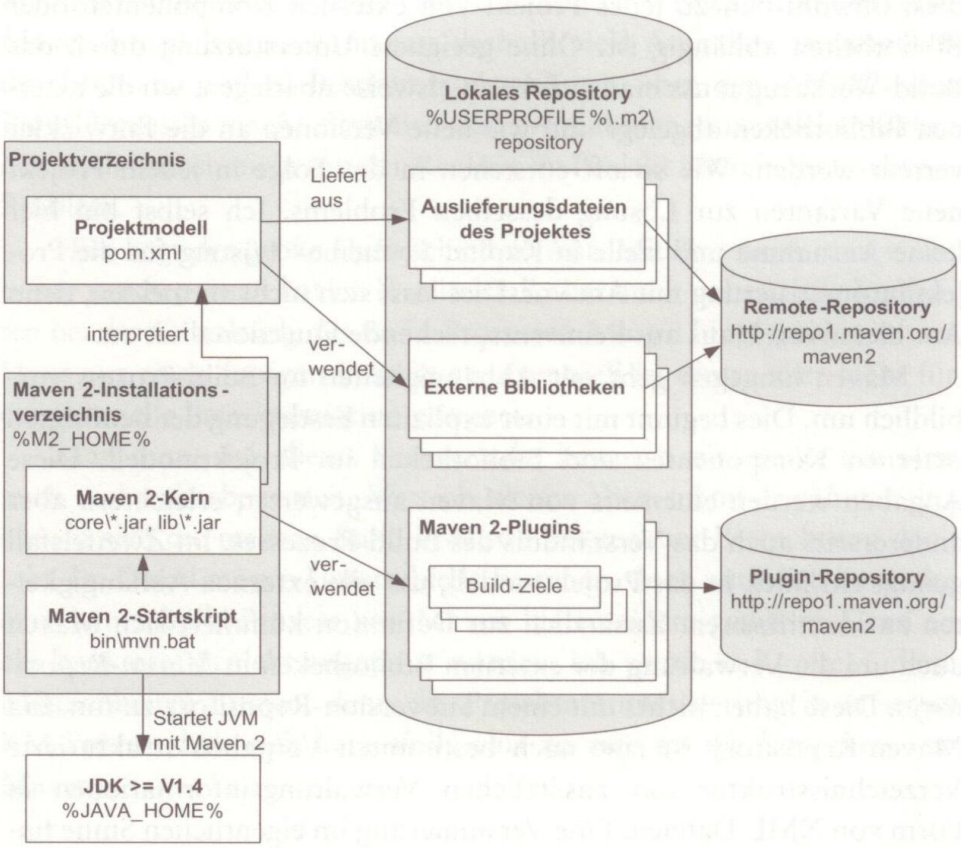


Bild 3.40: Maven 2 im Überblick, aus [Pop08]

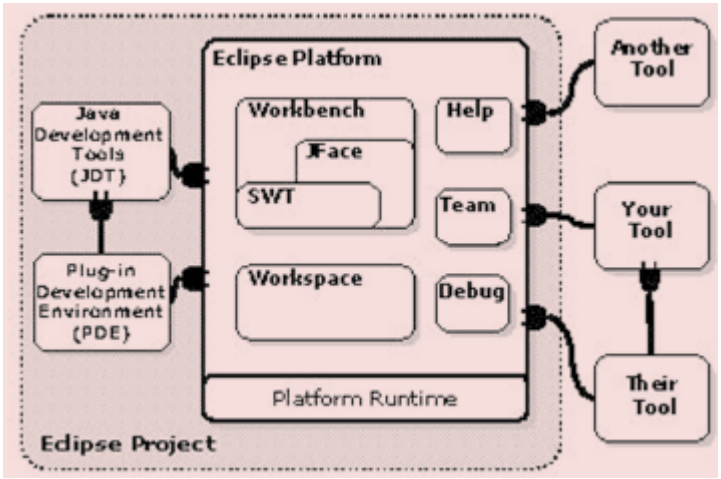


Bild 3.41: aus [MV03]

Eclipse

Zeus

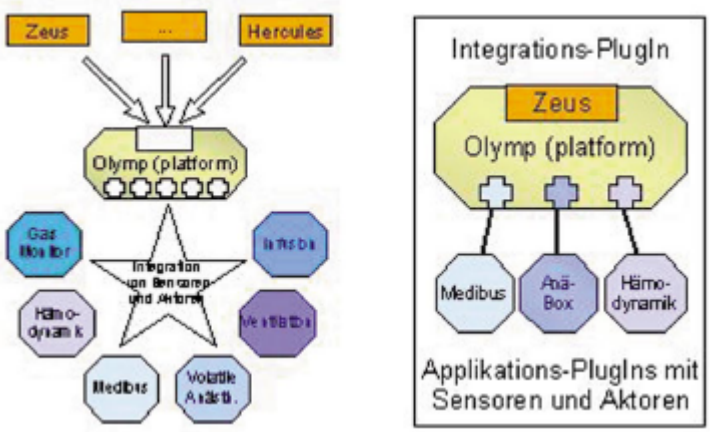


Bild 3.42: aus [MV03]

Dem Anästhesie-Arbeitsplatz Zeus liegt die Plug-In-Architektur Olymp zugrunde. Auf verfügbaren und zukünftigen Applikations-Plug-Ins lassen sich weitere Arbeitsplätze nach Kundenbedürfnissen integrieren

---

Plugin Vertrag

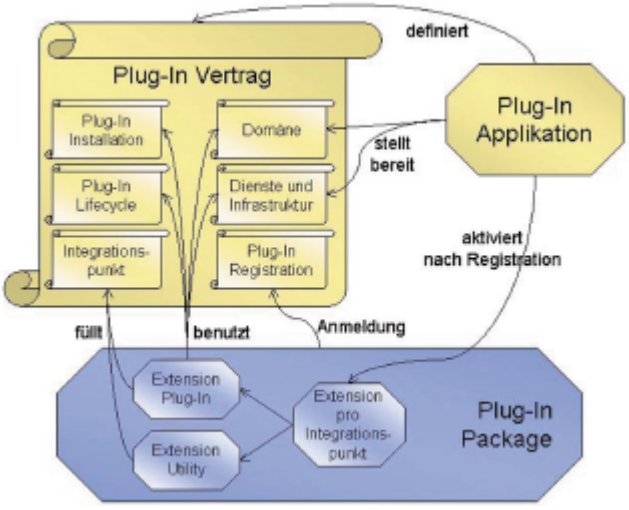


Bild 3.43: aus [MV03]

Der Plug-In-Vertrag gibt die Integrationspunkte, die Domäne und technische Schnittstellen für die Plug-Ins vor. Auf der anderen Seite verspricht er dem Plug-In die Implementierung einer Reihe von Diensten und eine funktionierende Infrastruktur.

3.4.12 Serviceorientierte Architekturen

- Serviceorientierte Architekturen sind eine Basisarchitektur, welche die fachlich funktionalen Schnittstellen von Software-Bausteinen als wiederverwendbare, verteilte, lose gekoppelte und standardisiert zugreifbare Dienste (englisch: Services) repräsentiert.

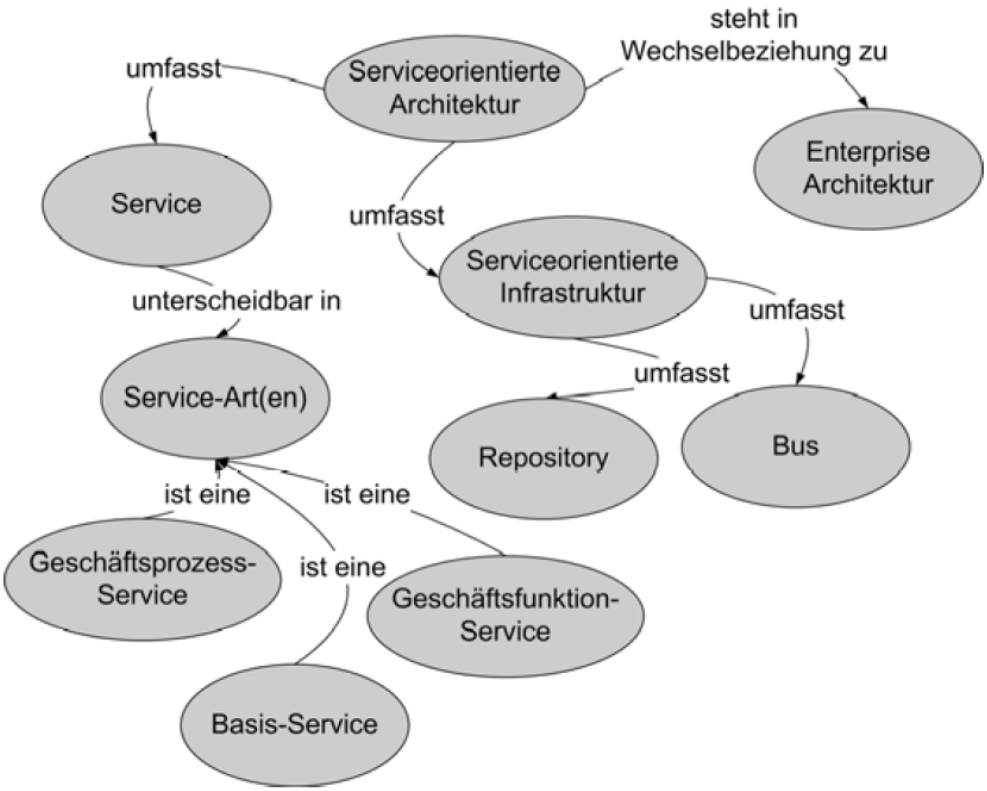


Bild 3.44: Konzept-Überblick zu serviceorientierten Architekturen, aus [VZM+09]

Eigenschaften von Services

- Services sind generell **grober granular als Komponentenschnittstellen** und hinsichtlich ihrer Geschäftsrelevanz stärker strukturiert als Komponenten.

- Services **kommunizieren technologieneutral und standardisiert** mit synchronen oder asynchronen Nachrichten.

- Services erlauben oft die **anonyme Nutzung**. Das heißt, man weiß nicht, wer den Service verwendet. Der Service funktioniert so, als ob er seinen Klienten nicht kennen würde. Mit anderen Worten: Der Klient und der Service sind lose gekoppelt.

- Services sind in gewissen Grenzen **selbstbeschreibend**. Selbstbeschreibung von Services ist häufig realisiert auf der Basis von Metadaten, deren Eigenschaften durch einen Lookup-Service ermittelt werden.
- Services sind im Idealfall **idempotent, zustandsfrei und transaktional** abgeschlossen. Als idempotent bezeichnet man generell Arbeitsgänge, die immer zu den gleichen Ergebnissen führen, unabhängig davon, wie oft sie mit den gleichen Daten wiederholt werden. Mathematisch knapper könnte man auch sagen: $a^n = a, n > 0$.
- Services bestehen aus der **Service-Schnittstelle und der Service-Implementierung**. Die Service-Schnittstelle besitzt Vertragscharakter (Design-by-Contract) und bindet Service-Konsumenten an Service-Anbieter (Prinzip Lose Kopplung). Die Service-Implementierung ist nicht Teil des Vertrags und unter Einhaltung der Schnittstellenzusagen austauschbar.

SOA Schlüsselabstraktionen

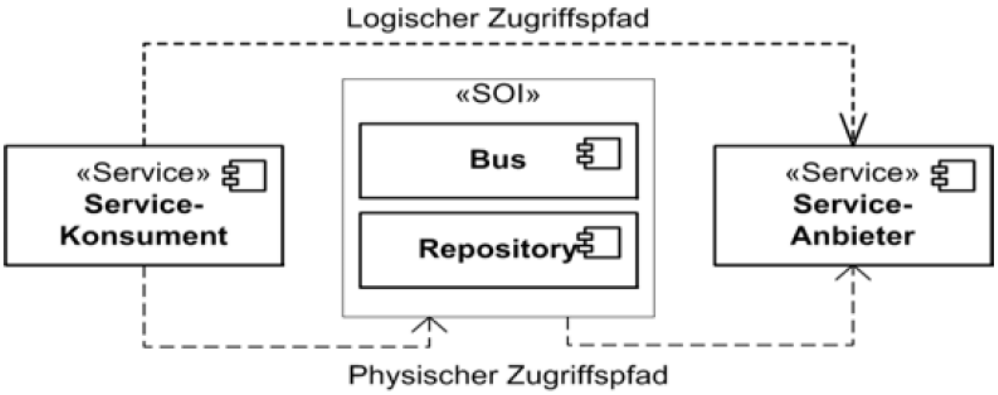


Bild 3.45: Schlüsselabstraktionen einer serviceorientierten Architektur, aus [VZM+09]

- SOI serviceorientierte Infrastruktur
- ESB Enterprise Service Bus (eine SOI)

Vertikale Gliederung einer SOA

3.4.13 Sicherheitsarchitekturen

Sicherheit als verteilter Aspekt

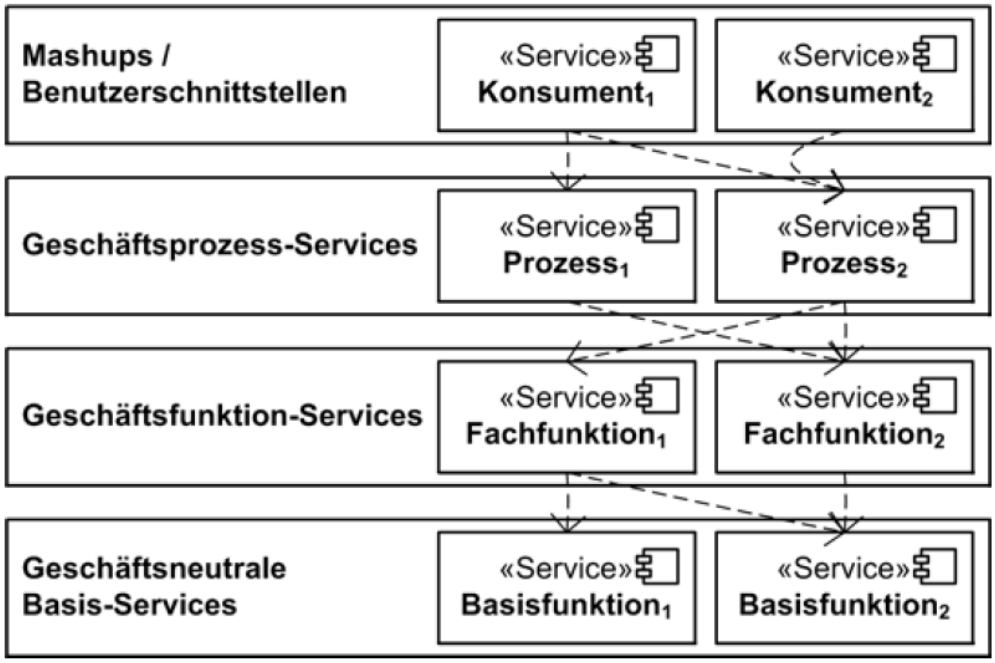


Bild 3.46: Vertikale Gliederung einer SOA, aus [VZM+09]

- durchdringender Belang (englisch: crosscutting concern)
- hochgradig verteilter Aspekt
- über viele Systembausteine hinweg realisiert
- Beispiele solcher Systembausteine:
  - Firewalls,
  - Public-Key-Infrastrukturen (PKI),
  - Reverse Proxies,
  - Web-Access-Management-Lösungen (WAM),
  - Verzeichnisdienste, z.B. LDAP

Begriff Sicherheitsarchitektur

bezieht sich auf

- Eine zu schützende bzw. sichernde Anwendung
- Weitere Systembausteine der unterliegenden Sicherheitsinfrastruktur

---

Schlüsselkonzepte

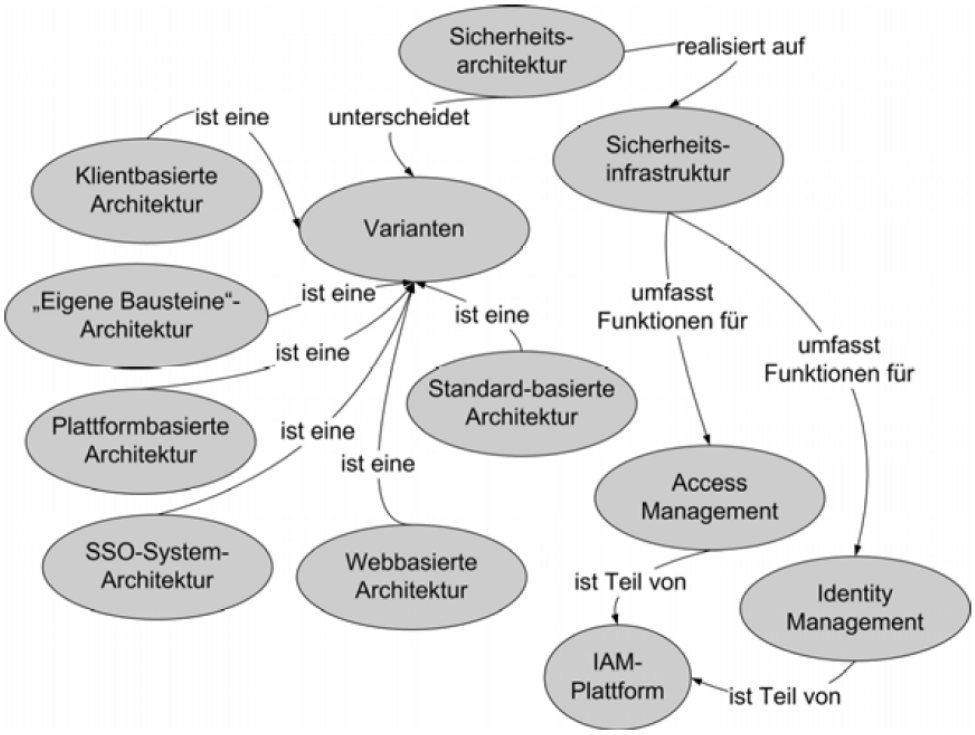


Bild 3.47: Konzept-Überblick zu Sicherheitsarchitekturen, aus [VZM+09]

Voraussetzungen für Sicherheitsarchitekturen

technische Voraussetzungen

- Existenz von Sicherheitssystemen auf allen Netzwerkebenen, einschließlich Betriebssysteme.

  Systeme zur

- Benutzer- und Identitätsverwaltung
- Anmeldungsüberprüfung (Authentifizierung)
- Rechteverwaltung und -durchsetzung (Autorisierung)
- geschützten Informationsübermittlung (englisch: privacy)
- Gewährleistung von Unbestreitbarkeit (englisch: non repudiation)
- Betriebsüberwachung und Angriffserkennung (englisch: threat detection)

---

organisatorische Voraussetzungen

- **Informations- und Schutzklasseneinteilung** im Unternehmen, Schutzbedürfnis für Informationen, Daten und Dokumente formulieren
- **Richtlinien** zum Umgang mit schützenswerten Gütern und Informationen auf den Ebenen Abteilung, Unternehmen, Partner, Staat etc.
- **Organisationen und Institutionen** im Unternehmen, die für die Planung, Umsetzung sowie Durchsetzung verantwortlich sind

---

Identity und Access Management (IAM)

- Dachbegriff für Gesamtheit der oben genannten Fähigkeiten
- Synonyme: IAM-System oder IAM-Plattform
- Sicherheitsarchitektur einer Anwendung oft auf der Basis der Sicherheitssysteme einer IAM-Plattform entworfen und implementiert
- üblicherweise zentral und unternehmensweise implementiert

Digitale Identität

- Repräsentation eines Subjekts, also einer Person, eines Prozesses, Dienstes oder Anwendung, mittels
  - eindeutiger Kennung (z. B. künstlicher oder abgeleiteter Schlüssel),
  - eines oder mehrerer Berechtigungsnachweise (z. B. Benutzername und Passwort) und
  - weiterer Attribute (z. B. E-Mail-Adresse, Alter, Position)

Access Management (AM)

Access-Management-Systeme (AM-System) stellen

- Funktionen zur Identitätsüberprüfung (Authentifizierung) sowie
- Zugriffs- und Berechtigungskontrolle (Autorisierung) zur Verfügung

---

Leistungsmerkmale und Funktionen

von Sicherheitsarchitekturen

Privatsphäre (englisch: privacy)

- Nachrichten, die zwischen zwei Systembausteinen ausgetauscht werden, können auf dem Kommunikationspfad selber nicht gelesen, bzw. verstanden werden
- erreicht durch Verschlüsselung und Entschlüsselung

Integrität

- Nachrichten sind auf dem Kommunikationspfad nicht änderbar
- typischerweise realisiert über Hashing-Verfahren

Authentifizierung (englisch: authentication)

- Vorgang der Identitätsüberprüfung
- Benutzerin beweist mithilfe von Berechtigungsnachweisen gegenüber der Authentifizierungsfunktion, dass sie die ist, die sie vorgibt zu sein
- verbreiteter Typ von Berechtigungsnachweisen: Kombination von Benutzername und Passwort

Autorisierung (englisch: authorization)

- Vorgang der Zugriffskontrolle durch einen entsprechenden Systembaustein
- Voraussetzung: Authentifizierung

Unleugbarkeit, bzw. Unbestreitbarkeit (englisch: non-repudiation)

- Fähigkeit, sicherheitsbezogene Ereignisse zweifelsfrei zu belegen
- zum Beispiel vor Gericht
- realisiert z.B. mit Journalen und/oder digitale Signaturen

Schutz vor Zerstörung und des Betriebs (englisch: intrusion protection)

- Menge aller Maßnahmen, die betriebliche Integrität und Vitalität gewährleisten
- z.B. Denial-of-Service-(DoS)-Attacken eine wirksame Verteidigung entgegensetzen
- z.B. Entdecken von Sicherheitslücken und -vorfällen

Sicherheitsarchitekturen auf Basis eigener Sicherheitsbausteine

- alle wesentlichen Sicherheitsfunktionen selbst implementiert
- Authentifizierungsfunktion z.B. mit Werteüberprüfung auf der eigenen Benutzerdatenbank
- Autorisierungsfunktion z.B. mit Rolleninformationen aus einer eigenen Privilegdatenbank
- Privatsphäre z.B.: Zugriff zu den Datensystemen kontrolliert und Datentransportwege selber verschlüsselt

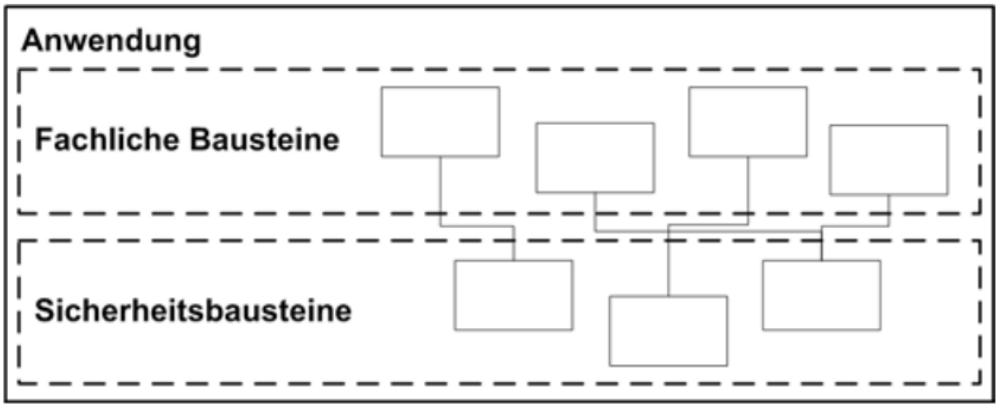


Bild 3.48: Sicherheitsarchitekturen auf Basis eigener Sicherheitsbausteine, aus [VZM+09]

- sehr verbreitet, da oft nicht davon ausgangen werden kann, in allen Zielumgebungen die notwendigen Systembausteine standardisiert und zentral angeboten vorzufinden
- Nachteil: schlecht mit bestehenden IAM-Systemen integrierbar
- Nachteil: selbst implementierte Verschlüsselungsverfahren erfüllen nur selten moderne Anforderungen

---

auf der Basis von Standarddiensten

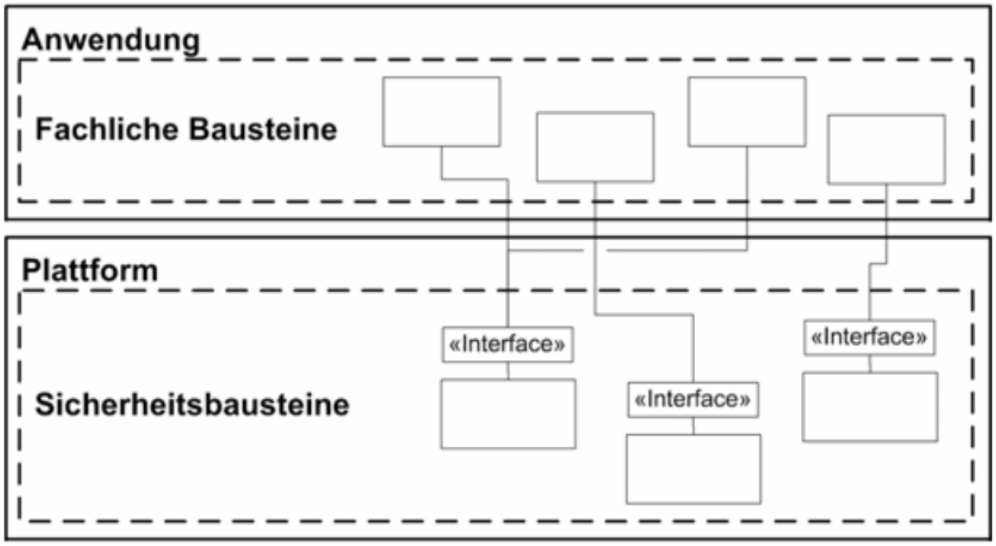


Bild 3.49: Sicherheitsarchitekturen auf der Basis von Standarddiensten, aus [VZM+09]

- Beispiel Standarddienst: LDAP-Verzeichnisdienste

- Beispiel Standarddienst: Public-Key-Infrastrukturen (PKI)
- Vorteil: sicherheitsbezogene Systembausteine können ausgewechselt werden, ohne Anpassung der Anwendung
- Nachteil: Vernetzung von sicherheitsrelevanten Informationen auf der Ebene der Anwendung
- Beispiel: Benutzerinformationen aus LDAP-Verzeichnis, Zugriffsrechte aus Privilegiendatenbank

---

auf der Basis von Komponentenplattformen

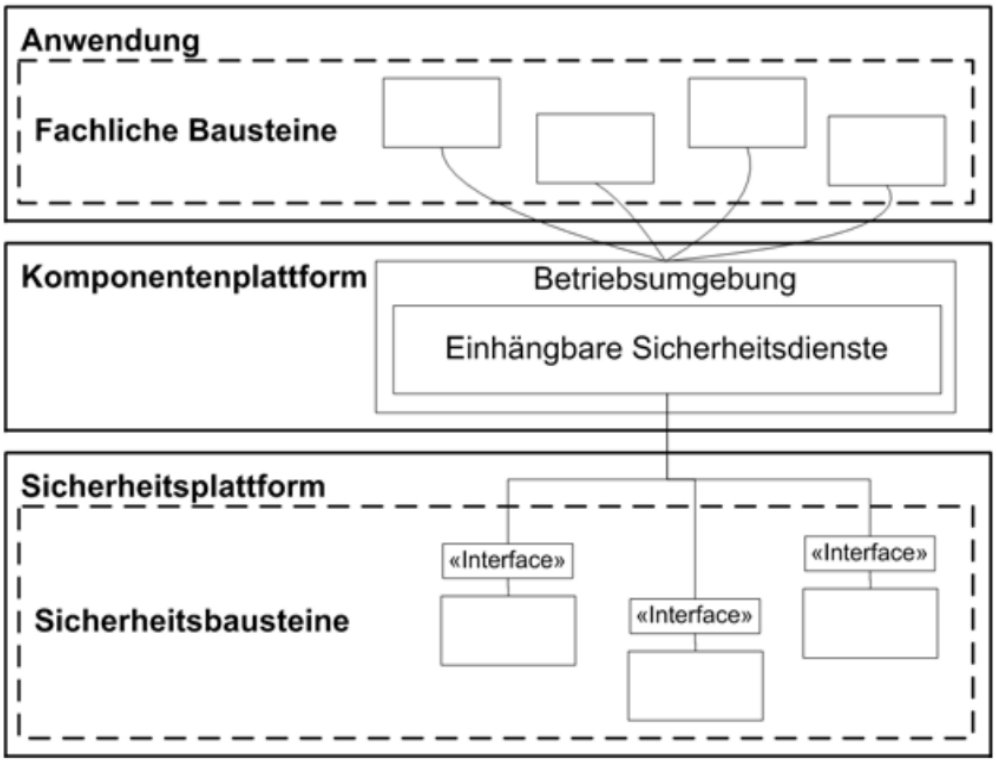


Bild 3.50: Sicherheitsarchitekturen auf der Basis von Komponentenplattformen, aus [VZM+09]

- Sicherheitsfunktionalität auf einem höheren Abstraktions- und Integrationsniveau gebündelt
- z.B. Vernetzung Authentifizierungs- mit Autorisierungsfunktionalität
- Beispiel: JEE Java Enterprise Edition, u.a. Java Authentication and Authorization Service (JAAS)
- Beispiel: .NET-Plattform
- Beispiel: CORBA

J2SE 6 Plattform

Klientenseitige Sicherheitsarchitektur

weitere Aspekte von Sicherheitsarchitekturen

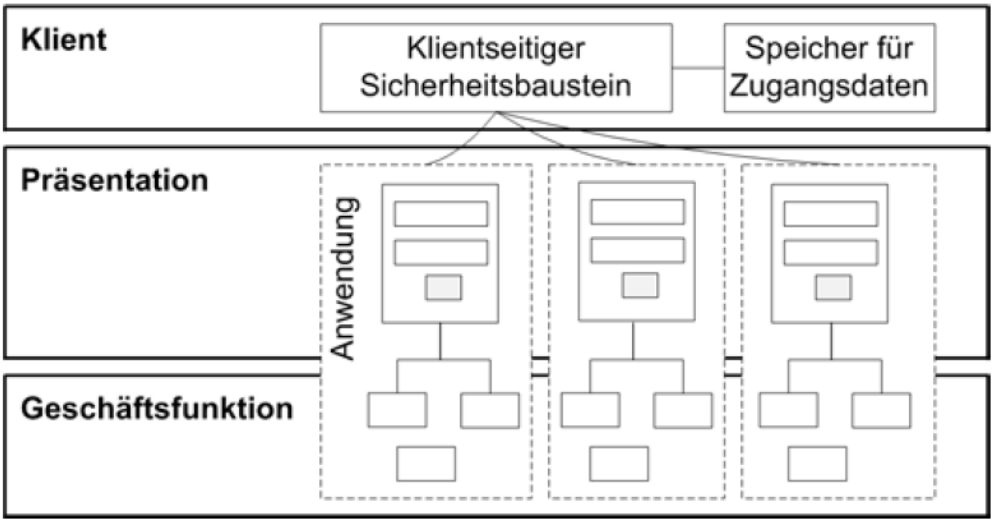


Bild 3.51: Klientenseitige Sicherheitsarchitekturen, aus [VZM+09]

- Software-Baustein, der auf dem Endgerät der Benutzerin (z. B. Laptop) installiert ist und die Zugangsdaten zu allen registrierten Anwendungen verwaltet
- non-invasiv
- simulieren die Benutzerin gegenüber allen registrierten Anwendungen

Web-zentrisch

- Reverse-Proxy-Architekturen
- beschränkt auf Web-Anwendungen.
- Web-Baustein (Reverse Proxy) schützt statische und dynamische Web-Anwendungen
- Authentifizierung, Autorisierung sowie die Sicherheitssitzung wandern auf die Systemgrenzen (Perimeter) und müssen nicht mehr in der Anwendung selbst programmiert werden müssen.

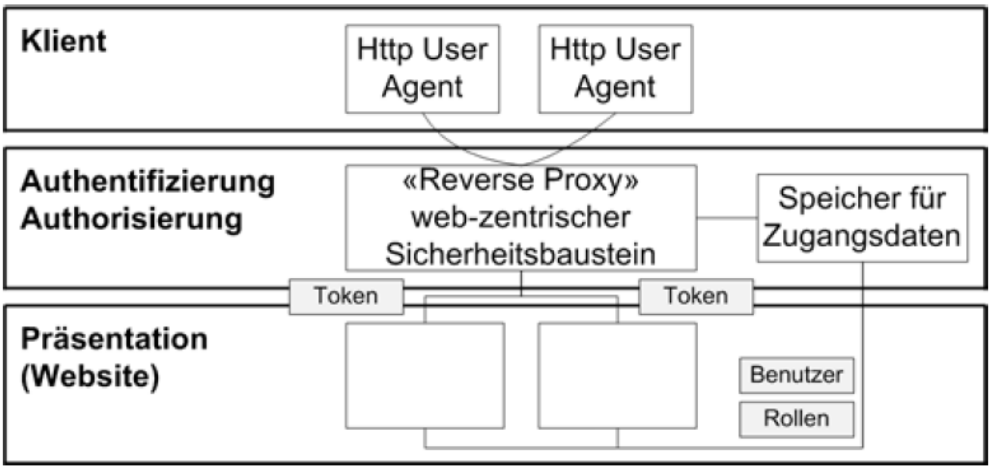


Bild 3.52: Web-zentrische Sicherheitsarchitekturen, aus [VZM+09]

Single-Sign-On

Einmalanmeldung, siehe Bild 3.53

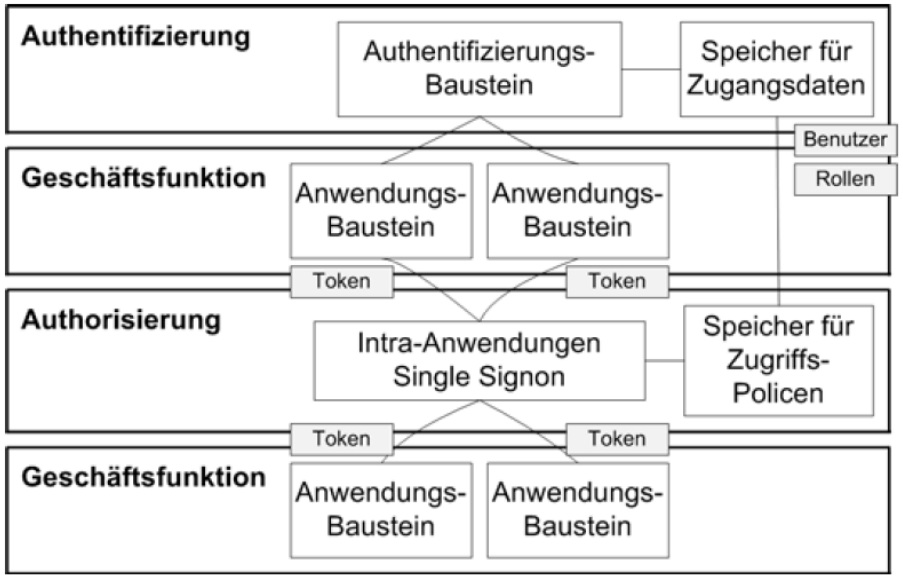


Bild 3.53: Single-Sign-On-Architekturen, aus [VZM+09]

- Systembausteine auf der Sicherheitsinfrastrukturebene
- übernehmen Authentifizierung, Sitzungsverwaltung sowie Tokenverwaltung und -verifizierung
- nach erfolgreicher Anmeldung wird eine Sicherheitssitzung mit einem Schlüssem (Token) angelegt
- tiefe Integration mit entsprechenden Anwendungen

Single User Sign-On

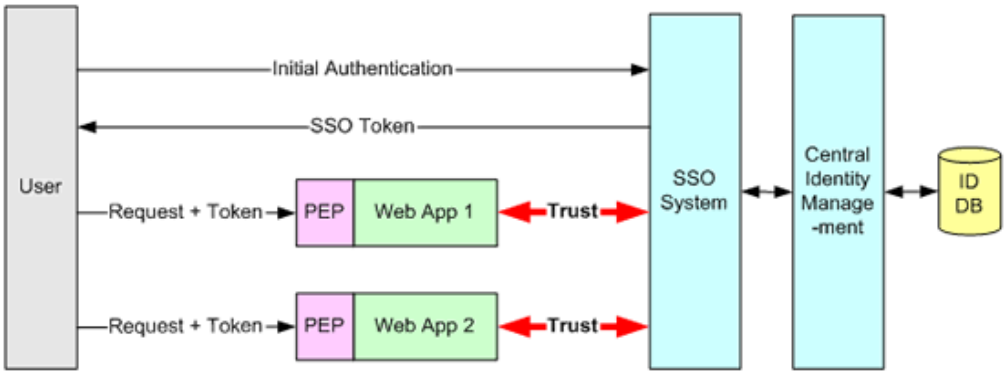


Bild 3.54: Single User Sign-On To Multiple Services, von `http://it-republik.de/jaxenter/artikel/Single-Sign-On-Systeme-1499.html`

	CAS	**OpenSSO**	**JOSSO**	**SiteMinder**
Architektur	zentrale Komponente	zentrale Komponente	zentrale Komponente	zentrale Komponente
Client PEP	Java EE, .NET, Perl, Acegi/Spring und andere	Java EE und andere	Java EE, PHP, .NET, Acegi/-Spring	Java EE, PHP, .NET, Acegi/-Spring, SAP
Identitäts-übertragung	Token im HTTP Header	Token im HTTP Header	Token im HTTP Header	Token im HTTP Header

Beispiele OpenSource Single-Sign-On Rahmenwerke, aus [Sam08]
CAS: `www.jasig.org/products/cas/`
OpenSSO: `https://opensso.dev.java.net/`
JOSSO: `www.josso.org`
SiteMinder: `www.ca.com`

Consumerization

Consumerization: Herausforderung für Sicherheitsarchitekturen

From Wikipedia, the free encyclopedia, 2011:

"Consumerization is a stable neologism that describes the trend for new information technology to emerge first in the consumer market and then spread into business organizations, resulting in the convergence of the IT and consumer electronics industries, and a shift in IT innovation from large businesses to the home.

For example, many people now find that their home based IT equipment and services are both more capable and less expensive than what is provided in their workplace.

The term, consumerization, was first popularized by Douglas Neal and John Taylor of CSC's Leading Edge Forum in 2001 and is one of the key drivers of the Web 2.0 and Enterprise 2.0 movements."

Beispiel: Geräte: PC, Laptop, Smart Phone, iPad,

Beispiel: Benutzungsoberfläche: mit wenigen Mausklicks eine Anwendung bedienen
Beispiel: Software: Kalender, Adressbuch, Aufgabenliste "mischen"

3.4.14 Zusammenfassung

- Es gibt **Basisarchitekturen**, die in vielen Systemen zum Einsatz kommen.
- **Schichtenarchitekturen** dienen der Strukturierung von Architekturen, indem sie Bausteine in Schichten anordnen und jede Schicht der darüber liegenden Schicht Dienste durch Schnittstellen bereitstellt.
- **Datenflussarchitekturen** strukturieren eine Architektur entlang der Datenflüsse und machen insbesondere dann Sinn, wenn eine komplexe Aufgabe in eine Reihe einfacher Aufgaben zerteilt und dann als Kombination unabhängiger Aufrufe dargestellt werden kann.
- In einer Shared-**Repository-Architektur** stellt ein Baustein des Systems einen zentralen Datenspeicher zur Verfügung.

- Eine grundlegende Frage ist die der **Zentralisierung gegenüber der Dezentralisierung**. Hier muss oft ein Kompromiss gefunden werden.
- Das klassische **Client-/Server-Modell** basiert auf einer 2-Tier-Architektur.
- **3-Tier-Architekturen** erweitern die 2-Tier-Architektur, indem sie eine Zwischenschicht zwischen Klient und Datenbankserver einführen.
- 2-Tier-Architekturen und 3-Tier-Architekturen sind Spezialfälle von **n-Tier-Architekturen**.
- Die Entscheidung zwischen einem **Rich Client** und einem **Thin Client** basiert auf der Frage, wie man die Funktionalität zwischen dem Klienten und dem Server aufteilt.
- **Peer-to-Peer** ist eine Basisarchitektur, die eine Reihe von gleichwertigen Peers zur (verteilten) Kommunikation nutzt.

- Eine **Publish-/Subscribe-Architektur** ist eine Basisarchitektur bei der Aufrufe nicht direkt unter den Kommunikationsteilnehmern versendet, sondern durch einen Vermittler weitergeleitet werden. Typischerweise wird in einer Publish-/Subscribe-Architektur über asynchrone Ereignisse kommuniziert.
- **PlugIn-Architektur**
- Die **Middleware** ist eine zentrale Technologie für viele verteilte Systeme. Sie ist eine Plattform, die Anwendungen Dienste für alle Aspekte der Verteilung anbietet, wie verteilte Aufrufe, effizienten Zugriff auf das Netzwerk, Transaktionen und viele andere.

- **Komponentenplattformen** basieren auf der Trennung von technischen Belangen und den fachlichen Anforderungen. Die technischen Belange werden automatisiert von einem Container übernommen. Beispiele für technische Belange im Enterprise-Umfeld sind Verteilung, Sicherheit, Persistenz, Transaktionen, Nebenläufigkeit und Ressourcenmanagement.

- **Serviceorientierte Architekturen** (SOA) sind eine Basisarchitektur, welche die fachlich funktionalen Schnittstellen von Software-Bausteinen als wiederverwendbare, verteilte, lose gekoppelte und standardisiert zugreifbare Services repräsentiert.
- **Sicherheitsarchitekturen** beziehen sich auf eine zu schützende Anwendung und eine darunter liegende Sicherheitsinfrastruktur.
