

Verteilte Systeme

Prof. Dr. Martin Becke

CaDS - HAW Hamburg

Version 0.9

Inhalt

- 1 Architekturparadigmen
 - Schichtenarchitektur
 - Single Node (Application0)
 - Overlay Network
 - Middleware Architektur
 - Client-Server-Architektur
 - Event-Driven Architektur
 - Microservices-Architektur
 - Peer-to-Peer-Architektur
 - Hexagonal Onion Architektur
 - Time vs Event Triggered

Architekturparadigmen

Überblick

- ▶ Schichtenarchitektur
- ▶ Client-Server-Architektur
- ▶ Service-Orientierte Architektur (SOA)
- ▶ Ereignisgesteuerte Architektur (Event-Driven)
- ▶ Microservices-Architektur
- ▶ Peer-to-Peer-Architektur (P2P)

Architekturparadigmen

Grundelement node

- ▶ Autonomes Computerelement
- ▶ Hardware-Systeme oder Software-Prozesse
- ▶ Teilen keinen gemeinsamen Speicher
- ▶ Kommunikation über Nachrichten

Schichtenarchitektur

Definition

- ▶ Hierarchische Struktur
- ▶ Definiert Abstraktionsebenen mit allgemeinen Schnittstellen
- ▶ Jede Schicht repräsentiert ein Set von Funktionen
- ▶ Jede Schicht hat eine oder mehrere Interpretationen
- ▶ Jede Interpretation implementiert Protokolle

Schichtenarchitektur

Bedeutung

- ▶ Komplexe Systeme in kleinere, einfachere Teile zerlegen
- ▶ Trennung von Verantwortlichkeiten
- ▶ Schichten können unabhängig entwickelt, getestet und gewartet werden
- ▶ Jede Schicht kann als Komponente verstanden werden
- ▶ Interoperabilität und Kompatibilität steigt
- ▶ Wiederverwendbarkeit steigt
- ▶ Chance und Risiko für Sicherheit
- ▶ Historisch wichtig und immer noch hohen Einfluss

Schichtenarchitektur

Beispiele

- ▶ ISO OSI Referenzmodell
- ▶ DECnet
- ▶ Cloud Architekturen
- ▶ Micro Service Architekturen

Layer

Definition

- ▶ Logische Gruppierung von Funktionen
- ▶ Schichten werden in der Regel vertikal angeordnet
- ▶ Jede Schicht erfüllt eine spezifische Funktion

Layer

Beispiel

- ▶ Presentation Layer
- ▶ Business Logic Layer
- ▶ Data Access Layer

Single Node (Application0)

Overlay Network

Middleware Architektur

Client-Server-Architektur

Event-Driven Architektur

Microservices-Architektur

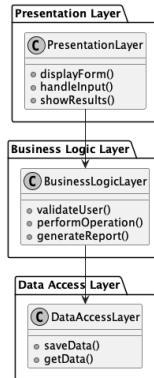
Peer-to-Peer-Architektur

Hexagonal Onion Architektur

Time vs Event Triggered

Layer

Als Komponenten



Tier

Definition

- ▶ Physische oder logische Aufteilung
- ▶ Jedes Tier hat normalerweise eine spezifische Funktion
- ▶ Normalerweise durch eine Netzwerkverbindung miteinander verbunden

Tier

Beispiel

- ▶ Präsentations-Tier
- ▶ Anwendungs-Tier
- ▶ Datenbank-Tier

Tier

Als Komponenten

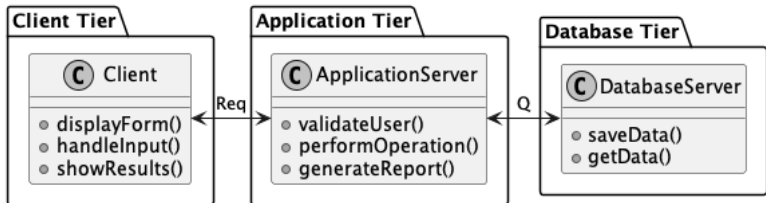


Figure – Einfaches Tier Modell

Single Node

Definition

- ▶ Architekturtyp
- ▶ Anwendungen werden auf einem einzelnen Knoten ausgeführt
- ▶ Keine Verteilung der Funktionen
- ▶ Knoten kann ein physischer Server oder eine virtuelle Maschine sein
- ▶ Knoten kann von verschiedenen Systemen sammeln und verarbeiten

Single Node

Vorteile

- ▶ Entwicklung und Testen
- ▶ Kleine Anwendungen
- ▶ Datenintensive Anwendungen

Single Node

Umsetzung und Beispiele

- ▶ Container-Technologien
- ▶ Serverless-Computing (AWS Lambda)
- ▶ Netflix : Netflix verwendet das Single-Node-Pattern, um die Datenverarbeitung in ihren AWS-Cloud-Instanzen
- ▶ Airbnb : Airbnb verwendet das Single-Node-Pattern, um ihre Webanwendung (nicht die Daten) in einem einzigen Knoten zu hosten

Overlay

Idee

- ▶ Logische Verbindung muss keiner physikalischen Verbindung entsprechen
- ▶ Genutzte Netzwerkstruktur besteht häufig auf der obersten Schicht einer physikalischen Netzwerkstruktur
- ▶ Zusätzliche Abstraktion und Organisation
- ▶ Zusätzliche Komplexität und Overhead

Overlay

Beispiel P2P

- ▶ In einem P2P-Netzwerk sind die Knoten gleichberechtigte Teilnehmer
- ▶ Teil oder voll vermascht
- ▶ Bekanntes Beispiel für ein P2P-Overlay-Netzwerk ist BitTorrent

Overlay

P2P - Strukturierte Netzwerke

- ▶ Geordnete Struktur
- ▶ Verwenden meist deterministische Verfahren oder Algorithmen
- ▶ Routing effizienter und vorhersagbarer
- ▶ Ein Beispiel für ein strukturiertes Routing-Verfahren ist DHT
- ▶ Suche kann in logarithmischer Zeit ($O(\log N)$) durchgeführt werden

Overlay

P2P - Un-Strukturierte Netzwerke

- ▶ Keine Struktur
- ▶ Verwendung auf Basis von Heuristiken und Zufall
- ▶ Routing kaum effizient zu gestalten
- ▶ Typisch ist hohe Netzwerklast
- ▶ Flooding- oder Random-Walk-Verfahren für die Suche
- ▶ Schädliche Teilnehmer ein besonderes Problem

Middleware Architektur

Idee

- ▶ Kapselung der Aufgabe
- ▶ Eigene Schicht für die Herausforderung der verteilten Systemen
- ▶ Schicht wird als « Middleware- »Schicht bezeichnet
- ▶ Bietet im besten Fall die Schnittstelle des kohärenten Systems

Middleware Architektur

Definition nach Tanenbaum

- ▶ Bereitstellung eines breiten Angebotes für die Kommunikation
- ▶ Dem (Un-)Marshaling Prozess
- ▶ Protokolle zur Namensauflösung
- ▶ Sicherheitsprotokolle
- ▶ Mechanismen zur Steigerung der Skalierung

Middleware Architektur

Erweiterte Anforderungen

- ▶ Mehrere Sprachen
- ▶ Mehreren Betriebssystemen und Hardwaretypen
- ▶ Mehrere Netzwerkprotokolle zur Einbindung von Internetstrukturen

Middleware Architektur

Kommunikation

- ▶ Point to point
- ▶ Point to multipoint
- ▶ Publish/subscribe
- ▶ Client/Server, Request/Reply
- ▶ Mobile code
- ▶ Virtual shared

Middleware Architektur

Message Oriented Middleware (MOM)

- ▶ Direkte Kommunikation zwischen Anwendungen (Message Passing)
- ▶ Indirekte Kommunikation über eine Warteschlange (Message Queueing)
- ▶ Herausgeber stellt dem Abonnenten Nachrichten zur Verfügung (Publish & Subscribe)

Client-Server-Architektur

Modell

- ▶ In dieser Architektur gibt es zwei Hauptkomponenten
- ▶ Client - nutzt (Consumer)
- ▶ Server - bietet an (Provider)
- ▶ Eine spezielle « 2-Tier Architektur »
- ▶ Front-End-Client und einem Back-End-Server

Client-Server-Architektur

Beispiele

- ▶ Webanwendungen
- ▶ Datenbank Anwendungen

Client-Server-Architektur

Kaskadenartige Kommunikation

- ▶ n-Tier-Architektur sind Tiers sowohl Consumer- als auch Provider
- ▶ Auswirkung auf Latenz, Komplexität, Fehleranfälligkeit, Skalierbarkeit
- ▶ Einführung von Optimierungsmechanismen, wie Caching, Lastverteilung, asynchrone Kommunikation

Client-Server-Architektur

SSP - Stub/Skeleton Chains

- ▶ Eine Implementierungsform von Interface-Definitionen zur Kommunikation
- ▶ Methodenaufrufe von einer Schicht zur anderen
- ▶ Stub ist ein Platzhalter für eine entfernte Methode (Lokal oder Remote)
- ▶ SSP Stub/Skeleton Chains sind eine Sequenz von Stubs und Skeletons
- ▶ Gemeinsame Sprache zwischen verschiedenen Schichten oder Tiers

Event-Driven Architecture

Idee

- ▶ Kommunikation getrieben über Events oder Nachrichten
- ▶ Komponenten reagieren auf Events oder Nachrichten
- ▶ Keine direkte Kommunikation (Vergleich Eventbus)

Event-Driven Architecture

Beispiele

- ▶ Aktienhandelssystem
- ▶ IoT-Sensornetzwerk
- ▶ E-Commerce-Plattform

Event-Driven Architecture

Aufbau

- ▶ Event-Producer
- ▶ Event-Channel
- ▶ Event-Consumer

Event-Driven Architecture

Erweiterung um Lambda Architecture

- ▶ Häufig im Kontext von Big-Data- und Echtzeitanalysen
- ▶ Datenverarbeitungsarchitekturmuster
- ▶ Latenzarme und fehlertolerante Analyse- und Verarbeitungssysteme

Lambda Architecture

Aufbau

- ▶ Batch-Layer
- ▶ Speed-Layer
- ▶ Serving-Layer

Microservices-Architektur

Idee

- ▶ Anwendung ist Sammlung kleiner, unabhängiger und modularer Dienste
- ▶ Dienste sind lose gekoppelt, können unabhängig voneinander entwickelt, bereitgestellt und skaliert werden
- ▶ Entwicklerteam je Dienst
- ▶ Erhöhte Komplexität, Kommunikations-Overhead, Sicherheits- und Authentifizierungsfragen

Microservices-Architektur

Beispiele

- ▶ Online-Shop
- ▶ Content-Management-System
- ▶ Bankanwendung

Peer-to-Peer-Architektur

Idee

- ▶ Einzelne Knoten (Peers) kommunizieren direkt miteinander
- ▶ Knoten sowohl Clients als auch Server
- ▶ Teilen Ressourcen und tragen gemeinsam zur Leistung und Skalierbarkeit des Netzwerks bei

Peer-to-Peer-Architektur

Beispiele

- ▶ Dateifreigabe
- ▶ Kommunikation und Messaging
- ▶ Blockchain und Kryptowährungen
- ▶ Beispiel wird nochmals Overlay und P2P Architektur deutlich machen

Hexagonal Onion Architektur

Idee

- ▶ Kombination aus zwei bekannten Architekturmustern :
Hexagonale Architektur und Zwiebelarchitektur
- ▶ Die Kernlogik des Systems ist unabhängig von externen Einflüssen
- ▶ Ports und Adaptern dienen als Schnittstelle zwischen der Kernlogik und externen Anliegen

Hexagonal Onion Architecture

Typische Schichten

- ▶ Domain
- ▶ Application
- ▶ Ports
- ▶ Adapters

Hexagonal Onion Architecture

Beispiel mit Microservices

- ▶ Domain : Verwalten von Produkten und Dienstleister
- ▶ Application : Prozesse für Kauf, Verkauf, Retoure, Wartung, etc
- ▶ Microservice : Online Bestellungen-Service mit Abhängigkeit Bezahlservice
- ▶ Microservice : Online Produkt-Service
- ▶ Microservice : Online Zahlungs-Service
- ▶ Microservice : Online Versand-Service

Time Triggered vs Event Triggered

Diskussion

- ▶ Synchronisation
- ▶ Determinismus
- ▶ Koordination
- ▶ Latenz
- ▶ Ressourcen