Grundlagen (Architektur)

PROF. ANDREAS HARTMANN – VERTEILTE ANWENDUNGEN

Quellenangaben

Die Inhalte sind angelehnt an - und Zitate stammen, wenn nicht anders gekennzeichnet, aus:

- [SP17]Schönbächler, M.; Pfister, C.: IT-Architektur. Grundlagen, Konzepte und Umsetzung. epubli, Berlin, 2017.

Softwarearchitektur

Strukturierung einer Anwendung

Horizontale und vertikale Aspekte

Softwarearchitektur einer verteilten Anwendung überschneidet sich mit IT-Architektur einer Anwendungslandschaft

→Mehr im Modul Software-Engineering

Horizontal (Schichtenmodell):

- View Layer
- Controller Layer
- Model Layer
- Service Layer
- Business Layer
- Persistence & Integration Layer

Vertikal (schichtübergreifende Funktionalitäten)

- Transaktionsbehandlung
- Systemmeldungen
- Protokollierung
- Internationalisierung
- Sicherheit
- Mandantenfähigkeit
- Frontend-Backend
- Verteilungsanforderungen

Wiederholung: Schichten

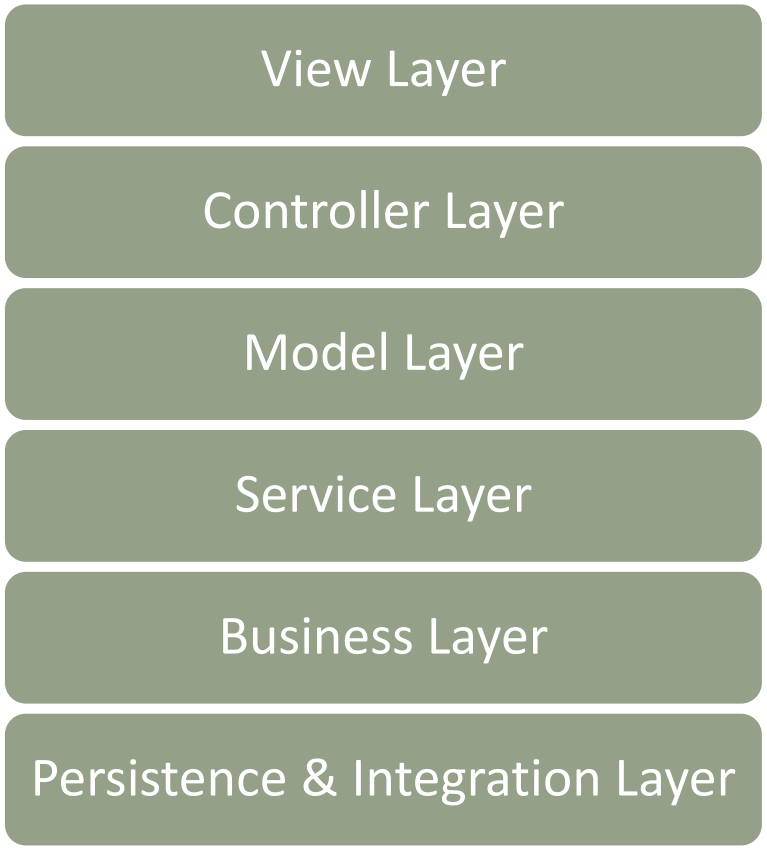
Untere Schichten (Layer) dürfen keine Annahmen über darüberliegende Schichten treffen (Gewährleistung der Entkopplung)

Operationen werden immer von überliegenden Schichten initiiert – niemals von unten nach oben

Untere Schichten können durch obere Schichten übersprungen werden, wenn dadurch nicht-funktionale Anforderungen erfüllt werden

Die technische Implementierung der Schichten hängt vom Anwendungsfall ab

Horizontale Aspekte



View Layer

Visuelle Aufbereitung und Anzeige der Daten (Informationen)

Keine Fachlogik, lediglich Validierung

Benutzereingaben entgegennehmen

Enthält datenmäßigen Zustand aktueller Benutzereingaben

View-Elemente registrieren Events anderer View-Elemente

Verarbeitet alle Events und sendet diese an den Controller

Stellt dem Controller Set/Get-Methoden zur Steuerung der View-Elemente bereit

Controller Layer

Behandlung der durch den Benutzer auf der View ausgelösten Events

Implementiert die Logik der Dialogsteuerung

Steuert die View vollständig und aktualisiert Zustände

Empfängt keine Events direkt vom Model

Instanziiert sowohl Model als auch View

Model Layer

Beinhaltet Daten und ist für Caching der Daten zuständig

„beschafft“ Daten von einem Server (bzw. Service)

Ist Zugriffsschicht auf die von einer Anwendung angebotenen Services

- Basis ist die Integrationsarchitektur

„Service Proxy Layer“

Hält die vom Service Layer angelieferten Daten (keine weitere Transformation)

Service Layer

Bietet die Services an, die vom Clientteil der Anwendung (oder Komponenten oder anderen Anwendungen) genutzt werden können

Angebotene Services sollen unabhängig voneinander genutzt werden können

Eine Explizite Reihenfolge der Nutzung von Services darf beim Servicekonsumenten nicht vorausgesetzt werden

Verantwortlich für den korrekten Aufruf der Funktionen des Business Layers

Business Layer

Erzeugt und aktualisiert das servereigene Objektmodell

- Business Model (Geschäftsmodell)

Stellt feingranulare Geschäftsfunktionen zur Verfügung

Diese Geschäftsfunktionen dürfen nur vom Service Layer konsumiert werden

Diese Geschäftsfunktionen dürfen nicht von einem Service Consumer aufgerufen werden

Persistence Layer

Stellt technische Adaption der darunter liegenden Datenbank sicher

- Bei relationalen Datenbanken die objektrelationale Abbildung
- Objektrelationale Abbildungen bilden die Attribute und Beziehungen der Objekte auf Tabellenstrukturen ab

Befindet sich aus Sicht der Architektur auf der gleichen Ebene wie Integrationsschicht

Integration Layer

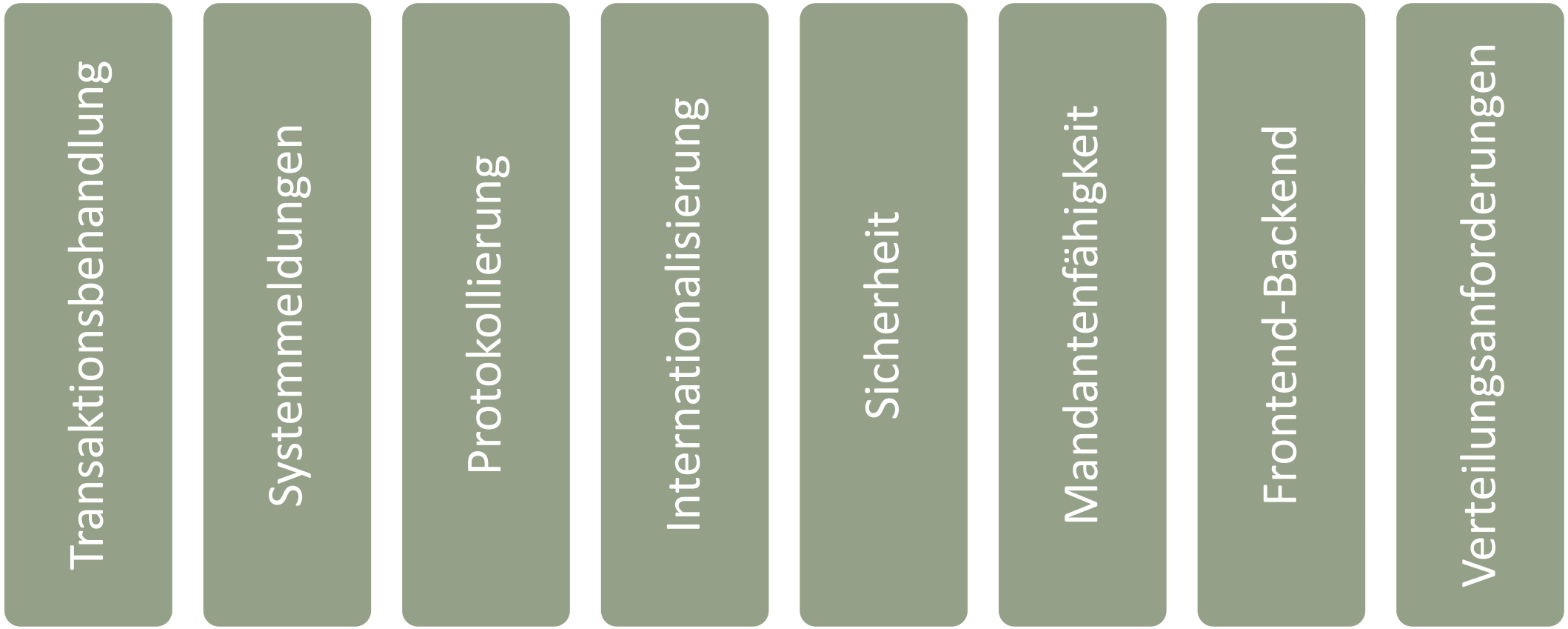
Verantwortlich für den Aufruf fremder Services

Verantwortlich für Integration der von fremden Services bezogenen Daten in das servereigene Objektmodell

Integration zwischen Anwendungen erfolgt standardmäßig auf dieser Schicht

Diese und weitere Festlegungen werden in der Integrationsarchitektur definiert

Vertikale Aspekte



Transaktionsbehandlung

Verantwortlich, dass eine durch die Anwendung vorgenommene Datenänderung die Daten in fachlich korrektem Zustand hinterlässt

Geschäftsregeln beschreiben korrekten Zustand (Konsistenzbedingungen)

Implementierung je nach Komponente

Transaktionale Anforderungen geben sowohl wichtige Service-Designziele als auch die Grenzen der Servicebildung vor

Systemmeldungen

Exception Handling

Definition und Aufbereitung sämtlicher Systemmeldungen an unterschiedliche Empfängergruppen:

- Anwendungsnutzer, Kundendienst, Service Desk, Fachsupport, Betrieb (IT-Ops), Entwicklung (IT-Dev)

Systemmeldungen umfassen fachliche und technischer Fehlermeldungen, Warnungen und Informationen

Validierungen werden als Bestandteil von Systemmeldungen verstanden

- Pre- und Postconditions bei Serviceaufrufen zwingend auf Serverseite

Meldungskategorien, Meldungsarten, Meldungsstrukturierung, Meldungsbehandlung und Meldungstransport sind zu definieren

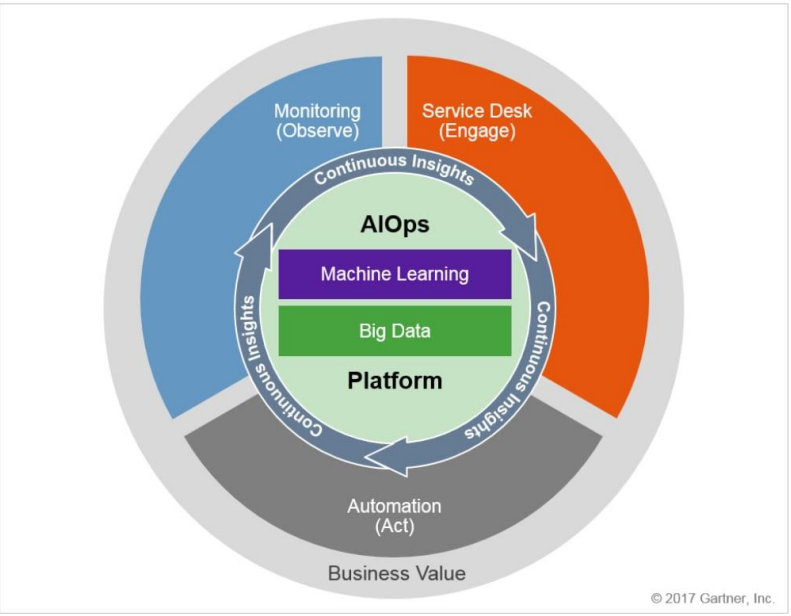
Protokollierung

Dient der Aufzeichnung der Systemmeldungen

Fachliche und technische Nachvollziehbarkeit der Systemnutzung (+ Systemoptimierung und Fehlerlokalisation)

Üblicherweise durch Schnittstelle zu einem Monitoring-System (z.B. Nagios oder Kibana)

Wichtig für automatisierte Abläufe bis hin zur automatischen Selbstheilung (vgl. AI-Ops)



Source: Gartner (August 2017)

Internationalisierung

Oft als i18n bezeichnet

Anpassung an neue Sprachen und Kulturen ohne Änderung des Source-Codes

Siehe auch: Gebietsschema

Lokalisierung umfasst neben der Sprache:

- Einstellungen zum Zeichensatz (vgl. „Ä“ ohne UTF)
- Tastaturlayout
- Datum-, Zeit-, Währungs- und Zahlenformate
- Sortierung von Listen
- Übersetzung der Systemmeldungen
- Übersetzung der Druckerzeugnisse

Zahlenwert	Zeichen	bevorzugte Lesung
0	零 / ○*	zero (ゼロ)
1	一	ichi
2	二	ni
3	三	san
4	四	yon (よん)
5	五	go
6	六	roku
7	七	nana (なな)
8	八	hachi
9	九	kyū
10	十	jū
20	二十	nijū
100	百	hyaku
1.000	千	sen
10.000	万**	man

Quelle: https://kryptografie.de/kryptografie/chiffre/japanische-zahlen.htm

Sicherheit

Schutzmaßnahmen richten sich nach Schutzbedarf der verarbeiteten Daten

Beispiel: „Nicht alles unterliegt dem Datenschutz“

Schutzbedarf bestimmt jedoch Systemrollen und deren Rechte (Lesen, Schreiben, …)

Identifikation, Authentifizierung, Autorisierung

- Kernaspekte der praktischen Umsetzung

Komplexe Anforderungen bei verteilten Anwendungen (vgl. „Systemrollen“)

Mandantenfähigkeit

Mandant: „Kunde bzw. Auftraggeber einer Anwendung, der eine datentechnisch und organisatorisch abgeschlossene bzw. von übrigen Kunden getrennte Einheit bildet“

Unterschiedliche Ausprägungen:

- Einfach – Anpassung des Persistence Layers (Datenmodell mit Mandanten-ID) sowie Business und Service Layer (Durchreichen der ID)
- Fortgeschritten – Business und Service Layer mit fachlicher Anpassung auf individuelle Geschäftsregeln je Mandant
- Voll – volle Konfigurierbarkeit je Mandant einschließlich Benutzeroberfläche

Komplex und aufwändig in der Umsetzung (Source Code)

Zustände einer Anwendung

Zustände müssen jederzeit rekonstruiert oder reproduziert werden können

Wir unterscheiden hierbei Client und Serversicht

- Client startet Serviceaufruf an den Serverteil, der auf persistente Daten zugreift und bei Bedarf weitere Daten bei anderen Anwendungen über Services abfragt
- Integration in eigenes Objektmodell
- Server verwaltet keine Zustände des Clients!
- Will ein Client sicherstellen, dass sein Zustand nicht-flüchtig ist, muss er seine Daten persistieren lassen

Nicht mit „session state“ verwechseln (Netzwerksicht)

Einhaltung von Verteilungsanforderungen

Logische und physische Verteilbarkeit

Logische Verteilbarkeit ist Voraussetzung für physische Verteilbarkeit

… ist ein Aspekt der Softwarearchitektur

Mit physischer Verteilbarkeit lassen sich Plattformfunktionalitäten nutzen

- Insbesondere Skalierbarkeit und Ausfalltoleranz

Logische Komponenten werden i.d.R. erst dann physisch verteilt, wenn nicht-funktionale Anforderungen nur dadurch wirtschaftlich erfüllt werden können

Definition: „Sämtliche Softwarebestandteile sind als physisch verteilt zu betrachten, wenn sie nicht im gleichen Betriebssystemprozess laufen.“[SP17], S. 464

Integrationsarchitektur

Definition: „Die Integrationsarchitektur definiert die technische Umsetzung der Kommunikationsbeziehungen von physisch verteilten Komponenten – dies auch innerhalb einer Anwendung, wie z.B. zwischen Client- und Serverkomponente. Sie wird z.T. auch als Kommunikationsarchitektur bezeichnet.“[SP17], S. 466

Enge versus lose Kopplung

Ziel:

- Integration von physisch verteilten Komponenten effizient in Bezug auf deren Entwicklung und Betrieb
- Änderungen an einer Komponente ohne Auswirkung auf andere Komponenten (Dependencies)

Jede „Dependency“ (technische Abhängigkeit) ist eine „Technische Schuld“

Technische Schulden, siehe:

- Lilienthal, Carola (2016): Langlebige Softwarearchitekturen. Technische Schulden analysieren, begrenzen und abbauen. 1. Auflage. Heidelberg: dpunkt.verlag.

Integration → Ausgestaltung der „Kopplung“ zwischen physisch verteilten Komponenten

Wir unterscheiden eine…

…FACHLICHE KOPPLUNG

Abgeleitet aus der Anwendungsarchitektur

Bildet oft organisatorische Zuständigkeiten ab

Hat das Ziel der fachlichen Stabilität

Im Umkehrschluss bedeutet eine Neuausrichtung der Anwendungsarchitektur daher oft auch eine Re-Organisation des Unternehmens! Hier scheitern viele Institutionen, da Organisation und Prozesse außen vor gelassen werden!

…TECHNISCHE KOPPLUNG

Vgl. Dependencies

Technische Umsetzung von Schnittstellen

Minimieren!

Aufgabe: Sehen Sie sich die vertikalen Schnittstellen einer Cloud-Infrastruktur genauer an und bewerten Sie die technische Kopplung! (siehe Bild nächste Folie)

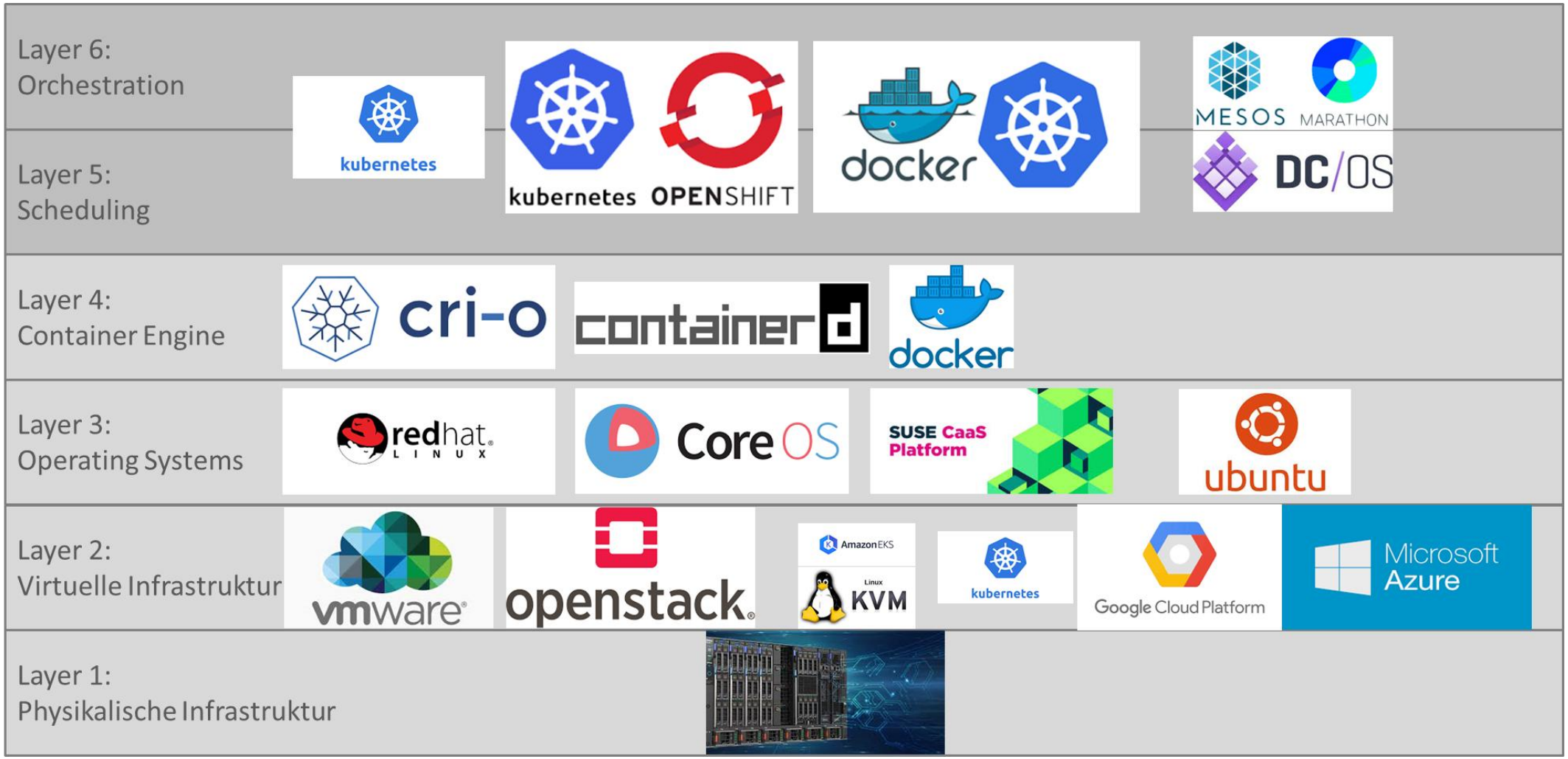


Bild zur Aufgabe

Vgl. vorhergehende Folie

Anforderungen an lose Kopplung

Änderbarkeit

- Vollständige Spezifikation, keine impliziten Annahmen, Kapselung der Implementationsdetails

Robustheit

- Robust gegenüber Fehlerkonstellationen aufseiten des Service Providers

Ausfallsicherheit

- Geschützt gegenüber Ausfällen

Je änderbarer, robuster und ausfallsicherer ein Service ist, desto ***loser*** ist die Kopplung von Service Consumer mit dem Service Provider.

Integrationsarten

Es handelt sich dabei um Entwurfsmuster und Entwurfsentscheidung

Einsatz eines Musters oder einer Referenzarchitektur ist an den Gestaltungszielen ausgerichtet

Häufig sind:

- Über die Datenbank (Persistence Layer); kann zu Redundanz in der Geschäftslogik führen
- Über den Service Layer, häufige Variante – geführt über Integration Layer
- Über den View Layer – erfordert hohes Maß an Standardisierung (heute flächendeckend praktisch nicht mehr möglich!)

- Beachten: sogenannte Self-Contained-Services (z.B. Microservices) bieten technologisch entkoppelbare Ansätze zur View Layer Integration

Kommunikationsmodus

SYNCHRON

Service Consumer *wartet* auf Antwort

Dadurch kann er geblockt werden (z.B. „einfrieren“ einer Dialogmaske)

Führt zu „Timeouts“ der Laufzeitumgebung

Anwender bemerkt i.d.R. den Fehler

ASYNCHRON

Service Consumer wartet nicht auf Antwort

Kann weiterarbeiten

Komplizierter zu entwickeln (Nebenläufigkeit, Fehlertoleranz, …)

Anwender bemerkt den Fehler nicht bzw. in einer besser kontrollierten Form

- Vgl. Amazon oder Netflix

Service Design

Ein Service bildet eine transaktional abgeschlossene Einheit

Services sind kontextfrei und zustandslos zu entwerfen

Die Interfaces gliedern die Operationen eines Service:

- Fachliche Zusammengehörigkeit
- Transaktionales Verhalten
- Benutzergruppen getrennt
- Ähnliches Antwortzeitverhalten
- Kapselung sensitiver Daten (vgl. Datenschutz)

Aspekte der Modellierung von Operationen

Falsches Design der Parameter kann schnell eine enge Kopplung verursachen

Komplexe Parameter sind wenn möglich über Interfaces zu abstrahieren

- Service Provider stellt Factory-Klassen zur Verfügung

Für Parameter sind Zusatzfunktionen zur Visualisierung von Meldungen auf einem Medium zur Verfügung zu stellen (vgl. Protokollierung)

Es dürfen keine Algorithmen in den Parametern versteckt werden

Utility-Klassen (zur Prüfung, Formatierung) sind über Interfaces zu abstrahieren

- Service Provider stellt Factory-Klassen zur Verfügung

Parameter sind vollständig, d.h. alle Attribute sollen gesetzt sein

Hierarchische Datenstrukturen anstelle Vererbung bei unterschiedlichen Technologien

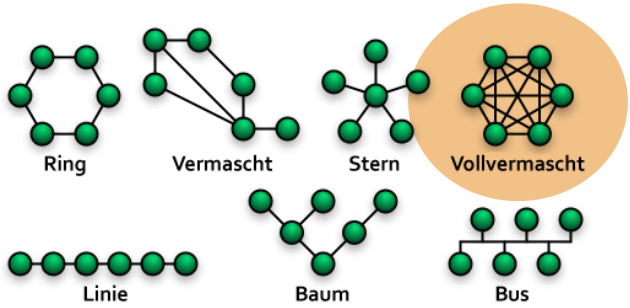
Integrationstopologien

Hierbei handelt es sich um Referenzmodelle zur *Strukturierung der Kommunikationsflüsse* zwischen physisch verteilten Komponenten basierend auf einer Kommunikation über das Netzwerk

Wir unterscheiden im Wesentlichen drei Modelle:

- Peer-to-Peer Topologie
- Hub-and-Spoke Topologie
- Bus Topologie

Peer-to-Peer



Komponenten werden bei Bedarf direkt miteinander verbunden

Maximale Anzahl an Schnittstellen → hohe Komplexität

Niedrige Startkosten

Geeignet für eine geringe Anzahl an Komponenten

Standardisierung und Homogenisierung sehr schwierig wenn Anzahl steigt

Das Fehlen einer Architektur führt fast immer zu diesem Modell (ungesteuert, unkontrolliert)!

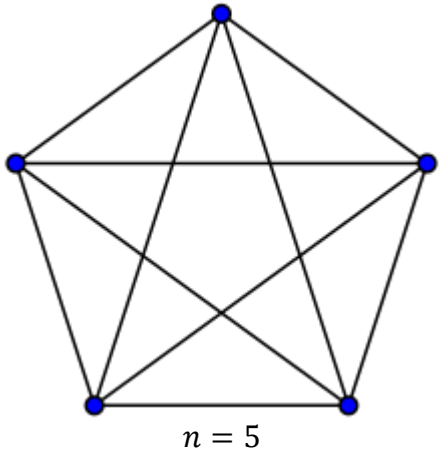
- **Spaghetti-Architektur**

Quelle Grafik: Von NetworkTopologies.png: Foobazderivative work: Parzi - Diese Datei wurde von diesem Werk abgeleitet: NetworkTopologies.png:, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=20287897

Komplexitätsbetrachtung

Bei n Komponenten in einer IT-Landschaft ist (aufgerundet) jedes System mit jedem verknüpft.

Diese Anordnung entspricht der *Peer-to-Peer* Topologie.

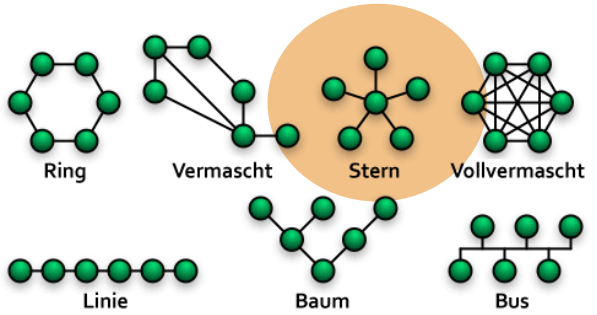


$n = 5$

In der Graphentheorie entspricht sie einem vollständigen Graphen:

- Alle Knoten sind über eine Kante verbunden.
- Die Anzahl der Kanten wird bestimmt durch: $\mathrm{k} = \binom{n}{2} = \frac{n \cdot (n-1)}{2} = \Delta_{n-1}$
- Wird n um eins erhöht, kommen n Kanten hinzu.
- Wird n um zwei erhöht, kommen $n + n + 1$ Kanten hinzu (mehr als verdoppelt).
- Frage: bleibt das Wachstum $2 \cdot n$ unter der oberen Schranke einer Vervierfachung?
- Für obenstehende Formel lässt sich ableiten: $\frac{n \cdot (n-1)}{2} \sim \frac{n^2}{2} \sim n^2$ also $\boldsymbol{f \in O(n^2)}$
- Quadratisches Wachstum

Hub-and-Spoke



Auch Sterntopologie genannt

Nachrichten werden über einen zentralen *Hub* verteilt

- Komponenten senden an Hub, Hub sendet an Empfänger

Geeignet für Datenverteilung (n:m Beziehungen) und Konvertierungsmechanismen

Zentrale Funktionalitäten wie Service Registry, Service Discovery können zentral implementiert werden

Technologiekonversion: technische Kompatibilität bei heterogenen, verteilten Komponenten

- Vgl. Enterprise Service Bus
- Heute durch technische Standards abgelöst: z.B. REST

Quelle Grafik: Von NetworkTopologies.png: Foobazderivative work: Parzi - Diese Datei wurde von diesem Werk abgeleitet: NetworkTopologies.png:, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=20287897

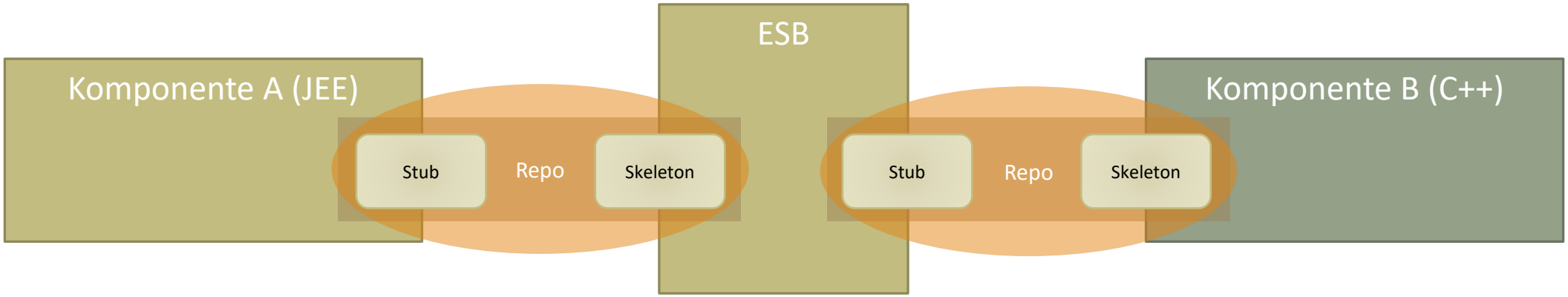
Middleware

Eigene Abbildung, angelehnt an: [SP17], S. 488

Die zentrale Informationsdrehscheide (Hub) stellt eine klassische *Middleware* dar

In größeren IT-Landschaften gibt es Middleware-Engineering-Teams, die sich um Standards und Kompatibilität (Technologiekonversion) kümmern:

- Damit der Service Provider nicht alle Technologien bedienen muss, werden vom Middleware-Engineering-Team sogenannte Service-Proxy je Technologie eingesetzt
- Ein Service-Proxy besteht aus einem clientseitigen Stub und einem serverseitigen Skeleton-Teil
- Werden je Technologie im Service Repository hinterlegt



Notwendige Festlegungen

Für das Service-Design verwendbare Datentypen einer Programmiersprache müssen definiert werden

Vererbung und Polymorphismus sind verboten

Untypisierte Datentypen sind verboten

Zu versendende Daten können serialisiert werden

- Siehe auch: Serialisierung, Marshalling
- Stub serialisiert, Skeleton deserialisiert

Am besten sind technologie-unabhängige Datenaustauschformate

- So könnte auf dem Hub ein einheitliches Format gehalten werden, Konversion über Stub/Skeleton

Vor- und Nachteile

Gut bei komplexen Datenverteilungs- und Konvertierungsanforderungen

Gut bei vielen (nicht *zu vielen*) Kommunikationsbeziehungen

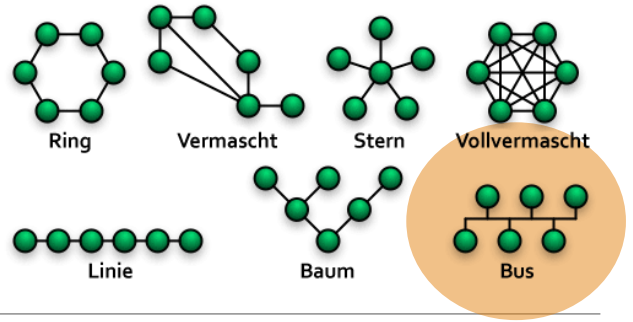
Gut bei unterschiedlichen Technologien

Hohe Startkosten

Hub kann zum Bottleneck werden, da dort immer ein weiterer Serviceaufruf erfolgt

Hoher Reifegrad in der Architektur (Vorgaben, Middleware-Team) notwendig

Bus Topologie



Nachrichten werden über einen *Bus* verteilt (*Bussystem*)

Anwendungen „registrieren“ sich beim Bus, wenn sie an Nachrichten interessiert sind

- „Abonnieren“ Nachrichten, die sie empfangen möchten

Beispiel: MQ – „Message Queues“

Unterschied zu Hub-and-Spoke:

- Keine zentrale Middleware
- Verteilungsfunktionalität in die Service Endpoints verschoben
- Zentrale Funktionen der Middleware sind jetzt selbst Service Endpoints

Hohe Startkosten, geringere Folgekosten

Quelle Grafik: Von NetworkTopologies.png: Foobazderivative work: Parzi - Diese Datei wurde von diesem Werk abgeleitet: NetworkTopologies.png:, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=20287897

Beipsiel: Identity Management

Vgl. CODEX Projekt der Thüringer Hochschulen

Identitäten werden über einen Bus verteilt

Liefernde Systeme (z.B. Personalsoftware, Studierendensoftware) melden neue Identitäten auf den Bus

Konsumierende Systeme abonnieren entsprechende Nachrichten und werden über neue Identitäten informiert

Dadurch können die konsumierenden Systeme jederzeit auf alle Identitäten zugreifen und beispielsweise mit Rollen und Rechten verknüpfen

- Z.B. ein Active-Directory

Die Logik auf dem Bus besitzt eine hinreichend hohe Komplexität und für jedes angeschlossene System muss ein *Connector* implementiert werden

https://www.tu-ilmenau.de/fileadmin/media/ub/doc/dv_thuer/ws2004/metadir.pdf

Orientierung

Topologie	Wenige Kommunikationsbeziehungen		Viele Kommunikationsbeziehungen	
	Wenige Technologien	Viele Technologien	Wenige Technologien	Viele Technologien
Peer-to-Peer	++	0	--	--
Hub-and-Spoke	--	0	+	++
Bus	--	-	++	0

Legende:

++	Wirtschaftlich beste Lösung
+	Lohnt sich tendenziell
0	Einsatz ist zu prüfen
-	Lohnt sich tendenziell nicht
--	Kein Einsatz

Tabelle nach [SP17], S. 494

IT-Landschaften

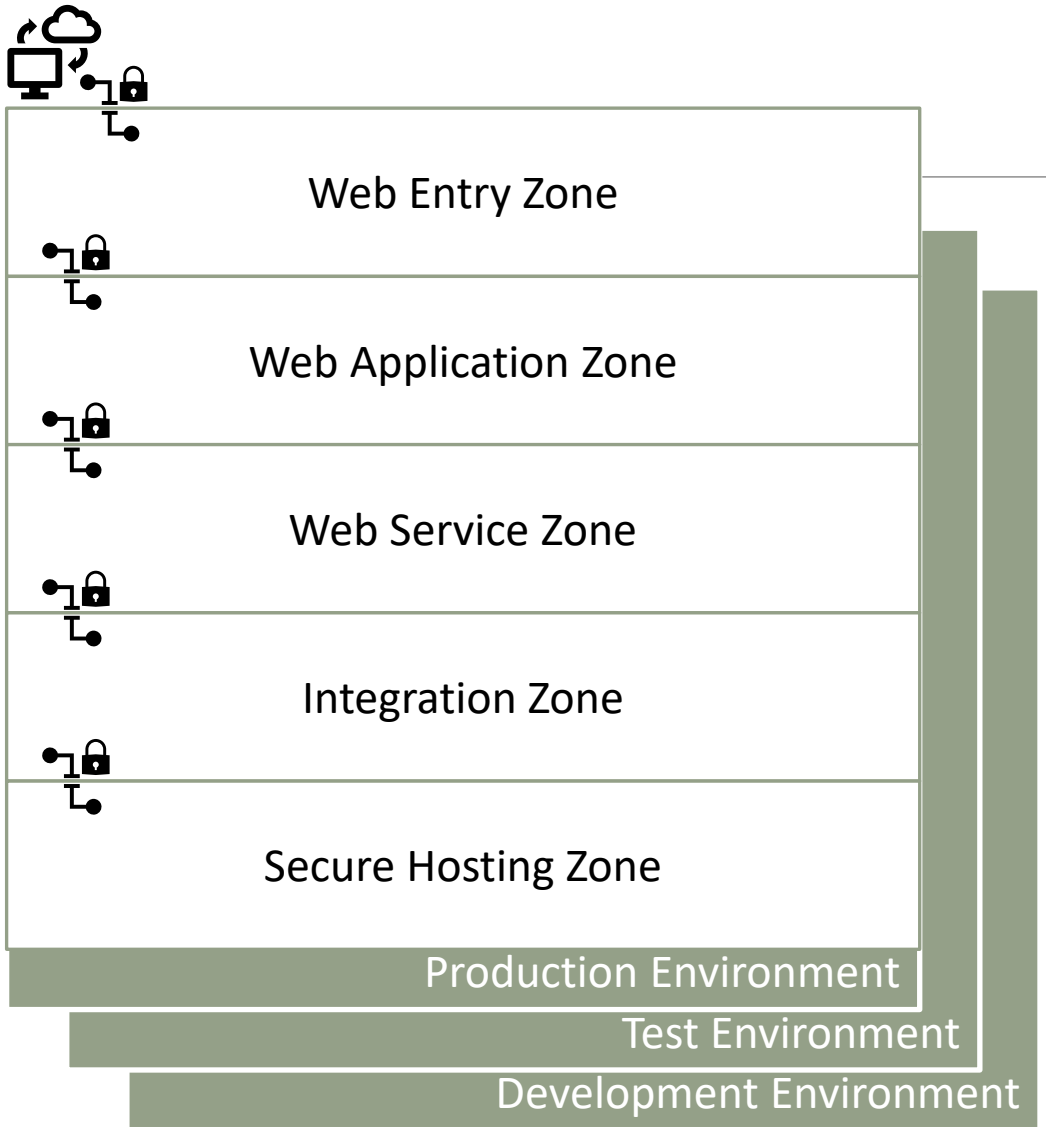
In größeren IT-Landschaften kommen oft alle Topologien in mehreren Instanzen zum Einsatz

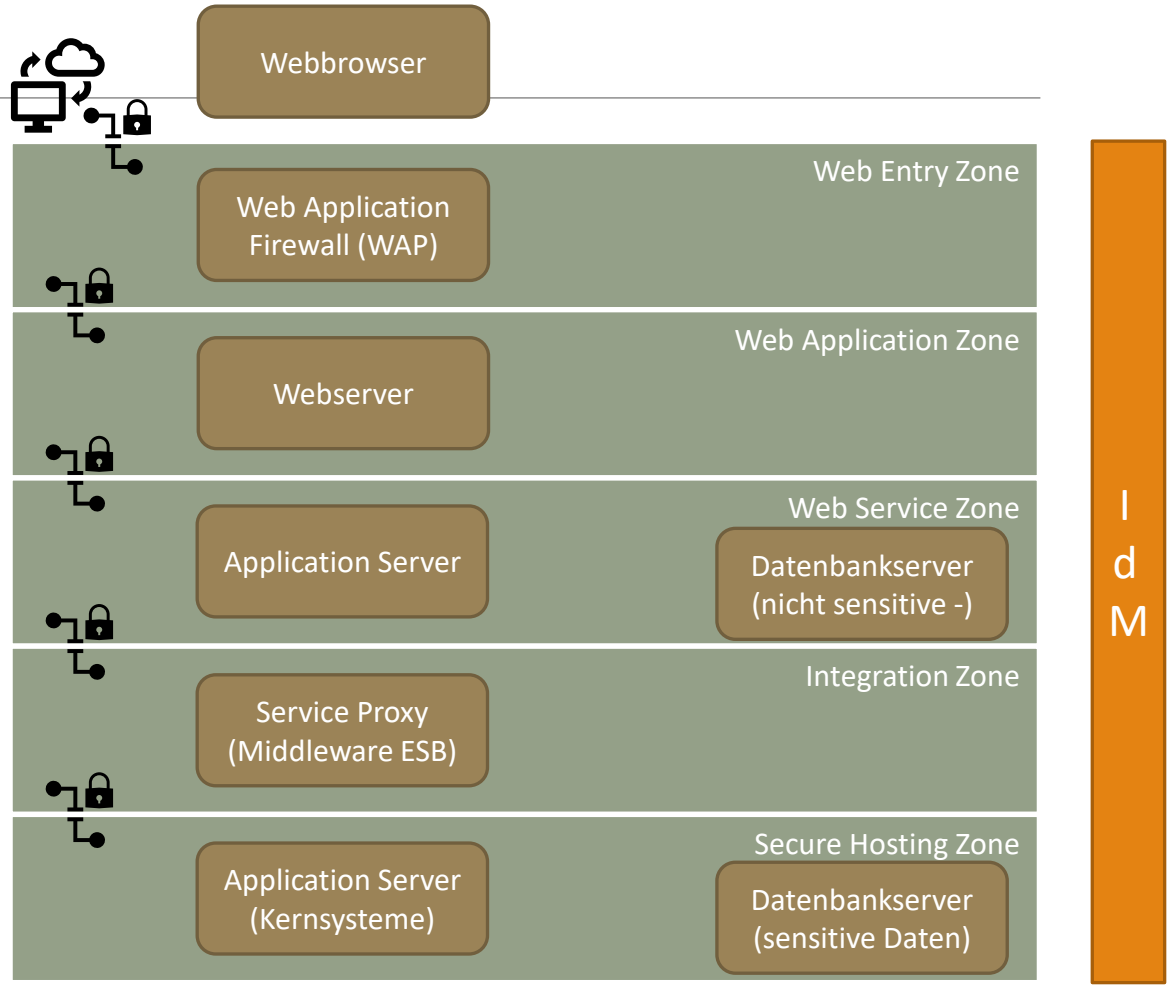
Bei mehreren hundert Applikationen steigt die Komplexität der Schnittstellen stark an

Daher werden sogenannte „Integrationszonen“ eingeplant

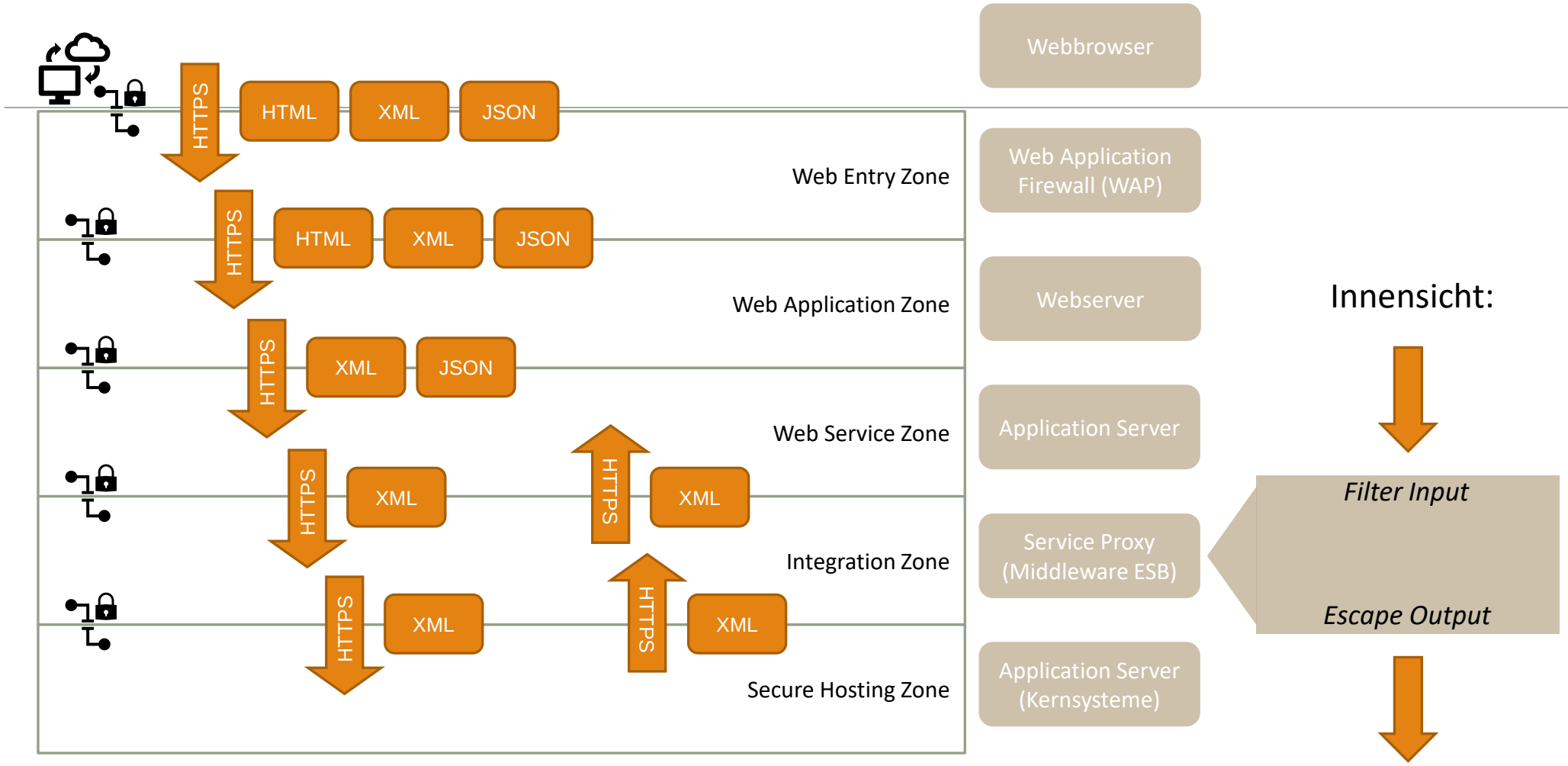
Die Referenzarchitektur auf der nächsten Seite zeigt ein Zonenmodell für Webanwendungen

Referenzmodell



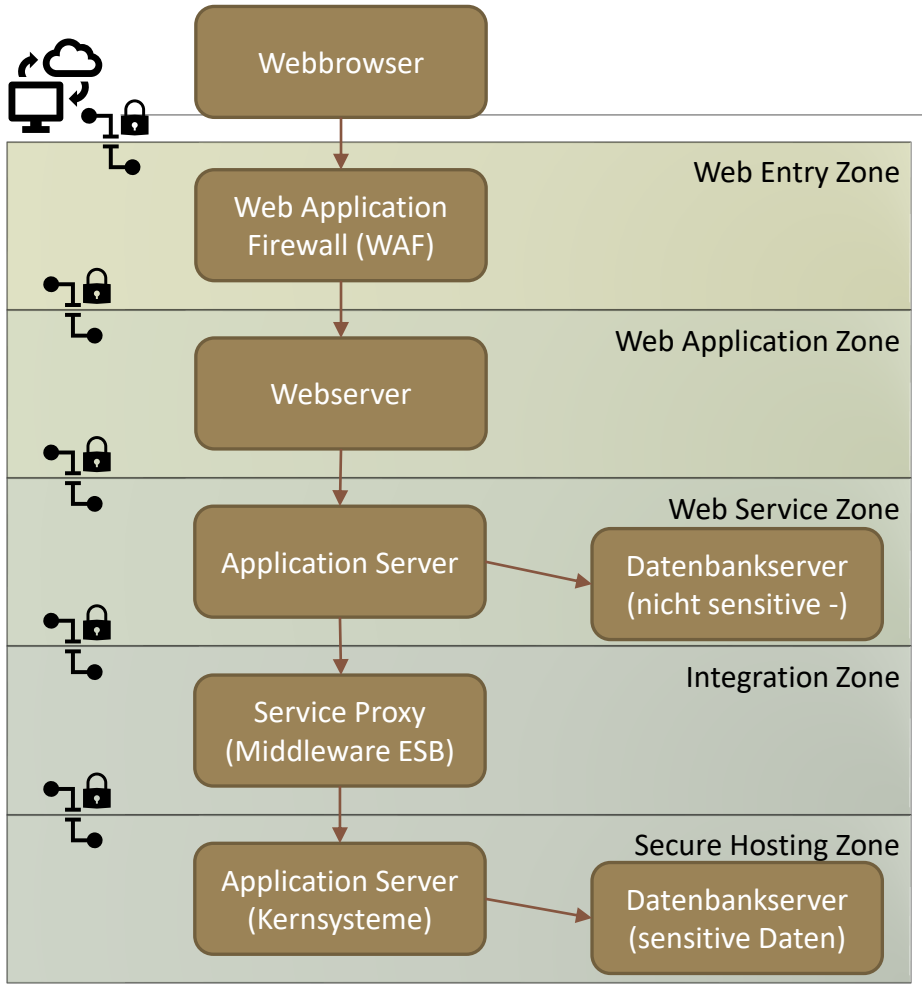


Protokolle und erlaubte Zugriffe



Beispiel – ausreichend für die meisten Szenarien im Dienstleistungssektor

Schutz der Vertraulichkeit und Integrität



Webbrowser – GET-Request an URI

WAF verlangt Authentifizierung (z.B. LDAP) und gibt (nach unten) Security Token mit

Webserver leitet Request an Application Server

Application Server prüft Autorisierung (LDAP)

- Zugriff auf nicht sensitive Daten bereits hier

Wenn die Internetanwendung auf sensitive Daten zugreift, dann müssen Services der Kernsysteme verwendet werden (über einen ESB: Middleware).

Sicherheit und Qualität

	Verfügbarkeit	Integrität	Vertraulichkeit
Konstruktive Qualitätsmaßnahmen: • **Technisch** • **Organisatorisch**	Qualität durch Anwendung von: Methoden, Sprachen, Werkzeugen, Richtlinien, Standards und Checklisten Scrum, SAFe XML, IDL, WSDL SonarQube, Veracode, … DSGVO ISO***, CMMI		
Analytische Qualitätsmaßnahmen: • **Analysierend** • **Testend**	Qualität durch Anwendung von: Inspektion, Verifikation, Review, Audit, Symbolischer Test, Dynamischer Test z.B. Last-Simulation, DDoS-Simulation DSVGO-Audit z.B. PenTest (siehe Übung)		