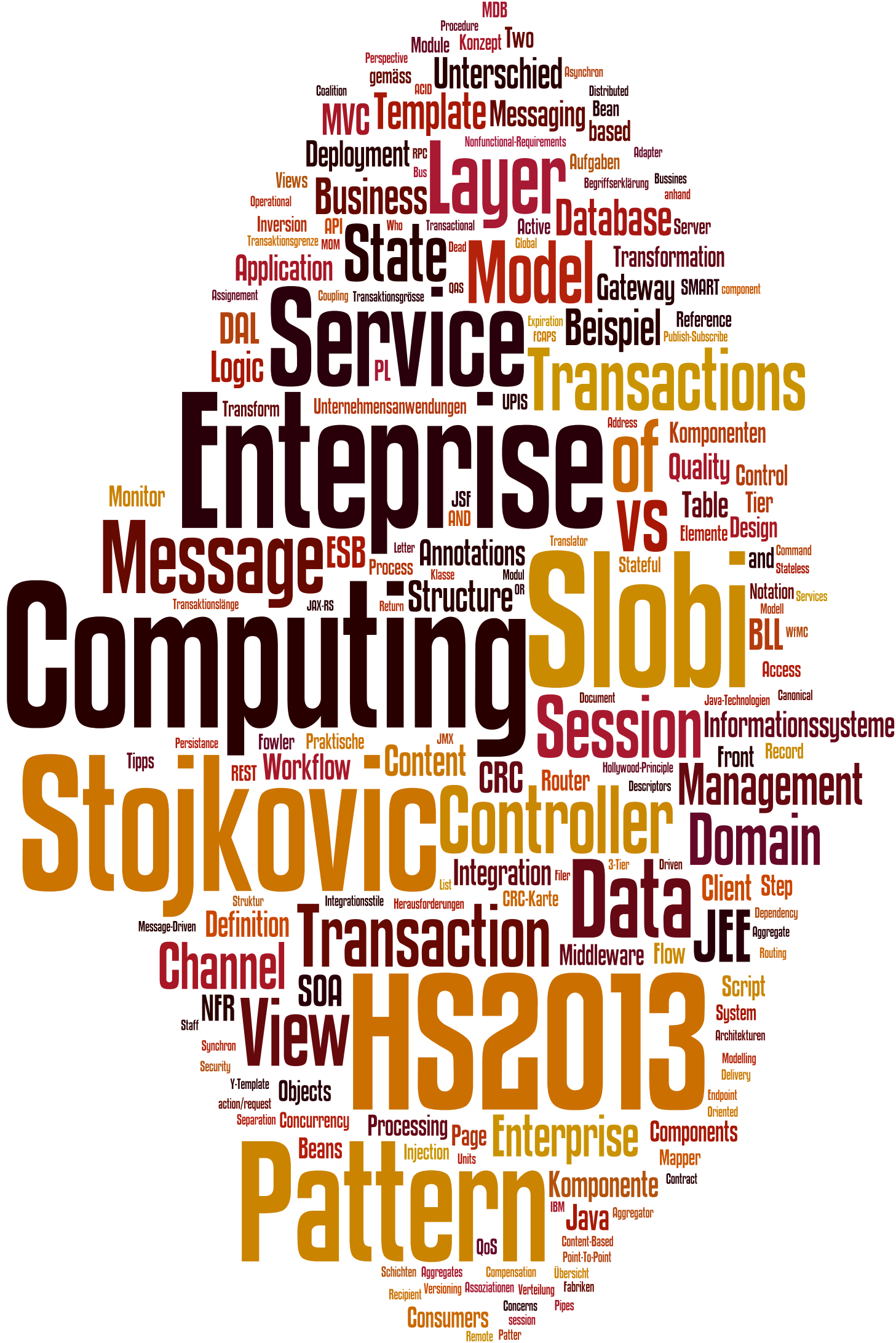

MDB
Procedure
Module
Konzept
Two
Perspective
gemäss
Unterschied
Asynchron
Coalition
ACID
Distributed
MVC
Template
Messaging
Bean
based
Nonfunctional-Requirements
Deployment
RPC
Layer
Aufgaben
Adapter
Views
Bus
Begriffserklärung
Bussines
Business
anhand
Operational
Inversion
API
Who
Transactional
Active
Database
Server
Transaktionsgrenze
MOM
Dead
Global
State
Application
Model
Transformation
QAS
Assignement
Coupling
Transaktionsgrösse
Gateway
SMART
component
DAL
Service
Expiration
fCAPS
Beispiel
Reference
Publish-Subscribe
Logic
PL
Transactions
UPIS
Transform
Unternehmensanwendungen
Enteprise
of
Address
Komponenten
Quality
Control
Monitor
JSF
AND
vs
Table
Tier
Elemente
Design
Message
ESB
Letter
Annotations
Translator
Stateful
and
Command
Stateless
Process
Klasse
Modul
Structure
OR
Notation
Services
Modell
Transaktionslänge
JAX-RS
Return
Computing
Slobi
BLL
WfMC
Access
Document
Java-Technologien
Canonical
Session
Informationssysteme
Persistance
Fowler
Praktische
JMX
Front
Record
Tipps
REST
Workflow
Content
CRC
Router
Hollywood-Principle
Descriptors
Management
Stojkovic
Controller
Domain
Integration
Filer
3-Tier
Driven
List
Data
Client
Step
Struktur
Integrationsstile
Herausforderungen
CRC-Karte
Dependency
Message-Driven
Definition
Transaction
JEE
Aggregate
Middleware
Flow
Routing
Channel
SOA
HS2013
Script
NFR
System
Staff
View
Architekturen
Synchron
Modelling
Security
Delivery
Y-Template
Objects
Endpoint
action/request
Oriented
Separation
Concurrency
Processing
Page
Enterprise
Components
Beans
Injection
Units
Mapper
Pattern
Komponente
Contract
IBM
Java
Aggregator
QoS
Content-Based
Point-To-Point
Schichten
Aggregates
Compensation
Übersicht
Recipient
Versioning
Assoziationen
Verteilung
Fabriken
Concerns
Pipes
Consumers
session
Remote
Patter

Enterprise Computing

Inhalt

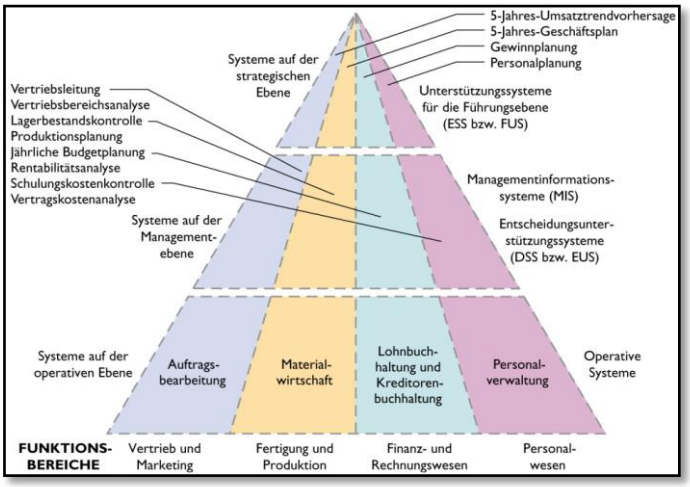
1. Grundlagen Informationssysteme und Unternehmensanwendungen

1.1 Informationssysteme

1.1.1 Definition Informationssysteme

- Ein Informationssystem ist ein System, das die Deckung von Informationsnachfrage zur Aufgabe hat. Es handelt sich dabei um ein so genanntes **Mensch/Aufgabe/Technik-System**, das Daten (bzw. Informationen) produziert, beschafft, verteilt und verarbeitet.
- Synonyme Begriffe: Anwendungssystem, Unternehmensanwendung

1.1.2 Bedarf für Informationssysteme im Unternehmen



Lenkungsebenen

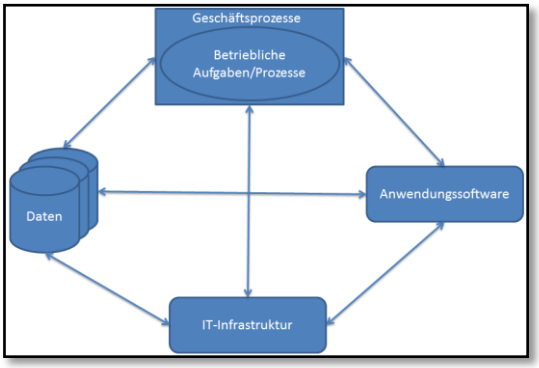
- Vorstand
- Mittleres Management
- Operative Funktionen (Sachbearbeitung)

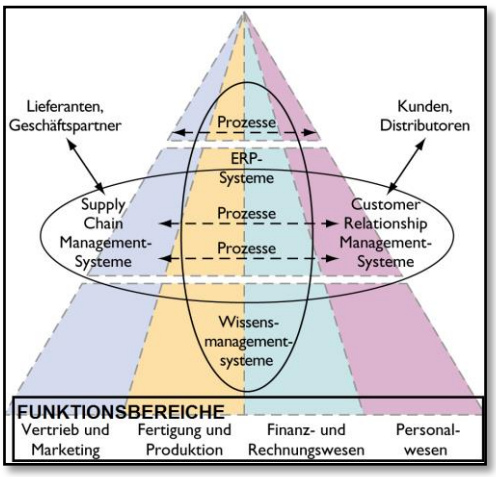
Diverse Funktionsbereiche

- Entlang der Wertschöpfungskette
- Unterstützende Funktionen

1.1.3 Komponenten von Informationssystemen/Anwendungssystemen

Unternehmensweite Anwendungssysteme automatisieren Geschäftsprozesse, die mehrere Geschäftsfunktionen und Organisationsebenen sowie externe Geschäftspartner und Kunden umfassen können.



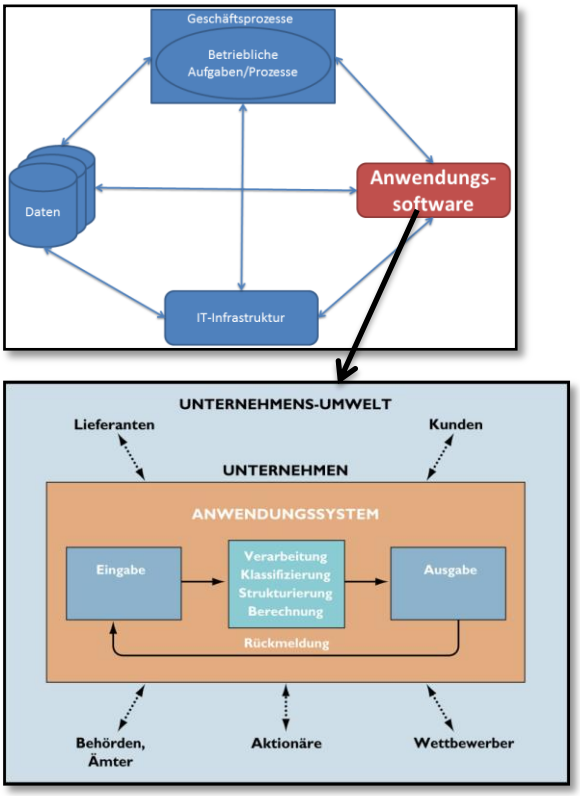


Unternehmesweite Anwenungssysteme

- Enterprise Resource Planning Systeme (ERP)
- Supply Chain Management Systeme (SCM)
- Customer Relationship Management Systeme (CRM)
- Wissensmanagement-systeme (KM)

1.2 Unternehmensanwendungen

1.2.1 Grundfunktionen und Kontext



System von Systemen

- Interne und externe Benutzer als Teil des Systems (z.B. Start eines Vorgangs, Prüfungen)

Commercially-Off-The-Shelf (COTS) vs. Do-It-Yourself (DIY)

- COTS: z.B. SAP, Oracle
- DIY: im Unternehmen oder Ausschreibung/Vergabe externer Aufträge

Viele Umsysteme, überlappende Realisierungen von Funktionen

- Systemintegration erforderlich
- firmenweite Bebauungspläne (Analogie: Stadtplanung)

Beispiele von Unternehmensanwendungen

- Shipping tracking, cost analysis, credit scoring, insurance controller, supply chain management, Customer service

KEINE Unternehmensanwendungen

- Elevator controller, telephone switch, operating system, compilers

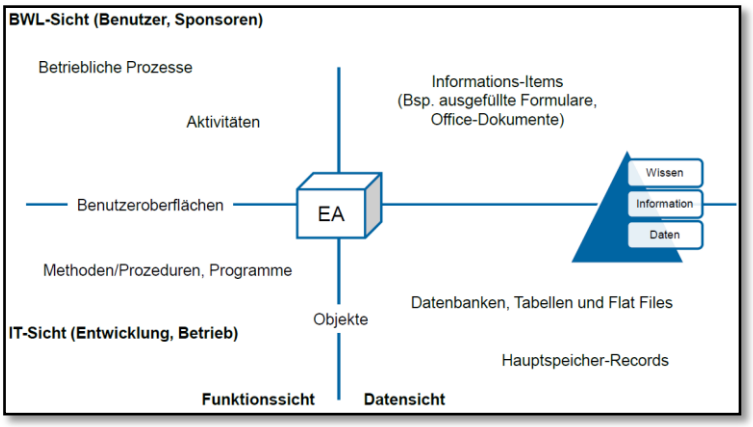
1.2.2 Definition Enterprise Computing

- Enterprise Computing beschäftigt sich mit der ingenieursmässigen (d.h. u.a. der qualitätsorientierten, methodisch unterstützten) Konstruktion, Integration und Bereitstellung von verteilten Informationssystemen, die Geschäftsprozesse unterstützen und teilweise automatisieren.
- Für Entwicklung und Integration dieser Enterprise Applications (EAs), dt. Unternehmensanwendungen, werden sowohl universelle als auch fachdomänenspezifische Middleware, Frameworks (JEE, Spring) und Tools (BuildServer, SE-Werkzeuge, z.B. Modelldriven Development, etc..) verwendet.

1. Welche Teile der Definition sind domänenspezifisch?

- Geschäftsprozess (z.B. Schadenmeldung bei Versicherungen)

2. Welche Unternehmensanwendung gibt es an der HSR und wie sind sie realisiert?
   - unterricht.hsr.ch ; Geschäftsprozess: Modulanmeldung, Prüfungsanmeldung



1.3 Design-Herausforderungen (UPIS) / Anforderungen

1.3.1 UPIS

- ***U*ser and Channel Diversity (Zugangsvielfalt)**
  - Solutions: e.g. PoEAA patterns (presentation layer)
  - Multichannel
  - Viele verschiedene Rollen, z.B. unterricht.hsr.ch -> students, dozenten
  - Mehrere verschiedene Benutzer
- ***P*rocess and Resource Integrity (Datenintegrität -> I von ACID)**
  - Solutions: e.g. PoEAA patterns (business logic/persistence layer)
- ***I*ntegration Needs (Heterogenität)**
  - Solutions: e.g. EIP patterns and messaging, service orientation
  - Verschiedene Programmiersprachen
- ***S*emantics (Komplexe Daten und Verarbeitungsregeln)**
  - Solutions: e.g. Object-Oriented Analysis and Design (OOAD), Domain Modeling, Domain-Driven Design

1.3.2 Weitere Anforderungen und Herausforderungen

- **Compliance („Regelbefolgung“)**
  - FATCA, SOX und andere (Motivation, Einfluss auf Projekt, Reality Check)
  - “Business rules that cannot be applied to simple logical reasoning”
- **Concurrency**
  - Marktdynamik
  - Projekte im Haus und bei Partnern
  - Abhängigkeiten zwischen Systemen, fehlende zentrale Kontrolle
- **Company Politics**
  - Führt zu Mehrfachimplementierungen, Coderedundanz, Dateninseln

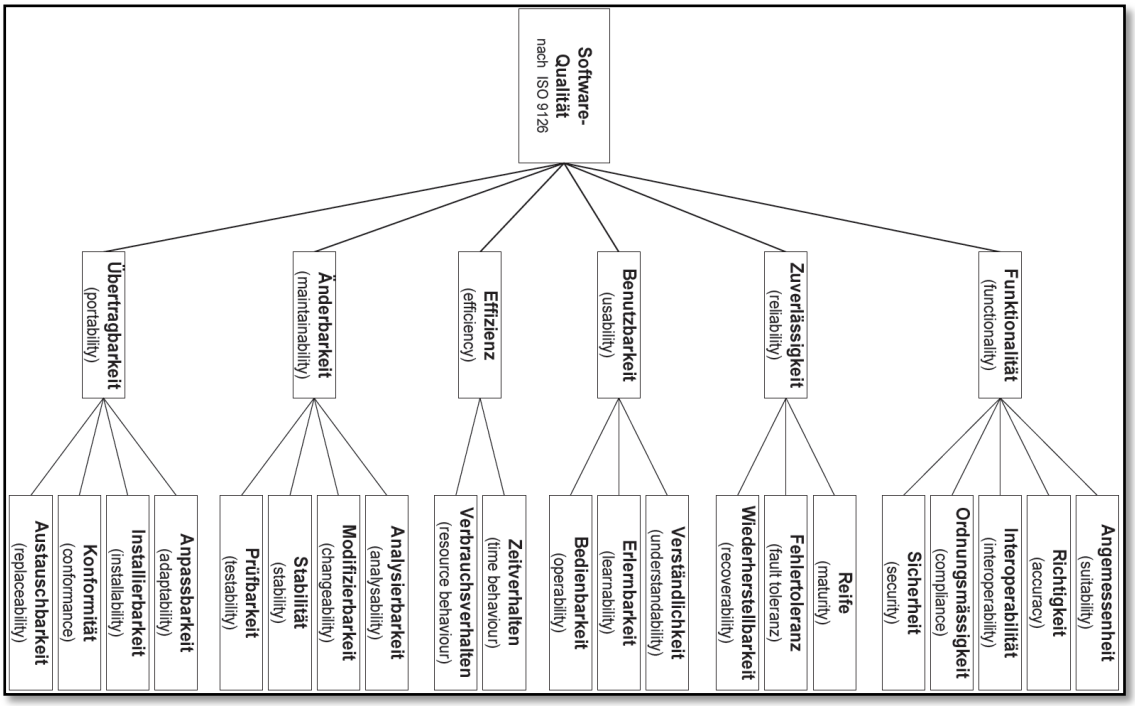
2. SMART Nonfunctional-Requirements

2.1 SMART

	Bedeutung	Beschreibung
S	Spezifisch	Ziele müssen eindeutig definiert sein (nicht vage, sondern so präzise wie möglich) • Ist die funktionale Anforderung oder Architekturkomponente definiert, auf die sich das NFR bezieht?
M	Messbar	Ziele müssen messbar sein (Messbarkeitskriterien) • Kann in Design, Programmierung und Test die Einhaltung des NFRs überprüft werden?
A	Akzeptiert	Ziele müssen von den Empfängern akzeptiert weden/sein (auch: angemessen, attraktiv, ausführbar oder anspruchsvoll
R	Realistisch	Ziele müssen möglich sein
T	Terminierbar	Zu jedem Zeitpunkt gehört eine klare Terminvorgabe, bis wann das Ziel erreicht sein muss.

2.2 NFR Nonfunctional-Requirements

- Welche Qualitätseigenschaften müssen erfüllt sein (wie sind die Features umgesetzt?)
- Meist sind die NFR's die „-ility" Eigenschaften
- Sie messen/definieren:
  - Leistung (wie schnell?): z.B. Antwortzeiten, Durchsatzraten
  - Mengen (wie viel?): z.B. Anzahl Kundensätze, Anzahl gleichzeitige Benutzer
  - Qualitätsmerkmale (wie gut? Siehe Grafik)
  - Randbedingungen (wie?): z.B. Einschränkungen wie Programmiersprachen, DB, etc.



2.2.1 Beispiel NFR's für ein Konzert-Ticket-System

Für die folgenden NFR's ist das Beispiel ein **Konzert-Ticket-System:**

Beschreibung	NFR
Availability/Reliability/Fault Tolerance **Verfügbarkeit/Zuverlässigkeit/Fehlertoleranz** Die Systemverfügbarkeit für den Benutzer jederzeit.	99.99% Verfügbarkeit (dies sind maximal 52min down-time in einem Jahr)
Usability / Benutzbarkeit Das System ist einfach bedienbar und navigierbar für einen Laien.	Der Laie sollte nicht mehr wie 3 Clicks benötigen um den gewünschten Screen oder das gewünschte Feature zu erreichen.
Performance Der Durchsatz der Requests.	Der Request und Response sollte unter einer Sekunde bearbeitet werden, nachdem der Benutzer die Aktion „Ticket kaufen" ausgelöst hat.
Security / Sicherheit Wie sicher ist das System gegen Hacker oder unauthorisierte Benutzer	Das System muss die Kommunikation zwischen dem Server und Client verschlüsseln und signieren. **Besser: genauer, z.B. wie verschlüsselt?**
Portability **Übertragbarkeit** Die Fähigkeit ein System auf verschiedene Plattformen laufen zu lassen.	Das System muss fähig sein, auf folgende Datensourcen erweitert oder portiert zu werden: MySQL, Oracle, DB2, MS SQL Server **Besser: Aufwand (10PersonenTage, < CHF 5000.-)**
Scalability **Skalierbarkeit** Die Skalierbarkeit bei grosser Anzahl von aktiven Benutzern.	Das System muss 10'000 Benutzer gleichzeitig verarbeiten können. **WICHTIG: *Gleichzeitige* user**

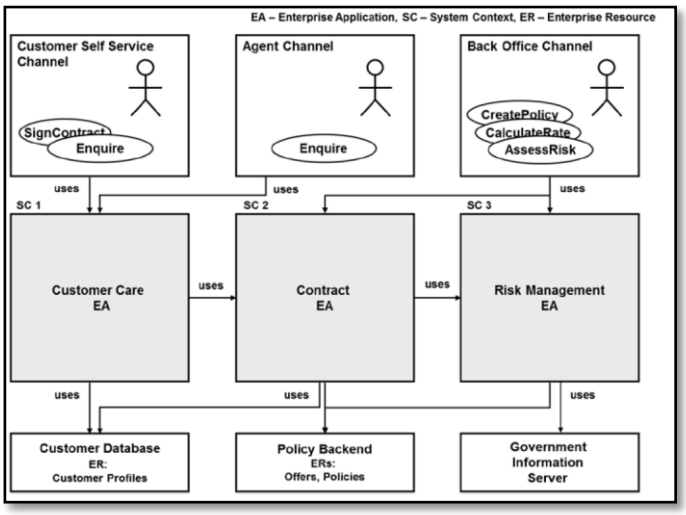
2.3 QAS Quality Attribute Scenario

2.3.1 Template

Scenario	[Wie heisst das Szenario, um welche Art von NFR geht es?]
Source	Wer startet das Szenario (Quelle)? Woher kommt die Anforderung? [Bsp. Benutzer/Aktor, systeminterner Trigger] (Die Source kann aus dem System Context Diagram (SCD) entnommen werden.)
Stimulus	Was macht die Source, um das Szenario zu starten (Ereignis, Trigger/Auslöser)? [UseCase oder Aktivität im Geschäftsprozess] (Der Stimulus kann aus den Use Cases oder dem BPMN-Diagramm entnommen werden)
Artifact	Wo findet das beschriebene Response-Verhalten, das von der Source stimuliert worder ist, statt? Welcher Systemteil ist betroffen? [Ganze Anwendung oder einzelne Komponenten] (Artifact ist eine einzelne Komponente oder das ganze System (bei Runtime QAs))
Environment	Wann kann das spezifizierte Response-Verhalten beobachtet werden? Um welche Umgebung/welchen Systemzustand geht es? [Laufzeit oder Phase im Software Engineering/Wartung]
Response	Wie reagiert die Anwendung qualitativ auf den Stimulus? Was soll gemessen werden? [nach aussen sichtbares Verhalten, Messgrösse, koch kein Design] (Die Response kann ein Pattern sein (muss die S-Eigenschaft bringen))
ResponseMeasure	Wie kann die Response quantifiziert werden, um sie überprüfbar zu machen? Welches Messergebnis wird angestrebt? [Messeinheit, konkrete Zahlenangabe im Kontext der vorherigen QAS-Template-Elemente wie Source, Stimulus, etc..] (Das Response Measure sollte testbar sein (M-Eigenschaft))

2.3.2 Beispiel

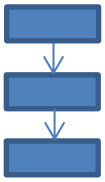
Scenario	NFR 4: „Sub-second response time is required in the customer self service and the agent channels (for all processing steps defined in the analysis-phase BPM).”
Source	Ein externer Systembenutzer über die drei Kanäle (siehe Abbildung), also entweder Kunde, Agent oder Back Office-Mitarbeiter (external to system)
Stimulus	Ausführung einer der Use Cases in der (siehe Abildung) bzw. der Aktivitäten im Geschäftsprozess also Enquire, Sign Contract, Assess Risk, Calculate Rate, oder Create Policy
Artifact	Eine der drei Unternehmensanwendungen, bei Enquire durch Kunde also z.B. die Contract Anwendung oder die Customer Care Anwendung (eine Anwendung pro Quality Attribute Szenario).
Environment	Zur Laufzeit (Normaler Betrieb inklusive Peaks wie z.B. Jahresendgeschäft)
Response	Die Antwortzeit aus Sicht der externen Source (s.o.) soll gemessen werden, vom Absenden des Stimulus (s.o.) bis zum Empfang der ersten Antwort der Anwendung (also dem Artifact). Bsp. Bestätigung des Anfrageeingangs an den Kunden im Self Service Channel bei Aufruf von Enquire.
ResponseMeasure	Die Response (wie oben definiert) soll innerhalb von weniger als einer Sekunde eingehen – im Kontext von (Source, Stimulus, Artifact, Environment).



3. Layer vs. Tier, 2-Tier, 3-Tier

3.1 Layer (logische Verteilung)

3.1.1 Konzept



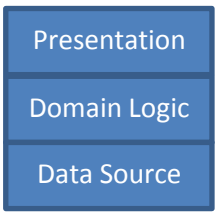
- Each layer rests on a lower layer using the services the lower layer provides
- Each layer is unaware (unwissen) of layer above
- Usually each layer hides his lower layers to above layers

3.1.2 Vorteile

- Better understanding of a single layer (not knowing much about lower layers, but trusting their functionality) -> Responsibility (Verantwortlichkeit)
- Substitution (Austausch) of layers possible (providing alternative implementations for the interfaces)
- Minimize layer dependency (FTP still works when cable company changes physical transmission and makes IP work on it)
- Layers are good places for standardization (see Transmission Control Protocol, TCP)
- Layers can be reused by higher level services (e.g. HTTP and FTP can both be based on TCP/IP)

3.1.3 Three Principal Layers

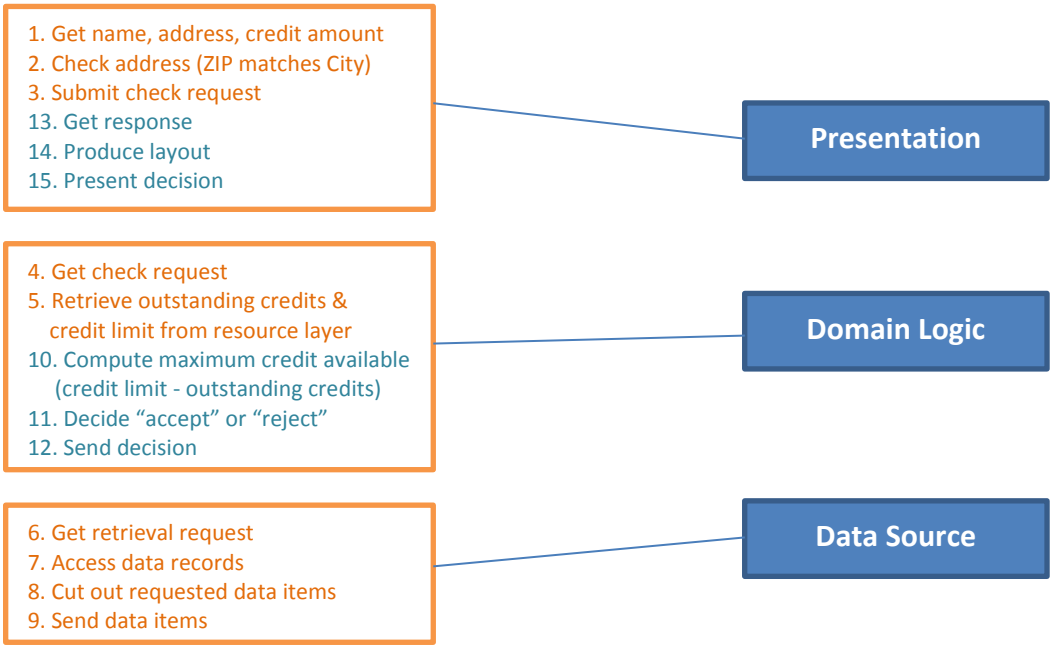
- Top Layer: **Presentation**
  - Handle interaction between user and software, e.g. rich client or browser
  - Display information and transform user actions into commands for the domain and data source layer
- Middle Layer: **Domain logic**
  - Domain logic is the work the application needs to do for this domain (business logic), e.g. calculations based on input, data validation, data source logic to dispatch
- Lowest Layer: **Data source**
  - Communicating with other systems, that carry out tasks on behalf of the system, e.g. with a database, transaction manager or messaging system



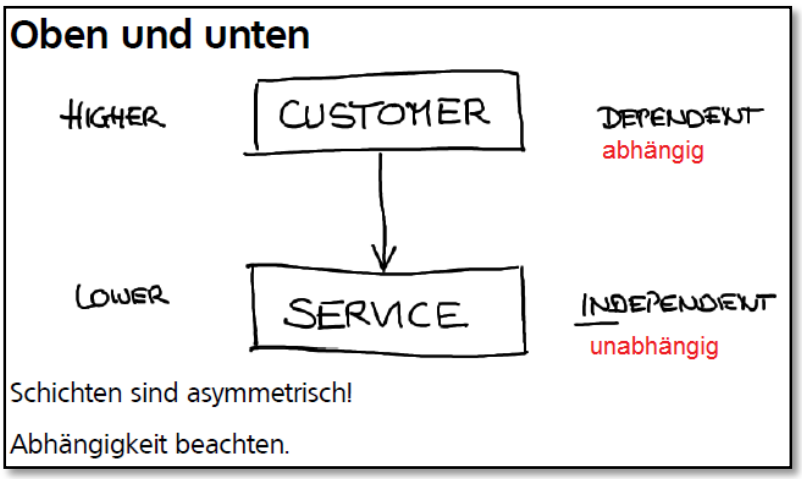
3.1.4 *Where to run the layers?*

- Layers define logical separation of functionality to reduce coupling
- Physical structure of a system makes a difference
  - Decision whether to run on a client, desktop machine or server
- Data source layer often runs on servers
  - Exceptions: disconnected mode (beware of synchronization)
- Presentation depends on interface selection
  - HTML browser:
    - Presentation layer runs on server
    - UI available for many clients, but bad responsiveness
  - Rich Client:
    - Presentation layer runs on client/desktop
    - More functionality offered by rich clients, but different rich clients on different machines to maintain
- Domain logic
  - Running on server increases maintainability
  - Running on client increases responsiveness and allows disconnected mode
    - Still keep domain logic separate from presentation
    - Use **Domain Model Pattern** or **Transaction Script** Pattern
  - Running domain logic on both, partially on client and partially on server makes things worse
- If layers need to be separated into discrete processes, use **Remote Facades** and **Data Transfer Objects** Patterns

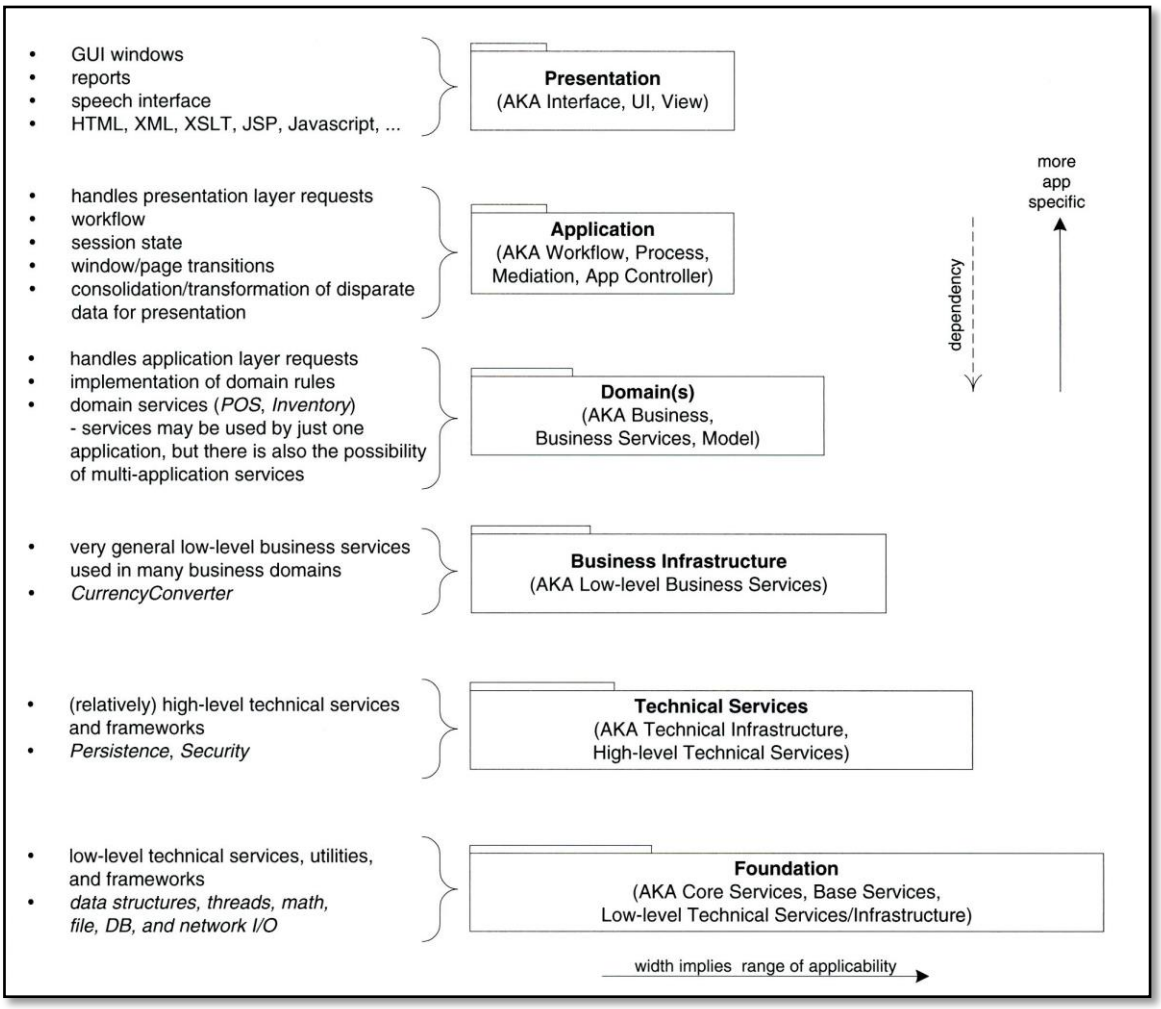
3.1.5 *Layer Example: Credit Check*



3.1.6 Schichten: oben/unten, asymmetrisch

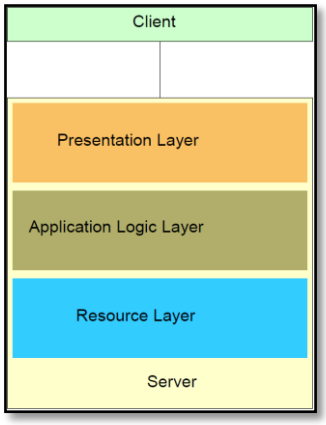


Klassisches Schichtenmodell



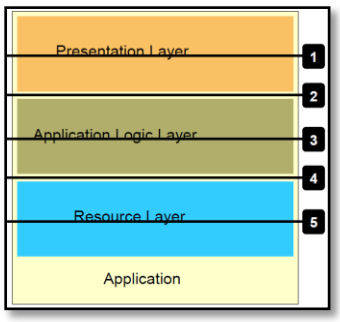
3.2 Tier (physische Verteilung)

3.2.1 1-Tier Structure: Centralized Computing



- This is the structure of applications in mainframe environment
- Clients are “dumb terminals”
  - ..or ATMs (automated teller machines) or…
- **Advantages** of 1-tier
  - Everything is centralized, managing and controlling resources is “easy”
  - Design can be highly optimized by “blurring the layers”
- **Disadvantages**
  - Ignores compute power available on clients
  - Code hard to maintain
  - Reuse of functions very difficult

3.2.2 2-Tier Structure: Client-Server

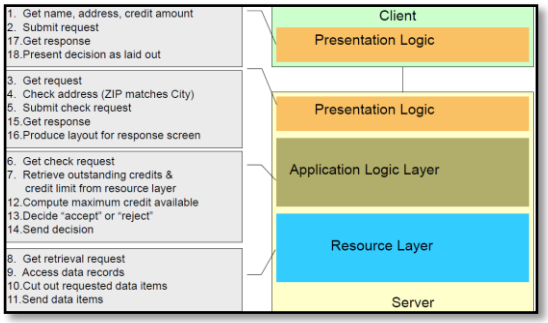


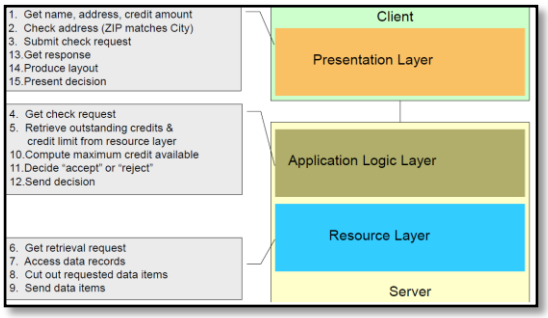
- A “client/server” system results in cutting an application horizontally into two parts
  - The “upper” part is called client
  - The “lower” part is called server
- Depending on the location of the cut, different topologies (with pros and cons) result
- The problem is always **“where to put which functionality”** based on non-functional goals
  - Performance, reuse, manageability, costs, ..

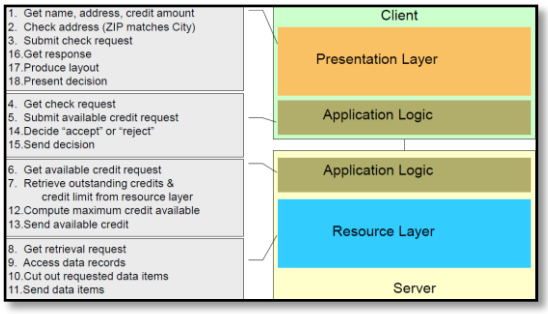
Limitations:

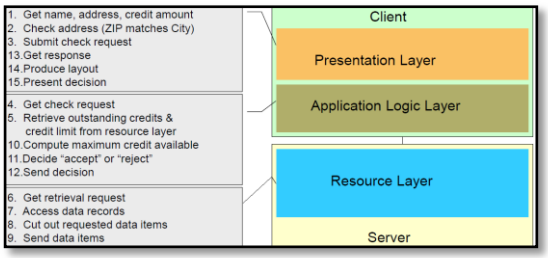
- Often, a client needs to access many servers
  - I.e. multiple applications perhaps on different machines
    - So-called “islands of information”
- *Consequences*
  - Client must integrate different functions, e.g.: (from different [styles of – e.g. class lib & messages] APIs)
    - Combine, split, transform data,…
    - Consistency (data cleansing, transactions across apps,…)
  - Client must deal with heterogeneity
    - Synchronous, asynchronous APIs,…
    - Different formats (Java API, C# API,…)
    - Different authentication mechanism
- *Resulting problems*
  - Complexity
    - Client design becomes very difficult
  - Fragility
    - If one API changes, client has to be changed too
  - Performance
    - Client machine has only limited capabilities

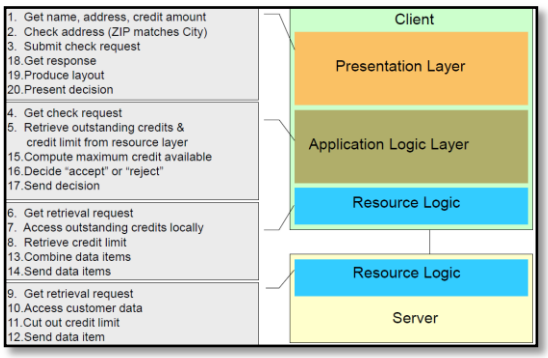
Cut	Beschreibung	Beispiel

1	• Browser – some presentation logic as
	  ○ applets, apps,… on client
	  ▪ Checking inputs (Before sending request to server)
	  ○ servlets on server
	  ▪ Generation of HTML (On sending response back)
	• X-Windows
	  ○ X-Display-Server runs on client
	  ○ X-Client runs on server
	• Fat client – some presentation logic as
	  ○ WIN GUI
	  ○ program on server
		

2	• Fat client model
	  ○ Windows GUI
	  ○ Terminal emulation
	• Browser with applets
		

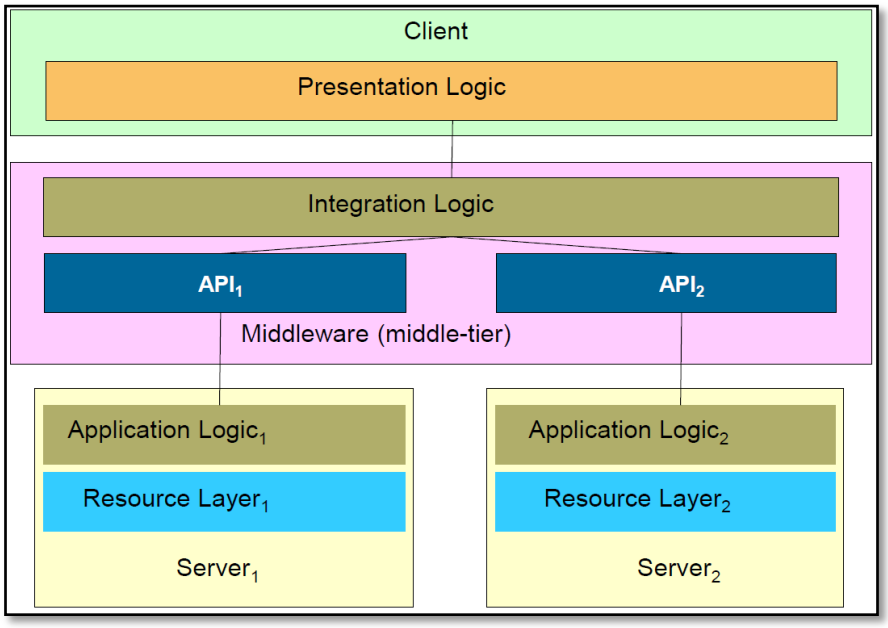
3	• Fat client model
	• Application logic split between client and server
	  ○ Application logic on client delegates remaining processing of logic to an application function on server, e.g.
	  ▪ A Java program on the client calls another Java program on the server
	  ▪ The client application part calls a transaction on the server
		

4	• Access via high-level interface to resources
	  ○ SQL (JDBC, ODBC,…)
	• This interface provides for location transparency of resources
	  ○ Application is not aware whether resources are on same machine or not
		

5	• Data caches
	  ○ Entity EJBs, Persistent EJBs (JPA)
	  ○ Web caches
	• Data replicas
	  ○ Lotus Notes, ObjectStore,…
	• Fragmented data
	  ○ Some data stored on client, some data stored on server
		

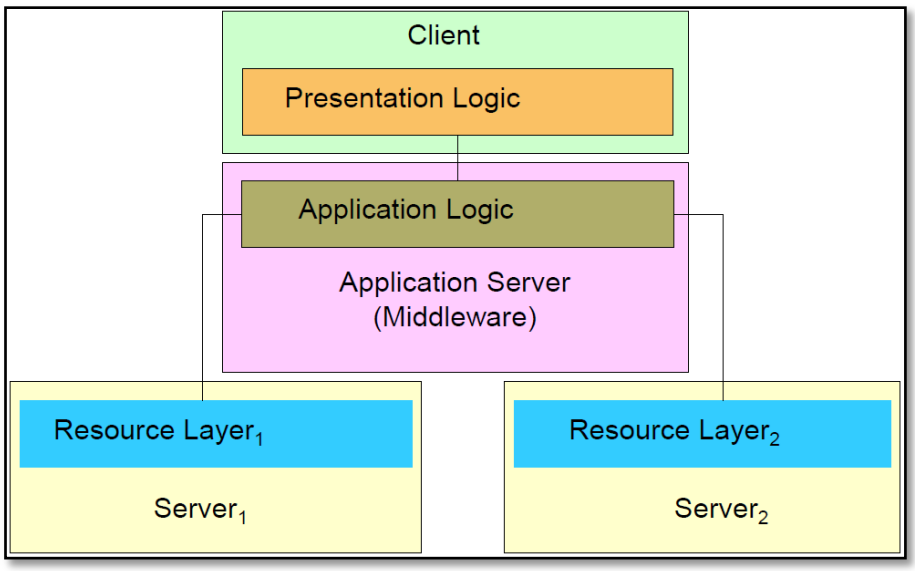
3.2.3 Middleware

- In general, middleware is used to build 3-tier structures
- The middleware runs the integration logic (!)
- Client (presentation logic) simply invokes integration logic to access (federated) functionality
    - Again, this can be via an API
- Thus, from a client perspective integration logic is the application logic
- From an integration logic perspective application logic is the resource layer
- Typically, the layers are distributed across different machines taking advantage of the complete modularity of the design
- The middleware runs on the **middle-tier**



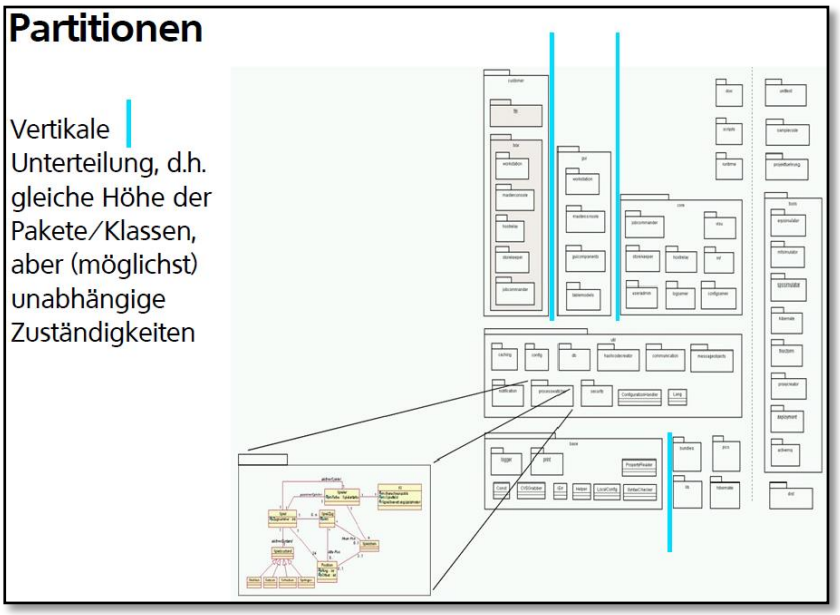
3.2.4 3-Tier Structure And Application Logic

Since there is no distinction between application logic and integration logic at all from a client perspective why not using middleware also to implement application logic that doesn‘t deal with integration? ***=> Application Server***

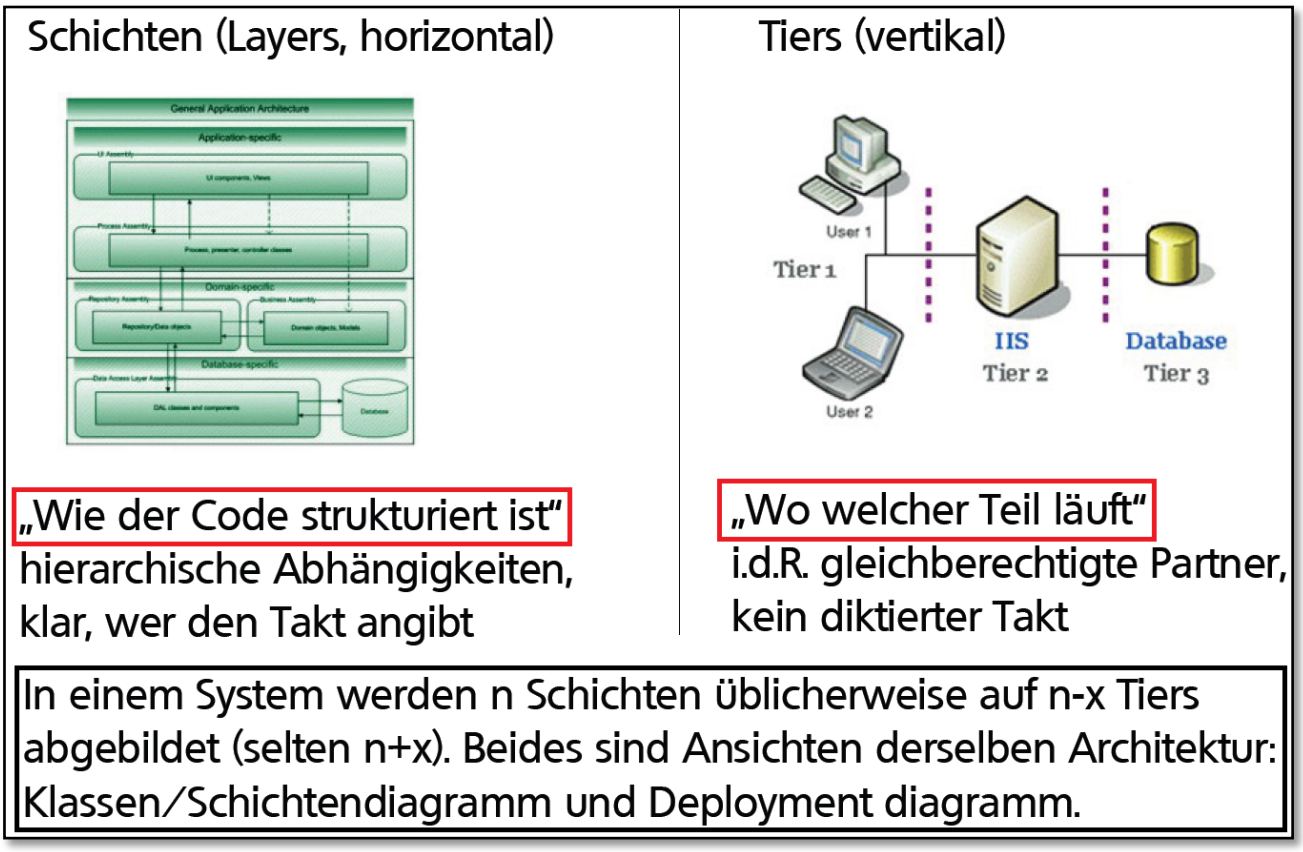


3.3 Schicht/Layer, Partition, Tier

Partitionen



Layers & Tiers



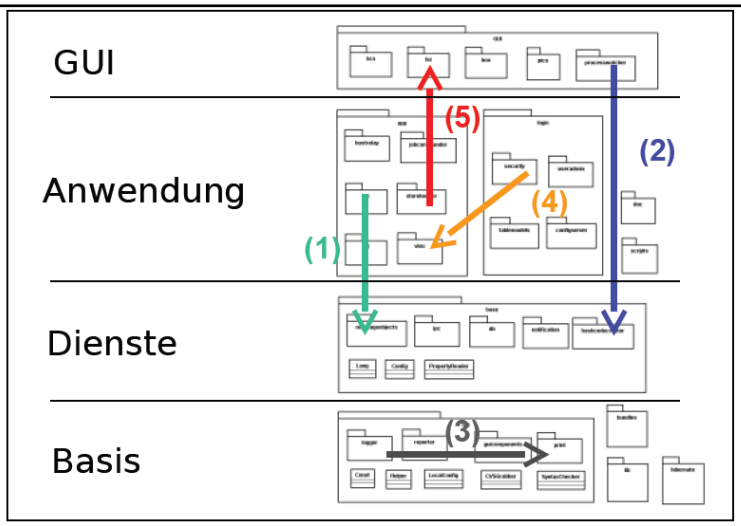
Kommunikation in Layers & Tiers

Layers: Aufrufe zwischen Schichten sind synchron

Tiers: Interfaces zwischen Tiers sind asynchron, d.h. potentiell langsamer und „teurer"

UNTERSCHIED: Logischer Viewpoint (Komponenten/Klassen) vs. physischer Viewpoint (Netz/Server)

3.4 Regeln für Zugriffe („nie von unten nach oben, nicht quer,..“)



Aufrufe:

(1) … von oben auf die nächste darunterliegende Schicht sind immer OK

(2) … nach unten, die *eine* Schicht überhüpfen sind oft auch OK

(3) … innerhalb einer Schicht und Partition sind OK, sollten aber minimiert werden

(4) … in einer Schicht quer über Partitionen sollten dringend vermieden werden

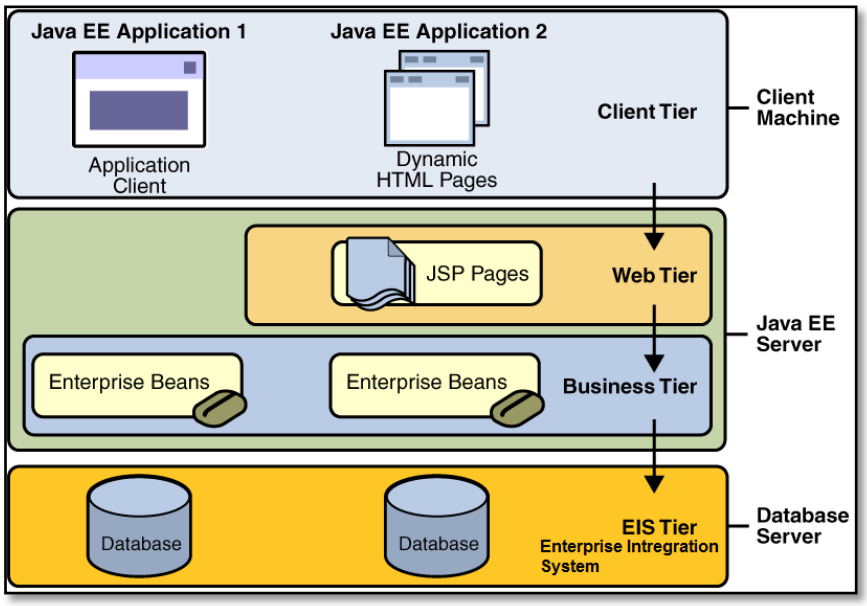
(5) … NIE von unten nach oben, ausser callbacks (z.B. Observer pattern)

3.5 Separation of Concerns

- Klare Aufteilung der Zuständigkeiten:
  - Immer nur ein Thema pro Klasse/Methode, das ergibt dann auch die erwünschte hohe Kohäsion.
  - **Kopplung:** Aufrufe von einer Klasse zur anderen (z.B. „feature envy“), von einem Package zum anderen
  - **Kohäsion:** Zusammenhalt innerhalb einer Klasse weniger auch: Zusammenhalt innerhalb eines Packages („warum sind wir zusammen, was ist der gemeinsame Zweck?“)
- Präzisierung:
  - immer nur ein Thema pro Methode, alles andere ist Unsinn, keine Ausnahmen
  - manchmal mehr als ein Thema pro Klasse (weil man öfter mal eine Klasse mit unterschiedlichen Fähigkeiten ausstatten will), aber aufpassen, Klasse nicht überladen.

4. JEE Java Enterprise Edition

4.1 Structure of JEE Application

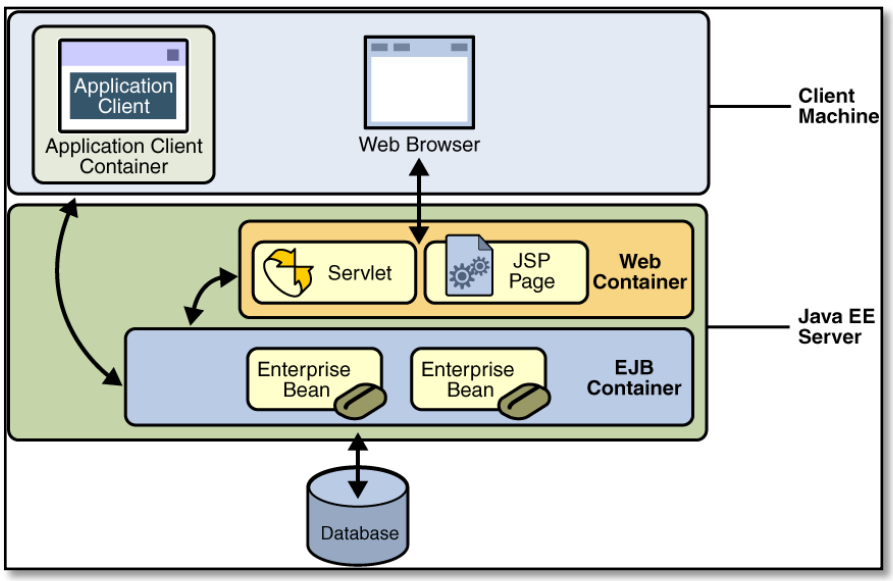


4.1.1 *Notation of a „Container“*

- …is the „place“ where components are active
- …is an interface between a component and the lowlevel platform-specific services
  - Persistence, transactions, security, naming, remoting,…
- Container provides services tailored toward components need based on **deployment descriptor (DD) or annotations**
  - Persistence, transactions, security, naming, remoting,…
- Container provides resource management
  - Pooling of components, activation & passivation,…

Welche Arbeiten übernimmt ein Container?

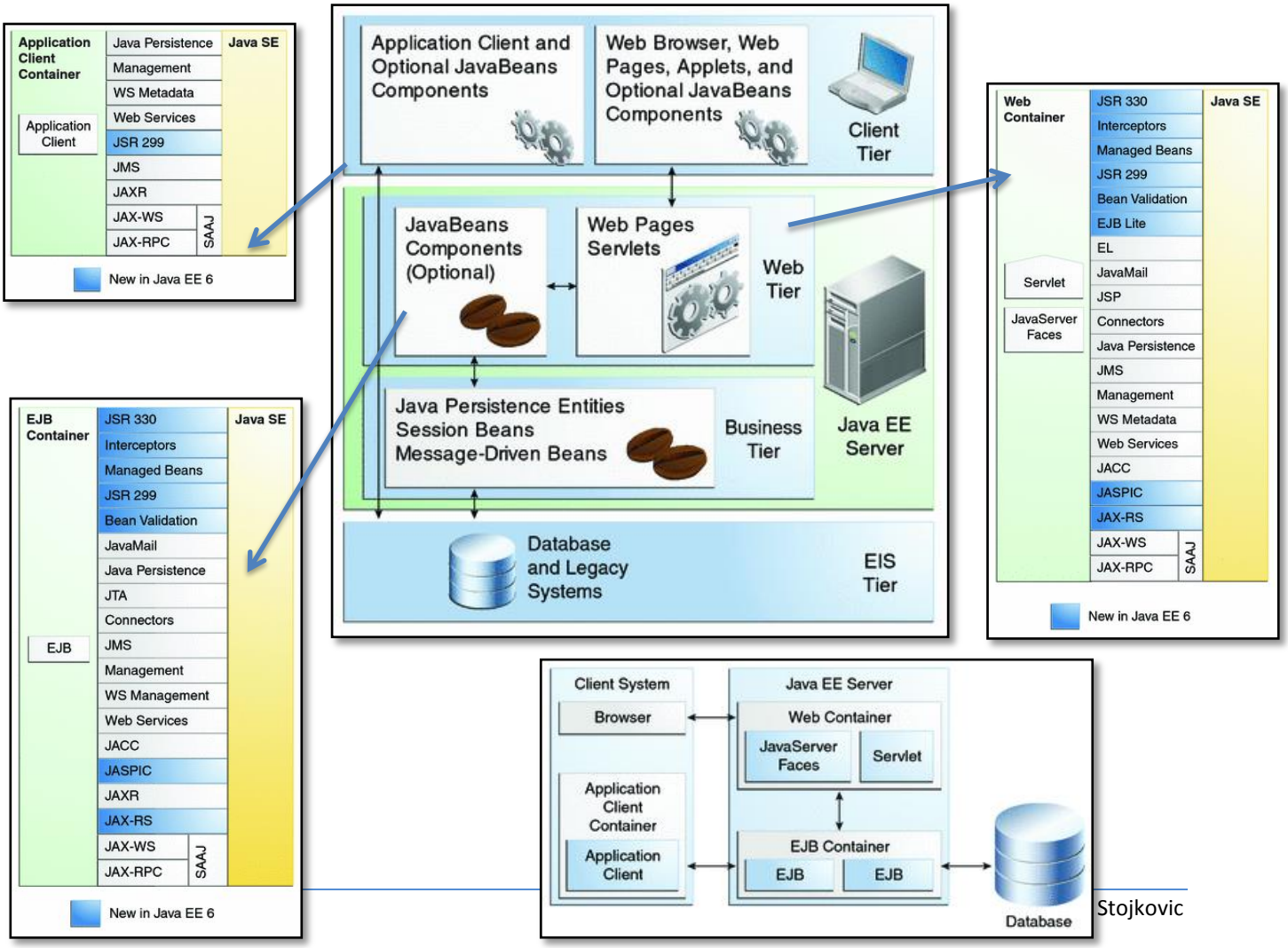
- Lifecycle-Management der Komponenten (Start, Stop)
- Load Balancing (Instance Pooling, Caching)
- Transaction Management
- Remoting, Data Access
- Security
- Logging



4.2 JEE API's

Name	Beschreibung	Layer
Enterprise Java Beans (EJB)	• z.B. für Berechnungen und Zugriff auf den DAL als Stateless Session Bean kapseln • Kapselung und Pooling von Business Logik Objekten • Remote Zugriff auf Business Logik Objekte via RMI/IIOP • Vier Arten von Beans: - Stateless Session Beans - Stateful Session Beans - Entity-Beans (persistente Datenhaltung) - Message-Driven-Beans (Informationen via Nachrichten austauschen)	BLL
Java Servlets	• Extend WebServer functionality • Used to built Web applications • Ein Servlet pro UseCase, als Front Controller oder Page Controller • Entgegennahme von und Antwort auf HTTP-Requests	PL
Java Server Pages (JSP)	• Mix static content (e.g. XML, HTML, WML) with dynamic content (Java code encapsulated in JSP tags) • Used to create dynamic Web pages • Eingabe-Formulare, Query Ergebnisse im WebChannel (View in MVC Pattern)	PL
Java Server Faces (JSF)	• User interface framework for building Web applications • Wie JSP, jedoch mehr Möglichkeiten.	PL
Java Message Service (JMS)	• Standard for messaging using Java • API to JMS-compliant message oriented middleware • Java Message Service (JMS) ist eine Programmierschnittstelle (API) für die Ansteuerung einer Message Oriented Middleware (MOM) zum Senden und Empfangen von Nachrichten aus einem Client heraus, der in der Programmiersprache Java geschrieben ist. JMS hat das Ziel, lose gekoppelte, verlässliche und asynchrone Kommunikation zwischen den Komponenten (z.B. Web-Container und EJB-Container) einer verteilten Anwendung zu ermöglichen. • Eingehende Messages sind im PL; Ausgehende im DAL	PL oder DAL
Java Transaction API (JTA)	• Java Transaction API (JTA) specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications. • BLL-Layergrenze eignet sich gut, um DB-Transaktionen zu öffnen und schliessen. • z.B. 2-Face-Commit	**DAL: (OR-Mapper)** **BLL: POJO's** **PL: Servlet doGet/Post**
Java Mail API	• API to send E-Mail from Java • Web-Technologie, z.B. um Kunden über Abschluss der Offerten Stellung zu informieren.	PL
Java API for XML Processing (JAXP)	• Part of Java SE • Supports processing of XML documents using DOM, SAX, StAX and XSLT • Konfigurationsfiles, XML als Messageaustausch	Alle

Java Architecture for XML Binding (JAXB)	• Bind XML schema to an object representation in Java	Alle
Java API for XML Web Services (JAX-WS) (SOAP/WSDL)	• Nicht Kern-BLL wegen der Abhängigkeit auf HTTP; Web Services sollten als RemoteFacades definiert werden oberhalb des eigentlichen Domain Models (Kern-BLL)	PL oder eigener Service-Layer
Java API for RESTful Web Services (JAX-RS) (REST)	• Gleiche Abstraktionsebene und HTTP-Abhängigkeit wie JAX-WS	PL oder eigener Service-Layer
Java Database Connectivity API (JDBC)	• Interface to access relational databases • Service provider interface to plug databases into JEE • Zugriff auf DB. • Datenbank-Queries	DAL
Java Persistence API (JPA)	• Object-Relational mapping between Java objects and relational databases • Query language (z.B. Hibernate) • Bindeglied BLL zu DAL, Bsp. Mapping Customer-Objekte auf Customer-Tabelle in DB	BLL, DAL
Java Naming and Directory Interface (JNDI)	• Komponenten- und Ressourcenlookup • Mapping logische Namen auf Objektreferenzen (Mapping logischer Name auf EJB)	Alle
Java Authentication (wer bin ich?) und Authorization (auf was darf ich zugreifen?) Service (JAAS)	• Security-Prinzipien “so früh wie möglich”, “tief”; Security ist Cross-C utting Concern, der alle Schichten angeht	Alle, Speziell PL und BLL-Eingang



5. Applikationsserver JEE Middleware

5.1 Definition und Tenets

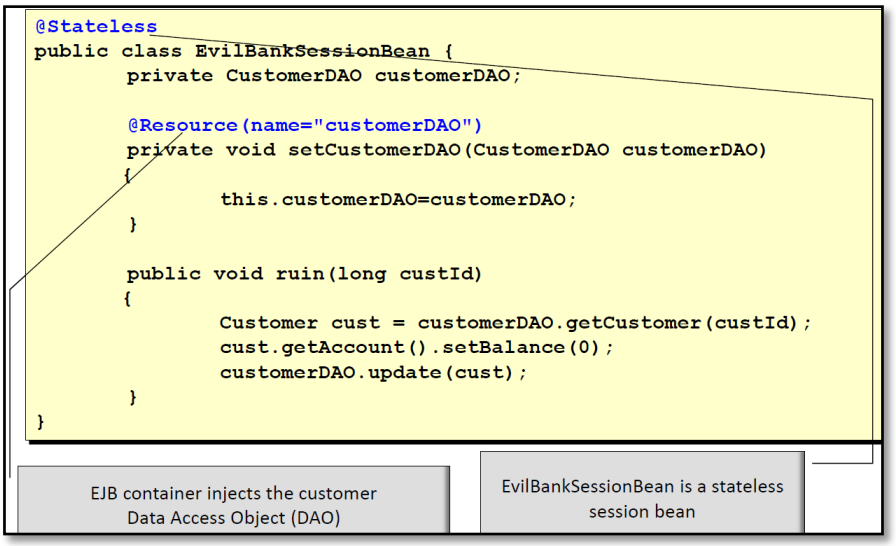
- Offers a managed, robust, reliable and scalable server side execution infrastructure that simplifies the construction and deployment of business components and applications
- Application developer can focus on developing business logic when programming. Application server declaratively inserts important system level services
    - Database connectivity and connection pooling
        - Connection Pooling: Es werden mehrere Connections vorbereitet, dadurch skalierbar.
    - Transaction support
    - Clustering support
        - Clustering: Es werden mehrere Server miteinander verbunden.
    - Authentication / Authorization

Weitere wichtige Definitionen:

- Vision of component-based development supported by middleware
    - Let the middleware do the hard work, so programmer can focus on domain-specific features (e.g. credit risk scoring algorithms, order processing)
    - Here: Enterprise = an organization with advanced NFRs/QASs regarding mission-critical application software (e.g. performance, availability)
- **Tenets (dt. Grundsätze)** for JEE middleware (application server):
    - Separation of concerns (middleware vs. application components)
    - Information hiding (with local and remote Java interfaces)
    - Shared services, resource pooling (e.g. database connections)
        - E.g. role-based application security ensured/taken care of by container
        - E.g. transaction management
        - E.g. logging
- Declarative configuration (instead of imperative programming)
    - e.g. cache sizes, resource pooling (in response to NFRs/QASs)
- Portability (avoid vendor lockin)

5.2 Annotations and Deployment Descriptors

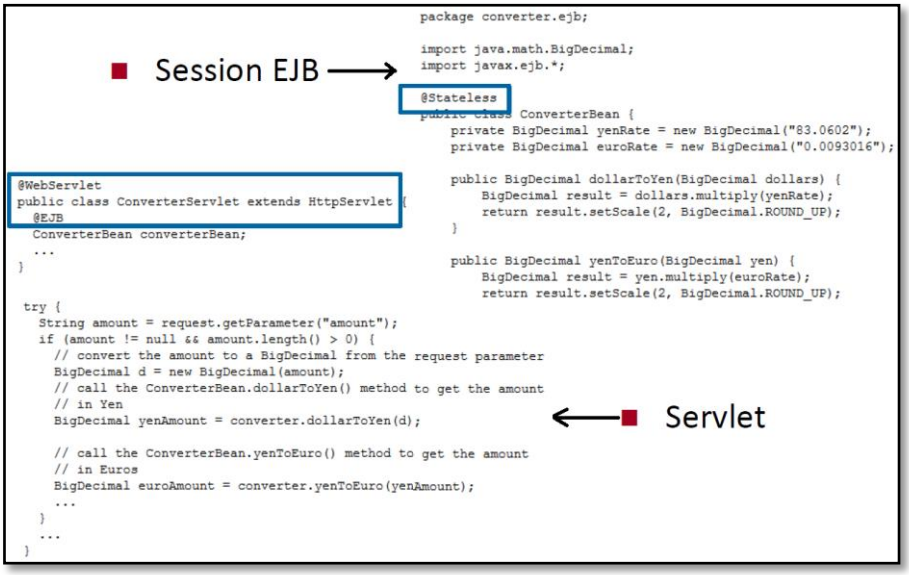
- **Annotation:** Metadata placed directly in Java source code
  - Enables declarative programming, aspect-oriented programming, dependency injection, …
- Annotations can be applied to
  - packages, classes, constructors, methods (including their parameters and return type) or fields
- Annotations can be compiled into byte code
  - Can be used during build time (e.g. for code generators, static analysis, aspect oriented programming (AOP) weaver)
  - Can be used during runtime (e.g. for containers, dependency injection, exposing Web Services, declarative transactions)



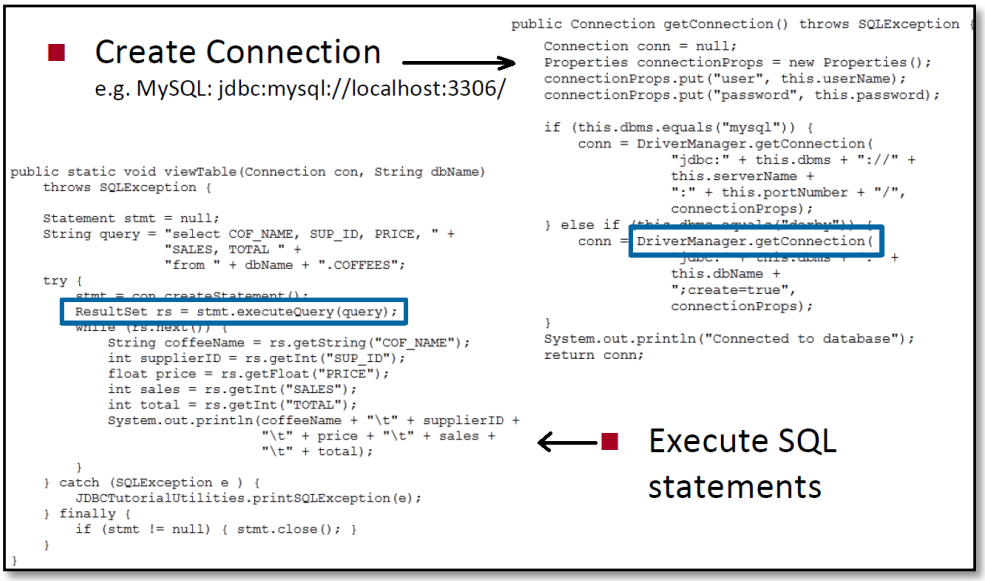
5.2.1 *Vergleich Deployment Descriptor und Annotations*

- Deployment descriptor is separate artifact (XML)
  - Pro: Modifications do not require to manipulate the component (aka bean) code
    - E.g. changing security is done by security officer but not by programmer
  - Pro: All QoS related aspects are in a single place
  - Con: Level of indirection is a bit more complicate
- Annotation scattered within component codes
  - Pro: Reduced number of artifacts
  - Con: QoS changes require code modifications
- **If you specify both, deployment descriptor overrides corresponding annotations**

5.2.2 „Hello World“ of JEE with Annotations

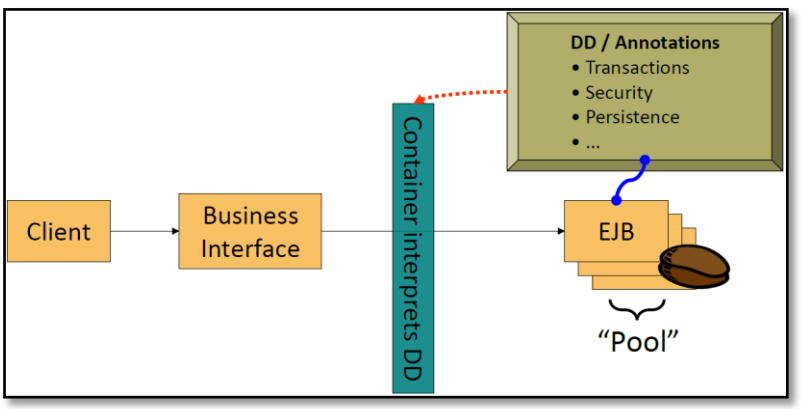


5.2.3 JDBC Basics



5.2.4 Quality of Service (QoS) Injection with Annotations and DD

- Client is not working directly with the EJB itself but with its Business Interface
  - An interface created that is a proxy of the EJB proper
- Why? Because this allows to fold in QoS as specified in the EJB’s Deployment Descriptor (DD) or annotations
- QoS policies (settings) derived from NFRs/QASs elicited during requirements engineering



5.2.5 Security of Enterprise Beans

- Security can be specified with annotations or deployment descriptors
  - Access to beans or users via roles
  - Methods to retrieve the roles of a caller from program code (`getCallerPrincipal(), isCallerInRole()`)
- Example: Allowing only a specific role to access a method in a Session Bean

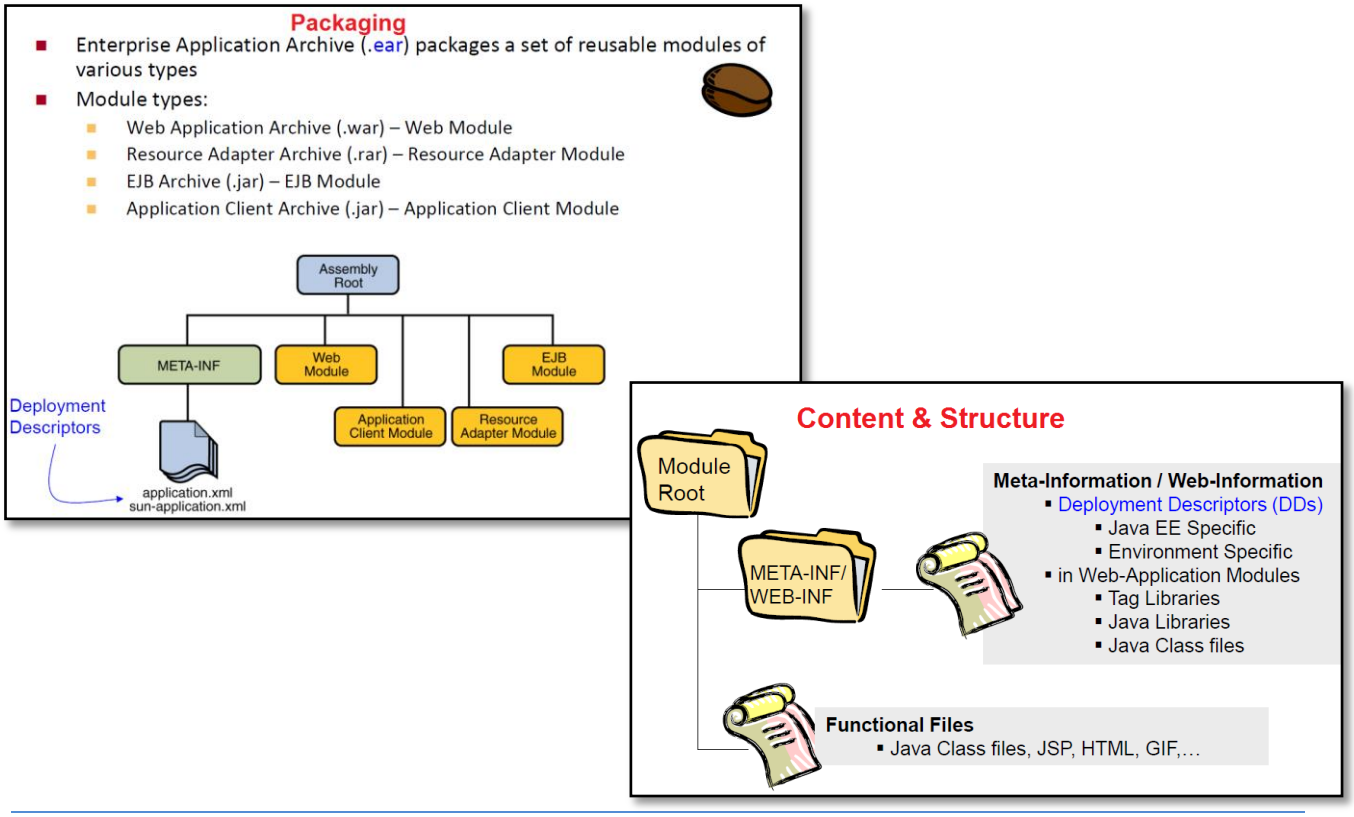
```
@Remote
public interface BankAccount {
        @RolesAllowed("AccountOwner")
        public void deposit(…) throws AccountException;
        public void credit(…) throws AccountException;
}
```

5.2.6 Transaction in Enterprise Beans

- Container-Managed
  - Transaction Demarcation Transaction boundaries are set by container and are configured via transaction attributes
    - Required – uses client's transaction or starts new one
    - RequiresNew – starts a new transaction
    - Mandatory – has to be called with a transaction
    - NotSupported – does not use a transaction
    - Supports – can be called with or without a transaction
    - Never – cannot be called with a transaction
- Bean-Managed Transaction Demarcation Programmer handles transaction boundaries in the code via the JTA API

```
@TransactionAttribute(REQUIRED)
@Stateful
public class BankAccountBean implements BankAccount {
        public void deposit(…) throws AccountException{…}
        public void credit(…) throws AccountException{…}
}
```

5.2.7 JEE Modul Content & Structure AND Packaging



6. BLL Bussines Logic Layer

6.1 BLL Aufgaben gemäss CRC-Karte

6.1.1 CRC Components, Responsibilities, Collaborations (CRC) Card Template

Komponente: [Name der Komponente]	
Verantwortungsgebiet: • [Wozu ist die Komponente fähig? (wird ein Services angeboten?)] • [Welche Daten bearbeitet die Komponente?] • [Wie hält die Komponente die System-Qualitäten ein?]	**Zusammenarbeit mit:** • [Wer ruft die Komponente auf? (gibt es Service Konsumenten?)] • [Wen ruft die Komponente auf um Ihre Verantwortungen zu erfüllen? (service providers?)] • [Gibt es externe Verbindungen (beide aktiv oder passiv)?]
Known uses (Implementationen): • [Welche Technologien, Produkte (kommerziell, open source) und interne Werte verwenden die Funktionalität der Komponente?]	

Gute Komponentennamen fördern das Verständnis

- Oft eine Metapher (wenn gut gewählt)
- Starke über schwache Semantik, Bsp. Browser (Duschkopf), Server

Auf Widerspruchsfreiheit achten (nicht unbedingt auf Vollständigkeit)

- Gute Komponenten-Beschreibungen sind SMART (wie Ziele)
- Zu jeder ausgehenden Kollaborationsbeziehung muss es bei einer anderen Komponente ein Pendant geben (Diensterbringung)

Einheitlichen Detaillierungsgrad durchziehen (weniger ist oft mehr)

- Zu genau – schwierig zu implementieren und zu ändern
- Zu unpräzise – kein zusätzlicher Wert zum Code, Implementierungen lassen sich nur schwer integrieren (Interoperabilitätsprobleme)

Component: [Name of Component]	
Responsibilities: • [What is this component capable of doing (services provided)?] • [Which data does it deal with?] • [How does it do this in terms of key system qualities?]	**Collaborations (Interfaces):** • [Who invokes this component (service consumers)?] • [Whom does this component call to fulfill its responsibilities (service providers)?] • [Any external connections (both active and passive)?]
Known uses (Implementations): • [Which technologies, products (commercial, open source) and internal assets realize the outlined component functionality (responsibilities)?]	

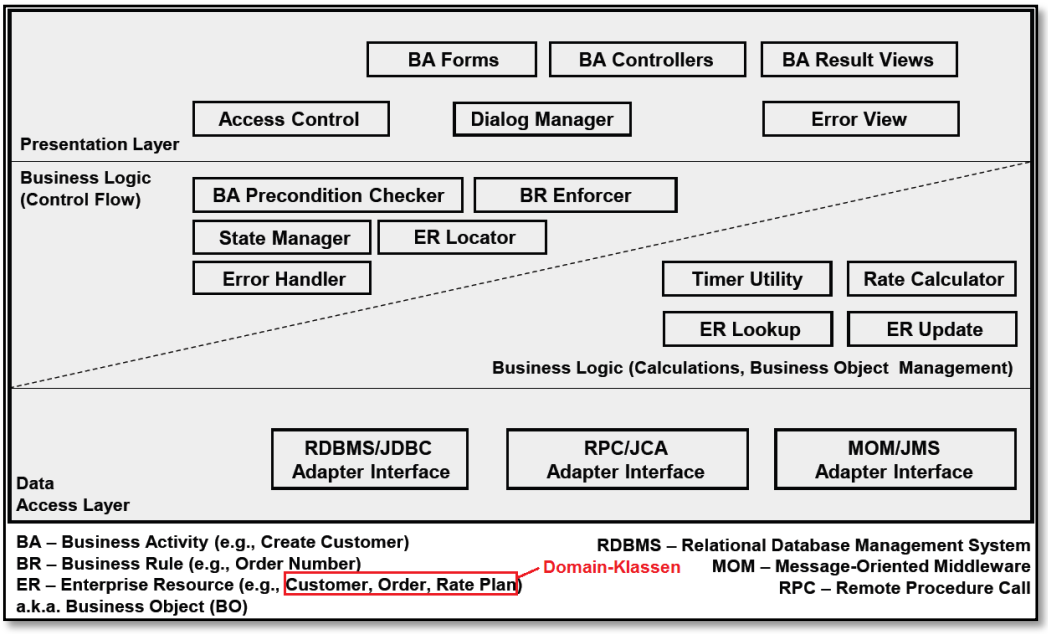
6.1.2 Beispiel anhand Proxy-Pattern

Component: Proxy Pattern (Class: Proxy)	
Responsibilities: • Liefert dem Client das Interface zum Original • Sorgt für einen sicheren, effizienten und korrekten Zugriff auf das Original.	**Collaborations (Interfaces):** • Original
Known uses (Implementations): • Lazy-Loading • Remote-Proxy: Lokaler Stellvertreter für Objekt in anderem Adressraum • Protective-Proxy: Zugriffsrechte kontrollieren und evtl. verweigern	

6.1.3 *Beispiel anhand vom BLL*

Component: Business Logic Layer (all Components in this Container)	
Responsibilities: • Receives service requests • Authenticates request originator, authorizes service request • Keeps track of processing state (workflow, not page flow!), manages transaction boundaries • Accesses business objects • Performs calculations • Updates business objects and application state • Sends response • Logs layer activities in audit trail	**Collaborations (Interfaces):** • Presentation layer • Data access layer • Cross-cutting infrastructure features • Component container (Inversion of Control)
Known uses (Implementations): • Service layer, process layer in "T" case study • BLL in PQG case study • domain.model package in DDD Sample • HSR enterprise applications such as unterricht.ch, AVT	

6.2 Logische Komponenten im BLL



6.3 Inversion of Control Pattern (Hollywood-Principle)

Inversion of Control ist das Prinzip/Pattern, welches beschreibt wie ein z.B. ein Framework arbeitet (Hollywood-Prinzip) **oder** wie unter anderem Abhängigkeiten, Testbarkeit, Kopplung, Erweiterbarkeit, etc. umgesetzt werden sollten. **Wer spricht wen an!?** Wobei Dependency Injection eine konkrete Implementierung des IoC Prinzip/Patterns ist.

6.3.1 *Inversion of Control Detail*

Problem[1]

You have classes that have dependencies on services or components whose concrete type is specified at design time. In this example, ClassA has dependencies on ServiceA and ServiceB. Figure 1 illustrates this. (*ClassA has dependencies on ServiceA and ServiceB)*

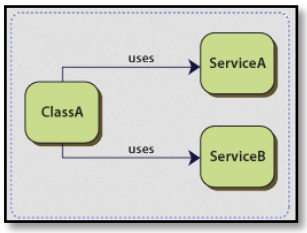


Figure 1

- To replace or update the dependencies, you need to change your classes' source code.
- The concrete implementations of the dependencies have to be available at compile time.
- Your classes are difficult to test in isolation because they have direct references to dependencies. This means that these dependencies cannot be replaced with stubs or mocks.
- Your classes contain repetitive code for creating, locating, and managing their dependencies.

Forces

Any of the following conditions justifies using the solution described in this pattern:

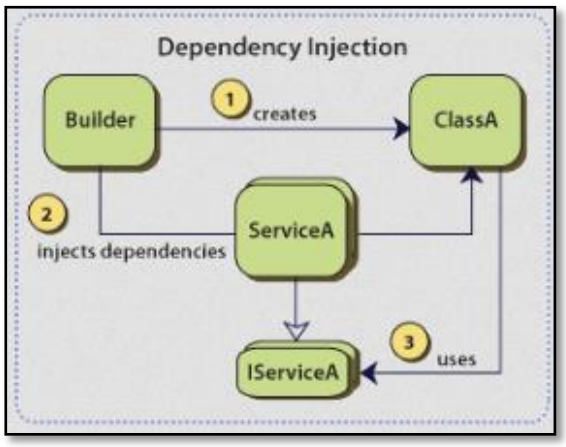
- You want to decouple your classes from their dependencies so that the dependencies can be replaced or updated with minimal or no changes to your classes' source code.
- You want to write classes that depend on classes whose concrete implementations are not known at compile time.
- You want to test your classes in isolation, without using the dependencies.
- You want to decouple your classes from being responsible for locating and managing the lifetime of dependencies.

Solution

Delegate the function of selecting a concrete implementation type for the classes' dependencies to an external component or source.

Implementation Details

Siehe “Dependency Injection”



6.3.2 *Dependency Injection*

[1] http://msdn.microsoft.com/en-us/library/ff921087.aspx

6.4 Komponente vs. Service (nach Fowler)

6.4.1 *Components*

- A self-contained shrink-wrapped piece of software (i.e. a marketable entity!)
- It provides a subset of functions needed in a particular business domain
- Must be augmented by functionality (of other components) to result in complete application
- A software artifact with immediate business meaning (then often called *business object*)
- It is **not** a stand-alone application
- It has an identity but cannot be instantiated (unlike class) **(kein Konstruktor)**
- It can be manipulated in a visual builder
    - Customization, augmentation, ...

Beispiele (Komponenten kommen aus dem Domain Model)

- Banking
    - CurrentAccount, AccountStatement, CreditRequest,...
- Finance
    - Stock, BlockOfShares, ShareHolder, ...
- Telco
    - PhoneCall, Charge, ConnectionStatement,..

6.4.2 *Komponente (Nach Fowler)*

"I use component to mean a glob of software that's intended to be used, without change, by an application that is out of the control of the writers of the component. By 'without change' I mean that the using application doesn't change the source code of the components, although they may alter the component's behavior by extending it in ways allowed by the component writers."
(Inversion of Control / Dependency Injection)

6.4.3 *Service (Nach Fowler)*

"A service is similar to a component in that it's used by foreign applications. The main difference is that I expect a component to be used locally (think jar file, assembly, DLL, or a source import). A service will be used remotely through some remote interface, either synchronous or asynchronous (e.g. Web service, messaging system, RPC, or socket.)"

6.4.4 *Zusammenhang Klasse, Komponente und Service*

Jede Komponente besteht aus mind. einer Klasse, mehrere Klassen bilden eine Komponenten. Komponenten welche Remote erreichbar sind oder einen Service zur Verfügung stellen, sind Services.

=> Komponenten gruppieren Klassen, haben eine Identität und stellen Services über Interfaces bereit

6.5 Domain Logic Patterns (Transaction Script, Domain Model, Table Module)

How do I Structure my domain Logic?

- The logic is simple -> Transaction Script
- The logic is complex -> Domain Model
- The logic is moderate (mässig) and there are good tools around *Record Set (Pattern)* -> Table Module

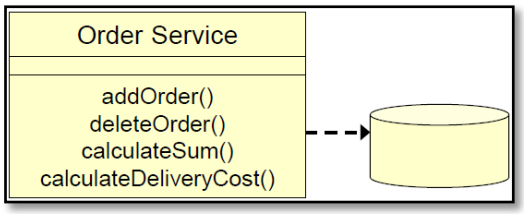
6.5.1 *Transaction Script*

"Organizes business logic by procedures where each procedure handles a single request from the presentation."

- Transaction Script ist im Wesentlichen eine Prozedur, welche die folgenden Aufgaben durchführt
  - Eingaben von der Präsentation entgegennehmen,
  - die Eingaben prüfen und
  - Berechnung durchführen.
  - Daten in die Datenbank schreiben und Operationen von anderen Systemen aufrufen
- Jede Aktion ist normalerweise eine Prozedur
- Procedure takes input, processes it, accesses data in database, invokes other services
- For each action of the user you need to provide a Transaction Script (Best way: A Java-Class for one Transaction Script)

Beispiel: Verkaufssystem hat für folgende Aktionen jeweils ein Transaction Script

- Checkout, Einfügen von Waren in den Warenkorb, Anzeige des Lieferstatus, etc..
- **IN JEE: message driven bean oder stateless bean**



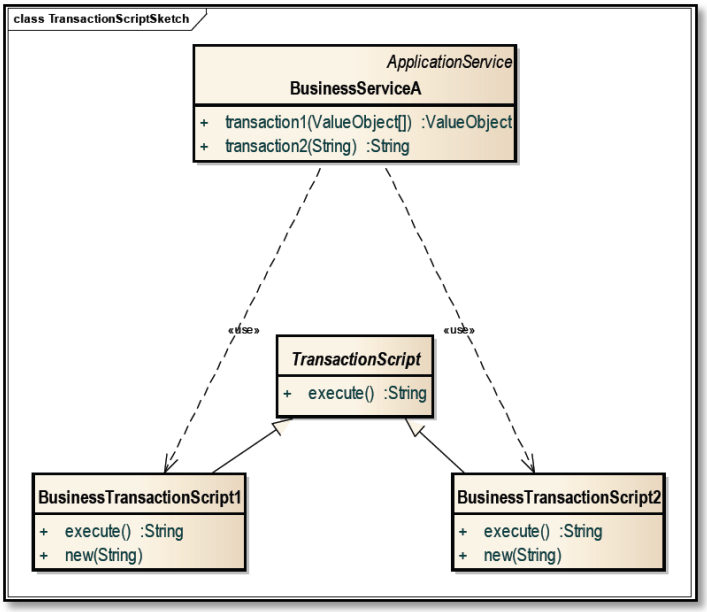
Advantages:

- Simple to understand
- Works with simple data source layer, e.g. Row Data Gateway
- Easy for transactions: on startup open transaction, on end close it

Disadvantages:

- Not adequate for complex domain logic

  => Code duplication in Transaction Scripts with similar actions

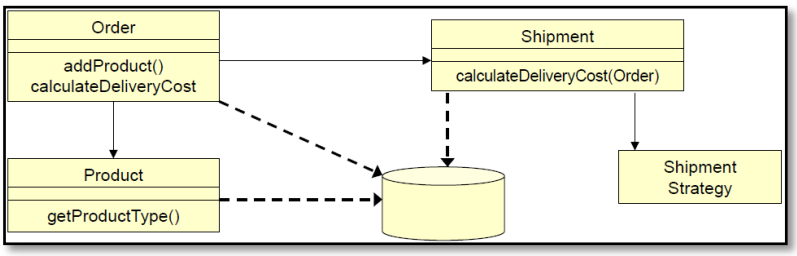


6.5.2 *Domain Model*

"An object model of the domain that incorporates both behavior and data."

- Domain Model for complex domain logic
  - Object-oriented way
  - Define a class model for the application domain
  - Logic for validation and calculation will be handled by appropriate object

Example: the calculation of the shipment will be handled by the shipment object



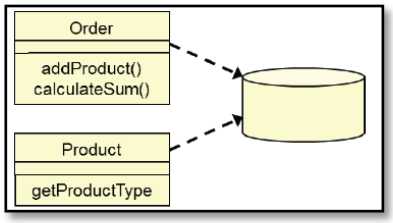
6.5.3 *Unterschied Domain Model und Transaction Script*

- Assume different calculations based on a product type
- A Transaction Script has to do all the work
  - determine product type, get the product and shipment type, calculate result and write to database
- In Domain Model we delegate the work to the product
  - The product object is responsible for work
    => May delegate work to another object (e.g. shipment object)
- **Advantage of Domain Model**
  - Handle complex domain logic
    - Easy add new product types or shipments (shipment strategies)
- **Disadvantage**
  - More complex data source layer
    - Mapping of domain objects to relational database (see *Data Mapper*)
  - Unfamiliar usage
    - Each object keeps domain logic

6.5.4 *Table Module*

"A single instance that handles the business logic for all rows in a database table or view."

- Similar to Domain Model with classes for domain objects
- **Difference:** Domain Model holds one instance for each object in the domain and Table Module has only one instance for a class of objects
  - **Example: In a Domain Model all products are available as separate instances, whereas in Table Module there is only one instance that represents all products (object often represents database table)**
- Table Module uses Record Set pattern for data handling
  - Issue query to database, create record set, add objects to record set
  - Get object instance with object id



- Better than Transaction Script, because organized logic around tables, but no finer grained structuring possible (like with inheritance in the objectoriented Domain Model)
- Often used, because many GUI environments are used to result sets, e.g. provided by SQL

6.5.5 *Auswahl des geeigneten Patterns für die Domain Logic*

Choice based on complexity of domain logic:

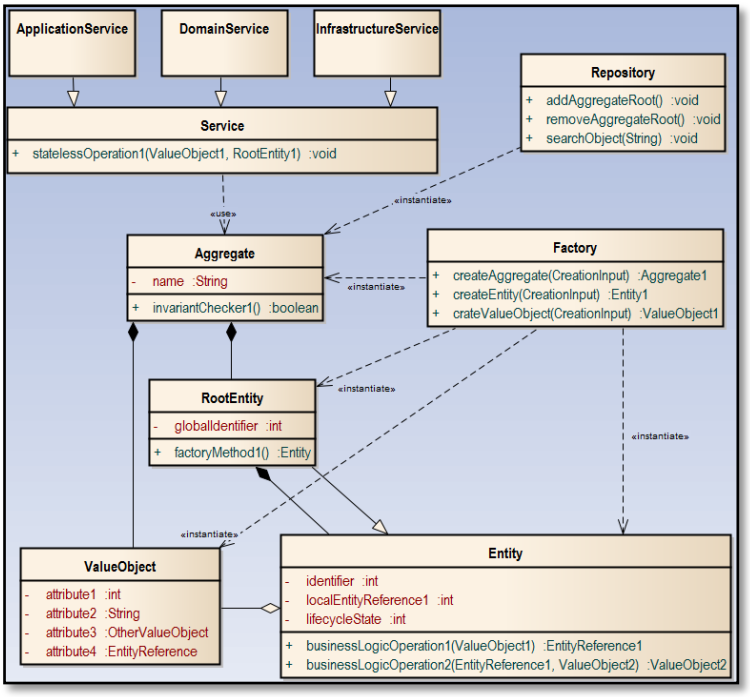
- With increasing complexity of the domain logic a **Domain Model** or a **Table Module** are good choices
- The **Table Module** depends much on the support of a **Record Set** structure and has also drawbacks for very complex domains
- For simple domain logic a **Transaction Script** is more efficient, foregoing (vorangehend) the complexity of the data source

=> It is hard to measure the complexity of domain logic

- Requirements engineering required (domain expertise)

6.6 DDD Domain Driven Design verfeinert Domain Model

Domain-Driven Design (DDD) ist eine Herangehensweise an die Modellierung komplexer objektorientierter Software. […] Domain-Driven Design ist nicht nur eine Technik oder Methode. Es ist viel mehr eine Denkweise und Priorisierung zur Steigerung der Produktivität von Softwareprojekten im Umfeld komplexer fachlicher Zusammenhänge.[…] Domain-Driven Design ist selbst unabhängig von Programmiersprachen, Tools und Frameworks.



6.6.1 *Entitäten (Entities, Reference Objects)*

- Hat ID
- Hat State
- Hat Verhalten

Objekte des Modelles, welche nicht durch ihre Eigenschaften, sondern durch ihre **Identität** definiert werden. Beispielsweise wird eine Person meist als Entität abgebildet. Eine Person bleibt somit dieselbe Person, wenn sich ihre Eigenschaften ändern; und sie unterscheidet sich von einer anderen Person, auch wenn diese dieselben Eigenschaften hat. Entitäten werden oft mit Hilfe von **eindeutigen Identifikatoren** modelliert.

Root-Entity:

- Ausserhalb des Aggregates sichtbar

Beispiele:

- Student, Person, Kunden, Velo im Veloshop, Log-Entry, Stundenplan,…

6.6.2 *Wertobjekte (Value Objects)*

- Kein Verhalten

Objekte des Modelles, welche **keine konzeptionelle Identität** haben oder benötigen und somit allein durch ihre **Eigenschaften** definiert werden. Wertobjekte werden üblicherweise als unveränderliche Objekte (**immutable objects**) modelliert, damit sind sie wiederverwendbar (vgl. **Flyweight**) und verteilbar. Value Objects können nicht angefasst werden und ändern sicht nie.

Beispiele:

- Color.blue, String, Enum, Integer, Money,…

6.6.3 *Aggregate (Aggregates)*

Aggregate sind **Zusammenfassungen von Entitäten und Wertobjekten** und deren **Assoziationen** untereinander zu einer gemeinsamen **transaktionalen Einheit**. Aggregate definieren genau eine Entität als einzigen Zugriff auf das gesamte Aggregat. Alle anderen Entitäten und Wertobjekte dürfen von ausserhalb nicht statisch referenziert werden. Damit wird garantiert, dass alle Invarianten des Aggregats und der einzelnen Bestandteile des Aggregats sichergestellt werden können.

6.6.4 *Assoziationen (Associations)*

Assoziationen sind, wie bei UML definiert, **Beziehungen zwischen zwei oder mehr Objekten** des Fachmodells. Hier werden nicht nur statische, durch Referenzen definierte Beziehungen betrachtet, sondern auch dynamische Beziehungen, die beispielsweise erst durch die Abarbeitung von SQL-Queries entstehen.

6.6.5 *Fabriken (Factories)*

Fabriken dienen dazu, die **Erzeugung von Fachobjekten** in spezielle Fabrik-Objekte auszulagern. Dies ist sinnvoll, wenn entweder die Erzeugung komplex ist (und beispielsweise Assoziationen benötigt, die das Fachobjekt selbst nicht mehr benötigt) oder die spezifische Erzeugung der Fachobjekte zur Laufzeit ausgetauscht werden können soll. Fabriken werden üblicherweise durch erzeugende Entwurfsmuster wie **Abstrakte Fabrik**, **Fabrikmethode** oder **Builder** umgesetzt.

6.6.6 *Repositories*

Repositories **abstrahieren die Persistierung und Suche** von Fachobjekten. Mittels Repositories werden die technische Infrastruktur sowie alle Zugriffsmechanismen auf diese von der Geschäftslogikschicht (application layer/middle tier) getrennt. Für alle Fachobjekte, welche über die Infrastruktur-Schicht geladen werden, wird eine Repository-Klasse bereitgestellt, welche die verwendeten Lade- und Suchtechnologien nach aussen abkapselt. Die Repositories selbst sind Teil des Fachmodells und somit Teil der Geschäftslogikschicht. Sie greifen als einzige auf die Objekte der Infrastruktur-Schicht zu, welche meist mittels der Entwurfsmuster Data Access Objects, Query Objects oder Metadata Mapping Layers umgesetzt werden.

6.6.7 *Serviceobjekte (Services)*

Bei Domain-Driven Design werden Funktionalitäten, welche ein wichtiges Konzept der Fachlichkeit darstellen und konzeptionell zu mehreren Objekten des Fachmodelles gehören, als eigenständige Serviceobjekte modelliert. Serviceobjekte sind üblicherweise zustandslose (eng. *stateless*) und daher wiederverwendbare Klassen ohne Assoziationen, mit Methoden, die den angebotenen Funktionalitäten entsprechen. Diese Methoden bekommen die Wertobjekte und Entitäten übergeben, die zur Abarbeitung der Funktionalität notwendig sind.

6.7 Splitting Domain Logic

- Common practice is splitting domain logic in two layers
  - **Service Layer** as a pattern for a layer on top of
  - **Domain Model** or **Table Module**
- Presentation layer interacts with domain purely through Service Layer
  - Service Layer as API for application
  - Put transaction and security on Service Layer
  - Behavior implementation
    - Service Layer is a façade to underlying objects and delegate calls to lowerlevel objects for real behavior, provide API around use cases
    - Or Transaction Scripts inside the Service Layer (underlying domain objects are simple, e.g. Active Record as a representative of a database record/row with additional domain logic (operations))
    - Or mixed behavior implementation
      => controller-entity style, define some business logic in Transaction Scripts and the business logic that is common to all entities in a domain object

6.7.1 Service Layer Pattern

Problem:

- How to hide complex domain models from presentation layer?

Solution:

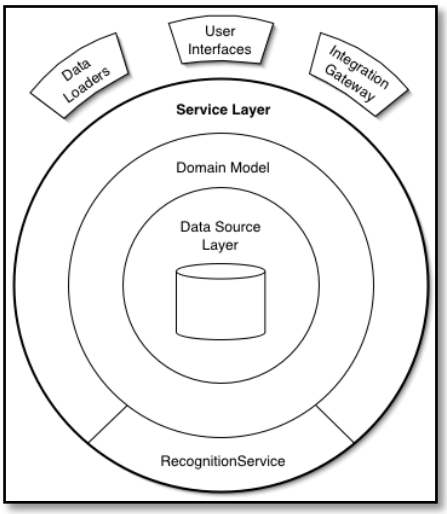
- Define an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation

Was kann der Service Layer noch tun?

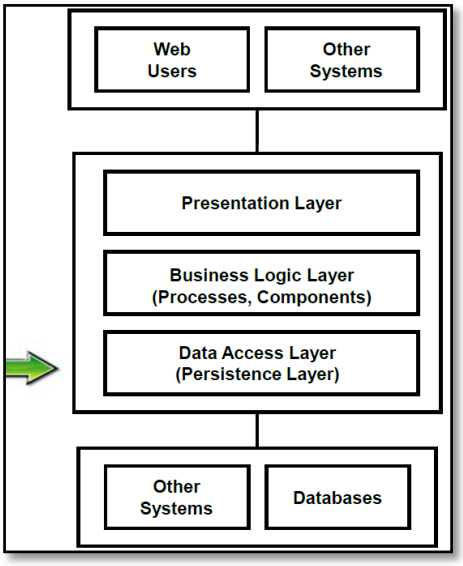
- Role-Based Access Control (taking data into account)
- Transaction Control, Business-Level Undo (Compensation)
- Activity Logging (e.g. Metering, Monitoring)
- Exception Handling

Wann ist es sinnvoll einen Service Layer einzuführen?

- z.B. bei viele Aktoren und Kanälen, langlaufenden Prozessen (Ausblick: Service Composition, Workflow Engines – SOA-Teil der VL)



7. DAL Data Access Layer (Persistance Layer)

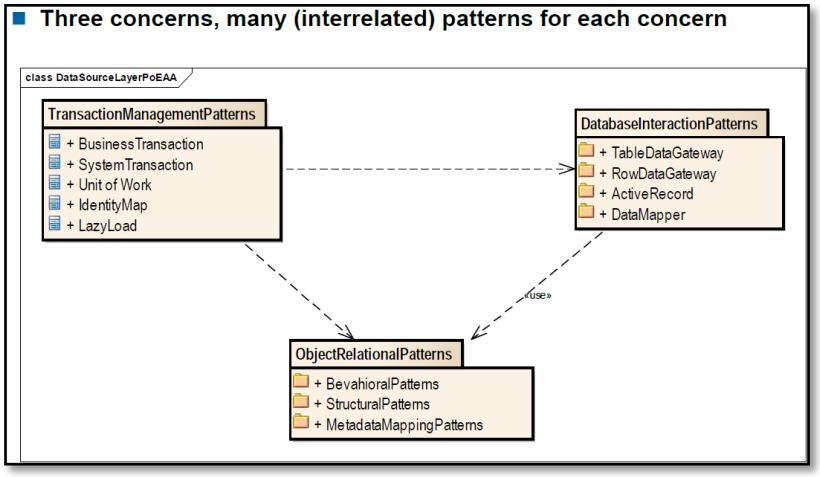


7.1 DAL Aufgaben gemäss CRC-Karte

Component: Data Access Layer (in PoEAA Layering)	
Responsibilities: • Wrap database structure and database API • Map classes and objects to tables and rows (with inheritance, relations) • Pool database connections • Cache query results , provide cursor/iterator for stepwise result processing • Support transaction management with/without two-phase commit (2PC)	**Collaborations (Interfaces):** • Business Logic Layer • Data storage, database management system (via network) • Other systems (e.g. via Java Connector Architecture, JCA)
Known uses (Implementations): • Databases are a dominant part of data storage infrastructure, relational databases are very common because of SQL standard • Java Persistence API (JPA) • Hibernate • Eclipse Link • Oracle TopLink	

7.2 DAL Data Access Layer Patterns im Überblick

Data Access Layer patterns describe how domain logic interacts with database.



7.3 DAL Patterns (Row/Table Data Gateway, Active Record, Data Mapper)

How do I interect with the database?

- I'm using *Transaction Script* -> **Row Data Gateway**
- I'm using *Transaction Script* -> **Table Data Gateway**

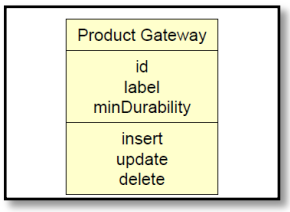
- I have a *Domain Model* that corresponds closly to my database tables -> **Active Record**
- I have a rich *Domain Model* -> **Data Mapper**

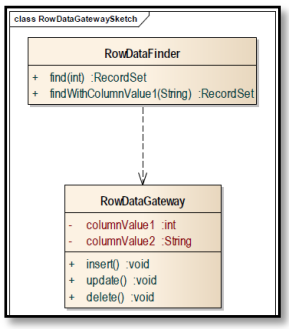
- I'm using *Table Module* -> **Table Data Gateway**

7.3.1 Row Data Gateway

„An object that acts as a Gateway to a single record in a data source. There is one instance per row"

- Klassen bilden den *Gateway* zur Tabelle in der Datenbank
- Der Rest der Anwendung darf nicht über SQL wissen
- Objektorientiertes Denken: **jede Zeile in einer Tabelle wird eine neue Instanz**
- Passt zum *Transaction Script*





7.3.2 Table Data Gateway

„An object that acts as a Gateway to a database table. One instance handles all the rows in the table."

- Pro Tabelle meist eine Klasse
- Eine Instanz pro Tabelle in der Datenbank
- *Table Data Gateway* stellt Methoden zu Verfügung, welche die Datenbank abfragen und ein *Record Set* zurückgeben.
- Da *Table Data Gateway* mit *Record Sets* arbeitet, passt es sehr gut für *Table Module*.
- Good place to wrap calls to stored procedures in database

7.3.3 Active Record

„ An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data."

- Ein Objekt enthält Daten sowie auch Verhalten (behavior)
- Die Daten werden meist auf einer Datenbank gespeichert
- Der Datenzugriff wird in die Domain-Object eingefügt
- Active Record ist ein Objekt, welches als Adapter (Wrapper) zu einer Zeile in einer Datenbanktabelle oder Datenbanksicht (View) dient.
- Der Adapter beinhaltet hierbei den Datenbankzugriff und Geschäftslogik für die Daten.
- Es handelt sich im Grunde um einen **Row Data Gateway**, welcher um die Geschäftslogik erweitert wird und deshalb sowohl Daten (Eigenschaften) als auch Verhalten (Methoden) implementiert.

7.3.4 Data Mapper

„A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself."

- Use Data Mapper to isolate Domain Model from database
- Use Data Mapper for complex Domain Models
- in-memory objects need NOT to know about database or query languages
- Most complicated of database mapping architectures, but O/R Mapping Tools available
- Handles all the loading and storing between database and Domain Model

7.3.5 Database Connections

- Connections sind sehr teure Aktionen
- Connections müssen gemanaged werden, d.h. z.B. wenn Connections nicht benötigt werden, dann müssen sie auch wieder geschlossen werden, bzw. muss dafür gesorgt werden, dass sie geschlossen werden

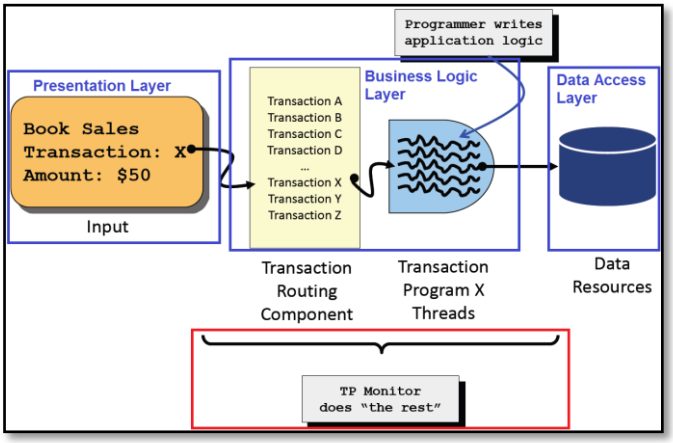
=> connection pooling

7.4 Transactions

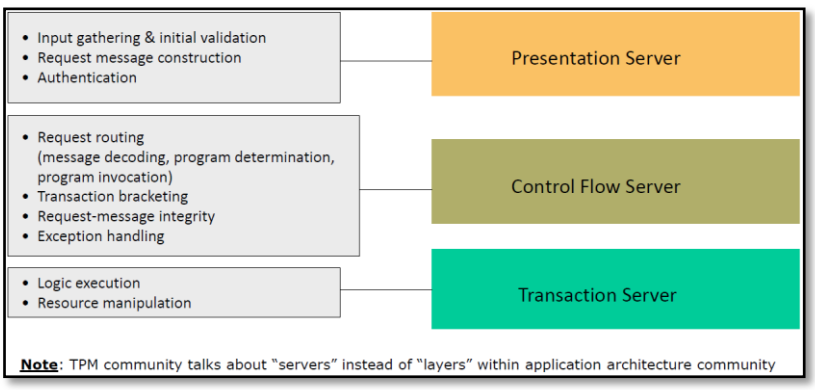
7.4.1 Concurrency Probleme bei Transactions

- Lost Update
- Dirty Read
- Non-Repeatable Read (Variation: Phantom Read)
- Incorrect Summary
- Deadlock
- Livelock (2 Systeme arbeiten und hängen)

7.4.2 Transaction Processing Monitor



7.4.3 Transaction Processing Monitor Struktur



7.4.4 Local Transactions vs. Distributed (Global) Transactions

Globale Transactions:

- Mehrere Ressourcen
- Ressource-Manager (mehrere) und Transaktion-Manager (einer, übergeordnet) aufgeteilt
- Sie arbeiten mit mehr als nur einer Datenbank (mehrere Ressource-Manager)
- Sind besonders teuer (grosser Koordinationsaufwand im verteilten System)
- Nur in Spezialfällen einzusetzen

Locale Transactions:

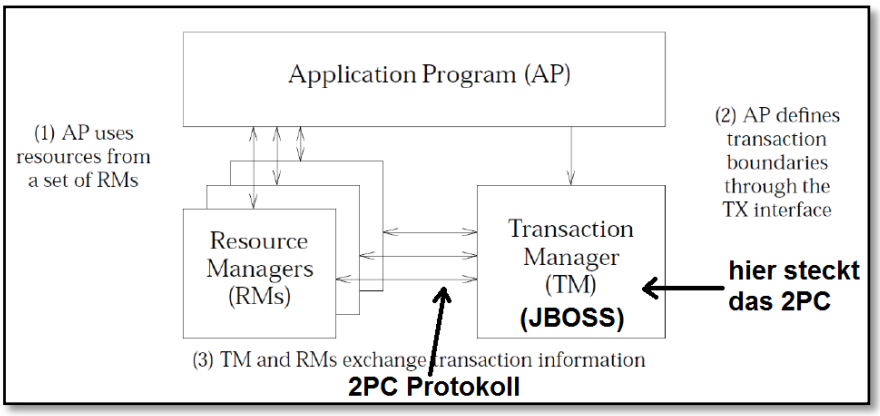
- NUR eine Ressource
- Ressource-Manager und Transaktion-Manager sind zusammen eine Middleware
- Hier gibt es prinzipiell nur ein „One-Phase-Commit“, d.h. wenn der Ressource-Manager sagt, dass bei ihm alles ok ist, wird die Transaktion durchgeführt.

Ressource-Manager:

- Datenbanksystem, z.B. SQL Server in Winterthur, Oracle in Hamburg

Transaktion-Manager:

- JEE Applikation Server, verwaltet Ressource-Manager
- Mit dem 2PC (Two-Phase Commit Protokoll) sorgt dafür, dass sich die Ressource-Manager untereinander nicht in die Quere kommen und durch das 2PC wird verwaltet der Transaktion-Manager die Ressource-Manager



7.4.5 System Transactions vs. Business Transactions

System Transactions:

- Transaction concept used so far corresponds to a system transaction (e.g supported by a database management system)

Business Transactions:

- For users of a business system the notion of a business transaction is more appropriate
  - Example: Logging in to a bank account system, preparing a credit transfer, accepting the transfer and logging out.
  - ACID-Properties should also apply to business transactions
  - Taking a system transaction for the execution of a business transaction would result in long transactions
    - Depending on concurrency requirements not always efficient for most transaction systems

=> Break business transaction into series of short transactions

7.4.6 Compensation (Aufräumverhalten / Reverse-Actions) insert < - > delete

- Not every action has a reverse that undoes it in the sense that the state can be changed as if the action would never have been done (real action or undoable action)
  - Example: sending a letter, drilling a hole, dispensing money by an automatic teller machine
- In reality, the effects of an arbitrary action cannot be simply undone, i.e. the initial state cannot be recreated
  - E.g. canceling a flight short before take off may cost 80% of the ticket price
- **An action used to reverse the effects of another action is called compensation action**
- Semantic Recovery:
  - Recovery schema base on compensation
- CAVEAT: Compensation very likely one of today's most frequently exploited techniques in transaction processing

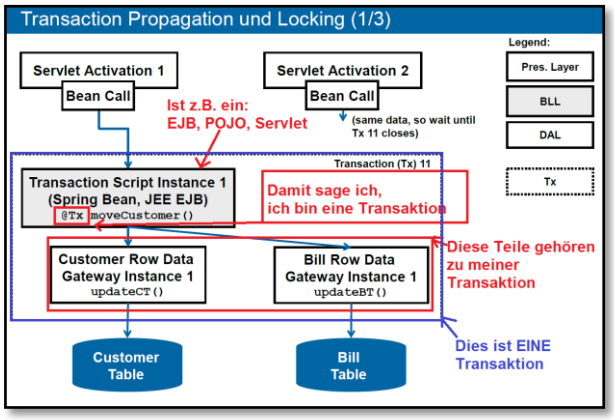
7.4.7 Transaktionsgrenze , Transaktionslänge, Transaktionsgrösse

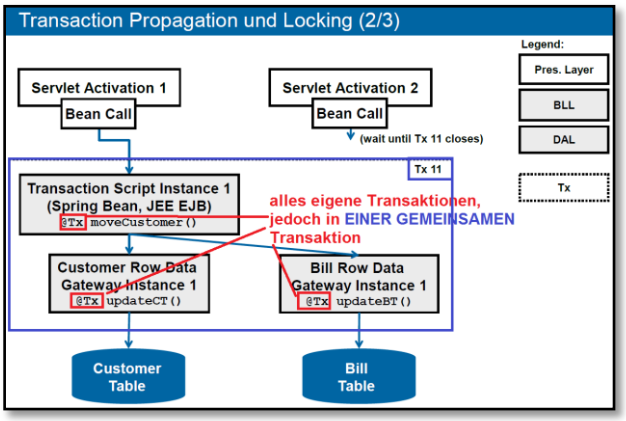
Es gibt **KEINE** opimale Transaktionsgrenze!!

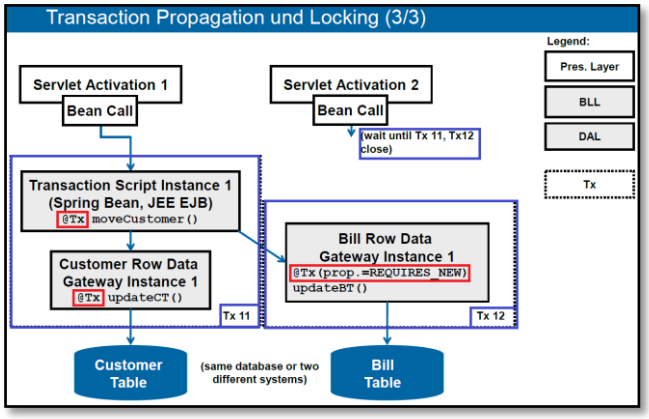
Die Grenzen einer Transaktion sind durch den Open-Call (Anfang) und dann das Commit bzw. den Rollback definiert (Ende).

Länge und Grösse beziehen sich auf die Anzahl und die Art der Zugriffe auf die Ressourcen innerhalb der Transaktionsgrenzen (also z.B. die SQL INSERT und UPDATE Statements in einer Datenbanktransaktion, die zu committen oder zurückzurollen sind: wie viele Statements? Wie viele Tabellen und Zeilen sind betroffen?).

7.4.8 Transaction Propagation







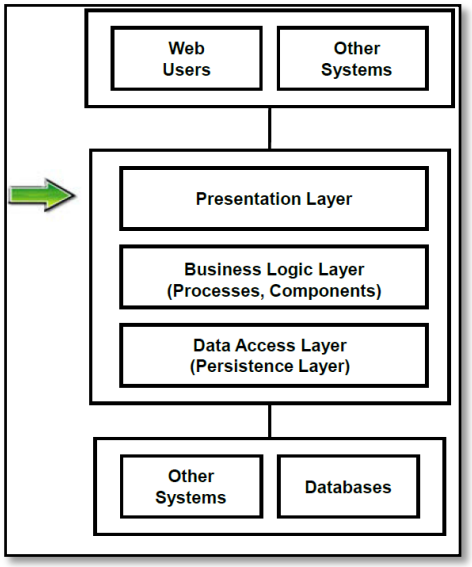
7.4.9 Praktische Tipps!!!

- Integrating existing autonomous applications into an application system will always be hard
  - Since the systems to be integrated are autonomous, they establish their own transaction boundaries
  - Thus, no tree of subordinated transactions can be build: Each such "subtransaction" simply commits without taking care of others (!)
- Combine **business transaction** and **system transaction** patterns
  - Do not use 2PC unless all resource managers are physically close to each other and have similar availability characteristics
- **Don't forget to clean up**
  - Commit opened transactions (or roll them back)
  - Close connections (only rely on garbage collection as a fallback)

7.4.10 ACID

Begriff	Beschreibung
Atomicity	All actions performed within a transaction either all succeed or are all Undone. *Folge von Operationen entweder ganz oder gar nicht ausgeführt. Keine Teilweise Ausführung.*
Consistency	A transaction transforms resources from one consistent state to another consistent state. *Eine Transaktion führt die Daten von einem konsistenten Zustand in den anderen über.*
Isolation	The program realizing a transaction is isolated from all other actions within the system. *Eine Transaktion soll so ausgeführt werden, als sei sie isoliert von anderen.*
Durability ***Dauerhaftigkeit***	Once successfully completed the effects of a transaction will survive any failure. *Änderungen einer Transaktion sind dauerhaft. Sie dürfen nicht auf Grund von Fehlern verloren gehen.*

8. PL Presentation Layer



8.1 PL Aufgaben gemäss CRC-Karte

Component: Presentation Layer im Mid Tier (in PoEAA Layering)	
Responsibilities: • Zeigt Eingabemasken an • Prüft Benutzerinput (incl. Security Checks) • Sendet Benutzerinput an Service-Komponenten im Business Logic Layer (BLL) • Nimmt Ergebnisse und Fehlermeldungen aus dem BLL entgegen • Entscheidet über nächsten Anzeigeschritt (Web: Page Flow)	**Collaborations (Interfaces):** • End User • Client Tier • BLL
Known uses (Implementations): • Servlet API • Java Server Pages (JSPs), Java Server Faces (JSF) • Microsoft: u.a. Active Server Pages (ASPs)	

8.2 Drei Arten von Benutzer

- Fach-Experten auf Abteilungs- oder Anwendungsinsel
  - Kennen Anwendung oft besser als Support und Maintenance Team
    - Bsp. «Frontbesuch» Back Office Mutationen Festnetztelefonanschlüsse
- Informatikinteressierte und IT-Quereinsteiger
  - Digital Natives mit/ohne tiefe Informatikkenntnisse
  - Mischen manchmal fachliche und technische Anforderungen
  - Programmieren z.T. selbst (z.B. Excel-Makros)
- IT-Gelegenheitsuser
  - Kaum/keine PC Kenntnisse
    - Bsp. Aufruf Internetseite immer über Google-Suche

=> Im Requirements Engineering muss man herausfinden, mit welchen Arten von Benutzern bei einer neuen Enterprise Application zu rechnen ist

8.3 Patterns

8.3.1 MVC Model View Controller

- Model, view and controller work together in a web server
    - Controller pulls out information of request and forwards to appropriate model handle business logic
    - Controller decides on the results of model object which view to take
    - It forwards the response to the view, which handles the result presentation

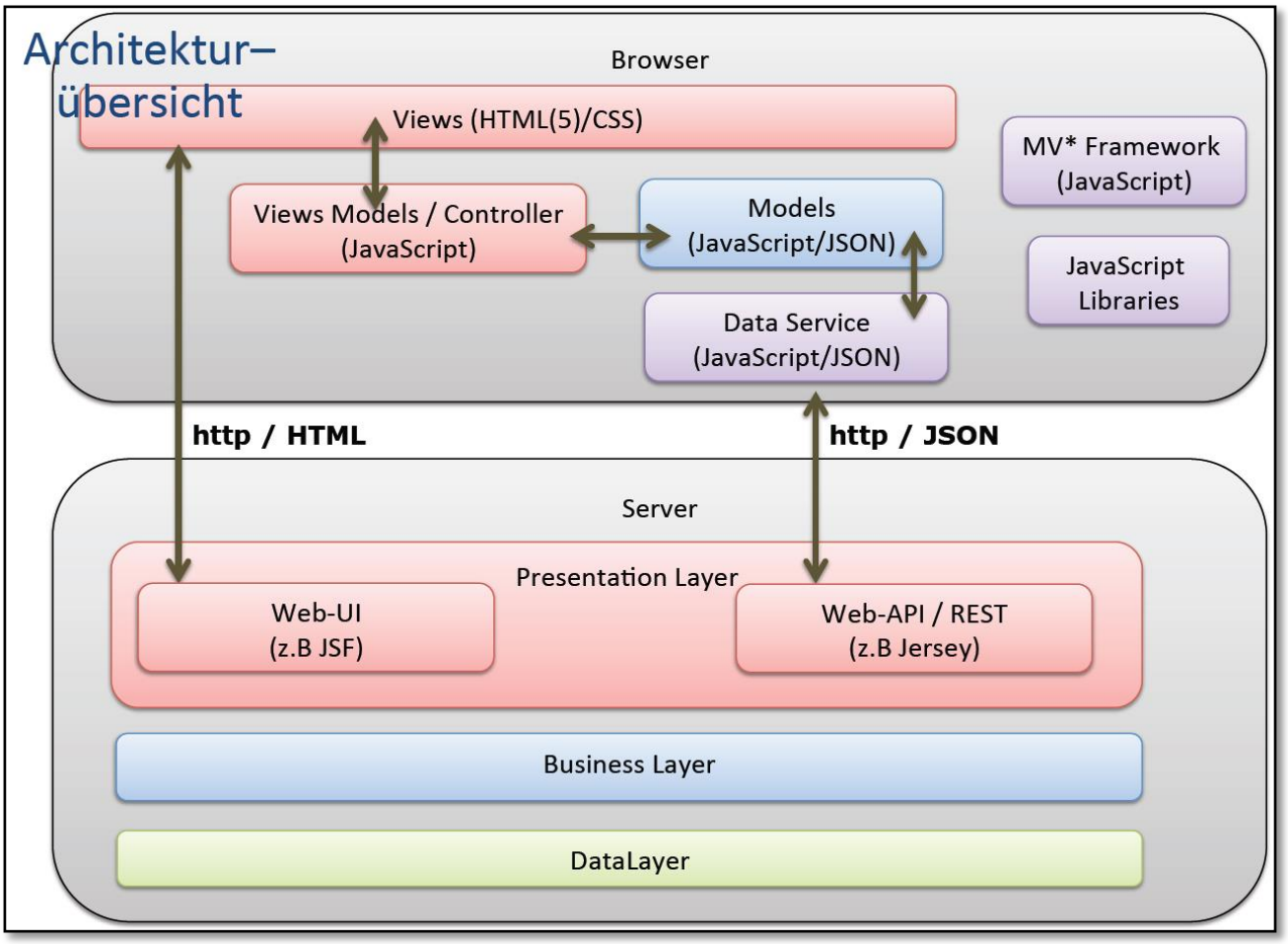
  ***=>** Forwarding based on HTTP request and session objects shared between controller and view*
- Reasons for applying MVC
    - **(Separation of Concerns)** Separate the model code from the web presentation

  ***=>** Use also Domain Model or Transaction Script for Domain Logic*
- Easy modification or addition of different presentations

ALLE WEITEREN BEHANDELTEN PATTERN`S –> SIEHE NÄCHSTES KAPITEL!

9. Web-Architektur Patterns

9.1 Übersicht



9.1.1 *Unterschied action/request based und component based*

Action based oder Request based	Component based
• http-Request löst Actions aus • http angewendet (Request, Response) • Einfacher MVC Control Flow • Beispiele: Struts, Spring MVC, Grails, Zend (PHP), ASP.NET MVC Ruby on Rails	• Web-UI aus UI-Komponenten mit EventListenern • http abstrahiert • MVC Control Flow meist komplexer • Wiederverwendbare Komponenten, Komponentenbibliotheken • Beispiele: JSF, Tapestry, Wicket

9.2 Pattern Wahl (Template View, Transform View, Two Step View)

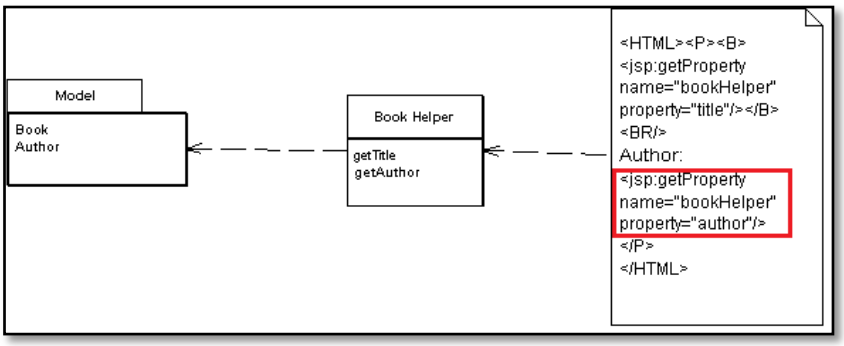
„How do I control the formatting of mx Web pages?

- **I like to edit the page an put hooks of dynamic data -> Template View**
- **I think of the page as a transform of domain data -> Transform View**
- **I need to make general changes to my site's look and feel -> Two Step View**
- **I need to multiple appearances (Auftritte) for the same logical screen format -> Two Step View**

=>	1. Entscheidung: ***Template View*** oder ***Transform View*** 2. Entscheidung: Einstufige View oder ***Two Step View***

9.3 Template View

„Renders information into HTML by embedding markers in an HTML page."



- Es werden Markierungen eingebettet um anzuzeigen wo dynamische Daten geladen werden.

Beispiele:

- JSP, ASP, PHP

Vorteile:

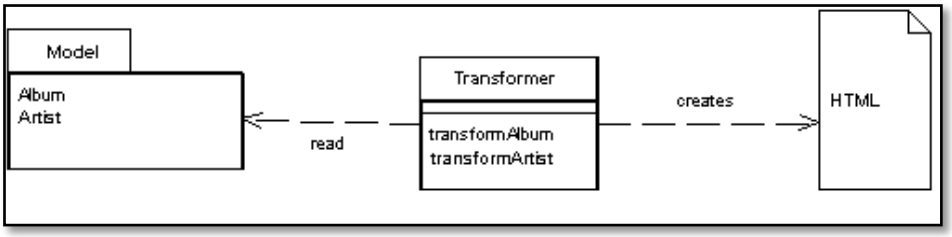
- Hohe Leistungsstärke und Flexibilität
-

Nachteile:

- Die View könnte zu viel Programmlogik beinhalten -> Hilfsobjekt erstellen
- Es kann chaotischer Code entstehen, dadurch wird die Wartung sehr schwierig
- Ist schwer zu testen, denn es wird normalerweise ein Web Server benötigt. Anders wie bei *Transform View* (Siehe nächstes Kapitel), welches sich einfach auch ohne Web Server testen lässt.

9.4 Transform View

„A view that processes domain data element by element and transforms it into HTML."



- **..**

Beispiele:

- XSLT

Vorteile:

- …
-

Nachteile:

- …

9.5 Two Step View (und einstufige Views)

9.5.1 *Einstufige Views*

Die zweite Entscheidung betrifft die Wahl, ob eine einzige Stufe verwendet werden soll (siehe Abbildung 4.2) oder ob *Two Step View* eingesetzt werden soll. Ein einstufiger View enthält meistens eine View-Komponente für jede Bildschirmmaske in der Anwendung. Der View nimmt Geschäftsdaten und stellt sie in HTML dar. Ich sage »meistens«, weil logisch ähnliche Bildschirmmasken Views gemeinsam nutzen können. Doch meistens gilt die Faustregel: ein View pro Bildschirm.

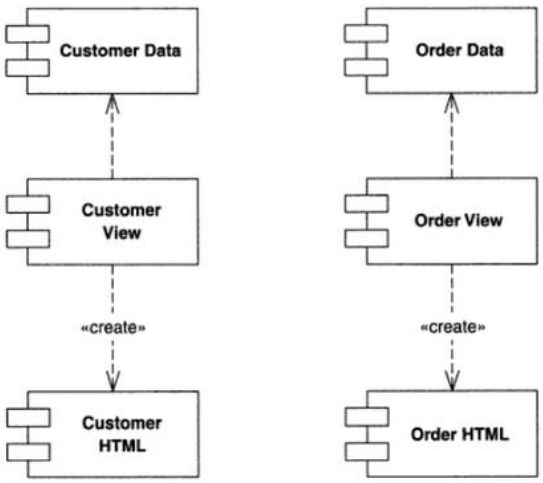
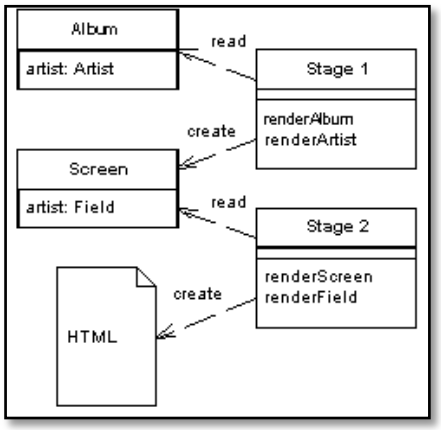


Abb. 4.2: Ein einstufiger View

9.5.2 *Two Step View*

„Turns domain data into HTML in two steps: first by forming some kind of logical page, then rendering the logical page into HTML."

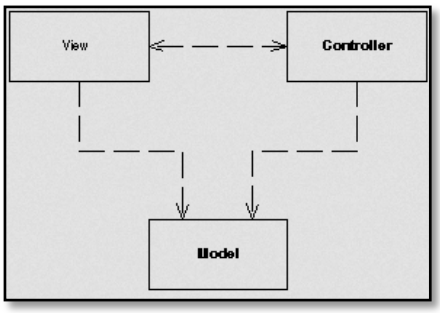


Beispiele:

- Die Web-Anwendung bietet Dienste an, z.B. mehrere Fluggesellschaften können dasselbe grundlegende Reservierungssystem nutzen, jedoch kann jedes Frontend verschieden aussehen
- Es können verschiedene Ausgabegeräte mit der selben „Logik" bedient werden

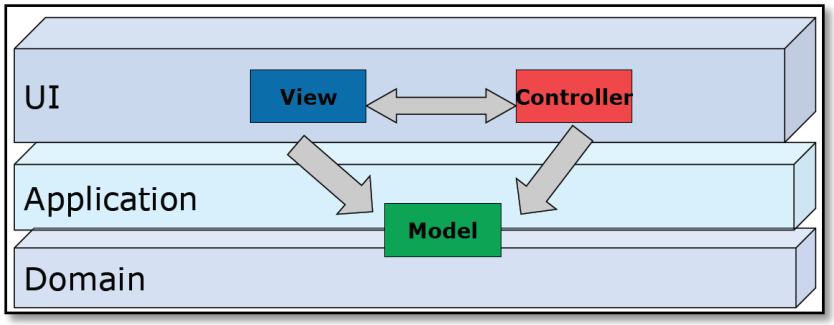
9.6 MVC Model View Controller

„Splits user interface interaction into three distinct roles."



Teilt Interaktion in 3 Teile:

- **View:** Darstellung der Informa5on für Benutzer
- **Controller (Input Controller/Application Controller):** Nimmt Benutzer-Eingaben entgegen und leitet sie an das Model zur Verarbeitung weiter, nach der Verarbeitung entscheidet der Controller welche View angezeigt wird und übergibt die Daten.
- **Model:** Business Logik



9.6.1 MVC und JSF

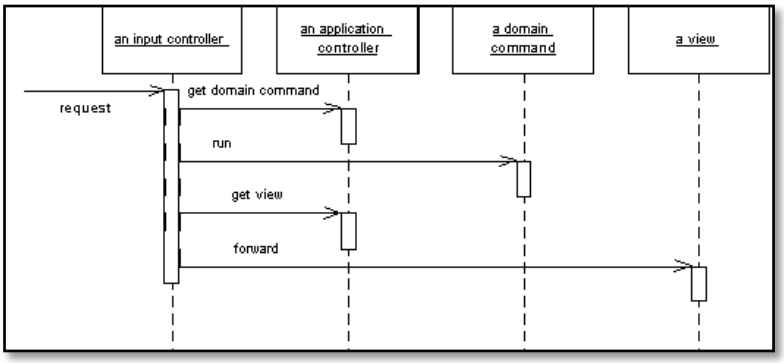
- JSF hat MVC-Architektur
- Model:
  - z.B. Java-Klassen, aus **Backing Beans** aufgerufen
- View:
  - **Facelets** oder JSP-Seiten als Template Views
- Controller:
  - **Faces-Servlet** als ***Front Controller*** (Input Controller)
    - Aber ***Front Controller*** dient nur dazu Life Cycle in Gang zu setzen
  - ***Application Controller*** sind eigentlich mit Page verknüpfte Actions und Listeners **=>** also ***Page Controller***

9.6.2 Umsetzung mit Java-Technologien

- Was eignet sich für Views?
  - JavaServer Pages, Facelets
- Was eignet sich für (Input-)Controller?
  - Servlets
- Was eignet sich für Model
  - Java(Beans), EJBs
- Web Frameworks: Struts, Spring MVC, JSF und … arbeiten alle nach dieser Architektur

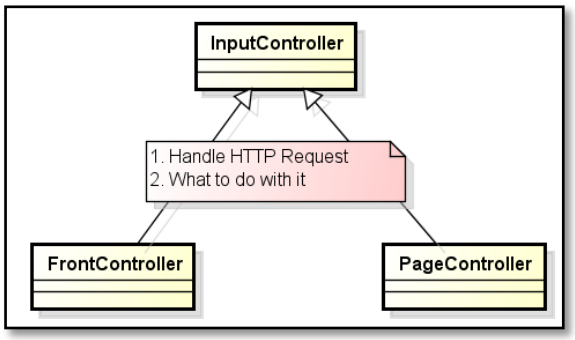
9.7 Application Controller

„A centralized point for handling screen navigation and the flow of an application."



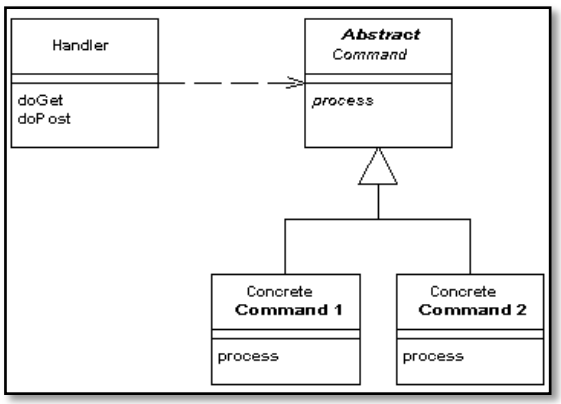
- Zentraler Punkt, um Kontrollfluss und Navigation der Präsentation zu steuern
- Steuert den Flow der Applikation oder die Navigation
- Einfacher Hinweis wie entschieden werden kann ob ein Application Controller benötigt wird ist, die Frage: „Hat der Rechner die Kontrolle über die Reihenfolge der Views, dann wird ein Application Controller benötigt, hat jedoch der Benutzer die Kontrolle, dann benötigt man keinen."

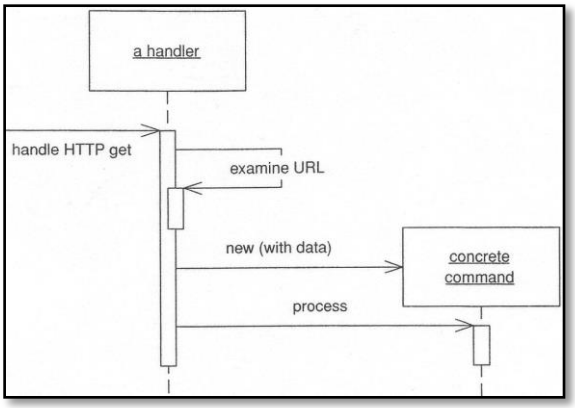
9.8 Input Controller



9.8.1 *Front Controller*

„A controller that handles all requests for a Web site."

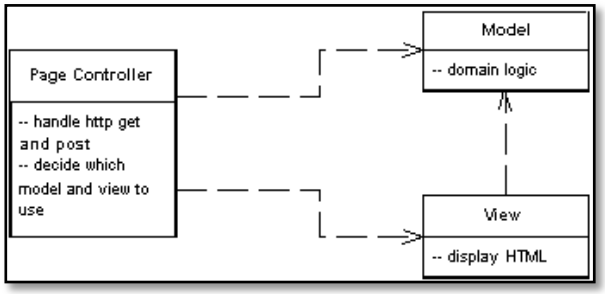




- Nur ein einziger zentraler Input---Controller
  - Oft aufgespalten in Handler
  - und Commands nach GoF „Command Pattern"
- Entscheidet was zu tun ist und delegiert weiter

9.8.2 Page Controller

„A controller that handles all requests for a Web site."



- Ein Controller für jede Web Page oder für jede Aktion (z.B. Button oder Link)
- Kann in Web Page integriert sein, aber besser in separatem Objekt!

9.8.3 Page- vs. Front- Controller

Page Controller	Front Controller
• Verteilter Controller • Leicht verständlich • Vor allem mit Controller in JSP-Seite angewandt • Achtung nicht in Scriptlets sondern in separate Klasse	• Zentraler Controller • Ein Erweiterungspunkt für gemeinsames Verhalten aller Requests • Dynamisches Hinzufügen von Verhalten (z.B. Security) => Intercepting Filters • Wird in vielen Web Frameworks verwendet: Struts, Spring MVC, Rails, ...

10. Session State Management

10.1 Stateful Applications (Begriffserklärung: session)

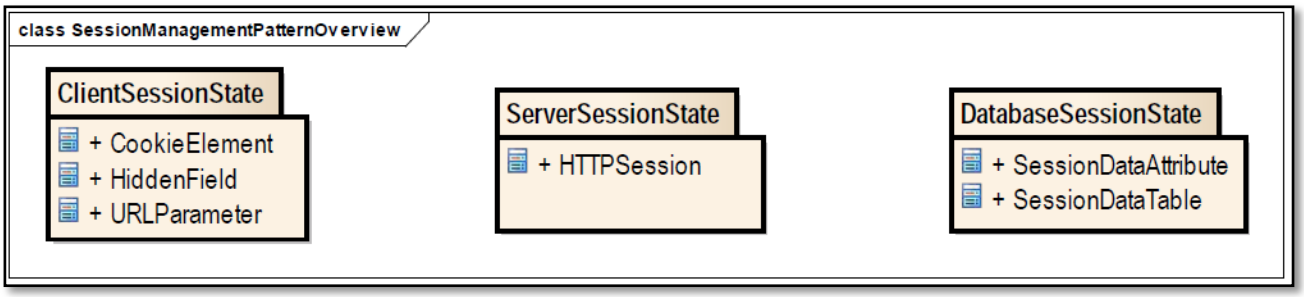
- Some applications need to manage state during client-server interactions
- Classical example: Shopping cart
  - Multiple interactions between client and server
  - Cart/Order contents (= state) have to be preserved between method calls
  - State data is finally written to record data (in case of purchase) or discarded

=> A stateful interaction is also called session

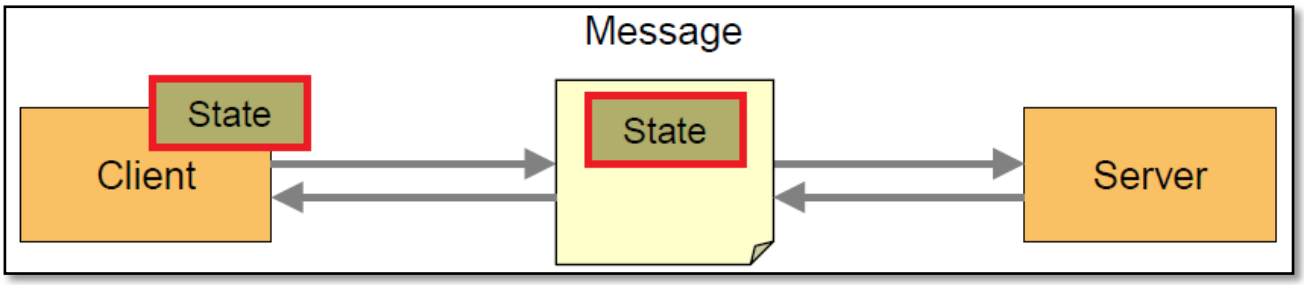
10.1.1 Stateful vs. Stateless

- Data used during a business transaction should be stored until the final commit to the database
- **Stateless** servers do not cache the results based on requests
  - Each request to a stateless server is seen as a new request
  - Objects that handle requests will be always reinitialized
  - Pool of objects can handle all requests
  - E.g. a web server that returns a web page based on the given URL
- **Stateful** servers have to maintain data to relate it to an ongoing session
  - Data is used for other requests from the same session
  - For each session a own object has to be maintained
  - E.g. a shopping cart that maintains a list of products while the user is still shopping and adding more and more

10.2 Client Session State, Server Session State, Database Session State



10.3 Client Session State

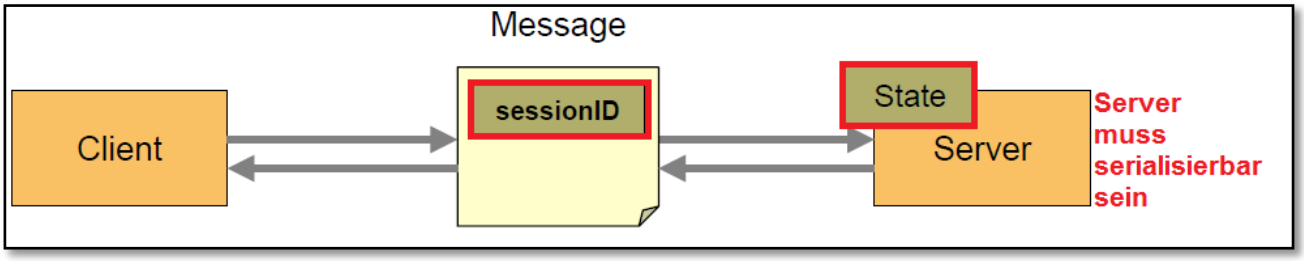


- Several ways:
  - Cookies
  - URL-Rewriting -> encoding data in URL
  - hidden fields
- Needs more bandwidth if amount of transferred session data is high
- Session state has to be transformed to useful objects
- **Security** and **integrity** should be considered (possibly encryption needed)

Was sollte berücksichtigt werden:

- Security, Privacy
  - Cookies sind böse!!
  - URL Rewriting ist besser
- Allgemein Client Session State
  - Grosser Overhead

10.4 Server Session State



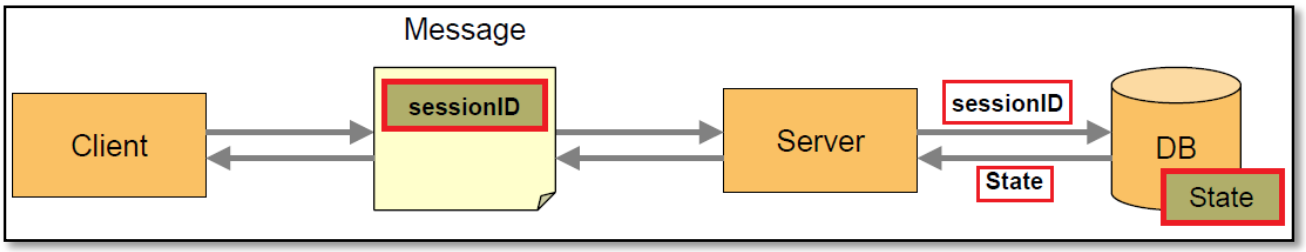
- In memory **(Arbeitsspeicher des Servers)** or in form of serialized objects
- Easy acquisition of session state by loading objects
- Good for systems, where a mass of data is to be transferred with each request

Beispiele:

Servlet: request.getSession().setAttribute ("CustomerName", request.getParameter("name"));

JSP: <%=session.getAttribute("CustomerName")%>

10.5 Database Session State



- Database Session State breaks up **session data** into **tables** and **fields**
  - Also server-side storage but with the intention of a longer-lasting incident
  - **Isolation** of session data from real data necessary (**eigene Session Tabelle**)
  - Session state has to be queried against database
  - Good for systems, where users are mostly idle, e.g. public retail system

=> Diese Variante ist grundsätzlich zu verwenden!

10.6 Frage zu REST und Statemanagement

Braucht es mit REST noch State Management?
-> Ja, je nach Anwendung, d.h. Wenn z.B. die Applikation über 10 Schritte geht, sollte man sich diese zwischenspeichern.

10.7 Herausforderungen im PL

- Input Validation (was wenn Database Data nötig)
- Interface Modality (Blocking or Non-Blocking?)
- (Multi) Threading in Client Tier and/or Mid Tier
- Authentication, Role-Based Access Control (RBAC), Confidentiality
- Error Handling (Business-level Undo)
- Software Updates (e.g. Java Web Start)
- Natural Language Support
- Rich Client vs. Thin Client vs. Rich Internet Application (RIA)
- Page Flow vs. Process Flow (in Multi-Channel Context)
- Offline Processing (e.g. HTML 5) and Content Management
- Browser Compatibility
- Graphical Design and Screen Layout (e.g. Widget/Facelet Composition)
- Plethora of User Interface Frameworks (e.g. JavaScript, but also Java)
- Testing – Simulation of End User Input and Donstream Layer Interfaces

11. Integration & Messaging

11.1 Integrationsstile

Ansätze, wie zwei verschiedene Programme (z.B. Java und C#) miteinander kommunizieren können.

11.1.1 File Transfer

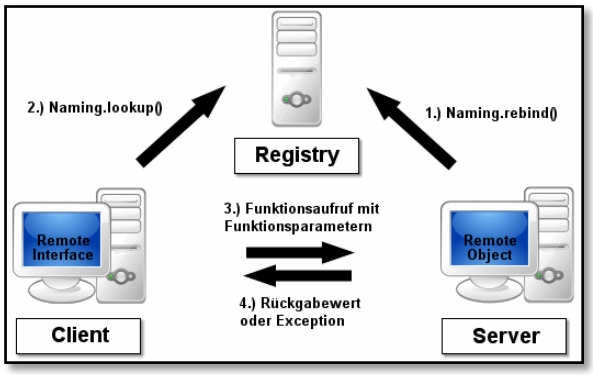
Bsp. java.io, FTP

11.1.2 Shared Database

Bsp. JDBC/SQL

11.1.3 Remote Procedure Call RPC

Bsp. RMI: **Remote Method Invocation** (RMI, deutsch etwa „Aufruf entfernter Methoden“), gelegentlich auch als Methodenfernaufruf bezeichnet, ist der Aufruf einer Methode eines entfernten Java-Objekts und realisiert die Java-eigene Art des Remote Procedure Call.



11.1.4 Messaging

Bsp. JMS (Wichtig: JMS ist nur API, kein MOM-Provider!! MOM Provider für Interoperabilität)

Java Message Service (JMS) ist eine Programmierschnittstelle (API) für die Ansteuerung einer Message Oriented Middleware (MOM) zum Senden und Empfangen von Nachrichten aus einem Client heraus, der in der Programmiersprache Java geschrieben ist. JMS hat das Ziel, lose gekoppelte, verlässliche und asynchrone Kommunikation zwischen den Komponenten einer verteilten Anwendung zu ermöglichen.

11.1.5 Pipes & Filters Pattern (POSA 1)

11.1.6 Fragen zu Integrationsstilen

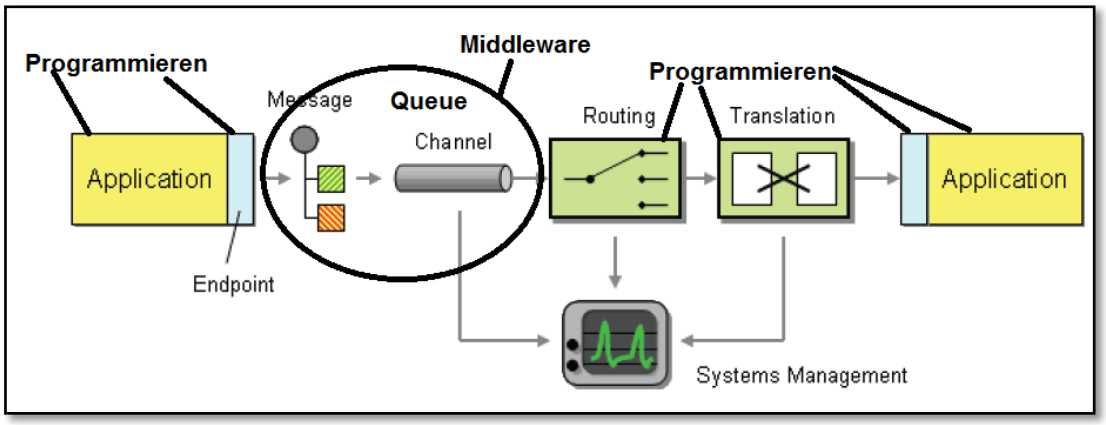
Is messaging simpler than RPC?

-> Ja, weil das API einfacher ist, Nachricht einfach in Queue schreiben

Does it perform better?

-> Kann man nicht sagen

11.2 Integration/Messaging Übersicht (in 5 Schritten)



In essence, a message is transmitted in five steps:

1. **Create** – the sender creates the message and populates it with data.
2. **Send** – the sender adds the message to a channel.
3. **Deliver** – the messaging system moves the message from the sender's computer to the receiver's computer, making it available to the receiver.
4. **Receive** – the receiver reads the message from the channel.
5. **Process** – the receiver extracts the data from the message.

11.3 Messaging System (MOM)

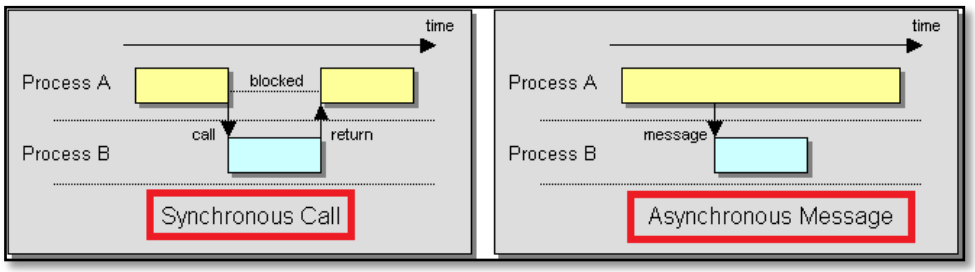
Nachrichtenorientierte Middleware bzw. Message Oriented Middleware (MOM) bezeichnet Middleware, die auf der asynchronen oder synchronen Kommunikation, also der Übertragung von Nachrichten (Messages) beruht. Das Format für die Nachrichten ist nicht festgelegt, in der Praxis hat sich jedoch XML als beliebtes Format etabliert.
MOM unterstützt drei verschiedene Kommunikationsprotokolle:

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

MOM: Messaging-oriented Middleware is a software system that provides Messaging; it is comparable with database systems that manage data persistence.

- Database administrator populates database with schema for application data; MOM administrator configures the messaging system with channels that define the paths of communication between the applications. Messaging system manages sending and receiving of messages.
- Database makes sure each data record is safely persisted, and likewise main task of messaging system is to move messages from the sender's computer to the receiver's computer in a reliable fashion.
- The reason a messaging system is needed to move messages from one computer to another is that computers and the networks that connect them are inherently unreliable:
    - Just because one application is ready to send a communication does not mean that the other application is ready to receive it.
    - Even if both applications are ready, the network may not be working, or may fail to transmit the data properly.
- A messaging system overcomes these limitations by repeatedly trying to transmit the message until it succeeds. Under ideal circumstances, the message is transmitted successfully on the first try, but circumstances are often not ideal.

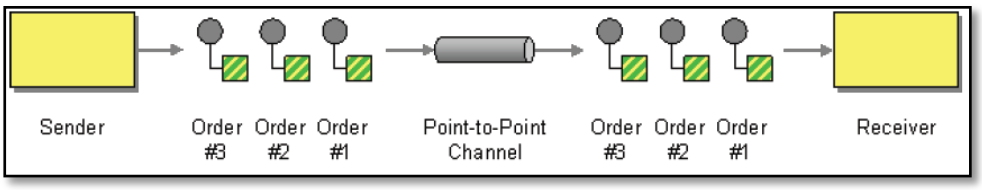
11.3.1 Synchron vs. Asynchron



11.4 Point-To-Point Channel

Question

„How can the caller be sure that exactly one receiver will receive the document or perform the call?"



Sender <-> Receiver -> asynchron

Answer

„Send the message on a Point-to-Point Channel, which ensures that only one receiver will receive a particular message."

- Ensures that **only one receiver** consumes any given message. If the channel has **multiple receivers, only one** of them can successfully consume a particular message.
- If multiple receivers try to consume a single message, the channel ensures that only one of them succeeds, so the receivers do not have to coordinate with each other.
- The channel can still have multiple receivers to consume multiple messages concurrently, but only a single receiver consumes any one message.

Example

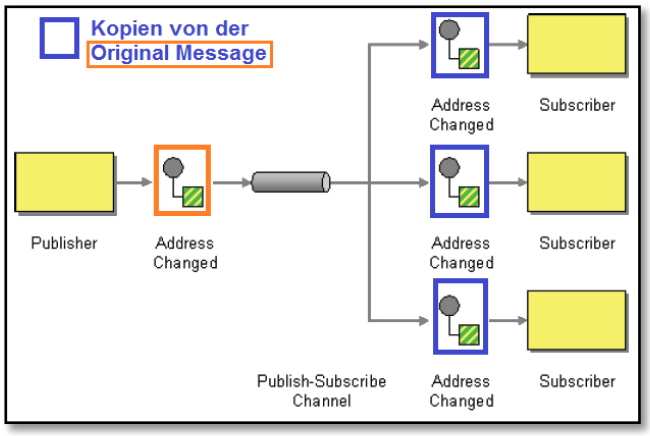
Stock Trading

In a stock trading system, the request to make a particular trade is a message that should be consumed and performed by excactly ione receiver, so the message should be placed on a Point-To-Point Channel

11.5 Publish-Subscribe Channel

Question

„How can the sender broadcast an event to all interested receivers?"



Answer

„ Send the event on a Publish-Subscribe Channel, which delivers a copy of a particular event to each receiver."

- Has one input channel that splits into multiple output channels, one for each subscriber.
- When an event is published into the channel, the Publish-Subscribe Channel ***delivers a copy*** of the message to each of the output channels.
- Each output channel has only one subscriber, which is only allowed to consume a message once.
- Each subscriber only gets the message once and consumed copies disappear from their channels.

Example

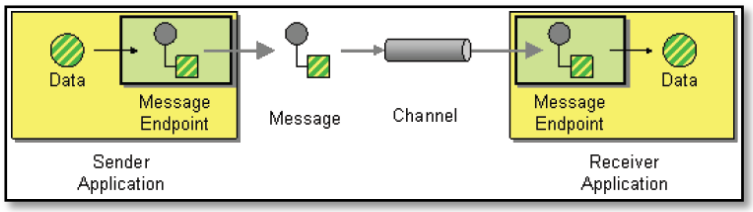
Stock Trading

In a stock trading system, many systems may need to be notified og the completion of a trade, so make them all subscribers of a Publish-Subscribe Channel that publishes trade completions.

11.6 Message Endpoint Pattern (Design Considerations)

Question

„ How does an application connect to a messaging channel to send and receive messages?"



Answer

„ Connect an application to a messaging channel using a Message Endpoint, a client of the messaging system that the application can then use to send or receive messages."

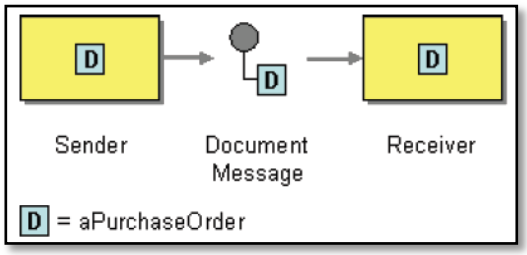
Desgin Erwägung/ Architectural Decisions

- *Number of consumer endpoints?*
- *Behavior of consumers: competing? browse message or remove it from queue?*
- *Idempotency (i.e., resending identical request messages does not change state of receiving application)?*
- *Transactional semantics (ACID properties, local/distributed 2PC)?*

11.7 Document Message Pattern

Question

„How can messaging be used to transfer data between applications?"



Answer

„Use a Document Message to reliably transfer a data structure between applications."

- Whereas a **Command Message tells the receiver to invoke certain behavior**, a **Document Message just passes data** and **lets the receiver decide what**, if anything, to do with the data.
- The data is a single unit of data, a single object or data structure which may decompose into smaller units. (**JMS-> Text Messages or Object Messages**)
- E.g. SQL row or row set (in information integration context)

Example

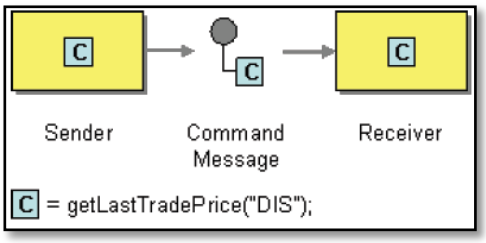
SOAP and WSDL

With the SOAP Protocol and WSDL service description, when using document-style SOAP messages, the SOAP message is an example of a DOCUMENT MESSAGE. The SOAP Message body is an XML document, and the SOAP message transmits that document from sender to the receiver.

11.8 Command Message Pattern

Question

„How can messaging be used to invoke a procedure in another application?"



Answer

„Use a Command Message to reliably invoke a procedure in another application."

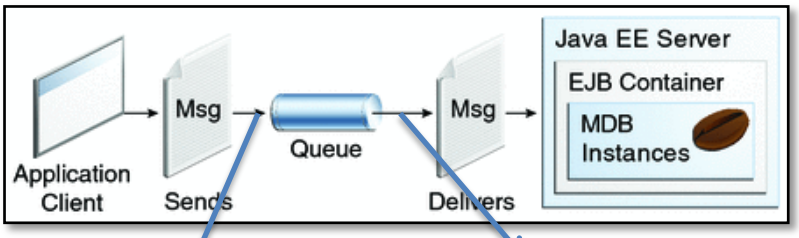
- There is no specific message type for commands; a Command Message is simply a regular message that happens to **contain a command.**
- **In JMS**, the command message could be any type of message; examples include an **ObjectMessage** containing a **Serializable** command object, a **TextMessage** containing the command in XML form, etc.
- In .NET, a command message is a Message with a command stored in it.

Example

SOAP and WSDL

With the SOAP protocol and WSDL service desciption, when using RPC-style SOAP messages, the request message is an example of this COMMAND MESSAGE pattern. Whit this usage, the SOAP message body contains the name of the methode to invoke in the receiver and the parameter values to pass into the method.

11.9 MDB Message-Driven Bean



SENDEN

```
@Resource(mappedName="jms/ConnectionFactory")
private static ConnectionFactory connectionFactory;

@Resource(mappedName="jms/Queue")
private static Queue queue;
```

Name der Queue: Queue
EIP-Pattern: Point-To-Point

Next, the client creates the connection, session, and message producer:

```
connection = connectionFactory.createConnection();
session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
messageProducer = session.createProducer(queue);
```

Finally, the client sends several messages to the queue:

```
message = session.createTextMessage();
for (int i = 0; i < NUM_MSGS; i++) {
    message.setText("This is message " + (i + 1));
    System.out.println("Sending message: " + message.getText());
    messageProducer.send(message);
}
```

EIP-Pattern: Document Message

Empfangen

```
@MessageDriven(mappedName="jms/Queue", activationConfig = {
        @ActivationConfigProperty(propertyName = "acknowledgeMode",
                                  propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName = "destinationType",
                                  propertyValue = "javax.jms.Queue")
})
public class SimpleMessageBean implements MessageListener {
    @Resource
    private MessageDrivenContext mdc;
    ...

    public void onMessage(Message inMessage) {
        TextMessage msg = null;

        try {
            if (inMessage instanceof TextMessage) {
                msg = (TextMessage) inMessage;
                logger.info("MESSAGE BEAN: Message received: " +
                    msg.getText());
            } else {
                logger.warning("Message of wrong type: " +
                    inMessage.getClass().getName());
            }
        } catch (JMSException e) {
            e.printStackTrace();
            mdc.setRollbackOnly();
        } catch (Throwable te) {
            te.printStackTrace();
        }
    }
```

11.9.1 Unterschied MDB und Statless Session Bean

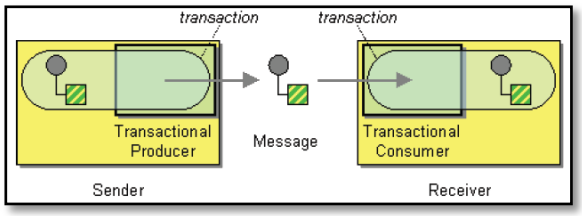
MDB: Asynchron; Schreibt in die Queue

SSB: Synchron; Holt Referenz und ruft dann die Methode auf.

11.10 Transactional Client Pattern

Question

„How can a client control its transactions with the messaging system?"



Answer

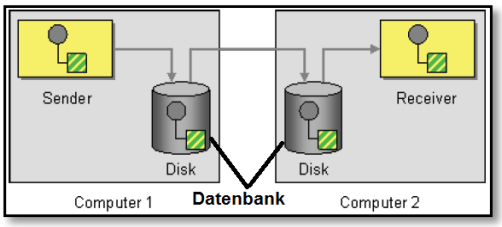
„Use a Transactional Client – make the client's session with the messaging system transactional so that the client can specify transaction boundaries."

- Both a sender and a receiver can be transactional. With a sender, the message **isn't "really" added** to the channel until the sender commits the transaction. With a receiver, the message **isn't "really" removed** from the channel until the receiver commits the transaction.
- A sender using explicit transactions can be used with a receiver using implicit transactions, and vise versa.
- A single channel might have a combination of implicitly and explicitly transactional senders (as well as receivers).

11.11 Guaranteed Delivery Pattern

Question

„ How can the sender make sure that a message will be delivered, even if the messaging system fails?"



Answer

„ Use Guaranteed Delivery to make messages persistent so that they are not lost even if the messaging system crashes."

- With Guaranteed Delivery, the messaging system uses a built-in data **store to persist messages**. Each computer the messaging system is installed on has its own data store so that the messages can be **stored locally**.
- When the sender sends a message, the send operation does not complete successfully until the message is safely stored in the sender's data store. Subsequently, the message is not deleted from one data store until it is successfully forwarded to and stored in the next data store.
- In this way, once the sender successfully sends the message, it is always stored on disk on at least one computer until is successfully delivered to and acknowledged by the receiver.

Example

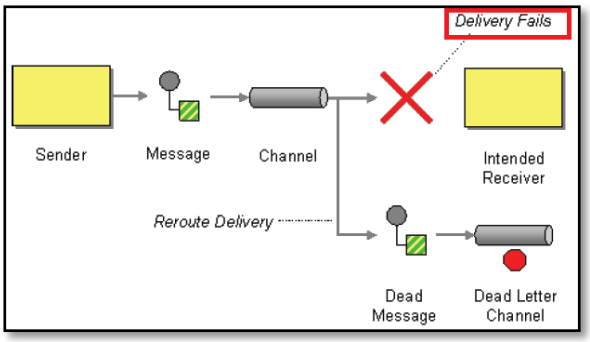
Stock Trading

In a stock trading system, trade requests and trade confirmations should probably be sent with GUARANTEED DELIVERY to help ensure thaht none are lost.

11.12 Dead Letter Channel Pattern

Question

„What will the messaging system do with a message it cannot deliver?"



Answer

„When a messaging system determines that it cannot or should not deliver a message, it may elect to move the message to a Dead Letter Channel."

- The specific way a Dead Letter Channel works depends on the specific messaging system's implementation, if it provides one at all. The channel may be called a **"dead message queue"** or **"dead letter queue."**
- Typically, each machine the messaging system is installed on has its o**wn local Dead Letter Channel** so that whatever machine a message dies on, it can be moved from one local queue to another without any networking uncertainties. This also records what machine the message died on. When the messaging system moves the message, it may also record the original channel the message was supposed to be delivered on.

Example

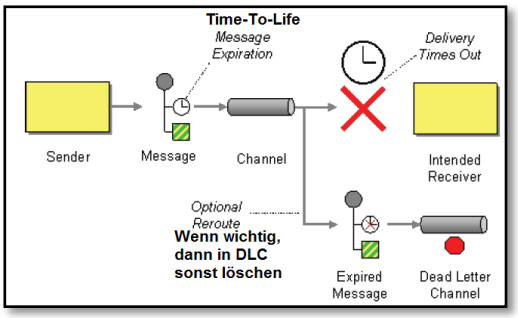
Stock Trading

In a stock trading system, an application that wishes to perform a trade can send a trade request. To make sure that the is received in reasonable amount of time (less than five minutes, perhaps), the requestor sets the request's MESSAGE EXPIRATION (siehe nächstes Kapitel) to five minutes. If the messaging system can not deliver the message in time, then the messaging system will tae the message off of the trade request channel and put the message on the DEAD LETTER CHANNEL. The trading system may wish to monitor the system's DEAD LETTER CHANNELS to determine if it is missing trades.

11.13 Message Expiration Pattern

Question

„ How can a sender indicate when a message should be considered stale and thus shouldn't be processed?"



Answer

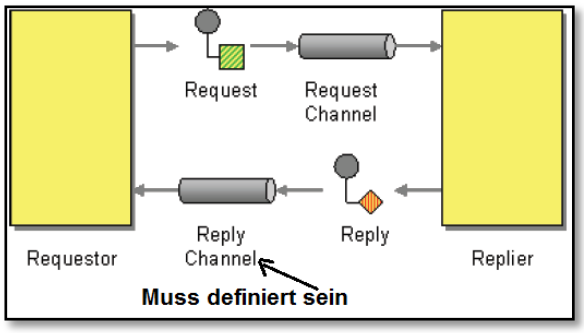
„ Set the Message Expiration to specify a time limit how long the message is viable."

- Once the time for which a message is viable passes, and the message still has not been consumed, then the message will expire.
- The messaging system's consumers will ignore an expired message; they treat the message as if it where never sent in the first place.
- Most messaging system implementations reroute expired messages to the Dead Letter Channel, while others simply discard expired messages; this may be configurable.

11.14 Request-Reply Pattern

Question

„When an application sends a message, how can it get a response from the receiver?"



Answer

„Send a pair of Request-Reply messages, each on its own channel."

- The request should contain a RETURN ADDRESS (siehe nächstes Kapitel) to tell the replier where to send the reply.

Example

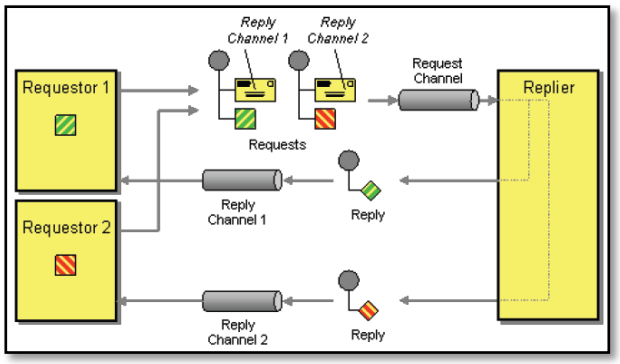
Woher weiss der Replier/Receiver, wohin die Replies zu senden sind (Reply Channel, Requestor)?

Muss nich 1:1 sein -> mir kann es egal sein wer meinen Request bearbeitet, hauptsache ich erhalte Reply.

11.15 Return Address Pattern

Question

„How does a replier know where to send the reply?"



Answer

„The request message should contain ta RETURN ADDRESS that indicates where to send the reply message."

- This way, the replier does not need to know where to send the reply; it can just obtain the reply channel address from the request message.

11.16 Praktische Tipps

- Messaging = Loose Coupling in Time + Reliability
  - Has its place, but comes at a price (design/test/management effort)
- Programmieraufwand ist beherrschbar mit JMS als Standard-API
  - Weniger ist mehr – exotische/komplexe JMS Features meiden (oft nicht portabel, nicht interoperabel)
  - Umgang mit sehr grossen Datenmengen?
- Vendor Lockin vermeiden
  - Gegen JMS-API programmieren, nicht ActiveMQ- oder IBM-Spezifika (o.ä.)
  - Bsp. Security Bridge/Gateway/HTTP Tunnel für Messaging über das Internet? At-Least-Once vs. Exactly-Once Delivery?
- Testen, testen, testen – insbesondere zeitliches Verhalten
  - Concurrency, Simulation Timeouts, Ausfall Message Recipient/Subcriber
  - Test-Messages gemeinsam mit Code ablegen/versionieren
- Messaging-Komponenten in Systemmanagement integrieren
  - Überwachung/Clustering Brokerprozess (Queue Manager)
- Schnittstellenverträge und Änderungsmanagement definieren
  - Insbesondere bei Document Messages (vom Typ Text)

12. Integration & Messaging (Routing & Transformation)

12.1 Problem of Integration

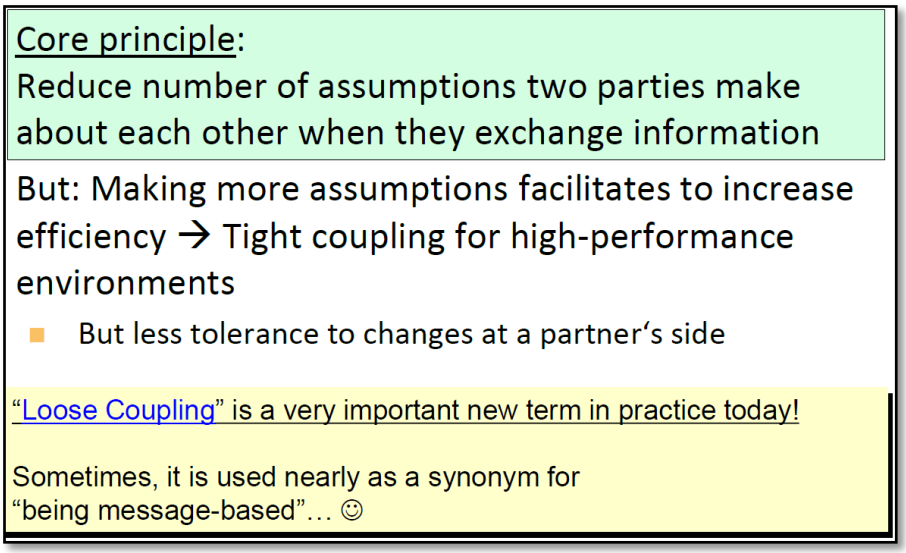
Most applications are not built to integrate with other applications

- They have been built as self-contained, standalone, one-in-all applications
  - This is nearly always the case for mainframe applications
    - Thus, “legacy integration” is a real “pain point”

Adding code even to custom (RYO*) applications to integrate with others is often not an option

- Increases complexity of already complex systems
- Danger to introduce undesired side-effects
- Requires application developers to understand integration technology
- Sometimes, source code is no longer available or source code is “incomprehensible” (remember Y2K?)
  - Access to source code is no option at all in case of packaged applications anyhow
    - So, we have to solve the problem anyhow ☺
    - Roll Your Own (“selbst gedrehte”) – bezeichnet selbst entwickelte Anwendungen

12.2 Loose Coupling Begriffserklärung (vier Autonomietypen)

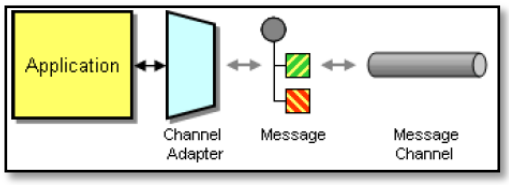


- **(Object) Reference autonomy** (also called location transparency)
  - Producers and consumers don't know each other
  - z.B. IP und Port nicht direkt im Code, sondern auslagern in Config-File
- **Platform autonomy**
  - Producers and consumers may be in different environments, written in different languages,…
- **Time autonomy**
  - Producers and consumers access channel at their own pace
    - Communication is **asynchronous**
    - Data exchanged is persistent
- **Format autonomy**
  - Producers and consumers may use different formats of data exchanged
    - Requires transformation "on the wire" (aka mediation)

12.3 Channel Adapter Pattern

Question

„ How can you connect an application to the messaging system so that it can send and receive messages?" (Wie verbinde ich die "nicht-messaging-Welt mit der messaging-Welt)



Application kann auch eine DB sein.

Answer

„ Use a Channel Adapter that can access the application's API or data and publish messages on a channel based on this data, and that likewise can receive messages and invoke functionality inside the application."

- The adapter acts as a messaging client to the messaging system and invokes applications functions via an application-supplied interface.
- This way, any application can connect to the messaging system and be integrated with other applications as long as it has a proper Channel Adapter.
- E.g. SQLChannelAdapter (in information integration context)

Example

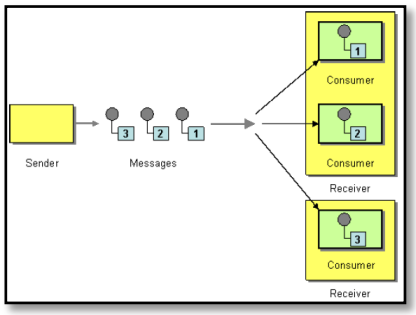
Stock Trading

A stock trading system may wish to keep a log of all of a stocks' prices in a database table. The messaging system may include a relational database adapter that logs each message from a channel to a specified table and schema. This channel-to-RDBMS adapter is a CHANNEL ADAPTER. The system may also be able to ti receive external quote requests from the Internet (TCP/IP or HTTP) and send them on its internal quote-request channel with the internal quote requests. This Internet-to-channel adapter is a CHANNEL ADAPTER.

12.4 Competing Consumers Pattern

Question

„ How can a messaging client process multiple messages concurrently?"



Answer

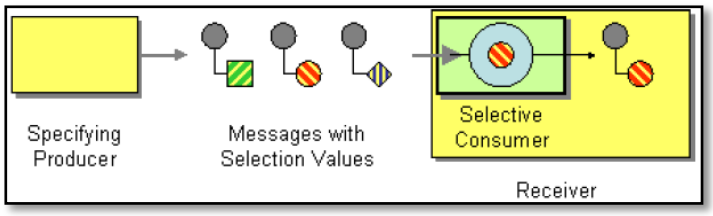
„Create multiple Competing Consumers on a single channel so that the consumers can process multiple messages concurrently."

- Lastverteilung, Spitzen-Abfang, Ausfallsicherheit
- Competing Consumers are multiple consumers that are all created to receive messages from a single Point-to-Point Channel.
- When the channel delivers a message, any of the consumers could potentially receive it. The messaging system's implementation determines which consumer actually receives the message, but in effect the consumers compete with each other to be the receiver.
- Once a consumer receives a message, it can delegate to the rest of its application

12.5 Selective Consumers Pattern

Question

„How can a message consumer select which messages it wishes to receive?"



Answer

„ Make the consumer a Selective Consumer, one that filters the messages delivered by its channel so that it only receives the ones that match its criteria."

- Implementation of the pattern in JMS 1.1:
  - MessageProducer adds a string property to message
    - Message.setStringProperty("req_type", "stock quote");
  - MessageConsumer is created with a selector string
    - String selector = new String("req_type = 'stock quote'");
    - MessageConsumer mc = session.createConsumer(destination, selector)
- .NET has the option to peek message (i.e. look at them without consumption)

Example

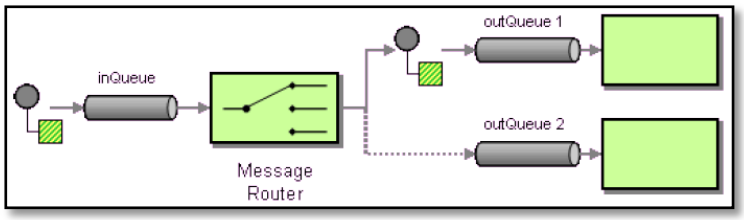
Stock Trading

A stock trading system with limited number of channels might need to use one channel for both quotes and trades. The sender would set the selector value on a quote message to QUOTE and the SELECTIVE CONSUMER for quotes would consume only messages with that selector value. Same for trade messages with selector value TRADE.

12.6 Message Router Pattern

Question

„How can you decouple individual processing steps so that messages can be passed to different filters depending on a set of conditions?



Answer

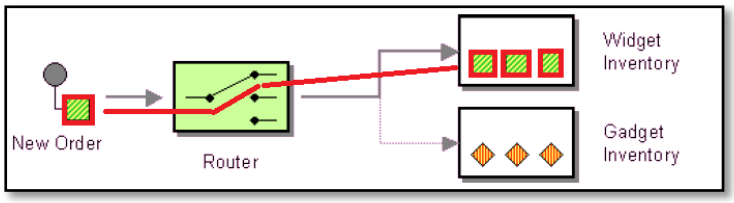
„Insert a special filter, a Message Router, which consumes a Message from one Message Channel and republishes it to a different Message Channel depending on a set of conditions.

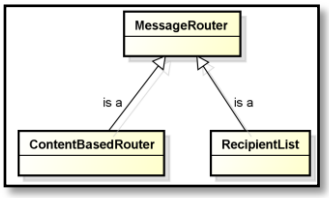
- **WICHTIG: Unterstützt alle VIER Autonomy-Attribute!**
- The Message Router differs from the most basic notion of Pipes and Filters in that it connects to multiple output channels.
- Thanks to the Pipes and Filters architecture the components surrounding the Message Router are completely unaware of the existence of a Message Router. A key property of the Message Router is that it does not modify the message contents. It only concerns itself with the destination of the message.

12.7 Content-Based Router Pattern

Question

„How do we handle a situation where the implementation of a single logical function (e.g., inventory check) is spread across multiple physical systems?"





Answer

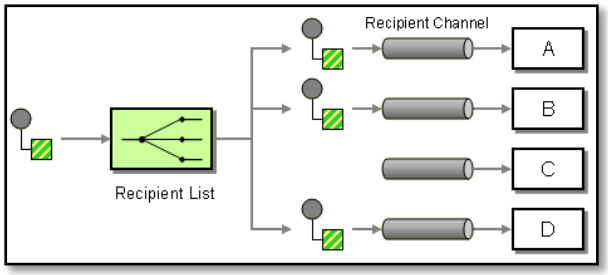
„Use a Content-Based Router to route each message to the correct recipient based on message content."

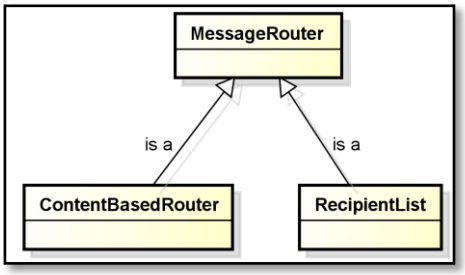
- The Content-Based Router **examines the message content and routes the message onto a different channel based on data contained in the message.**
- The routing can be based on a number of criteria, e.g., existence of fields, specific field values
- When implementing a Content-Based Router, special caution should be taken to make the routing function easy to maintain as the router can become a point of frequent maintenance.
- In more sophisticated integration scenarios, the Content-Based Router can take on the form of a configurable rules engine that computes the destination channel based on a set of configurable rules.

12.8 Recipient List Pattern

Question

„ How do we route a message to a list of dynamically specified recipients (Empfänger)?"





Answer

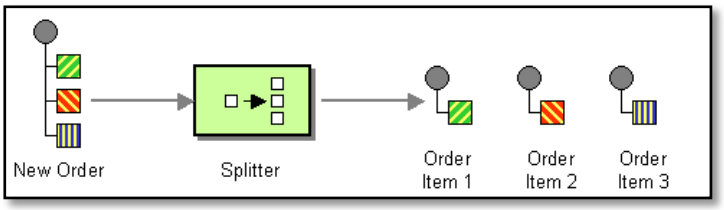
„ Define a channel for each recipient. Then use a Recipient List to inspect an incoming message, determine the list of desired recipients, and forward the message to all channels associated with the recipients in the list."

- Message wird **KOPIERT**!!
- Flexibel, z.B. wenn es eine neue Kundengruppe gibt einfach einfügen
- Hat Ähnlichkeiten mit dem Publisher-Subscriber Pattern
  - Publisher kenn seine Subscriber nicht
  - Router kennt seine Recipients
- The logic embedded in a Recipient List can be pictured as two separate parts even though the implementation is often coupled together. The first part computes a list of recipients.
- The second part simply traverses the list and sends a copy of the received message to each recipient.
- Just like a Content-Based Router, the Recipient List usually does not modify the message contents.

12.9 Splitter Pattern

Question

„How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?"



Answer

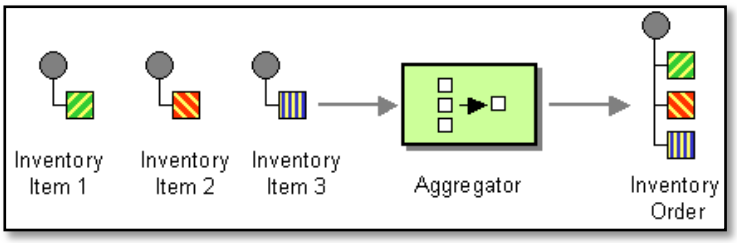
„Use a Splitter to break out the composite message into a series of individual messages, each containing data related to one item."

- Use a Splitter that consumes one message containing a list of repeating elements, each of which can be processed individually.
- The Splitter publishes a one message for each single element (or a subset of elements) from the original message.

12.10 Aggregator Pattern

Question

„How do we combine the results of individual, but related messages so that they can be processed as a whole?"

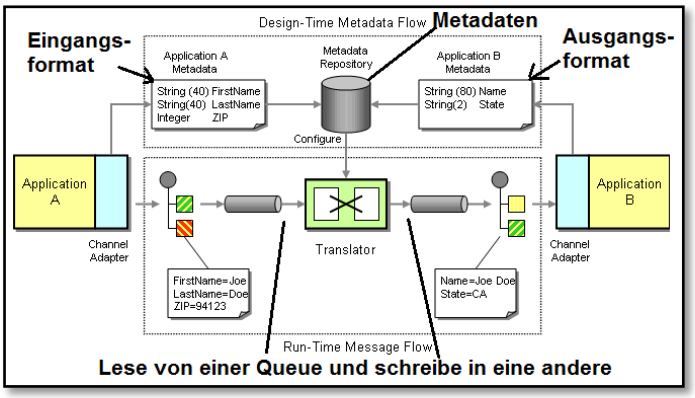


Answer

„Use a stateful filter, an Aggregator, to collect and store individual messages until a complete set of related messages has been received. Then, the Aggregator publishes a single message distilled from the individual messages."

- The Aggregator is a special Filter that receives a stream of messages and identifies messages that are correlated. Once a complete set of messages has been received the Aggregator collects information from each correlated message and publishes a single, aggregated message to the output channel for further processing.
- Aggregation strategies **(Wann mache ich weiter?)**: wait for all, timeout, first best, timeout with overrides, external event
- Aggregation algorithms **(Wie mache ich weiter?)**: select "best" answer, condense data, collect data for later evaluation

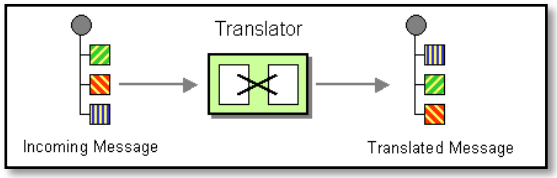
12.11 Message Transformation Allgemein



12.12 Message Translator Patter

Question

„How can systems using different data formats communicate with each other using messaging?"



Answer

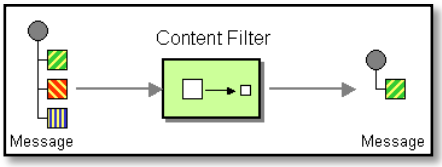
„Use a special filter, a Message Translator, between other filters or applications to translate one data format into another."

- The Message Translator is the messaging equivalent of the general Adapter design pattern.
- An adapter converts the interface of a component into a another interface so it can be used in a different context.
- **Example: XSLT!**
- **Ist das GoF-Pattern Adapter für Messaging**

12.13 Content Filer Pattern

Question

„How do you simplify dealing with a large message, when you are interested only in a few data items?"



Answer

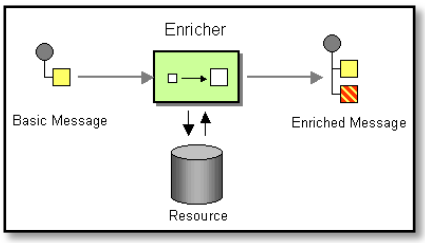
„Use a Content Filter to remove unimportant data items from a message leaving only important items."

- The Content Filter does not necessarily just remove data elements.
- A Content Filter is also useful to simplify the structure of the message. Often times, messages are represented as tree structures. Many messages originating from external systems or packaged applications contain many levels of nested, repeating groups because they are modeled after generic, normalized database structures.
- Frequently, known constraints and assumptions make this level of nesting superfluous and a Content Filter can be used to 'flatten' the hierarchy into a simple list of elements than can be more easily understood and processed by other systems.

12.14 Content Enricher Pattern

Question

„How do we communicate with another system if the message originator does not have all the required data items available?"



Answer

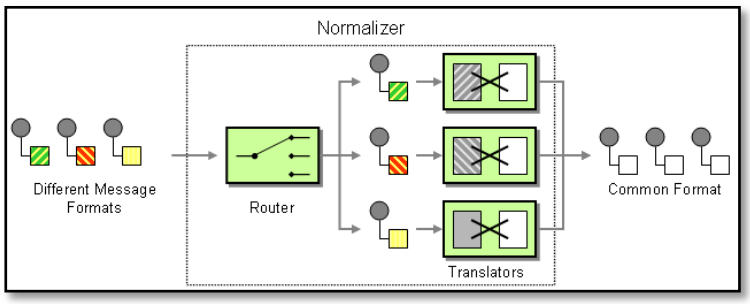
„Use a specialized transformer, a Content Enricher (a.k.a. Data Enricher), to access an external data source in order to augment a message with missing information."

- The Content Enricher uses information inside the incoming message (e.g. key fields) to retrieve data from an external source.
- After the Content Enricher has retrieved the required data from the resource, it appends the data to the message.
- The original information from the incoming message may be carried over into the resulting message or may no longer be needed, depending on the needs of the receiving application.

12.15 Normalizer Pattern

Question

„How do you process messages that are semantically equivalent, but arrive in a different format?"



Answer

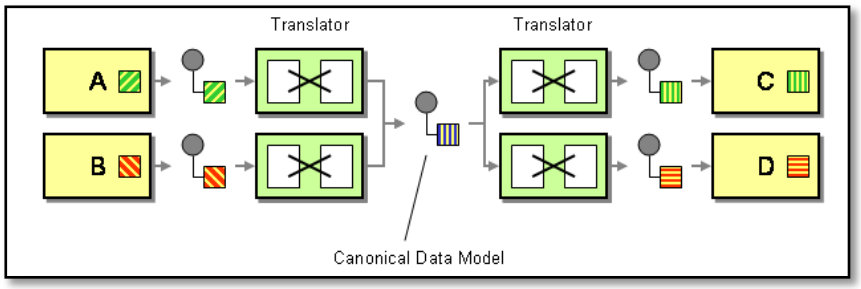
„Use a Normalizer to route each message type through a custom Message Translator so that the resulting messages match a common format."

- The Normalizer uses a Message Router to route the incoming message to the correct Message Translator. This requires the Message Router to detect the type of the incoming message.
- Many messaging systems equip each message with a type specifier field in the Message Header for this.
- However, in many B2B scenarios messages do not arrive as messages compliant with the enterprise's internal messaging system, but in diverse formats such as comma separated files or XML document without associated schema.

12.16 Canonical Data Model Pattern

Question

„How can you minimize dependencies when integrating applications that use different data formats?"



Answer

„Therefore, design a Canonical Data Model that is independent from any specific application. Require each application to produce and consume messages in this common format."

- The Canonical (autorisiert, anerkannt) Data Model provides an additional level of indirection between application's individual data formats. If a new application is added to the integration solution only transformation between the Canonical Data Model has to created, independent from the number of applications that already participate.

Example

WSDL (WebServices)

When accessing an external service from your application, the service may already specify a CANONIVAL DATA MODEL to be used. In the worlds of XML Web services, the data format is specified by a WSDL. The WSDL specifies the structure of request and reply messages that the service can consume and produce. In most cases, the data format specified in the WSDL is different than the internal format of the application providing service.

13. Architekturentscheide Templates

Wichtig: Antworten auf WARUM-Fragen und die Entscheide begründen!!

Beispiel eines Architekturenscheids: (Von Prof. Dr. O. Zimmermann, Beratungsstunde HS2013)
Wir nehmen HTTPS und nicht HTTP, weil sicherheitskritische Daten übertragen werden und diese dürfen nicht mitgelesen werden. Das kostet uns ein bisschen Performance, da Verschlüsselungsalgorithmen berechnet müssen, aber das nehmen wir gerne in Kauf.

13.1 IBM Template

Besonders wichtig im Zusammenhang mit der EnCom-Vorlesung und für die Prüfung HS2013:

- Assumptions (Annahmen)
- Alternatives (Alternativen)
- Justification (Begründung)

Dieses Beispiel zeigt einen Architekturentscheid über Integration:

Subject Area	Process and service layer design	**Topic**	Integration
Name	Integration Style	**AD ID**	2
Decision Made ***Entscheidungen***	We decided for RPC and the Messaging pattern (Enterprise Integration Patterns)		
Issue or Problem ***Problem***	How should process activities and underlying services communicate?		
Assumptions ***Annahmen***	Process model and NFRs/QA requirements are valid and stable		
Motivation ***Motivation***	If logical layers are physically distributed, they must be integrated.		
Alternatives ***Alternativen***	File transfer, shared database, no physical distribution (local calls)		
Justification ***Begründung***	This is an inherently synchronous scenario: VSP users as well as internal Telco staff expect immediate responses to their requests. Messaging system will ensure guaranteed delivery and buffer requests to unreliable data sources.		
Implications ***Konsequenzen***	Need to select, install, and configure a message-oriented middleware provider.		
Derived Requirements ***Abgeleitete Anforderungen***	Many finer grained patterns are now eligible and have to be decided upon: message construction, channel design, message routing, message transformation, system management (see Enterprise Integration Patterns book).		
Related Decisions ***Zusammenhängende Entscheidungen***	Next, we have to decide on one or more integration technologies implementing the selected two integration styles. Many alternatives exist, e.g., Java Message Service (JMS) providers		

13.2 Y-Template

Dieser Satz ist immer gleich aufgebaut: (englisch)..

In the context of *<use case uc and/or component co>,*

… facing *<non-functional concern c>,*

… we decided for *<option o1>*

and neglected *<options o2 to on>,*

… to achieve *<quality q>,*

… accepting downside *<consequence c>.*

(deutsch)..

Im Zusammenhang mit.. *<use case uc oder komponente co>,*

in anbetracht … *<nicht-funktionale Relevanz c>,*

haben wir uns für *<Option o-1>* entschieden..

und nicht für *<Optionen o-2 bis o-n>,*

… um *<Qualität q>* zu erreichen,

… dafür akzeptieren wir folgende Konsequenzen *<Konsequenz(en) k>.*

13.2.1 Beispiel eines Y-Statements

AD (Architecture Decision) 1: Apply Messaging pattern and RPC pattern

- **In the context of** the order management scenario at T,
- **facing** the need to process customer orders synchronously, without loosing any messages,
- **we decided** to apply the Messaging pattern and the RPC pattern
    - and neglected File Transfer, Shared Database, no physical distribution (local calls)
- **to achieve** guaranteed delivery and request buffering when dealing with unreliable data sources
- **accepting that** follow-on detailed design work has be performed and that we need to select, install, and configure a message-oriented middleware provider.

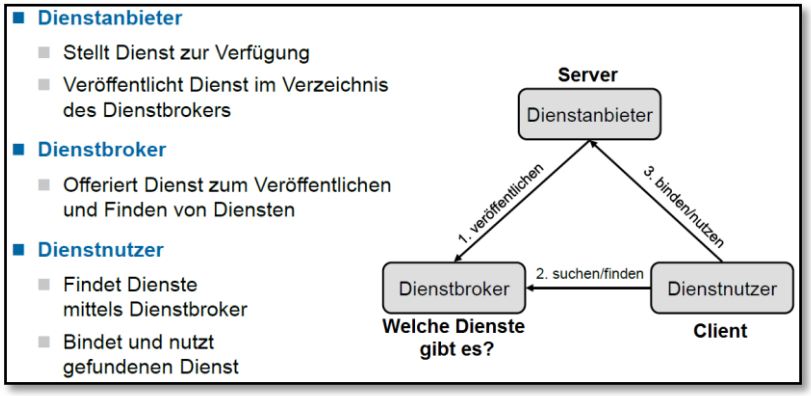
//unabhängiges Beispiel

AD (Architecture Decision) 2:Master Data Reporting (Information Integration) mit Extract-Transform-Load (ETL)

- **In the context** of the Marketing Campaign use case,
- **facing** the need to integrate information from heterogeneous sources,
- **we decided** for the Data Consolidation pattern as supported by ETL concepts, technologies and products,
    - and neglected the Data Virtualization pattern and its implementations
- **to achieve that**
    - all customers are represented in the same format and that
    - no duplicates exist,
- **accepting that**
    - data has to be stored redundantly in separate places/stages (materialization) and that
    - updates to the data sources do not become visible in the consolidated view until the next ETL run.

14. SOA Service Oriented Architecture

14.1 Definition



SOA can be defined as an architectural style promoting the concept of business-aligned ("Geschäftsprozess-orientiert") enterprise service as the fundamental unit of designing, building and composing enterprise business solutions.

WICHTIG zu beachten:

- **SOA ist nicht Webservices** – SOA beschreibt losgelöst von konkreten Implementierungsmethoden und -techniken ein Architekturparadigma.
- **SOA ist individuell** – Es gibt keine „Standard-SOA". Ein Unternehmen muss eine SOA immer auf die eigenen Bedürfnisse zuschneiden.
- **SOA ist nicht neu** – Eine serviceorientierte Architektur konnte auch schon vor einem Jahrzehnt mit den damals vorhandenen Methoden und Verfahren umgesetzt werden und fand unter anderem mit CORBA ihre Anwendung.
- **SOA ist keine Lösung für fachliche Probleme** – Als Architekturparadigma gibt SOA keine Empfehlung zur Behandlung von fachlichen Problemen. Siehe hierzu auch den Abschnitt Kritik.

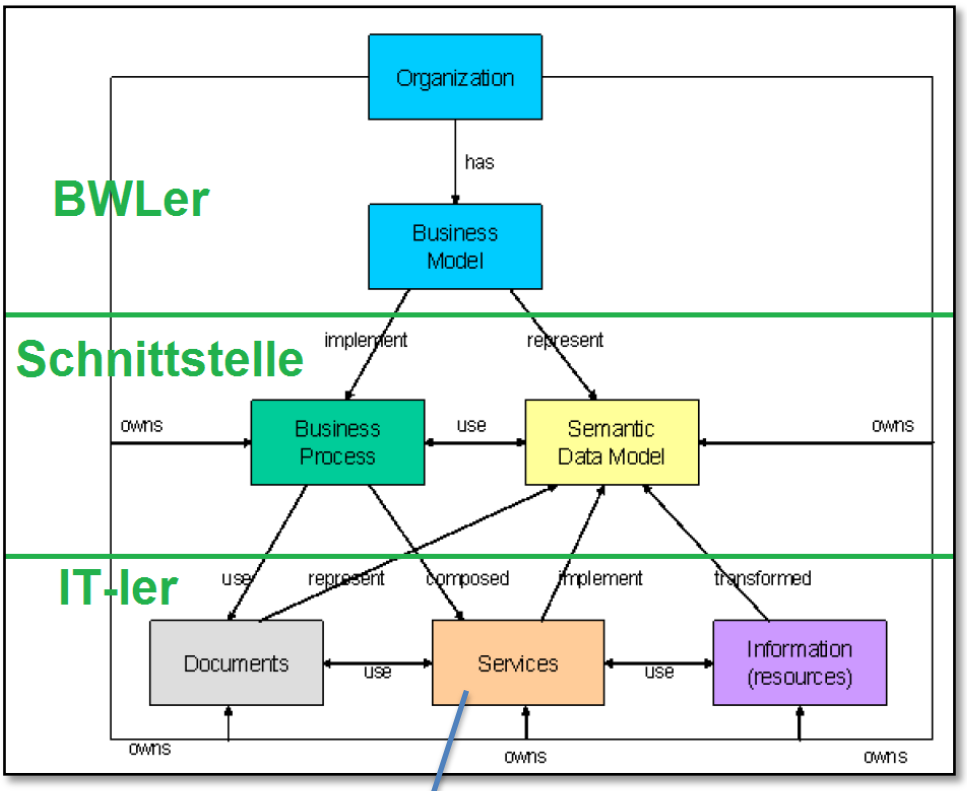
14.1.1 Beispiel

Als Beispiel für einen Geschäftsprozess dient die Bestellung eines Kunden bei einem Versandhändler. Bei diesem gibt es folgende Prozessschritte:

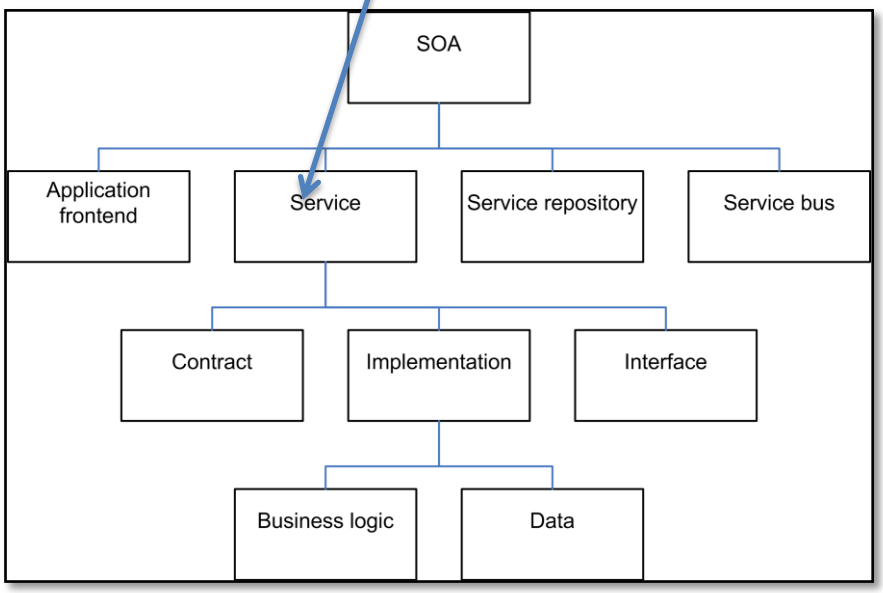
- Erfassung
- Verfügbarkeitsprüfung
- Bonitätsprüfung
- Bestellung
- Kommissionierung
- Versand
- Rechnungsstellung
- Zahlungseingang

Für jeden Geschäftsprozessschritt gibt es einen Dienst (Service). Dabei kann die Implementierung, Programmiersprache, etc, unterschiedlich sein. Die Abwicklung muss nicht sequentiell sein, wenn z.B. die Bonitätsprüfungs fehlschlägt, dann kann alles abgebrochen werden. Es können auch Schritte parallel ablaufen, z.B. Versand und Rechnungsstellung. Die Dienste können auch ausgetauscht werden, d.h. wenn das Unternehmen nun die Bonitätsprüfung in andere Hände gibt, dann ändert sich nur dieser Schritt, der Rest bleibt gleich.

14.2 Elemente von SOA (Sicht der BWL'er , der ITler und die Schnittstelle)



Quelle Wiki:



14.3 Service Contract

- **Business Perspective** - Used primarily for the choosing of the appropriate service during implementation design
- **Technical Perspective** - Used primarily for building consumers for a given service

14.3.1 Business Perspective (Semantik)

- Business functionality, including side effect effect (impacts on existing environments), provided by the service
- Conditions under which particular outcomes will occur
- Limitations of current service implementation
- Prerequisites for successful service invocation
- Detailed definition (in business terms) of service requests and responses (**z.B. Domain-Modell für Business Leute, d.h. nicht zu technisch**)
- SLAs supported by the service

14.3.2 Technical Perspective (Syntax)

- Service endpoint address(es), including communication protocols
- Message formats and encoding
- Definition of the errors that service invocation can return
- Service invocation policies such as security requirements, etc.
- Required vs optional information for the service request

14.3.3 QoS Quality of Service

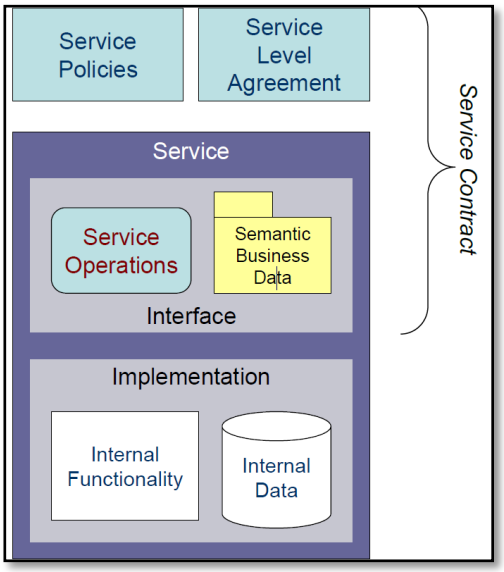
Fortschreibung der (hoffentlich SMARTen) NFRs aus dem Requirements Engineering (Analyse) ins Design und in die Implementierung. Also: wie erbringt der Service die mit Syntax und Semantik beschriebene Funktionalität?

14.3.4 Unterschied NFR & SLA

NFR: Welche Qualitätseigenschaften müssen erfüllt sein (wie sind die Features umgesetzt?)
SLA: Was kann der Code genau?

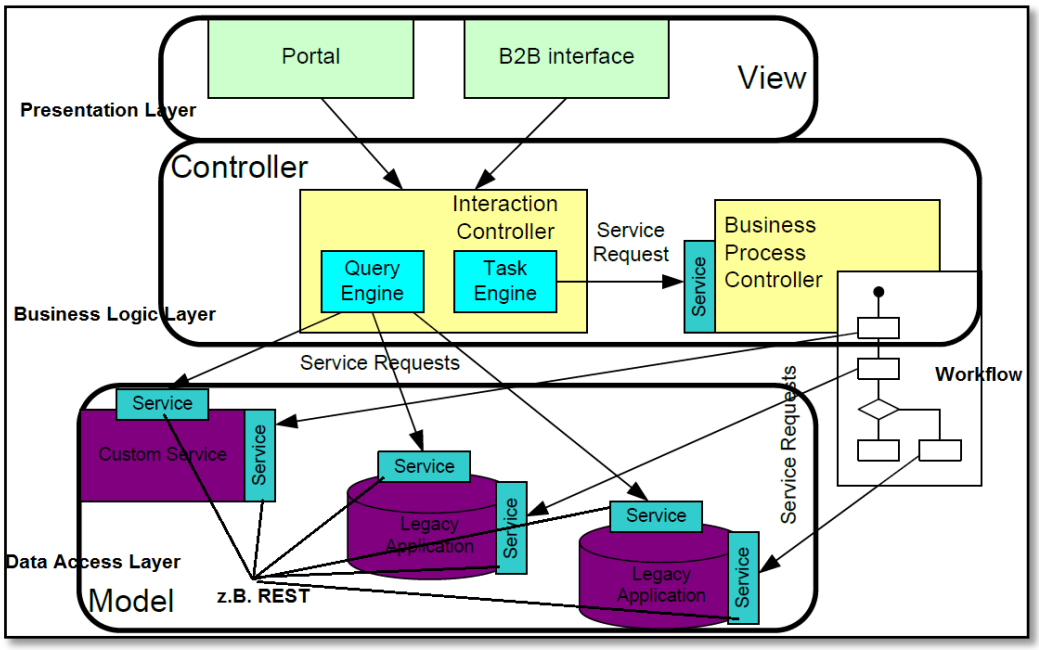
14.4 Was ist ein Service?

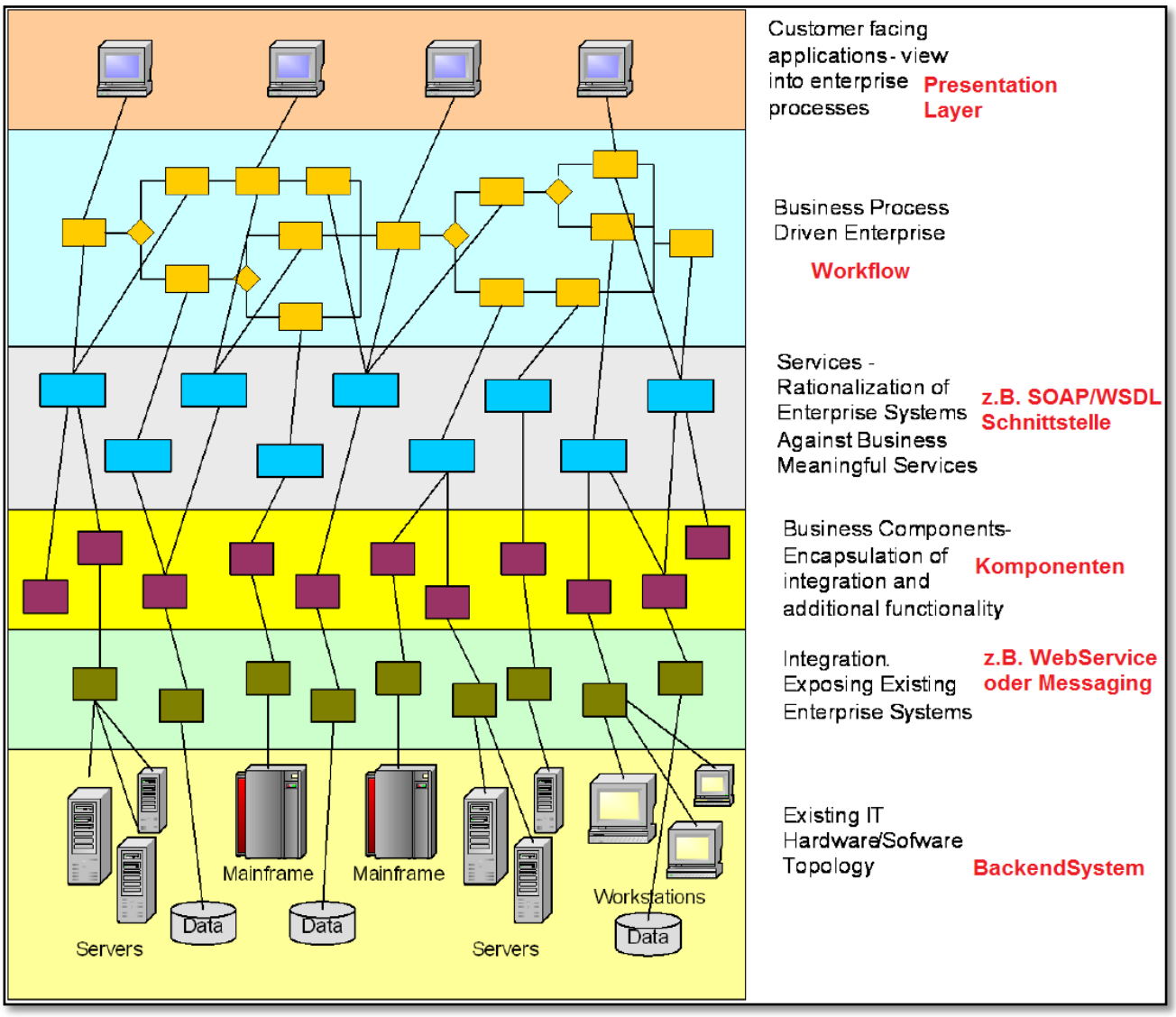
„Encapsulates a unit of work, made available through a service contract."



Policies z.B.: HTTP -> Kann ich Klartext übermitteln oder muss ich HTTPS verwenden?

14.5 SOA Architekturen





14.6 Unterschied SOAP und REST

Criterion	WS-*	RESTful HTTP
Primary concept	Service (operation) / RPC	Resource (object with state)
Original application domain	Enterprise application integration (B2B, CRM, etc.)	Distributed hypermedia systems (page downloads)
Tools	Many	Few (needed?)
Application/Transport Protocol	HTTP (in reality)	HTTP
Workflow	None or BPEL (full/fixed)	HATEOAS (partial/incremental)
Versioning	XML namespaces (in WSDL and XML Schema)	URIs, media types
Contract Nature	Static (WSDL)	static or dynamic (WADL, RAML)
Web Architecture Affinity	Medium/low	High
Engineering effort	Low (when sticking to defaults and best practices)	High

Pattern/Konzept	JAX-WS (SOAP over HTTP)	JAX-RS (RESTful HTTP)
Integration Style [EIP]	RPC (also kein Messaging)	RPC (also kein Messaging)
Message Type im Request (Command vs. Document) [EIP]	Command Message	Document Message
Channel Type [EIP]	Point-to-Point	Point-to-Point
Reference Autonomy (vgl. VL letzte Woche)	Vorhanden (wenn Service URI in Registry abgelegt und nicht hartkodiert wird)	Vorhanden (wenn Resource URI in Registry abgelegt und nicht hartkodiert wird)
Time Autonomy	Nicht vorhanden (synchroner Call)	Nicht vorhanden (synchroner Call)
Format Autonomy	Teilweise vorhanden mit XML	Teilweise vorhanden mit XML und JSON
Platform Autonomy	Vorhanden	Vorhanden
Service Contract	WSDL	Kein etabliertes Standardformat (Vorschläge: WADL, RAML)
Channel Adapter auf Clientseite [EIP]	Web Service Proxy	HTTP Client
Request-Reply [EIP]	Unterstützt	Unterstützt
Correlation Identifier [EIP]	Kann in SOAP Header oder Body (Payload) eingetragen werden	HTTP Header oder Payload
Message Expiration [EIP]	Nicht direkt unterstützt (aber HTTP Request Timeout)	Nicht direkt unterstützt (aber HTTP Request Timeout)
Runtime Protocol ("Transport")	HTTP (in realitas), viele Protokolle (in der Theorie durch Bindings)	HTTP
Message Exchange Format	XML	Alle MIME-Types, insbesondere auch JSON

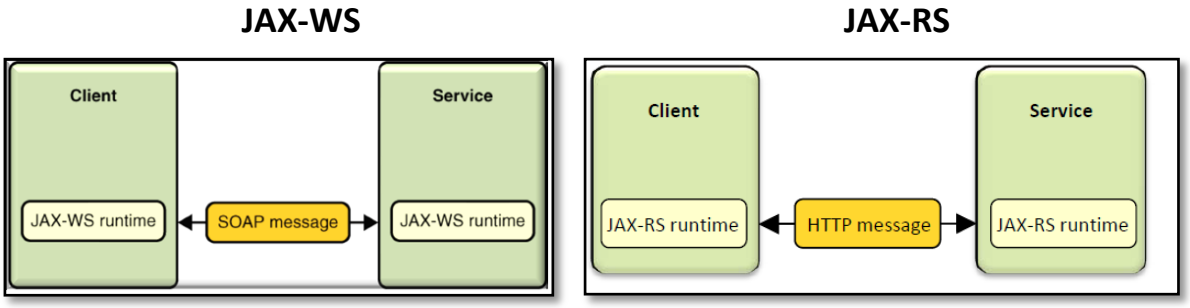
14.6.1 Folgerungen aus Architektursicht

- Web Services (SOAP) und REST sind für unterschiedliche Aufgabenstellungen konzipiert worden
  - Web Services (SOAP) sind für den Aufruf von Funktionen in verteilten Unternehmenssystemen konzipiert worden
  - REST ist für Zugriff und Manipulation von Ressourcen in grossen Hypermedia-Systemen konzipiert worden (zzgl. Unterstützung des Hypermedia-Konzepts)
  - Beide Konzepte sind für die klassischen Aufgabenstellungen des anderen Konzepts nicht sonderlich gut geeignet
- Ein allgemeiner direkter Vergleich macht keinen Sinn
  - „Vergleich von Äpfeln mit Birnen“
- Scheinbare Konkurrenten stehen in keinem direkten Wettbewerb
  - Aussagen bzgl. “besserer” Eignung nur in Bezug auf eine konkrete Aufgabenstellung möglich

14.6.2 Empfehlungen aus Architektursicht

- Nutze Web Services, wenn
  - Funktionsaufrufe im Vordergrund stehen
  - besondere Anforderungen an Integrität, Security, etc. gestellt werden
- Nutze REST, wenn
  - Ressourcenzugriff und -manipulation im Vordergrund stehen
  - keine besonderen Anforderungen an Integrität, Security, etc. gestellt werden
  - es um Hypermedia-Anwendungen geht (Mashups)

14.6.3 Unterschied JAX-WS und JAX-RS



14.7 WSDL

WSDL wird häufig in Kombination mit SOAP und dem XML Schema verwendet, um Webservices im Internet anzubieten. Ein Client, der einen Webservice aufruft, kann WSDL lesen, um zu bestimmen, welche Funktionen auf dem Server verfügbar sind. Alle verwendeten speziellen Datentypen sind in der WSDL-Datei in XML-Form eingebunden. Der Quellcode, der zum Zusammensetzen der gesendeten Objekte auf der Client-Seite notwendig ist, kann automatisiert aus der WSDL-Datei generiert werden. Der Client kann nun SOAP verwenden, um eine in WSDL gelistete Funktion letztlich aufzurufen.

14.7.1 Elemente des XMLs

```
<definitions name="StockQuote"
         targetNamespace="http://example.com/stockquote.wsdl"
         xmlns:tns="http://example.com/stockquote.wsdl"
         xmlns:xsd1="http://example.com/stockquote.xsd"
         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns="http://schemas.xmlsoap.org/wsdl/">

   <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2001/XMLSchema">
         <element name="TradePriceRequest">
            <complexType>
                <all>
                    <element name="tickerSymbol" type="string"/>
                </all>
            </complexType>
         </element>
         <element name="TradePrice">
            <complexType>
                <all>
                    <element name="price" type="float"/>
                </all>
            </complexType>
         </element>
      </schema>
   </types>

   <message name="GetLastTradePriceInput">
       <part name="body" element="xsd1:TradePriceRequest"/>
   </message>

   <message name="GetLastTradePriceOutput">
       <part name="body" element="xsd1:TradePrice"/>
   </message>

   <portType name="StockQuotePortType">
       <operation name="GetLastTradePrice">
          <input message="tns:GetLastTradePriceInput"/>
          <output message="tns:GetLastTradePriceOutput"/>
       </operation>
   </portType>

   <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
       <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
       <operation name="GetLastTradePrice">
          <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
          <input>
              <soap:body use="literal"/>
          </input>
          <output>
              <soap:body use="literal"/>
          </output>
       </operation>
   </binding>

   <service name="StockQuoteService">
       <documentation>My first service</documentation>
       <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
          <soap:address location="http://example.com/stockquote"/>
       </port>
   </service>

</definitions>
```

Definition: element must be the root element of all WSDL documents. **It defines the name of the web service**, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.

Data types: the **data types** - in the form of XML schemas or possibly some other mechanism - to be used in the messages

Message: an **abstract definition of the data**, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.

Port type : an abstract set of **operations mapped to one or more end points**, defining the collection of operations for a binding; the collection of operations, because it is abstract, can be mapped to multiple transports through various bindings.

Binding: the concrete protocol and data formats for the operations and messages defined for a particular port type.

Port: a combination of a binding and a network address, providing the target address of the service communication.

Service: a collection of related end points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.

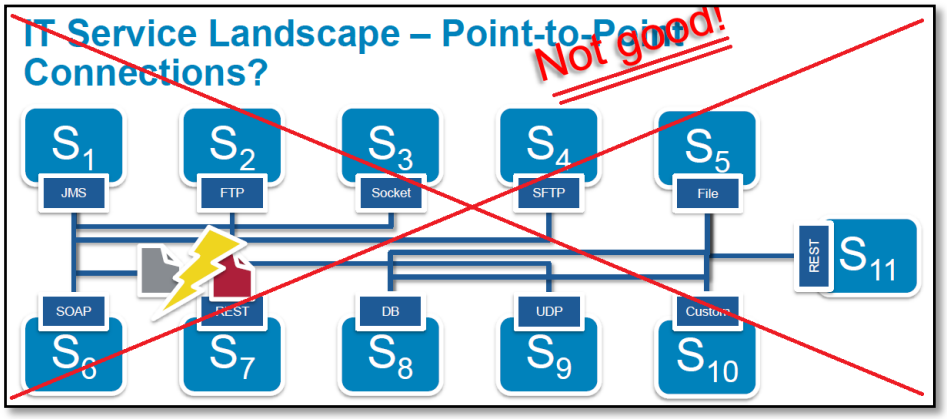
15. ESB Enterprise Service Bus

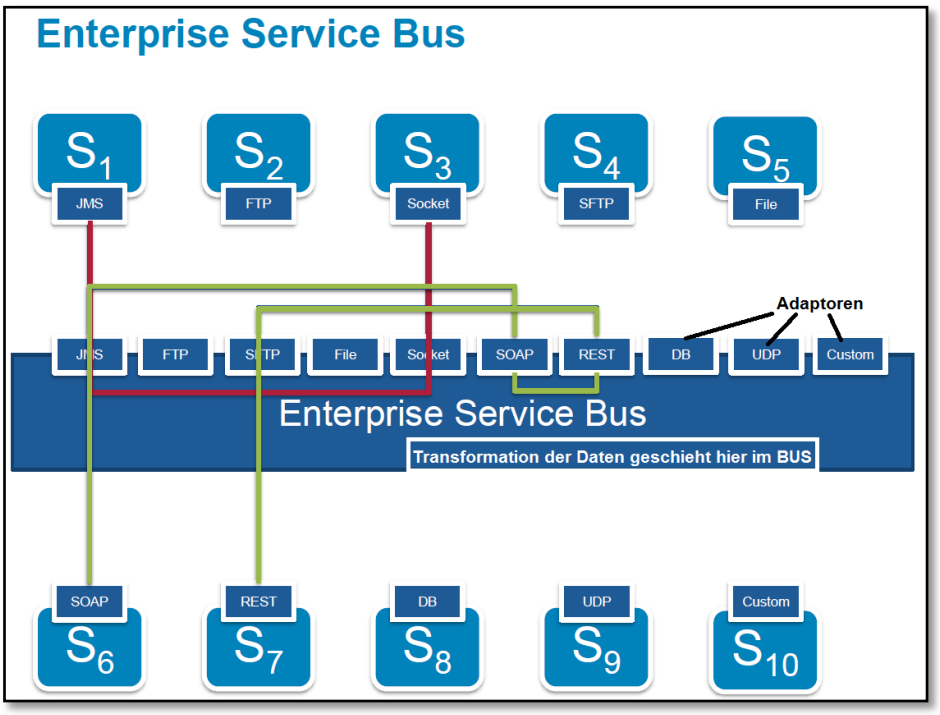
15.1 Definition

“An enterprise service bus (ESB) is a software architecture model used for designing and implementing the interaction and communication between interacting software applications in service-oriented architecture (SOA).“

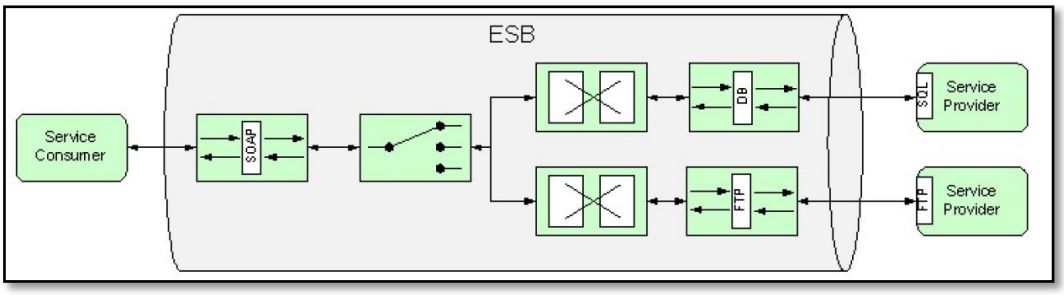
15.2 Wieso ein ESB?

- Services use different communication technologies such as JMS, SOAP, DB, REST, SFTP, FTP, File, Socket, UDP, Custom, etc.
- Different services use different data formats and encodings.
- IT becomes more and more key success factor for business.
- Business demands for new services to differentiate from competitors.
- Time-to-market is critical for the success of new services.
- Point-to-point integration results in custom integration code being spread among services with no central way to monitor or troubleshoot.
- It creates tight dependencies between applications.
- Reinventing the wheel over and over.



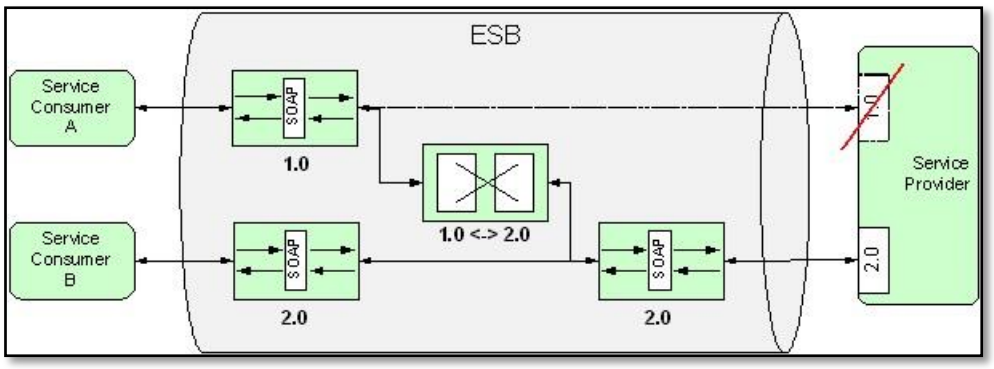


15.3 Wie sieht ein ESB von Innen aus?



- Based on information in the message from the service consumer the message is routed to one of the service provider. **=> Content-based routing**.
- Technology adapters mediate at protocol level to enable the message exchange.
- The message is **transformed** depending on the needs of the service provider.

15.4 ESB Versioning



- Service Consumer A has been using Interface 1.0 and doesn't want to switch to version 2.0, since it would not bring any added value to their business. However, the service provider does not want to keep running the two versions in parallel.
- The message is transformed by the ESB from versions 1.0 to 2.0 and sent to the new service.

16. Workflow

16.1 Konzept

16.1.1 Business Flow

Siehe Vorlesung 12 (Service Composition) S. 11

16.1.2 Workflow (What? Who? With? – Dimensionen)

Siehe Vorlesung 12 (Service Composition) S. 14

16.1.3 What? (Control-, Data- und Organization- Flow)

Siehe Vorlesung 12 (Service Composition) S. 17

16.1.4 Who? (Staff Assignement)

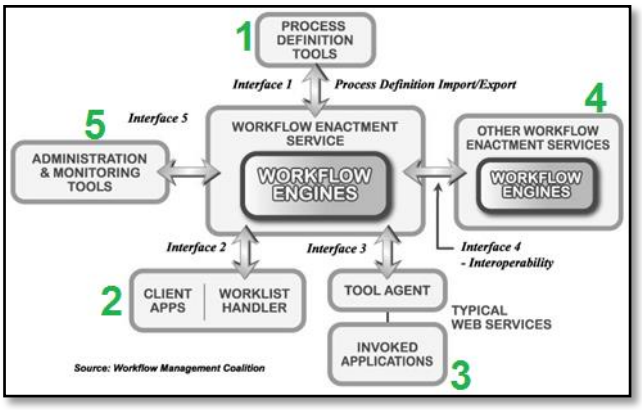
Siehe Vorlesung 12 (Service Composition) S. 20

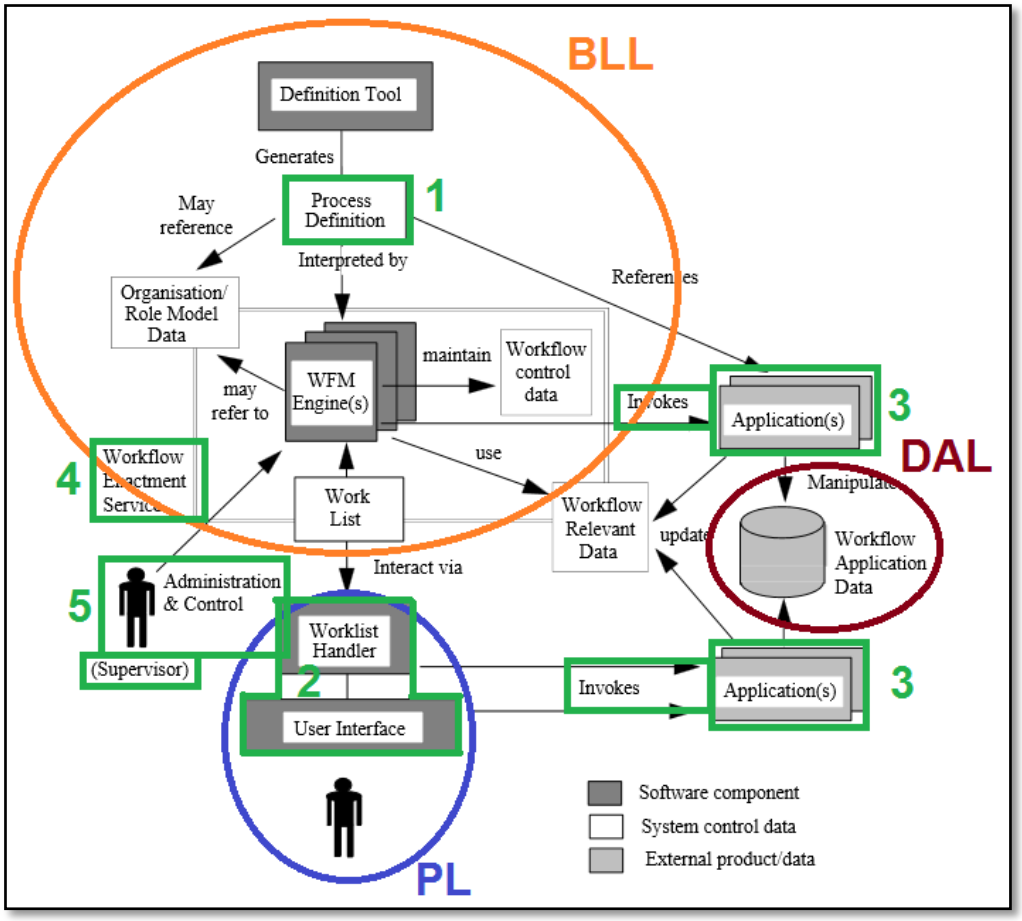
16.1.5 With?

Siehe Vorlesung 12 (Service Composition) S. 21

16.2 WfMC (Workflow Management Coalition) Reference Model Komponenten

Interface (Component)	Description
Process Definition Tools Interface (1)	Definition of a standard interface between process definition and modeling tools and the work flow engine(s).
Workflow Client Application Interface (2)	Definition of APIs for client applications to request services from the workflow engine to control the progression of processes, activities and work-items.
Invoked Application Interface (3)	A standard interface definition of APIs to allow the workflow engine to invoke a variety of applications, through common agent software.
Workflow Interoperability Interface (4)	Definition of workflow interoperability models and the corresponding standards to support interworking.
Administration & Monitoring Tools Interface (5)	The definition of monitoring and control functions.





16.3 BPMN Business Process Modelling Notation

16.3.1 Konzept

- Flow Objects:
  - For modelling nodes in a business process
  - Activities, gateways and events


Knoten

- Connecting Objects:
  - Sequence flow and message flow for connecting nodes, associations for additional information

Kanten

- Pools and Lanes
  - For partitioning sets of activities for business-to-business interaction

- Artifacts
  - Further elements such as data objects
  - **Datenobjekt**

16.3.2 Tasks (=Activity)

Eine Task ist eine atomare Arbeitseinheit. Sie repräsentiert die Aufgabe, die zu tun ist.

Different types of tasks:

- General task
- User task (staff assignment)
- Service task (Java, Web)
- Script task
- Receive task (asynchron)
- Send task (asynchron)
- Manual task

Loop Task

- Loop Condition
- while / repeat until

16.3.3 Gateways (AND, OR, XOR)

- "Normal" Gateway

- AND ("und")

Bei einer Verzweigung werden **alle ausgehenden** Kanten des Gateways simultan aktiviert.

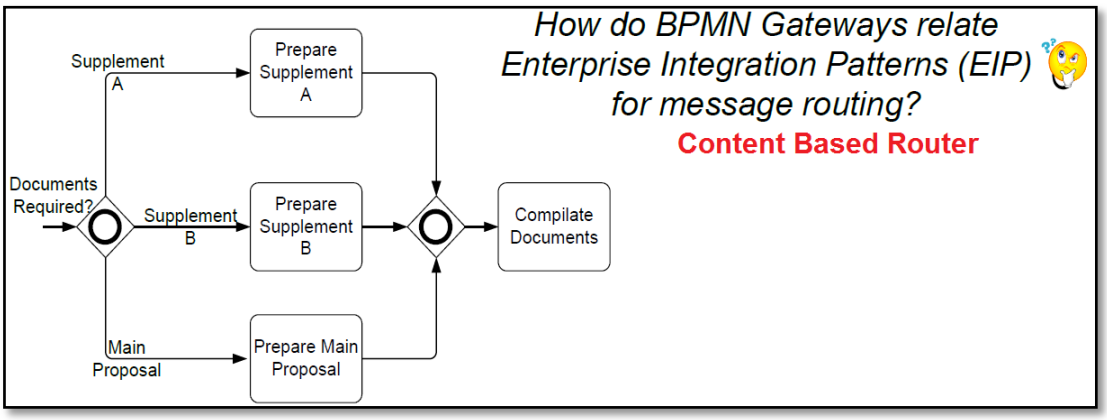
- XOR ("entweder-oder")

Abhängig von der Verzweigungsbedingung wird der Sequenzfluss **zu genau einer** ausgehenden Kante geleitet.

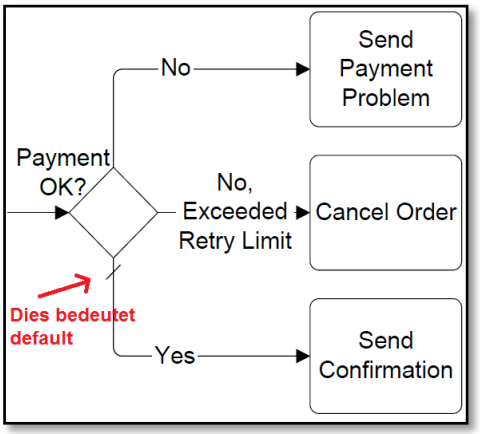
- OR ("oder")

Bei einer Verzweigung werden abhängig von Verzweigungsbedingungen **eine oder mehrere** ausgehenden Kanten aktiviert.

16.3.4 Inclusive Gateways Beispiel (AND, OR)



16.3.5 Default Sequence Flow



16.3.6 Special Events (Start, Intermediate, End)

Start:

- Hier beginnt der normale Prozessfluss durch eine nicht näher beschriebene Auslösebedingung.

End:

- Untypisiertes Endereignis, welches ein normales Ende eines Prozesses markiert.

Intermediate:

- Dieses Ereignis markiert das Erreichen eines definierten Zustandes im Prozess. Die Prozessausführung wird durch das Ereignis nicht verzögert.

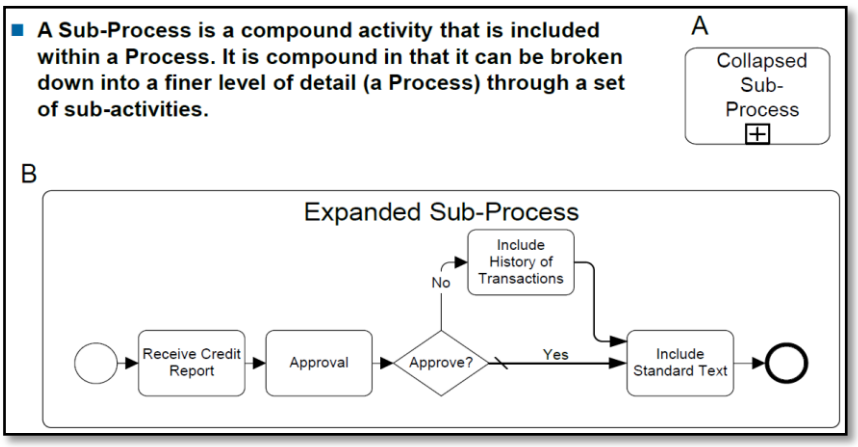
Intermediate events placed into normal flow:

- Catching or throwing an event

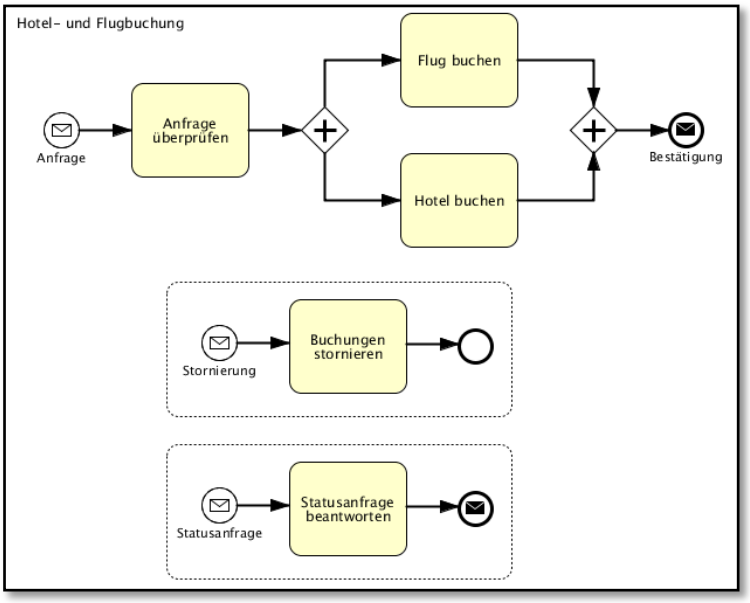
Intermediate events placed at the boundary of an activity:

- Exception flow, catching of an event

16.3.7 Subprozesse



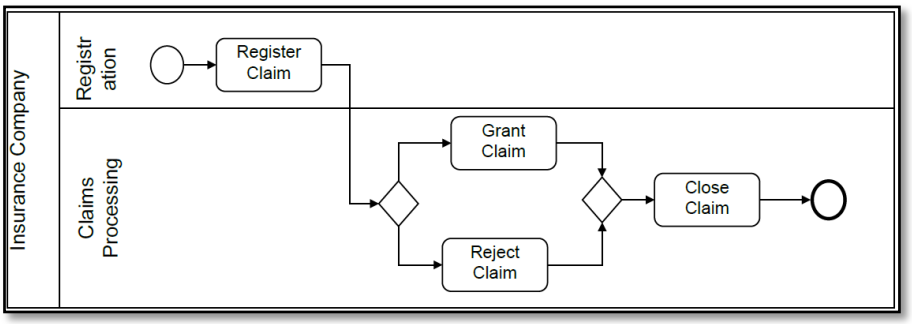
Event-Subprozesse (siehe unten Hotel- und Flugbuchung) sind optionaler Bestandteil von Subprozessen und dienen dem Verarbeiten von auftretenden Events innerhalb des Subprozesses. Event-Subprozesse werden von einem entsprechenden Start-Event initiiert und unterscheiden sich von klassischen Subprozessen insofern, dass sie nicht direkt im Kontrollfluss liegen. Vielmehr sind Event-Subprozesse freistehend, werden aber im Kontext des umgebenden Subprozesses ausgeführt.



16.3.8 Pools And Lanes

Name	Zeichen	Beschreibung
Pool (Pool)		Pools repräsentieren Organisationseinheiten und können durch Lanes weiter aufgefächert bzw. unterstrukturiert werden. **Sind mehrere Pools vorhanden, so laufen diese PARALLEL!**
Zugeklappter Pool (Collapsed Pool)		Ein zugeklappter Pool (anonym) versteckt den dahinter stehenden Prozess. Er kommt zum Einsatz, wenn der innere Ablauf eines Kommunikationspartners keine unmittelbare Rolle spielt.
Lane (Lane)		Pools und Lanes repräsentieren Rollen. Lanes repräsentieren Verantwortlichkeiten, wie etwa Organisationseinheiten, Stellen oder IT-Systeme. Lanes können hierarchisch in weitere Unter-Lanes gegliedert sein. **Lanes unterteilen Pools.**

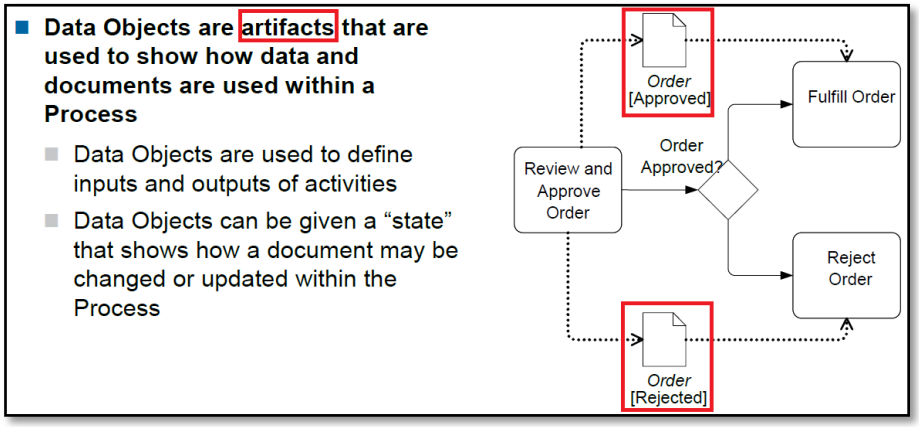
Beispiel:



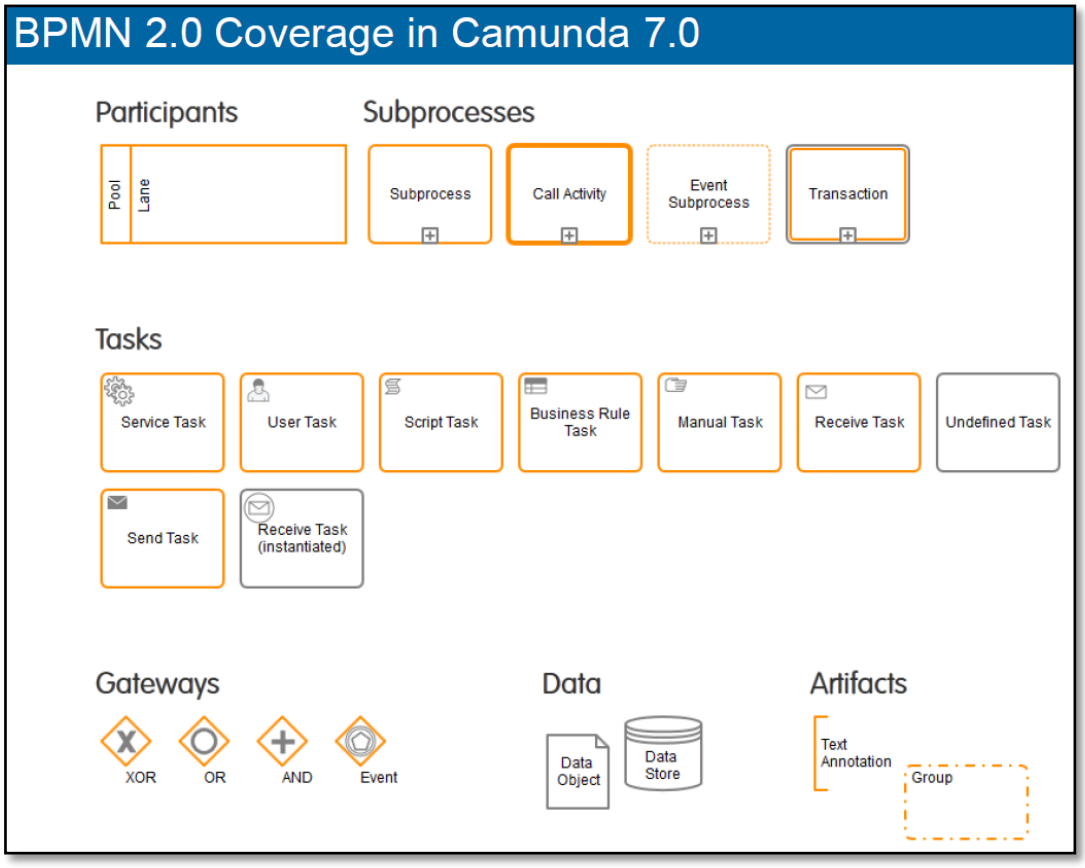
16.3.9 Connectors

Connector	Zeichen	Beschreibung
Sequenzfluss (Sequence Flow)	——▶	Ein Sequenzfluss definiert die Ausführungsreihenfolge von Aktivitäten.
Ungerichtete Assoziation (Association (undirected))	·····	Die Zuordnung eines Datenobjekts zu einem Sequenzfluss über eine ungerichtete Assoziation weist auf eine Informationsübergabe zwischen den Aktivitäten hin.
Gerichtete Assoziation (Association (unidirectional))	····▷	Eine gerichtete Assoziation zeigt den Informationsfluss. Ausgehende Kanten zeigen Schreiben, eingehende zeigen Lesen an.
Beidseitige Assoziation (Association (bidirectional))	◁····▷	Eine beidseitig gerichtete Assoziation zeigt, dass das Datenobjekt während der Ausführung einer Aktivität verändert wird, d.h. sowohl gelesen als auch geschrieben.
Nachrichtenfluss (Message Flow)	○····▷	Nachrichtenflüsse symbolisieren Informationen, die über organisatorische Grenzen hinweg fließen. Nachrichtenflüsse können an Pools, Aktivitäten oder Nachrichten-Ereignissen angeheftet sein. Die Abfolge des Nachrichtenaustauschs kann durch das Kombinieren von Nachrichtenfluss und Sequenzfluss spezifiziert werden.

16.3.10 Data Objects (Artifacts)



16.3.11 Camunda Overview

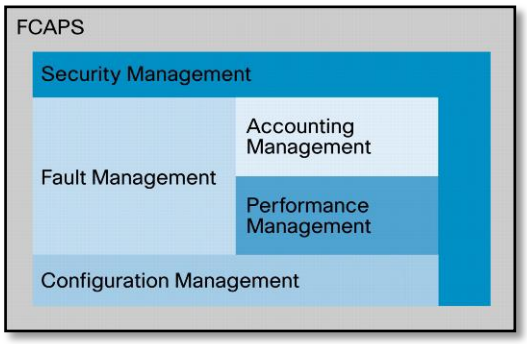


16.4 Praktische Tipps

- Keep the layers clear/clean
  - No business logic in integration flow (enterprise service bus)
  - No integration in process model (workflow management system)
  - Keep expressions simple and delegate complex computations to services and/or utility POJOs
- Remember the PoEAA/EIP connection when defining activity interfaces
  - E.g. business activity as PoEAA transaction script
  - E.g. Inclusive Gateway in BPMN combines Content-Based Router and Aggregator patterns (from EIP); process manager pattern in EIP book
- Use workflow engines only in suited domains/problem areas
  - Systems management, scientific computing, integration

17. Architekurmanagement

17.1 FCAPS (Management Disziplinen)



FCAPS	Beschreibung
Fault Management / Fehlermanagement	Das Ziel von Fault Management ist das **Erkennen, Isolieren, Beheben und Protokollieren von** im Netz aufgetretenen **Fehlern**. Isolieren der Störung meint: Im Störungsfall muss man die gemeinsame Ursache des Ausfalls von Leitungen feststellen können (root cause analysis).
Configuration Management / Konfigurationsmanagement	Configuration Management beinhaltet folgende Punkte: • **Sammeln und Speichern** von Konfigurationen von Netzkomponenten • **Vereinfachen der Konfiguration** einer Netzkomponente • **Aufspüren von Änderung** der Netzkonfiguration • **Konfiguration** von Leitungen oder Pfaden durch das Netz oder einen Teil des Netzes (sub network)
Accounting Management / Abrechnungsmanagement (Administration)	Das Accounting Management stellt brauchbare Statistiken über die Verwendung von Netzressourcen bereit. Damit können Kosten abgerechnet oder Quotas kontrolliert werden. Für nicht nach Benutzung abgerechnete Netze wird der Begriff ***Administration*** statt **Accounting** verwendet. **Administration** enthält die **Verwaltung von Benutzern, Passwörtern und Zugriffsberechtigungen**.
Performance Management / Leistungsmanagement	Performance Management erlaubt den Anwendern, das Netz für die Zukunft vorzubereiten. Durch **Sammeln** und **Analysieren** von Leistungsdaten kann die Stabilität des Netzes überwacht werden. Trends können auf zukünftige Probleme bei der Kapazität oder der Zuverlässigkeit des Netzes hinweisen.
Security Management / Sicherheitsmanagement	Die erste Aufgabe von Security Management ist die **Identifikation von Risiken aller Art** im Netz. Weiterhin müssen diese Risiken abgesichert und bewältigt werden.

17.1.1 Deployment Units

Deployment units can/should be **managed**

- Exceptions, errors (F in FCAPS)
- Component lifecycle, versioning (C in FCAPS)
- Usage statistics to serve as billing input (A in FCAPS)
- Response times, throughpout, resource consumption (P in FCAPS)

Deployment units can/should be **secured**

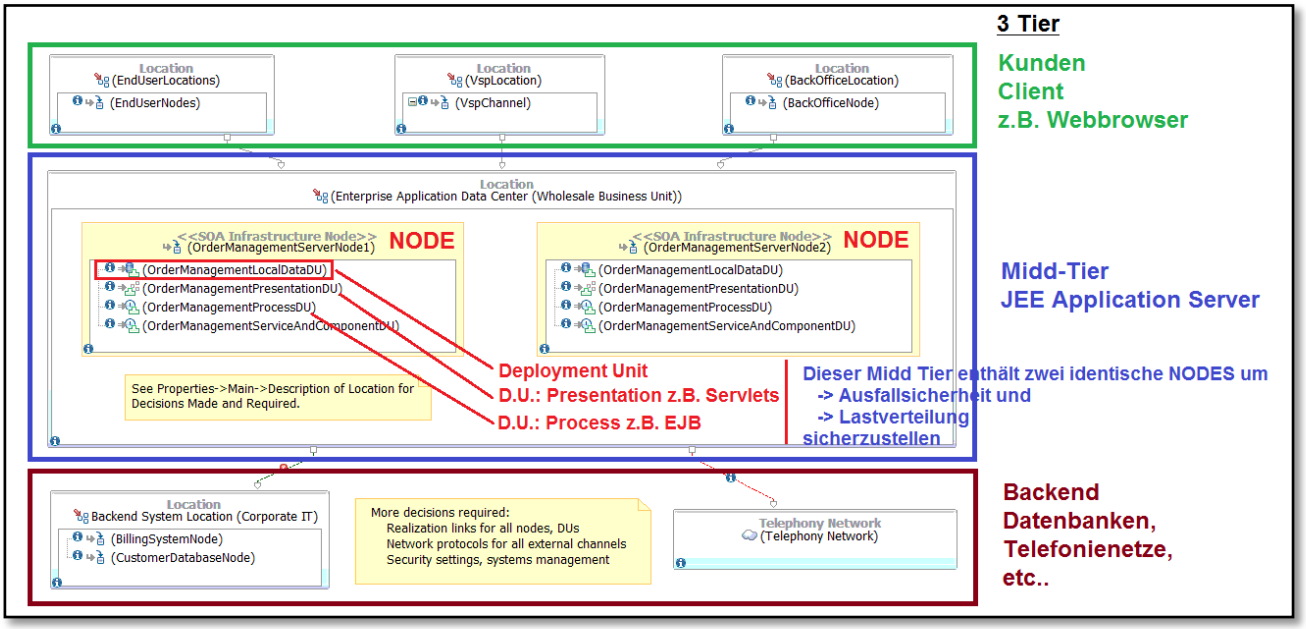
- Role-based access control (S in FCAPS)

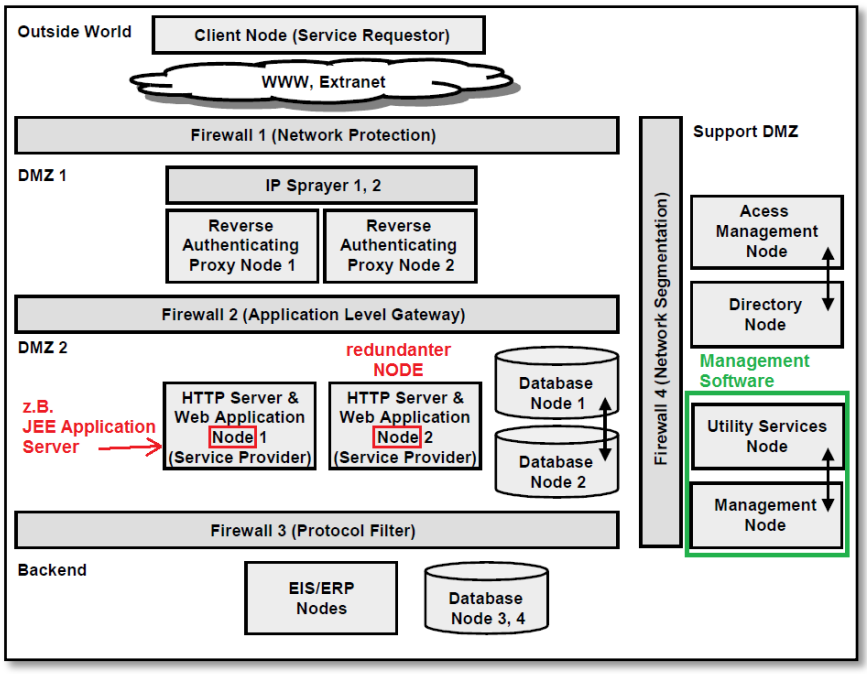
17.2 Operational Modell

- Das Operational Modell ist zum **designen** und **spezifizieren** der Infrastruktur.
- Ein wichtiger Begriff ist **Node**.
- Node ist eine Abstraktion einer Hardware, so z.B. eines Servers, virtuell oder physisch spielt keine Rolle.
- Wenn die Schichtentrennung beim Komponenten Modell korrekt geplant ist, dann kann man z.B. sagen, jetzt leg ich meinen Datenbankserver auf den **NODE X** oder meinen Application Server auf den **Node Y**. (siehe Bild)

„What is the relation between deployment units in an operational model and the JEE concepts on the presentation layer, business logic layer, data access layer?"

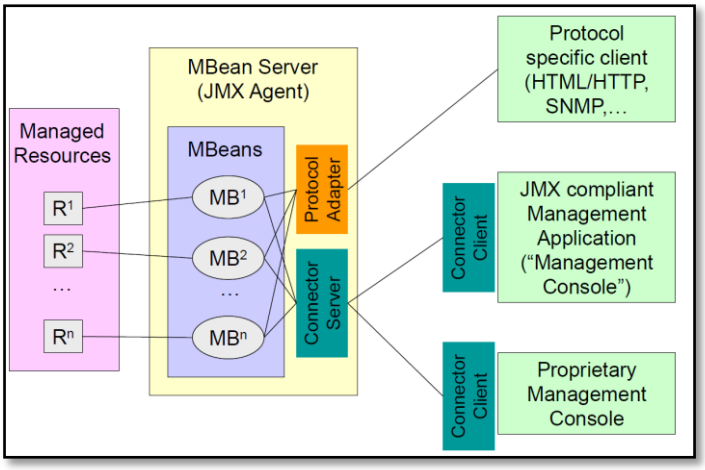
Die definierten Schichten in einem Komponenten Modell könen beim Operational Modell als selbstständige Deployment Unit spezifiziert werden. Auf einem Node laufen mehrere Deployment Units. (siehe Bild)

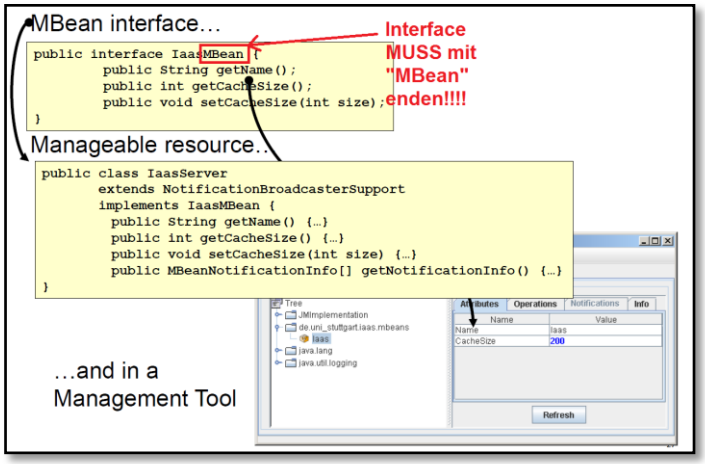




17.3 JMX Java Management Extension

Java Management Extensions (JMX) ist eine vom Java Community Process (JSR-3) entwickelte Spezifikation zur Verwaltung und Überwachung von Java-Anwendungen. JMX ist nicht nur eine geeignete Technologie, um das Verhalten von Systemen zu kontrollieren, sondern erleichtert auch die Kommunikation von unterschiedlichen Java-Programmen. In der ursprünglichen API unterstützte JMX nur die Kommunikation innerhalb einer JVM (Java Virtual Machine), aber seit der Java Version 6 wurde auch die Kommunikation mit anderen JVM unterstützt. Dies ist möglich durch die Unterstützung von Adaptern und Konnektoren. Damit lässt sich leicht ein HTTP-Adapter implementieren und die Anwendung ist über einen Webbrowser steuerbar. Sollte das System in ein schon bestehendes administriertes Netzwerk integriert werden, können Adapter für SNMP oder CIM/WBEM der Anwendung hinzugefügt werden.[2]





[2] http://de.wikipedia.org/wiki/Java_Management_Extensions

18. Q&A

1. **Welche technische Problematik ergibt sich aus der zusätzlichen RPC-Kommunikation (Tipp: Vor- und Nachteile von RPC wie z.B. in der VSS-Vorlesung vorgestellt)?**
   Zeitlich enge Kopplung (beide Systeme müssen gleichzeitig aktiv sein, um kommunizieren zu können), evtl. Performance- und Skalierbarkeitsprobleme; Interoperabilität muss sichergestellt werden (z.B. durch Verwendung von HTTP und JSON oder XML).
2. **Welche organisatorische Problematik ergibt sich aus der zusätzlichen RPC-Kommunikation (Tipp: was bedeutet das für die Gesamtsicht auf alle uses-Beziehungen zwischen den drei Unternehmensanwendungen hinsichtlich der Abhängigkeiten)?**
   Es entsteht eine zyklische Abhängigkeit, die Releases von Updates erschweren (verzögern) oder sogar verunmöglichen kann (Versionsmanagement, Testing); auch anspruchsvoll: Eigenentwicklung auf Hostsystem soll COTS-Software auf Windows aufrufen.
3. **Welches Designprinzip bzw. Pattern wird verletzt, wenn die Web-Anwendungen, die den Back Office Channel implementieren, direkt auf die Customer Database zugreifen (z.B. über JDBC)?**
   Layering/Layers (DB-Acceess-Code im Presentation Layer). Ausweg: über Risk Management Anwendung kommunizieren.
4. **Entscheiden Sie, mit welchen Pattern aus dem PoEAA-Buch von Fowler Sie den Business Logic Layer (BLL) und den Data Access Layer (DAL) der AW umsetzen wollen:**
   - BLL: Transaction Script oder Domain Model oder Table Module?
     - (optionale Zusatzfrage) Falls Domain Model: welche Domain-Driven Design Patterns sollen zur Anwendung kommen (also z.B. Aggregates, Entities und Value Objects)?
   - Transaction Script reicht hier aus, da weder komplexe Berechnungen noch Datenhaltung im BLL nötig sind. Für eine Diskussion der DDD Patterns im Kontext der gesamten PQG-Case Study siehe vorherige Übungen.
5. **Mit welchem API werden Messages versendet?**
   JMS
6. **Wie kann man eindeutige URIs pro Web Service erreichen?**
   In dem man jeweils nur eine Methode pro Web Service definiert (Command Pattern).
7. **Welches HTTP-Verb wird in SOAP 1.1/WSDL 1.1 verwendet? Wie sieht es in SOAP 1.2/WSDL 2.0 aus?**
   POST (WSDL 2.0 hat auch ein direktes HTTP Binding, wird aber kaum von Tools unterstützt).
8. **Ist es möglich, HATEOAS mit WSDL/SOAP zu realisieren? Gibt es evtl. sogar einen WS-* Standard, der sich zur Benennung (und damit Adressierung) von Services eignet?**
   Ja, das XML kann ja Links enthalten, z.B. XLinks oder auch WS-Addressing Konstrukte.
9. **In welchem JSE oder JEE API ist die folgende Funktionalität zu finden?**
   a. Datenbank-Queries: **JDBC**
   b. Mapping logische Namen auf Objektreferenzen: **JNDI**
   c. Entgegennahme von und Antwort auf HTTP-Requests: **Servlet API**
   d. Kapselung und Pooling von Business Logik Objekten: **EJB**
   e. Remote Zugriff auf Business Logik Objekte via RMI/IIOP: **EJB**
10. **Braucht es mit REST noch State Management?**
    Ja, je nach Anwendung, d.h. Wenn z.B. die Applikation über 10 Schritte geht, sollte man sich diese zwischenspeichern.

11. **Sie haben auf den Folien (W6 Seite 3) geschrieben: „JEE Applikationen Server oder Spring Framework können als globale Transaktionsmanager (TP-Monitore) fungieren“**
    a. Ich habe auf wikipedia (http://de.wikipedia.org/wiki/Transaktionsmonitor) folgendes gefunden: „Ein Transaktionsmonitor sollte nicht mit einem sog. Transaktionsmanager verwechselt werden. Während das Aufgabenspektrum eines Transaktionsmanagers lediglich das Bereitstellen eines transaktionalen Kontextes für verschiedene Anwendungen/Operationen umfasst, bieten Transaktionsmonitore transaktionsorientiertes Scheduling und Management für verschiedene Ressourcen.“
    b. Meine Frage ist, habe ich Ihre Aussage falsch interpretiert oder war der Kontext anders?

    Antwort: Der Kontext stimmt und Sie haben die Vorlesungsfolie richtig verstanden. Ich habe im Unterschied zu dem Wikipedia-Zitat nicht zwischen Transaktionsmanager und Transaction Processing (TP) Monitor unterschieden. Dies kann man tun, muss man aber nicht. Die Literatur ist uneinheitlich, das Wikipedia-Zitat ist schon in sich leicht widersprüchlich, da es dem Monitor Management-Features attestiert. Das IBM-Redpaper „Transactions in J2EE
    This“ auf dem Skriptserver z.B. weist dagegen dem Transaktionsmanager diese Management-Aufgaben zu.

    Für die Prüfung ist die Unterscheidung aber nicht relevant.
12. **Zur Frage auf der Folie W5 Folie 49 „Was/wo in JEE ist das AP? Der/die RM? Der TM?“**
    a. Application Program -> Servlet Antwort: Stimmt, und auch EJB-Implementierungen (durch Anwendungsentwickler erstellt)
    b. Ressource Manager -> Datenbanksystem (z.B. SQL Server) Antwort: Stimmt
    c. Transaktion Manager -> EJB Container Antwort: Stimmt, oder auch Web-Container (z.B. wenn man JTA benutzt). JEE-Applikationsserver würde auch als Antwort reichen (denn dieser enthält ja die beiden Container).
13. **Ist beim Point-To-Point Channel die Kommunikation zwischen Sender und Receiver IMMER asynchron?**
    Dies ist eine definierende Eigenschaft des Integrationsstils Messaging, also nicht channel-spezifisch.
14. **Kann der Subscriber beim Publish-Subscribe Channel eine Message NUR EIN MAL empfangen? Und ist diese Kommunikation auch IMMER asynchron?**
    Siehe Antwort 13, Messaging-Kommunikation ist immer asynchron (im Sinne der zeitlichen Autonomie). Das Empfangsverhalten hängt von den gewählten Endpoint-Patterns und API-Calls ab (Bsp. JMS: Browsen vs. konsumierendes Lesen). Siehe Link in Antwort 10 für gängige Kombinationen. Subscriber ist dabei eine logische Rolle, eine Anwendung kann u.U. mehrere Subscriber enthalten.
15. **Haben beim Transactional Client Pattern der Sender und Receiver JEWEILS EINE EIGENE Transaktion oder ist dies EINE GEMEINSAME Transaktion.**
    beides möglich, man muss aber die Verteilung im Auge behalten: normalerweise liegen Sender und Empfänger ja auf verschiedenen
    Servern und stehen mglw. nicht unter der Kontrolle/Koordination desselben Transaktionsmanagers (der dafür aber noch andere resourcenverändernde Aufrufe managed, z.B. das SQL INSERT eines aus einer Message Queue entnommenen Kundenobjekts in eine lokale Datenbank).

16. **Stimmt diese Aussage bezüglich dem Unterschied von Publisher-Subscriber-Pattern und Recipient List Pattern: „Bei Recipient list wird anhand von dem Message Content entschieden welche Empfänger die Message erhalten und bei Publisher-Subscriber wird jede Message an die Subscriber weitergeleitet. Weiterführend ist die Liste der Subscriber statisch, wobei die Liste der „Recipients“ dynamisch angepasst werden kann und immer pro Message geschaut wird wohin sie weitergeleitet wird.“**
    Nicht ganz, beide Pattern haben dynamischen Charakter. Die Konfiguration der Recipient List kann, muss aber nicht aus dem Content der Message kommen (kann auch anders erstellt werden, lokales Konfig. File oder separater Control Channel). Und lt. Patternscreibung im Buch erhält dann jeder Recipient eine Kopie. Subscriber können auch zur Laufzeit hinzukommen und verschwinden.
    Seite 255 im EIP-Buch hat eine Vergleichstabelle.

19. Glossar

Begriff	Ähnliche Begriffe / Erklärungen

POJO	Ein POJO (Plain Old Java Object) ist ein Java-Objekt, das keinerlei Einschränkungen bis auf die der Java Language Specification hat. Das heisst, ein POJO sollte keine:
	1. vorspezifizierte Klasse erweitern, wie z. B. hier:
	`public class Foo extends javax.servlet.http.HttpServlet { …`
	2. vorspezifiziertes Interface implementieren, wie z. B. hier:
	`public class Bar implements javax.ejb.EntityBean { …`
	3. vorspezifizierte Annotation enthalten, wie z. B. hier:
	`@javax.ejb.Entity`
	`public class Haus { …`

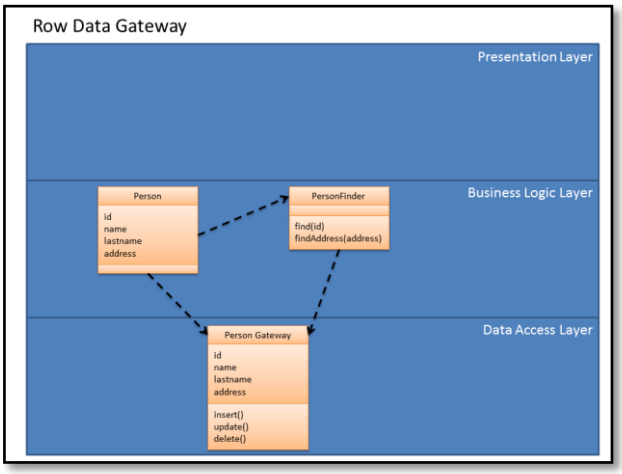
MOM	Messaging-oriented Middleware is a software system that provides Messaging; it is comparable with database systems that manage data persistence.
Considerations	Erwägung
Kardinalität	Beziehung 1:1 1:n

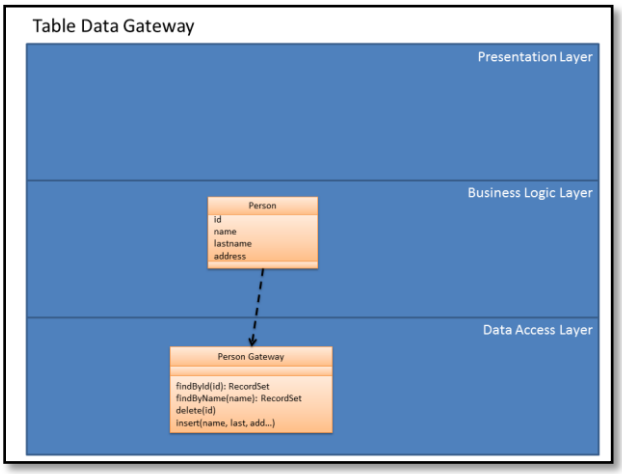
20. Anhang

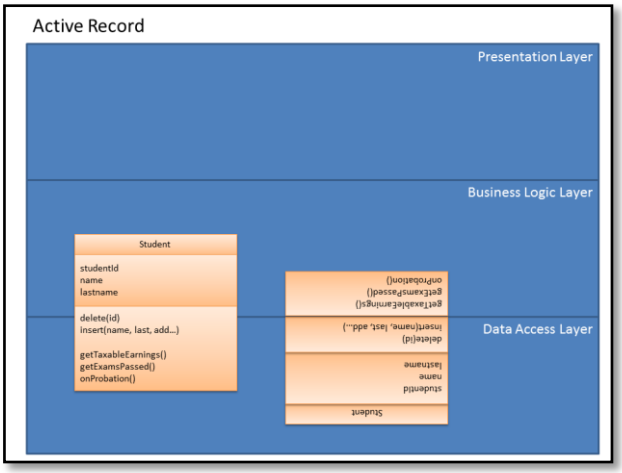
19.1 ACID

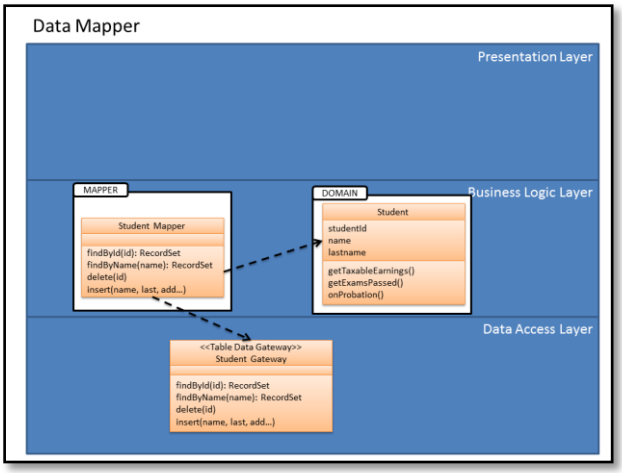
Begriff	Beschreibung
Atomicity	All actions performed within a transaction either all succeed or are all Undone. *Folge von Operationen entweder ganz oder gar nicht ausgeführt. Keine Teilweise Ausführung.*
Consistency	A transaction transforms resources from one consistent state to another consistent state. *Eine Transaktion führt die Daten von einem konsistenten Zustand in den anderen über.*
Isolation	The program realizing a transaction is isolated from all other actions within the system. *Eine Transaktion soll so ausgeführt werden, als sei sie isoliert von anderen.*
Durability ***Dauerhaftigkeit***	Once successfully completed the effects of a transaction will survive any failure. *Änderungen einer Transaktion sind dauerhaft. Sie dürfen nicht auf Grund von Fehlern verloren gehen.*

19.2 Row Data Gateway, Table Data Gateway, Acvtive Record, Data Mapper









21. Index

Q

R

S

T

U

V

W

Y