

## SOA – Service Oriented Architecture

- ist ein Architekturstil der es ermöglicht Geschäftsprozesse aus verschiedenen über Netzwerk angebotenen Services zusammenzustellen. Die Quelle des Service ist dabei variabel, kann also von beliebigen Systemen stammen. → Wiederverwendbarkeit von bereits bestehenden Diensten

### Eigenschaften dieser Architektur:

#### Lose Kopplung (geringe Kopplung)

d.h. schmale, eindeutige Schnittstellen der Angebotene Dienste

- dies garantiert hohe Portabilität und Wiederverwendbarkeit,
- Veränderungen an Diensten können ohne große Veränderungen an anderen Diensten realisiert werden, da diese unabhängig von einander fungieren

Realisierung durch:

#### Middlewareplattformen

Nachteil: Plattformübergreifende Kommunikation aufgrund verschiedener Transportprotokolle und Objektmodelle sehr schwierig.

Vorteil: komponentenbasierte Systeme, mit eben geringer Kopplung

#### Message-basierte Übertragung

d.h. die zu kombinierenden/integrierenden Dienste kommunizieren untereinander,

hierfür ist notwendig:

- ein Adapter zum Erstellen der zu Versendenden Nachrichten
  - ist für jeden Dienst gesondert zu implementieren
  - entnimmt dem Sender die Nachricht, und übergibt sie an den Übertragungskanal
- ein Übertragungskanal
  - implementiert eine **asynchrone, zuverlässige** Übertragung
  - transformiert die Nachricht in ein Format das vom Empfänger verstanden werden kann
- ein Adapter zum Decodieren der empfangenen Nachrichten, und zur Verfügungstellung des Dienstes
  - ist für jeden Dienst zu implementieren
  - entnimmt die Nachricht dem Übertragungskanal, und übergibt sie dem Dienst

#### Publish-Find-Bind-Prinzip:

hierbei geht es um die Vermittlung und das Auffindbarmachen von Diensten, es legt

drei wesentliche Rollen zugrunde:

Rollen:

- Service **Provider**  
(dieser bietet einen Service an)
- Service **Consumer**  
(ist eine Anwendung die einen Service benutzen möchte)
- Service **Discovery**  
(ist der Vermittler zwischen Provider und Consumer und enthält ein Verzeichnis der verfügbaren Services)

**Publish** als Teil 1 des Prinzips:

- Service-Provider veröffentlicht seinen Service  
(er übermittelt in einem Contract eine Beschreibung des Service an einen/mehrere Service Discoveries)  
Contract
  - beschreibt Funktionalität
  - enthält Informationen die notwendig sind um den Dienst zu nutzen

(Bsp: Serverendpoint-Adresse oder physikalische Kommunikationseigenschaften)

**Find** als Teil 2 des Prinzips:

- der Service-Consumer benötigt einen Service, er teilt dem Service-Discovery mit, welche Bedingungen der Service erfüllen muss
- die Service-Discovery erstellt jetzt eine Liste der ihm bekannten Services zusammen mit einer Leistungsbeschreibung, die die gegebenen Bedingungen erfüllen und sendet sie dem Service Consumer zurück
- der Service-Consumer entscheidet sich für einen Kandidaten und teilt die Entscheidung der Service-Discovery mit

**Bind** als Teil 3 des Prinzips:

- Service Discovery übermittelt einen Contract an den Service-Consumer
- der Service Consumer erhält in dem Contract die Informationen, die nötig sind um den Service zu binden.
- Binden bedeutet hierbei, dass der Consumer die Serviceanfrage an das vom Provider erwartete Transportprotokoll und Nachrichtenformat bindet.
- der Service kann jetzt benutzt werden, also der Provider führt den Service mit den zum ausführen notwendigerweise übermittelten Daten aus, und liefert das Ergebnis zurück.

zum Contract:

- jede Anfrage muss dem im Contract festgelegten Format entsprechen
- vertragsgebundene Vor- und Nachbedingungen müssen erfüllt sein
- nichtfunktionale Eigenschaften müssen auch erfüllt sein
- alle Informationen die der Consumer benötigt werden zur Laufzeit zur Verfügung gestellt
- Provider und Consumer stehen also in einer direkten unbeeinflussten Abhängigkeit

## REST-Architektur (Representational State Transfer)

– ist ein Architekturmodell, welches die Realisierung eines Webservices auf Basis der Webarchitektur beschreibt

Eigenschaften der Architektur:

Ressourcen:

- sind die Objekte der Anwendung, alles was über eine URI angesprochen werden kann (Bilder, Script, PDF-Files, htm-Seiten usw...)
- realisieren den Webservice
  - ein Service wird hierbei im eigentlichen Sinne nicht geleistet, er wird realisiert, durch das aufrufen von Ressourcen
- diese Ressourcen sind erreichbar über http, somit ist die Interaktion zwischen Service-Nutzer und Anbieter von Haus aus statuslos, ist ein Status von Nöten, so muss dieser in der Kommunikation immer mit übertragen werden
- Ressourcen werden nicht direkt manipuliert, sondern werden über die URI angesprochen, und werden auch über die URI manipuliert

Repräsentationen:

- wie die Ressourcen repräsentiert werden, wird in REST nicht vorgeschrieben jedoch Server und Client müssen ein gemeinsames Verständnis über die Repräsentation besitzen
  - Bsp: Eine Anwendung ruft als Ressource ein Bild auf, erwartet dieses als Binärcode eines \*.gif -Files übermittelt wird aber eine \*.jpg kodierte Datei in Hexadezimal,... Die Anwendung wird damit ziemlich sicher nix anfangen können.

Zustand (state):

- Repräsentationen von Ressourcen können auf andere Ressourcen verweisen, der Client folgt hierbei einem Link, und gelangt so von einem Zustand in einen anderen

Verben:

- als Verben werden die durch http zur Verfügung gestellten Methoden „GET“ ; „POST“; „PUT“; und „DELETE“ bezeichnet
- GET:
  - frei von Seiteneffekten
  - lässt caching zu
  - fordert eine Repräsentation einer Ressource an
- POST:
  - verursacht Seiteneffekte
  - verändert z.B. Datenbankeinträge oder fügt einem Warenkorb Elemente hinzu
- PUT:
  - ermöglicht das erstellen neuer Ressourcen
  - muss nicht implementiert sein, beziehungsweise kann nur für eingeschränkte Nutzergruppen verfügbar sein
- DELETE:
  - ist eine Empfehlung zum Löschen einer Ressource
  - diese Empfehlung ist nicht bindend, das heißt über das Löschen der Ressource kann serverseitig entschieden werden.

Rollen:

- Client
  - verwaltet seinen Status selbst
  - entscheidet selbst über die Reihenfolge der aufzurufenden Methoden und Ressourcen
- Server

- kennt seinen Status
- kennt den Status des Clients nicht

	SOA	REST
Vorteile/Nachteile/Security:	- wird meist über gekapselte Protokolle wie bspw. SOAP realisiert,... http dient hier als reines Transportprotokoll - eine generelle Ausfilterung von SOAP ist möglich durch die Firewall, jedoch nicht eine ausgewählte Filterung, da die Firewall im Allgemeinen keine Kenntnis über die Struktur der übertragenen Nachrichten besitzt - aus Sicherheitsgründen könnte SOAP somit Opfer der Firewalls werden - kein Cachen möglich → skaliert mies - Nachrichten nicht direkt adressierbar - müssen erst von einem Dispatcher ausgewertet werden (SOAP-Router) und weitergeleitet - ist protokollunabhängig, ein Dienst kann über verschiedene Protokolle/Schnittstellen erreichbar sein (Bindings)	- praktisch keine Probleme mit Firewalls, da http wird für das surfen im Internet quasi immer zugelassen - über Firewalls lässt sich der Zugriff mit bestimmten Methoden beschränken, zum Beispiel könnten von extern nur GET Anfragen zugelassen werden - es lassen sich auch bestimmte URLs ausfiltern - die Bedeutung einer Nachricht geht aus den HTTP-Anfragen hervor und können im Log des Webservers verfolgt werden - ein generelles Blockieren von Rest ist nicht möglich, außer man blockiert den Webzugriff komplett - Anwendung hat keine Grenzen, Anwendungswechsel fordert keine Propagierung des Status, da die Kommunikation stateless ist - leicht erweiterbar, neue Technologien können ohne Probleme dank MIME hinzugefügt werden, und zwar ohne Auswirkung auf bereits bestehende Service - erweitern von bestehenden Services durch einfaches hinzufügen/austauschen von Ressourcen möglich - Komposition von Diensten, durch einfache Ausweitung des Adressraumes, auf die den weiteren Dienst implementierenden Ressourcen - nicht Serveraffin (folgender Request kann von anderem Server verarbeitet werden als der davor → Skalierbarkeit) - immer an ein Protokoll gebunden
Merkmale:	- Consumer sagt was er will nicht wie er es will, es ist deskriptiv nicht instruktiv - Zur Erweiterung des Dienstes ist altes Übertragungssystem nicht gebrauchen → neuer Webservice notwendig → Schrittweise Evolution nicht möglich	- Kommunikation erfolgt auf Abruf, aktiver Client, passiver Server - Ressourcen besitzen eine URI - Repräsentationen von Ressourcen können als Dokument vom Client gefordert werden - jede Serveranfrage muss alle Informationen enthalten, die zum Interpretieren der Nachrichten nötig sind - Caches werden unterstützt, der Server kann seine Antwort als cachefähig oder nicht cachefähig kennzeichnen - nur schlecht für Anwendungen geeignet die eine geringe Verzögerungszeit besitzen müssen,... Bsp. Application Server und DB,... hier ist zum Beispiel DCOM besser geeignet.

## SOAP – ursprünglich „Simple Object Access Protocol“ jetzt Eigennamen ohne Bedeutung

- ist kein Protokoll sondern lediglich eine Messagearchitektur mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können.

Eigenschaften dieses Protokolls:

- es läuft Huckepack auf einem Transportprotokoll und verkapselt so SOAP-Dokumente
- der Weg (message path) muss nicht direkt vom Sender (initial sender) zum Empfänger (ultimate receiver) verlaufen, sondern kann über mehrere Relay-Stationen (intermediares) geleitet werden, die Reihenfolge der Stationen ist nicht vorgegeben

Aufbau eines Soap-Dokumentes:

- eine Soap Nachricht ist ein XML-Dokument
- bestehend aus 3 Wesentlichen Elementen **Envelope, Header, Body**
  - **Envelope**
    - ist das Wurzelement
    - enthält **Header, Body**
    - innerhalb des Envelope's können XML Tags benutzt werden, die SOAP-Elemente enthalten
  - **Header**
    - optional
    - unterteilbar in Headerblöcke
    - enthält Anwendungsspezifische Daten, bspw. für Authentisierung, Transaktionen und Routing, Metadaten halt
    - beim Routing können die Header durch die Zwischenknoten untersucht/verändert/gelöscht werden
      - Die Verarbeitung:
        - Header werden eingelesen,... verarbeitet, nach der Verarbeitung müssen die Header gelöscht werden, wenn ein Löschen nicht wünschenswert ist, so kann der gelöschte Block vor Weiterleitung wieder eingefügt werden
      - die Headerblöcke:
        - Attribut 1 (role)
          - können Rollen besitzen, anhand der die Station weiß ob sie den Header verarbeiten soll
          - hierzu wird dem env:Actor-Attribut die entsprechende URI zugewiesen
          - 3 Verschiedene Standard-Rollen
            - none** – der Header muss nicht verarbeitet
            - next** - fordert die aktuelle Station soll den Header verarbeiten
            - ultimate Receiver** – die Nachricht darf ausschließlich vom End-Empfänger verarbeitet werden oder eine selbst definierte – denk dir halt was aus ^^
      - Attribute 2 (must Understand)
        - env:mustUnderstand auf 1 gesetzt besagt, ob die Zwischenstation den Header erfolgreich verarbeiten muss
          - falls eine Station einen erforderlichen Block nicht verarbeiten kann, muss er einen env:fault erzeugen und zurückschicken → die Nachricht wird nicht weiter verarbeitet und nicht weitergeleitet
      - Attribut 3(relay)
        - optional, ob ein nicht verarbeiteter Headerblock weitergeleitet werden muss

## • Body

- unterteilbar in Bodyblöcke und einen optionalen SOAP-Fault Block
- enthält die eigentliche SOAP-Nachricht also beliebigen wohlgeformten Namensraum-qualifizierten XML-Code
- Inhalt ist in der Regel nur für den End-Empfänger bestimmt
  - SOAP-Fault Block
    - optional
    - enthält Tag mit dem (Fehler-) Code (als Value) und Tag mit dem Grund (Reason) (als Text)
    - enthält optional eine „Role“ die den Verursacher des Fehlers identifiziert, wenn der Fehler auf dem Weg zum Empfänger aufgetreten ist
    - enthält optional das Feld „Detail“, dieses sollte gefüllt werden, wenn der Body nicht fehlerfrei bearbeitet werden konnte

## • SOAP für RPC

- im SOAP-Body stehen die serialisierten Informationen für den entfernten Methodenaufruf: Methodennamen samt Signatur serialisiert heißt: Objekte sind in ein Format überführt, das gespeichert oder versendet werden kann  
Informationen heißt:
  - für einen erfolgreichen Aufruf werden ein Methodennamen,
  - eine optionale Methodensignatur,
  - die Parameter für die Methode und
  - optionale Header-Informationen benötigt
- asynchrone Kommunikation mit „GET“ synchrone mit „POST“
  - synchron entspricht dem entfernten Methodenaufruf im klassischen RPC-Umfeld (Request/Response-Muster) ist aber nicht vorgeschrieben
- Struktur des SOAP-Bodys bei RPC
  - eine Methodenaufrage wird als Struktur modelliert
  - die Namensgebung der Struktur entspricht dem Methodennamen
  - Jeder Parameter (egal ob „in“ oder „in/out“) wird als Kindelement dargestellt, **der Rückgabewert ist der Erste der Liste**
  - die Namensgebung der Kindelemente entspricht den Namen der Methodenparametern
  - das gleiche gilt für die Antwort

## Probleme mit SOAP

- Inkompatibilität bzw. eingeschränkte Interoperabilität zwischen Server und Client möglich wegen:

- Unterschiedlicher Formatierungen im SOAP-Body (RPC oder document)
- Darstellung der serialisierten Datentypen kann entweder encoded oder literal erfolgen

## IDL – Interface Description Language

- ist eine abstrakte rein funktionale Beschreibung der Schnittstellen einer Anwendung,
- es werden keine Aussagen getroffen zu nichtfunktionalen Eigenschaften
- es werden keine Angaben gemacht, zur Erreichbarkeit der Schnittstellen
- eine mit IDL erstellte Schnittstelle, lässt sich automatisiert transformieren in eine programmiersprachenspezifische Schnittstelle
- eine mit IDL erstellte Schnittstelle ist abstrakt gehalten, also selbst nicht in einer programmiersprachenspezifischen Schreibweise angegeben

## WSDL – Web Service Description Language

-definiert eine plattform-, [programmiersprachen](#)- und [protokollunabhängige XML](#)-Spezifikation zur Beschreibung von Netzwerkdiensten ([Web Services](#)) zum Austausch von Nachrichten.

beschrieben werden:

- die angebotenen Funktionen
- Daten
- Datentypen
- Austauschprotokolle
- von „außen“ zugängliche Operationen inkl. Parameter und Rückgabewerte

nicht beschrieben werden:

- Quality-of-Service-Informationen
- Taxonomien/Ontologien zur semantischen Einordnung des Service

Aufbau:

- besteht aus den 6 Elementen Datentypen (**types**), Nachrichten (**message**), Port-Typen(**portType**), Bindung (**binding**), Ports (**port**), Services (**service**)
- alle bis auf **types** dürfen mehrfach vorkommen → unterschiedliche Services mit unterschiedlicher Funktionalität und unterschiedlichen Protokollen möglich
- **types (Datentypen)**
  - Definition der Datentypen die zum Austausch der messages benutzt werden
- **message (Nachrichten)**
  - beschreibt die Nachrichten die zwischen Requestor und Provider ausgetauscht werden können.
  - die Richtung ist hier unerheblich
  - besitzen einen Namen
  - besitzen optional beliebig viele „**Part**“-Tags
    - „**Part**“-Tags bestehen aus 2 Attributen, „**name**“ und „**type**“
- **portType (Port-Typen)**
  - beschreibt die Schnittstelle des Services
  - besitzen einen Namen
  - eine Menge von abstrakten Arbeitsschritten werden zu Operationen zusammengefasst
  - abhängig von den Ein- bzw. Ausgabeparametern gibt es in WSDL 4 Nachrichtenaustauschmuster (Message Exchange Patterns)
    - **One-Way**
      - Service Provider bekommt einen INPUT von einem Service Requestor
    - **Request-Response**
      - Service Provider bekommt einen INPUT von einem Service Requestor
      - der Service Provider antwortet mit einem OUTPUT an den Service Requestor
    - **Solicit-Response**
      - Service Provider startet den Austausch mit einem OUTPUT an dem Service Requestor
      - der Service Requestor antwortet mit einem INPUT an den Service Provider
    - **Notification**
      - der Service Provider übermittelt lediglich einen OUTPUT an den Service Requestor
- **binding (Bindung)**
  - **port-Types** werden hierdurch an eine Serialisierungsvorschrift und ein Transportprotokoll

gebunden

- **service (Services)**
  - stellt aus einem oder mehreren ports einen Dienst zusammen
- **port (Ports)**
  - geben die Netzwerkadresse an, unter der das in ihnen benannte Binding (welches in diesem WSDL-Dokument auch definiert ist) zu finden ist

Man unterscheidet diese Basiselemente in **abstrakte Definitionen**

- **types, message, portType** (hier wird geklärt „was“)

und **konkrete Definitionen**

- **binding, service** (hier wird geklärt „wie“)

## AJAX – Asynchronous Javascript And XML

beschreibt eine Technologie, mit der Daten zwischen Client und Server mittels Javascript ausgetauscht werden können, sodass es reicht nur Teile anstatt die Ganze Web-Seite neu zu laden.

Vorteile	Nachteile
<ul style="list-style-type: none"><li>- man kann Dokumententeile nachladen/austauschen</li><li>- Serverlast wird verringert</li><li>- Änderungen werden schneller sichtbar → schnelleres Feedback für den User</li><li>- verbesserte Usability</li><li>- Zustand bleibt beim Nachladen erhalten (Cursor, Positionen von Elementen etc.)</li><li>- Desktop ähnliche Oberflächen sind schaffbar</li><li>- aufwendigere Webapplikationen möglich, Unabhängigkeit von lokal installierten Programmen</li></ul>	<ul style="list-style-type: none"><li>- setzt JavaScript Aktivierung voraus → hauptsächlich für Web-Applikationen eingesetzt oder im Intra-Net</li><li>- Mehraufwand bei server- und clientseitiger Programmierung, sowie durch das momentan noch notwendige Anbieten, von Alternativseiten zu den AJAX-Inhalten</li><li>- Trennung von Struktur und Layout kann flöten gehen</li><li>- Seiten die mit AJAX verändert worden, können u.U. nicht gedruckt werden, da sie im ursprünglich vom Server geladenen Zustand gedruckt werden</li><li>- nicht barrierefrei (das setzt Bedienbarkeit ohne JavaScript voraus)</li><li>-browserabhängige Verhaltensweisen, aufgrund unterschiedlicher/gar nicht Implementierungen von bestimmten Methoden</li><li>- der direkte Zugriff auf die Zwischenablage fehlt (aus Sicherheitsgründen)</li><li>- die Kontrolle der Bedienoberflächen verbleibt beim Browser</li></ul>

### XMLHttpRequest-Objekt

- ist eine API zum Transfer von beliebigen Daten über das Protokoll HTTP

Vorgehensweise:

1. XMLHttpRequest-Objekt instanzieren (Browserspezifisch, daher Browserweiche notwendig)
2. Request senden
  - setzt auf HTTP als Transportprotokoll → HTTP-Methoden stehen zur Verfügung
  - nur Daten von gleicher Domain anforderbar
1. Kommunikation mit Server starten
  1. Öffnen der Verbindung (hier auch Festlegung ob asynchrone
    - dann ist eine Methode notwendig, die die ankommenden Daten entgegennimmtoder synchrone
    - blockiert Programmablauf bis zur AntwortKommunikation)  
Befehl: `Object.open( HTTP-Request-Methode, URL-angeforderte_Seite, Synchron?)`
  2. senden des Request's zum Server (der übergebene Parameter wird als POST gesendet)
    - bei Get wird „null“ gesendet
    - ein Query-String sonstAchtung: MIME-Typ der Anfrage im Header muss angepasst werden  
`Object.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');`
3. Verarbeiten der Serverantwort
  - Erfolgt über eine Funktion die als Event-Handler fungiert
  - die Funktion, muss dem Objekt über „`Object.onreadystatechange = Name_Der_Funktion;`“ bekannt gemacht werden (vor Senden des Request)

Die Möglichkeiten der Funktion:

- kann den Status eines Requests abfragen, also wie weit die „Abarbeitung“ des Request fortgeschritten ist
  - kann nach vollständigem Abarbeiten des Request, den Antwortstatus des Request abfragen, also den HTTP-Fehlercode des Request
  - kann bei erfolgreichem Request die Daten als Textstring aus der Antwort lesen (`Object.responseText`)
  - kann bei erfolgreichem Request die Daten als XML-Document-Object aus der Antwort lesen (`Object.responseXML`)
1. Daten überprüfen mittels Statusabfragen
  2. Daten verarbeiten als String oder XML-Dom-Object

### Portale

- Oberflächen unter der viele Dienste gesammelt sind, und zu denen der Zugang mit Single-Sign-On möglich ist
- personalisierter Zugang

## J2EE - Java 2 Platform Enterprise Edition → neue Version JEE

- J2EE ist ein Java-basierter Applikationsserver, der verteilte Applikationen unterstützt
- wie alle Applikationsserver stellt J2EE eine durchgängige Infrastruktur zur

### Komponenten-basierten

- Komponenten sind Objekte
- Komponenten sind wiederverwendbar (Source oder anderer Anwendungsfall)
- es gibt sichtbare und nicht sichtbare Komponenten
- Komponenten sind keine „stand-alone“ Programme
- Komponenten werden durch Scriptsprachen benutzt und durch Programmiersprachen erstellt
- Komponenten sind interoperabel mit anderen Systemumgebungen und in Komponenten und Komponenten-Systemen die in anderen Sprachen erstellt wurden
- Komponenten realisieren die Geschäftslogiken aus denen Applikationen aufgebaut sind
- eine Komponente ist eine „black box“ mit einer Beschreibung der Funktionalität sowie den angebotenen Schnittstellen
- Anwendungen lassen sich aus Komponenten modellieren (Ähnlich der GUI, die man aus GUI-Komponenten konstruiert)
- Komponenten sind Software-Fragmente, die man selbst veräußern könnte
- Komponenten stellen Funktionen aus einem bestimmten Anwendungsbereich zur Verfügung
- Komponenten müssen in der Regel noch durch Funktionalitäten angereichert werden, die aus anderen Komponenten kommen, um damit Anwendungen zu modellieren
- Komponenten besitzen eine Laufzeitumgebung (Container)

### Entwicklung von Anwendungssystemen zur Verfügung

- durchgängige Infrastruktur heißt, sie umfasst alle Schichten, vom Client bis zum Datenbankservice
- definierte Schnittstellen dienen dazu, dass diese Verteilten Anwendungen in einem einheitlichen Programmiermodell nach Funktionalität getrennt realisiert werden können (Model-View-Controller → BusinessObject-PresentationObject-BusinessProcessObject)
- der J2EE Server stellt das Kommunikationsglied zwischen Client und Back-End (z.B. eine DB) dar, er realisiert das Middle-Tier (Stufenarchitektur)
- Komponentenmodell: Corba abgelöst durch .Net und J2EE und man erwartet, dass Web-Services diese ablösen

J2EE tritt in mehreren Konfigurationen/Topologien in Erscheinung:

### Bean

- ist Komponente

### EB – Enterprise Bean

- praktisch nur für Anwendungen interessant
- Anwendung greift auf die Anwendungen des Java-Komponentenmodells zu
- laufen nur auf Servern

## JCA - Java Connector Architecture

- ermöglicht die Einbindung von Nicht-J2EE-Anwendungen in das Gesamtsystem
  - Beispiele:
    - ein System zu einer EPR (Endpunktreferenz) wird eingebunden
    - oder um den Zugang zu existierenden Systemen (Legacy-Anwendungen) zu ermöglichen
      - konkret: Banken verfügen zumeist über ausgereifte Systeme in Cobol, neuere J2EE-Anwendungen benutzen jetzt häufig JCA um auf diese „Artefakte“ zuzugreifen.

## EJB – Enterprise Java Bean

- enthält ausschließlich Anwendungslogik und macht keinerlei Aussagen über die Quality of Services
- beim Deployment werden die QoS-Informationen an die Komponente „angeklebt“ (diese stehen im

### DD – Deployment Descriptor

- enthält die Angaben über die Quality of Services
- „stülpt“ von außen die QoS über die Komponente (Bsp: in einem Kontext ist Komponente nur mit Security zu nutzen, im anderen ist es wurscht)
- beschreibt eine oder mehrere Beans

und es werden 2 weitere Java-Klassen generiert, das

### „Home-Interface“

- erlaubt einem Client die Implementierung der EJB zu finden
- Client stellt Anfrage nach einer bestimmten EJB an das Home-Interface
- Home-Interface sucht in der Directory nach der Entsprechenden EJB
- findet die EJB und lädt sie an, und übergibt dem Client den Zugriff auf das Remote-Interface, nicht die EJB selber

### und das „Remote-Interface“

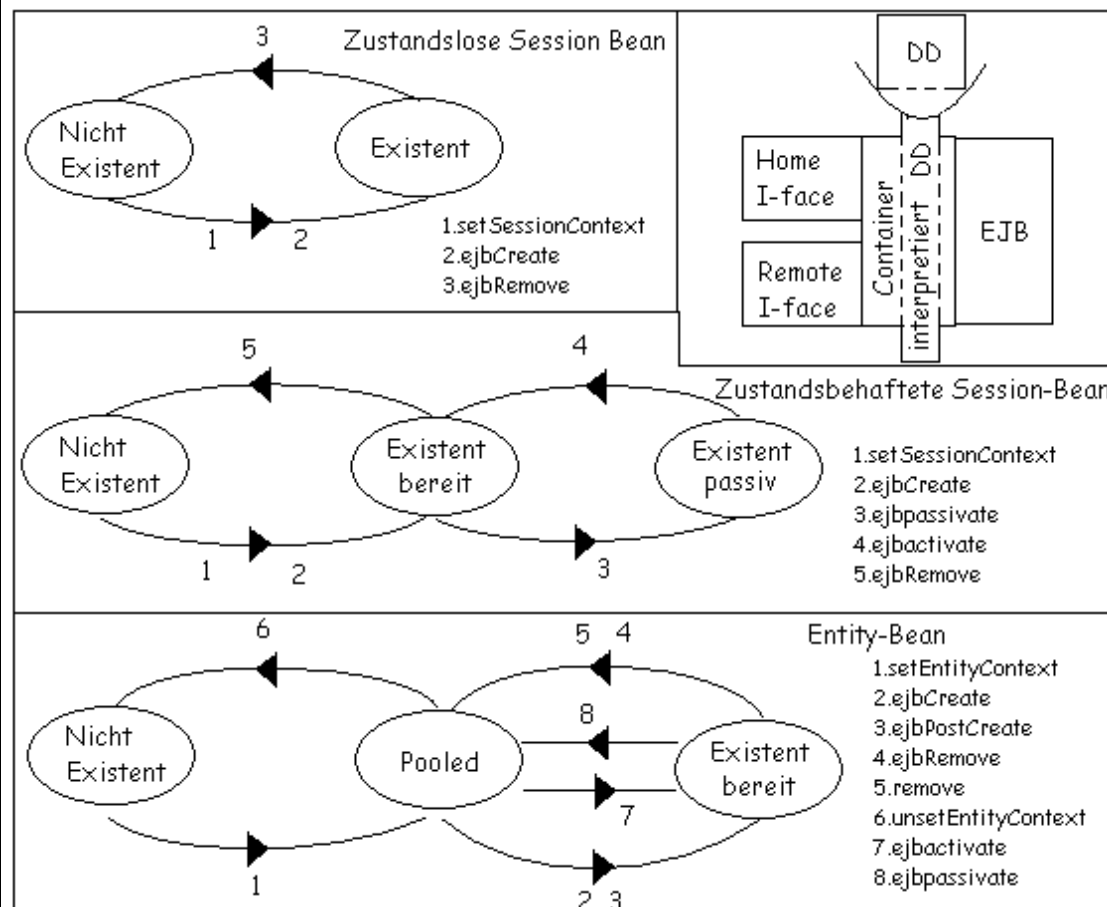
- hat das gleiche Interface wie die EJB, die gleichen Methoden (sichtbaren)
- **nur !!** beim Deployment wurde in die Methoden Code eingefügt, der mit dem Container interagiert und so hat der Container die Möglichkeit da dieser den Deployment Descriptor interpretiert, die Ausführung abzufangen bzw. unter die QoS zu stellen
- stellt sicher, das eine Komponente unter verschiedene Voraussetzungen wiederverwendet werden kann

• drei Typen von EJB's

- Session Bean
  - jede Instanz einer Bean stellt eine private Ressource eines bestimmten Clients dar
  - sie existieren genau so lange wie die Sitzung des Clients läuft, der die Bean nutzt
  - Fordert ein Client eine Bean an, wird eine Instanz durch den Server erstellt
  - wird sie nicht mehr benötigt, wird sie „weggeworfen“
  - man unterscheidet in **zustandslose**
    - speichert keine Daten zwischen den Methodenaufrufen
    - arbeiten nur mit Daten aus Parameterübergaben
  - alle zustandslosen Beans einer Sorte besitzen die gleiche Identität, es besteht keine Notwendigkeit sie zu unterscheiden
- und **zustandsbehaftete** Session-Beans
  - speichern Daten über (mehrere) Methodenaufrufe hinweg
  - aufrufe der Session-Beans können zu einer Zustandsänderung der Beans führen
  - der Zustand geht beim Beenden der Sitzung oder des Servers verloren
  - zustandsbehaftete Beans einer Sorte besitzen zur Laufzeit unterschiedliche Identitäten
  - der EJB-Container muss sie unterschieden können, da unterschiedliche Clients durch unterschiedliche Beans verwaltet werden
- Entity Bean
  - repräsentieren Dinge/Objekte des realen Lebens, welche persistente Daten besitzen (z.B. Kunde, Konto, Produkt,...)

- eine Instanz einer Bean kann durchaus von mehreren Clients genutzt werden
- implementieren keine Geschäftsprozesse
- sie modellieren Daten und dienen bspw. Session-Beans zur Datenrepräsentation
- Entity-Beans werden zur Laufzeit über einen Primärschlüssel identifiziert, den sie vom Container bekommen
- somit repräsentiert jede Bean Daten, die unter diesem Schlüssel erreichbar sind → die Bean ist an die Daten gebunden
- die Identität einer Entity-Bean ist nach außen sichtbar
- man unterscheidet in **bean-managed Persistent**
  - die Beans sind selber dafür verantwortlich, das ihre Daten persistent gemacht werden
- und **container-managed Persistent**
  - der EJB-Container macht die Daten persistent
- Message Bean
  - zum Verarbeiten von asynchronen Nachrichten
  - Sender unabhängig von Zustand/Verfügbarkeit des Empfängers
  - jede Art von Bean kann Nachrichten versenden
  - Message-Driven Beans können Nachrichten empfangen

## Transaktion bei EJB → Siehe Transaktionen



## Servlet

- Servlets sind eigentlich Java-Objekte die einzelne Verbindungen innerhalb einer Java Virtuell Maschine bedienen
- Servlets benötigen Servlet-Container
  - dieser kann entweder in einen Web-Server oder einen Applikations-Server eingebettet sein
  - Container wacht über die Lebenszyklus eines Servlets, und verwaltet die Kommunikation nach außen
  - verwaltet alle Servlets, und überzieht die Servlets mit dem Deployment Constructer-Vorschriften
  - übergibt Datenströme zur Ein- und Ausgabe an die Servlets
- benötigen eine JVM auf dem Webserver auf dem das Servlet laufen soll
- die JVM muss durchgängig laufen → sonst wäre die Performance fürchterlich

### Funktionsweise:

- Client richtet eine HTTP-Anfrage an den Web-Server
- Dieser bemerkt, dass es sich um ein Servlet handelt und gibt die Anfrage an den Container weiter
- Dieser wiederum verwaltet alle Servlets und spricht genau das Servlet an, das der Benutzer nutzen wollte
- und übergibt Datenströme zur Ein- und Ausgabe
- Servlet liest über den Eingabekanal optional Formularinhalte
- Servlet generiert über den Ausgabestrom eine HTML-Seite
- Servlet übergibt die Seite an den Container
- Container reicht die Seite an den Client weiter
- Sie sind hervorragend für die Generierung binärer Inhalte wie bspw. Bilder oder PDFs geeignet. Und sie sind als [Frontcontroller](#) für Web-Anwendungen der ideale Einstiegspunkt

### Was sind die Vorteile von Servlets/JSP gegenüber CGI-Programme?

- die Effizienz und daraus resultierende Performance
- liegt daran das die JVM im Server integriert ist, und keine externen Programme durch CGI-Skripte gestartet werden müssen
- Jede Verbindung des Servlets wird durch ein Thread-Objekt gehandhabt, hierdurch ist die Speicherverwendung optimaler, als bei Verwendung externer Programme die in einem eigenen Prozess laufen.
- Servlets können Daten teilen (es lassen sich z.B. Datenbankverbindungen oder vorberechnete Daten gemeinsam nutzen) , CGI-Programme laufen unabhängig voneinander

## JSP – Java Server Pages

- JSPs sind eine Möglichkeiten dynamischen Content für das Web (und darüber hinaus) zu generieren
- jeder JSP Container ist gleichzeitig ein Servlet Container
- der Container transformiert jede JSP in ein Servlet

### Hintergrund:

- das Erzeugen von HTML-Seiten mittels Servlets benötigt die Verwendung mehrerer out.println()-Statements
  - →
    - extrem aufwendig und nicht sehr übersichtlich
    - für Tools die zum HTML-Scripting verwandt werden (Bsp. Dreamweaver) unbrauchbar
    - Trennung zwischen Webdesign und Programmierung der dynamischen Inhaltsanforderungen ist nicht sauber möglich
- JSPs trennen Webdesign und Programmierung
- reine HTML-Seiten werden um spezifische Tags und JSP-Angaben erweitert
- diese werden nun in Servlets umgewandelt, die dann bei HTTP-Anfragen zur Ausführung kommen
- Die Umwandlung übernimmt der Container, wenn
  - die Seite das allererste mal aufgerufen wird
  - der Zeitstempel des Quellcodes neuer ist als der Zeitstempel des bei einem früheren Aufruf evtl. bereits generierten Servlet-Quellcodes ist (eine Änderung an einer JSP führt also automatisch zu einer Neu-Übersetzung. )
- nach der Umwandlung wird vom Container der gewöhnliche Java-Compiler angeworfen, um aus dem generierten Servlet-Quellcode eine Java-class-Datei zu erzeugen
- JVM lädt die Klasse, und erzeugt ein Objekt
- das Objekt wird initialisiert,... → dann kommt die Service Methode → wenn das Objekt nicht mehr gebraucht wird, kommt die Destroy-Methode (Ressourcen werden wieder frei gegeben) → der Container gibt die Methode zur Garbage Collection frei

## XSL - Extensible Stylesheet Language

- beschreibt ein Sprachfamilie zur Erzeugung von Layouts für XML-Dokumente aus den zwei Bestandteilen

- **XSL-FO** (XSL-Formatting Objects)
  - beschreibt die Anordnung der Elemente einer Seite und die Formattierung dieser Elemente, ... ähnlich CSS nur basierend auf XML
- **XSLT** (XSL-Transformations)
  - Sprache zur Transformation von XML Dokumenten in andere XML-Dokumente aber auch Textdateien oder Binärdateien erzeugt werden.

indirekt gehört auch dazu:

- **XPath** (Adressierung von Baubestandteilen)
  - XPath selber ist nur eine Notation von Pfaden, mit denen man Pfade abstrakt beschreiben kann. Diese X-Path Ausdrücke können in verschiedenen anderen Programmiersprachen verwandt werden. Beispiel: XSLT
  - Die primäre Aufgabe von XPath besteht in der Adressierung von Teilen eines XML-Dokuments.
  - XPath benutzt eine kompakte Nicht-XML-Syntax, um die Verwendung von XPath-Ausdrücken innerhalb von URIs und XML-Attributen zu erleichtern.
  - XPath operiert auf der abstrakten, logischen Struktur eines XML-Dokuments, nicht auf seiner äußerlichen Syntax. Dabei werden die Elemente zu Knoten mit Attributen und Eigenschaften
  - Funktionen:
    - // <-- fängt ein Befehl so an,... so werden alle Elemente im Dokument ausgewählt die die folgenden Eigenschaften erfüllen
    - / <-- fängt ein Befehls so an,... so wird lokal von der Wurzel aus ein Befehl ausgewählt
    - Path[**irgendeine**] <--- wähle jene Elemente in dem Pfad Path aus, die **irgendeine** Eigenschaft haben
    - Path[**@attribut**] <-- wähle die Elemente im Path aus, für die gilt sie haben ein Attribut mit dem Namen **attribut**
    - weitere Funktionen: descendent, count, \*, following, preceding, ...

## EPR – Endpunkt-Referenz

- wird z.B. von einem Verzeichnisdienst zurückgegeben wenn der Dienst ermittelt wurde
- ist ein „Pointer“ auf diesen Dienst
- besteht aus mehreren Teilen
  - z.B. eine URI (nicht unbedingt eine URL) → muss nicht web heißen,... kann auch z.B. eine URI sein die einfach nur einen Java Klassenpfad definiert, also irgendwelche Codestücke, die sich in irgendwelchen Verzeichnissen, in irgendwelchen Maschinen befinden
  - die Adresse ist verbindlich (mandatory)
  - alle sonstigen Bestandteile (Felder) sind Optional (z.B. Instanz ID)
- der Client muss informiert werden über:
  - welche Schnittstelle ist implementiert
  - welche Funktionen ruf ich auf
  - wie ist die Message/Dateninhalt serialisiert → damit der Client sie deserialisieren kann

## Cookies

- Cookies ermöglichen das clientseitige Speichern von Information, die auch vom Server stammen können und die bei weiteren Aufrufen für den Benutzer transparent an den Server übertragen werden
- erleichtern die Benutzung von Webseiten, die auf Benutzereinstellungen reagieren
- ermöglichen den Aufbau von Sessions
- Cookies werden in den Header-Teilen von HTTP-Anfragen und -Antworten übertragen
- bei jedem HTTP-Request sucht der Browser nach Cookies, die von der selben Website (und Verzeichnis) stammen, und schickt diese Cookie-Daten im Header des HTTP-Requests mit
- Ob ein Cookie angenommen (clientseitig gespeichert) wurde, muss die serverseitige Anwendung in weiteren HTTP-Requests erkennen, da vom Client keine Rückmeldung erfolgt
- Der Server kann ein Cookie durch Überschreiben mit leeren Daten löschen.
- realisieren von Sessions durch speichern einer eindeutigen Session-ID in dem Cookie um genau diesen Client bei weiteren Aufrufe wieder zu erkennen und damit nicht bei jedem Aufruf einer Unterseite das Passwort erneut eingegeben werden muss
- Bestandteile:
  - Name
  - ein Wert
  - mehrere benötigte oder optionale Attribute mit oder ohne Wert
  - einige Attribute sowie deren Einschließen in Hochkommas werden empfohlen.

## URL-Rewriting

- Beim URL-Rewriting werden nun dem Benutzer virtuelle Seiten-Adresse zugetragen.
- Dynamisch generierte Seiten haben in der URL den anzuzeigenden Datenbankinhalt als Query-String enthalten
  - → nicht jeder Datensatz besitzt eine eigene Seite
    - → Durch Rewriting wird aber genau dies simuliert → Spider indizieren die Seite, bzw. Seite kann gecacht werden
- Query enthält u.U. Primärschlüssel des Datensatzes
  - Bsp: <http://www.alewo-online.de/blog/?p=45> (hier ist Datensatz-ID direkt im Namen)
  - durch Rewriting wird dann zum Beispiel <http://www.alewo-online.de/blog/Beitrag45> daraus
- Server kann aus der Rewrited-Adress die damit verknüpfte Seite ermitteln
- Der Client bekommt von diesem Eingriff nichts mit.

## Negotiations

- Um jedem Nutzer individuell das Erscheinungsbild des Dienstes möglichst optimal darstellen zu können, soll der Dienst hinsichtlich der durch ihn angebotenen Inhalte flexibel gestaltet werden.
- Man unterscheidet verschiedene Arten von Negotiations. Es gibt vier wichtige Negotiations-Begriffe:

- **"Content-Negotiation"**

- ist ein Mechanismus der eine Abstimmung der Inhalte des aufgerufenen Dokumentes ermöglicht.  
Beispielsweise wird die Sprache der Webseite automatisch an die Einstellungen des Users angepasst, und die Seite dann in entsprechender Sprache angezeigt. Somit sind quasi "verschiedene" Seiten unter ein und derselben URL erreichbar.
- 2 Arten von "Content Negotiations" in HTTP, (können getrennt oder zusammen auftreten) "**Server**"- bzw. "**Client**"-Driven Negotiations
- **"Server-Driven Negotiation"**
  - Bei Server-Driven Negotiations trifft ein Algorithmus der sich auf dem Server befindet der eine Antwort("Response") schicken muss, die Auswahl der Darstellung die übermittelt werden soll.
  - Die Auswahl basiert auf den verfügbaren Möglichkeiten, die die Antwort bietet. Header-Informationen die der User-Agent mit übermittelt hat, und andere Request Informationen wie zum Bsp. die IP

Pro's / Wann kommen Sie zum Einsatz	Kontra's / Nachteile
<ul style="list-style-type: none"> <li>- Der Auswahlalgorithmus der optimalen Darstellung ist sehr kompliziert/aufwendig und kann deshalb dem User nicht zugemutet werden</li> <li>- Der Auswahlalgorithmus geht den User nix an</li> <li>- um weitere Anfragen vom Client (die zu Verzögerungen (Bsp. beim Seitenaufbau) führen könnten) zu unterbinden (durch hin- und hersenden weiterer requests/response)) werden bestmögliche Einstellungen ("best guess") der Representation geschickt</li> </ul>	<ul style="list-style-type: none"> <li>- bestimmte Auswahlkriterien erfordern Kenntnisse aller Details über den User-Agent und den Zweck der Verwendung der Response (z.B. Ausgabe auf Drucker/Bildschirm)</li> <li>- Wenn der User-Agent seine Einstellungen bei jedem Request mitschickt, wird das Verfahren, vorausgesetzt der Anteil an multiplen Darstellungen ist gering, sehr ineffizient.</li> <li>- dieses Verfahren verkompliziert die Implementierung für einen Origin-Server und den Algorithmus, nachdem response-Messages generiert werden.</li> <li>- es kann die Verfügbarkeit in öffentlichen Caches verringern</li> </ul>

- **"Agent-Driven Negotiation"**

- die Auswahl einer Darstellung einer Antwort("Response") trifft der User-Agent
- Entscheidung basiert auf dem initialen Response des Servers

- aus Header-Feldern dieser Response-Message kann eine Liste generiert werden
- In dieser Liste kann dann wiederum automatisiert oder auch als Benutzeranfrage beispielsweise per Auswahlliste die beste Einstellung/Auswahl getroffen werden

Pro's / Wann kommen Sie zum Einsatz	Kontra's / Nachteile
<ul style="list-style-type: none"> <li>- wenn die Auswahl von häufig eingesetzten Unterscheidungskriterien abhängt (Content-Type, Content-Language, Content-Encoding)</li> <li>- wenn der Server aus einem Request, die Fähigkeiten eines User-Agents nicht ermitteln kann</li> <li>- wenn der Verwendungszweck die Auswahl der Response beeinflusst (Ausgabe: Drucker/Bildschirm)</li> <li>- wenn der Datenaustausch über einen Proxy läuft</li> <li>- Caching kann eingesetzt werden, da zu gleichen Einstellungen gleich Response erwartet werden können</li> </ul>	<ul style="list-style-type: none"> <li>- 2 Requests nötig (erst eine Liste des Angebotes anfordern und dann die Auswahl treffen)</li> </ul>

- und **"Transparent Negotiation"**

- Proxy (normal ein Cache) trifft die Auswahl über die Repräsentation
- Er übernimmt dabei die Funktion des Servers (basierend auf dessen Agent-Driven Informationen)
- Server wird entlastet → geringere Antwortzeiten
- zweiter Request, der bei Agent-Driven Negotiation aufgetreten wär entfällt

## WS-Policy

- Angebote von nichtfunktionalen Eigenschaften oder Anforderungen von nichtfunktionalen Eigenschaften werden als **Policys** bezeichnet
- aus der angebotenen und erwarteten QoS wird mittels **Intersektion** für jede Interaktion eine „effektive Policy“ berechnet
  - die gegebenen Policys müssen in Normalform vorliegen
  - die erwartete und die angebotene Policy enthalten beide jeweils Alternativen, aus diesen Alternativen wird einfach eine ausgewählt, die in beiden Enthalten ist
  - da nur die Q-Names von WS-Policy untersucht werden, kann bei der Intersektion dies nicht unterschieden werden `<payment monthly=true>` und `<payment monthly=false>` ist also das Gleiche für WS-Policy → domainspezifische „Bean“ notwendig zum genauen untersuchen
  - bei der Intersektion werden also zuerst alle aufgrund des Q-Names möglichen Kombinationen ausgewählt und aufgelistet
  - beim Nächsten schritt der Intersection wird eine „match“-ende Variante erstellt, indem alle Assertions der „match“-enden Alternativen in eine neue Alternative gepackt werden, semantisch gesehen werden Assertions also vereinigt
  - diese Vereinigung muss nun eine domainspezifische Anwendung verarbeiten und entscheiden ob es eine gültige Policy sein kann
- legt das Verhalten die Interaktion zwischen 2 Web-Services fest
- können eine URI besitzen
- wenn sie eine URI besitzen kann auf sie verwiesen werden, sozusagen „ausgelagert“
- man kann auch innerhalb einer Datei referenzieren

Standard besteht aus:

- WS-Policy Assertions (das die Sprache zur Beschreibung von Policys)
  - sagt wie kann man nichtfunktionale Eigenschaften (keine Services) beschreiben
    - die in einer Message geschickt werden,
    - oder auch umgebungsfrei (nicht enviromental) sind also einfach nur wichtig für "Selection" und "Usage" oder "Trust"
  - es können Assertion-Types definiert werden, also eine Klasse/Typ von Assertions
- WS-Attachments

Eine spezielle Policy besteht aus:

- einer Sammlung von Policy-Alternativen  
Policy Alternative
  - alle Elemente die zum Vokabular der Alternative gehören und in einer Assertion nicht vorkommen, werden so interpretiert, als wenn sie verboten sind
  - besteht aus:
    - einer Sammlung / Liste von Policy Assertions  
besteht aus:
      - atomaren Assertions das sind Policy-relevante Funktionen die Anwendungssemantik haben und nicht weiter aufgelöst werden können

Begrifflichkeiten:

- Assertion
  - Zusicherung unter der ein Dienst zur Verfügung steht
  - Assertions sind domainspezifisch, das heißt abstrakt erklärt in einer Policy stehen zwar Assertions, jedoch sind das quasi Variablenbezeichner von einem Datentyp, mit dem die Policy nix anfangen kann, sie hält diese „Variablen“ jedoch für andere domainspezifische Anwendungen zur Verarbeitungen bereit
  - sind für WS-Policy nur anhand des Q-Names unterscheidbar  
( `<blub at=at1>` und `<blub at=at2>` hat beides den Q-Name blub und ist für WS-Policy

das selbe

`<blub_at1>` und `<blub_at2>` besitzen unterschiedliche Q-Names → für WS-Policy unterscheidbar)

- Vokabular
  - die Menge der definierten Assertion-Types
- Attachment
  - gibt die Möglichkeit QoS an Endpunkte (Webservices/Operationen/individuelle Messages/einen Dienst in einer Discovery Komponente) anzuheften (diese Endpunkte heißen Policy Subject → Policy Scope sind alle Policys)
  - bindet u.U. einen laufenden Service gegen eine Quality of Services (muss also nicht in eine bestehende WSDL Datei integriert werden, sondern wird angeheftet → Das integrieren stellt eine andere Möglichkeit dar, die aber oft komplizierter ist, dann ist kein Attachment notwendig)
  - ist ein Eigenes Element `<wsp:PolicyAttachment>`, das extern abgelegt wird, besteht aus 3 Elementen:
    - **AppliesTo**-Tag (genau einem)
      - hier steht die Resource/der Dienst für den dieses Attachment gelten soll
    - **Policy/PolicyReference**-Tag (einem oder mehr)
      - hier steht ein Verweis auf eine Policy die angeheftet wird
        - und zwar in **Policy**-Tags, wenn die Policy sich im selben Dokument befindet
        - und **PolicyReference**-Tags, wenn die Policy unter eine externen URI zu finden sind
    - **Security**-Tag (optional) (ist nicht in WS-Policy sondern in WS-Security definiert--> `<wsse:Security>`)
- Domain Expression
  - assoziiert einer Liste von Ressourcen eine Policy

Normalform:

```
<wsp:Policy ...> //hier kommt eine Policy
  <wsp:ExactlyOne> //genau eine der folgenden Alternativen (Assertions) muss gewählt werden
    <wsp:All> //alle folgenden Assertions müssen gewählt werden
      <Assertion>...</Assertion>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy ...>
```

Kompakte Form

- erlaubt das Benutzen von optionalen Assertions
- ```
...
<Assertion wsp:Optinal="xs:boolean" ...> ... </Assertion>
...
```
- Kompaktform lässt sich in Normalform umwandeln

## UDDI – Universal Description Discovery and Integration

- beinhaltet Informationen über:
    - den Anbieter
    - die Funktionen die dieser Anbieter anbietet
    - die Technizitäten , wie mache ich das , → bindings zum Beispiel
  - man unterscheidet in UDDI drei verschiedene Arten von Diensten, die nach ihrer Typ entweder den
    - **White Pages**
      - sind die „normalen“ Telefonbuchseiten mit Namensregister und angebotenen Dienst mit Kontaktinformationen
    - **Yellow Pages**
      - entspricht dem Branchenbuch, den gelben Seiten, also was mach ich eigentlich,...
      - enthält auch geographische Taxonomien, zum suchen lokaler Services
    - oder **Green Pages**
      - enthalten Detailinformation,... (Gerichtsstand, Art des Unternehmens etc.)
      - wie macht man Service mit mir, also ein Binding Template ist auch enthalten
- zugeordnet werden
- der Standard garantiert ein 24-stündigen Datenabgleich
  - Anmeldung an einem Beliebigen Knoten, also einer öffentlichen Implementierung einer UDDI, und 24 Stunden später kennen mich alle UDDI-Knoten
  - Public UDDI ist sinnlos weil keiner weiß was von den dort stehenden Services echt ist
  - Interne UDDI wird als function-Repository „mißbraucht“
  - **Vertikal UDDI**
    - nicht öffentlich , aber im Internet
    - Bezahlverzeichnisse
    - sehr erfolgreich

UDDI implementiert einige SOAP-Schnittstellen und ist aufgrund von Web-Services zugreifbar Verfügbar machen:

- abstractes Definieren der Dienste die man anbieten will, in WSDL (funktionale und auch nichtfunktionale Eigenschaften)

## Metadata Exchange – Metadaten austausch

- Wird benutzt wenn der Anbieter eines Services bekannt ist
- Endpunkt bekannt
- frage Endpunkt über Metadaten ab
  - gib mir deine Policy
  - deine WSDL
  - dein Schema etc.
- jeder Endpunkt muß Metadata Exchange unterstützen
- Abfrage erfolgt über SOAP mit <wsx:getMetadata> steht im Soap-Body
- Metadaten können auch in beliebig einschränkbarem Volumen abgefragt werden
- benutzt man zum „Bootstrapping“ (der einfachere Dienst führt zum Starten des Komplexeren)

## MIME - Multipurpose Internet Mail Extensions

- MIME ist eine Hinweis der mittels Mitteilung vom Server mit übertragen wird
- der MIME-Typ gibt an, von welchem Typ die Daten die geschickt werden sind, ... also jpg, oder text etc.
- MIME-Typen müssen vom Server ermittelt werden, dies geschieht normalerweise über die Dateiendung, kann aber auch über die Analyse einiger Bytes des Inhaltes passieren
- MIME-Typen bestehen aus 2 Hälften, erstens der **Kategorie** und zweitens dem **Datentyp**
  - **Kategorien**
    - text ; image; audio; video; application; multipart...
  - **Datentypen**
    - jpg; gif; html; plain; x-wav; quicktime; pdf ...
- Beispiel für einen MIME-Typen:
  - text/html → Standardtyp für Webseiten, Servlets setzten dieses z.B.
  - text/plain → reine Textdatei, die nicht vom Browser interpretiert wird, ist dieser MIME-Tag gesetzt, interpretiert der Browser selbst HTML Code nicht, sondern gibt ihn als Klartext aus
  - image/jpeg oder image/gif → Das übertragene Objekt ist ein \*.jpg bzw. \*.gif Bild
  - audio/basic oder audio/x-wav
  - video/mpeg oder video/quicktime
  - application/pdf oder application/msword oder application/vnd.ms-excel
  - multipart/mixed oder multipart/news → der Datenstrom beinhaltet unterschiedliche Teile (kommt oft bei e-mail, die einen Teil Text, und einen Teil Datenanhang besitzen vor)

## Caching

- bezeichnet das Speichern von Kopien von Webseiten bzw. Repräsentationen die durch eingehende Requests aufgerufen werden
- man unterscheidet den **Web Cache** in den **Browser Cache**, den **Proxy Cache**, und den **Gateway Cache**
  - **Web Cache**
    - Liegt zwischen dem Client und dem Origin Server hierbei existiert nicht ein Cache sondern jeder Rechner über den ein Hop geroutet wird, besitzt einen Cache
    - Zweck:
      - dient der Reduktion von Netzwerktraffik
      - dient der Skalierbarkeit des Webs
      - steigert die Performance für die Clients
  - **Browser Cache**
    - der Browser speichert dir Repräsentationen von besuchten Seiten lokal auf dem Rechner
    - so muss er nicht jedes mal ins Internet gehen, wenn eine Request aus dem Cache befriedigt werden kann
    - im Allgemeinen überprüft der Browser einmal pro Sitzung die Schalheit der Repräsentationen im Cache, so ist sichergestellt, dass im Cache immer nur fresh Sources verfügbar sind
    - der Browser holt die Ressourcen aus dem Cache wenn man den Back-Button betätigt, oder auf einen Link klickt, den man davor schon einmal benutzt hat
  - **Proxy Cache**
    - Proxy organisieren den Traffic von mehreren Clients (das können wenige aber auch einige Tausend sein),
    - jeder Request eines Clients führt dazu dass der Request versucht wird aus dem Cache zu befriedigen → geht dies nicht, wird der Request weitergeleitet, und die Seite nach dem Response lokal gecacht
    - Internet Service Provider (ISP) betreiben sie als intermediaries (standalone devices), durch die ein Request zuerst befriedigt werden soll, sie besitzen sogenannte „Cache-Farmen“,...
    - man kann das Routing und somit den Zugriff auf bestimmte Caches manuell manipulieren, indem man in den Browsereinstellungen manuell eingibt, welchen Proxy er nutzen soll
    - ansonsten kommen sogenannte Interception Proxies zum Einsatz, hierbei leitet das Zugrunde liegende Netzwerk den Request automatisch zu ihm um, und versucht den Request so zu befriedigen
    - Proxy Caches sind „shared cache“ da eine Vielzahl von Usern auf ihn zugreifen können
    - werden oft auch „On-The-Edge-of-the-Enterprise“ eingesetzt, um Kosten zu sparen, und den Zugriff zu beschränken („Proxy mit Firewall“), diese Caches laufen auf Edge-Servern, das sind Gateways zwischen Inter und Intranet
  - **Gateway Caches**
    - sind auch Intermediaries, jedoch unterliegen sie nicht der Gewalt eines Netzwerkadministrators, sondern den Betreibern von Webseiten, sie halten sich Gateway-Server vor mit dessen Cache die Geschwindigkeit bzw. gefühlte Performance verbessert werden sollen → bessere Skalierbarkeit
    - es gibt einige Wege Request zum Gateway Cache zu leiten, der einfachste ist, man benutzt den Gateway als Origin Server
    - Content Delivery Networks wie Akamai vermieten ihre Cachefarmen an Interessenten und verdienen so mit Caching ihr Geld

### Funktionsweise:

- alle Caches haben eine bestimmte Menge an Regeln die sie befolgen müssen, diese Regeln sind natürlich nicht vorgeschrieben, jedoch gibt es eine Hand voll Regeln, die von eigentlich allen Caches befolgt werden.
  - Regeln:
    1. Repräsentationen werden nicht gecashet wenn:
      1. der Response-Header sagt, die Repräsentation soll nicht gecashet werden
      2. wenn die Nachricht keinen Validator enthält, so wird angenommen die Nachricht ist nicht cachable
      3. wenn der Request „Authentifizierung“ verlangt oder authentifiziert ist, so wird nicht gecashed
      4. wenn der Request „secure“ ist, wird nicht gecashed

5. wenn eine Methode die Seiteneffekte besitzt aufgerufen wird, wird die Repräsentation nicht gecashed, und ist sie bereits gecashed wird sie schal bzw. muss neu validiert
2. Repräsentationen werden gecashet wenn:
  1. Sie frisch sind, in diesem Fall sogar ohne den origin Server zu kontaktieren
3. Sonstige Regeln:
  1. gecashete Repräsentationen werden „frisch“ gehalten, oder nur zurückgeliefert wenn sie frisch sind
  2. wenn eine Repräsentation schal ist, wird der Origin Server kontaktiert, um das Document zu validieren oder upzudaten (hierbei kann es vorkommen, dass auf dem Weg zu Origin-Server eine aktuelle Version gefunden wird, dann wird der Cash der den Request befriedigen kann, dies auch tun)

- ist das Validieren einer Repräsentation notwendig, wird oft nur der Header angefordert, um dann mit dem vorliegenden verglichen zu werden,... unterscheiden sich die Header nicht, wird angenommen die gespeicherte Kopie ist noch frisch, ansonsten das Gegenteil

- der Response header ermöglicht folgende Elemente als Cache-Control

- max-age → eine Dauer für die die Repräsentation als frisch angenommen werden kann
- s-max-age → ist das gleiche wie max-age, ist jedoch nur für shared caches betreffend
- public → authentifizierte Response werden normal nicht gecashet, mit public kann man angeben, dass sie dennoch gecashet werden dürfen
- no-cache → sorgt dafür, dass die Caches, bevor sie die gecashete Kopie zurückgeben, das Dokument beim Origin-Server validieren lassen
- no-store → weist die Caches an, unter keinen Umständen eine Kopie des Content zu machen, „cachen verboten“
- must-revalidate → weist den Cache an, sich strikt an deine Regeln zu halten, und alle Information die ihm gegeben sind zu beachten
- proxy-revalidate → wie must validate, nur das diese Anweisung nur für Proxy-Caches gilt

„frisch“ - (fresh) bedeutet:

- eine Repräsentation besitzt ein Ablaufdatum oder besser ein Verfallsdatum, ist das aktuelle Datum vor dem Verfallsdatum, so ist eine Repräsentation „fresh“
- eine Repräsentation wurde erst kürzlich von einem Proxy gesehen, und die letzte Veränderung am Inhalt liegt sehr weit zurück
- eine Repräsentation liegt im Browsercache, und die Überprüfungsdauer ist auf einmal pro Session eingestellt

„schal“ - (shale) bedeutet:

- eine Repräsentation ist nicht frisch

### Typischer HTTP-response-Header

```
HTTP/1.1 200 OK
Date: Fri, 30 Oct 1998 13:19:41 GMT
Server: Apache/1.3.3 (Unix)
Cache-Control: max-age=3600, must-revalidate          <-- Cache-Steuerung
Expires: Fri, 30 Oct 1998 14:19:41 GMT              <-- Wichtig! Verfallsdatum
Last-Modified: Mon, 29 Jun 1998 02:28:12 GMT        <-- für Validierung!!
ETag: "3e86-410-3596fbbc"                          <--"Checksumme" für Validierung
Content-Length: 1040
Content-Type: text/html
```

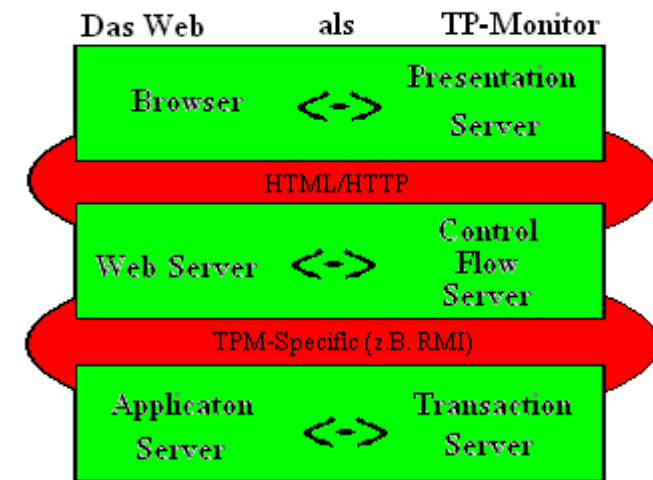
- **Portale**
- Oberflächen unter der viele Dienste gesammelt sind, und zu denen der Zugang mit Single-Sign-On möglich ist
- personalisierter Zugang

**inifizierend** – ist ein Service genau dann, wenn er die Middleware zwingt sich nach seinen Vorstellungen zu verhalten (Beispiel: Transaktionskontext)

**TP-Monitor** – Transaction Processing Monitor (Überwachungsprogramme für die Transaktionsverarbeitung → wird ACID eingehalten)

- Sorgen für Skalierbarkeit; Effizienz, Datensicherheit, und Geschwindigkeit bei Abwicklung vieler Useranfragen
- TP-Monitor ist ein Programm, das in einem Computer-Netz den synchronen (transaktionalen) Datenaustausch zwischen Clients und Servern steuert
- dienen dem Online-Datenaustausch in Echtzeit, können aber auch zur Stapelverarbeitung genutzt werden
- managed die vorhandenen Ressourcen,
- Vergibt Prioritäten für Abrufe und Aufträge,
- überwacht Transaktionen
- schützt Daten vor unberechtigten Zugriffen
- stehen zwischen Front end und Back end
- steuern das Schreiben und Lesen transaktionaler Daten
- sorgen für Datensicherheit und Datenintegrität
- Einsatz erfolgt in Netzen mit hohem Transaktionsaufkommen
- Einsatz bei verschiedenen Anwendungen die Grundfunktionen wie Sicherheit und Verzeichniszugriff gemeinsam benötigen
- in TP-Monitoren können weitere Anwendungen hinzugefügt werden, ohne die Grundfunktionen zu verändern
- besteht aus 3 Schichten
  - Präsentation Server
    - Eingabeannahme
    - Validierung der Eingaben
    - Authentifizierung
    - Aufbau der Anfragenmessage
  - Control Flow Server → Erfüllt die Funktion eines Routing-Layers
    - Weiterleitung des Requests
      - Decodieren der Message
      - Bestimmen des Programmes
      - Aufruf des Programmes
    - überwachen laufender Transaktion, und eventuell erforderliches Abbrechen
    - überwachen der Request-Message-Integrität
    - Exception Handling
  - Transaction Server
    - Ausführung der Logik
    - Manipulation der Daten

→ die Parallelen zur Web-Architektur springen ins Auge



# Architektur von Anwendungssystemen

## Funktionale Aspekte der Architektur

- Komponenten - modulare Einheit von Funktionalität die durch ein oder mehrere Schnittstellen erreichbar ist
- Subsysteme - an Gruppe von Komponenten in einem IT-System
- Collaboration - Use Cases die einen Anwendungsfall zwischen Komponenten beschreiben
  - besteht aus einer Abfolge von Operationen von Komponenten
- Interaction - Austausch zwischen 2 Komponenten

## Operationale Aspekte

- Node - eine Plattform auf der Software ausgeführt wird
- Connection - physikalische Daten-Verbindung zwischen Nodes (LAN, WAN...)
- Deployment Unit - eine oder mehrere Komponenten die zusammen auf einem Node platziert werden, wobei die Ausführung, der Zustand, und die Intallationsaspekte unterschiedliche Platzierung besitzen können

## Repräsentation

UML für die funktionalen Aspekte

- **Statische** Beziehungsdigramme
  - Klassendiagramm, Beziehungsdigramme
  - Entwicklungseinheiten/-abschnitte
- **Dynamische** Beziehungsdigramme
  - (zeitorientierte) Sequenzdiagramme
  - (message orientierte) Kollaborationsdiagramme
  - Zustandsdiagramme

ansichtsabhängige Stile für operationale Aspekte

- **statische** Ansichten
  - Konzeptionelle, Spezifische und Physikalische
- **dynamische** Ansichten
  - Walkthroughs und Zeitliche (Temporale) Aspekte

## Modellierung

- ein Modell ist eine abstrakte Vorstellung eines existent oder nonexistenten Gegenstandes
- oder ein (real existierender) Prototyp / Vorstellung die Beispielhaft eine Gruppe von Gegenständen (anhand von gewissen Eigenschaften) darstellt/klassifiziert

Eigenschaften von Modellen:

- abstrakt, verständlich, korrekt, Vorbildhaft, vergleichsweise billig

Einsatzgebiete von Modelle in der Software

- Entwicklung von Informations- & Anwendungssystemen
- Beschreibung von Informations- & Anwendungssystemen
- Umsetzung/Weiterentwicklung/Auslagerung von Informations- & Anwendungssystemen
- Reengineering
- Beschreibung der Aufbau- bzw. Ablauforganisation

Software-System-Modelle:

- für die Anforderungen: Use-Cases
- für die Struktur: Klassendiagramme
- Verhaltensmodelle (Behavioural models)
- Process/Workflow Modelle (Prozess- /Arbeitsflußmodelle)
- Entwicklungsmodelle (Pattern) ...

## ERM - Entity Relationship Model

- modelliert eine Beziehungsgeflecht mithilfe von **Entitäten und Beziehungen (Realtionships)**
- Entitäten (Entitys) sind reale Objekte mit Eigenschaften (Attributen)
  - (zum Beispiel: „Mensch“ ist eine Entität, und besitzt das Attribut „Name“)
- eine Beziehung (relationship) ist eine Assoziation zwischen verschiedenen Entitäten

verschiedene Darstellungen:

- Notation 1
  - Entitäten Typen (Entity Sets / „Klassen“) → Rechtecke
  - Beziehungstypen (Relationship Sets) → Rauten („Diamonds“)
  - Attribute → Ellipsen
  - identifizierendes Attribut – Primärschlüssel (Primary Key) → Unterstrichen
  - mehrwertige Attribute (multi-valued) → doppelte Ellipse
  - abgeleitete Attribute (geerbte Attribute/derived) → gestrichelte Ellipse
  - Rollen (roles) → stehen als Bezeichnung über den Pfeilen der Relations
  - Kardinalitäten: eine gerichtete Kante steht für den Wert „eins“ eine ungerichtet Kante für den Wert viele
    - Entität\_1 ausgehende gerichtete Kante Relationship Kante Entität\_2 → Entität\_2 steht in Beziehung zu genau einem Element von Entität\_1
    - Totale Partizipation (jedes Element dieses Entitytyps erfüllt genau diese Relation) → als doppelte Linie
    - Teilweises Partizipation (nicht alle Elemente des Entitytyps besitzen zwingend diese Relation) → einfache Linie
  - Weak Entitys (schwache Entitäten /sind nur identifizierbar durch ihren „Besitzer“ (eine andere Entity)) → doppelte Umrandung
  - Spezialisierungen → Dreiecke, wobei die Vater-Entität die Breitseite bekommt, und das Dreieck in „Pfeil“-richtung wegzeigt, ... die Kind-Entitäten stehen entsprechend in „Pfeil“-Richtung
  - Generalisierung → entweder als Umkehrung der Spezialisierung dann siehe oben
    - oder andere Notationen:
      - Wie im Klassendiagramm mit Vererbungspfeile, die für disjoint Generalization in einem Klassifizierenden Attribut gesammelt werden, und dann mit der Vater-Entität Connected
      - oder als Splitting,/Aufteilung/Partial, dann durch einen Kreis gekennzeichnet, der auf dem EntitätsPfeil liegt
      - überlappende Generalisierung wird dargestellt als, Vererbungspfeile mit Querbogen und kleinem Kreis
    - Anhäufungen/Aggregationen (wirkt wie Zoom,... ermöglicht den Bruch des bipartiten Graphen) → man rahmt sie einfach ein
- Notation 2
  - Kardinalität:
    - eine Pfeilspitze → genau 1
    - eine Pfeilspitze mit Kreis → 1 oder keins
    - zwei Pfeilspitzen → viele, mindestens 1
    - zwei Pfeilspitzen mit Kreis → beliebig (viele oder auch nix)
- Notation 3 (min-max - Notation)
  - Kardinalität:
    - als Zahlen über den Relationskanten, jedoch hier die Seiten Vertauscht (was in Notation 1 links steht steht in dieser Rechts über der Kante , u. umgekehrt), die Anzahl steht also direkt an der zugehörigen Entität

## Datenbanken

- ursprünglich Filesystem,... → Mehrbenutzerzugriff → Inkonsistenz
- verschiedene Abstraktionslevel:
  - Physical Level
    - beschreibt, wie ein Datensatz gespeichert wird, und wie man auf sie zugreift (Indexe, Pfade)
  - Logical Level
    - beschreibt den Aufbau eines Datensatzes, also die Elemente des selbigen und die Beziehung zwischen den Elementen
  - View Level
    - beschreibt die Darstellung der Daten durch verschiedene Programme, hierbei können verschiedene Programme verschiedene Darstellungen erlauben, gespeicherte Elemente „hide“n oder auch nicht

## Schema

- beschreibt die logische Struktur der Datenbank, der das Design der entsprechenden Levels folgt

## Instance

- aktueller Zustand der Datenbank zu einem bestimmten Zeitpunkt
- Physikalische Datenunabhängigkeit (ich kann das Physikalische Schema ändern ohne das logische ändern zu müssen)
- Logische Datenunabhängigkeit (ich kann das Logische Schema der Datenbank ändern ohne die Anwendung ändern zu müssen)

## DDL – Data Definition Language

- Sprache zum definieren eines Datenbankschemas

## DSL – Data storage language

- spezifiziert die Speicherstruktur und die Zugriffsmethoden die von einem Datenbanksystem benutzt werden
- erweitert normalerweise die DDL

## DML – Data Manipulation Language

- ist eine Sprache um auf Daten in einem Model zugreifen zu können, und diese manipulieren zu können
- es gibt **prozedurale** (what? How?) und **nicht prozedurale/declarativ** (I'm only know what :) Sprachen

## SQL – Structured Query Language

- ist eine weit verbreitete deklarative Sprache für DDL und DML
- Zugriff ist Möglich, über in der Programmiersprache verfasste Aufrufe auf eine Datenbank, oder über SQL-Querys, aus Anwendungen heraus, über definierte Schnittstellen (ODBC/JDBC) → ermöglicht das Verschiedene Nutzergruppen verschiedenen Zugriff haben können
  - mögliche Nutzergruppen:
    - Application Programmer → arbeiten mit dem System über DML-calls
    - Sophisticated Users (Anspruchsvolle Nutzer) → Stellen Anfragen in einer Datenbank-Sprache
    - Specialized users → schreiben Datenbank Anwendungen die nicht in das herkömmliche Zugriffsmodell passen
    - Naive users → benutzen ein bestehendes Datenbankprogramm

## Transaktionsmanagement:

- **eine Transaktion ist eine Sammlung von Operationen die zusammen eine einzelne logische Funktion in einer Datenbank Anwendung ausführen**
- das Management stellt sicher das sich die Daten immer in einem konsistenten (nicht

widersprüchlichen/fehlerhaften) Zustand befinden trotz eventuell eintretender fehlerverursachender Fälle

- überwacht die Gleichzeitigkeit von Transaktionen, bzw. das diese nicht gleichzeitig also nebenzeitig ablaufen, und so nicht die Konsistenz der Daten gefährden
- Datenbankeinträge können über Keys identifiziert werden, ... hierfür ist es nötig, dass die Eigenschaft die sie widerspiegeln, für jedes Element der Datenbank eindeutig ist.
  - **superkey** – ein Superkey ist ein beliebiges Tupel/Relation aus identifizierenden Werten
  - **candidate key** – ist eine einzelnes/minimales identifizierendes Tupel
  - **primary key** – ist ein als allgemeiner Identifikator gewählter candidate key

## typische Query Struktur

- select A\_1 , A\_2, ... A\_n (repräsentieren Attribute)  
from r\_1, r\_2, ... , r\_m (repräsentieren Relationen)  
where P (ist ein Prädikat)
- das Resultat des Querys ist eine Relation von Daten aus der Datenbank

## Unterstützung von Datenbanken in Anwendungen:

### JDBC – Java Database Connectivity

- ist die Java-API um SQL unterstützt auf Datenbanken zuzugreifen
- unterstützt zahlreiche Methoden um Querys zu stellen oder Daten upzudaten, und die entsprechenden Ergebnisse zu empfangen
- unterstützt auch die Abfrage von Metadaten, sowie Datentyp und Relationstypen abfragen

### Verfahrensweise:

- Öffnen einer Verbindung
- Erstellen eines „Statement“- Objektes
- Ausführen der anstehenden Querys über das „Statement“-Objekt, welches die Queries an die DB sendet, und auch die Ergebnisse abfängt
- Es gibt Exception-Mechanismen um Fehler abzufangen

### SQL Stored Procedures (StP)

- Applikationen rufen Datenbankelemente ab, indem sie SQL Call's nutzen
- diese Funktionen werden bei Bedarf vom DBMS geladen und mit den Calls aufgerufen
- diese Funktionen können auch extern liegen und in einer anderen Sprache verfasst sein,... nur die Quelle muss halt bekannt sein

## Transaktionen

- eine feste Folge von Operationen die zusammen eine logische Einheit bilden

die Abwicklung von Transaktionen muss folgenden wichtigen (ätzenden) Eigenschaften genügen

### (ACID):

- **ATOMICITY** – eine Transaktion wird entweder ganz oder gar nicht ausgeführt, sie ist also nicht teilbar
- **CONSISTENCY** – die Ausführung einer Transaktion bewahrt die Konsistenz der Daten
- **ISOLATION** – Bei gleichzeitiger Ausführung mehrerer Transaktionen dürfen sich diese nicht gegenseitig beeinflussen
- **DURABILITY** – die Dauerhaftigkeit der Transaktionen muss gewährleistet sein, das heißt sie verblasen nicht, die Daten sind persistent, auch wenn System-Fehler auftreten

Transaktionen können **Interleaved (verzahnt)**, **Simultaneous (zeitgleich)** oder **Parallel (zur gleichen Zeit, aber nicht zeitgleich[quasi verzahnt und simultan])** verlaufen

Transaktionen werden beendet entweder durch

- commit
  - gibt an das die Transaktion erfolgreich durchgeführt werden konnte
  - die Daten können in der DB persistent gemacht werden
- abort
  - die Transaktion wurde abgebrochen
  - der Zustand vor beginn der Transaktion muss wieder hergestellt werden (**rollback**)
  - hierbei kann es vorkommen, dass das Rücksetzen das Rücksetzen anderer Transaktionen verursacht (**kaskadierender rollback**)

**Locking** – bezeichnet das Sperren von Daten zur Gewährleistung der ACID Eigenschaften

- Optimistisches Locking
  - Beim Schreiben wird überprüft, ob Daten auf denen die Transaktion basiert, verändert wurden
- Pessimistisches Locking
  - zu Beginn einer Transaktion, werden alle Werte die während der Transaktion irgendwann benötigt werden gesperrt

**Blockierung** – wenn eine Transaktion auf die Ausführung einer anderen Transaktion warten muss, so ist diese Transaktion blockiert

- shared lock
  - eine Transaktion sperrt den Schreibzugriff auf die von ihm benötigten Daten, so dass andere Transaktionen die Daten lesen dürfen, aber nicht verändern
- exclusive lock
  - die Transaktion sperrt den Zugriff auf die Daten komplett, sodass andere Transaktionen weder lesen noch schreiben können auf den Daten

**Deadlock** – wenn sich zwei (oder mehr) Transaktionen gegenseitig blockieren, so spricht man von einem Deadlock

Darstellung:

- eine sehr anschauliche Darstellung von Transaktionen erfolgt mit Hilfe von Grey-Reuter-Diagrammen

Aufbau:

```
begin of transaction    <-- BOT
  read x
  write y
end of transaction     <-- EOT
```

## Transaction Processing (TP)

- Welche Middleware unterstützt Transaction Processing?
  - Application Server
  - TP-Monitor
- Ziele:
  - Maximieren von
    - Durchsatz, Auslastung, Verfügbarkeit, Skalierbarkeit
  - Minimieren von
    - Antwortzeiten
- Statuus die Transaktionen besitzen können
  - Aktiv (activ), Erledigt (Done), Failed (Fehlgeschlagen), Aborted (abgebrochen) , Commit (erfolgreich abgeschlossen)

Allgemeine Probleme bei Transaktionen:

- non recoverable (nichtrücksetzbare Transaktionen)
- dirty read (lesen unbestätigter Daten / never trust uncommitted Data)
- dirty write (schreiben von unbestätigten Daten)
- lost update (verloren gegangene Änderungen)
- View <-/-> Daten (inkonsistente Sicht aber konsistente Daten)

Serialisierung:

- serialisierbar bedeutet, die Ausführung der Transaktionen in ihrem Schedule (Ausführungs-/Zeitplan) hat das gleiche Ergebnis, als wenn die Transaktionen seriell ausgeführt würden
- swap → Zwei aufeinander folgende Anweisungen von verschiedenen Transaktionen können im Schedule gewaped (vertauscht) werden, wenn sie nicht im Konflikt stehen, ... danach stehen sie auch nicht in Konflikt
- seriell
  - die Transaktionen werden nacheinander Abgearbeitet → ACID ist sichergestellt → miese Performance
- nebenläufig
  - die Transaktionen werden nebeneinander ausgeführt → ACID muss auf komplizierte Weise sichergestellt werden , ist jedoch schneller

Precedence Graph (Vorrangs/ Präzedenzgraph)

- Transaktionen sind nicht serialisierbar wenn dieser Zyklen enthält
- Idee:
  - stelle die Reihenfolge der Konfliktoperationen (read, write) durch Kanten in einem gerichteten Graphen dargestellt
- Konstruktion:
  - jede Transaktion ein Knoten
  - für jeden **Konflikt** wird eine Kante zwischen den Knoten gezeichnet, hierbei folgt die Richtung der Reihenfolge der Ausführung,... also von der früheren Transaktion zur späteren
  - da Konfliktoperationen nicht vertauscht werden dürfen, darf der Graph keinen Kreis enthalten, da wir dann nicht mehr sicherstellen können, dass alle Operationen des einen Graphen vor den des anderen ausgeführt werden , somit ist die Definition der Serialisierbarkeit verletzt, → nicht serialisierbar
  - sind die Transaktionen nicht serialisierbar, so wird ein rollback eingeleitet

## verteilte Transaktionen → 2-Phasen-Commit

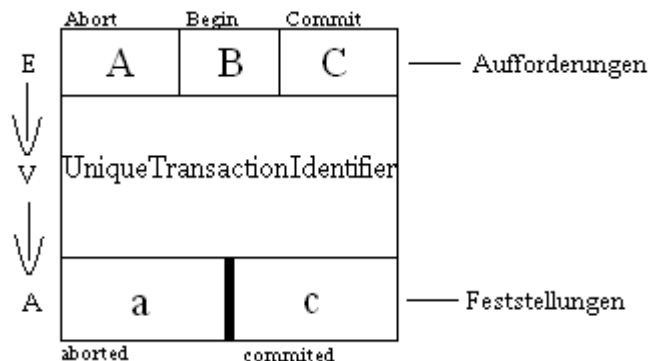
- geregelt über das ACP (Atomic Commitment Protocol) welches ACID sicherstellen will

### BOF

- 1.Phase  
Request: Prepare (führe diese Transaktion aus, und gib deinen Status zurück)  
Response: I'm Prepared (wenn Status == Done)
- 2.Phase  
Request: Commit (mache Commit)  
Response: OK (Commit erfolgreich durchgeführt)
- self.commit

### EOT

## Grey-Reuter-Diagramm



### Transaktionen bei EJB:

Bei verteilten Anwendungen entstehen oft Fehlersituationen durch den gleichzeitigen Zugriff mehrerer Benutzer auf gemeinsame Daten. Um diese Fehlersituationen zu behandeln, werden Entwickler von Transaktionen unterstützt. Die auszuführenden Aktionen werden vom Entwickler in Transaktionen unterteilt. Der EJB-Container muss die Sorge dafür tragen, dass die einzelnen Aktionen einer Transaktion erfolgreich durchgeführt werden müssen. Falls ein Fehler auftritt, müssen alle bisher einzelnen erfolgreich durchgeführten Aktionen wieder rückgängig gemacht werden. Die Unterstützung von Transaktionen ist ein wesentlicher Bestandteil der EJB-Spezifikation.

## UML-Sequenzdiagramm

- „temporal“ orientiert
- zeigt graphisch die Reihenfolge von Nachrichten
- zeigt nicht, wie man das Empfängerobjekt enthält

## UML-Kollaborationsdiagramm

- strukturell orientiert
- zeigt den Kontextaspekt also die Beziehungen zwischen Objekten
- die Reihenfolge ist nur an einer Nummerierung erkennbar
- die Zeit ist nicht ablesbar

## UML-Zustandsdiagramm

- beschreibt den Lebenszyklus einer Instanz einer Klasse
- oder den Ablauf einer Operation auf einer Instanz einer Klasse

## UML-Komponentendiagramm

- beschreibt die SW-Komponenten und ihre Abhängigkeiten
- Pakete können Komponenten beinhalten und andersherum

## Meta-Modellierung

- es gibt eine 4-Level Architektur der Modellierung
  - Level 1 – Daten

- Objekte, Prozess Instanzen, Daten
- Level 2 – Modelle
  - Schemata, Typen, Klassen, Prozess-Modelle
- Level 3 – Meta Modelle
  - die Elemente des Modells werden beschrieben durch dieses Modell
- Level 4 – Meta – metamodelle
  - beschreibt in einem Modell, den Aufbau von Modellen allgemein

Bsp: Max Lustig-Mustermann → Personen → Entität → ER-Modelle sind normal in Deutsch beschrieben :P

## XML – Extensible Markup Language

- ist ein Metamodell für Datenmodellierung
- erlaubt es Instanzen von Typen, und ihre Typen getrennt zu erstellen
  - DTD („Document Type Definition“) ist der Typ, und die Datei, die nach dem dort definierten Schema aufgebaut ist, ist die Instanz
  - XSD („XML Schema Definition“)

### Aufbau:

- Die Teile von XML-Dokumenten sind über Tag's definiert
- Es werden über die Tags nur Dateninformationen gespeichert, es wird keinerlei Aussage über die Repräsentation für den Nutzer getroffen, also keine Layout-Aussagen
- XML-Dokumente müssen wohlgeformt sein
  - 1. Tag ist `<?xml version="1.0" ?>`
  - Es existiert genau ein Wurzelement
  - die nested Elemente überlappen nicht

## DTD - Document Type Definition

- ist eine Deklaration für XML-Dokumenten, die die Struktur eines solchen Dokuments festlegt, also die Reihenfolge und die Verschachtelung der Elemente und die Art des Inhalts von Attributen
- in einer DTD werden
  - Dokumenttyp in der Dokumenttyp-Deklaration definiert:
    - `<!DOCTYPE Wurzelement PUBLIC "Public Identifier" "datei.dtd" [ ... ]>`
      - Public Identifier → bekannte Beschreibungen die öffentlich zugänglich sind, zum Beispiel für XHTML
    - Markup-Deklarationen
      - Elemente, Attribute von Elementen und Entitäten sowie Notationen können definiert werden

### Eigenschaften:

- werden anders als XML-Schema nicht in XML definiert

## XSD – XML-Schema Definition

- ist eine Deklaration für XML-Dokumenten, die die Struktur eines solchen Dokuments festlegt, also die Reihenfolge und die Verschachtelung der Elemente und die Art des Inhalts von Attributen
- Anders als bei den klassischen DTD's wird die Struktur in Form eines XML-Dokuments beschrieben. Darüber hinaus wird eine große Anzahl von Datentypen unterstützt.
- Ein konkretes XML-Schema wird als XSD bezeichnet und hat üblicherweise die Endung ".xsd"

### Datentypen:

- besitzt einige „vordefinierte“ rudimentäre Datentypen, aber ermöglicht auch die Erstellung komplexer eigener Datentypen

## DNS – Domain Name System

- wird zur Umsetzung von Domainnamen in IP-Adressen (forward lookup) benutzt
- Domains auf dem obersten Level, werden Top-Level-Domains genannt
- werden von rechts nach links gelesen
- die Trennung von DNS-Name und IP-Adresse, erlaubt es die IP zu ändern ohne das der Name sich ändert
- Seiten können von einem Server auf den anderen gespielt werden, da sich dabei lediglich die IP ändert

## URI – Uniform Resource Identifier

- ist ein Benennungsschema, welches alle Ressourcen im Internet mit einem eindeutigen Namen ausstattet
- eine URI ist per Definition entweder ein
  - **URL (Uniform Resource Locator)**
    - Identifizierung einer Ressource über ihren primären Zugriffsmechanismus (häufig http oder ftp) und den Ort (engl. *location*) der Ressource in Computernetzen
    - Bsp: `http://www.aol.com/homepages/jsmith_NYC.html`
  - oder ein **URN (Uniform Resource Name)**
    - dient als dauerhafter, **ortsunabhängiger** Bezeichner für eine Ressource
    - Der Aufbau: URN:NID:SNID:NSS  
(identifizierungskennung: namespace:subnamespace:namespacestring)
    - Bsp: `urn:Private Home Pages:USA/New York/Joe Smith`

## Namespaces - Namensräume

- ist der Kontext der betrachtet werden muss, um zu einem Namen ein Objekt eindeutig zuordnen zu können
- werden verwendet, um Konflikte bei der Namensvergabe zu verhindern
- Namensräume sind meistens hierarchisch aufgebaut
- sie können durch Trennzeichen abgetrennt und zusätzlich differenziert werden
  - übliche Trennzeichen sind : "(X-Path) ; "/" (URL); "\"(Dateisystem) ; ":"(URN | XML)
- Beispiel aus der Welt: Telefonnummern
  - Vorwahl = Namensraum
  - Nummer = Name/Identifizierbares Objekt
  - → eine Nummer kann mehrfach vergeben werden, in verschiedenen Namensräumen (Vorwahlen)

Expandet Name → ein Paar bestehend aus einem Namespace Namen und einem Localen Namen

Qualified Name (Qname) → benennt eine zusammengehörige Kombination zwischen Subjekt und Namespace (quasi, ein in der Real-World vorkommender Expandet Name Q-Name  $\subseteq$  Expandet Name)

Non-Colonized Name (NCName) → ein Teil des Namespace-Namens, in dem kein Trennzeichen vorkommen darf

Namespace-Interpretationen:

- der Namensraum ist der Identifier eines „Message Processors“ oder eine „Service Endpoint“ Adresse
- die Parameter identifizieren eine Ressource, die über einen Service erreichbar sind (Endpoint Reference/Endpunkt Referenz)

## XML-Information Set (InfoSet)

- Ein XML Dokument hat ein "information set" insofern es wohl geformt ist
- Das "information set" eines solchen Dokumentes kann folgende 11 Typen von Informationsträgern ("information items") enthalten:
  - Das Dokument
  - Die Elemente
  - Die Attribute
  - Die Verarbeitungsinstruktionen ("processing instructions")
  - Die unaufgelösten Referenzen zu Entitäten
  - Die einzelnen Buchstaben
  - Die Kommentare
  - Die [Dokumenttypdefinition](#)
  - Die unverarbeiteten Referenzen zu Entitäten
  - Die Notationsangaben
  - Die [Namensraumangaben](#)

## SEMANTIC WEB

### Ontologie:

- ein formal definiertes System von Konzepten und Relationen mit Schluss- und Korrektheitsregeln
- stellt ein Netzwerk von logischen Relationen dar (vgl. Taxonomien sind „nur“ Hierarchien)
- eine Ontologie modelliert Konzeptklassen sowie deren Eigenschaften und Beziehungen zwischen den verschiedenen Klassen. (wie bei Klassendiagrammen, nur nicht für Software^^)#
- ein Schlagwort in Bezug auf Semantic Web, da Ontologien in Bezug auf dies die Zielrichtung vorgeben → es geht darum Beziehungen zwischen Domänen herzustellen

### RDF - Resource Description Framework (System zur Beschreibung von Ressourcen)

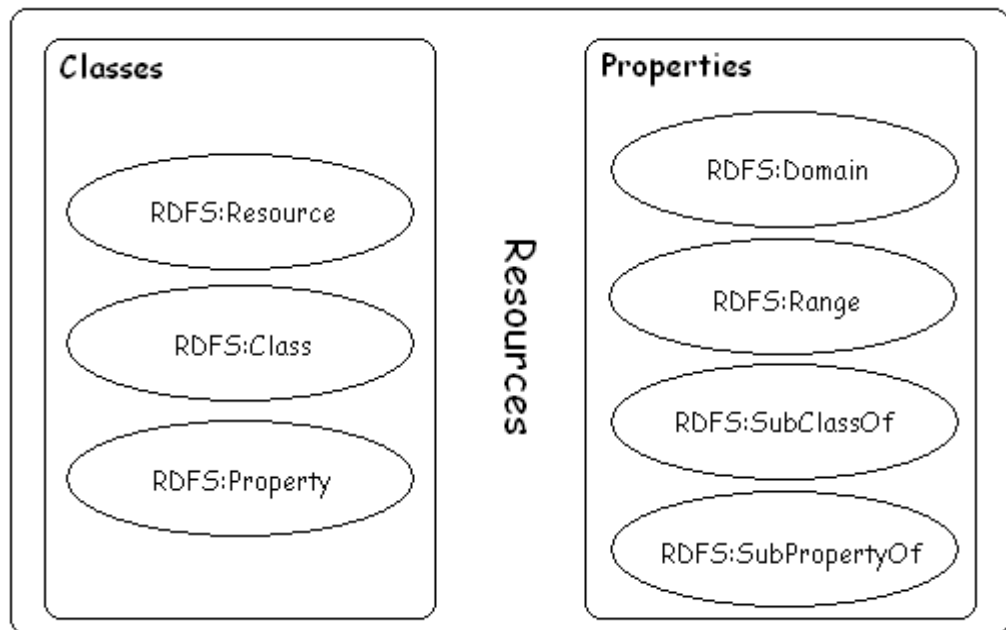
- formale Sprache zur Bereitstellung von Metadaten im WWW
- mit OWL Grundstein für Semantic Web
- Eigenschaften von Ressourcen sollen in maschinell verarbeitbarer Form beschrieben werden
- Beschreibung erfolgen als Graph (RDF-Modell(ein Verknüpfung von Tripeln)) oder als XML-Hypertext

### RDF-Modell (ein Tripel)

- subject (Resource → URI identifizierbare Elemente)      Bsp: page.html
- prädikat (Property → Beziehung zwischen Res. & Value)      hat den author
- object (Value → die Assoziationen zu den Res.      klein Kläuschen  
(kann Elementar sein, oder selber eine Res.)

### RDF-Schema

- fügt Taxonomien für Klassen und Eigenschaften (Vererbung?) und einige Metadaten hinzu (z.B. Domain)
- erweitert RDF und besitzt ein feingranulareres Vokabular zur Beschreibung



## OWL – (Web Ontology Language)

- OWL wurde designed zum domain-spezifischen folgern (von Semantik), da die meisten industriellen Standards auf Operationaler Semantiken basieren
- Es geht darum, Terme einer Domain und deren Beziehungen formal so zu beschreiben, dass auch Software (Agenten) die Bedeutung verarbeiten können.
- OWL basiert technisch auf der RDF-Syntax und historisch auf DAML+OIL
- geht über die Ausdrucksmächtigkeit von RDF-Schema hinaus
- Zusätzlich zu RDF und RDF-Schema werden weitere Sprachkonstrukte eingeführt, die es erlauben, Ausdrücke ähnlich der Prädikatenlogik zu formulieren
- Besitzt 3 Sprachebenen
  - OWL Lite
    - dient zur Schaffung einfacher Taxonomien und leicht axiomatisierter Ontologien
  - OWL DL (Description Logic)
    - erweitert OWL Lite um eine Beschreibungslogik (eine entscheidbare Teilmenge der Prädikatenlogik erster Stufe)
    - die Abbildbarkeit auf diese Logik wird durch diverse Einschränkungen von RDF-Schema Konstrukten erreicht
    - Trennung zwischen Klassen und Instanzen
    - Trennung zwischen Objekteigenschaften und Datentyp-Eigenschaften
    - Keine Kardinalitäten bei transitiven Eigenschaften
  - OWL Full
    - entspricht OWL DL bloß gibt es hier keine Einschränkung
    - eventuell sind die Ontologien unentscheidbar, die dargestellt werden, aber man kann Prädikatenlogik höherer Ordnung darstellen

### OWL unterscheidet Klassen, Eigenschaften (properties) und Instanzen

- Klassen sind Konzepte
- Klassen besitzen Eigenschaften
- Instanzen sind Individuen einer/mehrerer Klassen

### Kommunikationstypen

- beschreibt Nachrichtenaustauschmöglichkeiten diese können Descriptiv und auch Imperativ sein
- können synchron (RPC) oder asynchron (MQ) sein

### RPC- Remote Procedure Call (arbeitet synchron)

- Consumer hat eine dauerhafte Verbindung zu seinem Provider und darf mit der Verarbeitung nicht fortfahren bevor der Provider geantwortet hat
- ermöglicht den Aufruf von Funktionen über ein Netzwerk auf entfernten Rechnern
- viele Varianten
  - ONC RPC (Open Network Computing RPC)
  - DCE RPC (Distributed Computing Enviroment RPC)
  - ISO RPC (Standardisierungsversuch)
  - XML-RPC (verkapselt die Daten in ein XML Dokument und versendet diese via http)

### Funktionsweise:

- ein Portmapper nimmt den RPC-Aufruf entgegen, und Koordiniert die durch den Client gewünschten Funktionsaufrufe
- dem Portmapper müssen alle Dienste die zur Verfügung stehen bekannt sein
- der Aufruf muss den Namen(die ID) und die Parameter der Funktion enthalten
  - für das erfolgreiche ausführen ist es notwendig, das der Dienst beim Server registriert ist
  - und das es zu einem Look-up beziehungsweise Binding mit dem Client kommt

### MQ – Message Queuing (arbeitet asyncon)

- Consumer kann mit seiner Arbeit fortfahren, während sein Request verarbeitet wird
- wird oft in verteilten Systemen verwendet, wenn Daten vor der Weiterverarbeitung gepuffert werden müssen
  - BSP: Druckauftrag → landet in der Warteschlange → wir können trotzdem weiter arbeiten

### Themen die nicht aufgenommen wurden:

#### aus Architektur von Anwendungssystemen:

- Quality of Services
- Message Orinetation
- ADL
- ACME
- UML nur angeschnitten, weil schon weitläufig bekannt
- Repository
- Syntax der einzelnen Technologien
- zugrunde liegende Transporttechnologien (HTTP/IP/TCP etc.)

#### aus webbasierte Anwendungsintegration:

- CSS
- RSS
- Forms <-- ist wichtig
- Portale nur angeschnitten
- WAP & WML
- Grid Computing <-- ist aber wichtig
- Outsourcing → SaaS <-- wichtig
- „Web 2.0“

Folien die wichtig sind, ich aber rausgeworfen habe, und keine Lust hatte, sie nachzumalen:

### Portal-Metamodell

### WSDL-Struktur

### J2EE-Schichten

```

<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/2000/10/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>

```

SOAP-Anfrage vom Client (ohne HTTP-Header):

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <soapenv:Body>
    <ns1:validate
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="urn:CardValidator">
      <number xsi:type="xsd:string">1234 5678 9876 5432</number>
      <valid xsi:type="xsd:string">12/08</valid>
    </ns1:validate>
  </soapenv:Body>
</soapenv:Envelope>

```

SOAP-Antwort vom Server (ohne HTTP-Header):

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:validateResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="urn:CardValidator">
      <addReturn xsi:type="xsd:boolean">true</addReturn>
    </ns1:validateResponse>
  </soapenv:Body>
</soapenv:Envelope>

<wsp:PolicyAttachment>
  <wsp:AppliesTo>
    <xxx:DomainExpression/> +
  </wsp:AppliesTo>

  (<wsp:Policy>...</wsp:Policy> |
  <wsp:PolicyReference>...</wsp:PolicyReference>)+
  <wsse:Security>...</wsse:Security> ?
</wsp:PolicyAttachment>

```