M6

HTML = Hyper Text Markup Language

URL = Universal Resource Locator
abstrakt

HTTP = Hyper Text Transfer Protocol
Synchron
Stateless protocol
Request Message hat:

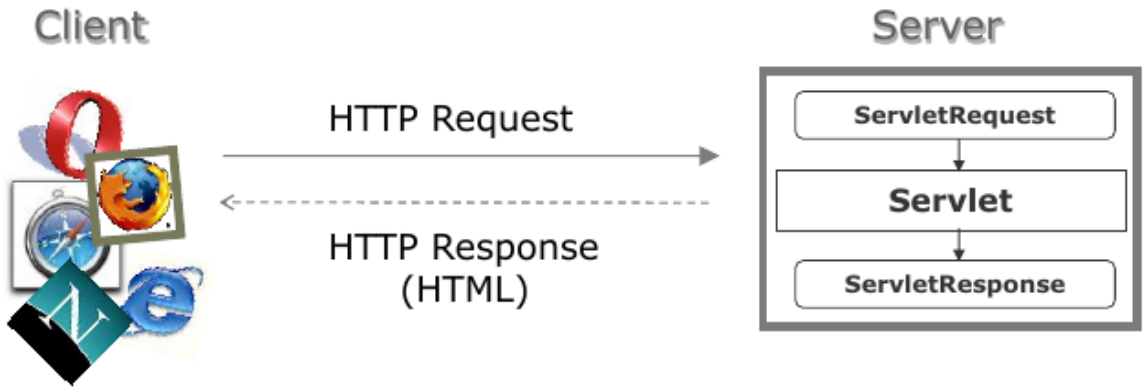
- refer to resource (URL)
- type (method) zB PUT,POST
- application data (body) = die nachricht
- application metadata
- request metadata

Response Message:

- status code
- application data (body)
- application metadata
- response metadata (server, date,..)

Servlets

Execute on server side for dynamically generated HTML documents



Consists of:

- Static resource
- dynamic generated resources
- further program elements (java classes,...)
- deployment descriptor (which servlets exist,web.xml → im WEB-INF Ordner)

Servlets deployed as .war files

Executed within a **Servlet Container** (is responsible for Life cycle of Servlet)

Hollywood principle : “Dont call us, we will call you”
→ Methods are called by Server (not by client)

HTTPServletRequest Folie 23

Java Servlet Session Tracking API: Because HTTP is stateless

Wenn Session ID in Request vorhanden → Session Object wird bei Response mitgeschickt;
Ansonsten neues Session Object erstellt und mitgeschickt

HTTPS (SSL/TSL) Verschlüsselung; Zertifikate

Access Control:

- Programmatic: *if(request.getRemoteUser() != “root”) → Applikation sicher machen*
- Declarative: *request.isUserInRole(“admin) → Webserver sicher machen*

Problems of Servlets:

- Unclear and Complex
- No direct use of HTML design tools
- No support of parallel development
- Maintenance demands a Java programmer

Java Server Pages (JSP)

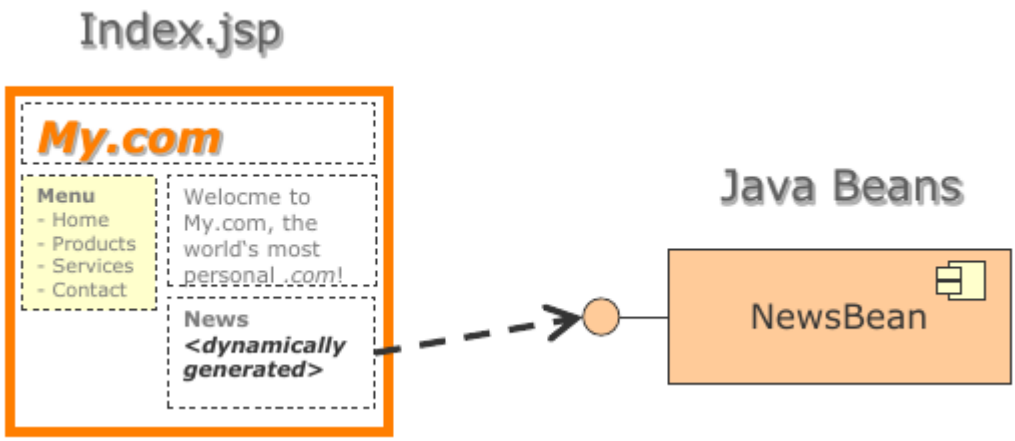
Java as Script Language

Java Code innerhalb von <% %> Tags
zB <% *new java.util.Date()* %>

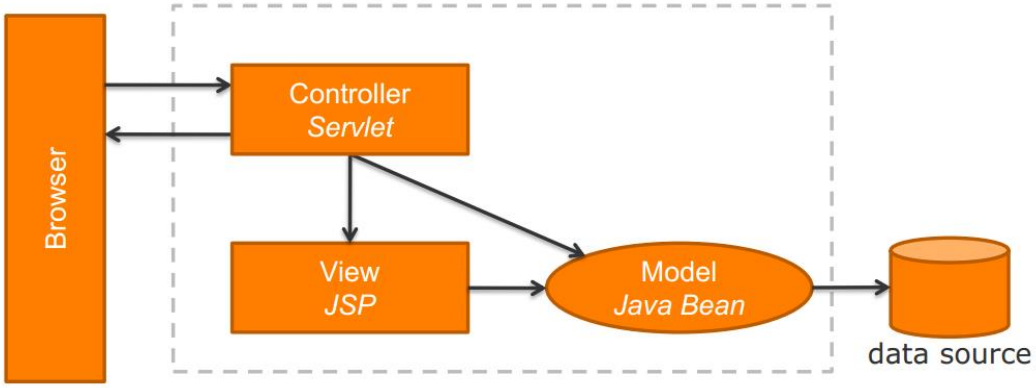
Wird in Servlets übersetzt

Java Beans

Outsourcen von Java Code



Servlet MVC Architektur



Java Server Pages

Example Java Beans (1/3)

UserData.java:

```
package myPackage;

public class UserData {
  String username, email;
  int age;

  public void setUsername( String value ) { username = value; }
  public void setEmail( String value ) { email = value; }
  public void setAge( int value ) { age = value;    }

  public String getUsername() { return username; }
  public String getEmail() { return email; }
  public int getAge() { return age; }
}
```

UserData.html

```
<html>
 <body>
   <form method="post" action="SaveName.jsp">
     <label for="name">What's your name?</label>
     <input id="name" type="text" name="username" size="20"/>
     <input type="submit"/>
   </form>
  </body>
</html>
```

Java Server Pages

Example Java Beans (2/3)

SaveName.jsp

```
<jsp:useBean id="user" class="myPackage.UserData"
        scope="session"/>
<jsp:setProperty name="user" property="*"/>
<html>
  <body>
    <a href="NextPage.jsp">Continue</a>
  </body>
</html>
```

NextPage.jsp

```
<jsp:useBean id="user" class="myPackage.UserData"
                scope="session"/>
<html>
  <body>
    You entered<br/>
    Name: <%= user.getUsername() %><br/>
    Email: <%= user.getEmail() %><br/>
    Age: <%= user.getAge() %><br/>
  </body>
</html>
```


Java Server Pages

Example Java Beans (3/3)

UserData.html

What's your name? Max

Send

SaveName.jsp

Continue

NextPage.jsp

You entered
Name: Max
Email: null
Age: 0

Pro	Contra
Simple to Learn Allows the use of HTML design Tools	Code is hard to maintain Mixture of control and presentation code

→ Combine Servlets and JSP

MVC PATTERN

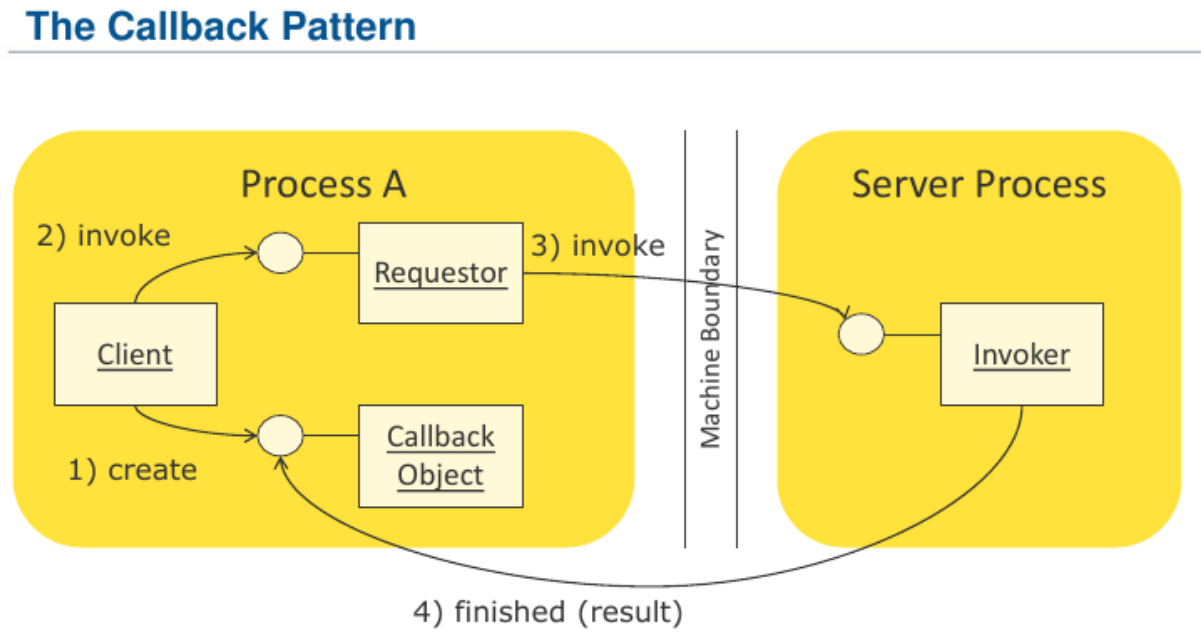
- ➔ **M**odel = Data
- ➔ **V**iew = Display
- ➔ **C**ontroller = Logic and Input

Callback Pattern

Asynchronous Invocation (=Abfrage); Client wants Data and should not wait for Response, but should continue working

→ Solution: Callback-based interface: “Abfrager/Proxy”, welches die Abfrage losschickt, aber sofort wieder zum Aufrufer zurückkommt. Sobald die Daten verfügbar sind, wird eine Methode ausgeführt

zB Observer



Node.js

Javascript runtime für Server
javascript files = modules

mit “var my_module = require(moduleName)” wird es importiert

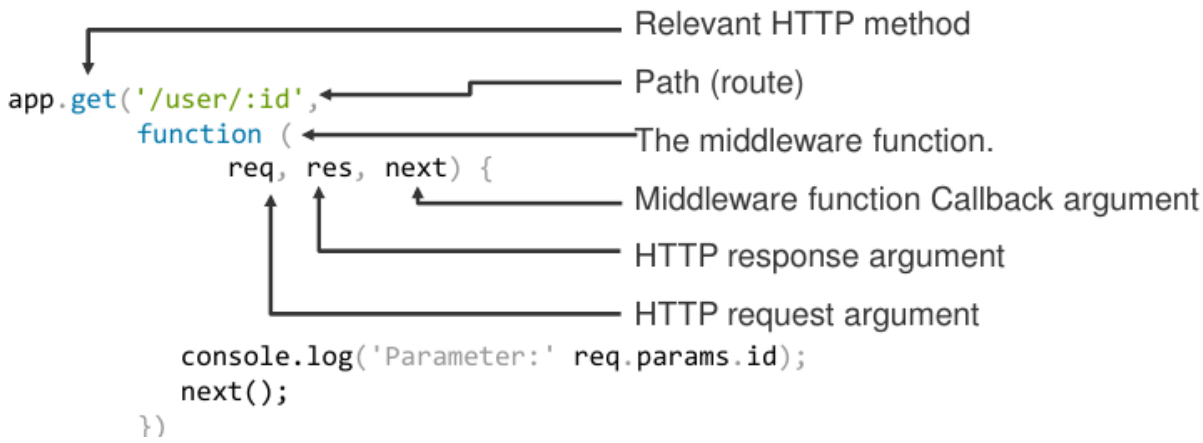
HTTPS-Server erstellen
http.createServer(function(request,response))

Express Module

= Abstraktion von HTTP request Handling

var express = require('express');
var app = express(); app.use('/stat(ic)?',express.static('public'))
app.listen(3000, function () {console.log('Started listening!')
})

Wenn man zB jetzt eine Seite auf localhost/XY machen will, braucht man folgendes:

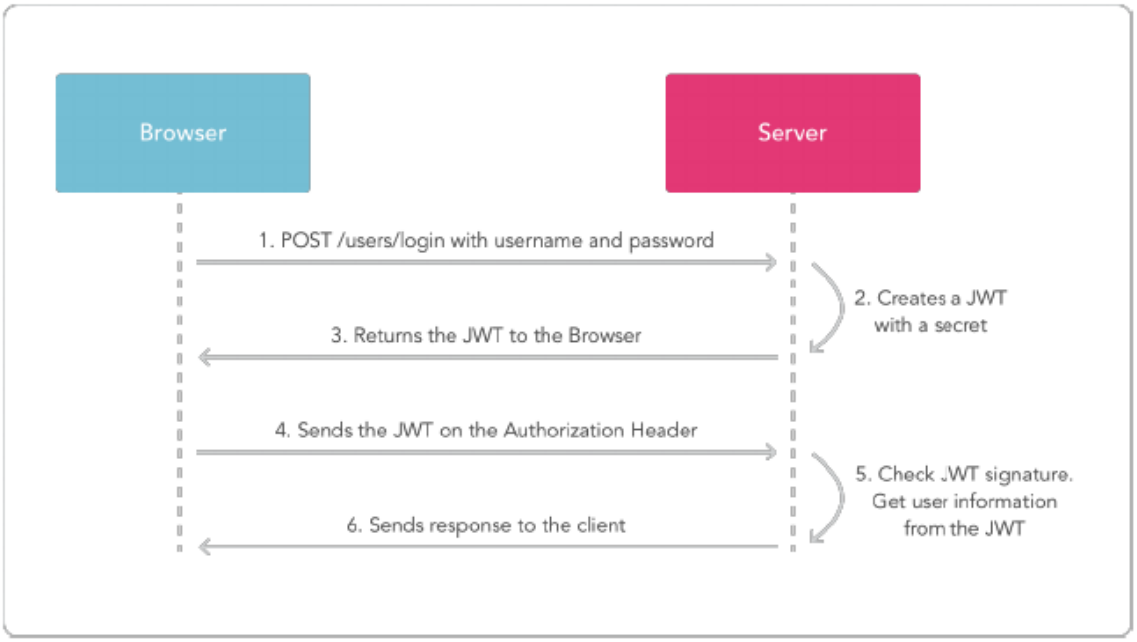


files können mit *fs.open* und *fs.readFile* gelesen werden

JSON Web Token

JSON Web Token

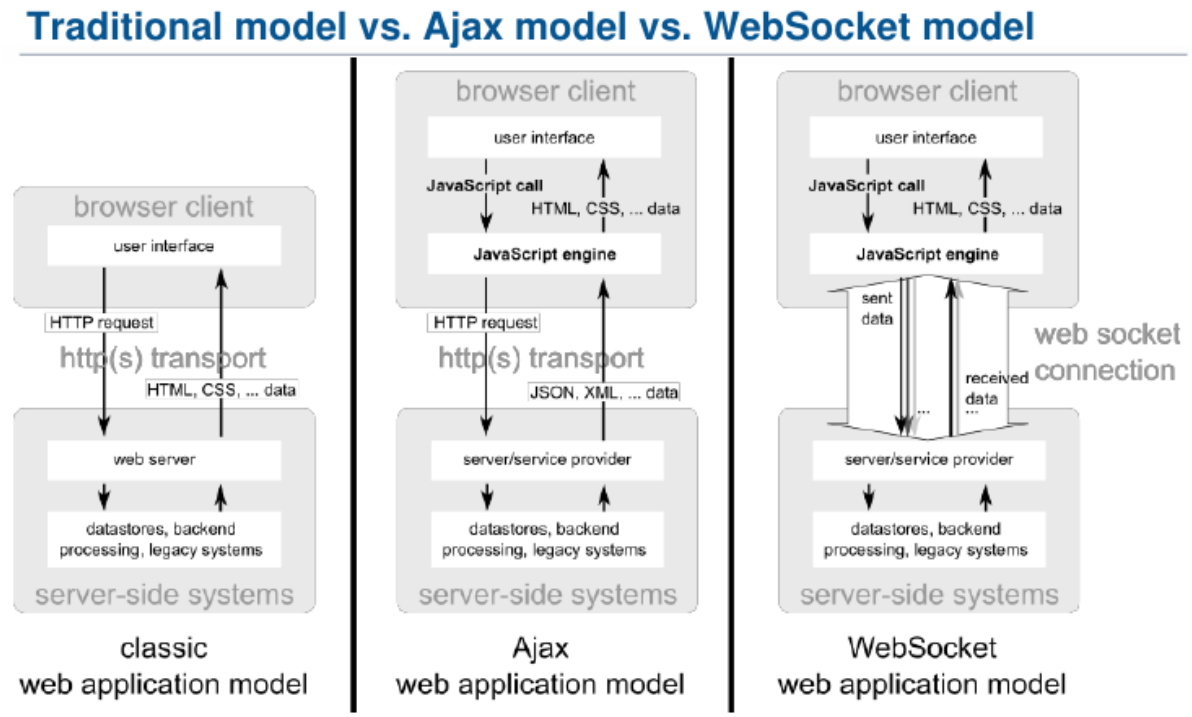
(Repitition)

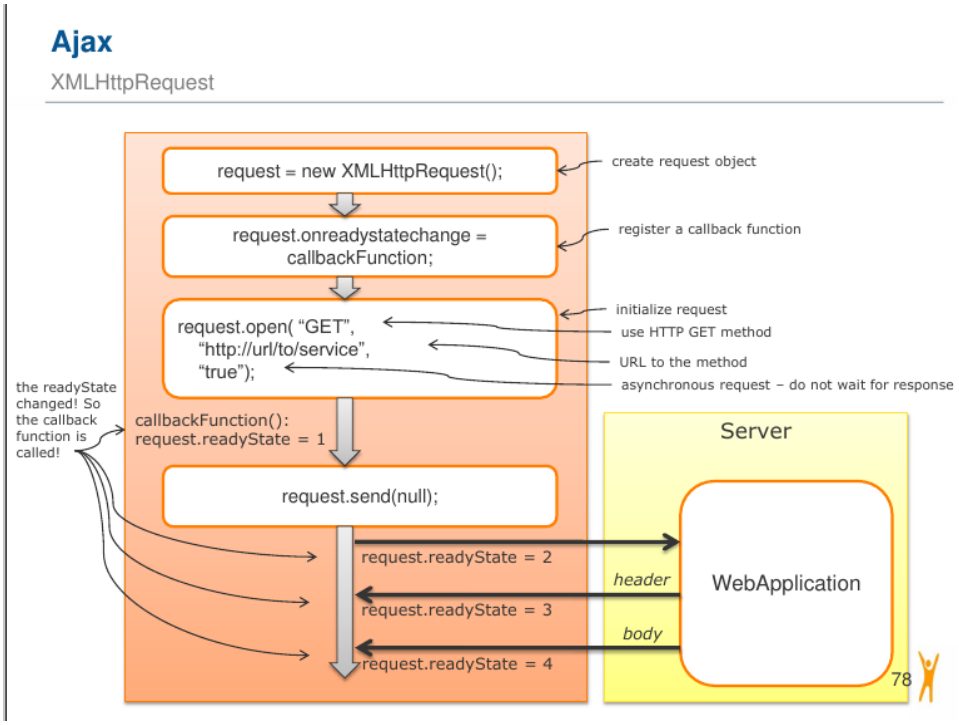


jwt.sign erstellt jwt
jwt.verify überprüft jwt
jwt.decode entschlüsselt jwt

Ajax

Asynchronous Javascript and XML





Jquery unterstützt Ajax

$.ajax

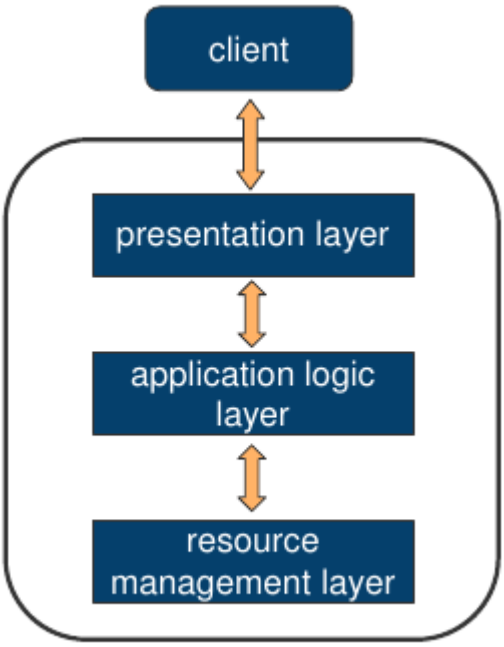
```
$.ajax("service.php",
 {
async: true,
 method: "GET",
 data: {
 'mnr': 1234567
}
});
```

Angular HTTP Service

performs Ajax request; returns Observable

```
import 'rxjs/add/operator/toPromise';
…
constructor(http: Http) {
   let params = new URLSearchParams();
   params.set('mnr', 1234567);
   http.post('service.php', {'mnr':
1234567}).toPromise().then(response => {
         response = response.json();
         if (response.ok) {
            this.name =
response.data.firstName;
         }
}).catch(function(){alert('Error!');};
}
```

M7



Service-oriented computing: sollen zum Austausch zwischen zwei (auf verschiedenen Architekturen basierenden) Systemen dienen

Web Services: self-contained modules → for machine interaction

Service Providers: are organizations that provide service implementations

Service Clients: are end-users and organizations that use some service

Service aggregators: kombinieren Services in ein neues, eigenständiges

Service Oriented Architecture (**SOA)**: ist eine Design-Art um services zur Verfügung zu stellen (s.u.)

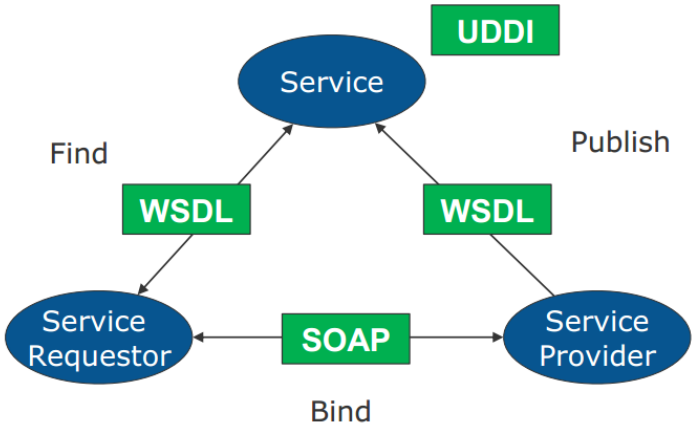
Eigenschaften von Web Services:

- eigenständige Software die eine Aufgabe erfüllt
- Design-By-Contract
- Abstrakte Programm-Logik
- Loosely coupled software module
- Wiederverwendbar
- Dynmaisch und können eingebunden werden
- sind in Standard description Language beschrieben
    - Web Service Description language (WSDL)

    - Web Application Description Language (WADL)
- Are distributed over the internet (no shit, sherlock!)

SOA

abstract pattern



	business perspective	technical perspective
service provider	owner of the service	platform that hosts the service
service requestor	business that requires certain functionality	application that invokes or interacts with the service
service registry	searchable registry of service descriptions where service providers publish their service descriptions	

SOAP

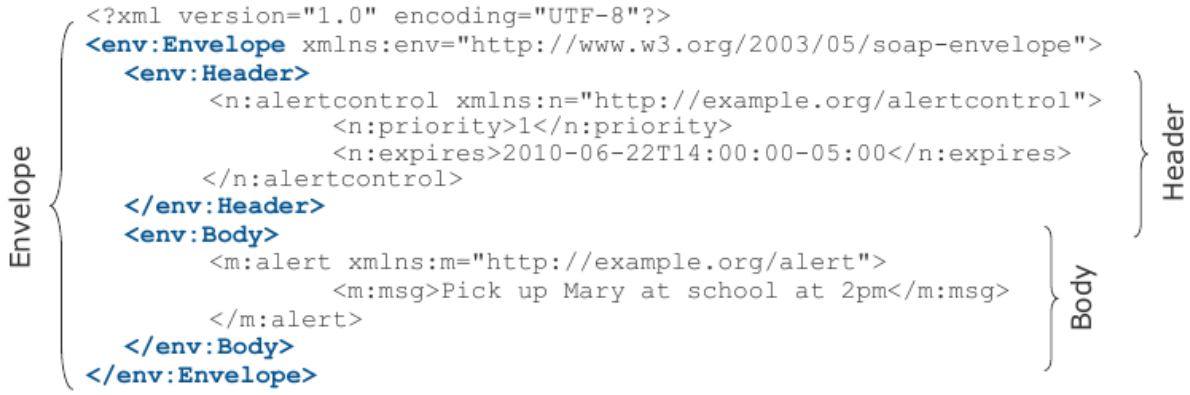
= Message exchange Format

SOAP is a **protocol** for exchanging structured information in a **decentralized**, distributed environment and defines:
an **envelope** that defines a framework for describing **what is in a message** and **how**

to process it
a set of **encoding rules** for expressing instances of application-defined data types
conventions for representing **remote procedure calls and responses**

SOAP

Example SOAP Message



RPC = Remote Procedure Calls

SOAP Styles

1) 2 Kommunikation Styles
   a) RPC (Remote Procedure Calls)
   b) Document Exchange → nur die wirkliche Message (ohne Overhead)
2) 2 Encoding Styles
   a) Encoded → Message encoded
   b) Literal → bezug auf XML Schema

Four potential styles:
- RPC/Literal
- Document/Literal
- RPC/Encoded
- Document/Encoded

Only two are relevant according to the WS-I Basic Profile 1.0:

- document/literal
- RPC/literal

WSDL 42 - Web Service Description Language

XML basierte Spezifikation zum beschreiben eines Web Services in Form von Dokumenten

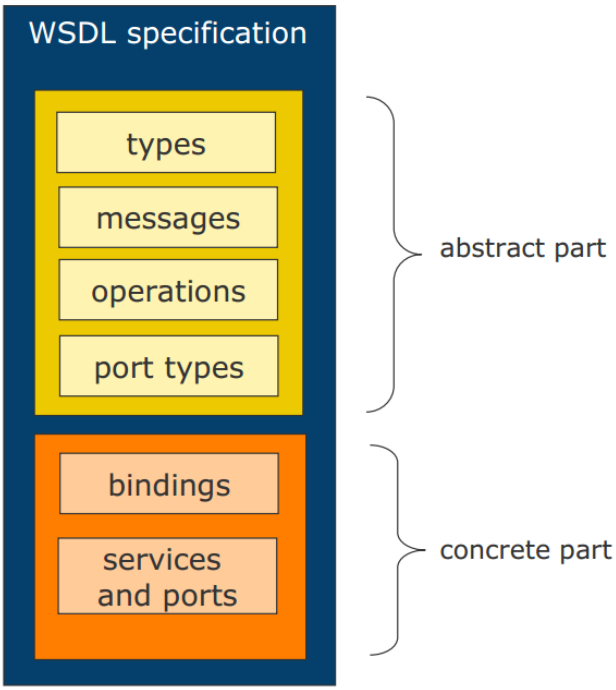
→ Vertrag zwischen Server und Client zum Aufruf

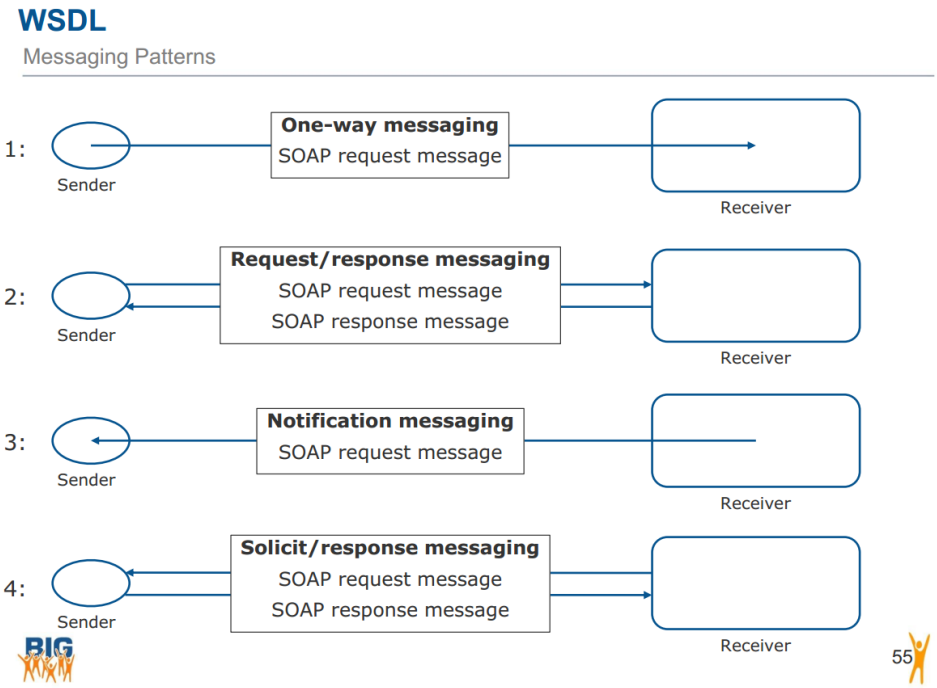
von Services

2 wichtigste Teile von WSDL

1) Service Implementation (concrete)
    a) Struktur
    b) Parameter
    c) Operationen
    d) Abstrakte Datentypen

2) Service Interface Description (abstract)
    a) Interface wird gebunden an:
    -)Konkrete Netzwerkadresse
    -)Spezifisches Protokoll
    -)Konkrete Datenstrukturen
    b) Web Service Client kann an solche Implementation gebunden werden und das Service aufrufen





UDDI

Universal Description Discovery and Integration
definiert ein Schema zum veröffentlichen und finden von Web Services von Unternehmen
“Gelbe Seiten” für Web Services

- Three main pillars of a UDDI registry
  - “white pages”: address, contact, and known identifiers
  - “yellow pages”: industrial classification
  - “green pages”: meta information on services (reference to service description in WSDL)

RESTful Web Services

Representational State Transfer
Architektur Prinzipien
Rund um das Konzept Ressourcen und die Manipulation von Ressourcen mittels HTTP-Req

Method	Action	Properties
GET	Retrieve a resource	safe, idempotent
PUT	change the resource state, i.e., update a resource	Idempotent
POST	create a new resource	Idempotent
DELETE	remove a resource	

safe = State wird nicht verändert
idempotent=mehrere Aufrufe machen das gleiche wie ein einziger Aufruf

Problem:

Server muss den richtigen State haben und Client muss über den richtigen State bescheid wissen (trotz Statlessness) → Session overhead?

URI wird zum adressieren von Resourcen verwendet (selbst beschreibende Interfaces) z.B.:

- http://www.mycompany.com/users
- http://www.mycompany.com/users/internal
- http://www.mycompany.com/users/internal/34
- http://www.mycompany.com/users/external
- http://www.mycompany.com/users/external/432
- http://www.mycompany.com/users/external/premium
- http://www.mycompany.com/users/external/regular

Gibt auch Java API für Restful Web Services

Web Socket

Modul: ws

TODO???

M8 - Linked Open Data

Open Data Inhalt kann von jedem gratis benutzt, verändert und geteilt werden für jeden Verwendungszweck.

Prinzipien:

1) Erreichbarkeit und Zugang
    a) Immer erreichbar (vorzugsweise übers Internet)
    b) muss in praktischer Form verfügbar sein
2) Wiederverwendung und Distribution
    a) Die Daten müssen Wiederverwendung und Weitergabe erlauben (und auch den Mix mit anderen Daten-sets zulassen)
3) Universelle Mitarbeit
    a) Jeder kann nutzen, verändern und weitergeben (keine Diskriminierung)

Erweiterte Prinzipien

1) Komplett → alle öffentlichen Daten werden verfügbar gemacht
2) Primär → direkt von der Quelle in feiner Granularität
3) Zeitlich → schnellstmöglich
4) Zugänglich → für jeden
5) Machine Readable
6) Nicht-Diskriminierend
7) Uneingeschränkt → jeder Nutzer hat die gleichen Rechte
8) Lizenzfrei
9) Permanent
10) Gratis Zugänglich

Linked Data

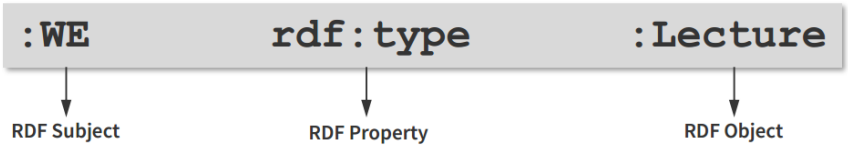
Content can be read and interpreted correctly by machines (semantic web) →

Verknüpfungen durch Semantisches Verständnis

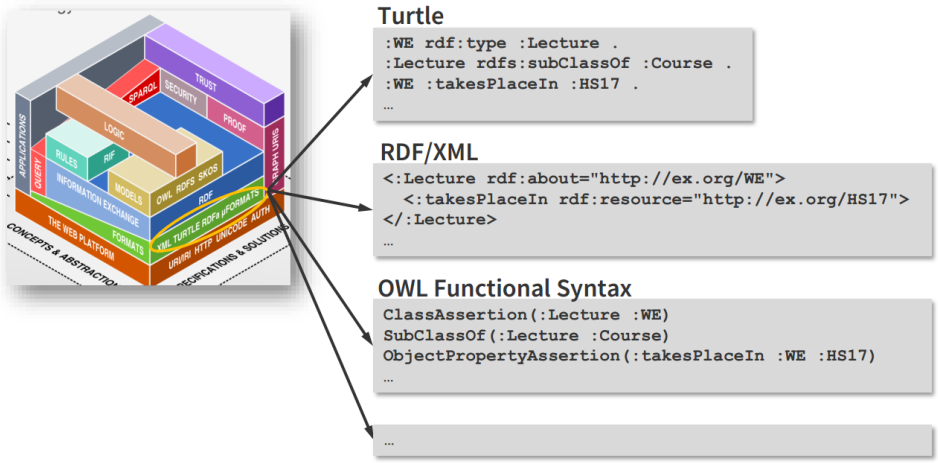
Ontologie = sprachlich gefasste und formal geordnete Darstellungen einer Menge von Begrifflichkeiten und der zwischen ihnen bestehenden Beziehungen in einem bestimmten Gegenstandsbereich; Wissensaustausch von Anwendungsprogrammen und Diensten

→ Meaning/Semantik musste zuerst formalisiert werden damit sie interpretiert und verlinkt werden kann

RDF = Resource Description Framework



Subjekt, Prädikat, Objekt → wie ein deutscher Satz



SPARQL = graphenbasierte Abfragesprache für RDF

ASK – Is the Amazon river longer than the Nile river?

```
PREFIX prop: <http://dbpedia.org/property/>
ASK{
  <http://dbpedia.org/resource/Amazon_River> prop:length ?amazon .
  <http://dbpedia.org/resource/Nile> prop:length ?nile .
  FILTER(?amazon > ?nile) .
}
```

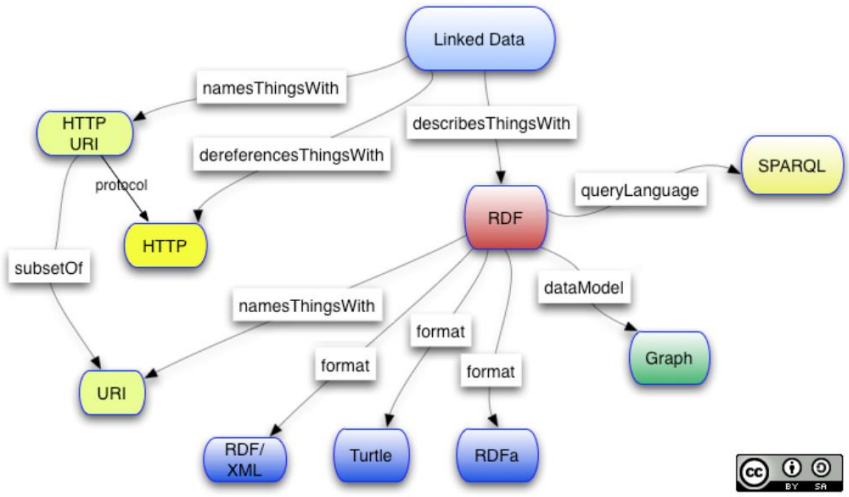
SELECT – Find all landlocked countries with a population > 15 mio.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?country_name ?population
WHERE {
   ?country a type:LandlockedCountries .
   ?country rdfs:label ?country_name .
   ?country prop:populationEstimate ?population .
   FILTER (?population > 15000000) .
}
```

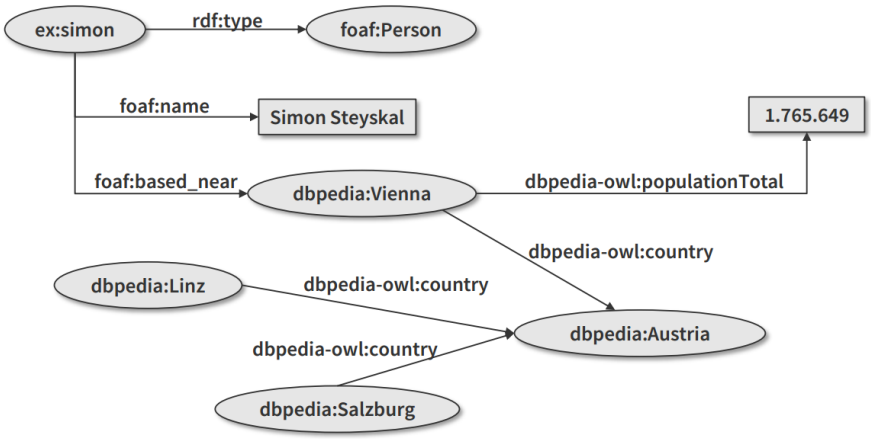
Linked Data Principles:

1) Alles bekommt eine URI (Papier, Personen, Gespräche, Themen,...)

2) HTTP URIs werden verwendet damit Personen nachsehen können was diese Namen bedeuten
3) Wenn URI geöffnet wird → sinnvolle Informationen mit SWT zeigen (Java GUI)
4) Beim Dereferenzieren der URIs sollte mehr Information feststellbar sein



FOAF = Friend of a Friend; Präfix für SPARQL Abfrage bezüglich Personen



Non Functional Requirements

Tuning Methoden:

- -) Asset Optimization
    - Größe von Text-Blöcken minimieren (Whitespaces, New Lines, etc. → Tools e.g. Uglify JS)
    - Image Optimierung→ unnötige Bilder weg; CSS3 Effekte wenn möglich statt Bildern, Vektor und Raster Grafiken verwenden; Kompressions Mechanismen (richtiges Format und skalierbare Bilder)
    - Font Optimierung

- -) Reduzieren von HTTP Requests → Bandbreite sparen; Verwendung von Inline Javascript, CSS Sprites; keine 3rd Party Frameworks; CSS und JS Kombinieren
- -) Redirects vermeiden
- -) Unnötige Downloads entfernen
- -) HTTP Caching
- -) Server Optimierung

Kompression von transferierten Daten → Programme verwenden: **GZIP**, LZMA (7zip), …

GZIP

- nicht standardmäßig enabled bei jedem Browser
- funktioniert am besten bei Text-based Assets
- Ordnet JSON Daten neu an (speichersparend)
- kann auch Ressourcen größer machen (zB bei PNG Files problematisch) → wenn sich byte Sequenzen nicht wiederholen

HTTP Caching:

- HTTP Header Directives sagen was, wann und wie lange eine Response gecachet wird
- Mechanismus: Freshness (Response kann wiederverwendet werden ohne erneut vom Server gecheckt zu werden), Validation (check ob noch gut/gültig mit Validation Token), Invalidation (Resource expired → renew/remove/update)
- **Max-Age** wird angegeben in Sekunden (z.B. 120 Sekunden)
- Cache Policy kann verändert werden (Gültigkeitsdauer, was wird gespeichert, etc.)
- **Delta Encoding** wird verwendet→ zuerst ganzes File, danach nur noch PATCH files mit den Änderungen

Server Optimierung

- Information wird am Server gespeichert und als statische Information verfügbar gemacht
- Files werden zurückgegeben wenn sie gebraucht werden
- Bsp: Bundespräsidentenwahl; ORF Seite hatte 14.100 HTTP Requests pro Sekunde
- Datenbankoptimierung → kleinere Tabellen, Index verwenden, DBMS settings, optimierte Abfragen
- HTTP/2 → encryption; Compressing headers, request pipelining, mehrere HTTP Requests über eine Verbindung
    - HTTP/2 Server Push → mehrere Responses für nur einen Request

Security (CIA):

- Confidentiality: nur der gewünschte Gegenüber bekommt die Informationen
- Integrity: Daten bleiben bei Übertragung unverändert

- Authentification: Sicherstellen, dass mit dem richtigen Server kommuniziert wird
- HTTPS→ HTTP encrypted mit SSL/TLS
    - HSTS man in the middle attacks können nicht vorkommen
    - HTTP Request kann erst nach sicherem Verbindungsaufbau gesendet werden (SSL Handshake)

Performance:

- Latenz: Anzahl der Requests minimieren
- Größe: keine unnötigen Ressourcen downloaden, caching verwenden, minify
- Kompressions Mechanismen (zweckgerecht) verwenden und richtes Fileformat
- geeignete Algorithmen

HTTPS

SSL Certificates

- Authentification
    - Identifying information about the certificate holder
        - Who, Where, Contact information …
    - Signed with a public/private key
    - Validity period, may be revoked
- Certificates are valid for a certain host only (Common Name)
    - Wildcards (e.g. *.example com) match example.com, a.example.com, but not a.b.example.com
    - Subject Alternative Names (SANs) allow the definition of additional hosts
- HTTPS certificates come from a Certificate Authority (CA)
    - Ensure correct identity of the third party
    - Certification chain to a root authority which must be trusted and are imported in the browser

Alte Fragen

Welche 4 HTTP-Methoden werden mit REST verwendet? Ordnen Sie diese folgenden Anwendungsfällen zu:

GET: Ressource abfragen
PUT: Ressource update (Eselsbrücke: U kommt in P**U**T und **u**pdate vor; In POST nicht)
POST: Ressource erstellen
DELETE: Ressource löschen

Beispiel:

a) Anzeige aller Schüler
b) Löschen eines Schülers
c) Ändern eines Schülers
d) Erstellen eines Schülers

Lösung:

a) GET
b) DELETE
c) PUT
d) POST

Nennen und beschreiben Sie 4 Technologien/Standards, welche Sie in Verbindung mit Linked Open Data gelernt haben.

RDF

RDFS

OWL

SPARQL

HTML-Lückenfüllen mit AngularJS (1.x)-Direktiven (mit ng-app, ng-controller, ng-model, ng-repeat, etc)

Frage 2. Ergänzen Sie den folgenden Code mit AngularJS 1.x-Direktiven und Ausdrücken, so dass (unter Annahme englischer Spracheinstellungen) das angegebene Ergebnis erscheint! *(8 Punkte)*

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Book App</title>
<script src="angular.js"></script>
<script type="text/javascript">
var App = angular.module('Main',[]);
App.controller('ctrl',function($scope){
        $scope.books = [        {"title": "",
         "lend": [{"retDate": new Date(2017,06,13)}]
        },
        {"title": "Web Engineering",
        "lend": [{"retDate": new Date(2017,06,17)},
          {"retDate": new Date(2017,07,13)}]
        }
        ];
});</script>
</head>
<body ng-app="Main" ng-controller="ctrl">
<h1>Books</h1>
<article ng-repeat="x in books">
<form name="bookForm">
<p><input type="text" required="required" name="bookname" ng-model="x.
   title" /></p>
<p ng-repeat="l in x.lend">{{l.retDate}}</p>
<div style="color:red" ng-show="bookForm.bookname.$error.required">
        There must be a title!</div>
</form>
</article>
</body>
</html>
```

Books

Jul 13, 2017

There must be a title!

Web Engineering

Jul 17, 2017

Aug 13, 2017

Multiplechoice-Fragen

Die Kommunikation in SOA erfolgt üblicherweise synchron	Nein
Die zwei SOAP-Kommunikationsstile sind Remoteprozeduraufrufe und Dokumentenaustausch.	Ja
WSDL ist ein Standard, der den Austausch zwischen Webserviceanbieter und Webserviceanfragenden beschreibt.	Nein?
Daten, die als Open Data veröffentlicht wurden, dürfen nicht für kommerzielle Zwecke eingesetzt werden.	Nein; Open Data darf für alles verwendet werden
Open Data sollte wenn möglich in aggregierter Form veröffentlicht werden.	Nein; Zum Beispiel nicht nur Steuereinnahmen von Österreich sondern von den einzelnen Bundesländern,Städten,.. → feinste Granularität
In Single-Page-Webanwendungen antwortet der Server typischerweise mit Daten, die der Client rendert.	Ja
Wird merge in einem EntitiyManager für ein Objekt aufgerufen, das noch nicht persistiert ist, wird eine Exception geworfen.	Nein
@Embeddable wird verwendet für Klassen, deren Felder in der Table der besitzenden Entität abgebildet werden.	Ja
ng-show ist ein Alias für ng-if	nein
Bei orf.at wird das CSS-Styling via IDs gegenüber Klassen bevorzugt. (Anm: Kommt wahrscheinlich nicht)	Nein