FH Bielefeld
University of
Applied Sciences

Campus Minden

Webbasierte Anwendungen

SS 2018

WebServices

Dozent: B. Sc. Florian Fehring

mailto: florian.fehring@fh-bielefeld.de

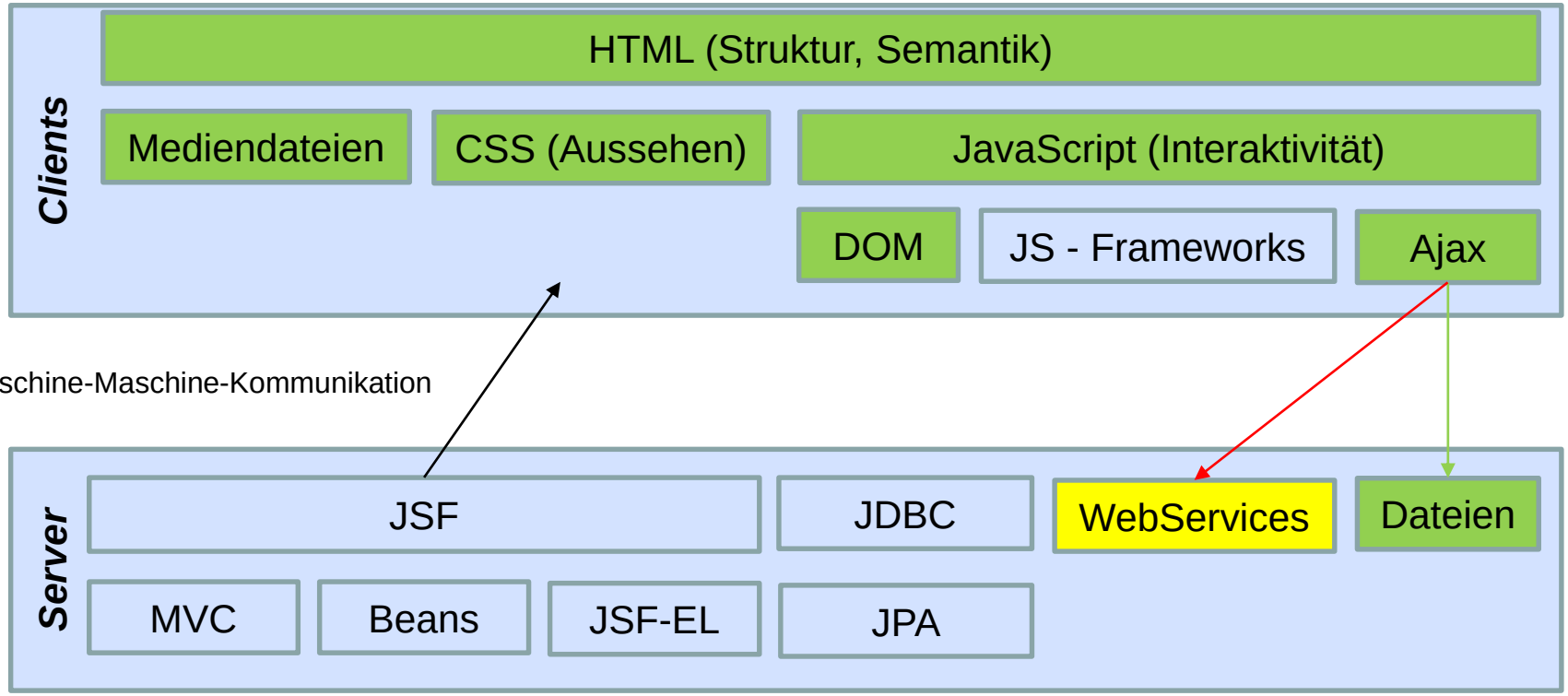
Studiengang Informatik

WebServices

1. **Kontext und Motivation**
2. SOAP WebServices
3. REST WebServices
4. Darüber hinaus
5. Projekt

Problemfelder

Mensch-Maschine-Kommunikation



Maschine-Maschine-Kommunikation

Anforderungen

Welche Anforderungen werden als nächstes bearbeitet?

TODO

- Artikel zum Server übertragen
- Kommentare zum Server
- Medien zum Server
- Kommentare speichern
- Kommunikation untereinander

DONE

- Technologische Grundlagen erarbeiten
- Was ist eine Web-Anwendung?
- News darstellen
- Projekte vorstellen
- Aufgaben darstellen
- Formular für Kommentare
- Schickes Design für die Seite
- Mediendatein einbinden
- Animationen
- Mehrsprachen-Fähigkeit
- (lokales) Speichern von Artikeln
- Client-Position anzeigen
- Offline-Verwendung ermöglichen
- Inhaltsverzeichnisse
- Formlareingaben in Seite einfügen
- Navigation über Tastaturkürzel
- Externe Inhalte einbinden
- Artikel vom Server einbinden
- Kommentare vom Server

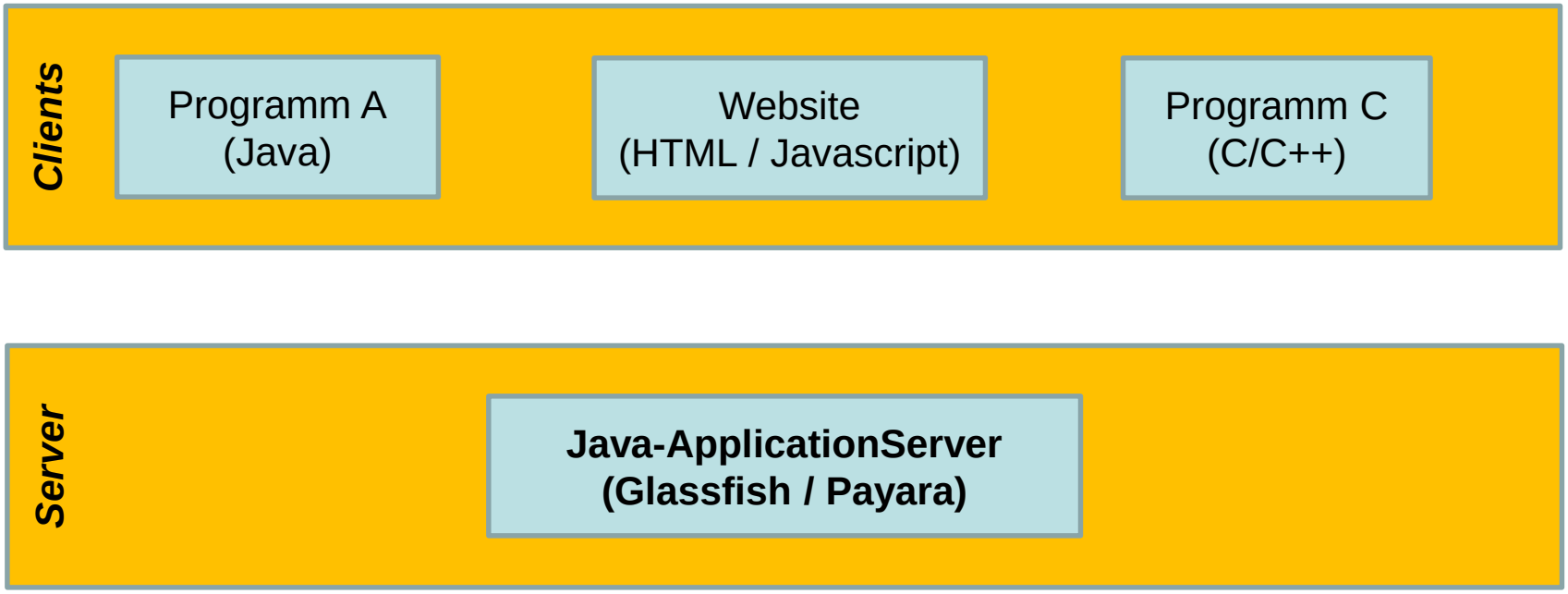
Problemstellung

Anforderung

Andere Firmen sind auf unsere Produkte aufmerksam geworden und möchten gerne Produktdaten von uns. Sie möchten diese in Kataloge integrieren oder auf ihrer eigenen Seite darstellen.

Problem

Wie können wir unsere Produktdaten anderen zur Verfügung stellen?



Entfernte Aufrufe

Definition: Entfernte Aufrufe sind die Aufrufe von Funktionen, Methoden oder Diensten, die von einem Server angeboten werden.

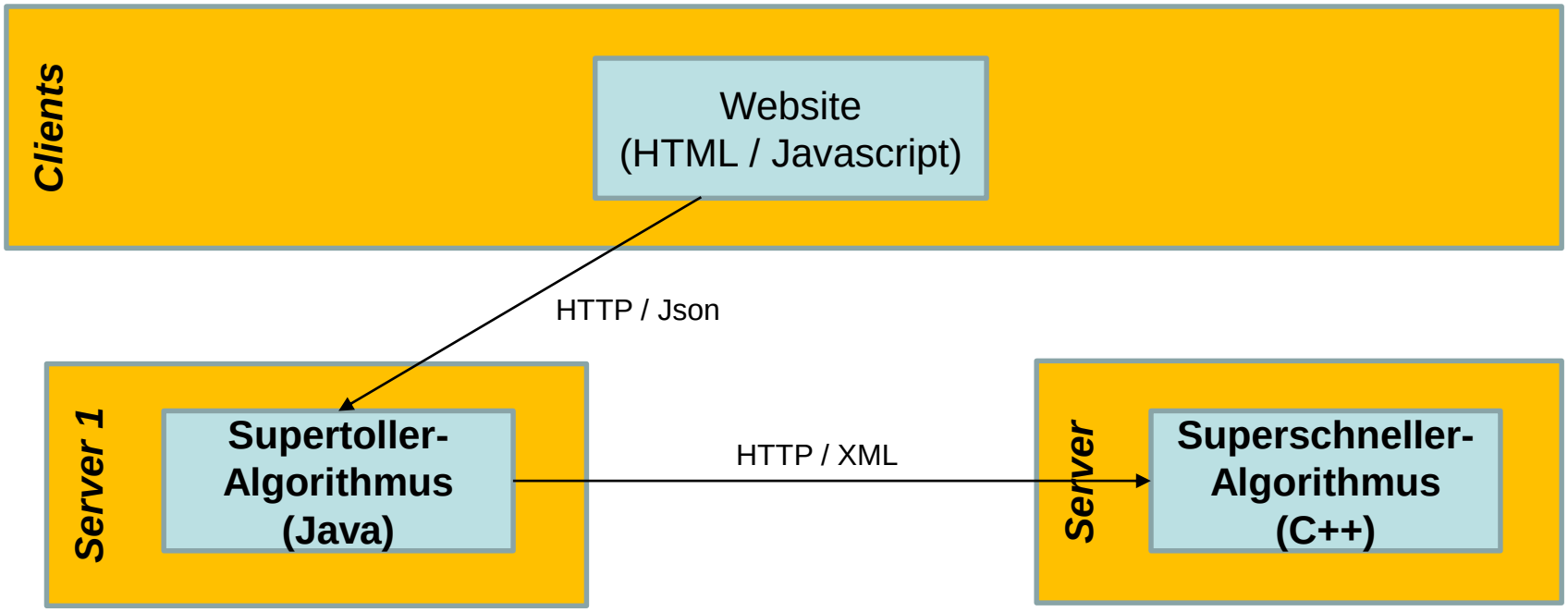
Techniken:
- Remote-Procedure-Call (RPC)
- Remote Method Invocation (RMI)
- CORBA

Bekannt aus Verteilte Systeme und Kommunikationsnetze

Nachteile im Webumfeld:
- Nur für die Kommunikation von Programmen derselben Sprache (RPC,RMI)
- Komplex (CORBA)
- Eigene Protokolle für die Datenübertragung
- Eigene Ports oder Tunneln notwendig

WebService

Definition: Ein Web Service ist eine Schnittstelle, mit der Daten zwischen Programmen ausgetauscht werden können. Dabei werden Web-Technologien verwendet.



WebService

Eigenschaften

- Basiert auf offenen (Web-)Standards (HTTP, XML, Json)
- Verbindet heterogene Anwendungen (Java, C++,JS, PHP,…)

Vorteile

- können in allen Programmiersprachen geschrieben werden
- hohe Durchdringung durch Verwendung offener Standards
- überschaubare Einstiegshürden
- Leichtes Konzept, Unterstützung durch IDEs

ServiceOrientierteArchitekturen

Definition: *Service Orientierte Architekturen (SOA) sind Architekturmuster zur Nutzung von Algorithmen, auch verschiedener Herkunft, zur Realisierung von Geschäftsprozessen.*

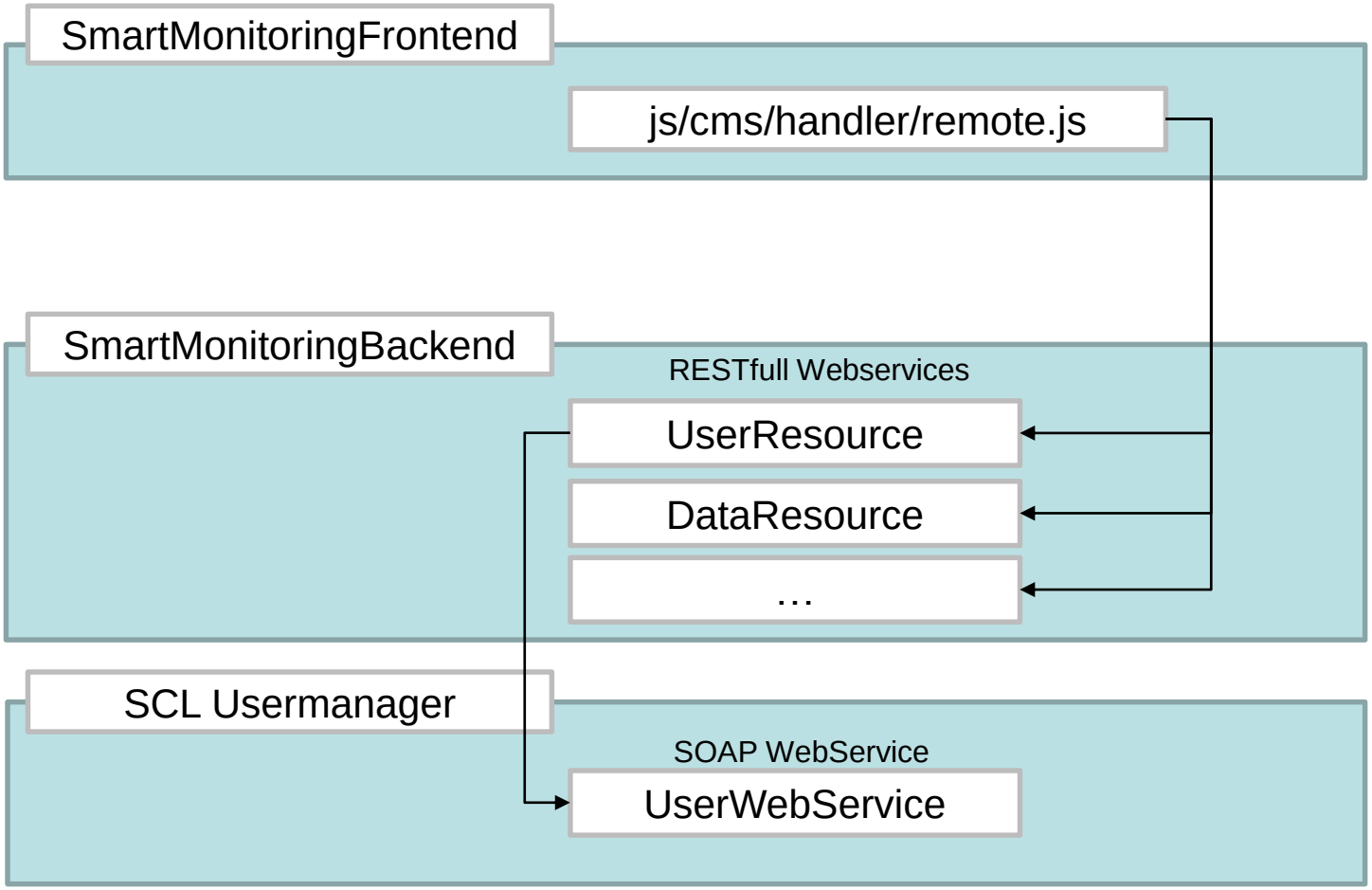
Eigenschaften:

- Verteilung von Anwendungslogik auf verschiedene Anwendungen
- Anwendung A benutzt die Dienste der Anwendungen B,C,…
- Lose Kopplung der verschiedenen Anwendung
- Services können zusammengefasst werden. (Orchestration)

WebServices:

- Sind eine Technologie zur Realisierung von SOA
- Es sind auch SOA ohne WebServices denkbar
- Ein Webservice definiert noch keine SOA

Praxisbeispiel



WebServices

1. Kontext und Motivation
2. **SOAP WebServices**
3. REST WebServices
4. Darüber hinaus
5. Projekt

SOAP WebServices

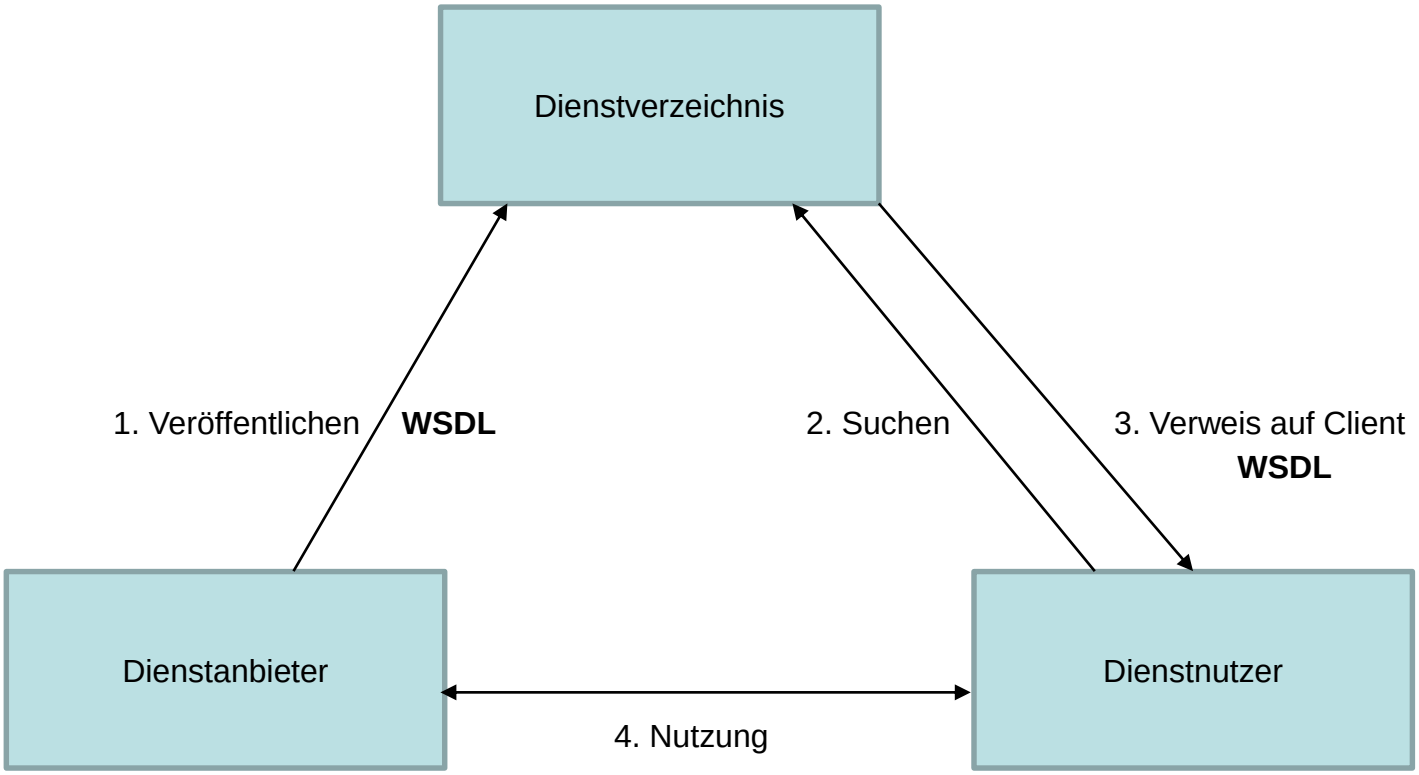
Möglichkeiten

- Datenaustausch zwischen Applikationen
- Automatische Generierung von Quellcode

Eigenschaften

- standardisiert für Java als JAX-WS
- SOAP bezogene Klassen im Package: javax.jws.*
- Objektbezogen (1 Objekt = 1 WebService = 1 fachlicher Kontext)
- Deklarativ: Klassen und Methoden werden durch Annotationen zu einem WebService.
- Standardisiertes Protokoll, nutzt XML und HTTP POST
- Die Beschreibung des WebService kann generiert werden
- Aus den Beschreibungen sind Klassen generierbar

SOAP Komponenten



SOAP Klassen

Eigenschaften

- Normale Klasse der Geschäftslogik
- Schnittstellen-Klassen
- Erweitert um **deklarative** Anweisungen (Annotationen)
  - Kennzeichnen das Objekt als zugreifbar von Außen @WebService
  - Kennzeichnen eine Methode als Zugriffspunkt @WebMethod
  - Kennzeichnen Parameter als zur Schnittstelle gehörend @WebParam

```
 @WebService
public class UserWebService {

    @WebMethod(operationName = "loginUser" )
    public String performLogin(
          @WebParam(name = "username" ) String username,
          @WebParam(name = "password" ) String password) {

          User user = this.login(username,password);
          return user.toString();
    }
}
```

SOAP Veröffentlichen

SOAP WebServices werden veröffentlicht, indem sie auf dem ApplicationServer deployed werden.

SOAP WebServices im GlassFish:

- WebServices werden unter der Anwendung gelistet
- Zugriff über „View Endpoint“ auf die WSDL
- Zugriff über „View Endpoint“ auf eine generierte Oberfläche zum Testen des WebServices

Modules and Components (7)

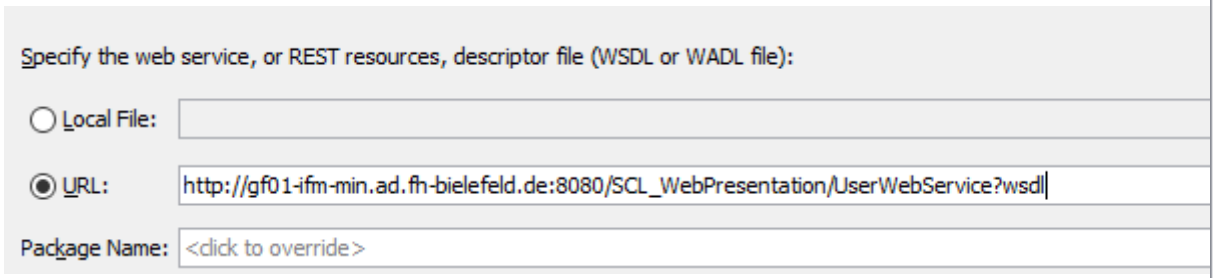
Module Name	Engines	Component Name	Type	Action
SCL_WebPresentation	[ejb, web, webservices, weld]	----------	----------	Launch
SCL_WebPresentation		default	Servlet	
SCL_WebPresentation		Faces Servlet	Servlet	
SCL_WebPresentation		jsp	Servlet	
SCL_WebPresentation		LoginBean	StatefulSessionBean	
SCL_WebPresentation		ApplicationConnectorWebService	Servlet	View Endpoint
SCL_WebPresentation		UserWebService	Servlet	View Endpoint

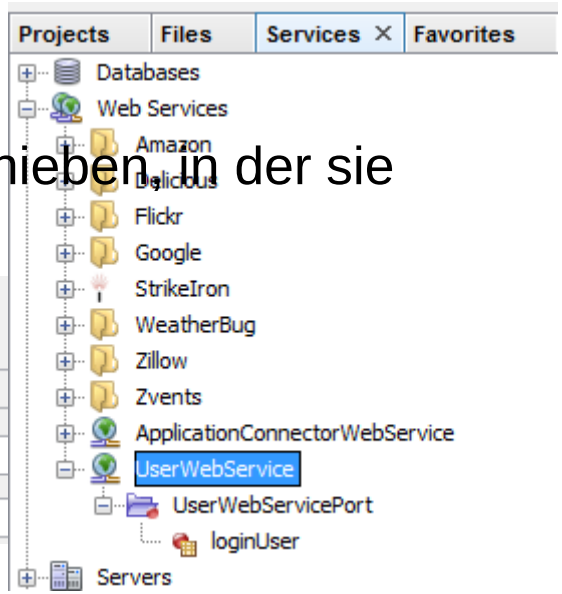
SOAP Verwenden

SOAP WebServices können über Kommandozeile oder bequemer über die IDEs verwendet werden. Alle benötigten Klassen werden generiert.

SOAP WebServices in NetBeans:

1. „Services“-Window / Web-Services
2. Rechtsklick / Web-Service hinzufügen
3. URL des WebService eingeben
4. Der WebService erscheint nun in der Liste
5. Methode die benutzt werden soll suchen
6. Methode per Drag & Drop in die Methode schieben, in der sie verwendet werden soll.





WebServices

1. Kontext und Motivation
2. SOAP WebServices
3. **REST WebServices**
4. Darüber hinaus
5. Projekt

REST WebServices

Möglichkeiten

- Datenaustausch zwischen Applikationen
- Verwendung normaler URLS
- Selbstlernende Applikationen

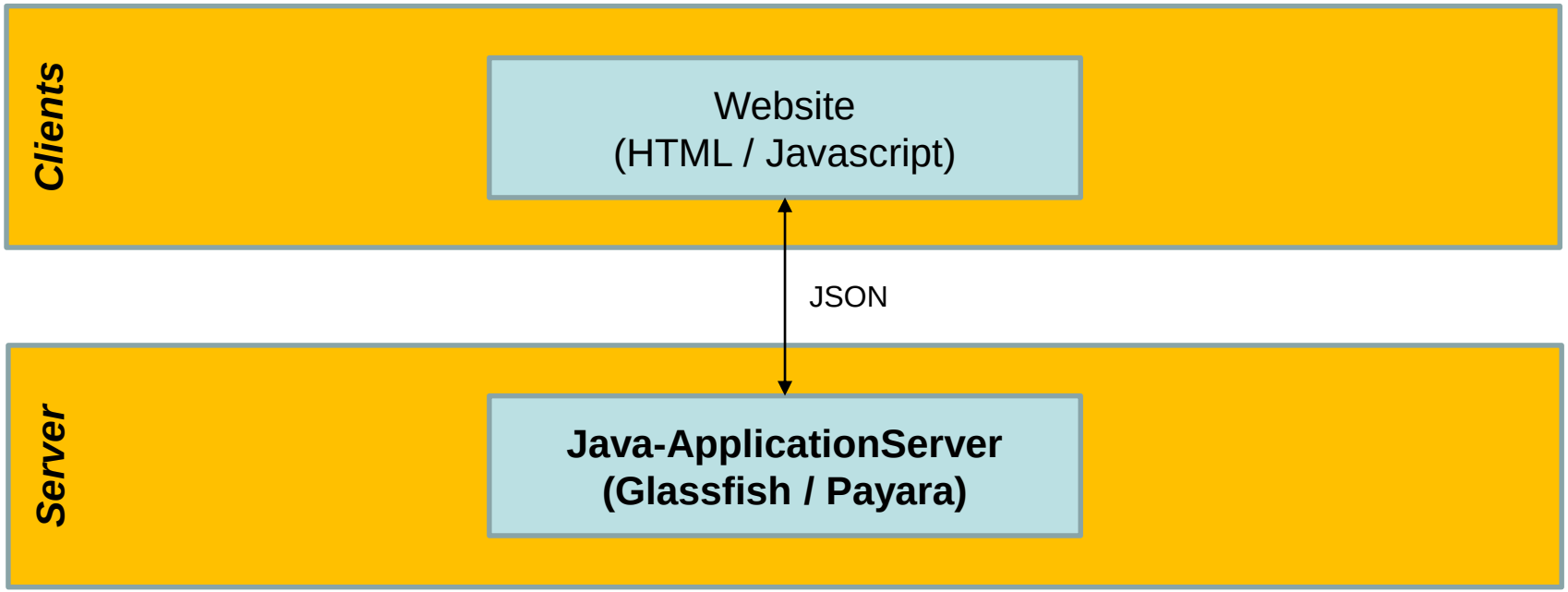
Eigenschaften

- Standardisiert für Java als JAX-RS
- Jeder Application-Server (ab JavaEE6) enthält eine JAX-RS Implementierung.
- Mit Bibliothek auch ohne Application-Server verwendbar
- Referenzimplementierung Jersey: (https://jersey.dev.java.net/)
- Ressourcenbezogen (Text, Bilder, …)
- Jede Ressource hat eine URL
- Ressourcen haben Methoden
- Methoden werden über HTTP abgebildet (GET, POST,…)
- Ist nicht nur Technologie sondern auch Architektur-Stil

REST I – Server und Clients

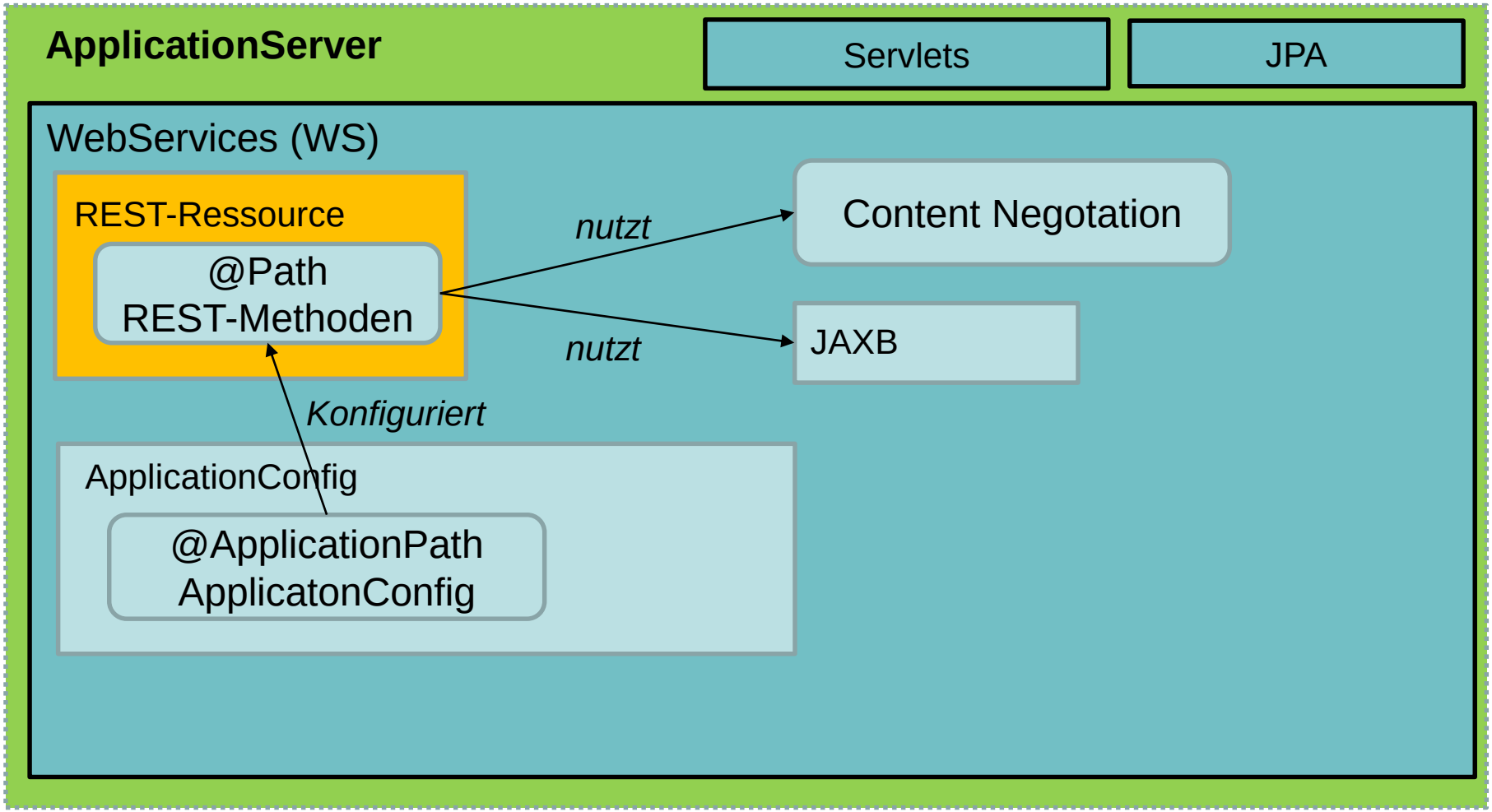
Gemeinsame Eigenschaften

- sind in vielen Sprachen implementierbar
  - In Java mit „Jersey“ (Referenzimplementierung)
  - In JavaScript mit Ajax-Technologie
- müssen das Datenformat für die Ausgetauschten Daten aushandeln
  - JSON derzeit häufigstes Austauschformat



Legende: API Konzept

REST I – ApplicationServer



REST II - Ressourcen

Eigenschaften

- tragen Daten über ihre Methoden (Texte, Bilder, Objekte, …)
- bilden eine logische Einheit (1 Klasse)
- sind stark voneinander abgegrenzt (Starke Aufgabenteilung)
- besitzen eine URL (http://service.de/resource)

Implementierung

- Eine Resource wird durch **genau eine** Klasse abgebildet

Verwendung

- Wird über ein HTTP-Request angesprochen

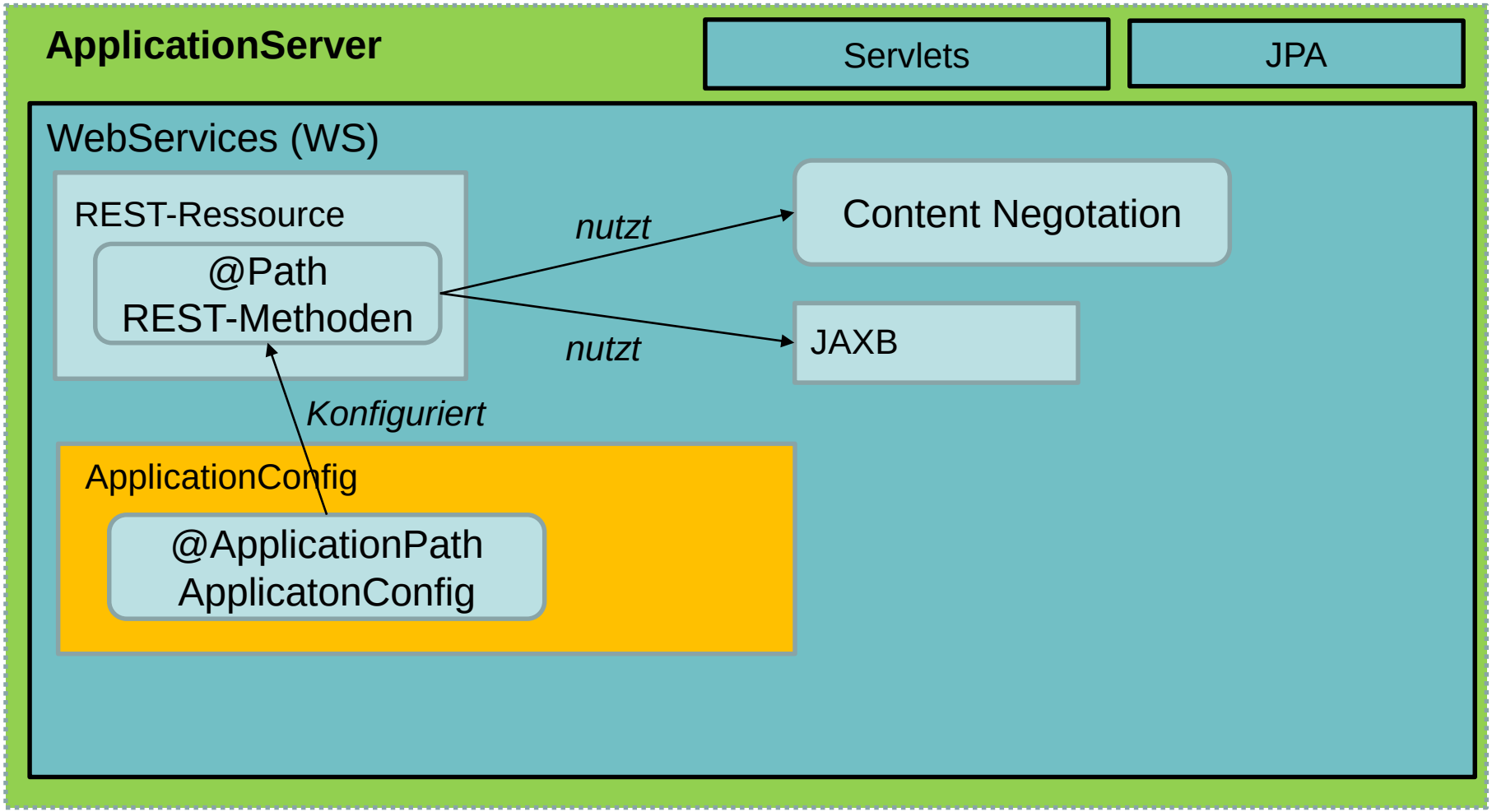
```
@Path("location")
public class LocationResource implements Serializable {

   // Implementiere hier die Methoden der Ressource

}
```

Legende: API Konzept

REST I – ApplicationServer



REST III - Konfiguration

REST WebServices werden (nach Java EE6) über eine Application-Klasse konfiguriert.

Eigenschaften

- Listet alle REST-Klassen
- Wird durch die meisten IDEs automatisch gepflegt

```
@javax.ws.rs.ApplicationPath("webresources")
public class ApplicationConfig extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> resources = new java.util.HashSet<>();
        addRestResourceClasses(resources);
        return resources;
    }

   private void addRestResourceClasses(Set<Class<?>> resources) {
        resources.add(de.fhbielefeld.rest.LocationResource.class);
    }
}
```

REST IV - Operationen - Server

Eigenschaften

- Jede Operation stellt eine Funktion zur Verwaltung der Ressource dar.
- Jede Operation wird durch ein HTTP-Verb in seiner Art festgelegt

Bedeutung	**HTTP-Verb**
Abholen / auflisten	GET
Anlegen	POST
Löschen	DELETE
Aktualisieren	PUT

Implementierung

- Die Methode einer Klasse wird mit einem HTTP-Verb annotiert

```
@GET
@Path("get")
public String get() {
    …
}
```

REST IV - Operationen - Client

Eigenschaften

- HTTP-Request an den Server
  - Angabe des Pfades, der sich aus den @Path Annotationen ergibt
  - Verwendung der zur Operation passenden HTTP-Methode
  - Mitzusendende Daten als Plain-Text, JSON, … (je nach Impl.)
- HTTP-Response vom Server
  - Statuscode (z.B. HTTP 200 OK)
  - Antwort als Plain-Text, JSON, … (je nach Implementierung)

HTTP-Request

```
GET http://www.webshop.de/article/list HTTP/1.1
```

HTTP-Response

```
HTTP/1.1 200 OK
Date: Mon, 17 Oct 2017 12:28:53 GMT
Content-Type: application/json

{“price“,16.99,“title“,“Rest-Services Tutorial“}
```

REST: GET-Parameter - Server

Eigenschaften

- schränken Operationen auf Ressourcen ein
- werden als Teil der URL angegeben
  - Variante 1: Parameter und Werte als Teil des Path
    - Name des Parameters als teil des Pfades, gefolgt von
    - Platzhalter-Variablen mit {} in der Path-Angabe für die Werte
  - Variante 2: Parameter als URL-Parameter

Implementierung

- durch Annotation der Methoden-Parameter

Variante 1:

```
@GET
@Path(“get/id/{id}/anzahl/{anz}“)
public String get(@PathParam(“id“) Integer id) {
          …
}
```

Variante 2:

```
@GET
@Path("get")
public String get(@QueryParam("id") Integer id) {
          …
}
```

REST: GET-Parameter - Client

Eigenschaften

- Übertragen nicht zu umfangreicher Daten (255 Bytes)
- werden als Teil der URL angegeben
  - Variante 1: Parameter und Werte als Teil des Path
    - Z.B.: http://meinservice.de/ressource/parmeter1/wert1/
  - Variante 2: Parameter als URL-Parameter
    - Z.B.: http://meinservice.de/ressource?parameter1=wert1
- Werte müssen kodiert werden (base64-Kodierung)

HTTP-Request

```
GET http://meinservice.de/location/get?id=1 HTTP/1.1
```

REST: GET-Parameter - Client

```
/*
•  Anfrage an den Webserver senden und asynchron auf Antwort
•  Warten.
*/
function getResource(requestdata, successHandler) {
   $.ajax({
        url: "article/get",
        type: "GET",
        dataType: "application/json; charset=utf-8",
        data: requestdata,
        success: successHandler,
        timeout: 30
   });
}

// Funktion die bei Erhalt einer Antwort ausgeführt wird
function callback(response) {
        // Antwort verarbeiten, z.B. Daten in das DOM
}
```

REST: POST-Parameter - Server

Eigenschaften

- Zum Übertragen auch umfangreicherer Daten

Implementierung

- Durch Annotation der Methode mit @POST
- Durch Angabe des MIME-Types der zu verarbeitenden Daten
- Parameter (je Eingabefeld einer) werden mit @FormParam versehen

```
@POST
@Path("create")
@Consumes("application/x-www-form-urlencoded")
public void create(
   @FormParam("name") String name,
   @FormParam(„street")String street) {
  …
}
```

REST: POST-Parameter - Client

Eigenschaften

- Verwendung eines Standard HTML Formulars
  - Form-Post-Request
  - POST mit Content-Type: application/x-www-form-urlencoded

```
<form action="/article/create" method="post">
        <input type="text" name="name">
        <input type="number" name="price">
</form>
```

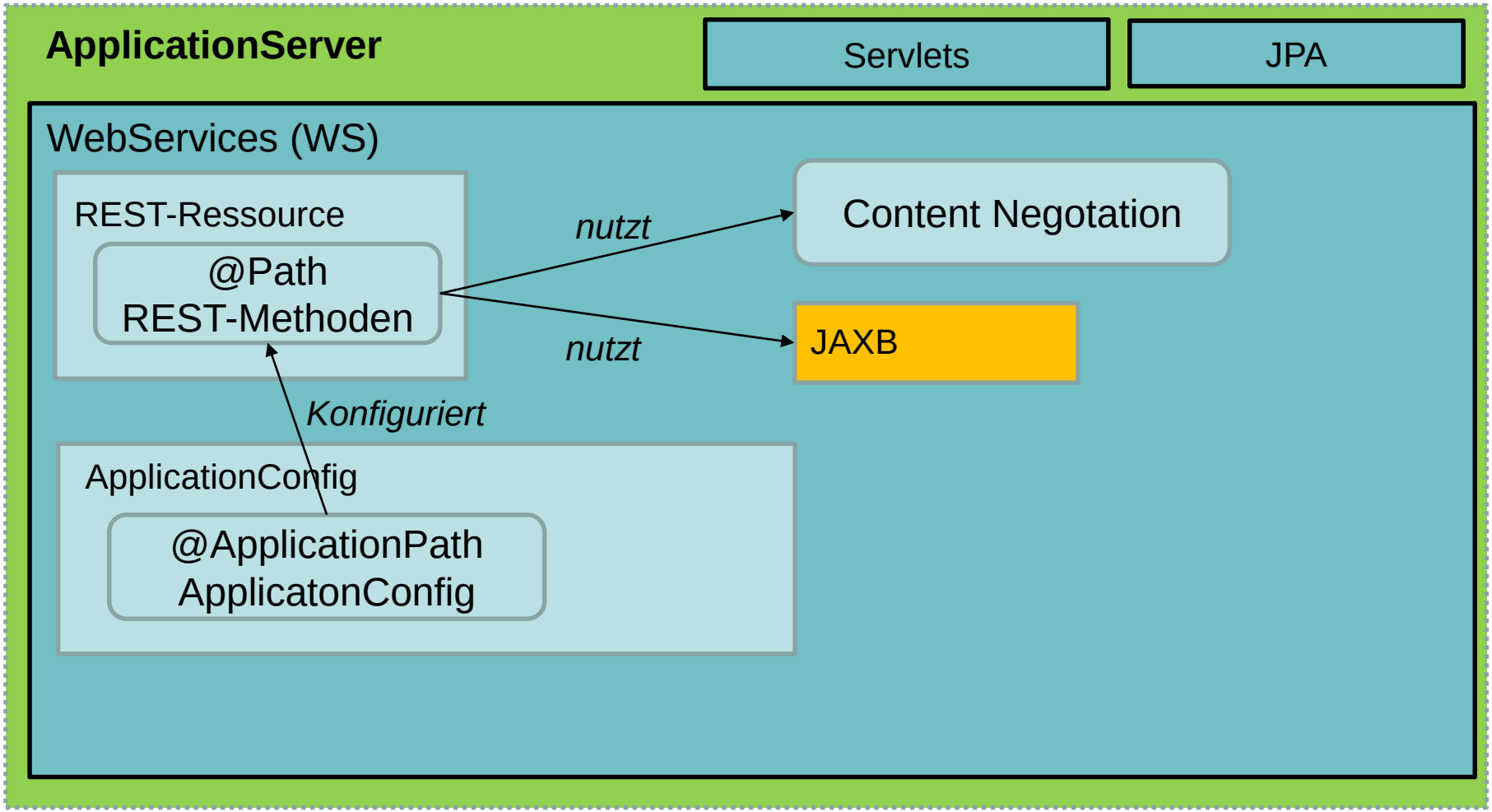
HTTP-Request

```
POST /create HTTP/1.1
Host: www.webshop.de
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
Content-Length: 25

name=WBA Tutorials&price=16.99
```

Legende: API Konzept

REST I – ApplicationServer



REST: DatenObjekte - Server

Die REST Implementierung in Java kann aus empfangenen Daten automatisch Objekte erstellen.

Eigenschaften

- Annotation von Entities ermöglicht die XML/JSON Serialisierung
- JAX-RS (Rest-Implementierung) verwendet (JAXB XML/JSON Interface)
- XML und JSON können in Objekte transformiert werden
- Angabe des MIME-Types notwendig

Entity-Klasse:

```
@XmlRootElement
public class Article {
    …
}
```

REST-Methode

```
@POST
@Path("create")
@Consumes(MediaType.APPLICATION_JSON)
public void create(Article article) {
   …
}
```

REST: DatenObjekte - Client

Eigenschaften

- HTTP-POST Anfrage per Ajax
- Verpacken aller Daten des Objekts in JSON oder XML

```
function createArticle(articledata,successHandler) {

 $.ajax({
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json; charset=utf-8,
        },
        type: "POST",
        url: "/article/create",
        data: JSON.stringify(articledata),
        success: successHandler,
        dataType: "json",
        timeout: 30,
});
```

REST: Rückgabewerte - Server

Eigenschaften

- Bestehen aus einer kompletten HTTP Response (inkl. Status)
- Besitzen mindestens einen MIME-Typ
- Eine Methode kann Daten in verschiedenen MIME-Typen zurückgeben.

Implementierung

- Rückgabe eines Response-Objektes
- Durch Annotation der Methode mit @Produces
- Einbinden einer Data-Binding Bibliothek (z.B. Genson)

```
@GET
@Path("get")
@Produces(MediaType.APPLICATION_JSON)
public Response get() {
   Response.Status status = Response.Status.OK;
   ResponseBuilder rb = Response.status(status);
   // Schreibe Content in die Response
   rb.entity("{"id":1,"name":"WBA-Tutorial"}");
   return rb.build();
}
```

REST: Rückgabewerte - Client

Eigenschaften

- Client erhält die Antwort mit einem normalen HTTP-Response
- Client kann schon am Rückgabestatus den Erfolg seiner Anfrage erkennen. (z.B. HTTP 200 OK, HTTP 404 Not Found)

Implementierung

- Asynchrone Antwort fangen und Status prüfen
- Eventuelle allgemeine Fehlermeldungen generieren
- Empfange Daten verarbeiten

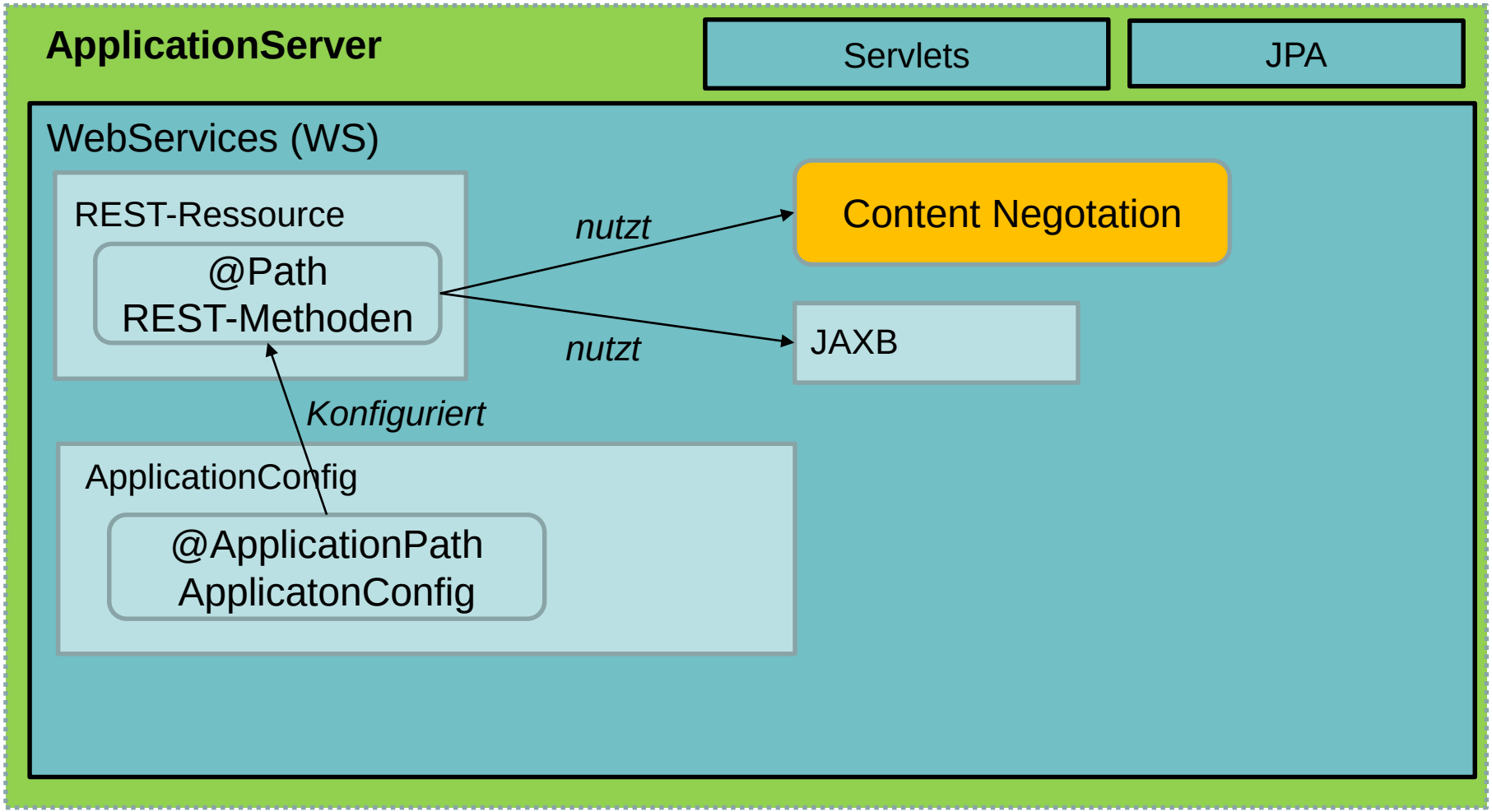
HTTP-Response

```
GET http://webshop.de/article/get?id=1 HTTP/1.1
Accept: application/json
```

```
// Funktion die bei Erhalt einer Antwort ausgeführt wird
function callback(response) {
        console.log(response.id)  // Gibt 1 aus
        console.log(response.name)         // Gibt den Namen aus
}
```

Legende: API Konzept

REST I – ApplicationServer



Content Negotiation - Server

Eigenschaften

- Eine Technik des HTTP
- Ermöglicht es dem Client, dem Server zu sagen, in welchem Typ die angeforderten Daten geliefert werden sollen
- Im REST Umfeld hauptsächlich mit XML und JSON verwendet
- Nutzt in der JAX-RS Spezifikation JAXB und ermöglicht so automatisches Generieren von JSON/XML aus JavaObjekten

Implementierung

- Annotation des Entities mit @XmlRootElement (auch für JSON)
- Annotation der REST Methode mit @Produces

```
@GET
@Path("get")
@Produces({MediaType.TEXT_XML,MediaType.APPLICATION_JSON})
public Response get() {
        Article article = new Article(“WBA-Tutorial“);
        return Response.ok(article).build();
}
```

Content Negotiation - Client

Eigenschaften

- Anforderung des gewünschten Content-Types über den Header des HTTP-Reqests

Implementierung

- Header-Definition im Ajax-Request (bei Verwendung von jQuery)

```
$.ajax({
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json; charset=utf-8'
        },
        type: "GET",
        url: "webshob/article",
          data: JSON.stringify(data),
        success: successHandler
          }
    });
```

HTTP-Request

```
GET http://meinservice.de/get HTTP/1.1
Accept: application/json
```

REST: Architekturstil

REST definiert als Architektur-Stil folgende Prinzipien

1. Ressourcen basiert
   - Schnittstellen dienen immer für genau eine Ressource
2. Verwendung von Webstandards
   - Verwendung von HTTP, URLs, MIME, XML, Json, …
3. Client-Server-Architektur
   - Ein Client fordert Informationen / Aktionen vom Server
4. Zustandslos
   - Weder Server noch Client merken sich Zustände
   - Alle benötigten Daten werden mit einer Anfrage gesendet
5. Caching
   - Einmal gelieferte Daten werden zwischengespeichert
6. Ressourcen verweisen aufeinander (HATEOAS)
   - Links für mögliche Folgeaktionen werden angegeben

REST Prinzipien I

REST-Prinzip: Client-Server-Architektur

Dieses Prinzip ist die Verwendung des allgemein im Webumfeld verwendeten Client-Server-Modells für REST

Eigenschaften

- Der Server verwaltet alle Ressourcen
- Der Server stellt Methoden zur Verfügung
- Der Client nutzt die Methoden des Servers

Implementierung

- Der Client wird nicht vom Server aus angefragt
- Der Server agiert nur auf Anfrage von einem Client

REST Prinzipien II

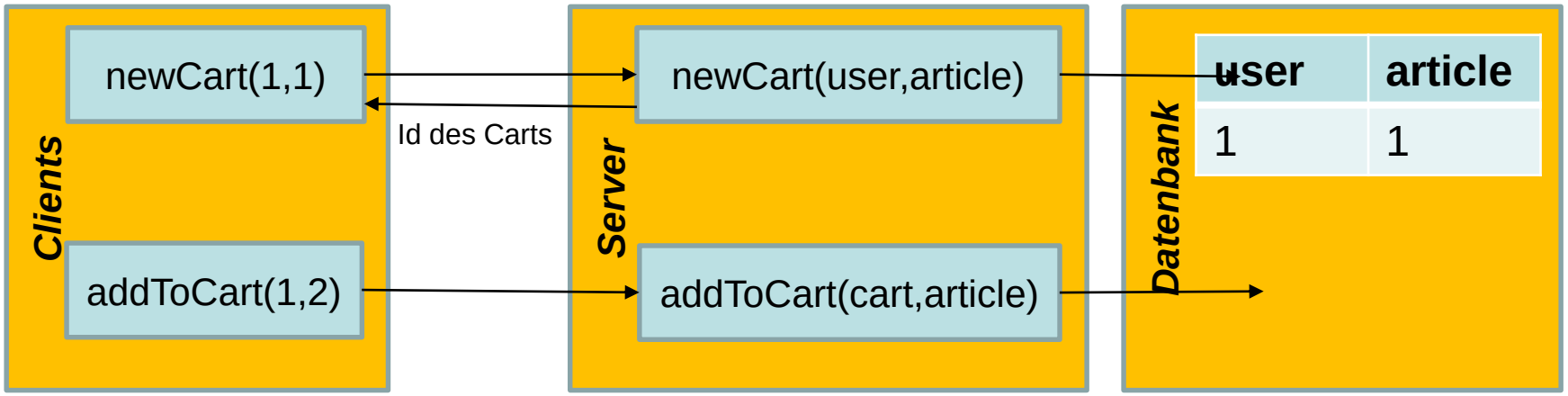
REST-Prinzip: Zustandslos

Eigenschaften

- Weder Server noch Client merken sich in den REST-Klassen Daten über mehrere Anfragen hinweg.

Implementierung

- Keine Verwendung von statischen Klassen
- Keine Verwendung von Objekt-Eigenschaften die auf eine Anfrage bezogen sind
- Aber: Der Server und Client können sich Daten z.B. in einer Datenbank merken.



REST Prinzipien III

REST-Prinzip: Caching

Eigenschaften

- REST-Methode verwendet den Cache des Servers
- Client **kann** einmal abgerufene Daten zwischenspeichern

Implementierung

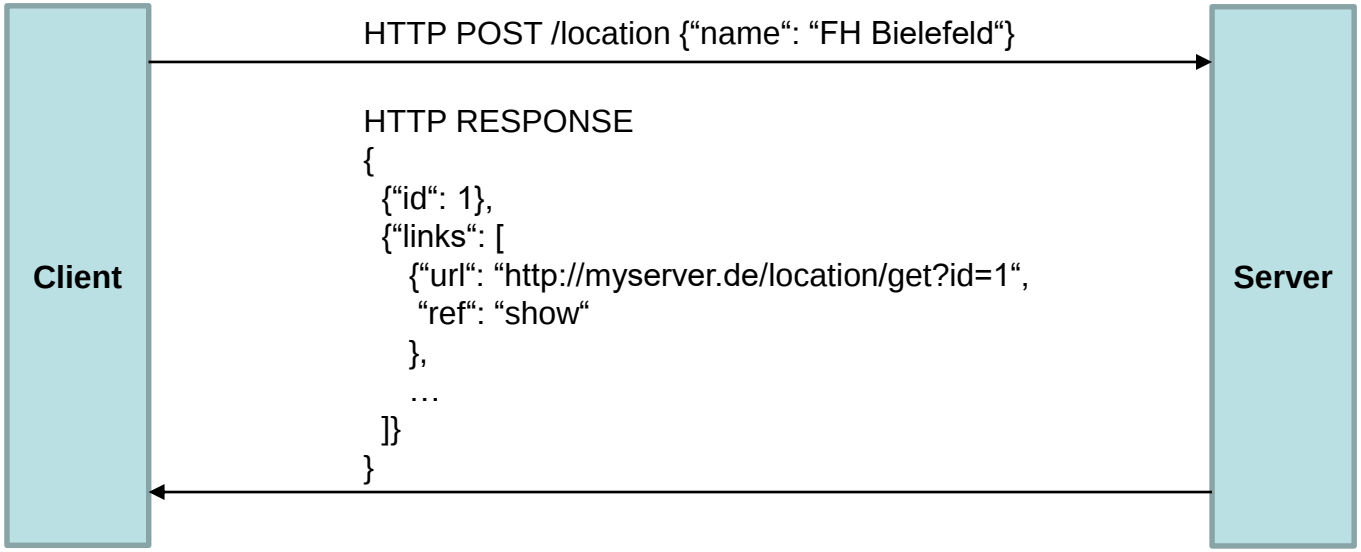
- Server-Caching meistens per Standard aktiviert
- Client-Caching (Browser-Cache) ebenso
- Client-Erweiterung mit HTML5-WebStorage Technologie möglich

REST HATEOAS I

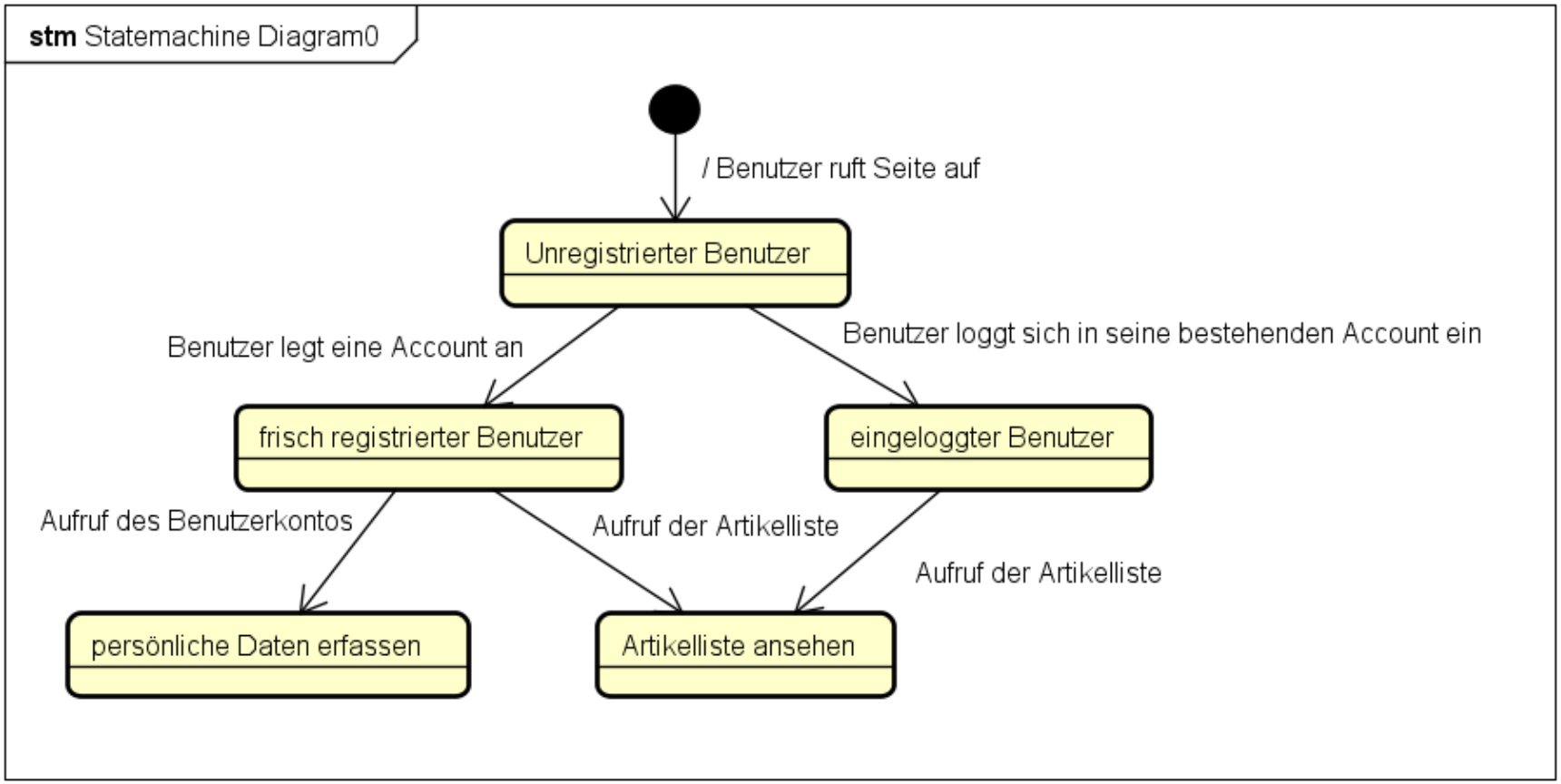
Hypermedia as the Engine of Application State ist eine der wichtigsten Forderungen von Rest und die am häufigsten missachtete.

Eigenschaften

- WebAnwendungen werden als State-Maschine konzipiert
- Jede Anfrage wird vom Server mit dem aktuellen Status und einer Liste von möglichen Statusübergängen beantwortet
- Statusübergänge werden in der Form von URLs repräsentiert



REST HATEOAS II



REST HATEOAS - Server

Implementierung

- Erzeugen einer URI zu einer REST-Operation
- Hinzufügen der URI zur Response der REST-Methode

```
@GET
@Path("get")
public Response get() {
   // Baue einen ResponseBuilder mit Header „HTTP 200 OK“
   Article art = new Article(“WBA-Tutorial“);
   ResponseBuilder rb = Response.ok(art).build();

   // Füge einen Link für einen möglichen Statusübergang ein
   URI delLocLink = URI.create(“/article/delete?id=“+art.id)
   rb.link(delLocLink);
   return rb.build();
}
```

REST HATEOAS - Client

Implementierung

- URI‘s werden aus der Response entnommen
- Client baut aus den URI‘s Anzeigeelemente und bietet sie dem Nutzer an

```
// Funktion die bei Erhalt einer Antwort ausgeführt wird
function callback(xhr) {
   // Header auslesen
   var r = parseLinkHeader(xhr.getResponseHeader('Link'));
   // Zugriff auf einen Link der mit „delete“ bezeichnet wurde
   var deletelink = r['delete']['href'];
   // Nun kann der Link irgendwo eingebaut werden
}
```

WebServices

1. Kontext und Motivation
2. SOAP WebServices
3. REST WebServices
4. **Darüber hinaus**
5. Projekt

JSON und Java

Definition: Es gibt verschiedene Implementierungen von JSON Interpretern für Java. Im SCL verwenden wir meistens JSONsimple. (https://github.com/fangyidong/json-simple)

Eigenschaften:

- Komplett Kompatibel zur JSON Spezifikation
- Einfach zu verwenden
- Verwendet die bekannten Map und List-Interfaces

JSON schreiben

Möglichkeit 1: String selber zusammenbauen

```
String ret = "{\n";
ret = ret + "\"status\": \"" + "ok" + "\",\n";
ret = ret + "\"fehlermeldung\": \"" + "" + "\"";
ret = ret + "\n}";
```

Möglichkeit 2: JSONsimple nutzen

```
JSONObject obj = new JSONObject();
obj.put("status", "ok");
obj.put("fehlermeldung", "");
return obj.toJSONString();
```

JSON lesen

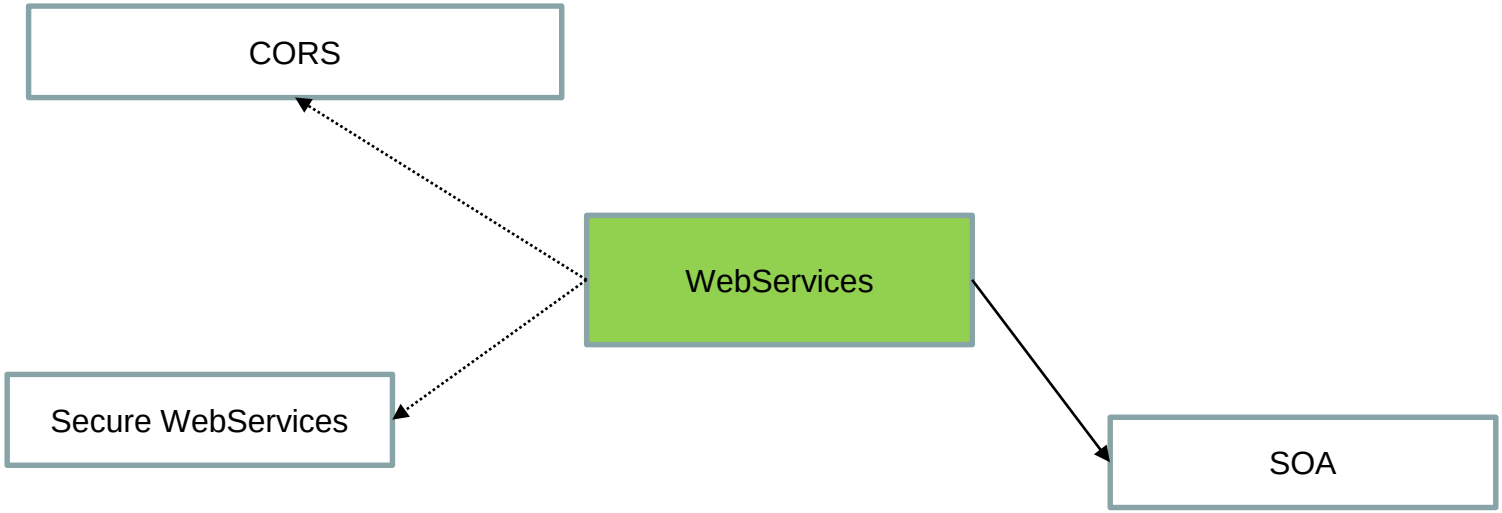
Möglichkeit 1: JSONsimple

```
JSONParser parser = new JSONParser();
JSONObject obj = (JSONObject) parser.parse(userjson);
String status = (String) obj.get("status");
String fehlermeldung = (String) obj.get(„fehlermeldung");
```

Für eventuelle Arrays im Objekt:

```
JSONArray msg = (JSONArray) obj.get("messages");
```

Darüber hinaus



Links:

CORS - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
SOA - http://www.torsten-horn.de/techdocs/soa.htm
Secure WebServices - https://www.javaworld.com/article/2073287/soa/secure-web-services.html

WebServices

1. Kontext und Motivation
2. SOAP WebServices
3. REST WebServices
4. Darüber hinaus
5. **Projekt**

Anforderungen

Welche Anforderungen werden als nächstes bearbeitet?

TODO

- Artikel zum Server übertragen
- Kommentare zum Server
- Medien zum Server
- Kommentare speichern
- Kommunikation untereinander

DONE

- Technologische Grundlagen erarbeiten
- Was ist eine Web-Anwendung?
- News darstellen
- Projekte vorstellen
- Aufgaben darstellen
- Formular für Kommentare
- Schickes Design für die Seite
- Mediendatein einbinden
- Animationen
- Mehrsprachen-Fähigkeit
- (lokales) Speichern von Artikeln
- Client-Position anzeigen
- Offline-Verwendung ermöglichen
- Inhaltsverzeichnisse
- Formlareingaben in Seite einfügen
- Navigation über Tastaturkürzel
- Externe Inhalte einbinden
- Artikel vom Server einbinden
- Kommentare vom Server

Literatur: Web Services und JSON

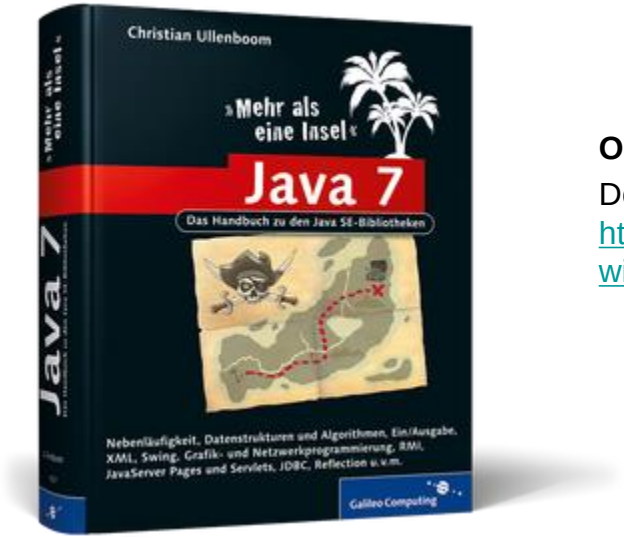
Melzer, Ingo et al. „Service-orientierte Architekturen mit Web Services“ Konzepte – Standards – Praxis 4. Auflage 2010, 381 Seiten, ISBN 978-3-8274-2549-2, Spektrum Akademischer Verlag über Springer Link

Christian Ullenboom: „Java 7 – Mehr als eine Insel

Das Handbuch zu den Java SE-Bibliotheken“

ISBN 978-3-8362-1507-7,

Rheinwerk Verlag 2012



Online-Quellen:

Dokumentation zu Jquery: https://learn.jquery.com/ajax/working-with-jsonp/