



# **REST in der Praxis**

**Entwerfen von RESTful Applikationen**



# Agenda

## **Einordnung**

Anatomie einer HTTP Interaktion

HTTP Caching

Adressierung von Ressourcen

Die Operationen auf eine Ressource (Uniform Interface)

Modellierung von Ressourcen

JAX-RS

Implementierung von Non-CRUD Operation

Stateful vs. Stateless

Zusammenfassung



## Über mich ...

- **Gregor Roth, Dipl.-Ing. Nachrichtentechnik (FH), Dipl.-Wirtsch.-Ing. (FH)**
- **Langjährige Erfahrung im Bau großer, verteilter (Java) basierter Anwendungssysteme**
  - im Finanzdienstleistungsbereich (ca. 10 Jahre)
  - im Bereich InternetProvider/WebPortale (ca. 5 Jahre)
- **Software-Architekt bei der 1&1, dort im Bereich der Portal-Mailsysteme**
- **Maintainer der OpenSource Java Netzwerkbibliotheken**
  - *xSocket* (NIO socket lib) und
  - *xLightweb* (HTTP client/server lib)

# REST

- **REST (Representational State Transfer) ist kein neues Netzwerkprotokoll oder ähnliches, sondern ein Architekturstil**
  - Definiert in der Dissertation von Roy Fielding, einer der Hauptautoren der HTTP und URI Spezifikation
  - REST erfindet nichts neues. Es beschreibt lediglich wie HTTP „richtig“ anzuwenden ist.
- **REST legt eine Reihe von Prinzipien fest:**
  - ***Resources and Resource Identifiers***: Die Schlüsselabstraktion einer Information ist die Ressource. RESTful HTTP adressiert eine Ressource mittels einer URI
  - ***Representation***: Ein Client interagiert mit einer Ressource grundsätzlich über eine Repräsentation der Ressource
  - ***Hypermedia***: Die Beziehungen von Ressourcen untereinander werden als Hyperlinks abgebildet. Somit kann von einer Resource zu einer anderen navigiert werden.
  - ***Unified Interface***: Operationen auf eine Ressource finden anhand einer begrenzten Menge von wohldefinierten Methoden statt (RESTful HTTP: GET, DELETE, PUT,...)
  - ***Statelessness***: Ein REST Interaktion ist zustandslos



Einordnung

**Anatomie einer HTTP Interaktion**

HTTP Caching

Adressierung von Ressourcen

Die Operationen auf eine Ressource (Uniform Interface)

Modellierung von Ressourcen

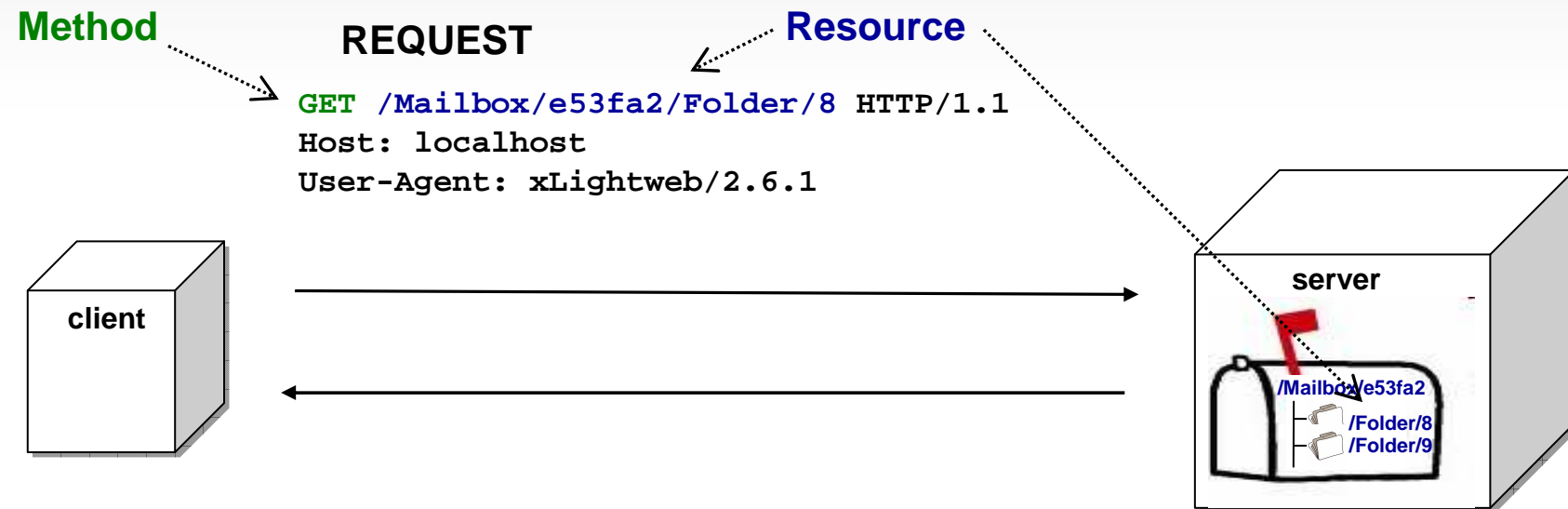
JAX-RS

Implementierung von Non-CRUD Operation

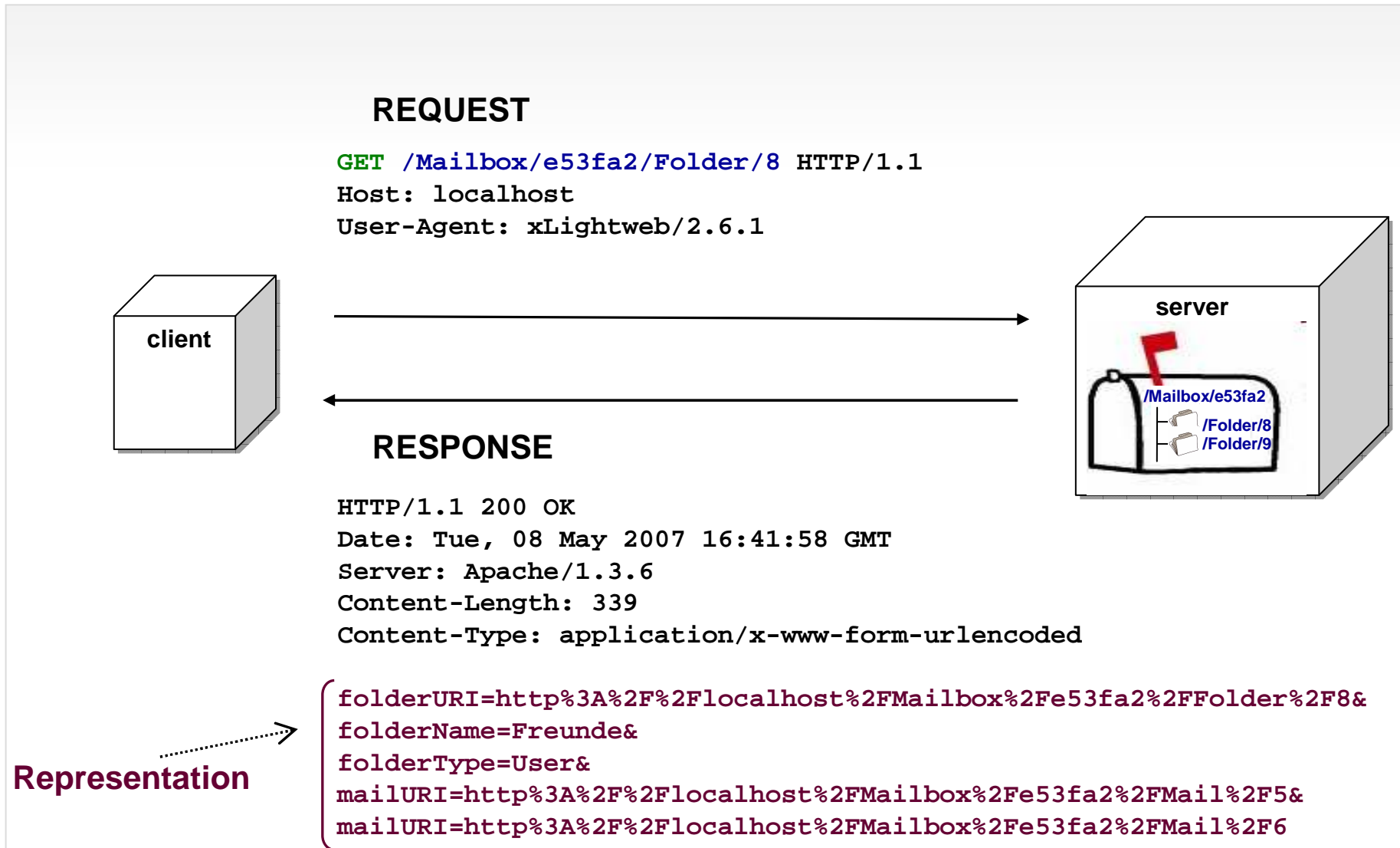
Stateful vs. Stateless

Zusammenfassung

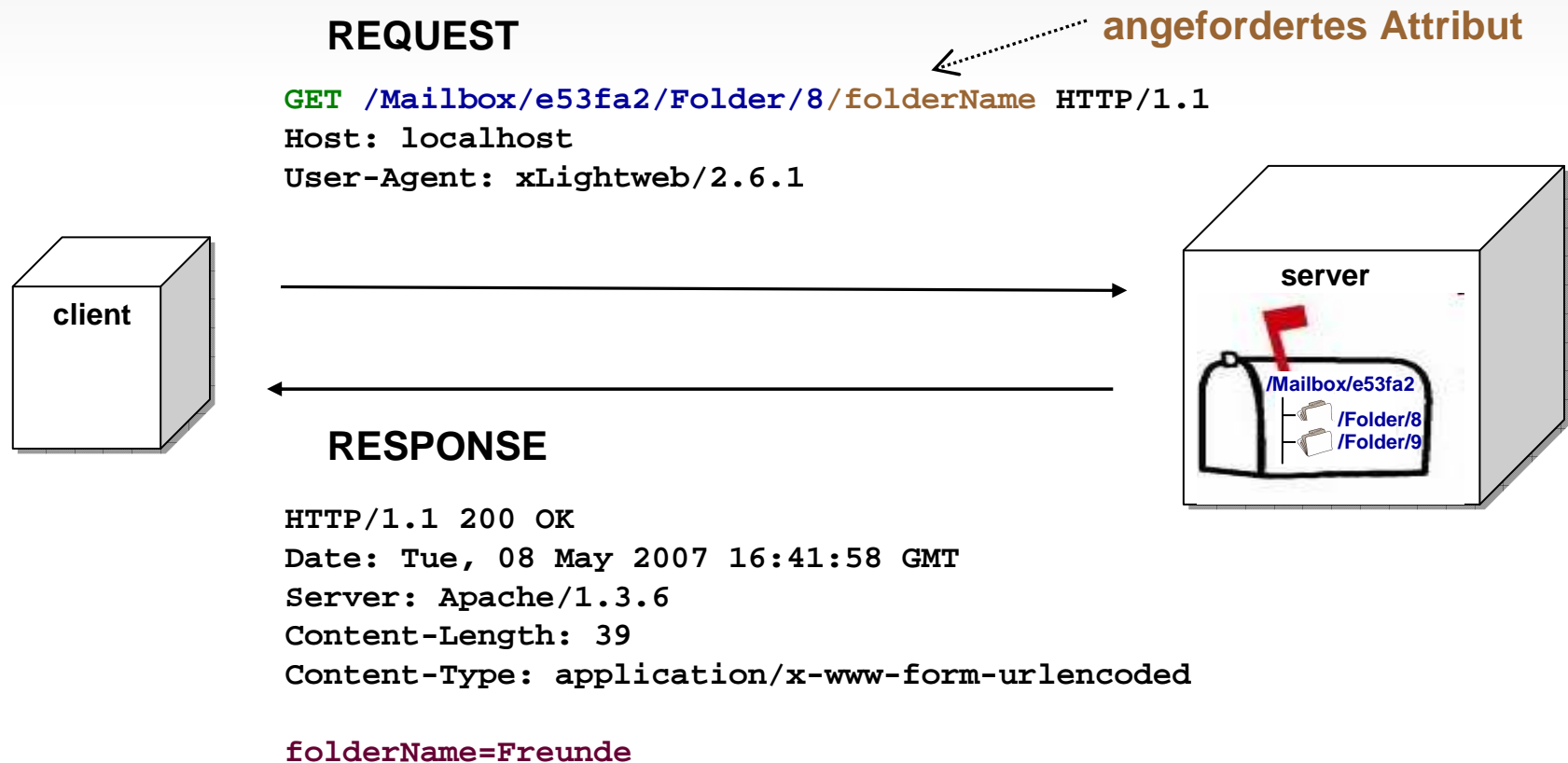
# Anatomie einer HTTP Interaktion



# Anatomie einer HTTP Interaktion

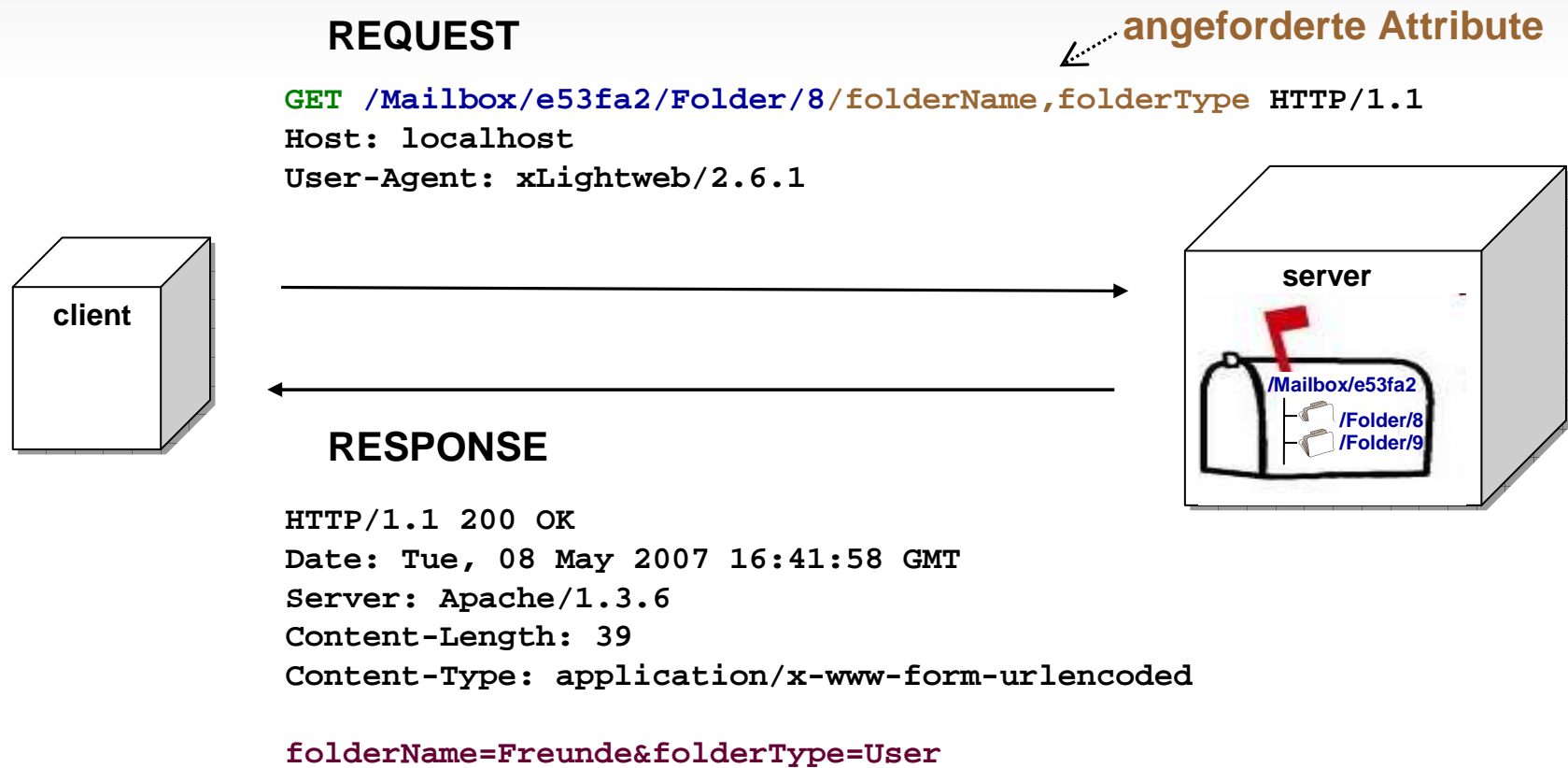


# Representation - Beispiel für das Filtern eines Einzel-attributes

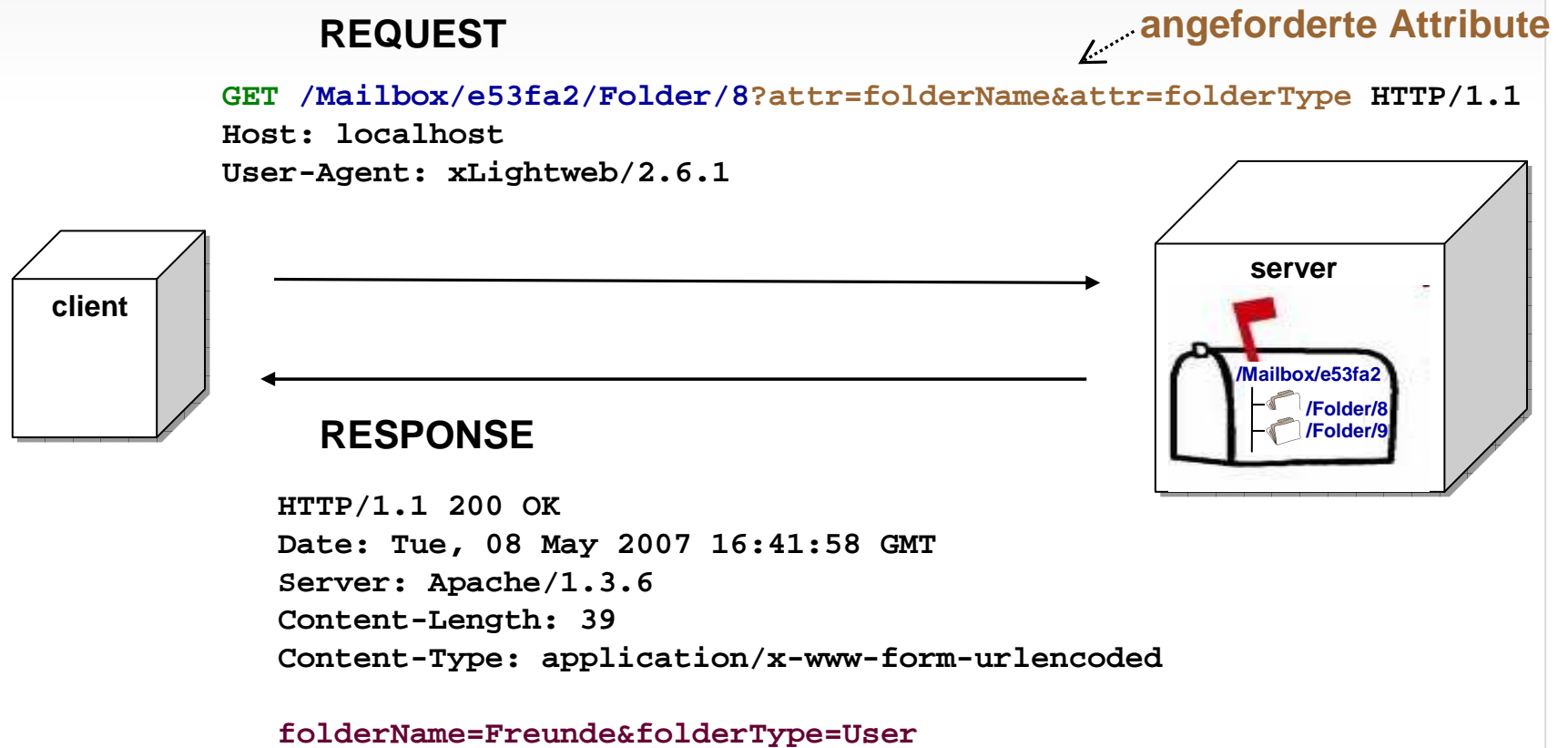




# Representation - Beispiel für das Filtern mehrerer Attributen



# Representation - Beispiel für das Filtern mehrerer Attributen auf Basis eines Query

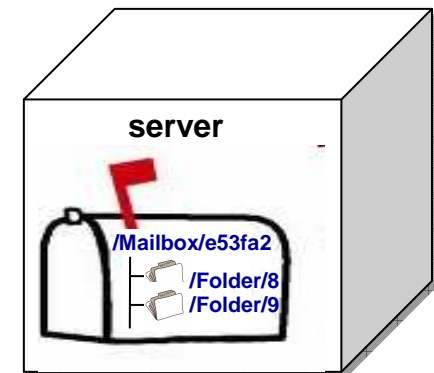
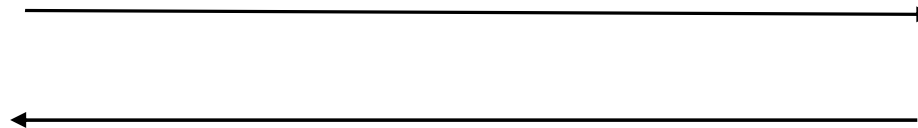
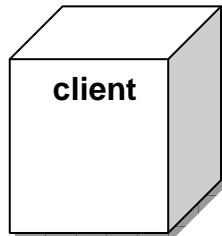


# Representation – Encoding (form-urlencoded)

Required  
Encoding

## REQUEST

```
GET /Mailbox/e53fa2/Folder/8 HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
Accept: application/x-www-form-urlencoded
```



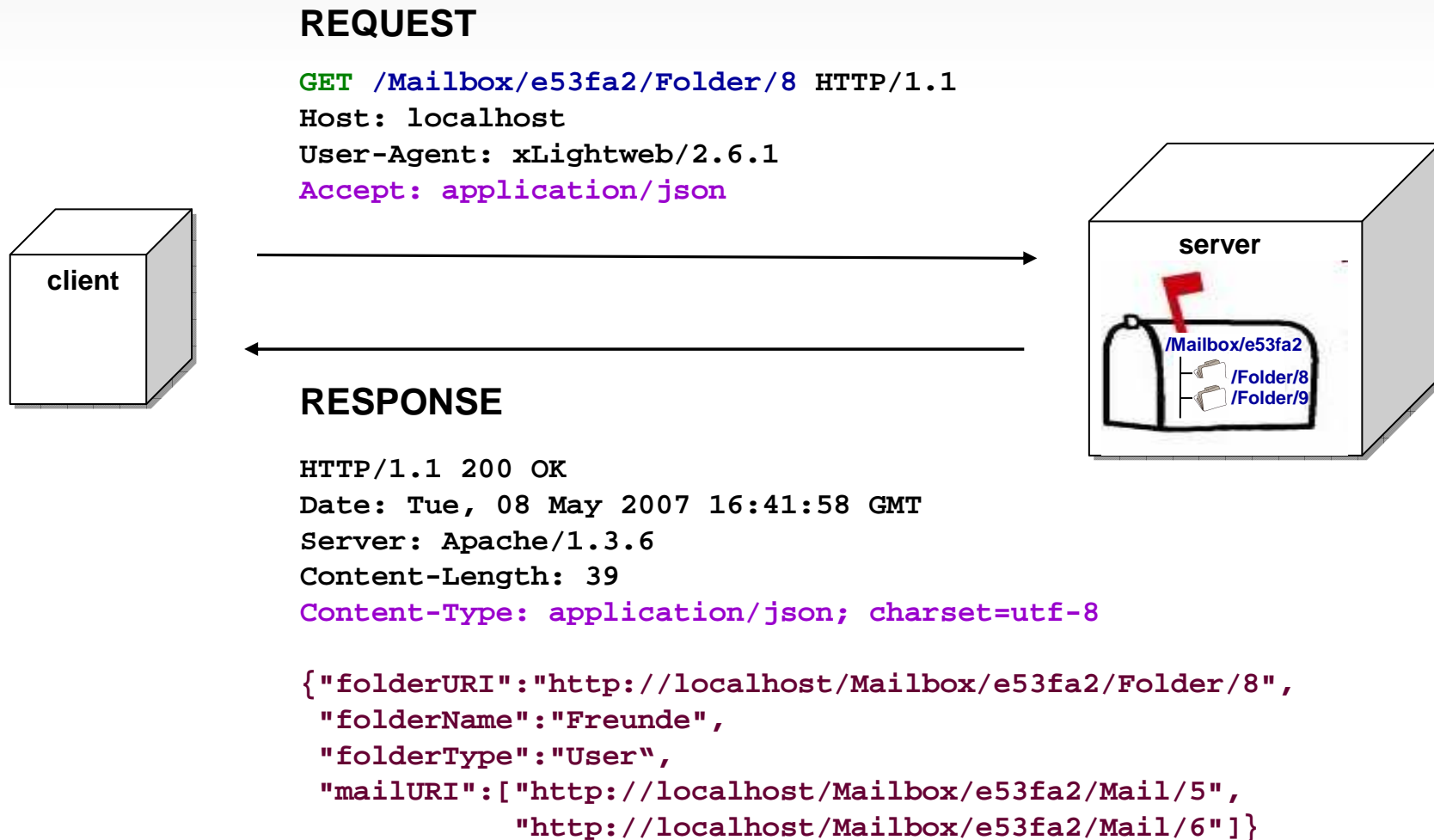
## RESPONSE

```
HTTP/1.1 200 OK
Date: Tue, 08 May 2007 16:41:58 GMT
Server: Apache/1.3.6
Content-Length: 39
Content-Type: application/x-www-form-urlencoded
```

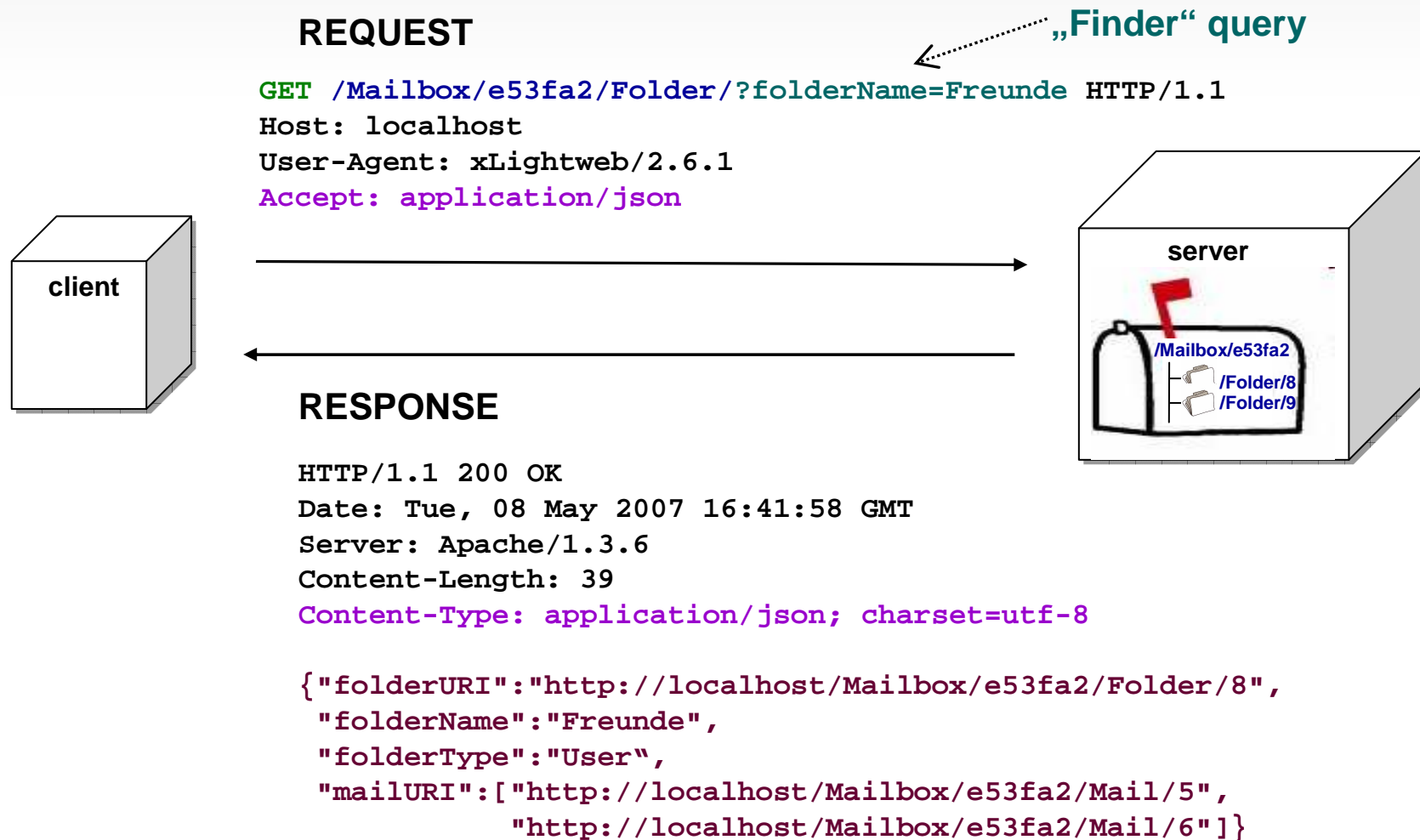
Representation  
Metadata

```
mailURI=http%3A%2F%2Flocalhost%2FMailbox%2Fe53fa2%2FMail%2F5&
mailURI=http%3A%2F%2Flocalhost%2FMailbox%2Fe53fa2%2FMail%2F6&
folderName=Freunde&folderType=User
```

# Representation – Encoding (JSON)



# Representation – Beispiel für die Selektion einer Ressourcenrepräsentation auf Basis eines Query





Einordnung

Anatomie einer HTTP Interaktion

**HTTP Caching**

Adressierung von Ressourcen

Die Operationen auf eine Ressource (Uniform Interface)

Modellierung von Ressourcen

JAX-RS

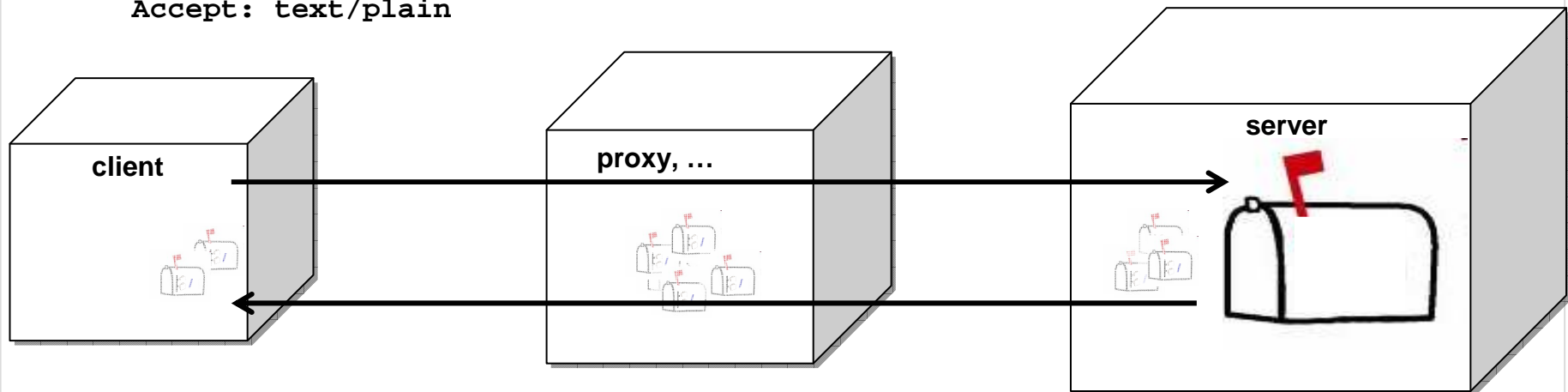
Implementierung von Non-CRUD Operation

Stateful vs. Stateless

Zusammenfassung

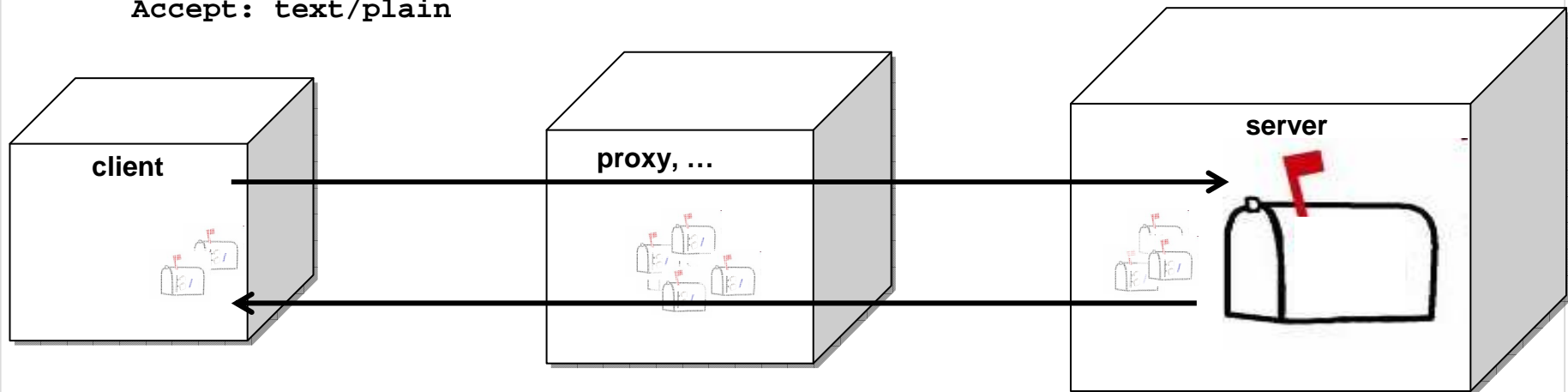
## Caching (Expired-based)

```
GET /Mailbox/e443/creationDate HTTP/1.1  
Host: localhost  
User-Agent: xLightweb/2.6.1  
Accept: text/plain
```



# Caching (Expired-based)

```
GET /Mailbox/e443/creationDate HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
Accept: text/plain
```



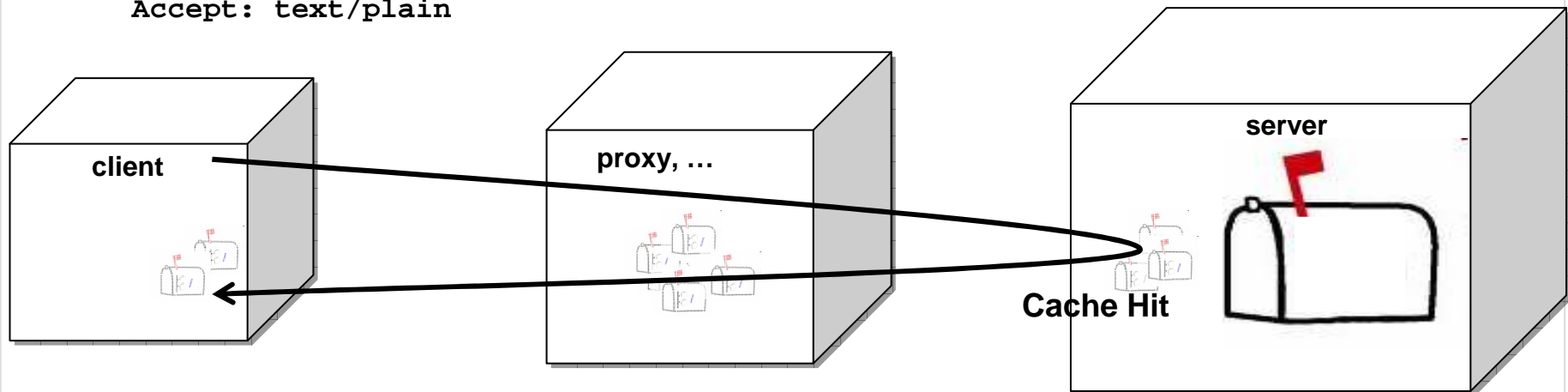
```
HTTP/1.1 200 OK
Server: Apache/1.3.6
Content-Length: 76
Cache-Control: public, max-age=84400
Content-Type: text/plain

2009-05-15T06:11:10.658
```



# Server-side Caching

```
GET /Mailbox/e443/creationDate HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
Accept: text/plain
```

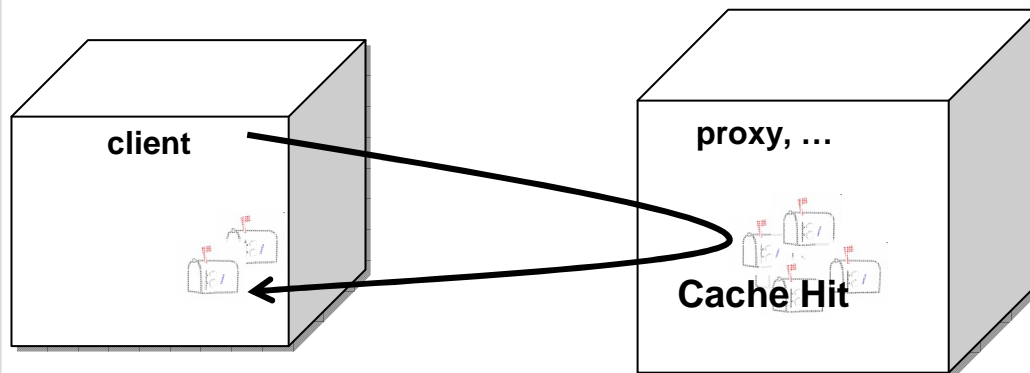


```
HTTP/1.1 200 OK
Server: Apache/1.3.6
Content-Length: 76
Cache-Control: public, max-age=84277
Content-Type: text/plain
X-Cache: HIT from myserver

2009-05-15T06:11:10.658
```

# Intermediary-based caching

```
GET /Mailbox/e443/creationDate HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
Accept: text/plain
```

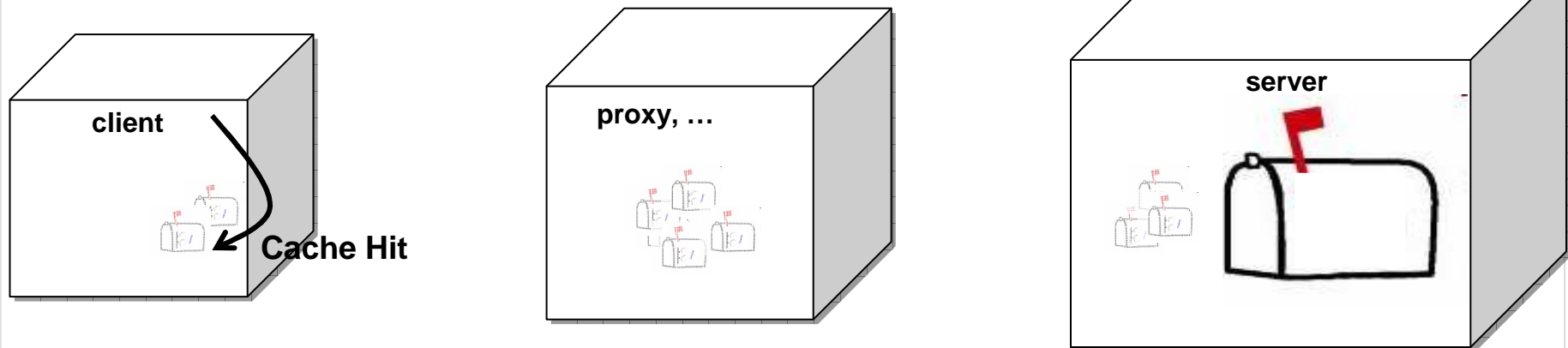


```
HTTP/1.1 200 OK
Server: Apache/1.3.6
Content-Length: 76
Cache-Control: public, max-age=84273
Content-Type: text/plain
X-Cache: HIT from myproxy

2009-05-15T06:11:10.658
```

# Client-side caching

```
GET /Mailbox/e443/creationDate HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
Accept: text/plain
```



```
HTTP/1.1 200 OK
Server: Apache/1.3.6
Content-Length: 76
Cache-Control: public, max-age=84101
Content-Type: text/plain
X-Cache: HIT from myclient

2009-05-15T06:11:10.658
```



Einordnung

Anatomie einer HTTP Interaktion

HTTP Caching

**Adressierung von Ressourcen**

Die Operationen auf eine Ressource (Uniform Interface)

Modellierung von Ressourcen

JAX-RS

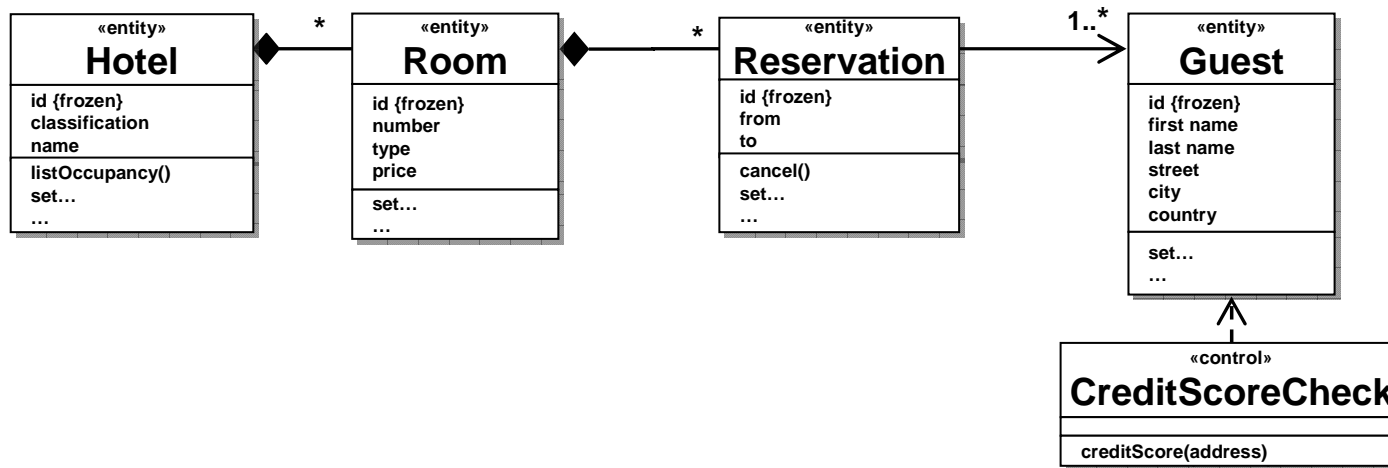
Implementierung von Non-CRUD Operation

Stateful vs. Stateless

Zusammenfassung

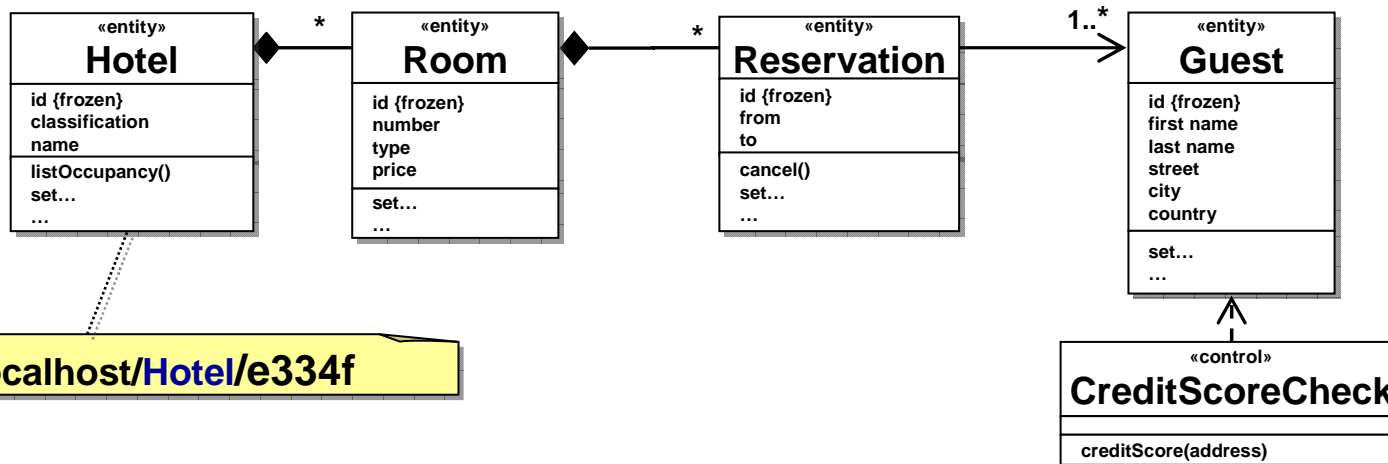
# Addressierung Ressourcen

- Ressourcen werden über deren URI(s) identifiziert.
- REST macht keinerlei Vorgaben bzgl. dem Aufbau von URIs. In der Praxis werden jedoch oft bestimmte Patterns verwendet.



## „normale“ Ressource

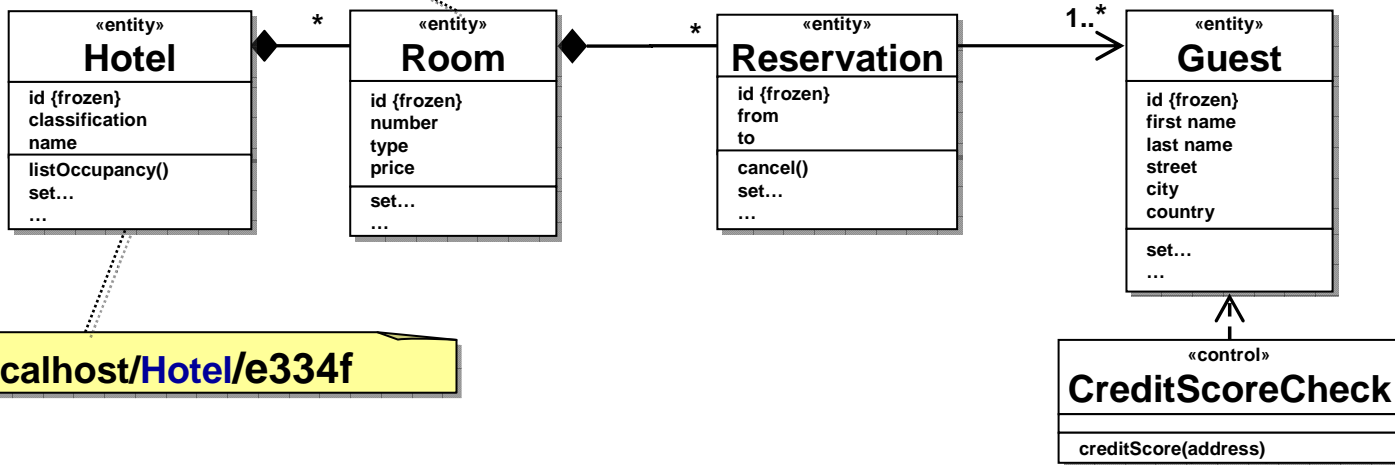
- Ressourcen werden über deren URI(s) identifiziert.
- REST macht keinerlei Vorgaben bzgl. dem Aufbau von URIs. In der Praxis werden jedoch oft bestimmte Patterns verwendet.



# Subressourcen

- Ressourcen werden über deren URI(s) identifiziert.
- REST macht keinerlei Vorgaben bzgl. dem Aufbau von URIs. In der Praxis werden jedoch oft bestimmte Patterns verwendet.

<http://localhost/Hotel/e334f/Room/5>



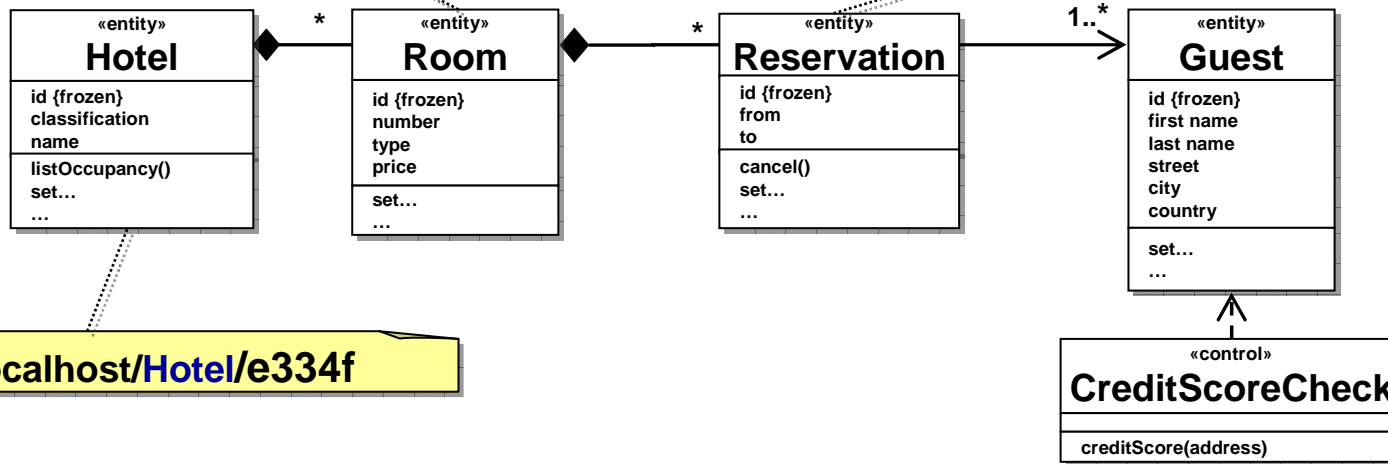
<http://localhost/Hotel/e334f>

# Adressierung von Objekt-Snapshots

- Ressourcen werden über deren URI(s) identifiziert.
- REST macht keinerlei Vorgaben bzgl. dem Aufbau von URIs. In der Praxis werden jedoch oft bestimmte Patterns verwendet.

<http://localhost/Hotel/e334f/Room/5>

<http://localhost/Hotel/e334f/Reservation/2v3>



<http://localhost/Hotel/e334f>

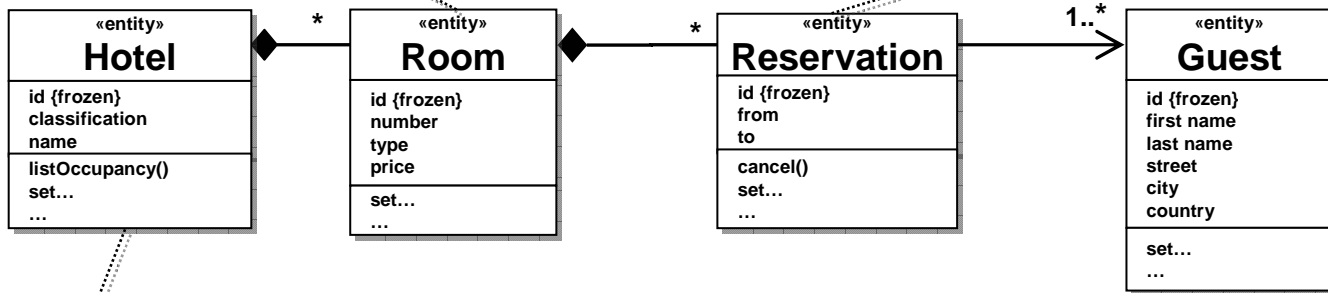


# „Schlechte“ URIs

- Ressourcen werden über deren URI(s) identifiziert.
- REST macht keinerlei Vorgaben bzgl. dem Aufbau von URIs. In der Praxis werden jedoch oft bestimmte Patterns verwendet.

<http://localhost/Hotel/e334f/Room/5>

<http://localhost/Hotel/e334f/Reservation/2v3>



<http://localhost/Hotel/e334f>



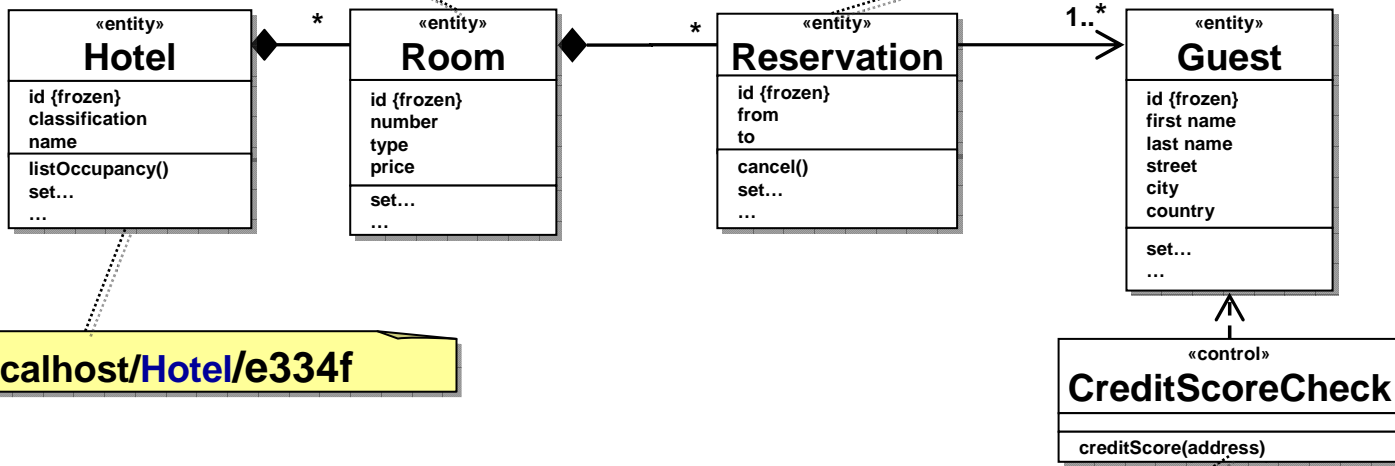
<http://localhost/CreditScoreService/check>

## „Schlechte“ URIs (II)

- Ressourcen werden über deren URI(s) identifiziert.
- REST macht keinerlei Vorgaben bzgl. dem Aufbau von URIs. In der Praxis werden jedoch oft bestimmte Patterns verwendet.

<http://localhost/Hotel/e334f/Room/5>

<http://localhost/Hotel/e334f/Reservation/2v3>



<http://localhost/Hotel/e334f>

~~<http://localhost/CreditScoreService/check>~~



Einordnung

Anatomie einer HTTP Interaktion

HTTP Caching

Adressierung von Ressourcen

**Die Operationen auf eine Ressource (Uniform Interface)**

Modellierung von Ressourcen

JAX-RS

Implementierung von Non-CRUD Operation

Stateful vs. Stateless

Zusammenfassung

# RESTful HTTP - Uniform Interface

{path=.../<name>}
<b>&lt;name&gt;Resource</b>
... state ...
«http method» <u>GET</u>
«http method» <u>PUT</u>
«http method» <u>POST</u>
«http method» <u>DELETE</u>

Method	Usage	Safe	Idem-potent
<b>GET</b>	<ul style="list-style-type: none"> <li>- retrieve a representation</li> <li>- retrieve a representation if modified (cache validation request)</li> </ul>	<b>Yes</b>	<b>Yes</b>
<b>DELETE</b>	<ul style="list-style-type: none"> <li>- delete the resource</li> </ul>	<b>No</b>	<b>Yes</b>
<b>PUT</b>	<ul style="list-style-type: none"> <li>- create a resource with client-side managed identifier</li> <li>- update a resource by replacing</li> <li>- update a resource by replacing if not modified (optimistic locking)</li> </ul>	<b>No</b>	<b>Yes</b>
<b>POST</b>	<ul style="list-style-type: none"> <li>- create a resource with server-side managed (auto generated) identifier</li> <li>- create a sub-resource</li> <li>- partial update of a resource</li> <li>- partial update a resource if not modified (optimistic locking)</li> </ul>	<b>No</b>	<b>No</b>

# Safety

- **Safety**

- Ist definiert im HTTP RFC
- Eine **Safe** Methode führt **nie zu einer Zustandsänderung einer Ressource**
- Aus Sicht des Zustandes einer Ressource hat die Ausführung einer Safe Methode den gleichen Effekt, als ob diese nie ausgeführt worden wäre
- GET ist Safe
- User-agents (Browser, HttpClient) und Intermediaries (Proxies, ...) kennen das Vokabular von HTTP und deren Bedeutung!
- Tritt beispielsweise ein Fehler beim GET-Aufruf auf, so kann ein HttpClient oder ein Proxy automatisch den Aufruf wiederholen ohne das irgendwelche Seiteneffekte auftreten. Dies setzt jedoch voraus, dass die Methode gemäß HTTP implementiert ist

```
GET /Hotel/e53fa2 HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
```

# Safety

- **Safety**

- Ist definiert im HTTP RFC
- Eine **Safe** Methode führt **nie zu einer Zustandsänderung einer Ressource**
- Aus Sicht des Zustandes einer Ressource hat die Ausführung einer Safe Methode den gleichen Effekt, als ob diese nie ausgeführt worden wäre
- GET ist Safe
- User-agents (Browser, HttpClient) und Intermediaries (Proxies, ...) kennen das Vokabular von HTTP und deren Bedeutung!
- Tritt beispielsweise ein Fehler beim GET-Aufruf auf, so kann ein HttpClient oder ein Proxy automatisch den Aufruf wiederholen ohne das irgendwelche Seiteneffekte auftreten. Dies setzt jedoch voraus, dass die Methode gemäß HTTP implementiert ist

```
GET /Hotel/e53fa2 HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
```

```
GET /Hotel/e53fa2/Reservation/?operation=cancel HTTP/1.1
Host: localhost
User-Agent: IgnorantHttpClient
```

# Safety

- **Safety**

- Ist definiert im HTTP RFC
- Eine **Safe** Methode führt **nie zu einer Zustandsänderung einer Ressource**
- Aus Sicht des Zustandes einer Ressource hat die Ausführung einer Safe Methode den gleichen Effekt, als ob diese nie ausgeführt worden wäre
- GET ist Safe
- User-agents (Browser, HttpClient) und Intermediaries (Proxies, ...) kennen das Vokabular von HTTP und deren Bedeutung!
- Tritt beispielsweise ein Fehler beim GET-Aufruf auf, so kann ein HttpClient oder ein Proxy automatisch den Aufruf wiederholen ohne das irgendwelche Seiteneffekte auftreten. Dies setzt jedoch voraus, dass die Methode gemäß HTTP implementiert ist

```
GET /Hotel/e53fa2 HTTP/1.1  
Host: localhost  
User-Agent: xLightweb/2.6.1
```

```
GET /Hotel/e53fa2/Reservation/?operation=cancel HTTP/1.1  
Host: localhost  
User-Agent: IgnorantHttpClient
```

# Idempotency

- **Idempotency**

- Der erfolgreiche Aufruf einer **idempotenten** Methode **führt immer zum selben Zustand der Ressource**. Das Ergebnis einer erfolgreich aufgerufenen idempotenten Methode ist unabhängig der Anzahl wie oft der Aufruf wiederholt wurde
- DELETE und PUT sind Idempotent
- Ein PUT speichert eine *vollständige* Ressource unter der angegebenen URI. Ist bereits eine Resource vorhanden, so wird diese ersetzt
- Schlägt der PUT oder DELETE Aufruf aus unbekannten Gründen fehl → Einfach nochmals ausführen! keine Seiteneffekte bzw. der Letzte gewinnt!

```
PUT /Hotel/e53fa2/Guest/455 HTTP/1.1
```

```
Host: localhost
```

```
User-Agent: xLightweb/2.6.1
```

```
Content-Length: 134
```

```
Content-Type: application/x-www-form-urlencoded
```

```
zip=30314&lastName=Gump&street=42+Plantation+Street&firstName=Forest&country=US&city=Baytown&state=LA
```



# POST vs. PUT

- **POST vs. PUT**

- POST ist nicht Idempotent (und erst recht nicht Safe).
- Schlägt der POST Aufruf aus unbekannten Gründen fehl – im Sinne hat es funktioniert oder nicht? - so darf der Request nicht einfach nochmals ausgeführt werden
- POST wird auch von Technologien wie SOAP verwendet, um die SOAP messages über HTTP zu tunneln. Im Endeffekt reduzieren Technologien wie SOAP das *Application-Level* Protokoll HTTP auf ein *Transport-Level* Protokoll.
- PUT dient als Factory Methode (client-side managed URI) und Replace
- POST dient als Factory Methode (server-side managed URI) und Partial Updates

```
POST /Hotel/e53fa2 HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
Content-Length: 28
Content-Type: application/x-www-form-urlencoded

classification=Comfort&name=Astoria
```

# Signalisierung von Fehler

- **Signalisierung von Fehler**

- Ein Fehler wird über den entsprechenden HTTP Status code signalisiert:
  - **1xx** provisional response
  - **2xx** successful response
  - **3xx** redirect
  - **4xx** client error
  - **5xx** server error
- HTTP Clients und Intermediaries interpretieren ebenfalls den Response code. HTTP ist ein *application-level* Protokoll!
- Manchmal wird zur Detaillierung des Fehlers ein X-Header verwendet, oder Details zum Fehler werden im Body angegeben

```
HTTP/1.1 400 Bad Request
Server: xLightweb/2.6
Content-Length: 1468
X-Enhanced-Status: BAD_ADDR_ZIP
Content-Type: text/html; charset=iso-8859-1
```

```
<html>
```

```
...
```



Einordnung

Anatomie einer HTTP Interaktion

HTTP Caching

Adressierung von Ressourcen

Die Operationen auf eine Ressource (Uniform Interface)

**Modellierung von Ressourcen**

JAX-RS

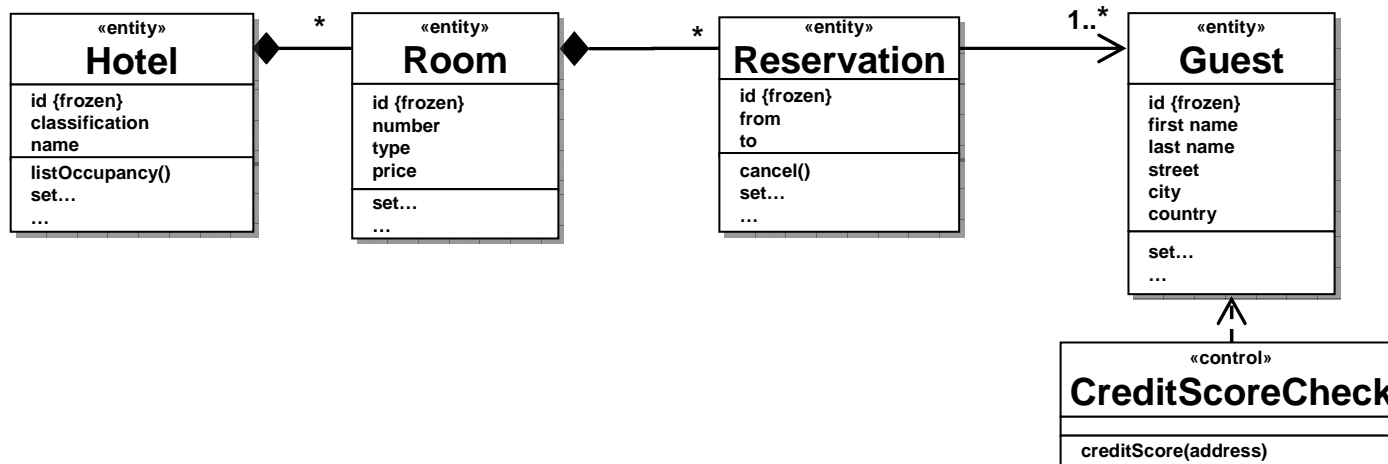
Implementierung von Non-CRUD Operation

Stateful vs. Stateless

Zusammenfassung

# Welcher Ansatz verfolgen SOAP, CORBA & Co.?

- Bei RPC-Technologien wie CORBA oder SOAP steht ein transparentes *Programming Language-Binding* im Mittelpunkt.
- Netzwerkaspekte wie beispielsweise Skalierbarkeit oder Robustheit spielen zwar eine sehr wichtige, aber eben nur noch zweite Rolle. Der Fokus liegt auf ein transparentes Objekt-Mapping



# Die 8 Irrtümer des verteilten Computing

- Bei RPC-Technologien wie CORBA oder SOAP steht ein transparentes *Programming*

- N...e sehr  
w...ekt-  
M...

## The Eight Fallacies of Distributed Computing

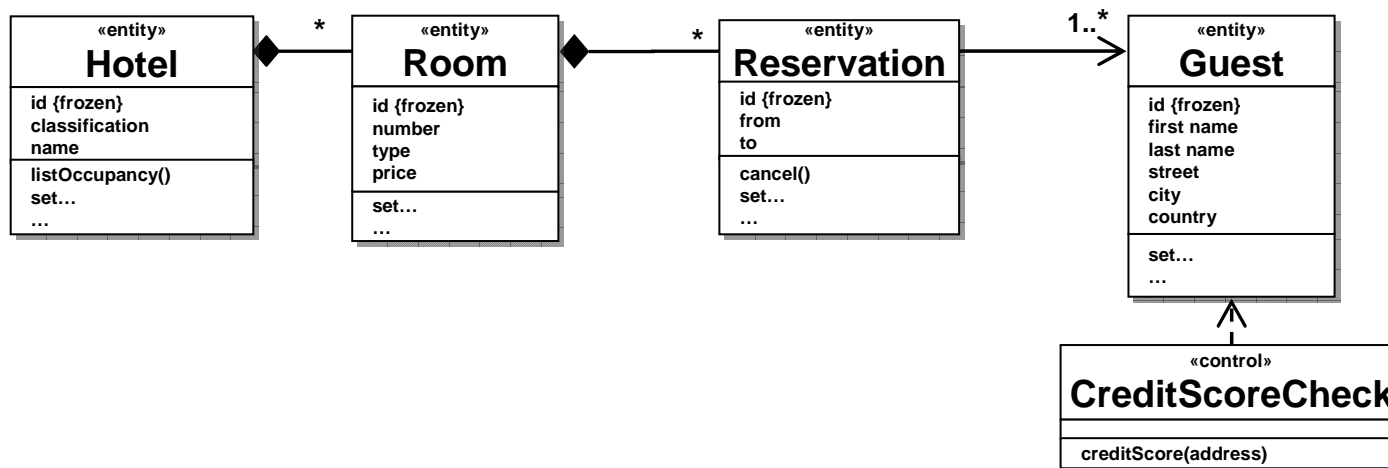
*Peter Deutsch*

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause *big* trouble and *painful* learning experiences.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

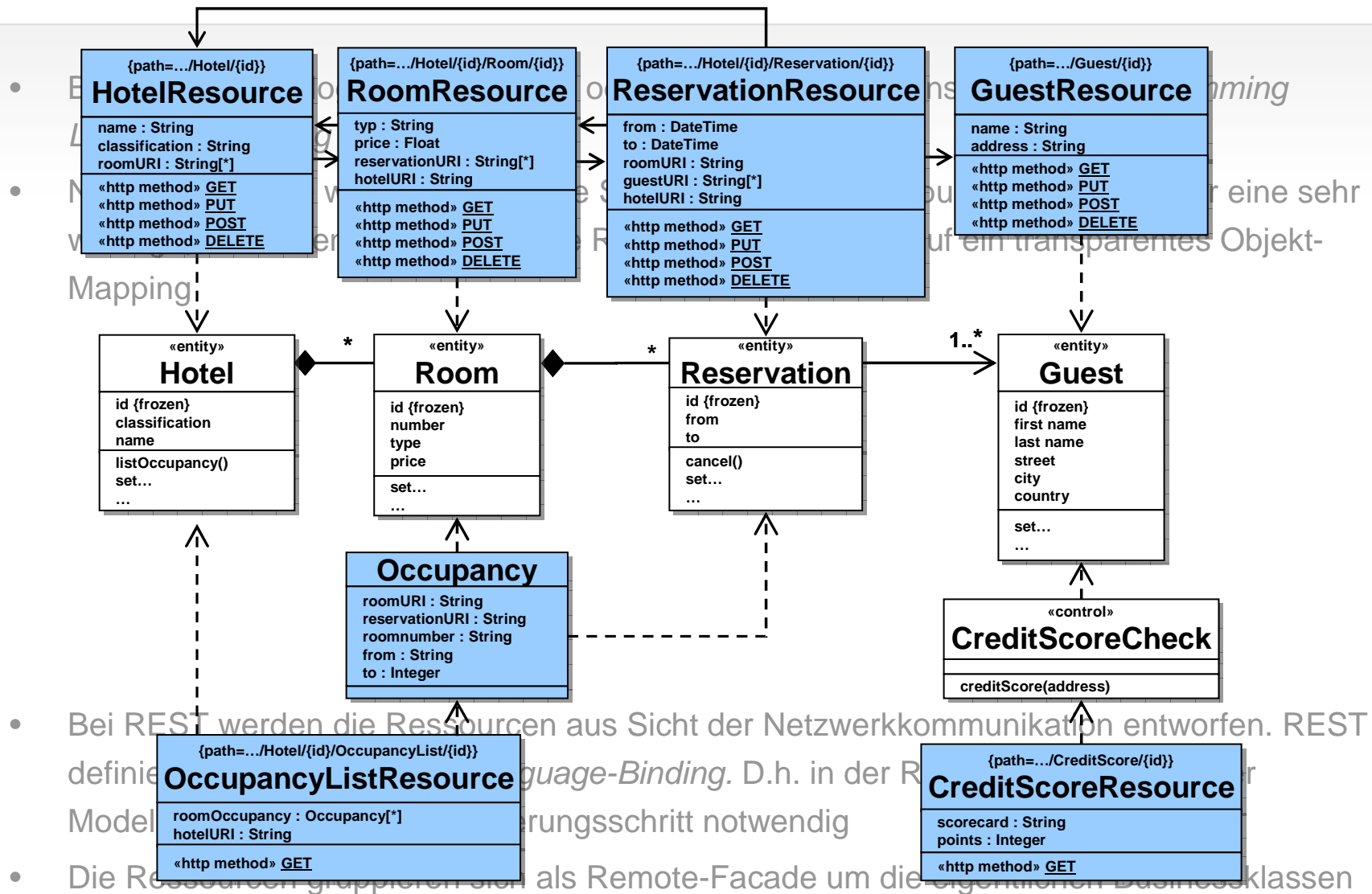
## ... und was macht REST?

- Bei RPC-Technologien wie CORBA oder SOAP steht ein transparentes *Programming Language-Binding* im Mittelpunkt.
- Netzwerkaspekte wie beispielsweise Skalierbarkeit oder Robustheit spielen zwar eine sehr wichtige, aber eben nur noch zweite Rolle. Der Fokus liegt auf ein transparentes Objekt-Mapping



- Bei REST werden die Ressourcen aus Sicht der Netzwerkkommunikation entworfen. REST definiert kein *Programming Language-Binding*. D.h. in der Regel ist ein zusätzlicher Modellierungs- und Implementierungsschritt notwendig
- Die Ressourcen gruppieren sich als Remote-Facade um die eigentlichen Businessklassen

# Die Ressourcen-Sicht





Einordnung

Anatomie einer HTTP Interaktion

HTTP Caching

Adressierung von Ressourcen

Die Operationen auf eine Ressource (Uniform Interface)

Modellierung von Ressourcen

**JAX-RS**

Implementierung von Non-CRUD Operation

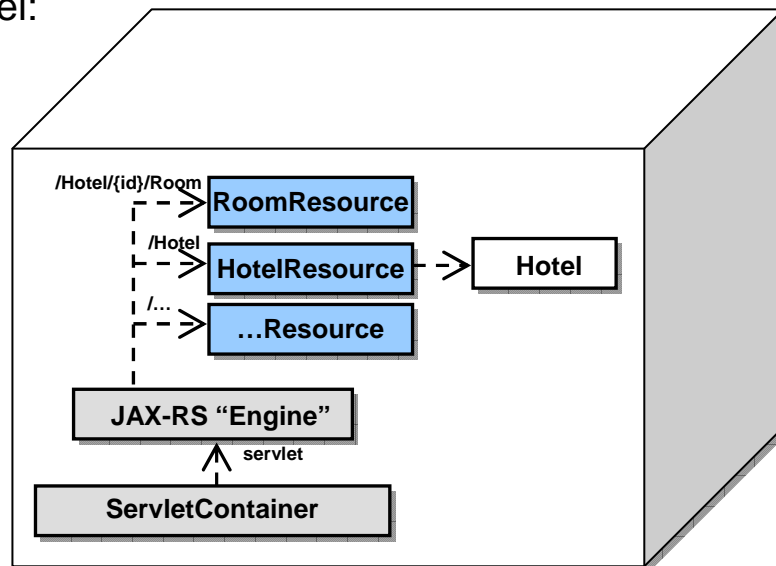
Stateful vs. Stateless

Zusammenfassung



# JAX-RS

- JSR 311: JAX-RS: The Java API for RESTful Web Services
- „The existing standard APIs Servlet and JAX-WS Provider *are too low-level*“ (Paul Sandoz, JSR 311 Spec lead)
- Die aktuelle Version 1.0 (Okt/2008) betrachtet nur die Serverseite
- Neben der Referenzimplementierung Jersey existieren eine Reihe von weiteren Implementierungen wie z.B. JBoss RESTEasy oder Apache Wink
- Beispiel:



# Implementierung einer Ressource mit Hilfe JAX-RS

```
@Path("/Hotel")
public class HotelResource {

    private final HotelDao hotelDao;

    public HotelResource(HotelDao hotelDao) {
        this.hotelDao = hotelDao;
    }

    @POST
    @Consumes("application/x-www-form-urlencoded")
    public Response createHotel(@FormParam("name") String name,
                                @FormParam("classification") String clf) throws IOException {
        Hotel hotel = hotelDao.createHotel(name, clf);
        return Response.created(URI.create("/Hotel/" + hotel.getId())).build();
    }

    @GET
    @Path("/{id}")
    @Produces("application/json")
    public Hotel getHotel(@PathParam("id") String id) throws IOException, NotFoundException {
        Hotel hotel = hotelDao.retrieveHotel(id);
        return hotel;
    }
}
```

# Manuelle Erzeugung eines Response

```
@Path("/Hotel")
public class HotelResource {

    private final HotelDao hotelDao;

    public HotelResource(HotelDao hotelDao) {
        this.hotelDao = hotelDao;
    }

    @POST
    @Consumes("application/x-www-form-urlencoded")
    public Response createHotel(@FormParam("name") String name,
                               @FormParam("classification") String clf) throws IOException {
        Hotel hotel = hotelDao.createHotel(name, clf);
        return Response.created(URI.create("/Hotel/" + hotel.getId())).build();
    }

    @GET
    @Path("/{id}")
    @Produces("application/json")
    public Hotel getHotel(@PathParam("id") String id) throws IOException, NotFoundException {
        Hotel hotel = hotelDao.retrieveHotel(id);
        return hotel;
    }
}
```

HTTP/1.1 201 Created  
Server: xLightweb/2.6  
Content-Length: 0  
Location: http://localhost/hotel/e53fa2

# Nutzung des Automappings

```
@Path("/Hotel")
public class HotelResource {

    private final HotelDao hotelDao;

    public HotelResource(HotelDao hotelDao) {
        this.hotelDao = hotelDao;
    }

    @POST
    @Consumes("application/x-www-form-urlencoded")
    public Response createHotel(@FormParam("name") String name,
                                @FormParam("classification") String clf) throws IOException {
        Hotel hotel = hotelDao.createHotel(name, clf);
        return Response.created(URI.create("/Hotel/" + hotel.getId())).build();
    }

    @GET
    @Path("/{id}")
    @Produces("application/json")
    public Hotel getHotel(@PathParam("id") String id) throws IOException {
        Hotel hotel = hotelDao.retrieveHotel(id);
        return hotel;
    }
}
```

Anhand der Kombination MimeType der Rückgabe, und des zurückgegebenen Objektes wird automatisch ein bestimmter JAX-RS MessageBodyWriter verwendet

# Beispiel einer einfachen Provider-Implementierung

```
@Provider
@Produces("application/json")
@Consumes("application/json")
public class JSONLibBasedProvider implements MessageBodyWriter<Object>,
                                           MessageBodyReader<Object> {

    // writer methods
    public boolean isWriteable(Class<?> type, Type genericType, Annotation[] annotations,
                               MediaType mediaType) {
        return (mediaType.getType().equalsIgnoreCase("application") &&
                mediaType.getSubtype().equalsIgnoreCase("json"));
    }

    public void writeTo(Object bean, Class<?> type, Type genericType, Annotation[] annotations,
                        MediaType mediaType, MultivaluedMap<String, Object> httpHeaders,
                        OutputStream entityStream) throws IOException, WebApplicationException {

        String serialized = JSONObject.fromObject(bean).toString();
        Writer writer = new OutputStreamWriter(entityStream, "ISO-8859-1");
        writer.write(serialized);
        writer.close();
    }

    // reader methods
    // ...
}
```



Einordnung

Anatomie einer HTTP Interaktion

HTTP Caching

Adressierung von Ressourcen

Die Operationen auf eine Ressource (Uniform Interface)

Modellierung von Ressourcen

JAX-RS

**Implementierung von Non-CRUD Operation**

Stateful vs. Stateless

Zusammenfassung

# Non-CRUD - asynchrone Operation

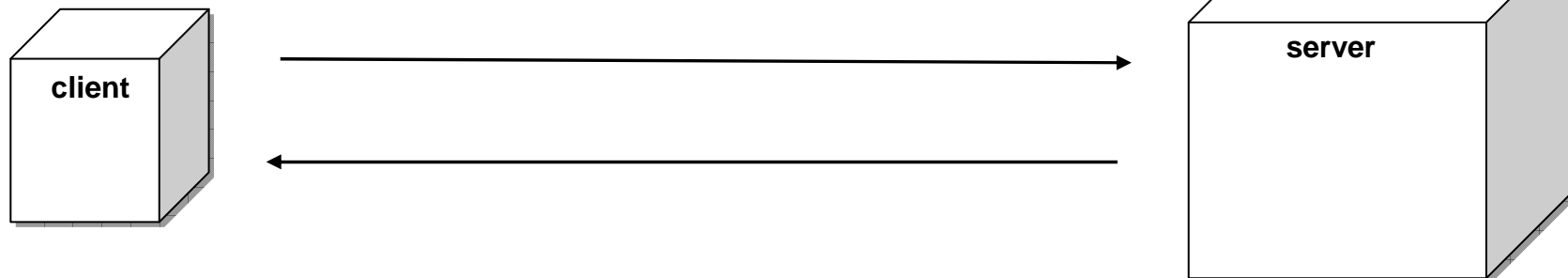
```
POST /Mailsubmission/ HTTP/1.1
```

```
Host: localhost
```

```
User-Agent: xLightweb/2.6.1
```

```
Content-Type: application/x-www-form-urlencoded
```

```
delaySec=600&mail=Message-ID%3A+%3C49A658D7.506454%40united.domain%3E0D%  
0ADate%3A+Thu%202C+26+Feb+2009+09%3A55%3A60+%2B0100%0D%0AFrom%3A+rest-rock  
s%40gmx.com%0D%0AX-Mozilla-Draft-Info%3A+internal%2Fdraft%3B+vcard%3D0%3B  
+receipt%3D0%3B+uuencode%3D0%0D%0AUser-Agent%3A+Thunderbird+2.0.0.19+%28W  
indows%2F20081209%29%0D%0ATo%3A+all%40xlightweb.org%0D%0ASubject%3A+hi+al  
l%0D%0A%0D%0AHi%2C%0D%0Athis+is+another+test+mail%0D%0A
```



## Non-CRUD - asynchrone Operation (II)

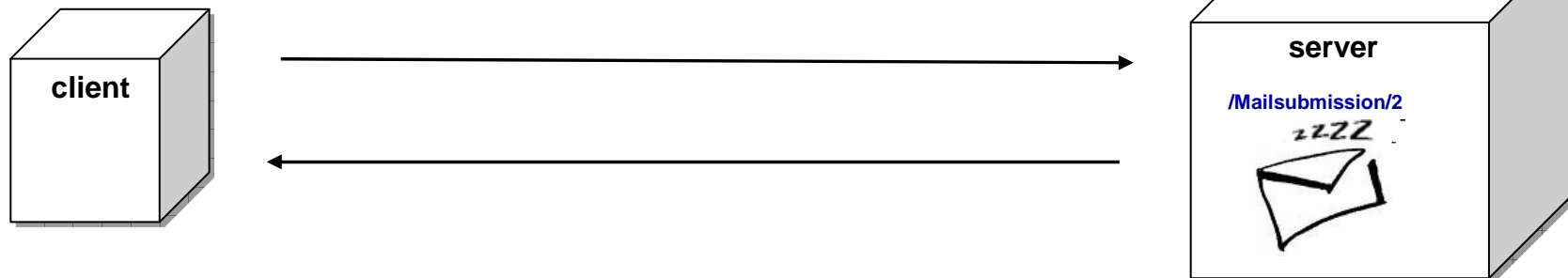
```
POST /Mailsubmission/ HTTP/1.1
```

```
Host: localhost
```

```
User-Agent: xLightweb/2.6.1
```

```
Content-Type: application/x-www-form-urlencoded
```

```
delaySec=600&mail=Message-ID%3A+%3C49A658D7.506454%40united.domain%3E%0D%0A  
Date%3A+Thu%2026+Feb+2009+09%3A55%3A60+%2B0100%0D%0AFrom%3A+rest-rock  
s%40gmx.com%0D%0AX-Mozilla-Draft-Info%3A+internal%2Fdraft%3B+vcard%3D0%3B  
+receipt%3D0%3B+uuencode%3D0%0D%0AUser-Agent%3A+Thunderbird+2.0.0.19+%28W  
indows%2F20081209%29%0D%0ATo%3A+all%40xlightweb.org%0D%0ASubject%3A+hi+al  
l%0D%0A%0D%0AHi%2C%0D%0Athis+is+another+test+mail%0D%0A
```





## Non-CRUD - asynchrone Operation (III)

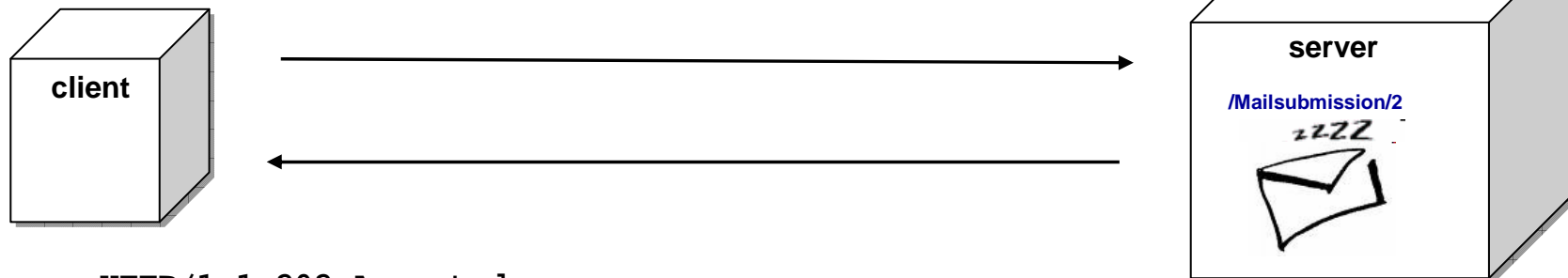
```
POST /Mailsubmission/ HTTP/1.1
```

```
Host: localhost
```

```
User-Agent: xLightweb/2.6.1
```

```
Content-Type: application/x-www-form-urlencoded
```

```
delaySec=600&mail=Message-ID%3A+%3C49A658D7.506454%40united.domain%3E%0D%0A  
Date%3A+Thu%2026+Feb+2009+09%3A55%3A60+%2B0100%0D%0AFrom%3A+rest-rock  
s%40gmx.com%0D%0AX-Mozilla-Draft-Info%3A+internal%2Fdraft%3B+vcard%3D%3B  
+receipt%3D%3B+uuencode%3D%0D%0AUser-Agent%3A+Thunderbird+2.0.0.19+%28W  
indows%2F20081209%29%0D%0ATo%3A+all%40xlightweb.org%0D%0ASubject%3A+hi+al  
l%0D%0A%0D%0AHi%2C%0D%0Athis+is+another+test+mail%0D%0A
```



```
HTTP/1.1 202 Accepted
```

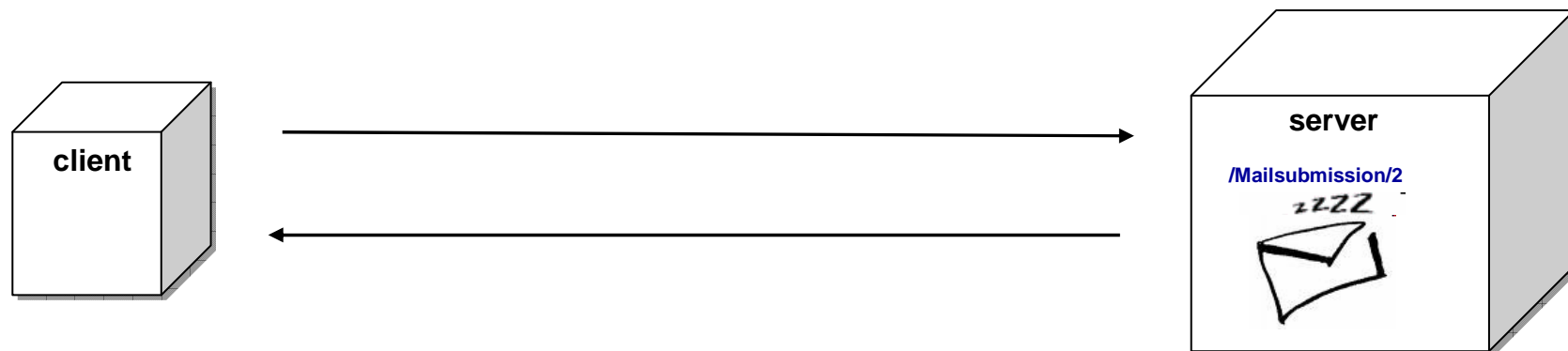
```
Server: Apache/1.3.6
```

```
Content-Length: 0
```

```
Location: http://localhost/Mailsubmission/2
```

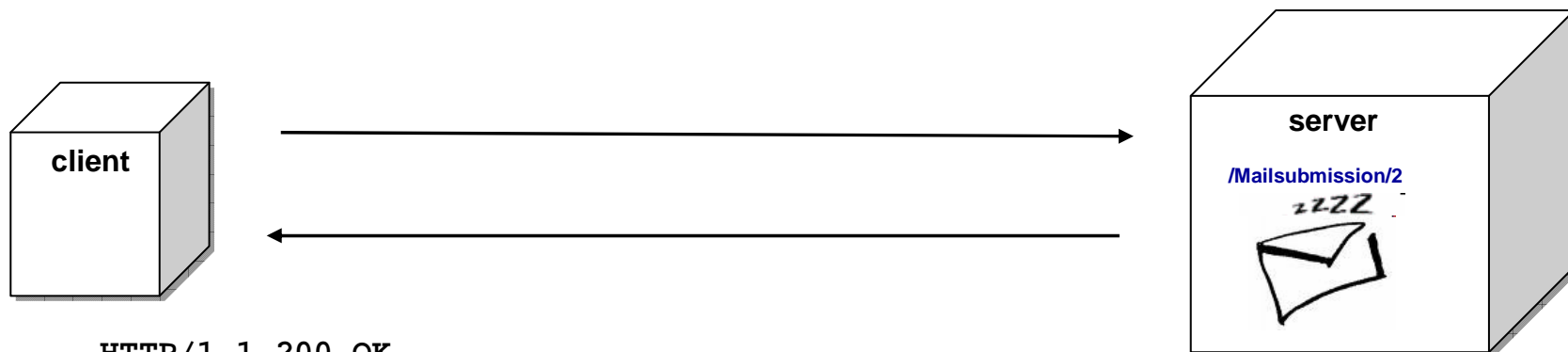
# Abfragen des Versendestatus

```
GET /Mailsubmission/2 HTTP/1.1  
Host: localhost  
User-Agent: xLightweb/2.6.1  
Accept: application/x-www-form-urlencoded
```



## Abfragen des Versendestatus (II)

```
GET /Mailsubmission/2 HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
Accept: application/x-www-form-urlencoded
```



```
HTTP/1.1 200 OK
Server: Apache/1.3.6
Content-Length: 14
Content-Type: application/x-www-form-urlencoded

state=pending&remainingTimeSec=412
```

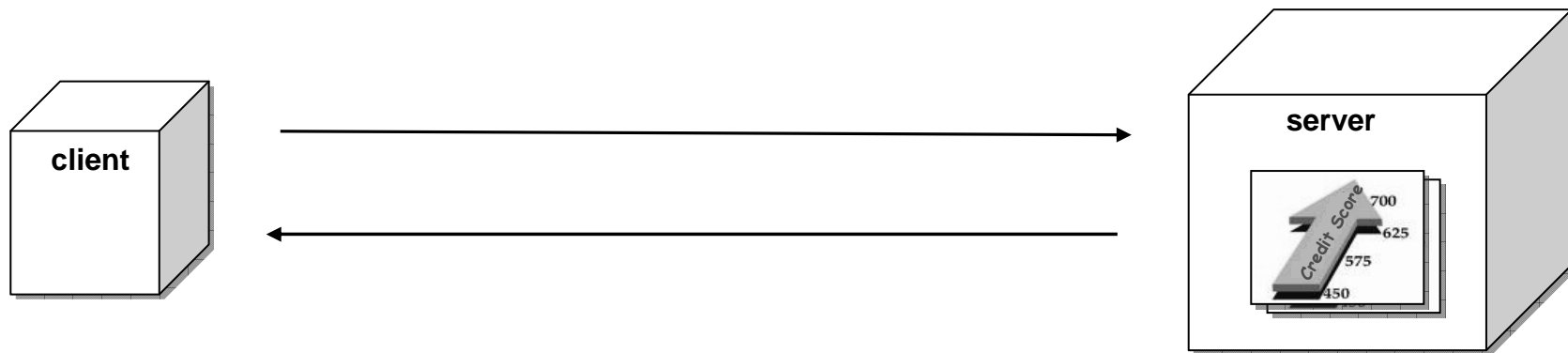
## Non-CRUD - synchrone GET Operation

```
GET /CreditScore/?zip=30314&lastName=Gump&street=42+Plantation+Street&  
firstName=Forest&country=US&city=Baytown&state=LA... HTTP/1.1
```

Host: localhost

User-Agent: xLightweb/2.6.1

Accept: application/x-www-form-urlencoded



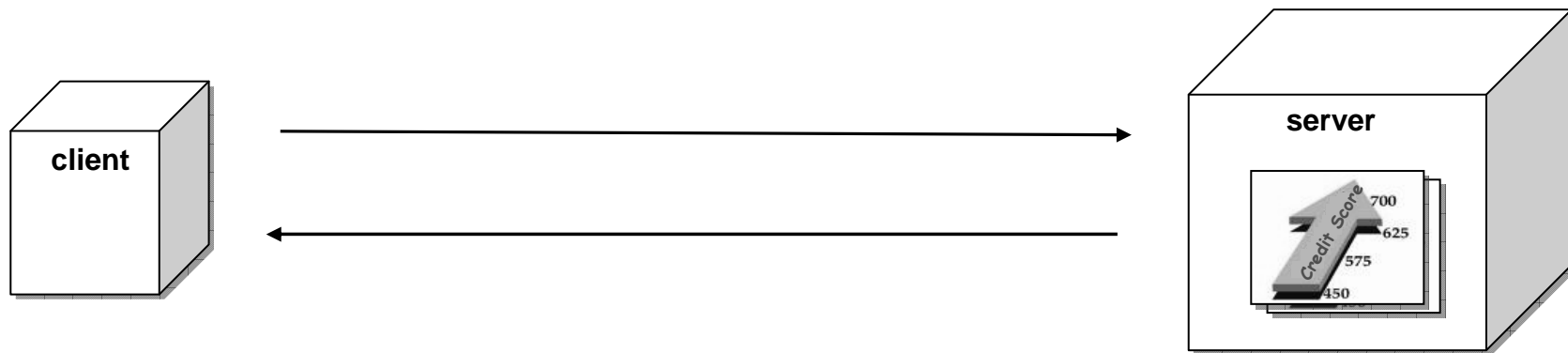
## Non-CRUD - synchrone GET Operation (II)

```
GET /CreditScore/?zip=30314&lastName=Gump&street=42+Plantation+Street&  
firstName=Forest&country=US&city=Baytown&state=LA... HTTP/1.1
```

```
Host: localhost
```

```
User-Agent: xLightweb/2.6.1
```

```
Accept: application/x-www-form-urlencoded
```



```
HTTP/1.1 200 OK
```

```
Server: Apache/1.3.6
```

```
Content-Length: 30
```

```
Cache-Control: public, max-age=60
```

```
Content-Type: application/x-www-form-urlencoded
```

```
scorecard=Excellent&points=92
```

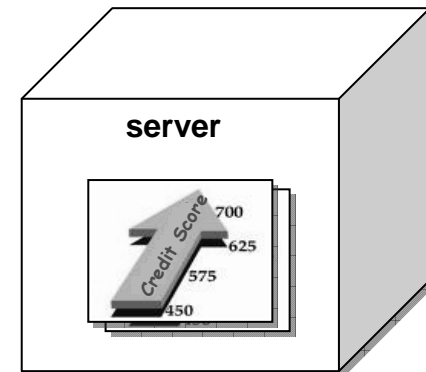
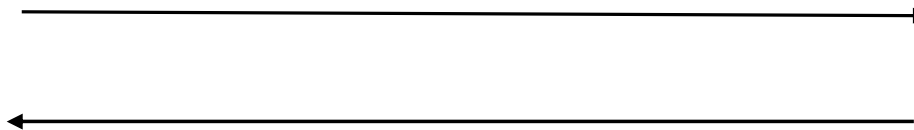
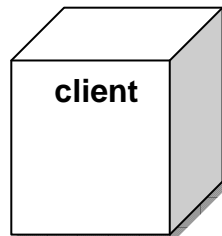
## Non-CRUD - synchrone GET Operation (II)

```
GET /CreditScore/?zip=30314&lastName=Gump&street=42+Plantation+Street&  
firstName=Forest&country=US&city=Baytown&state=LA... HTTP/1.1
```

```
Host: local  
User-Agent:  
Accept: ap
```

**Max Length URL!**

**Internet Explorer ~2 kbyte**  
**Apache 2.2 8190 bytes (default-Einstellung)**



```
HTTP/1.1 200 OK  
Server: Apache/1.3.6  
Content-Length: 30  
Cache-Control: public, max-age=60  
Content-Type: application/x-www-form-urlencoded  
  
scorecard=Excellent&points=92
```

## Non-CRUD - synchrone POST Operation

```
POST /CreditScore/ HTTP/1.1
```

```
Host: localhost
```

```
User-Agent: xLightweb/2.6.1
```

```
Content-Type: text/x-vcard
```

```
BEGIN:VCARD
```

```
VERSION:2.1
```

```
N:Gump;Forest;;;;
```

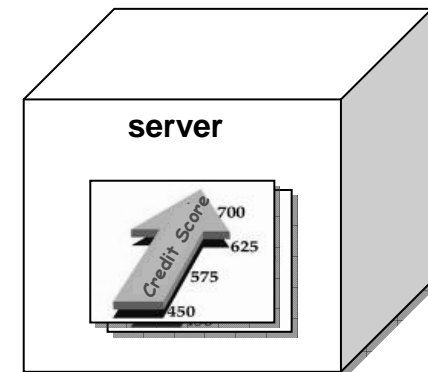
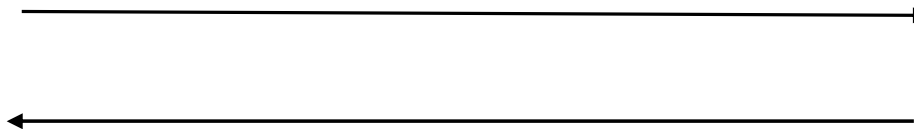
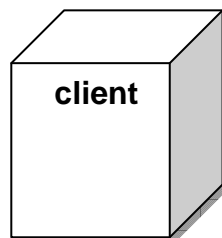
```
FN:Forest Gump
```

```
ADR;HOME;;;42 Plantation St.;Baytown;LA;30314;US
```

```
LABEL;HOME;ENCODING=QUOTED-PRINTABLE:42 Plantation St.=0D=0A30314
```

```
Baytown=0D=0ALA US
```

```
END:VCARD
```



```
HTTP/1.1 201 CREATED
```

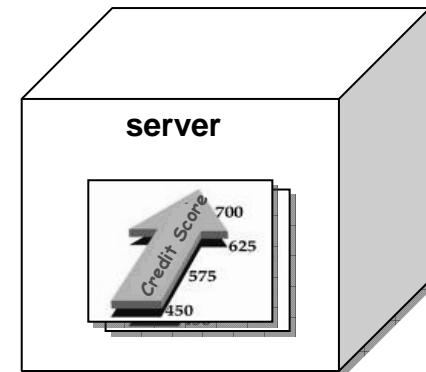
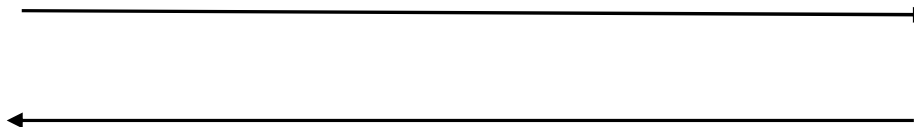
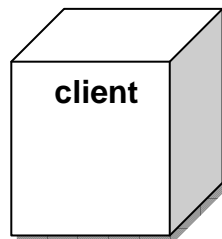
```
Server: Apache/1.3.6
```

```
Content-Length: 0
```

```
Location: http://localhost/CreditScore/100000001
```

## Non-CRUD - synchrone POST Operation (II)

```
GET /CreditScore/100000001 HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6.1
Accept: application/x-www-form-urlencoded
```



```
HTTP/1.1 200 OK
Server: Apache/1.3.6
Content-Length: 30
Cache-Control: public, max-age=60
Content-Type: application/x-www-form-urlencoded

scorecard=Excellent&points=92
```





Einordnung

Anatomie einer HTTP Interaktion

HTTP Caching

Adressierung von Ressourcen

Die Operationen auf eine Ressource (Uniform Interface)

Modellierung von Ressourcen

JAX-RS

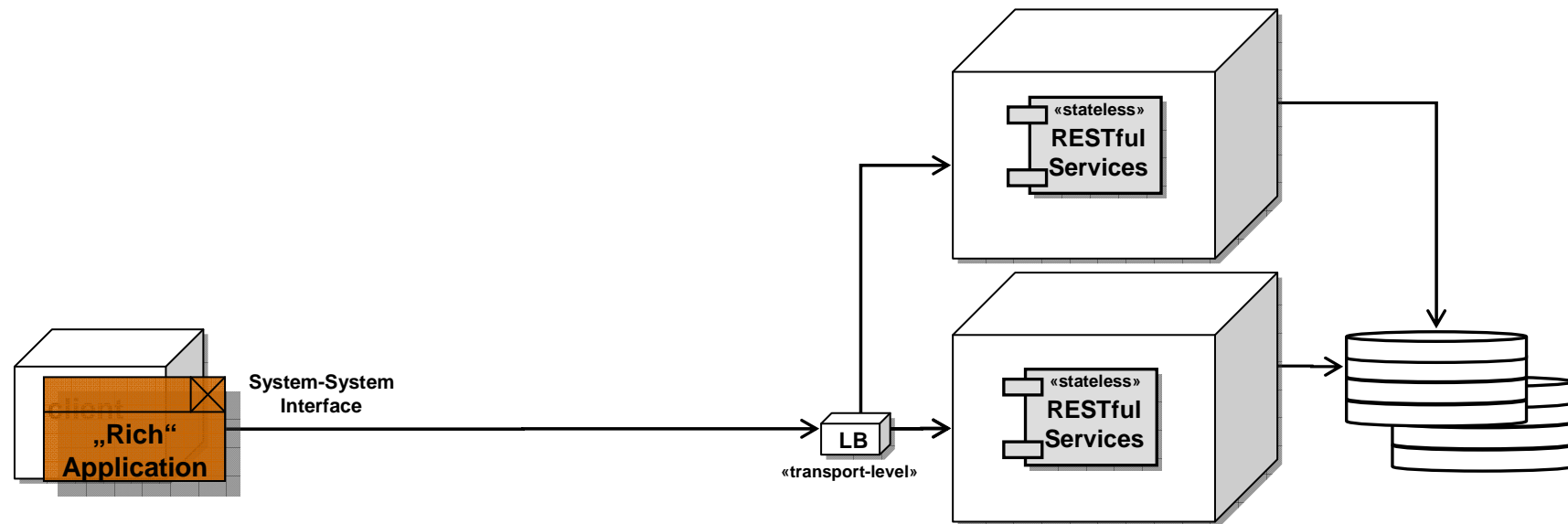
Implementierung von Non-CRUD Operation

**Stateful vs. Stateless**

Zusammenfassung

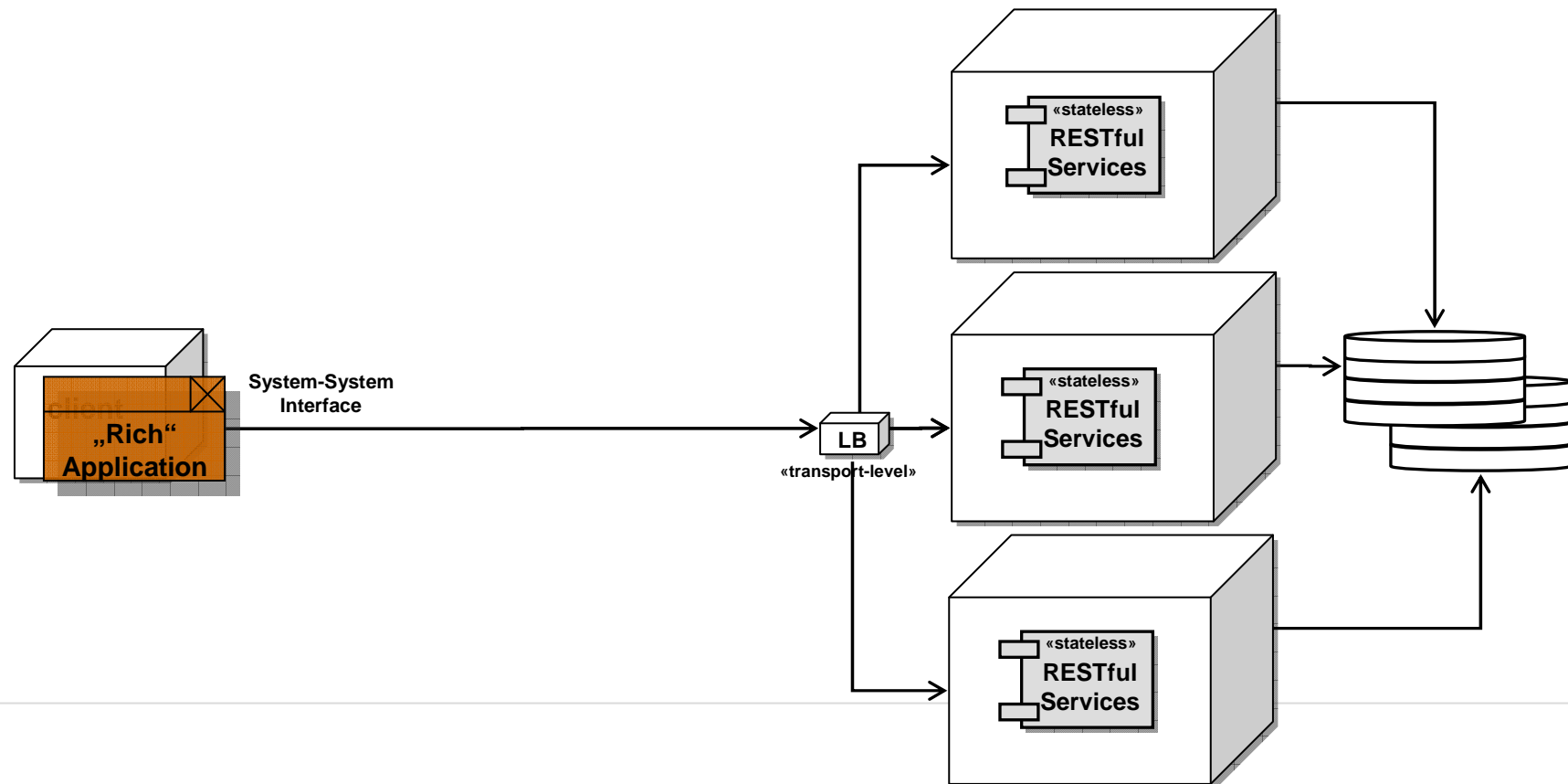
# Statelessness

- **REST Interaktionen sind stateless**
  - Ein Request enthält alle für die Prozessierung notwendigen Informationen
  - Requests können damit isoliert, auch von intermediaries (proxies, ...), behandelt werden



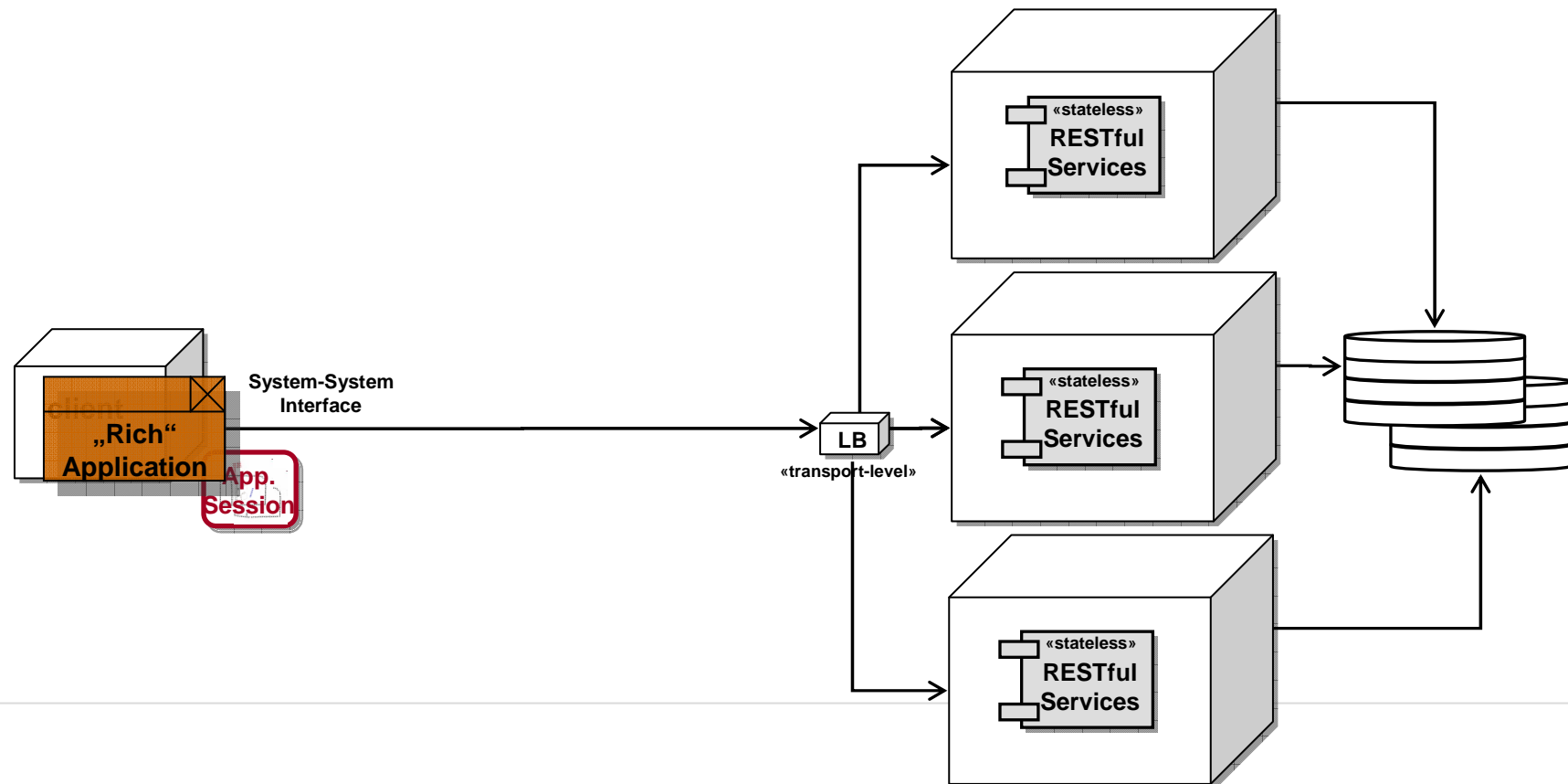
# Scale-up

- **REST Interaktionen sind stateless**
  - Ein Request enthält alle für die Prozessierung notwendigen Informationen
  - Requests können damit isoliert, auch von intermediaries (proxies, ...), behandelt werden



# Application state

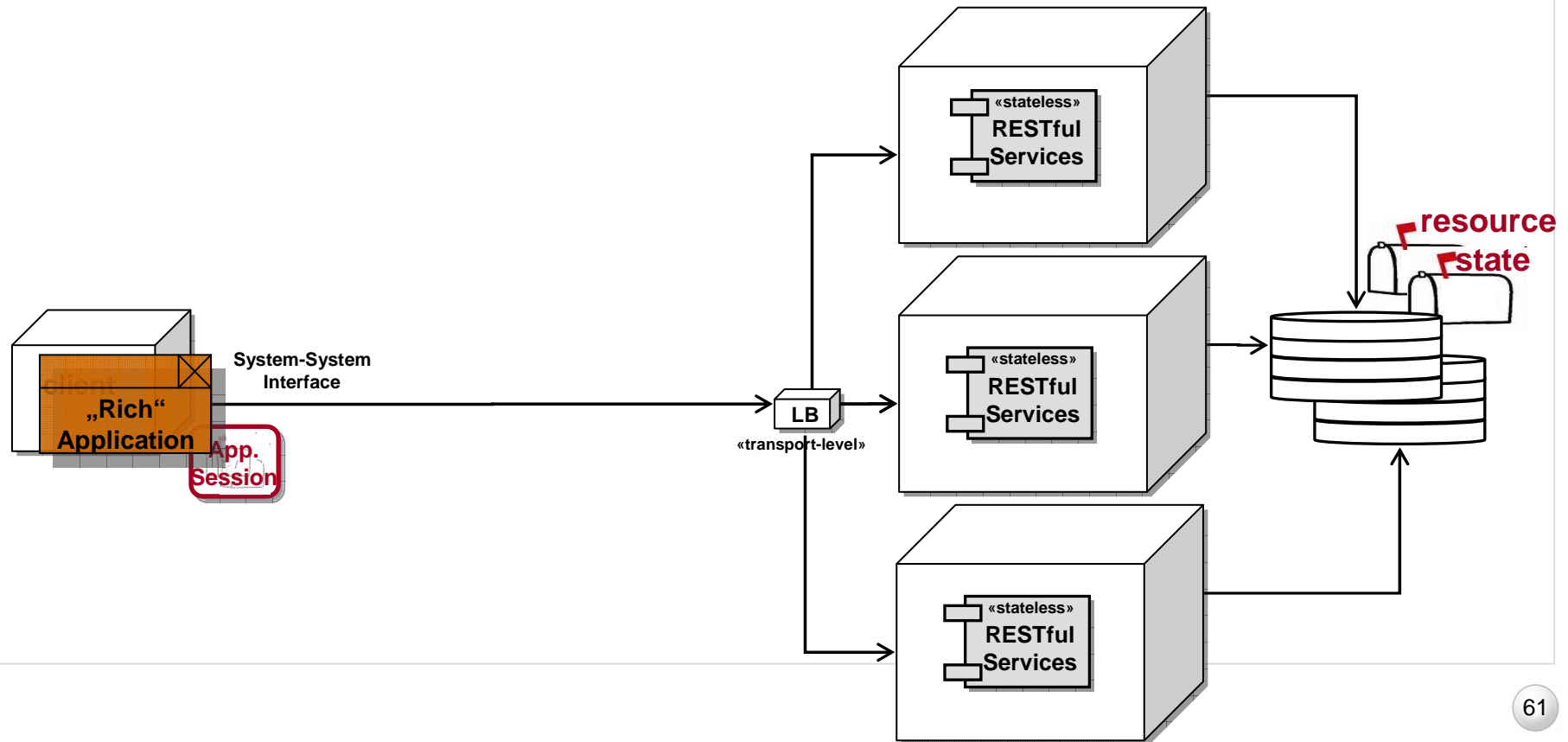
- **REST Interaktionen sind stateless**
  - Ein Request enthält alle für die Prozessierung notwendigen Informationen
  - Requests können damit isoliert, auch von intermediaries (proxies, ...), behandelt werden
  - Der Client ist verantwortlich für den *application state*



# Application state

- **REST Interaktionen sind stateless**

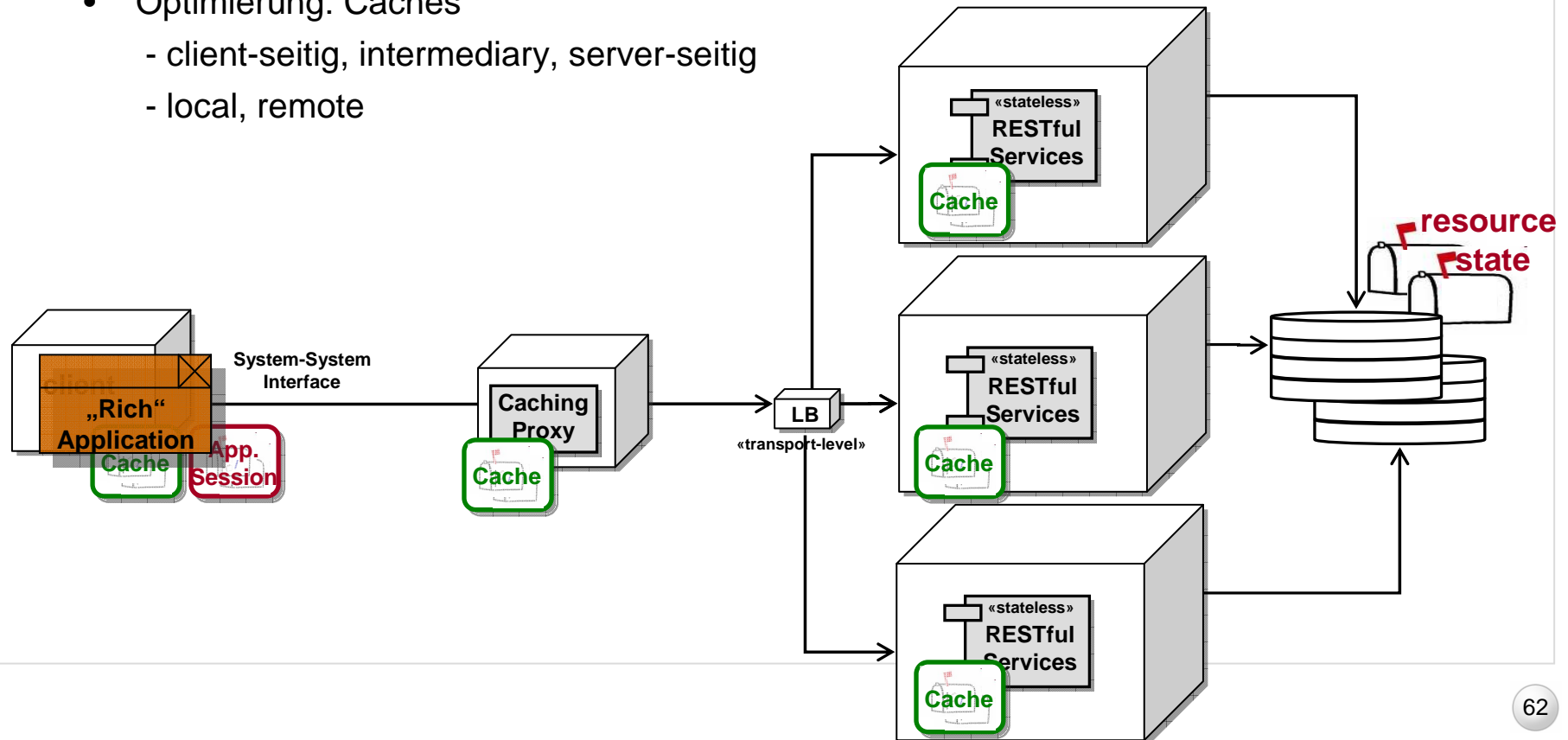
- Ein Request enthält alle für die Prozessierung notwendigen Informationen
- Requests können damit isoliert, auch von intermediaries (proxies, ...), behandelt werden
- Der Client ist verantwortlich für den *application state*, der Service für den *resource state*



# Caching

- **REST Interaktionen sind stateless**

- Ein Request enthält alle für die Prozessierung notwendigen Informationen
- Requests können damit isoliert, auch von intermediaries (proxies, ...), behandelt werden
- Der Client ist verantwortlich für den *application state*, der Service für den *resource state*
- Optimierung: Caches
  - client-seitig, intermediary, server-seitig
  - local, remote





Einordnung

Anatomie einer HTTP Interaktion

HTTP Caching

Adressierung von Ressourcen

Die Operationen auf eine Ressource (Uniform Interface)

Modellierung von Ressourcen

JAX-RS

Implementierung von Non-CRUD Operation

Stateful vs. Stateless

**Zusammenfassung**

# Zusammenfassung

- **RESTful HTTP ermöglicht sehr einfache Kommunikationsschnittstellen**
  - Minimale Infrastrukturvoraussetzungen. Der Aufrufer benötigt lediglich einen HTTPClient
  - *Self-descriptive Messages* ermöglichen eine sehr einfaches und zugleich flexibles Interface
  - RESTful HTTP ermöglicht eine echte, sprach-/technologieübergreifende Interoperabilität
- **RESTful HTTP ermöglicht eine effiziente Implementierung von robusten und hoch skalierbaren *WEB*-Anwendung**
  - Die Remote-Schnittstelle, die Ressourcen, berücksichtigt vom Kern auf Netzwerkaspekte.
  - RESTful HTTP ermöglicht im Gegensatz zu Webservice-Architekturen wie z.B. SOAP eine sehr effiziente Nutzung der WEB-Infrastruktur (Caching,...)
  - Die Berücksichtigung von Safety und Idempotency ermöglicht die Implementierung von robusten Netzwerk-Schnittstellen in einer sehr einfachen Art und Weise
  - REST setzt keine spezielle Middleware-Komponenten oder komplexe und aufwendige Middleware-Infrastrukturen wie beispielsweise Sessionfailover-Cluster voraus
  - REST definiert einen Architekturstil für *WEB-Applikationen*. Beispielsweise spielen dort verteilte atomic (2-Phase Commit) Transaktionen keine Rolle.



## Literatur

- **Roy Fielding – *Architectural Styles and the Design of Network-based Software Architectures***  
(<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>)
- **Stefan Tilkov – *REST und HTTP***  
(<http://www.dpunkt.de/buecher/3078.html>)
- **Steve Vinoski – *REST Eye for the SOA Guy***  
([http://steve.vinoski.net/pdf/IEEE-REST\\_Eye\\_for\\_the\\_SOA\\_Guy.pdf](http://steve.vinoski.net/pdf/IEEE-REST_Eye_for_the_SOA_Guy.pdf))
- **Gregor Roth – *RESTful HTTP in practice***  
(<http://www.infoq.com/articles/designing-restful-http-apps-roth>)