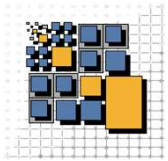
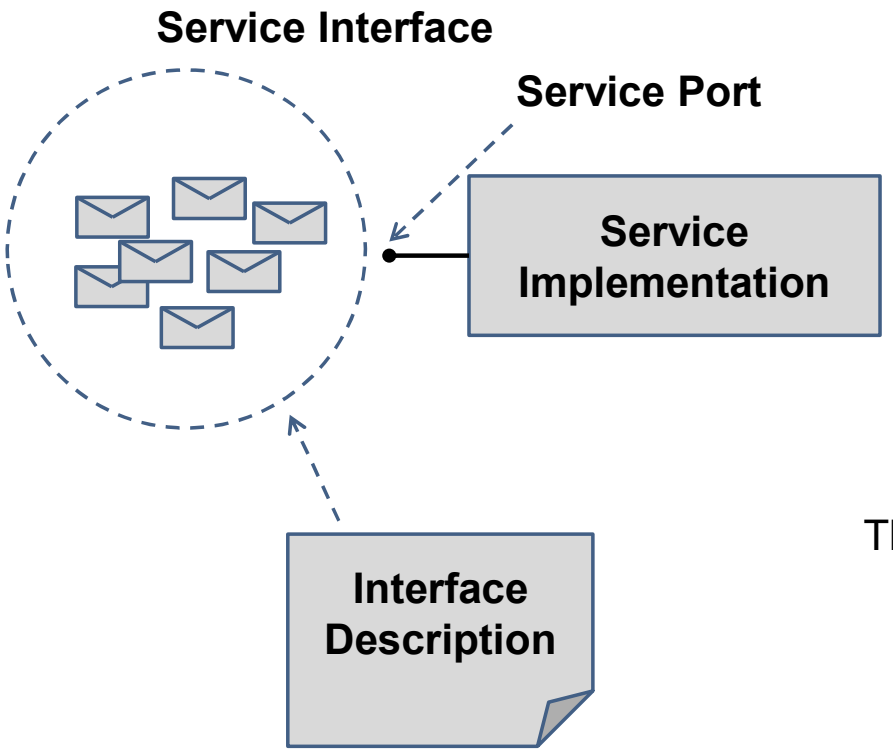


DSG-SOA-M 2024:
- Basic Service Implementation -

Dr. Andreas Schönberger

Lehrstuhl für Praktische Informatik
Fakultät WIAI
Otto-Friedrich-Universität Bamberg

Basic Choices for Service Implementation



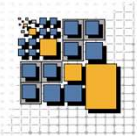
Let's reconsider the service definition of chapter 3 and think about implementation Choices:

- Protocol
  - Synchronous protocols such as HTTP
  - Asynchronous protocols such as MQTT and AMQP
- Design Style
  REST vs. RPC
- Interface Description
  Runtime vs. Buildtime

This lecture

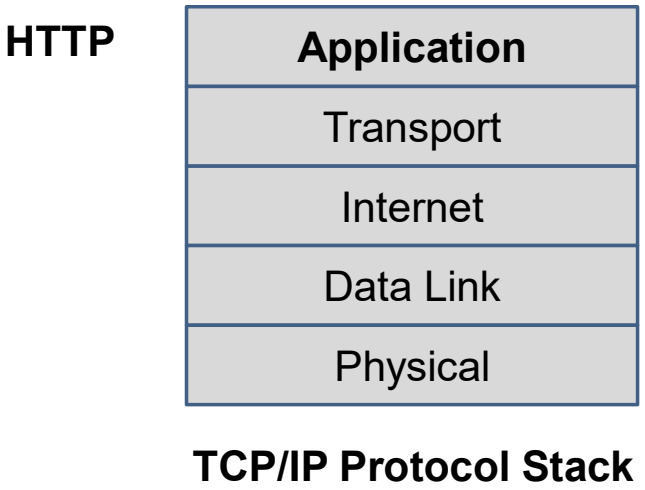
Choice of implementation framework deliberately left out: there are tons of options

HTTP/s – Common Services Protocol



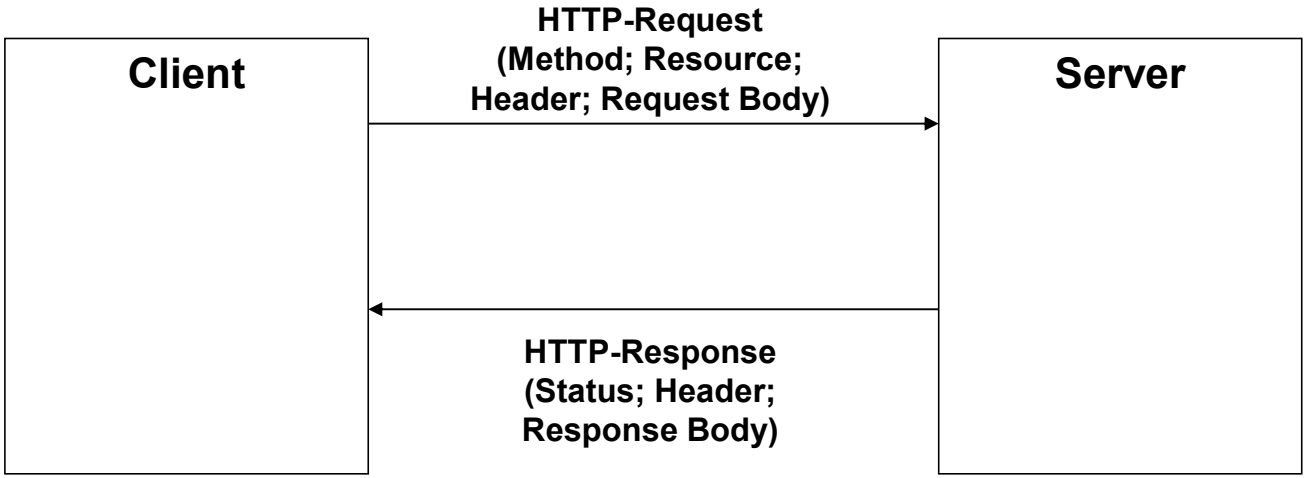
HTTP := Hypertext Transfer Protocol
IETF & W3C-Standard, first version 1996,
is the most widespread protocol for accessing services

- Important characteristics
  - **Synchronous**
    Call blocks until response
  - **Stateless**
    Two subsequent requests are separate transactions (yet, cookies possible)
  - **Content-agnostic**
    You can transfer almost any content via HTTP

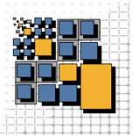


TCP/IP Protocol Stack

HTTP/s – Principal Interaction



HTTP/s – Sample Request



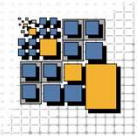
```
GET / HTTP/1.1
Host: www.kicker.de
```

Actual Request

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
DNT: 1
Connection: keep-alive
Cookie: ioam2018=0014f99570e6efeaf5ec04e34…truncated
Upgrade-Insecure-Requests: 1
TE: Trailers
```

Request Header

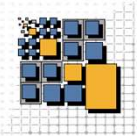
HTTP/s – Sample Response I



```
HTTP/2 200 OK
date: Sat, 16 May 2020 20:45:15 GMT
content-type: text/html; charset=utf-8
content-length: 79233
cache-control: public, max-age=60, s-maxage=60
content-encoding: gzip
expires: Sat, 16 May 2020 20:46:09 GMT
server: Footprint Distributor V6.1.1162
vary: Accept-Encoding, x-protocol
x-cache: HIT
x-kicker-powered-by: www64
x-ov-info: AppVersion: 20.20.7439.24537; BuildTime: 22:45:04; TimeSpan:
00:00:02.6875160; CacheTime: 60
age: 13
accept-ranges: bytes
X-Firefox-Spdy: h2
```

Response Header

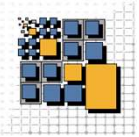
HTTP/s – Sample Response II



```
<!DOCTYPE html>
<html lang="de-DE">
<!--[if lt IE 7]> <html class="no-js ie6 oldie" lang="de"> <![endif]-->
<!--[if IE 7]>    <html class="no-js ie7 oldie" lang="de"> <![endif]-->
<!--[if IE 8]>    <html class="no-js ie8 oldie" lang="de"> <![endif]-->
<!--[if gt IE 8]> <html class="no-js" lang="de"><![endif]-->
<head>
   <!-- kicker_meta -->
<title>Alle News aus dem Fu&#223;ball und der Welt des Sports - kicker</title>
<meta name="Description" content="kicker.de ist das Online-Angebot zum bekannten Fu&#223;ballmagazin  - Topaktuelle News und Livescores aus den deutschen Amateurfu&#223;ball-Ligen, aus &#252;ber 20 internationalen Ligen und aus weiteren Sportarten wie Formel 1, Eishockey, Basketball, Handball, Tennis und Olympia
…truncated
```

Response Body

Most Important Request Types



GET: "The GET method requests transfer of a current selected representation for the target resource."

→ Retrieve resource, but should not alter target

POST: "The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics."

→ Create resource / append data / submit data to processing

PUT: "The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload."

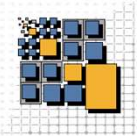
→ Update resource

DELETE: "The DELETE method [..] expresses a deletion operation on the URI mapping of the origin server rather than an expectation that the previously associated information be deleted"

→ Delete resource

Source of citations: https://tools.ietf.org/html/rfc7231

Most Important Status Codes



1xx: Informational response "The 1xx (Informational) class of status code indicates an interim response for communicating connection status or request progress prior to completing the requested action and sending a final response."

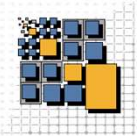
2xx: Success "The 2xx (Successful) class of status code indicates that the client's request was successfully received, understood, and accepted."

3xx: Redirection "The 3xx (Redirection) class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request."

4xx: Client errors "The 4xx (Client Error) class of status code indicates that the client seems to have erred."

5xx: Server erros "The 5xx (Server Error) class of status code indicates that the server is aware that it has erred or is incapable of performing the requested method."

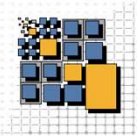
HTTP as a REST Stereotype



- In practice, the term HTTP is frequently used as synonym of REST
  … this is a source of misunderstanding
  - HTTP is a protocol that you can use to implement REST services
    BEWARE: You can build RPC-style services with HTTP, too
  - REST is an architectual style that may or may not be implemented with HTTP
- Example for breaking REST principles with HTTP

  Roy Fielding, originator of REST, says: *"An example of where an inappropriate extension has been made to the protocol to support features that contradict the desired properties of the generic interface is the introduction of site-wide state information in the form of HTTP cookies [73]. Cookie interaction fails to match REST's model of application state, often resulting in confusion for the typical browser application."*
- We will revisit REST in more detail later in the lecture

RESTful Services in Java?



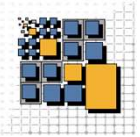
→ JAX-RS (JSR 370), Version 2.1, released in July 2017

- Jersey as reference implementation (https://jersey.github.io/)
- Dedicated to HTTP (as opposed to REST)
- Basically content agnostic
- Services provision in
  - Java SE (via JAX-WS)
  - Servlets
  - Stateless/Singleton EJBs

Jakarta RESTful Web Services 3.0/3.1 without substantial functional changes

- Core Terminology:
  - Application
  - Resources
  - Providers

Basic Example of a Resource



JSR 370: *"Resource classes are POJOs that have at least one method annotated with @Path or a request method designator."*

- Code:

Adapted from JSR 370

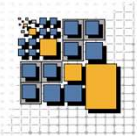
```
@Path("widgets")
public class WidgetsResource {

  @GET
  public WidgetList getWidgets() {…}

  @GET
  @Path("{id}")
  public Widget findWidget(@PathParam("id") String id)
  {return findById(id);}
}
```

→ Now, what if `widgets/5` and `widgets/6` is run concurrently against this resource?

Resource Interaction I



→ *How do you determine which requests to react to?*

Disregarding media types for the moment

- Selecting relevant HTTP methods via so-called request method designators
  - `@GET`, `@POST`, `@PUT`, `@DELETE`, `@HEAD`, `@PATCH`, `@OPTIONS` available as built-in designators
  - Custom designators possible
- Selecting relevant paths via URI templates
  - Mapping a class (resolved relative to deployment context and @ApplicationPath)

Adapted from JSR 370

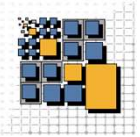
```
@Path("widgets")
public class Widget {...}
```

  - Mapping a method (resolved relative to the class mapping)

```
@Path("{id}")
public Widget findWidget() {…}
```

  - Use of regular expressions possible: `@Path("widgets/{path:.+}")`

Resource Interaction II



→ How do you access data in a resource method?

- Extracting HTTP protocol data
  - `@QueryParam`
  - `@PathParam`
  - `@CookieParam`
  - `@HeaderParam`
  - …
- Accessing form data

  → via @FormParam

```
@POST
@Consumes("application/x-www-form-urlencoded")
public void registerUser(
@FormParam("firstName") String firstName,
@FormParam("lastName") String lastName) {}
```

- (+ data sources of the respective environment)

Example:

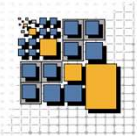
```
@Path("widgets")
public class WidgetsResource {
  @PathParam("id") String id;

  @GET
  public Widget getWidget() {
    return findById(id);
  }
}
```

Adapted from JSR 370

Adapted from JSR 370

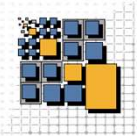
Resource Interaction III



→ *How do you produce results?*

- Return values
  - Actual return values
    → `void`, `Response`, or another Java type
  - HTTP status codes
    → default rules for `void`, `null`: 204
    → `status` property of the computed `Response`
- Exception Handling
  - Propagation to the client
    → `response` property of so-called `WebApplicationException`s or `ExceptionMappers`
  - Propagation to the surrounding container
    → means that container mapping rules can be used!
    - unchecked exceptions
    - exceptions that have not been mapped

Media Types



→ Request media types and request***ed*** media types affect the selection of resource methods!

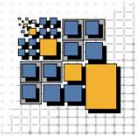
- Restriction of accepted / producible media types
  → via @Consumes, @Produces

```
@Path("widgets") @Produces("application/widgets+xml")
public class WidgetsResource {
@GET
public Widgets getAsXML() {...}
@GET @Produces("text/html")
public String getAsHtml() {...}
@POST @Consumes("application/widgets+xml")
public void addWidget(Widget widget) {...}
}
```

Adapted from JSR 370

- Per default, no restrictions are assumed
- Choices based on q-values and qs-values possible

Sub-Resource Locators



JSR 370: "[…]sub-resource locators return an object that will handle a HTTP request."

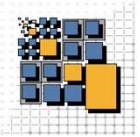
- public methods of resources annotated with `@Path` but not with a method designator

```
@Path("widgets")
public class WidgetsResource {
@GET @Path("offers")
public WidgetList getDiscounted() {...}
@Path("{id}")
public WidgetResource findWidget(@PathParam("id") String id) {
          return new WidgetResource(id);}}
```

```
public class WidgetResource {
public WidgetResource(String id) {...}
@GET
public Widget getDetails() {...}
}
```

Adapted from JSR 370

Providers I

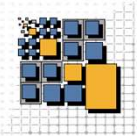


JSR 370: "Providers in JAX-RS are responsible for various cross-cutting concerns such as filtering requests, converting representations into Java objects, mapping exceptions to responses, etc. A provider can be either pre-packaged in the JAX-RS runtime or supplied by an application."

- Entity Providers: Serialization / Deserialization
- Exception Mapping Providers: Exception Handling
- Filters: Cross-cutting concerns
- Interceptors: Adaptations to Serialization / Deserialization
- Features: Runtime configuration
- Context Providers: Runtime adaptations

Does it make sense to look at Exception Mappers or Filters as Providers?

Providers II – Entity Providers



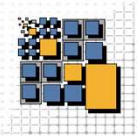
JSR 370: "Entity providers supply mapping services between representations and their associated Java types"

→ Basically provide app-specific serialization/deserialization

- Two types of providers are available
  - MessageBodyReaders (Serial-to-Java)
  - MessageBodyWriters (Java-to-Serial)
- Standard set of MessageBodyReaders/Writers
  - byte[] and java.lang.String for all media type configs (*/*)
  - Java streams for all media type configs (*/*)
  - XML (de-)serialization javax.xml.transform.Source and javax.xml.bind.JAXBElement for text/xml, application/xml, application/*+xml

incomplete list

Clients



Adapted from JSR 370

- Creating a basic request

```
Client client = ClientBuilder.newClient();
Response res = client.target("http://example.org/hello").
                  request("text/plain").get();
```

- Setting parameters / headers

```
Response res = client.target("http://example.org/hello")
                        .queryParam("MyParam","...")
                        .request("text/plain")
                       .header("MyHeader", "...").get();
```

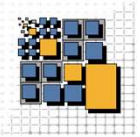
- Flexibly handling paths

```
WebTarget base = client.target("http://example.org/");
WebTarget hello = base.path("hello").path("{whom}");
Response res = hello.resolveTemplate("whom", "world").request("...").get();
```

- Creating typed requests

```
String res = client.target("http://www.kicker.de")
.request("text/plain").get(String.class);
```

Now, try yourself

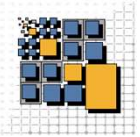


#bamberggutschein

- Let's return to our running example of the real-world coupon service.
  Create a service for registering merchants using JSON representations. To keep things simple,
  just assume a merchant-id, merchant-name and a list of coupon values as relevant fields.
- How would you design the following?
  - look up all merchants
  - look up a single merchant by id
  - look up coupon values of a merchant
  - create merchants
  - update merchants
  - delete merchants

Hint: Come up with HTTP methods, Path definitions and simple implementations

JAX-RS - Summary



- JAX-RS basically usable for true REST as well as HTTP-RPC
  → however, HTTP-RPC more relevant
- Requires more than WSDL/SOAP
  - Better understanding of HTTP
  - More low-level coding
- Requires less than WSDL/SOAP
  - Less restrictions on content and interaction styles
  - Less performance overhead
- Not covered in this lecture, but included in JAX-RS spec.
  - Filters and Interceptors
  - Validation
  - Asynchronous Processing
  - Lots of technical detail