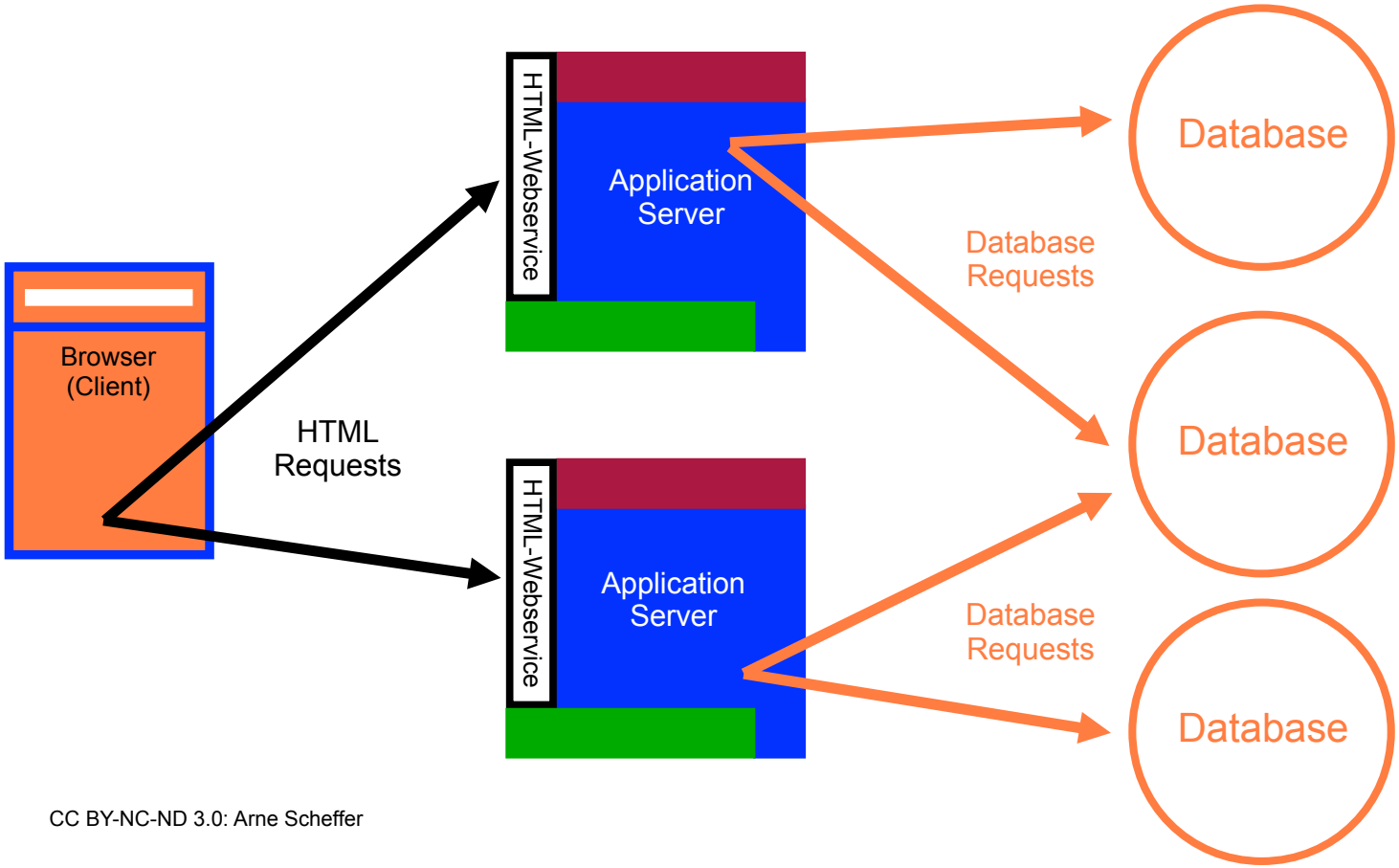
Enterprise Web Development (Java Enterprise Edition 6)

wissen.leben
WWU Münster

Dozent: Arne Scheffer

Wiederholung: Webservices – klassisch/bekannt



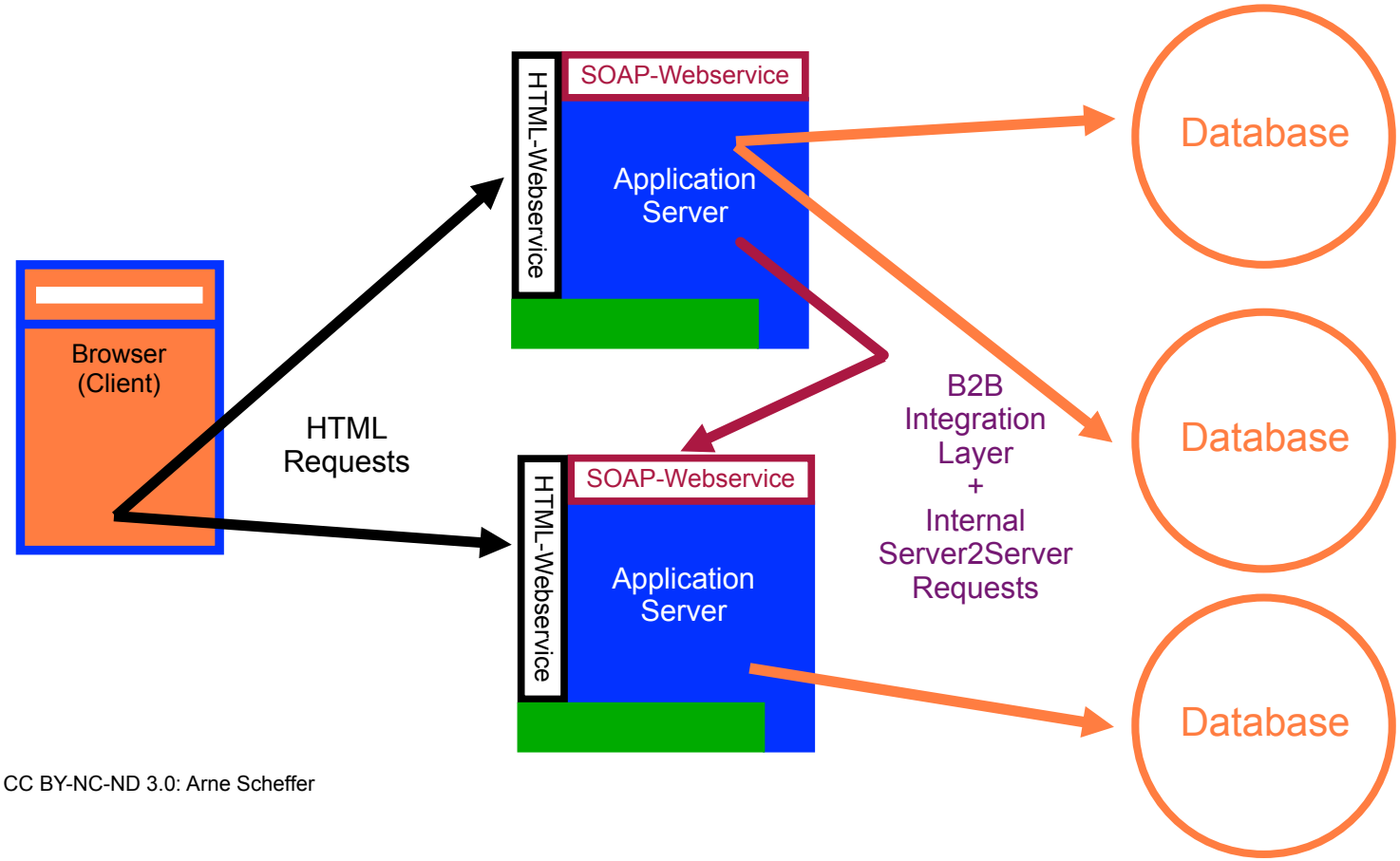
Wiederholung: Webservices – Wer mit wem ?

- Client-Server-Technik ?!
- Webservice = Webanwendung ?!

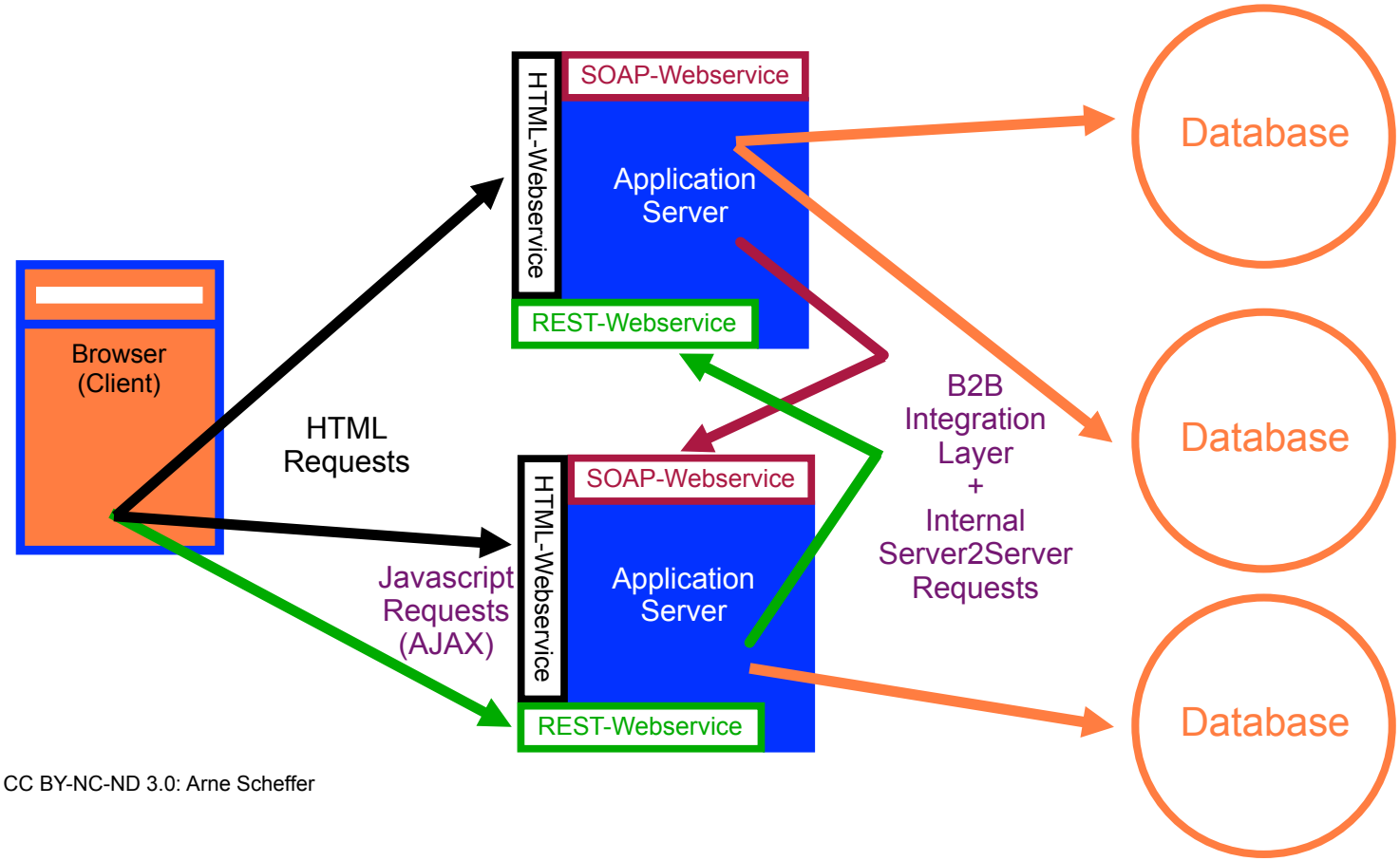
- Ein Webservice bietet
  - im Allgemeinen WENIGER als eine Webanwendung/-seite
    - keine optische Aufbereitung der Daten für den Nutzer
    - strukturierte Daten zur Weiterverarbeitung (z.B. XML)
  - zumindest Transparenz: „eine lose Kopplung“ in Bezug auf
    - verwendete Programmiersprache
    - verwendete Serverarchitektur
  - ein definiertes Interface zur Nutzung des Dienstes

- Hauptverwendung: B(usiness)2B(usiness)-Integration

Wiederholung: SOAP-Webservices



Heute: REST-Webservices



Wiederholung: Webservices in Java

- Ziel (falls die Erfordernisse das erlauben):
  - Java auf dem Server (, Java auf dem „Client“)
  - kein direktes Arbeiten mit dem XML-Code oder dessen Definitionen

- SOAP-Webservices
  - JAX-WS
    - JAXB
    - Web Services (Container-Implementierung)
    - WS-Metadata (Annotations)
    - JAXR (Registry-Handling)
- RESTful-Webservices
  - JAX-RS
    - JAXB

RESTful Webservices – W3C-Anmerkung

- REST-compliant Web services
  - primary purpose
    - to manipulate XML representations of Web resources
      - using a uniform set of "stateless" operations

- so zumindest ungenau

- eigene Definition für RESTful-Webservices mit JAX-RS:
  - purpose
    - to CRUD predominantly XML and JSON representations of Web resources
      - using the HTTP protocol to perform „stateless“ operations

RESTful Webservices - Ressourcen

- Ressourcen
  - werden alle dessen über einen Hyperlink erreichbaren Informationen genannt
    - erreichbar sind jeweils eine z.T. wählbare Repräsentation

- Beispiele
  - Buch
  - Bücher
  - Bücher, die eine bestimmte Bedingung erfüllen
  - Seiten
  - ...

RESTful Webservices - Repräsentationen

- wählbar
  - über mehrere Ressourcen mit entsprechenden Endungen im Hyperlink
  - über Content Negotiation
    - Auswahl der bestmöglichen Repräsentation für eine Information
    - vorwiegend gesteuert über den HTTP-Accept-Header
      - Beispiele für angebbare Content-Types
        - XML: z.B. application/xml
        - JSON: application/json,
        - Bilder: z.B. image/png
        - HTML: z.B. text/html

Eigenschaften von RESTful-Webservices

- Zustandslosigkeit
  - Server/Service hat kein Gedächtnis vergangener Requests
    - kein Session-Handling, Anfragen verteilbar
- einheitliches Interface
  - HTTP als dokumentbasiertes standardisiertes Interface
- Adressierbarkeit
  - Service sollte so adressierbar sein wie möglich
    - jede Ressource sollte durch eine URI erreichbar sein
- Verbundenheit
  - möglichst viele Wege zwischen den Ressourcen
    - die Ausgabe einer Ressource kann über URIs zu anderen „verbinden“

RESTful-Webservices: HTTP als Protokoll

- REST (Representational State Transfer) basiert auf der Art,
  - wie das Web funktioniert
    - erfunden von Roy Fielding, Mitdesigner des HTTP-Standards
- mehr Vorteile als bei SOAP
  - Eigenschaften des HTTP-Protokolls und dessen Umfeldes sind nutzbar
    - Caching
    - Authentifizierung (BASIC)
    - Content Negotiation
- Verwendung der bekannten HTTP-Befehle
  - GET
  - POST
  - PUT
  - DELETE

RESTful-Webservices: JAX-RS

- zentrale Annotationen:
  - @Path(„/pfad/“)
    - Beschreibung des Pfades der zu verwendenden URI
      - relativ: Klassen-Annotation konkateniert Methoden-Annotation
  - HTTP-Kommandos
    - @GET
    - @POST,
    - @PUT,
    - @DELETE
  - Beschreibung des verarbeiteten/ausgegebenen Contents
    - @Produces(„content type“),
    - @Consumes(„content type“)

Zurück zu unserem Beispiel: DruckerSpec

```
@XmlRootElement public class DruckerSpec {
  private boolean color;
  private String name=""; // Vermeidung von null
  public DruckerSpec() {} // Default-Konstruktor
  public DruckerSpec(String queue) {
      if (queue.equals("ein-ps"))                this.name = "HP Laserjet 9000";
      else if (queue.equals("cm8060"))       this.name = "HP MFP CM8060";
      else if (queue.equals("ZIVfarblaser")) this.name = "HP Laserjet CP6015";
      else if (queue.equals("hp650") || queue.equals("hp-foto"))
                                                 this.name = "HP DesignJet 5500 PS";
      else this.name = "not known";
      if (queue.equals("ein-ps")) this.color = false;   else   this.color = true;
   }
   public boolean isColor() { return color; }                    // Getter und Setter
   public String getName() { return name; }
   public void setColor(boolean color){ this.color = color; }
   public void setName(String name) { this.name = name; }}
```

Erstellen eines RESTful-Webservices:

- neues File → Web Services → RESTful Webservices from Patterns
  → Simple Root Resource
  → Paketnamen (z.B. Resources) vergeben
  → Pfadnamen (z.B. drucker) und Klasse (z.B. Drucker) vergeben
  → register as Application-Object
    - für den Web Services Manager

- abgucken und an eigene bekannte Bedürfnisse anpassen
  - Folgefolien

- Clean & Build → Run

- RESTful-Webservices → rechte Maustaste → Test RESTful-Webservices
  - (Zugriff erlauben)

RESTful-Webservices: Schablone für Klasse Drucker

```
@Path("drucker")
public class Drucker {
    @Context private UriInfo context;
    public Drucker() {}
     /** Retrieves representation of an instance of Resources.Drucker … */
    @GET @Produces("application/xml")
    public String getXml() { //TODO return proper representation object
                                throw new UnsupportedOperationException();    }

    /** PUT method for updating or creating an instance of Drucker
     * @param content representation for the resource
     * @return an HTTP response with content of the updated or created resource.
     */
    @PUT @Consumes("application/xml")
    public void putXml(String content) {
    }
}
```

RESTful-Webservices: Klasse Drucker

```
@GET
@Produces("text/plain")
@Path("/name/{queue}")
public String getName( @PathParam("queue") String queue ) {
    return new DruckerSpec(queue).getName(); }
@GET
@Produces("text/plain")
@Path("/color/{queue}")
public String isColor( @PathParam("queue") String queue ) {
    if (new DruckerSpec(queue).isColor()) return "true";
    return "false"; }
@GET
@Produces({"application/xml","application/json"})
@Path("{queue}")
public DruckerSpec getSpec( @PathParam("queue") String queue ) {
    return new DruckerSpec(queue); }
```

RESTful-Webservices: Einrichten eines Clients

- neues Projekt → neues File → Web Services → RESTful Java Client
- Facelet aus der letzten Vorlesung
- Rumpf von *Printer.java* aus der letzten Vorlesung
- Implementieren von *Printer.java*
  - mit Hilfe des kommentierten Client-Templates im Web Service Client
- bei der Verwendung „komplexer“ Klassen mittels JAXB
  - wie *DruckerSpec.java*
  - sind diese auch in das Client-Projekt zu kopieren

Facelet zur Anzeige aus der letzten Vorlesung

```
<h:form>
        <h:panelGrid columns="2">
          <h:outputLabel for="queue">Queue:</h:outputLabel>
          <h:inputText id="queue" value="#{printer.queue}"/>
        </h:panelGrid>
 </h:form>
 <h:outputText value="The name of #{printer.queue} is #{printer.name}."
                rendered="#{printer.queue ne ''}"/><br />
 <h:panelGroup rendered="#{printer.name ne 'not known'}">
        <h:outputText value="It is a color printer." rendered="#{printer.color}"/>
        <h:outputText value="It is a b/w printer." rendered="#{!printer.color}"/>
 </h:panelGroup>
```

Rumpf von Printer.java aus der letzten Vorlesung

```
@RequestScoped @Named
public class Printer {
  private String queue="not set";

    public String getName(){
    }
    public boolean isColor(){
    }

   public String getQueue() { return queue; }
   public void    setQueue(String queue) { this.queue = queue; }
}
```

Verwendung des kommentierten Client-Templates

```
public String getName(){
  DruckerWSClient client = new DruckerWSClient();
  String response = client.getName(queue);
  client.close();
  return response;
}

public boolean isColor(){
  DruckerWSClient client = new DruckerWSClient();
  String response = client.isColor(queue);
  client.close();
   if (response.equals("false")) return false;
  return true;
 }
```

Verwendung des kommentierten Client-Templates (JAXB)

```
public String getName(){
  DruckerWSClient client = new DruckerWSClient();
  DruckerSpec spec = client.getSpec_XML(DruckerSpec.class, queue);
  String response = spec.getName();
  client.close();
  return response;
}
public boolean isColor(){
  DruckerWSClient client = new DruckerWSClient();
  DruckerSpec spec = client.getSpec_XML(DruckerSpec.class, queue);
  Boolean response = spec.isColor();
  client.close();
  return response;
}                          /* DruckerSpec-Klasse muss beim Client vorliegen */
```

Demonstration: RESTful-Webservices from Entity Classes

- neues Projekt → neues File → Others → SQL-File (→ ausführen):

```
DROP TABLE zivdrucker;
CREATE TABLE zivdrucker(
 printer_queue VARCHAR(30) NOT NULL,
 printer_name VARCHAR(30) NOT NULL,
 printer_color smallint NOT NULL,
 primary key(printer_queue)
);
INSERT INTO zivdrucker VALUES ('ein-ps','HP Laserjet 9000', 0);
INSERT INTO zivdrucker VALUES ('cm8060','HP MFP CM8060', 1);
INSERT INTO zivdrucker VALUES ('ZIVfarblaser','HP Laserjet CP6015', 1);
INSERT INTO zivdrucker VALUES ('hp650','HP Designjet 5500ps', 1);
INSERT INTO zivdrucker VALUES ('hp-foto','HP Designjet 5500ps', 1);
```

- neues File → Persistence → Entity Classes from Database
- neues File → Web Services → RESTful Web Services from Entity Classes

Testen: RESTful-Webservices from Entity Classes

wieder mit:

- RESTful-Webservices → rechte Maustaste → Test RESTful-Webservices
  - (Zugriff erlauben)

- *zivdrucker* ausklappen und *{id} bzw. {printerQueue} anklicken*
  - als Format *application/xml* oder *application/json* auswählen
  - bei *{id} bzw. {printerQueue}* ist eine Queue für GET einzugeben

- ACHTUNG: dieser Webservice ist eine vollständige CRUD-Anwendung
  - nicht benötigte Funktionen (z.B. POST, PUT, DELETE) abklemmen
    - aus Sicherheits- und/oder Datenschutzgründen
    - z.B. erzeugte Methoden einschränken/Annotationen anpassen