Hochschule **Rosenheim**
University of Applied Sciences

Verteilte Verarbeitung

Kapitel 12

Verteilte Ressourcen:
REpresentational State Transfer

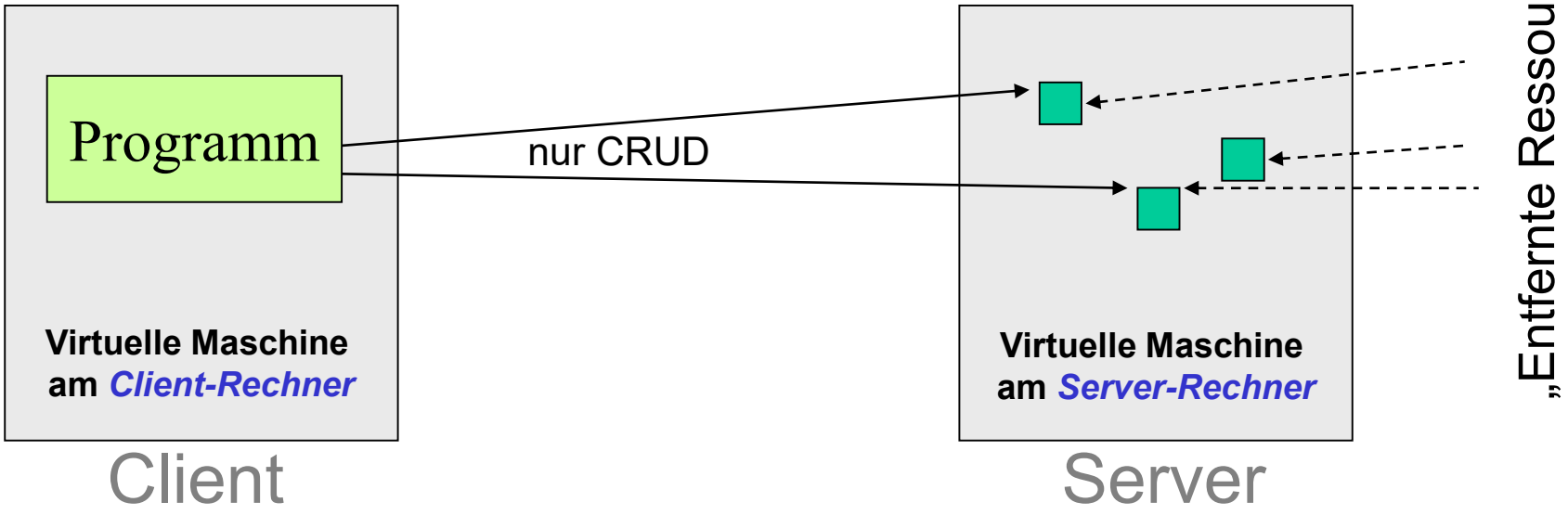
Was ist REST?

Roy T. Fielding

- ***REST* = *REpresentational State Transfer***
  - ***Architekturstil für verteilte Applikationen***
  - Idee stammt aus Dissertation von Roy Fielding (2000) „ Architectural Styles and the Design of Network-based Software Architectures”
  - Fielding hat HTTP mit spezifiziert (führende Rolle), Apache Foundation mit gegründet
- Idee: ***Ressourcen-Orientierte***-Architektur (ROA)
  - Jede Ressource (= Kunde, Konto, Vertrag, …) hat eindeutige URI
  - Manipulation der Ressourcen nur über HTTP
- Mittlerweile „Marktführer“ bei den Middleware-Ansätzen
  - Sehr dünner Protokoll-Stack im Vergleich zu SOAP-WebServices
  - Häufige Kombination: HTTP + TLS, JSON, OAuth2 / Basic-Auth

Basisschema für RESTful-WebService

- Ressource entspricht etwa einem C-Struct (Doofer Datencontainer)
- Aufrufe nur Anlegen, Ändern, Löschen, Suchen (CRUD)
  - Jede Ressource hat eindeutige URI
  - CRUD wird nur über HTTP erledigt
  - Jede Ressource hat mehrere Beschreibungsformen (XML, JSON, …)
- Wichtig: nur Bordmittel der Web-Techniken (HTTP, JSON, XML)



Grundprinzipien von REST

1. Ressourcen mit eindeutiger Identifikation
2. Verknüpfungen / Hypermedia
3. Standardmethoden von HTTP
4. Unterschiedliche Repräsentationen
   z.B. XML, HTML, JSON, …
5. Zustands-/Statuslose Kommunikation

Ressourcen

- haben eine eindeutige Identifikation (URI)
  - = ***Fachliche Identität***
  - z.B. **`https://inf-git.fh-rosenheim.de/api/v4/projects/3349`**
- können verknüpft werden (Hypermedia-Links)
- Können mehrere Darstellungen haben (XML, Text, JSON,…), Sie kennen JSON
- Beispiele:

  `http://swapi.dev/api/people/1`
  `http://swapi.dev/api/planets/`
  `https://api.spacexdata.com/v3/launches`
  `https://inf-git.fh-rosenheim.de/api/v4/projects/3349/issues`

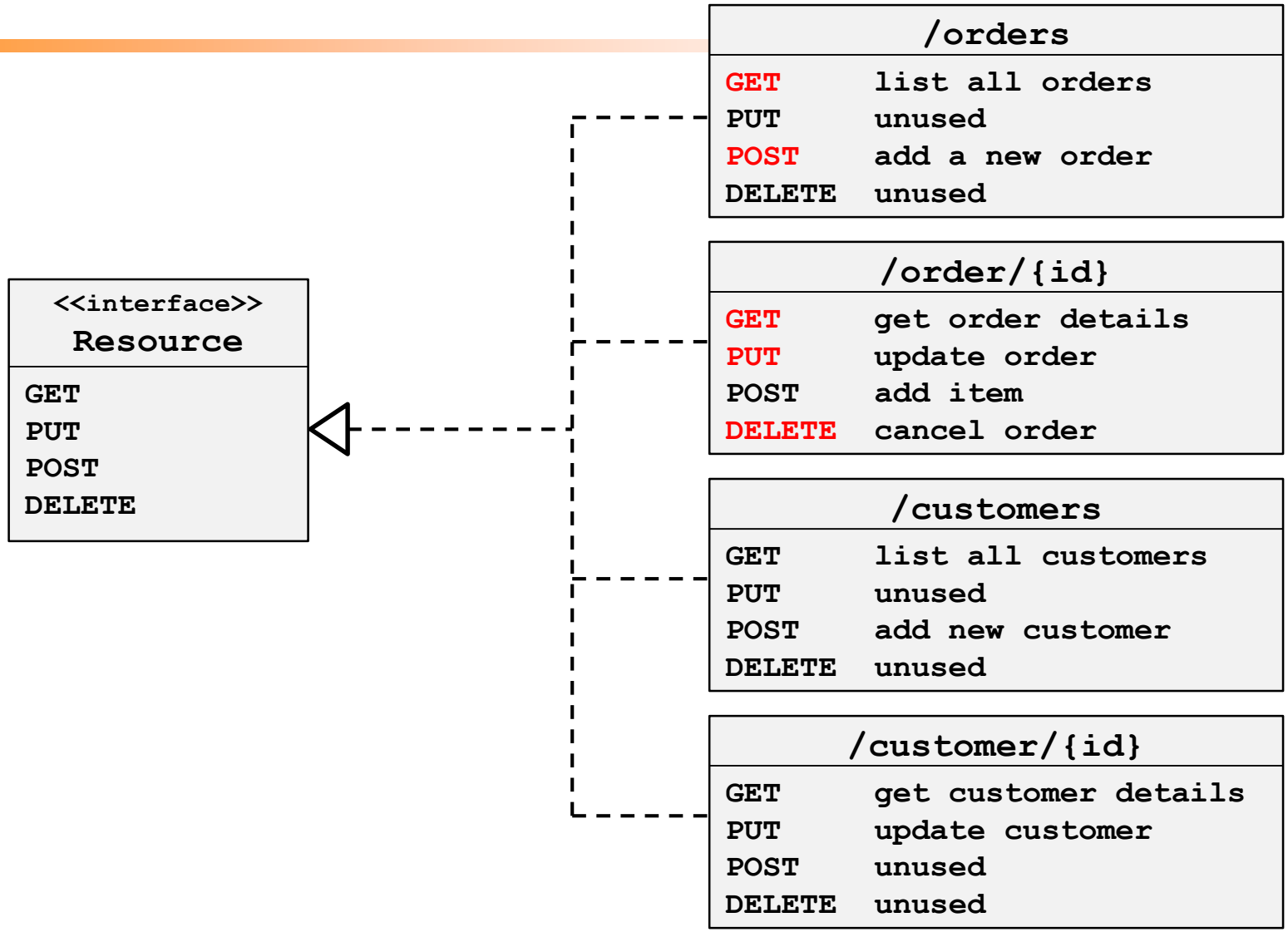
HTTP 1.1 -Verben

- HTTP bietet Methoden zum Anlegen, Ändern und Löschen von Daten
  - **`GET`** Laden von Daten, Query
  - **`PUT/PATCH`** Anlegen und Ändern von Daten
  - **`POST`** Anlegen von Unterknoten
  - **`DELETE`** Löschen von Daten
- Weitere Verben sind:
  - **`OPTIONS`** Welche Methoden stehen zur Verfügung?
  - **`HEAD`** Fordert HTTP-Header zu einer Ressource an.
- Daten werden jeweils durch URI identifiziert
  - Daten als Body des Requests bzw. des Responses

HTTP 1.1 Verben und CRUD

- **`GET, HEAD`** — ***READ***
  - Laden einer Ressource über Ihre URI vom Server
  - Query-Parameter möglich (z.B. "**`?name=hugo`**")
- **`POST`** — ***CREATE***
  - Erzeugung einer neuen Unter-Ressource (z.B. Bestell-Position)
- **`PUT / PATCH`**
  - Änderung einer bestehenden Ressource — ***UPDATE***
  - Erzeugung einer neuen Ressource mit bekannter URI
- **`DELETE`**
  - Entfernen von Ressourcen — ***DELETE***
- **`OPTIONS`**
  - Welche Operationen sind gerade erlaubt?

Eine RESTful Schnittstelle



Eine RESTful Schnittstelle

- *Container (z.B. Bestellung, Kundenliste, …)*
  - **`GET /container`** Liefert Inhalt des Containers
  - **`POST /container`** Item zum Container hinzufügen (Daten im Request enthalten)

    URI des Item enthalten im HTTP Response Header, etwa
    `http://host/container/item`
  - **`DELETE /container`** Container entfernen (inklusive items)
- Item
  - **`GET /container/item`** Item lesen
  - **`PUT /container/item`** Item ändern (Requet enthält geänderte Daten)
  - **`DELETE /container/item`** Item entfernen

Achtung: KEIN REST ist sowas:

- `GET http://swapi.dev/api/create/1`
- `GET http://swapi.dev/api/tryToFix?param=1&param2=7`

Parameter der Requests

- Identifikation einer Ressource
  - URL Parameter: … `/api/v4/projects/3349/issues/7`
- Parameter für einen Request:
  - Request-Parameter, z.B. für Queries
    z.B. …/`v3/launches?launch_year=2008&mission=Trail`
  - Auch genutzt für: Pagination / Query „Navigation“
    z.B. …`/v3/launches?limit=1&offset=5`
- Parameter im Body z.B. neue / geänderte Daten
- HTTP-Header
  - Optimistic-Locking
  - Authentisierung
  - Content-Negotiation
  - API-Versionierung

Header Parameter

- Content Negotiation
  - ACCEPT: z.B. **`application/json`** Server - > Client
  - CONTENT-TYPE z.B. **`application/json`** Client -> Server
- Authentisierung
  - AUTHORIZATION: z.B. bearer <credential>
  - Eigene Header-Parameter
- Optimistic Locking / Caching
  - ETAG
  - IF-MATCH
- Versionierung des APIs
  - Eigene Header Parameter

HTTP: Status Codes normale Aufrufe

- GET-Request
  - 200: OK alles prima, häufigste Antwort
  - 404: NOT FOUND (Ressource nicht gefunden)
- POST-Request
  - 201: CREATED neue Ressource angelegt
- PUT-Request
  - 200: OK Ändern erfolgreich
  - 201: CREATED neue Ressource angelegt (falls PUT == POST)
  - 404: NOT FOUND (Ressource nicht gefunden, also PUT != POST)
  - 409/412: CONFLICT Fehler beim Optimistischen Locking
- DELETE-Request
  - 204: NO-CONTENT
  - 409/412: CONFLICT Fehler beim Optimistischen Locking
  - 404: NOT FOUND (Ressource nicht gefunden)

HTTP: Status Codes normale Aufrufe

- Besonderheiten
  - 202: ACCEPTED Verarbeitung läuft noch, z.B. Videoupload
- Authentisierung
  - 401: UNAUTHORIZED, Authentifikation notwendig, z.B. Login oder andere Credentials
  - 403: FORBIDDEN, Credentials vorhanden aber nicht ausreichend, z.B. Login OK, aber Nutzer hat kein Recht die angegebene Ressource zu nutzen

Fehlerbehandlung

- 400: BAD REQUEST, Syntaxfehler im Request
  (z.B. falsche QueryParameter)
- 405: METHOD NOT ALLOWED, falscher Requesttyp (z.B. DELETE)
- 406: NOT ACCEPTABLE: Geforderter Datentyp kann vom Server nicht geliefert werden
- 415: UNSUPPORTED MEDIA TYPE: Gelieferter Datentyp wird vom Server nicht unterstützt
- 500: INTERNAL SERVER ERROR
  (Generische Fehlermeldung, z.B. Exception innerhalb des Servers)

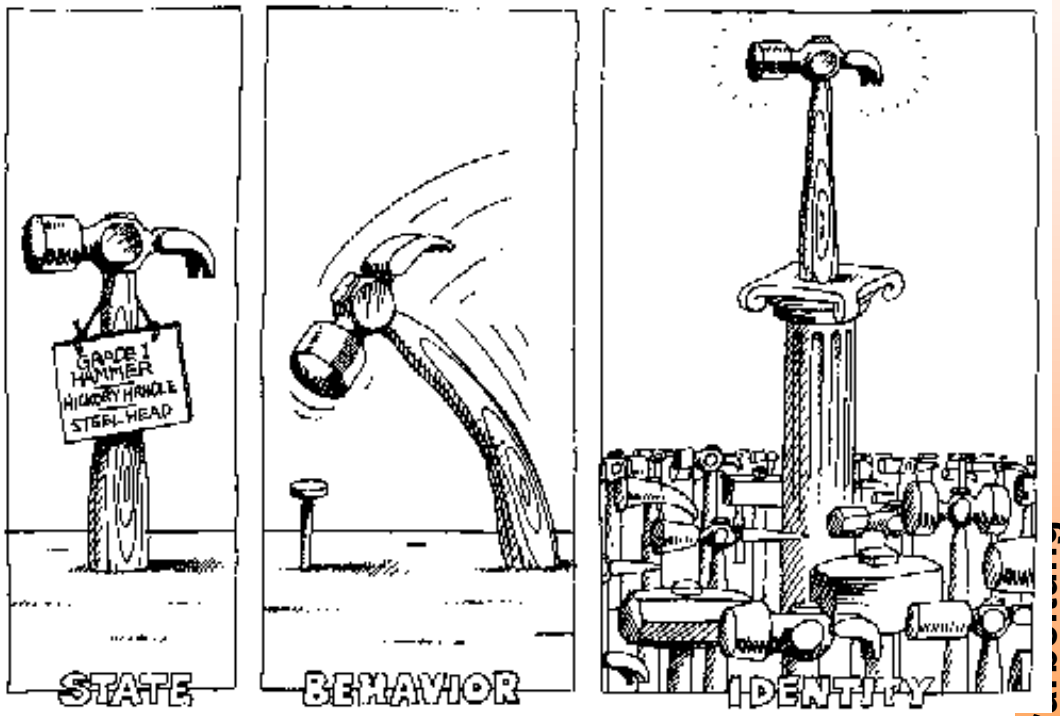
Weitere Themen für die nächsten Vorlesungen

- Versionierung eines APIs
- Optimistisches Locking
- Große Treffermengen (Pagination)
- Große Datenmenge (File Upload)
- Authentisierung mit Basic Auth und mit OAuth2

- Anhang

Was kennzeichnet ein Objekt

- **Zustand:**
  Ein Objekt hat Attribute, deren konkrete Werte sind der Zustand des Objekts
- **Verhalten:**
  Das Verhalten eines Objekts wird durch seine Operationen / Methoden beschrieben
- **Identität:**
  Jedes Objekt hat eine Identität, die es von anderen Objekten unterscheidet, sie ist unabhängig vom Zustand



Quelle der Grafik: G. Booch: Objektorientierte Analyse und Design, Addison Wesley 2001

Drei grundsätzliche Architekturoptionen Synchrone Kommunikation

- Verteilte Objekte
  - Technologien = RMI / .NET Remoting
  - Entfernter Zugriff auf verteilte ***Objekte*** auf Server(n)
  - Sehr verbreitet in den 1990ern (CORBA, OO-Hype)
- Verteilte Services
  - Technologien = WebServices (Basic Profile + WS*), RPC
  - Entfernter Zugriff auf Sammlungen von „***Prozeduren***“
  - Sehr verbreitet in den 2000ern (Service Orientierte Architekturen)
- ***Verteilte Ressourcen***
  - Technologie = REST (= HTTP)
  - Entfernter Zugriff auf ***Ressourcen*** (= Daten), nur CRUD-Operationen
  - Mit dem Aufkommen der Smartphones Marktführer