Verteilte Systeme

REST

by

Dr. Günter Kolousek

REST, HTTP und RESTful Webservices

- Representational State Transfer
- Ein Architekturstil[1] für verteilte Systeme
  - Zustand wird als Repräsentation von/zur Ressource übertragen
  - Client/Server, Request/Response, Cacheable
  - Satz von $\rightarrow$ Grundprinzipien
- HTTP ist eine weitgehend REST-konforme Implementierung
- RESTful Webservices = Webservices basierend auf REST mittels HTTP
- (für uns) Synonyme: REST, RESTful Webservice, RESTful HTTP (REST-konforme HTTP-Nutzung)
- Achtung: Vieles was sich RESTful nennt ist…

[1] Abstraktion spezifischer Architekturen

Warum?

- Loose Kopplung
  - $\rightarrow$ Identifikation mittels URI
  - $\rightarrow$ Uniform Interface
  - $\rightarrow$ Hypermedia
- Interoperabilität
  - HTTP, URI, JSON, XML, HTML
    - $\rightarrow$ unterschiedliche Repräsentationen
    - Verhandlung über Repräsentation (engl. negotiation mechanism)
- Performanz und Skalierbarkeit
  - interne Architektur (Implementierung)
  - Verteilungsarchitektur
    - Cache
    - $\rightarrow$ statuslose Kommunikation

Grundprinzipien

- Ressourcen mit eindeutiger Identifikation
- Hypermedia (Verknüfungen)
- Uniform Interface $\rightarrow$ Standardmethoden
- unterschiedliche Repräsentationen
- Statuslose Kommunikation

Ressourcen – Was?

- wichtig, um darauf Bezug zu nehmen $\rightarrow$ Hauptwörter!
- man sieht nie die Ressource $\rightarrow$ Repräsentation
  - Repräsentation ist eine Abstraktion
  - Ressource könnte man über die Menge ihrer Repräsentationen definieren
  - spiegelt vergangenen, aktuellen, gewünschten Status wider
- Ressource ’Person’
  - Repräsentationsformate könnten sein:
    - Textuell oder HTML
    - Bild oder Ton oder Video
- Andere Beispiele
  - Version 1.1 (oder letzte Version) einer Applikation
  - Liste der offenen Bugs
  - Kaffeemaschine im Raum 205

Ressourcen – Arten

- Primärressource: ”Bestellung”
- Subressource: ”Lieferadresse”
- Liste: ”Liste aller Bestellungen” (vgl. Selektion)
  - Filter: ”Liste aller stornierten Bestellungen”
  - Paginierung: ”Die ersten 10 offenen Bestellungen”
- Projektion: ”Name des Kunden” (aber nicht Bild, Video,…)
- Aggregation: ”Lieferaddressen aller offenen Bestellungen”
- Aktivität, wie z.B. zum Stornieren einer Bestellung

Ressourcen – Addressierbarkeit

- eindeutige Identifikation auf Basis der URI
  - URI, URL, URN
- Vorteile
  - URI bilden globalen Namensraum
  - keine eigene IDs notwendig
    - wie z.B. Kunden-ID, SSN,…
    - kein Rückschluss auf Implementierung, also eigener Datensatz in DB,…
  - wird weltweit verstanden
    - sowohl von Menschen als auch von Maschinen
- Beispiele
  - `http://htlwrn.ac.at/students/99001`
  - `http://htlwrn.ac.at/students/99001/address`
  - `http://htlwrn.ac.at/students`
  - `http://htlwrn.ac.at/students?vorname=max&nachname=mustermann`

Hypermedia

- Hypermedia As The Engine Of Application State (HATEOAS)
- Verknüpfungen zwischen Daten herstellen
  - Buchungszeile beinhaltet Link zu Produkt (Menge, Preis)
- den Anwendungsstatus steuern
  - Verknüpfungen geben mögliche Statusübergänge an
  - Verknüpfungen innerhalb des Response
    - z.B. Bestellung beinhaltet Link zur Stornierung, kein Link vorhanden $\rightarrow$ keine Stornierung möglich
- Vorteil
  - Verknüpfungen funktionieren anwendungsübergreifend!

Hypermedia – Beispiel

```
{
  ”customer”: {
    ”href”: ”http://example.com/customers/1503”
  },
  ”items” : [
    {
      ”product”: {
        ”href”: ”http://example.com/products/42”
      }
      ”amount”: 3
    }
  ],
  ”cancel”: {
    ”href”: ”./cancellation”
  }
}
```

Hypermedia – Formulare

- nicht nur Links…
  - auch hier steuert der Server, durch Inhalt der Formulare

```
<form action=”/reports” action=”POST”>
  <select name=”month”>
    <option value=”01”>01</option>
    ...
    <option value=”12”>12</option>
  </select>
  <input type=”number” name=”year”/>
  <select name=”status”>
    <option value=”cancelled”>cancelled</option>
    ...
  </select>
</form>
```

Uniform Interface

- Standardmethoden
  - `GET`, `POST`, `PUT`, `DELETE`, `PATCH`, `OPTIONS`
  - d.h. wie CRUD Operationen
  - anstatt anwendungsspezifische Operationen
    - wie z.B. `get_product_details(prod_id)`
- Eigenschaften
  - safe: keine Änderung am Server $\rightarrow$ caching!
  - idempotent: mehrfache Abarbeitung der Anforderung führt zu gleichem Ergebnis
- $\rightarrow$ gleiche Schnittstelle für **alle** Ressourcen!!

Uniform Interface

- Standardmethoden
  - `GET`, `POST`, `PUT`, `DELETE`, `PATCH`, `OPTIONS`
  - d.h. wie CRUD Operationen
  - anstatt anwendungsspezifische Operationen
    - wie z.B. `get_product_details(prod_id)`
- Eigenschaften
  - safe: keine Änderung am Server $\rightarrow$ caching!
  - idempotent: mehrfache Abarbeitung der Anforderung führt zu gleichem Ergebnis
- $\rightarrow$ gleiche Schnittstelle für **alle** Ressourcen!!
  - ja aber… wie z.B.
    - `cancel_order(order_id)`
    - `calculate_accounting_profit()`

    ?

Uniform Interface

- Standardmethoden
  - `GET`, `POST`, `PUT`, `DELETE`, `PATCH`, `OPTIONS`
  - d.h. wie CRUD Operationen
  - anstatt anwendungsspezifische Operationen
    - wie z.B. `get_product_details(prod_id)`
- Eigenschaften
  - safe: keine Änderung am Server $\rightarrow$ caching!
  - idempotent: mehrfache Abarbeitung der Anforderung führt zu gleichem Ergebnis
- $\rightarrow$ gleiche Schnittstelle für **alle** Ressourcen!!
  - ja aber… wie z.B.
    - `cancel_order(order_id)`
    - `calculate_accounting_profit()`

  ? $\rightarrow$ Ressourcen!

Uniform Interface – 2

- `GET`
  - safe, idempotent
  - `GET /products/4765`
  - `GET /products` → Liste aller Produkte, d.h. Übersicht mit allen *relevanten* Informationen
  - Return Code 200, 404 oder 400
- `HEAD`
  - safe, idempotent
  - liefert keinen Body
- `POST`
  - anlegen (bei Listenressource) oder beliebige Verarbeitung
  - nicht safe, nicht idempotent
  - `POST /products`
    - Return Code 201 & `Location` Header

Uniform Interface – 3

- PUT
  - anlegen (URI apriori bekannt) oder aktualisieren (vollständige Neuübertragung)
  - nicht safe, idempotent
  - `PUT /products/4765`
- `DELETE`
  - nicht safe, idempotent
  - `DELETE /products/4765`
  - Achtung: nochmaliges Löschen $\rightarrow$ 404, daher nicht idempotent, aber wenn in DB nur als ”deleted” markiert, dann…

Uniform Interface – 4

- PATCH
  - nicht safe, nicht idempotent
  - soll Änderungen übertragen
    - bezieht sich daher auf eine bestimmte ”Version” der Ressource $\rightarrow$ conditional request mit `IF-MATCH` Header und ETag
    - muss atomar erfolgen!
  - 2 Möglichkeiten
    - JSON Objekt nur mit Attributen, die geändert werden sollen. `null` für Attribute, die gelöscht werden sollen. Ähnelt PUT, wenn Ressource z.B. große Listen enthält!
    - Liste von Änderungen: JSON Objekt, das für jeden Pfad angibt, welche Änderung (”add”, ”remove”, ”change”) durchgeführt werden soll
  - Unterstützung dzt. noch mangelhaft

Uniform Interface – 5

- `OPTIONS`
  - safe, idempotent
  - liefert Informationen über Kommunikationsoptionen in Headerzeilen
  - `ALLOW`: Liste mit erlaubten Methoden für die spezielle Ressource
    - ohne `OPTIONS`: Methode verwenden, aber u.U. bekommt man ”405 Method Not Allowed”
  - `ACCEPT-PATCH`: Liste von Mediatypen für Patchdokumente

Uniform Interface – Return Codes

- 200 OK
- 201 CREATED
  - `LOCATION` Header!, bei `POST` auf Listenressource
- 204 NO CONTENT
  - z.B. bei `DELETE` oder `PUT`
- 400 BAD REQUEST
  - wenn Client eine Anfrage stellt, die nicht erfüllt werden kann, da ansonst ein ungültiger Zustand erreicht werden würde (Gültigkeitsüberprüfung fehlgeschlagen: fehlende Daten, Daten nicht im gültigen Bereich,…)
- 401 UNAUTHORIZED, 403 FORBIDDEN, 404 NOT FOUND
- 405 METHOD NOT ALLOWED
  - `ALLOW` Header ist im Response zu setzen
- weitere spezielle 4xx Codes

Repräsentationen

- Wie weiß Client wie Daten aufgebaut sind?
  - $\rightarrow$ HTTP Content Negotiation
  - Oft in JSON oder XML
  - Auch HTML $\rightarrow$ in Browser darstellbar!
- Beispiel
  - eigenes JSON Format

    ```
    GET /students/99001 HTTP/1.1
    Host: www.htlwrn.ac.at
    Accept: application/vnd.htlwrn.student+json
    ```
  - VCard

    ```
    GET /students/99001 HTTP/1.1
    Host: www.htlwrn.ac.at
    Accept: application/x-vcard
    ```

Statuslose Kommunikation

- Zwei Möglichkeiten
  - Client verwaltet Status
    - d.h. wird als Teile der Repräsentation vom Server zum Client übertragen
  - Server verwandelt Status in eine Ressource
    - $\rightarrow$ Ressourcenstatus
    - kann Referenz gespeichert werden, z.B. als Lesezeichen oder als Link in E-Mail versendet werden
- d.h. kein Sitzungsstatus $\rightarrow$ keine Sessions!
  - Kopplung zwischen Client und Server wird verringert
  - $\rightarrow$ Skalierbarkeit vereinfacht

Tipps

- Ressourcenamen: `/posts/42` anstatt `/api?type=post&id=23`
- feingranulare Ressourcen: `/posts/42/title`, `/posts/42/tags`
- Aktivitäten: `/user/123/orders/123/cancellation` ($\rightarrow$ PUT)
- XML vs. JSON
  - Client soll wählen können, z.B.
    - `/posts/42?type=xml` oder
    - besser: über `Accept: application/json`
  - default: JSON

Resource-Oriented Architecture

- ROA (im Gegensatz zu $\rightarrow$ SOA)
- Architektur basiserend auf REST
  - HTTP, HTML, JSON, XML,…
- Protokoll HTTP
- Ressource steht im Mittelpunkt, z.B. ”Kunde”
  - Finden geeigneter Abstraktionen von zentraler Bedeutung
- Hypermedia
- einheitliche Schnittstelle
- Eigene Middleware nicht notwendig, da
  - Proxy-Server, Gateways, Firewalls, Cache-Server