7363 - Web-basierte Anwendungen

Eine Vertiefungsveranstaltung
mit Schwerpunkt auf XML-Technologien

REST

***Re**presentational **S**tate **T**ransfer*

Ein Architekturstil zur Vereinfachung von Web Services

REST

- Ursprung von REST

  [1] Dissertation: Roy T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, UC Irvine, 2000.

  - http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

- Einführende Online-Literatur

  [2] Thomas Bayer, REST Web Services – eine Einführung, OIO, 11-2002

  - http://www.oio.de/public/xml/rest-webservices.htm

    (Vorsicht – der Artikel enthält einige Fehler!)

  [3] Ralf Wirdemann, Thomas Baustert, RESTful RAILS, b-simple.de, Hamburg, 01-2007

  - http://www.b-simple.de/download/restful_rails_de.pdf

REST

- Die Sicht von REST: Lauter Ressourcen!
  - Ressourcen sind Objekte, auf die sich per URL verweisen lässt, insbesondere:
    - XML-Dokumente
    - HTML-Dateien
    - RDF-Dateien, …
- REST, oder: Die Wiederentdeckung von HTTP
  - Von allen HTTP-Methoden verwenden Browser als auch Web Service-Komponenten praktisch nur GET und POST
  - REST greift das ursprünglich von HTTP verfolgte Konzept der verteilten Autorenschaft wieder auf.
  - REST verwendet dazu auch die HTTP-Methoden PUT, DELETE sowie HEAD und OPTIONS

REST

- HTTP und CRUD
  - Der Lebenszyklus eines Datenbankeintrags / einer Ressource, (meist aus objekt-relationaler Sicht) wird durch folgende Methoden realisiert:
    - **C** reate
    - **R** ead
    - **U** pdate
    - **D** elete
  - Direkte Entsprechung bei HTTP
    - C reate - POST <url>
    - R ead - GET <url>
    - U pdate - PUT <url>
    - D elete - DELETE <url>

REST

- Konsequenzen für webbasierte Anwendungen
  - Entfernung der Controller-Methoden aus den URLs
  - Enge Kopplung zwischen Controller-Methoden und HTTP-Methoden
  - Controller und Ressource verschmelzen zu einer Einheit
  - URLs werden zu reinen, einheitlich strukturierten *resource identifiers*

- URL-Analyse, gewöhnliche URLs
  - Szenario: Anlegen einer Bestellung

URLs enthalten ID, Controller und Methode!

GET *my_base_url*/cgi-bin/bestellen.pl?id=54321&action="create"

POST *my_base_url*/cgi-bin/bestellen.pl?id=54321&action="create"

HTTP-Methode

Zuständiger Controller

ID der Bestellung

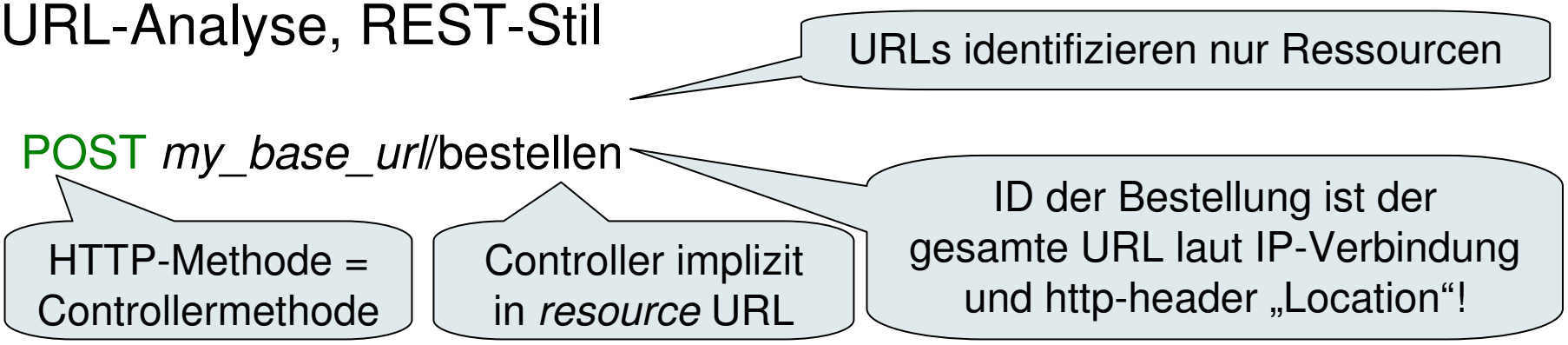
Aktion / Methode

  - GET oder POST? Beides könnte funktionieren!
    GET wäre aber schlechter Stil, da „Lesen“ keine Seiteneffekte haben sollte.

REST

- Beispiele
  - Abruf einer HTML-Seite
    - Gewöhnlich: GET *my_base_url*/index.html
    - REST-Stil: GET *my_base_url*/statische_seiten/1
  - Anlegen einer neuen Bestellung
    - Schlecht: GET *my_base_url*/cgi-bin/bestellen.pl?id=54321&action="create"
    - Besser: POST *my_base_url*/cgi-bin/bestellen.pl?action="create"
    - REST-Stil: POST *my_base_url*/bestellen
  - Löschen einer vorhandenen Bestellung
    - Gewöhnlich: POST *my_base_url*/cgi-bin/bestellen.pl?id=54321&action="delete"
    - REST-Stil: DELETE *my_base_url*/bestellen/54321

- URL-Analyse, REST-Stil



REST

- Beispiel
  - Quelle: http://rest.blueoxen.net/cgi-bin/wiki.pl?ResourcesAsObjects

```
C->S: GET http://example.com/lightbulb/on HTTP/1.1
C<-S: HTTP/1.1 200 OK
C<-S: Content-Type: text/plain
C<-S:
C<-S: false

C->S: PUT http://example.com/lightbulb/on HTTP/1.1
C->S: Content-Type: text/plain
C->S:
C->S: true
C<-S: HTTP/1.1 200 OK
```

REST

- Beispiel
  - Quelle: http://rest.blueoxen.net/cgi-bin/wiki.pl?ResourcesAsObjects

```
C->S: GET http://example.com/lights/ga_1_0_0 HTTP/1.1
C<-S: HTTP/1.1 200 OK
C<-S: Content-Type: text/plain
C<-S:
C<-S: off

C->S: PUT http://example.com/lightbulb/ga_1_0_0 HTTP/1.1
C->S: Content-Type: text/plain
C->S:
C->S: on  # oder: toggle
C<-S: HTTP/1.1 200 OK
```

REST vs. SOAP-RPC: Ein Versuch

- Beispiel Tilgungsplan, SOAP-RPC vs. REST
  - SOAP-RPC:

    POST *my_ws_url/*
    - Body enthält XML-Dokument (SOAP Envelope) mit allen Details der Anfrage, codiert gemäß SOAP-RPC
    - HTTP Response enthält Ergebnis (SOAP Envelope, analog)

  - REST-Stil:

    POST *my_ws_url*/tilgungsplaene # Location-Header liefert ID

    GET *my_ws_url*/tilgungsplaene/12345

    (DELETE *my_ws_url*/tilgungsplaene/12345)
    - Neuer Tilgungsplan-Auftrag wird zunächst angelegt, Inhalt von POST ist z.B. ein XML-Dokument, das die TP-Parameter enthält.
    - Der Server errechnet daraus den fertigen Tilgungsplan als XML-Dokument
    - Dieses Ergebnis-Dokument ist mit GET abrufbar
    - DELETE soll zum Ausdruck bringen, dass hier eine Ressource erzeugt wurde, um deren Löschen sich „jemand“ kümmern muss…

REST: Zusammengesetzte Objekte

```
GET /warenkorb/5873 HTTP/1.0
```

Warenkorb anfordern

Quelle: [2]

```
HTTP/1.1 200 OK
Content-Type: text/xml
Host: …
```

Ergebnis (Header ergänzt)

```
<?xml version="1.0"?>
<warenkorb xmlns:xlink="http://www.w3.org/1999/xlink">
  <kunde xlink:href="http://shop.oio.de/kunde/5873">5873</kunde>
  <position nr="1" menge="5">
    <artikel xlink:href="http://shop.oio.de/artikel/4501" nr="4501">
      <beschreibung>Dauerlutscher</beschreibung>
    </artikel>
  </position>
  <position nr="2" menge="2">
    <artikel xlink:href="http://shop.oio.de/artikel/5860" nr="5860">
      <beschreibung>Earl Grey Tea</beschreibung>
    </artikel>
  </position>
</warenkorb>
```

REST

- Zusammengesetzte Objekte
  - Kunden, Warenkörbe und Artikel werden als separate Ressourcen behandelt und geführt – d.h. mit eigenen URLs
  - Referenzierung von Objekten geschieht mittels deren URLs und dem Standard XLink (vgl. LV „XML-Technologie“)
  - Es besteht Konsistenz mit WWW-Gewohnheiten
    - Das XML-Dokument ließe sich mit CSS kombiniert direkt im Browser anzeigen
    - Die XLink-Elemente würde ein Browser beim Anklicken verfolgen (→ *drill down*)!

- Kommentare zum Beispiel
  - Warenkorb-ID und Kunden-ID sind identisch, ihre URLs nicht
  - Der Dokumententyp stellt einen Kompromiss zwischen Lesbarkeit für Menschen und maschineller Verarbeitbarkeit dar.
    - Einige Angaben sind redundant, z.B. Artikel- und Kundennummer
    - Das Element „Beschreibung“ ist für maschinelle Verarbeitung unnötig

  Anregung: Analoges Vorgehen bei Veranstaltung, Wettkampf, Teilnehmer?

REST

- Aktuelle Diskussionen
  - Ruby on Rails (RoR) 2.0:
    - ActiveService-Modul (XML-RPC, SOAP) wird ausgegliedert
    - Nachfolge: REST-Modul
    - Grund: Einfacher, besser mit RoR und Web-Prinzipien verträglich
  - Kritik an SOAP
    - SOAP hat sich von einem Protokoll zu einem Protokoll-Framework entwickelt.
    - SOAP versucht inzwischen derart viele Dinge zu ermöglichen, dass neue Verwirrung entsteht. Beispiel: RPC- versus Dokumentenmodus
    - SOAP-Typisierung ist inzwischen von XML Schema ersetzt worden. Dies war aber ein zentrales Anliegen, SOAP zu entwickeln.
    - SOAP funktioniert nicht so wie das WWW im allgemeinen (kein Ressourcen-Modell), REST sehr wohl.

- Aktuelle Diskussionen
  - Kritik & Fragen an REST
    - Sicherheitsaspekte, die über HTTPS hinausgehen, werden nicht berücksichtigt (vgl. späteres Kapitel „WS Security“)
    - Was wird aus WSDL und aus der darauf basierenden Code-Generierung?
    - Wie beschreibt man die auszutauschenden Ressourcen *einheitlich*? Wie beschreibt man insb. Nicht-XML-Ressourcen?
    - SOA auf Basis von SOAP & WSDL ist inzwischen in vielen Unternehmen eine akzeptierte Technologie. Sie erfährt umfangreichen Support u.a. von Microsoft, IBM und Sun. REST ist bei weitem nicht so weit verbreitet und unterstützt. (Wie schnell) wird sich das ändern?

- Weitere HTTP-Methoden für REST
  - HEAD
    - Fordert Metadaten zur angegebenen Ressource an
  - OPTIONS
    - Ermittlung der für eine Ressource verfügbaren Methoden

(leider noch keine Beispiele gefunden)