# Web-Engineering

## Web-Applikationen / REST



Hochschule Niederrhein
University of Applied Sciences
Elektrotechnik und Informatik
Faculty of Engineering and Computer Science

# Literatur

- Tilkov: "REST und HTTP", 2. Auflage 2011, d.punkt-Verlag

# Web-Applikationen: Architektur (1)

Aufteilung Verantwortlichkeiten, z.B.

- Server:
  - Datenhaltung
  - Applikationslogik
  - Auslieferung User Interface (Applikationslogik)
- Client:
  - User Interface (Applikationslogik)

# Web-Applikationen: Architektur (2)

Zustände:

- Server:
  - Nächste erwartete Operationen gemäß Applikationslogik
  - Benutzerspezifisch / Sitzungsspezifisch
- Client:
  - User-Interface-Zustände (lokal)
  - Problem: Synchronisation mit Serverzuständen
- Probleme:
  - passt nicht zu zustandslosem HTTP
  - Enge Kopplung

# Web-Applikationen: Architektur (3)

- Ziele:
  - Kopplung minimieren
  - Zustände einfacher verwalten
- Ideen:
  - Konzept „Ressource“ einführen
    - Definiert den Zustand
    - Wird ausgetauscht in unterschiedlichen Repräsentationen
  - Einheitliche Methoden => HTTP-Methoden
  - Hyperlinks ermöglichen Zustandswechsel

# Web-Applikationen: Architektur (4)

Konsequenzen:

- Server verwaltet keine Sitzungszustände
- Server verwaltet stattdessen Ressource-Zustände
- Zustand wird mit Ressource zum Client übertragen
- Architektur von Web-Applikation:
  - Erzeugen, Ändern und Präsentieren von Ressourcen
  - Zustand: Ressource im Client bearbeiten
  - Zustandswechsel: per Hyperlink zu einer anderen Ressource

Web als Beispiel: Kollektion von Ressourcen

# REST: Architekturansatz nach R. Fielding

Architekturansatz

- Beschrieben in der Dissertation von Roy Fielding

**REST = "Representational State Transfer"**

- Repräsentation
  - Versetzt Client in einen Zustand
- Zustand
  - Durch angeforderte Ressource, die als Repräsentation geliefert wird
- Übergang
  - In einen neuen Zustand durch Anforderung einer neuen Ressource (z.B. per Hyperlink)

# REST: Eigenschaften (1)

- resource identification
  - Eindeutig identifizierbare Ressourcen
- uniform interface
  - Standardmethoden
  - Standardrepräsentationen
  - Ressourcen mittels der Repräsentationen bearbeiten
- self-descriptive messages
  - Standard-Nachrichten
  - Gemeinsames (?) Verständnis der Kommunikationspartner

# REST: Eigenschaften (2)

- HATEOAS:
  - „Hypermedia as the engine of application state“
  - Weder Client noch Server verwalten einen Session-Zustand
  - Alle zustandsrelevanten Daten in den HTTP-Nachrichten
    - URI
    - Angaben in Message-Header und Message-Body
    - Enthaltene Hypermedia-Angaben (Hyperlinks)
- Fazit:
  - Standard-Methoden, -Protokolle, -Präsentationen nutzen
  - Zustandslosigkeit, lose Kopplung
  - Identifizierbare Ressourcen

# REST: Ressourcen

- Alles, was durch eine URI **eindeutig** identifizierbar ist
  - Allein durch Protokoll, Server, Pfad
  - Statische Webseiten
  - Kollektion von Ressourcen
  - Nicht nur (Daten-)Objekte im klassischen Sinne, auch z.B.:
    - Zwischenergebnisse (z.B. bei Bestellvorgang)
    - Endergebnisse (z.B. Rechnung) mit Verweisen auf andere Ressourcen
    - Transaktionen (Zusammenfassung von Operationen)
    - Applikationsübergreifende Methoden
- Repräsentationen
  - Standardformen verwenden
  - „Content Negotiation“: Abstimmung zwischen Client und Server über Form
  - Bearbeiten, d.h. keine „direkte“ Bearbeitung einer Ressource

# REST: uniform interface (1)

Generische Methoden: "uniform interface"

- Auf Ressourcen anwenden
- Mit HTTP-Befehlen/verbs implementieren, z.B.
  - GET: auf Ressource zugreifen (select)
  - POST: neue Ressource erzeugen (insert) (spezielle Operationen)
  - PUT: Ressource ändern (update)
  - DELETE: Ressource löschen (delete)
  - HEADER: Metainformationen zur Ressource
  - OPTIONS: Metainformationen zum Server

  (es gibt auch andere Einteilungen bei PUT/POST!)

# REST : uniform interface (2)

- Anfragen (Requests) verwenden dann keine Parameter mehr
  - Statt : GET /person/?id=4711&action=save&Name= …
  - Jetzt: PUT /person/4711
    - Daten im Request-Body
  - Statt: GET /person/delete/?id=4711
  - Jetzt: DELETE /person/4711

    (Eine numerische Objekt-Id ist nicht zwingend erforderlich)

# REST : uniform interface (3)

- HATEOAS: *(siehe: REST Eigenschaften (2) )*
  - „Hypermedia as the engine of application state“
- Hyperlinks:
  - Globales Namensschema => alle Ressourcen verknüpfbar
  - Steuerung Zustand:
    - Server teilt dem Client über Hyperlinks mit, welche Aktionen er als nächstes ausführen kann
    - Applikation erhält neuen Zustand, indem der Client einem Hyperlink folgt

# Implementierungsmöglichkeiten

- Bei CherryPy:
  - "MethodDispatcher" verwenden
    - Klassen erhalten das Attribut "exposed"
    - Methoden GET, POST, PUT, DELETE (ggf. weitere)
      - Alle Methoden in Großbuchstaben werden veröffentlich und auf HTTP-Verbs abgebildet
- Bei anderen Ansätzen
  - Aus Umgebungsvariablen ermitteln, welches HTTP-Verb verwendet wird