Verteilte Systeme

Prof. Dr. Martin Becke

CaDS - HAW Hamburg

Version 0.9

Inhalt

Topologien

Übersicht

- One-to-All
- Tree-based (Spanning Tree)
- Flooding
- Gossip

Protokolle

Produktive Protokolle - Beispiele

- TCP/IP
- MQTT
- HTTP

Protokolle

Basic Layers

- Low-Level-Network-Programming
- High-Level-Network-Programming
- Prompt Engineering (?)

Kommunikation

Eigenschaften

- Skalierbarkeit
- Fehlertoleranz
- Ausfallsicherheit
- Konsistenz
- Synchronisation
- Bandbreite, Latenz, Fehlerrate
- Kopplung

Auswahl nach Zerlegungsmethode

Ressourcen-orientierte

- HTTP für die Ressourcen-orientierte Zerlegung
- de-facto die Basis für die Restful API

```
POST /ressource HTTP/1.1
Host: beispiel.com
Content-Type: application/json
Content-Length: 25
{
  "name": "Beispielname"
}
```

Listing 1 – Ressource anlegen (POST)

Auswahl nach Zerlegungsmethode

Funktionale Zerlegung

- Simple Object Access Protocol (SOAP) als Beispiel für

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:per="http://example.com/person">
   <soapenv:Header/>
   <soapenv:Body>
      <per:CreatePersonRequest>
         <per:firstName>John</per:firstName>
         <per:lastName>Doe</per:lastName>
      </per:CreatePersonRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

REST

Eigenschaften

- Stateless
- Client-Server-Architektur
- Cachefähig
- Einheitliche Schnittstelle
- Ressourcenorientierung

RESTful API

Eigenschaften angelehnt an REST

- Einfachheit
- Zustandslos/ Skalierbar
- Interoperabilität (Client-Server-Architektur)
- Flexibilität
- Cachefähig
- Einheitliche Schnittstelle (HATEOAS)
- Ressourcenorientierung

RESTful API

HATEOAS - Großes Missverständniss

- Unklare Dokumentation
- Fehlende Standardisierung
- Begrenzter Einsatz
- Frontend-Technologien und Frameworks

RESTful API

HATEOAS - Beispiel

```
{
  "id": 1,
  "status": "off",
  "_links": {
    "self": {
      "href": "https://api.example.com/lamps/1"
    },
    "turnOn": {
      "href": "https://api.example.com/lamps/1/actions/turnOn",
      "method": "PUT"
    },
    "setBrightness": {
      "href": "https://api.example.com/lamps/1/actions/setBrightness",
      "method": "PUT"
    },
    "setColor": {
      "href": "https://api.example.com/lamps/1/actions/setColor",
      "method": "PUT"
    }
  }
}
```

RESTful API

Richardson Maturity Model

- Level 0 - RPC over HTTP
- Level 1 - low level REST
- Level 2 - de-facto standard
- Level 3 - HATEOAS

Richardson Maturity Model

Level 0

Einzige URI : https ://api.example.com/actions

Aktionen werden durch unterschiedliche Parameter im Anfrage-Body bestimmt.

```
{
  "action": "getLamp",
  "lampId": 1
}
```

Listing 4 – Level 0

Richardson Maturity Model

Level 1

URI je Ressource : https ://api.example.com/lamps/1

Aber um die Lampe ein- oder auszuschalten, wird immer noch POST verwendet :

```
{
  "action": "turnOn"
}
```

Listing 5 – Level 1

Richardson Maturity Model

Level 2

URI je Ressource : https ://api.example.com/lamps/1

Verwendung von standardisierten HTTP-Methoden.

- Um Informationen über die Lampe abzurufen : GET https ://api.example.com/lamps/1
- Um die Lampe einzuschalten : PUT https ://api.example.com/lamps/1/actions/turnOn
- Um die Lampe auszuschalten : PUT https ://api.example.com/lamps/1/actions/turnOff

Message Broker Protokolle

IETF Kontext

- AMQP
- STOMP
- MQTT

Message Broker Protokolle

Beispiel MQTT

- Leichtgewichtiges Publish-Subscribe-Protokoll
- Für eingeschränkte Umgebungen und Geräte mit begrenzter Rechenleistung
- Binäres Protokoll auf Basis von TCP/IP
- Gut geeignet ist für das Internet der Dinge (IoT)
- Geringe Latenz und geringer Bandbreitenverbrauch
- Geräte-zu-Geräte-Kommunikation

Message Broker Protokoll

Message Broker

- RabbitMQ
- Beispiel eines Vergleichs `https://www.confluent.io/blog/kafka-fastest-messaging-system/`
- ZeroMQ (Broker-less)
- Beispiel MQTT Broker `https://github.com/hobbyquaker/awesome-mqtt`
- IBM MQ (Closed Source)
- HiveMQ (as a Service -Germany)

Message Broker Protokolle

Beispiel Apache Kafka

- Verteiltes Streaming-System
- Verarbeitung großer Datenmengen und Hochdurchsatz
- Skalierbares und fehlertolerantes System
- Verarbeitung und Speicherung von Datenströmen in Echtzeit
- Großen Datenmengen
- Kommunikation in groß angelegten, verteilten Anwendungen

Message Broker Protokolle

Apache Kafka Alternativen

- Apache Pulsar
- NATS Streaming
- Amazon Kinesis
- Google Cloud Pub/Sub
- Redpanda
- Beispiel eines Vergleichs `https://www.confluent.io/blog/kafka-fastest-messaging-system/`

Message Broker

Properitäre Alternative : IBM MQ

- Message Queues
- Message Channel Agents
- Queue Manager
- Clients

Message Broker

Priorizierung Nachrichten

- Prioritätsniveaus
- Routing-Regeln
- Software-Tools

Message Broker

Herausforderungen

- Komplexität
- Ausfallsicherheit
- Skalierbarkeit
- Leistung
- Sicherheit
- Kompatibilität
- Wartung und Support

RESTful API und States

Idee

- REST ist ein resourcenorientiertes Designparadigma, das die Statemaschine in den Fokus stellt
- Fast ausschließlich jedes Programm besitzt eine Statemaschine
- States besitzen Eigenschaften und die Transitionen
- Es kann auch als Abstraktion bestehen

RESTful API und States

Motivation

- Konsistenz und Einfachheit
- Entkopplung von Client und Server
- Zustandslosigkeit
- Skalierbarkeit und Leistung
- Erweiterbarkeit und Anpassungsfähigkeit
- Einheitliche Schnittstelle

RESTful API und States

Fallbeispiel

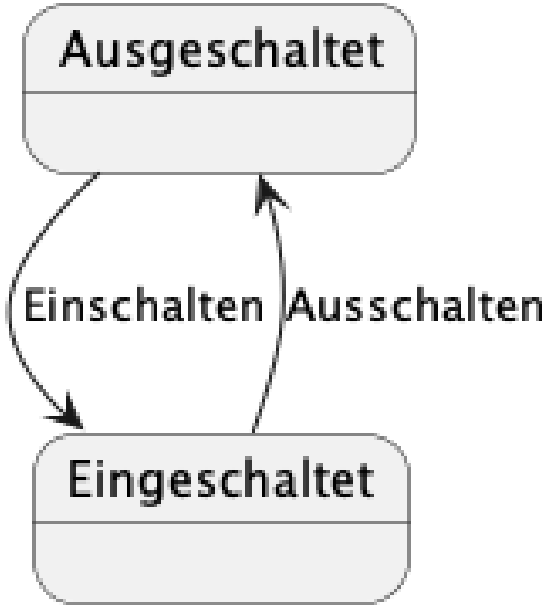

Figure – Einfache FSM Lampe

RESTful API und States

Fallbeispiel - Zustände abbilden

Ausgeschaltet : /lamp/off
Eingeschaltet : /lamp/on

HTTP-Verben für Zustandsübergänge :
Einschalten : PUT /lamp/on
Ausschalten : PUT /lamp/off

RESTful API und States

Fallbeispiel - GET /lamp

```
{
  "state": "Eingeschaltet",
  "links": [
    {
      "rel": "self",
      "href": "/lamp/on",
      "method": "GET"
    },
    {
      "rel": "Ausschalten",
      "href": "/lamp/off",
      "method": "PUT"
    }
```

RESTful API und States

Fallbeispiel - LampModel

```
class LampModel:
    def __init__(self):
        self.state = "AUS"

    def toggle(self):
        if self.state == "AUS":
            self.state = "EIN"
        else:
            self.state = "AUS"

    def get_state(self):
        return self.state
```

RESTful API und States

Fallbeispiel - LampController

```
class LampAdapter(Resource):
    def __init__(self, controller):
        self.controller = controller

    def get(self):
        response = {
            "state": self.controller.model.get_state(),
            "_links": {
                "self": {"href": "/lamp"},
                "on": {"href": "/lamp/on"},
                "off": {"href": "/lamp/off"},
            }
        }
        return jsonify(response)

    def put(self):
        action = request.form.get("action")

        if action == "on":
            self.controller.on()
        elif action == "off":
            self.controller.off()
```

RESTful API und States

Fallbeispiel - LampController

```
from flask import Flask, jsonify, request
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

lamp_model = LampModel()
lamp_controller = LampController(lamp_model)
lamp_adapter = LampAdapter(lamp_controller)

api.add_resource(lamp_adapter, "/lamp", "/lamp/on",
    "/lamp/off")
```

RESTful API

Alternativen/Beispiele

- CoAP
- XMPP
- DDS (Realtime Publish/Subscribe)
- OPC-UA (Industrielles Kommunikationsprotokoll)
- ROS
- Message Broker Protokolle

Message Delivery

QoS

- maybe (MQTT QoS 0)
- at-least-once (MQTT QoS 1)
- at-most-once
- exactly-once (MQTT QoS 2)

Message Delivery

exactly-once

- Zwei-Generäle-Problem (Gleich mehr)
- Unzuverlässige Kommunikation
- Unvorhersehbare Systemausfälle
- message processing vs message delivery

Byzantinische Generäle

Idee

- Zwei-Generäle-Problem (Gruppenarbeit)

Heartbeat

Idee

- Ein periodisches Signal
- Verfügbarkeit und Erreichbarkeit
- Heartbeats in verteilten Systemen haben mehrere Funktionen

Heartbeat

Ziel

- Fehlererkennung
- Synchronisation
- Lastverteilung

Heartbeat

Nachteile

- Zusätzlichen Netzwerkverkehr
- False positives
- Skalierbarkeit

Heartbeat

Konzepte

- Zentral vs. de-zentral
- Hierarchisch vs flach
- In-Band- und Out-of-Band-Kommunikation
- Priorisierung von Heartbeat-Prozessen
- Adaptiven und lernenden Heartbeat-Systeme

Multicast

Idee

- Ansatz für effiziente und skalierbare Übertragung
- Schont Netzwerkressourcen
- Problem : Mapping auf physikalisches Netz

Multicast

Ansätze

- Application-Level Tree-Based Multicasting
- Datenbankreplikationen
- Redundanz
- Flooding-basiertes Multicasting
- Gossip-basiertes Multicasting

Serialisierungsformate

Herausforderungen

- Protocol Buffers
- MessagePack
- JSON
- XML

Transport

Ideen

- Push vs Pull
- Daten vs. Funktionen

Namensräume

Begriffe

- Name
- ID
- Adresse

Service Discovery

Architektur I

- Zentralen Architektur
- Verteilte Service-Discovery-Architektur
- Hierarchische Service-Discovery-Architektur

Service Discovery

Architektur II

- Client-seitige Service-Discovery-Architektur
- Server-seitige Service-Discovery-Architektur

Namensräume

Strukturen

- Flacher Namensraum
- Hierarchischer Namensraum
- Attributbasierter Namensraum

Naming

UUID

- Version 1 (zeitbasiert)

$$= \text{Zeitstempel} + \text{Sequenznummer} + \text{MAC-Adresse} \quad (1)$$

- Version 2 (DCE Security)

$$= \text{Zeitstempel} + \text{Sequenznummer} + \text{MAC-Adresse} + textDCE \quad (2)$$

- Version 3 (MD5-Hash)

$$= \text{MD5-Hash}(\text{Namensraum-Identifikator} + \text{Name}) \quad (3)$$

- Version 4 (zufällig)
- Version 5 (SHA-1-Hash)

Locator/Identifier

Idee

- Identifikator ist ein eindeutiges Label oder eine Kennung
- Lokator ist eine Information, für physischen Ort oder Adresse

Locator/Identifier

Protokolle

- LISP
- HIP
- ILA

Anwendung :

- SIP
- Mobile IP