Theorie und Praxis einer JSON-RPC-basierten Web-API

Christian Krause

Christian.Krause@raritan.com

Raritan Deutschland GmbH

Chemnitzer LinuxTage 2015

Gliederung

Problemstellung

- Entwicklung eines neuen Produktes
  - PDU - Power Distribution Unit
  - Messen von Strom, Spannung, Energie und Leistung
  - Schalten der Ausgänge
  - Unterstützung einer Vielzahl von Sensoren und Aktoren
- Gesucht: Netzwerk-API
  - Abfrage aller Daten, Steuerung, Konfiguration
  - interne Nutzung (Web-Oberfläche)
  - Skripte für Test, Produktion und Installation
  - Integration

Anforderungen

- Nutzung von existierenden Standards
- Verschlüsselung, Authentifizierung
- Implementierbarkeit
  - Eignung für eingebettete Systeme
  - Eignung für effiziente IPC
  - Einfache Generierung einer Vielzahl von Sprach-Anbindungen
- Wartbarkeit / Debugging
  - Lesbarkeit des „On-Wire“-Protokolls
  - Nutzung mit Standard-Tools
- Formale Definition

Gewählte Technologien

- einfache, objektorientierte IDL
- JSON-RPC
- an REST-API angelehntes URI-Schema
- HTTP
  - HTTPS zur Verschlüsselung
  - Basic Auth zur Authentifizierung

Remote Procedure Call - RPC

- Aufruf von Funktionen
- über Prozess- und Systemgrenzen hinweg
- Client-/Server-Architektur
- Beispiele
  - CORBA
  - XML-RPC
  - D-Bus
  - SOAP

Interface Definition Language - IDL

- Formale Beschreibung einer Schnittstelle
  - Strukturen und Enumerationen
  - Klassen/Interfaces
  - Methoden
- Unabhängig von einer Programmiersprache
- Nutzung
  - Erstellung von Sprachanbindungen
  - Dokumentation

Beispiel

```
interface Outlet extends EDevice {
    enumeration PowerState {
        PS_OFF,
        PS_ON
    };
    int setPowerState(in PowerState pstate);
};
```

JavaScript Object Notation - JSON

- textbasiertes Datenaustausch-Format
- Basis: JavaScript

Datentypen

```
obj = { "string as key" : "a string value",
        "some integer" : 31,
        "some bool" : true,
        "an undefined value" : null,
        "an array type" :
            [ 1, false, { "key of inner object" : 42 } ]
      }
```

JSON-RPC

- RPC-Protokoll
- Serialisierung: JSON
- Client: Request-Objekt
  - Methode (String)
  - Parameter (Objekt)
  - ID
- Server: Response-Objekt
  - Resultat (Objekt) oder
  - Fehler (Objekt)
  - ID

Request

```
{ "jsonrpc": "2.0",
  "method": "setPowerState",
  "params": {
    "pstate": 1
  },
  "id": 23 }
```

Response

```
{ "jsonrpc": "2.0",
  "result": {
    "_ret_": 0
  },
  "id": 23 }
```

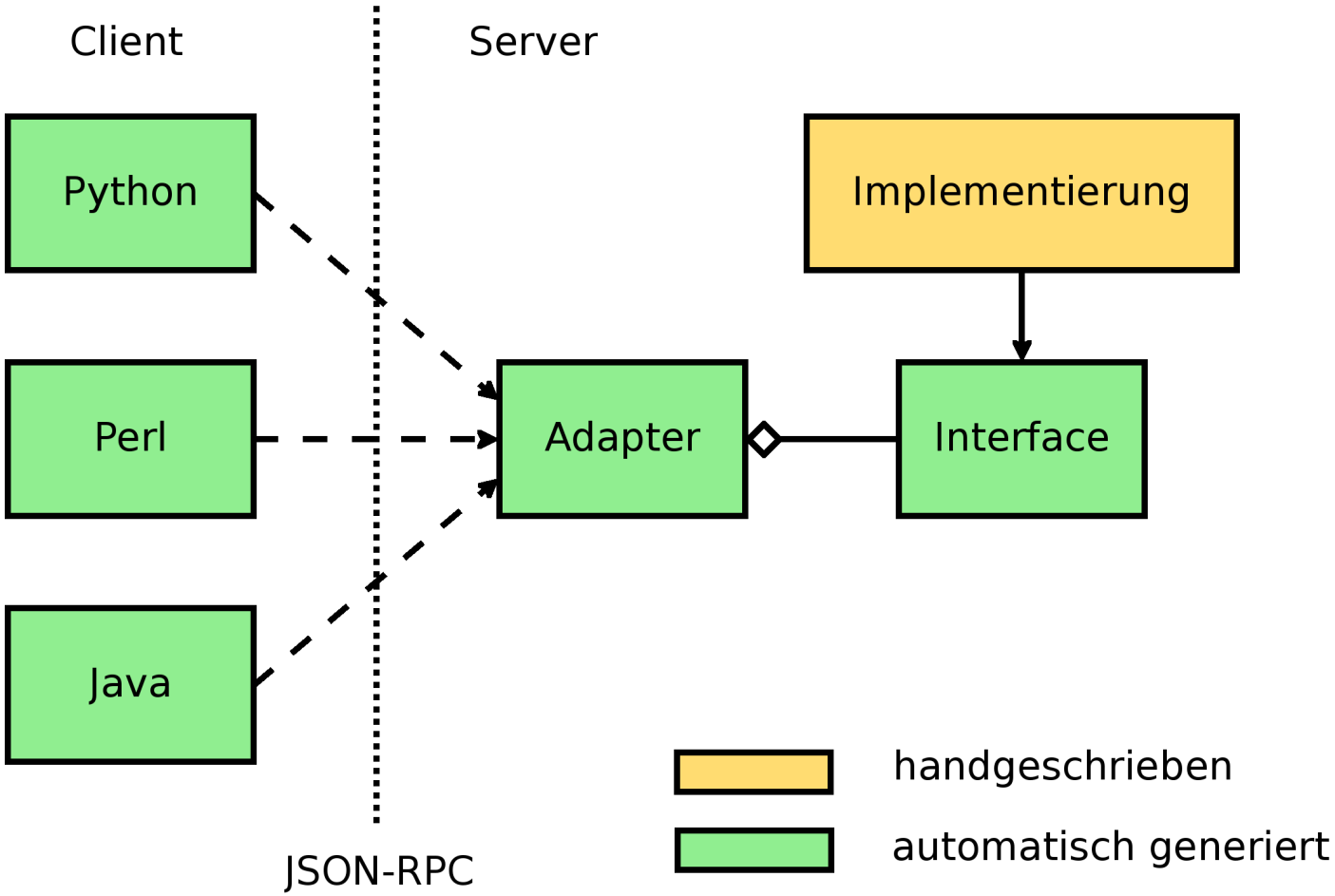
Representational State Transfer - REST

- Programmierparadigma
- URI identifiziert konkrete(s) Resource / Objekt
- URI beinhaltet keine Methodeninformation
- Beispiel
  - http://10.0.42.2/model/pdu/0/inlet/0

Vorgehensweise

- Abbildung der IDL auf entsprechende Klassen und Methoden
- (De)Serialisierung von/nach JSON-RPC
- Client: Objekt in Zielsprache agiert als Proxy für entferntes Objekt
- IDL-Compiler
  - geschrieben in Java
  - ein Backend pro Sprache und Protokoll-Mapping
  - C++ (Client und Server), Java (GWT), Java, JavaScript, Perl, Python, Ruby

System-Überblick



Beispiel

Abfrage von Daten auf der Kommandozeile

```
$ curl -s -k -d \
    '{"jsonrpc":"2.0", \
      "method":"getReading", \
      "params":{},"id":411}' \
http://user:pass@10.0.42.2/model/pdu/0/inlet/0/activePower

{
"jsonrpc": "2.0", "result":{
  "_ret_":{
   "timestamp": 1426690798, "available": true,
   "status": {
    "aboveUpperCritical": false, "aboveUpperWarning": false,
    "belowLowerWarning": false, "belowLowerCritical":false },
   "valid": true,
   "value":690.620033}
  }, "id":411}
$
```

Beispiele 1

IDL

```
interface Echo {
   structure EchoData {
      int a;
      string b;
   };

   int echo(in  EchoData datain,
            out EchoData dataout);
};
```

Perl

```
$datain = { "a" => 5,
            "b" => "Test" };
$ret = $intf->echo($datain,\$dataout);
print $dataout->{"a"};
```

Python

```
datain = EchoData(5, "Test")
(ret, dataout) = intf.echo(datain)
```

Java

```
EchoData datain = new EchoData();
datain.a = 5;
datain.b = "Test";
EchoResult r = intf.echo(datain);
int rv = r._ret_;
int a = r.dataout.a;
String b = r.dataout.b;
```

Beispiele 2

IDL

```
interface Echo {
   structure EchoData {
      int a;
      string b;
   };

   int echo(in  EchoData datain,
            out EchoData dataout);
};
```

C++ - Interface

```
struct EchoData {
   int32_t a;
   std::string b;
};
virtual int echo(int32_t& _ret_,
                 const EchoData& datain,
                 EchoData& dataout) = 0;
```

C++ - Server-Implementierung

```
class EchoImpl: public virtual Echo {
...  }

int EchoImpl::echo(int32_t& _ret_,
                   const EchoData& datain,
                   EchoData& dataout) {
    dataout = datain;
    _ret_ = 0;
    return 0;
}
```

Versionierung

- Schnittstellen ändern sich über die Zeit
  - neue Funktionen
  - Korrekturen
- Anforderungen
  - geringe Auswirkungen auf Clients
- Unterscheidung
  - Kompatible Änderung
    - Client muss nicht geändert werden
    - z.B. neues Element in out-Parameter-Struktur
  - Aufruf-kompatible Änderung
    - Client kann alle Methoden der Klasse aufrufen
    - Client kann ggf. mit dem Rückgabe-Objekt nichts anfangen
  - Inkompatible Änderung
    - Client kann einige Methoden nicht aufrufen

Versionierung 2

- Unterstützung mehrerer Versionen
  - vom Server
  - vom Client
- Granularität
  - Komplette API
  - Interface/Klasse
  - Methode
- Benutzung der Versionierung durch Clients
  - Versionierung ignorieren (kein Aufwand, „best effort“)
  - Unterstützung und Prüfung genau einer Version (wenig Aufwand, geringe Kompatibilität)
  - Unterstützung mehrerer Versionen (substantieller Aufwand)

Zusammenfassung

- Vorteile
  - Erstellung und Test einer einzigen API auf Server-Seite
  - Unabhängige Entwicklung von Oberflächen und Skripten
  - C++, Java (GWT), Java, JavaScript, Perl, Python, Ruby
- Praktische Anwendung
  - Web-Frontend: Java (GWT)
  - Web-Frontend für mobile Geräte: JavaScript
  - Anbindung an Steuerungssysteme: Ruby
  - Skripte für Groß-Installationen: Perl, Python
  - Skripte für Test und Produktion: Python
- Nachteile
  - Entwicklung von Clients für mehrere Versionen aufwendig
  - Bei inkompatiblen Änderungen muss Client geändert werden
  - Kompatible Änderungen sind zu bevorzugen

Vorführung

Praktische Vorführung

Ende

Fragen? Fragen!