REST Anwendungsarchitektur

Max Mustermann

Zusammenfassung
Das Abstract ist eine maximal 200 Worte lange Zusammenfassung des Inhalts der Arbeit, so dass sich der Leser vorab ein erstes Bild vom Inhalt machen kann.

Keywords
Web-Entwicklung, RESTful, SOA, JSON, Design, Java

Hochschule Kaiserslautern
Corresponding author: max.mustermann@fh-kl.de

Inhaltsverzeichnis

Einleitung

Die Einleitung sollte ausreichenden Hintergrund für den Leser liefern, so dass er sich ohne großes Studium von Sekundärliteratur in das Thema hineindenken kann. Auch sollte die Aufgabenstellung bzw. Motivation für die vorliegende Arbeit dargelegt werden, sowie die Zielsetzung, die man erreichen will. Weiter wird auch das Thema von eventuell verwandten Themen abgegrenzt. Hier kann auch die Literatur [1, 2] und [3] vorgestellt werden.

Konkret sollt hier das Ziel und die Motivation des Projekts erläutert werden. Am Ende des Abschnitts steht eine kurze Übersicht über die weiteren Abschnitte.

Auch wenn die Einleitung zu Beginn der Arbeit liegt, wird sie oft erst am Ende verfasst.

1. REST: State of the art

Um zu verstehen, weshalb REST allgegenwärtig in der Entwicklung von Web-Applikationen ist, werden im Folgenden die Eigenschaften datiert, welche ein REST Interface spezifiziert. Um weiterhin die Unterschiede, sowie Vor- und Nachteile, zu diskutieren, wird eine Gegenüberstellung mit dem SOAP (*Simple Object Access Protocol*) vorgenommen.

1.1 Paradigmen und Eigenschaften

Die Aufbau aktueller Web-Applikationen beruht auf der Überlegung einer Service orientierten Kommunikation. Aufbauend auf der *Service Oriented Aarchitecture(SOA)* wurde vom W3C die Definition für die noch heute gültige *Web Services Architecture*(vgl. [4]) festgelegt. Hier wird schon REST als Modell für das konstruieren für Web Services verwendet. Um jedoch eine REST-konforme Architektur zu realisieren wurden von Roy Thomas Fielding in seiner Dissertation[5] eine Anzahl von Voraussetzungen festgeschrieben, die ein Architekt auf Basis von REST erfüllen muss:

- **Client-Server:**
  Der Ansatz der Client-Server Architektur soll verfolgt werden. Durch das Lösen der UI von den Daten der Applikation wird eine Portabilität, Skalierbarkeit und das Vereinfachen der Server-Komponente realisiert, was mittlerweile als Grundlage der kompletten Internet-Architektur gilt, denn der HTTP-Standard selbst beruht auf dieser Architektur.

- **Stateless:**
  Die Zustandslosigkeit gilt für die Kommunikation zwischen Client und Server. Anfragen des Clients müssen alle Information enthalten um vom Server identifiziert und bearbeitet werden zu können. Einen Zustand oder Session-Informationen liegen somit komplett auf Seiten des Clients.

- **Cache:**
  Die Antwort auf eine Client-Anfrage kann vom Server als cacheble oder non-cacheble gelabelt werden. Der

Cache befindet sich auf der Client-Seite und erlaubt dem Client die Antwort für weitere Anfragen gleicher Art zu verwenden. Dadurch kann teilweise eine Kommunikation zwischen Client und Server unterbunden werden was die Performanz und Effizienz client-seitig stark erhöhen kann.

- **Uniform Interface:**
  Das Uniform Interface repräsentiert das fundamentale Wesen eines REST-Services. Durch die Uniformierung jeder Ressource werden die Daten von der Architektur gelöst, was eine starke Entkopplung sowie Vereinfachung der Interaktion mit sich bringt. Weiterhin soll jeder Client die nötigen Information und Daten erhalten, die es ihm erlaubt, Ressourcen zu identifizieren und dadurch auch zu modifizieren. Die angefragte Ressource muss nicht die selbe Repräsentation auf Server- und Client-Seite besitzen. Objekte vom Server werden also zum Beispiel via XML oder JSON übertragen, enthalten die definierten Daten, müssen aber nicht dem Datentyp des Server-Objekts entsprechen. Nach Fieldings ist hier das *Hypermedia as the engine of application state (HATEOAS)*-Prinzip mit der wichtigste Faktor. Dieses besagt, dass der Client alle Ressourcen nur über die definierten URI's beziehen kann. Durch diesen Zugang kann der Server dynamisch Daten kommunizieren und weiter verfügbare Adressen der Antwort hinzufügen. Somit wird verhindert, dass der Client fest codierte Information enthalten muss. Dadurch entsteht die Flexibilität des REST-Services.

- **Layered System:**
  Dem Client muss nicht bekannt sein, ob die Kommunikation direkt mit dem Server stattfindet oder nicht. Die Anfrage wird gestellt und welcher Server aus einem Kollektiv die Antwort sendet, kann nicht zugeordnet werden. Dadurch können Prinzipien wie Load-Balancing und unterschiedliche Web-Sicherheitskriterien realisiert werden.

- **Code on demand(optional):**
  Dieser Schritt ist optional und beschreibt die Möglichkeit nach Anfrage ausführbaren Code vom Server zum Client zu transferieren, wie zum Beispiel Skripte in JavaScript. Somit kann der Server zeitweise die Funktionalität der Client-Applikation beeinflussen.

Im Kontext von Web-Services, welche sich an die REST-Vorgaben binden, wird zumeist die Kommunikation über HTTP umgesetzt. Die API wird über eine Basis-URL angesprochen, über welche dann die benötigten Ressourcen angefragt werden müssen. HTTP bietet eine gewisse Anzahl an Standardmethoden(hauptsächlich: *GET, PUT, POST, DELETE*) die auch für den Datentransfer in einer RESTful-API genutzt werden. Die grundlegenden Eigenschafen der Methoden finden sich im Bild 1 Eine mögliche Variante mit den entsprechenden HTTP-Befehlen ist in der Tabelle 1 zu sehen.

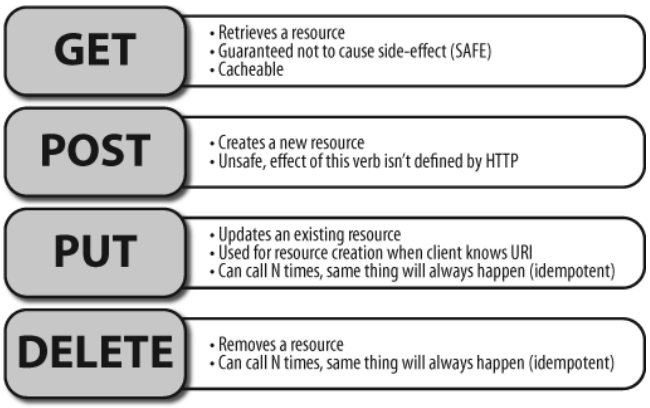

Abbildung 1. Beschreibung der Standard HTTP Methoden in der REST-Architektur. Quelle: http://www.itersdesktop.com/

Tabelle 1. HTTP Methoden für spezifische URL's

URL	**http://api.example.com/res/item42**
GET	Eine Repräsentation des Items42 wird an den Client übertragen
POST	Wird zum Erstellen neuer Ressourcen verwendet, würde in diesem Fall eher nicht genutzt werden
PUT	Item42 wird überschrieben oder erstellt wenn noch nicht vorhanden
DELETE	Das adressierte Element Item42 wird gelöscht

1.2 REST vs. SOAP

Bevor sich REST im Bereich der verteilten Anwendung etablierte war die größte Vertreter für die Kommunikation innerhalb einer SOA das SOAP. Bei SOAP handelt es sich um eine RPC Middleware die HTTP oder SMTP als Transportprotokoll für XML als Nachrichtenformat verwendet. Schon im grundsätzlichen Ansatz unterscheiden sich beide Herangehensweisen. Wird bei REST von einer exakten Uniformierung jeder Ressource gesprochen, die somit auch immer separat angesprochen werden können, so wird bei SOAP mit einem Dispacher gearbeitet. In sogenannten Envelopes werden alle Anfragen an den Server via XML mit dem HTTP-Post-Befehl an den zentralen Dispatcher geleitet. Die Anfrage gehen an genau eine URL die den Dispatcher adressiert. Im Listing 1 ist ein SOAP-Request mit seiner korrespondierenden Response zu sehen. Im Body wird eine *GetStockPrice* Anfrage gestellt für das Unternehmen IBM. Im Body der Antwort wird somit die *GetStockPriceResponse* mit dem Wert von 34,5 übermittelt. Die URL *http://www.example.org/stock* spricht den Dispatcher des Webservers an. Dieser verteilt die Anfrage an die richtige Methode, hier *GetStockPrice*. Diese Methode muss auf dem Server verfügbar sein. Durch die Anfrage wird

ein Methodenaufruf mit den gelieferten Parametern durchgeführt. Das Ergebnis des Methodenaufrufs wird als Response in einem neuen Envelope zurück geliefert. Da jede Antwort einer exakten Anfrage gilt, ist ein erneutes stellen der gleichen Anfrage nicht möglich. Somit entfällt eine Möglichkeit caching auf Seiten des Clients zu betreiben.

```
 POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=
    utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-
    envelope/"
soap:encodingStyle="http://www.w3.org
    /2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/
    stock">
<m:GetStockPrice>
<m:StockName>IBM</m:StockName>
</m:GetStockPrice>
</soap:Body>

</soap:Envelope>

-----------------------------------------

 HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=
    utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-
    envelope/"
soap:encodingStyle="http://www.w3.org
    /2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/
    stock">
<m:GetStockPriceResponse>
<m:Price>34.5</m:Price>
</m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Listing 1. Beispiel: SOAP-Request und SOAP-Response als Envelope

Ebenso wie REST wurde SOAP vom W3C spezifiziert. Unterschied ist allerdings, bei REST handelt sich um einen Architekturstil bei SOAP um ein Protokoll. in der Spezifikation steht zum Beispiel, dass ausschließlich XML als Datenformat verwendet wird und was ein XML wohl definiert um als SOAP konform zu gelten([6]). Der Unterschied zu REST liegt hier darin, dass außer XML bei REST weitere Datenformate wie zum Beispiel JSON und YAML verwendet werden können. Die genaue Adressierung der Ressourcen unter REST erlaubt eine leichtgewichtige Transportmöglichkeit. Metadaten, wie das Encoding oder der Content-Type, werden im Header des HTTP-Requests codiert. Zusätzliche Information werden nicht benötigt. SOAP hingegen bietet diese Möglichkeit nicht. Durch die Konvention, welche Bestandteile ein XML-Envelope vorzuweisen hat, ist diese Leichtigkeit nicht zu erreichen. Das XML-Dokument muss auf Seiten des Sender aufgebaut und anschließen validiert werden um die Konformität zu bewahren. Dadurch werden zusätzliche Metadaten zur Beschreibung der Datei dem XML-Dokument hinzugefügt. Gerade bei einfachen Anfrage, zum Beispiel eine Zustandsabfrage wahr oder falsch, ist das Verhältnis von Nutz- zu Metadaten eher mäßig. Je aufwändiger die Anfrage desto besser entwickelt sich das Nutzlastenverhältnis. Anfragen eine XML-Dokuments werden immer atomar ausgeführt. Somit können komplexe Sachverhalte in einer Anfrage definiert werden und sequentiell abgearbeitet werden. Bei REST werden meist leichtgewichtige Schnittstellen exponiert die genau eine Anfrage nach einer bestimmten Ressource beantworten. Somit sind bei komplexeren Schritten mehre Anfragen nötig. Dadurch ist der Kommunikationsaufwand etwas höher als bei SOAP. Relativ ähnlich verhalten sich die beiden in Bezug auf die Interaktion. Da beide Plattform unabhängige Datenformate verwenden entfaltet sich ein großen Potenzial in der Nutzung. Programme jeglicher Art, entwickelt in einer belieben Programmiersprache, welche die Datenformate ebenso unterstützen, können mit den Schnittstellen interagieren. Bei REST muss eine API-Beschreibung mit den exponierten URL's, bei SOAP die URL mit den möglichen Methodenaufrufen öffentlich gemacht werden. Bei SOAP wird die Beschreibung meist als WSDL(*Web Services Description Language*)(vgl. [7]) vorgelegt. In dem WSDL-File werden alle Funktionen, Daten und Datentypen beschrieben. Es werden im wesentlichen die Operation definiert, die von außen zugänglich sind. Mit diesem WSDL-File wird auch die Validierung des XML-Dokuments durchgeführt. Es handelt sich auch hierbei um einen weiteren Standard definiert vom W3C.

1.3 Evaluierung

Beschreibung vor und Nachteile REST interfaces

2. Entwicklungsansatz und Aufbau

Kurze Einführung in den Abschnitt. Es wird beschrieben was jetzt kommt. Erklärung Beispiel Projekt

2.1 JAX-RS - die Basis für REST-Services in JAVA

Erläuterung der API, Bestandteile, Features

2.2 Serverseite Beispielimplementierung
Erläuterung der API, Bestandteile, Features

2.3 Clientseite Beispielimplementierung
Erläuterung der API, Bestandteile, Features

3. Frameworks? Microservice-Architektur?

4. Zusammenfassung

Hier wird nochmal der Inhalt und die Ergebnisse der Arbeit erörtert. Im Ausblick werden Themen und Aufgabenstellungen genannt, die es lohnt weiter zu untersuchen.

Literatur

[1] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, June 1987.

[2] D. Harel, Y. Feldman, and M. Krieger. *Algorithmik*. Springer Berlin Heidelberg, 2006.

[3] Jilles Van Gurp and Jan Bosch. On the implementation of finite state machines. In *in Proceedings of the 3rd Annual IASTED International Conference Software Engineering and Applications, IASTED/Acta*, pages 172–178. Press, 1999.

[4] W3C Working Group. Web services architecture w3c working group note 11 february 2004. W3C.

[5] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures, 2000.

[6] W3C Working Group. Soap version 1.2 part 1: Messaging framework (second edition). W3C.

[7] W3C Working Group. Web services description language (wsdl) 1.1. W3C.

[8] Peter Rechenberg. *Technisches Schreiben (nicht nur) für Informatiker*. Hanser-Verlag, 2002.

Erklärung zur Ausarbeitung
Hiermit erkläre ich, *Vorname Nachname (Matrikel)*, dass ich die vorliegende Ausarbeitung selbstständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angegebenen Hilfen benutzt habe; dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

_____

Unterschrift

1. Bemerkungen zur Ausarbeitung

Der Umfang der Ausarbeitung sollte 10 Seiten nicht übersteigen. Die Ausarbeitung sollte dem Charakter nach eher einem *scientific paper* entsprechen. Siehe hierzu auch [8]. Keine Ich-Form benutzen. Aussagen sollten möglichst belegt oder begründet werden.

Die schriftliche Ausarbeitung hat folgende Zwecke:

- die Ausarbeitung soll Kommilitonen (die nicht an der Veranstaltung teilgenommen haben) in das Thema einführen.
- die Autorin bzw. der Autor soll üben, wie man technische Sachverhalte kurz und klar beschriebt.
- die Ausarbeitung bildet die Grundlage der Bewertung. In der Arbeit soll gezeigt werden, dass man das Thema geistig durchdrungen hat.

A.1 Latex-Syntax

Im folgenden finden Sie einige nützliche Latex-Anweisungen.

Abbildung 3 zeigt ein Bild, dass die komplette Seitenbreite einnimmt. Abbildung 2 ist dagegen in die Spalte eingebettet.

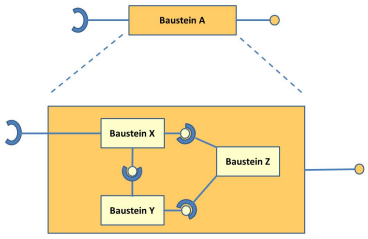

Abbildung 2. Beispiel für ein Bild in der Spalte

Hier sehen Sie, wie man ein Java-Code-Listing mit einbinden kann. Es kann auch ein Label vergeben werden, auf das dann referenziert werden kann (vgl. Listing 2).

```
public class Test
{
  private int mCount = 0;

  @Override
  public void xyz()
  {
    for(int i = 0; i< 10; i++)
    {
      this.mCount += i;
    }

    // Ein Kommentar
    this.mEnd = this.mCount;
  }
}
```

Listing 2. Eine Testklasse

Hier ein Beispiel für eine mathematische Formel ohne Nummer

$$\cos^3\theta = \frac{1}{4}\cos\theta + \frac{3}{4}\cos 3\theta$$

Oder mehrere mathematische Ausdrücke untereinander

$$f(x) = \frac{3e^x}{1-x^2} \quad (1)$$

$$g(x) = \sqrt[3]{\sin\alpha} \quad (2)$$

$$h(x) = \frac{2+3i}{1-i} \quad (3)$$

Hier noch ein Beispiel für eine Aufzählung, wobei die Aufzählungspunkte direkt nacheinander, d.h. ohne Leerzeile, aufgelistet werden.

1. First item in a list
2. Second item in a list
3. Third item in a list

Ein Beispiel für eine Tabelle, falls man ein solches Konstrukt benötigt.

Tabelle 2. Table of Grades

Name		
First name	Last Name	Grade
John	Doe	7.5
Richard	Miles	2

Siehe Tabelle 2. Und hier nochmal das Einbinden eines Bildes (vgl. Abb. 4.), wobei das Bild innerhalb der Spalte gezeigt wird.

A.2 Allgemeine Hinweise

Bei den Ausführungen fasst man sich so kurz wie möglich, aber so lange wie nötig um verständlich zu sein. Man schreibt die Projektarbeit für Leser, die die gleichen Vorkenntnisse haben wie man selbst. Die Projektarbeit besitzt im Prinzip den selben Aufbau wie dieses Dokument.

Eine Arbeit wird nur einmal geschrieben aber von vielen, d.h. oft gelesen. Der Verfasser sollte es sich deshalb bei der Formulierung schwer machen, damit es der Leser später um so leichter hat. Benutzen Sie Bilder und legen Sie einen roten Faden durch Ihren Text.

A.3 Bemerkungen zur Literatur

Die Literaturliste muss die Referenzen enthalten, auf die man vorher verwiesen hat - nicht mehr und nicht weniger. Sie werden sicherlich sehr oft referenzieren. Wenn ein Artikel eine Web-Adresse hat, muss die Web-Adresse aufgenommen werden, vgl. [**?**] oder [**?**].

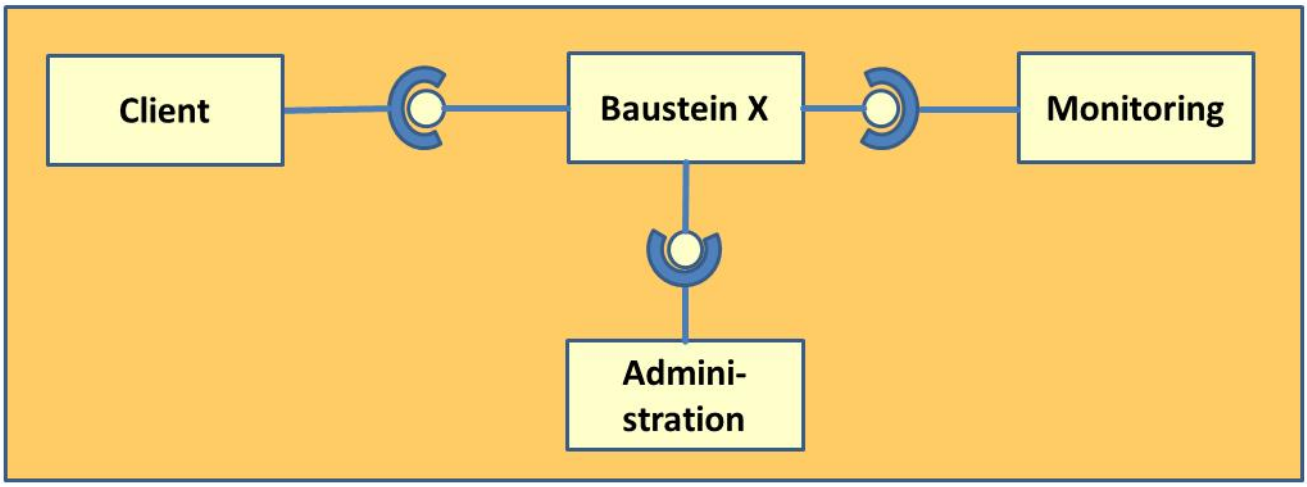

Abbildung 3. Beispiel für ein breites Bild

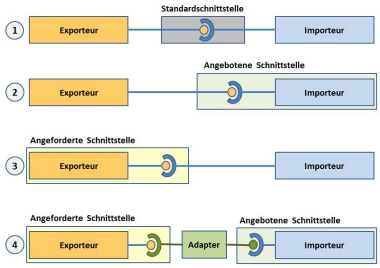

Abbildung 4. Beispiel für ein Bild in der Spalte

2. Formalien

B.1 Bewertungskriterien

Damit die Arbeit nicht nur eine reine Zusammenfassung des Vorlesungsinhalts bleibt, soll die Projektarbeit auch einen *Eigenanteil* enthalten. Dieser Eigenanteil kann z.B. eine eigene Literaturrecherche sein und somit Themen aufgreifen, die nicht explizit in der Veranstaltung besprochen wurden.

In die Bewertung der Projektarbeit gehen folgende Punkte mit ein:

1. Formale Kriterien: Äußere Form, Layout, Zitiertechnik und korrekte Angabe der Literatur, Stil, Abbildungen, Rechtschreibung, etc.
2. Systematik der Darstellung: Vollständigkeit (Wie breit und tiefgehend wird das Thema behandelt?), korrekte Wiedergabe (wurde das Thema richtig verstanden), logische Gliederung und Gedankenführung
3. Eigenständigkeit der Darstellung: Wurde in eigenen Worten zusammengefasst oder nur Zitate benutzt? Präsentation des Themas (Didaktik), wurden Aussagen zueinander in Beziehung gesetzt und wurde ein eigener gedanklicher Aufbau gewählt?
4. Tiefe und Umfang des Eigenanteils: Hat der Eigenanteil fachliche Substanz? Innovationspotential, d.h. wie viele eigene Ideen wurden eingebracht.

3. Listings

Hier im Anhang sind Listings aufgeführt, die besser im „großen" einspaltigen Format wiedergegeben werden.

```
import spellchecker.algorithm.EditDistance;

public class LevenshteinAlgorithm implements EditDistance
{
  public final int distance(String from, String to)
  {
     assert from != null && to != null : "Parameters should not be null";

     // Sonderfaelle
     if (from.equals(to))    return 0;
     if (from.length() == 0) return to.length();
     if (to.length() == 0)   return from.length();

     int width = from.length() + 1;
     int height = to.length() + 1;

     // Tabelle
     int[][] table = new int[height][width];

     // Initialisierung erste Zeile
     for (int i = 0; i < width; i++)
     {
        table[0][i] = i;
     }

     // Zeilen (to)
     for (int i = 1; i < height; i++)
     {
        table[i][0] = i;
        // Spalten (from)
        for (int j = 1; j < width; j++)
        {
           int delta = 0;
           if (to.charAt(i - 1) != from.charAt(j - 1))
               delta = 1;

           table[i][j] = min(
                table[i][j-1] + 1, table[i-1][j] + 1, table[i-1][j-1] + delta );
        }
     }

     return table[height-1][width-1];
  }

  public final static int min(int a, int b, int c)
  {
     if (a < b && a < c)
         return a;
     else if (b < c)
        return b;
     else
       return c;
   }
}
```

Listing 3. Der Levenshtein Algorithmus