Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Fachstudie Nr. 174

Vergleich von Sprachen, Methoden und Tools zur Modellierung und Beschreibung von REST Schnittstellen

Leonard Bruder, Fabian Harth, Nedim Karaoğuz

Studiengang: Softwaretechnik

Prüfer: Prof. Dr. Frank Leymann
Betreuer: Dipl.-Inf. Florian Haupt
begonnen am: 10.04.2013
beendet am: 10.10.2013

CR-Klassifikation: D.2.1, F.3.1

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

REpresentational State Transfer (REST), beschrieben im Jahr 2000 von Roy Fielding [1], ist ein Architekturstil der die vier Kernziele: Geschwindigkeit, Skalierbarkeit, Simplizität und Datenunabhängigkeit verfolgt und sich dazu auf vier Prinzipien stützt:

- Ressourcenidentifikation mittels URIs: Eine Ressource eines Dienstes ist eindeutig über ihre URI bestimmt und wird über diese adressiert.
- Einheitliches Interface: REST unterstützt nur die CRUD[1]-Methoden.
- Selbst-beschreibende Messages: Ressourcen sind losgelöst von ihrer Repräsentation, so dass sie in einer Vielzahl von Formaten zur Verfügung stehen können.
- Stateful Interaktionen über Hyperlinks: Jede Interaktion mit einer Ressource ist zustandslos, sodass der Status während der Interaktion übermittelt werden muss.

Durch sein hervorragendes Mapping auf die vier HTTP-Hauptmethoden: POST, GET, PUT, DELETE, und seine Leichtgewichtigkeit [2], existieren mittlerweile eine Vielzahl von Diensten, die ihre API mittels REST zur Verfügung stellen. Im Gegensatz zu Diensten, die auf dem WS*-Stack basieren und ihre funktionalen und nicht-funktionalen Eigenschaften mit Hilfe von WSDL bzw. WS-Policy beschreiben und sie dadurch Kunden zur Verfügung stellen, existiert bis jetzt kein vergleichbarer Industriestandard um REST APIs zu beschreiben und/oder zu modellieren. Dies bedeutet, dass viele REST-Dienste nur eine textuelle, lediglich für Menschen verständliche Beschreibung anbieten. Es gibt auch einige Meinungen[2], die entweder nur einen sehr geringen oder gar keinen Bedarf an der Dokumentation einer REST API sehen, weil diese selbstbeschreibend sein sollte. Dies wirft jedoch eine Vielzahl von Problemen auf: So ist beispielsweise die komplette Funktionalität einer selbstbeschreibenden REST API erst ersichtlich, nachdem man den gesamten Baum von Verlinkungen durchlaufen hat.

In dieser Fachstudie präsentieren wir einige Ansätze zur Beschreibung von REST Diensten und bewerten diese anschließend anhand eines Kriterienkatalogs. Dieser wird in Kapitel 2 näher vorgestellt. Des Weiteren stellen wir einige Ansätze aus wissenschaftlichen Veröffentlichungen vor und beschreiben diese näher.

Die meisten der untersuchten Sprachen verfolgen einen von zwei Ansätzen um REST-Dienste zu beschreiben. Entweder einen service- bzw. operationenorientierten oder einen ressourcenorientierten Ansatz. Als operationenorientierten Ansatz definieren wir in dieser Fachstudie Sprachen, die das Beschreiben der Funktionalität der verschiedenen Operationen eines Services in den Vordergrund stellen. Die eigentlichen Ressourcen rücken dabei in den Hintergrund. Eine weitere Möglichkeit, den einige Sprachen verfolgen, ist die Eigenschaften eines Dienstes mit Hilfe von semantischen Ausdrücken zu beschreiben. Diese beiden Ansätze werden von einigen entweder als ungeeignet für

[1] Create Retrieve Update Delete
[2] (09.10.2013): http://stackoverflow.com/questions/1966243/restful-api-documentation

eine Resource Oriented Architecture (ROA) angesehen, weil sie den Fokus auf die Modellierung der Repräsentationen der Ressourcen vermissen, oder als sehr aufwändig bzw. umständlich, weil das Semantic Web noch in einer frühen Phase der Entwicklung steckt. Lanthaler et al. fassen diese Zurückhaltung unter dem Begriff Semaphobia zusammen [3].

1.1 Zweck und Aufbau des Dokuments

Dieses Dokument bewertet einige Spezifikationen, Frameworks und wissenschaftliche Ausarbeitungen, die syntaktische, semantische oder konzeptionelle Beschreibungsmöglichkeiten für RESTful Services definieren bzw. verwenden. Als Grundlage dafür dient ein zu diesem Zweck entworfener Fragenkatalog, der im nachfolgenden Kapitel beschrieben wird.

Damit kann es Entwicklern von Schnittstellen einen Anhaltspunkt geben, welche Technologie geeignet ist, um diese zu dokumentieren. Weil ein zentraler Aspekt die Beschreibung von Schnittstellen in einem maschinenlesbaren Format ist, werden verschiedenste Ausdrucksmöglichkeiten dafür vorgestellt und können als Übersicht verwendet werden.

1.2 Gliederung

Der Rest des Dokuments ist wie folgt gegliedert:

Im nächsten Kapitel wird der für die Bewertung verwendete Kriterienkatalog beschrieben. Hierbei wird auch auf die unterschiedliche Gewichtung der Fragen eingegangen.

Das dritte Kapitel enthält die Beschreibungen und Bewertung der Spezifikationen und Ausarbeitungen, die nicht durch das Bewertungssystem evaluiert wurden. Die Bewertung ist also eher subjektiv und basiert auf den Erkenntnissen der Autoren über die anderen Sprachen.

Das vierte Kapitel fasst die Beschreibungen und Bewertungen der restlichen Spezifikationen und Frameworks zusammen, auf die der Kriterienkatalog „angewandt“ wurde. Hierbei wurde für jede Frage auf einer Skala von 0 bis 3 angegeben, inwieweit der jeweilige Prüfling die durch die jeweilige Frage ausgedrückte Anforderung erfüllt. Für jede Beschreibungssprache wurde ein Fazit verfasst, das den gewonnenen Eindruck zusammenfasst.

Im fünften Kapitel werden die Ergebnisse der Bewertungen mit dem Kriterienkatalog gesammelt und interpretiert.

Das letzte Kapitel dient dann als Zusammenfassung des Dokuments.

2 Kriterienkatalog

Um Rest-Schnittstellen zu modellieren gibt es sehr unterschiedliche Ansätze. Viele konzentrieren sich auf eine rein syntaktische Sichtweise, was bedeutet, sie modellieren Ressourcen, Methoden, erlaubte Inputs und Outputs statisch. Diese Beschreibungen dienen zum einen als Referenz für Entwickler, die clientseitig gegen diese Schnittstellen implementieren, zum anderen gibt es Möglichkeiten, daraus Codestubs zu generieren.

Eine weitere Möglichkeit ist der eher konzeptionelle Ansatz, der durch die Verwendung von UML-Diagrammen zum Entwerfen von Schnittstellen bzw. Services dient. Auch hier ist das Generieren von Codestubs teilweise vorgesehen.

Andere Ansätze konzentrieren sich darauf, die Semantik von Services oder von Methodenaufrufen formal zu fassen. In den meisten Fällen verwenden sie Ontologien, die wiederum RDF[3] verwenden, oder geben Möglichkeiten an, ihre Beschreibungen nach RDF zu konvertieren.

Mit diesem Hintergrund haben wir versucht, eine faire Vergleichsbasis zu schaffen. Weil sich die sprachlichen Ansätze von den konzeptionellen bzw. modellierenden sowohl in der Philosophie als auch dem zugrundeliegenden Anwendungsfall unterscheiden, haben wir letztere zwar begutachtet, jedoch nicht durch das Bewertungsschema erfasst.

Die Fragen zur Bewertung sind in vier Gruppen unterteilt: Syntax, Semantik, Lesbarkeit und Anwendbarkeit. Das ermöglicht eine präzisere Beurteilung der einzelnen Sprachen unter verschiedenen Gesichtspunkten.

In der ersten Kategorie des Kriterienkataloges sind syntaktische Aspekte zu finden, im zweiten Teil semantische, wodurch der Kriterienkatalog auf beide Ansätze anwendbar ist. Die dritte Gruppe behandelt die Lesbarkeit des Codes, in dem die Schnittstellenbeschreibung erfolgt. Im letzten Teil blicken wir über die formalen Spezifikationen hinweg und beurteilen die Relevanz in der Praxis, also, ob Tools angeboten werden und wie hilfreich diese wirklich sind.

Die Auswahl der Fragen erfolgte dabei nachdem wir uns die zur Begutachtung stehenden Ansätze etwas genauer angeschaut hatten. Ideen einzelner Arbeiten, die in ihrem Vorkommen einzigartig waren, wurden nicht mit aufgenommen. Vielmehr wollten wir Elementares erfassen und Erfolgsmethoden („best practice“) mit berücksichtigen. Mit dem Hintergrund, dass eine dieser Spezifikationen bzw. Ideen zur Beschreibung einer Schnittstelle eingesetzt werden sollte, sahen wir uns veranlasst, die Verständlichkeit und den erforderlichen Einarbeitungsaufwand ebenfalls zu hinterfragen.

Für die Evaluierung wurden die Prüflinge zu jeder Frage auf einer Skala von 0 bis 3 bewertet. Die Punktzahl drückt dabei aus, inwiefern der Prüfling die durch die Frage gestellte Anforderung erfüllt (1 Punkt: in Ansätzen erfüllt; 2 Punkte: zu Teilen erfüllt; 3 Punkte: vollständig erfüllt).bzw. nicht erfüllt (0 Punkte).

[3] (09.10.2013): http://en.wikipedia.org/wiki/Resource_Description_Framework

2.1 Gewichtung

Für die Gesamtbewertung haben wir zum einen die einzelnen Fragen innerhalb einer Gruppe gewichtet und uns zum anderen die Möglichkeit offen gehalten, die jeweiligen Gruppen selbst für eine Gesamtwertung zu gewichten. Ersteres ermöglicht die strenge Bewertung elementarer Aspekte und gleichzeitig das Aufnehmen von nützlichen und sinnvollen Features einzelner Ansätze.

Auch die Tatsache, dass einzelnen Fragen eine unterschiedliche Reichweite - also die Summe des durch die Beantwortung dieser Frage tatsächlich Bewerteten - zugrundeliegt, wird hierdurch berücksichtigt. (Die Frage „Wie gut ist die Spezifikation?“ hätte beispielsweise eine Reichweite von 100%, die Frage nach deren Ausdrucksstärke nur noch 50% usw.) Weil darüber hinaus nicht für jede Gruppe die gleiche Anzahl an Fragen vorhanden ist, kann eine Gewichtung einzelner Gruppen bzw. Fragen diesen Umstand relativieren.

Die Gewichtung der Gruppen ermöglicht hingegen die Bewertung eines Ansatzes über dessen eigentliche Möglichkeiten hinaus. Steht beispielsweise die Praxistauglichkeit mehrerer Sprachen mit vergleichbarer Ausdrucksstärke (syntaktisch / semantisch) zur Debatte, kann durch die Gewichtung der Gruppe Anwendbarkeit eine aussagekräftige Wertung für deren Einsatz erzielt werden.

Kombiniert man die Anzahl der Fragen mit der jeweiligen Gewichtung ergibt sich der Einfluss einer Gruppe auf die Gesamtwertung. Die Ausdrucksstärke (Syntax & Semantik) bilden dabei 52%. Während die Anwendbarkeit der Sprache mit 42% zur Geltung kommt, ist die Auswirkung der Lesbarkeit mit den verbleibenden 6% überschaubar.

Im Folgenden ist der fertige Kriterienkatalog sowie die verwendete Gewichtung aufgeführt. Wir wollten bei der Auswertung zum einen die Möglichkeiten einzelner Ansätze abbilden und zum anderen mögliche Hürden widerspiegeln, die eine spätere Anwendung erschweren (wie z.B. zu hoher Aufwand für die Einarbeitung). Die Bewertung gibt also eine Antwort auf die Frage, welcher Ansatz verwendet werden sollte, wenn eine Schnittstelle neu entwickelt wurde und dokumentiert werden soll, um verwendet werden zu können.

2.2 Fragen zur Bewertung der Ansätze

2.2.1 Ausdrucksstärke der Sprache - Syntax

- Können die Ressourcen der Schnittstelle definiert werden? – **doppelt gewichtet**
- Können pro Ressource erlaubte Operationen definiert werden? - **doppelt gewichtet**
- Können erwarteter Input bzw. Output des Servers definiert werden?
- Können Beziehungen zwischen Ressourcen definiert werden?
- Kann für Methoden beschrieben werden, welche Statuscodes erzeugt werden können?
- Bietet die Sprache vordefinierte Ressourcentypen an?
- Können Ressourcen geschachtelt werden, sodass URIs relativ zur Eltern-Ressource definiert werden können?
- Können in URI Definitionen Templates verwendet werden?

2.2.2 Ausdrucksstärke der Sprache - Semantik

- Kann die Bedeutung einer Ressource definiert werden? – **doppelt gewichtet**
- Kann die Bedeutung einer Operation definiert werden? – **doppelt gewichtet**
- Kann die Bedeutung einer Beziehung zwischen Ressourcen definiert werden?

2.2.3 Lesbarkeit

- Wie gering ist der Anteil an Syntax-Overhead im Code?
- Wie intuitiv ist den Code-Konstrukten ihre Bedeutung anzusehen?

2.2.4 Anwendung der Sprache

- In welchem Umfang wird die Sprache bereits eingesetzt?
- Wie gut ist die Sprache dokumentiert? – **doppelt gewichtet**
- Wie gut sind die Tools dokumentiert, die für diese Sprache angeboten werden? – **doppelt gewichtet**
- Einfachheit: wie gering ist der Aufwand zur Einarbeitung in die Sprache? – **doppelt gewichtet**
- Unter Verwendung frei verfügbarer Tools und Frameworks

- Wie gut wird der Entwickler beim modellieren unterstützt?
- Gibt es eine hilfreiche, graphische Darstellung der Modelle? – **doppelt gewichtet**
- Unterstützung für folgende Szenarien:
  - Schnittstellen eines bestehenden RESTful Web Service sollen in einem maschinenlesbaren Format beschrieben werden. Beschreibungen sind bisher gar nicht oder nur in Plaintext vorhanden. – **doppelt gewichtet**
  - Es soll die Schnittstelle eines RESTful Web Service vor dessen Implementierung entworfen werden. Aus dem Modell sollen Code-Stubs generiert werden können.

3 Individuell bewertete Spezifikationen und wissenschaftliche Ausarbeitungen

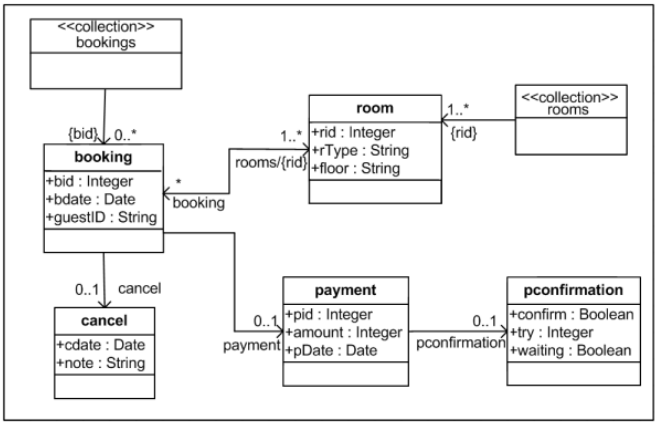
In diesem Kapitel wird eine individuelle Bewertung einzelner Spezifikationen und wissenschaftlicher Ausarbeitungen ohne die Verwendung des Kriterienkatalogs durchgeführt. Dieser konnte auf Grund der Eigenart einiger Ansätze nicht immer auf diese angewandt werden. Jeder Prüfling wird dabei in einem ersten Abschnitt beschrieben, um anschließend anhand unserer Erkenntnisse aus der Bewertung anderer Prüflinge bewertet zu werden.

3.1 Modeling Behavioral RESTful Web Service Interfaces in UML

3.1.1 Beschreibung

Die Arbeit von Porres et al. [4] erklärt, wie UML Diagramme verwendet werden können, um die Semantik von Webservices zu beschreiben. Die Autoren nennen als Vision eine „automatic service discovery“ und „service repositories“. Um diesen Ansatz zu motivieren, beziehen sie sich auf WADL und argumentieren, erstens sei es mit WADL nicht möglich, irgendetwas über die Semantik eines Webservice auszudrücken, zweitens erlaube es WADL, Webservices zu beschreiben, die die REST-Prinzipien nicht befolgen.

Für das Modellieren von Ressourcen eines Webservices schlagen die Autoren vor, Klassendiagramme zu verwenden. Sie erklären genau, wie die Beschreibungsmöglichkeiten, die UML definiert, auf Ressourcen abgebildet werden können: Klassenattribute werden zu Repräsentationen der Ressource, Assoziationen beschreiben referenzierte Ressourcen, und Beschriftungen auf den Assotionzionen geben den relativen URI Pfad an. Mengen von Ressourcen können als „<<collection>>“ markiert werden. Als Beispiel haben die Autoren einen Service für Hotelreservierungen modelliert. Dabei sind für die URI Angaben Platzhalter verwendet, „{bid}“, und „{rid}“, für die die Id der entsprechenden „booking“, bzw. „room“ Ressource eingesetzt wird.




```
/bookings/{bid}/
/bookings/{bid}/cancel/
/bookings/{bid}/payment/
/bookings/{bid}/rooms/{rid}/
/bookings/{bid}/payment/pconfirmation/
/rooms/{rid}/
/rooms/{rid}/booking/
/rooms/{rid}/booking/cancel/
/rooms/{rid}/booking/payment/
/rooms/{rid}/booking/payment/
 pconfirmation/
```

Abbildung 1: Modellierung von Ressourcen durch ein UML-Klassendiagramm und zugehörige URIs [4]

Um semantische Aspekte auszudrücken sollen UML Zustandsdiagramme verwendet werden, wobei Zustandsübergänge mit Vor- und Nachbedingungen beschriftet werden. Das bedeutet, dass boolsche Ausdrücke angegeben werden, die definieren, unter welchen Bedingungen eine Methode aufgerufen werden kann, und welche Bedingungen nach der Ausführung der Methode gelten.

3.1.2 Bewertung

Es gibt verschiedene Ansätze, UML zu verwenden, um Restschnittstellen zu modellieren. Das Verwenden von Klassendiagrammen um Ressourcen zu modellieren scheint uns eine gute Idee zu sein, da Klassendiagramme diese statischen Zusammenhänge gut ausdrücken können. Außerdem kann mit der Verwendung von UML auf bestehende Tools zurückgegriffen werden und die Notation ist weit verbreitet und akzeptiert.

Die Ansätze zur Formulierung von semantischen Zusammenhängen bewerten wir als zu komplex, um industriell Anwendung zu finden.

3.2 Towards a Model-Driven Process for Designing ReSTful Web Services

3.2.1 Beschreibung

Laitkorpi et al. [5] beschreiben einen Vorgang um funktionale Spezifikationen in einen RESTful Web Service zu transformieren.

Als Motivation geben die Autoren an, dass während des gewöhnlichen Designprozesses wesentliche Prinzipien von REST verworfen werden, wenn benötigte Funktionalität auf konkrete Elemente einer API gemappt werden. Ebenso bemängeln sie den fehlenden ressourcenorientierten Ansatz von vorhandenen Designprozessen, da diese eher einen objektorientierten Fokus besitzen. Einen bestehenden Ansatz, nach Richardson und Ruby [6], um RESTful Services zu designen, halten Laitkorpi et al. für ungeeignet, weil er eine Designlücke offenbart. Als Beispiel wird ein einfacher Service einer Airline beschrieben. Die nachfolgendende Abbildung zeigt einen Ausschnitt, den die Autoren als Beispiel für eine Designlücke aufführen, weil aus diesem Ansatz nicht hervorgeht, auf welche Art und Weise der abgebildete Schritt „7: addSeats(seats)“ RESTful implementiert werden kann.

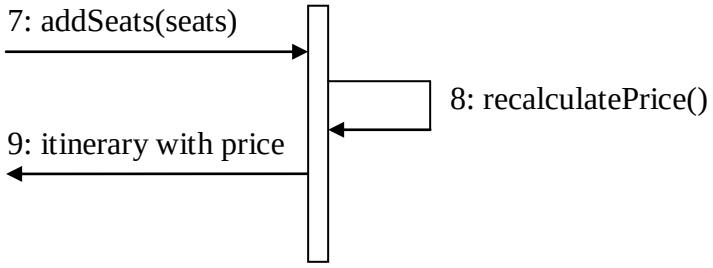


Abbildung 2: Ausschnitt aus der High-Level Designansicht eines Services [5]

Laitkorpi et al. schlagen einen mehrstufigen Prozess vor, der als Ziel hat, alle für einen RESTful Service relevanten Informationen zu beinhalten - sowohl für die Implementierung, als auch für die spätere Nutzung der API. Der Prozess unterteilt sich in folgende Phasen:

- Analysis: Die Autoren gehen davon aus, dass zu Beginn eine funktionale Spezifikation der Anforderungen des Services vorliegt - hauptsächlich bestehend aus einem UML Sequenzdiagramm. Zudem sollte diese Spezifikation zwei Sichten bieten. Zum einen die Businesssicht, mit den gegenseitigen Abhängigkeiten der Interaktionen, und eine High-Level Klassenansicht mit dem Vokabular der Domain.

- Behavioral canonicalization: Hierbei wird die High-Level Ansicht zwischen Client und Service heruntergebrochen, analysiert, um relevante Statusinformationen zu finden, geeignete primitive Operationen bestimmt, um diese Status zu manipulieren und daraus ein Information Model (unten stehende Abbildung) zu formen. Das darauf folgende Listing zeigt ein Beispiel für addSeats:

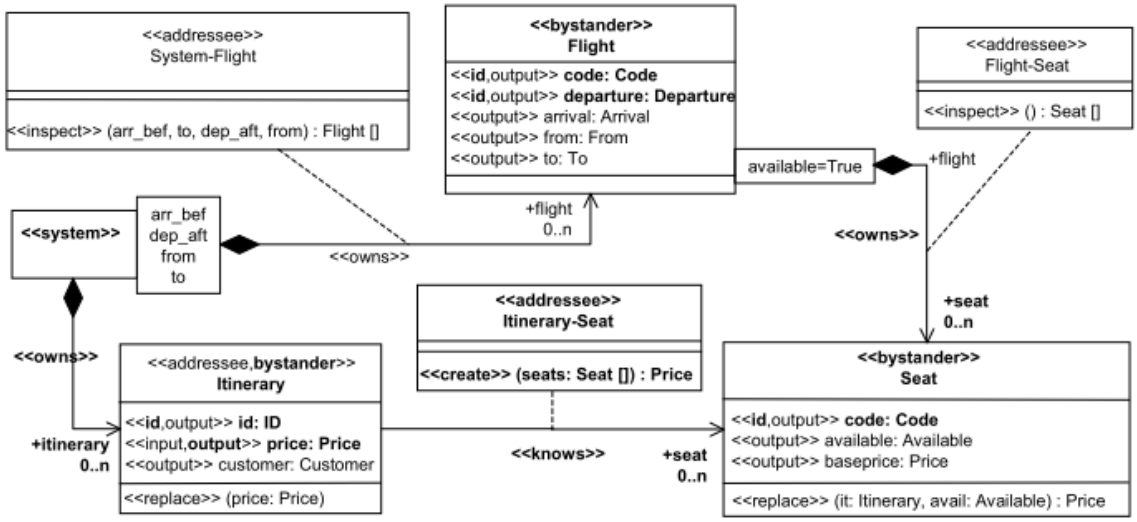


Abbildung 3: Ein Beispiel eines Information Model [5]

INTERACTION 7+9: addSeats	
Concern	Answer encoded as model elements
listener	Itinerary-Seat association class representing Itinerary --- <<knows>> ---> * Seat
bystanders	<<system>>, Itinerary, Seat, Flight
relationships	<<system>> --- <<owns>> --> * Itinerary
	Itinerary --- <<knows>> --> * Seat
	Flight --- <<owns>> --> * Seat
qualifiers	<<id>> attributes: Itinerary::id, Seat::code (values such as "K12"), Flight::code (values such as "AY204"), Flight::departure
Intention	<<stateChange>> (not shown in the model)
Effect	<<create>>
Content	<<input>>: Seat ([] indicates plural)
	<<input>> mapped to concept attributes: Seat::code, Seat::flight
	<<output>>: price
	<<output>> mapped to concept attributes: Itinerary::price; because Itinerary is not the same as <<addressee>>, this interaction must be split into two: Itinerary-Seat::<<create>>() and Itinerary::<<inspect>>(). Only the former is exempified

Listing 1: Information Model Elemente für addSeats [5]

- Structural canonicalization: In dieser Phase wird ein Resource Abstraction Layer über das zuvor erstellte Information Model gelegt, so dass Informationen nun mit aufrufbaren Ressourcenentitäten verknüpft sind.
- Service translation: Hier werden die nun erstellten Modelle auf konkrete Technologien übertragen. Laitkorpi et al. stellen in ihrem Paper WADL (siehe auch Kapitel 4.1 WADL) als eine Möglichkeit vor. Der beschriebene Ansatz ist jedoch nicht auf WADL beschränkt.

Die Autoren geben an, dass sie derzeit an einer Pattern Language arbeiten die grundsätzliche Prinzipien für die Entwicklung von RESTful Web Services beinhalten soll. Leider konnten wir während der Erstellung dieser Fachstudie keine Ergebnisse dieser Arbeit an einer Pattern Language finden.

Bezüglich möglicher Toolunterstützung sehen Laitkorpi et al. nur WADL-Tools, jedoch wären Tools zur Unterstützung des Modellierungsprozesses hilfreich, müssten aber in einer separaten Arbeit behandelt werden.

Zum Schluss geben die Autoren noch einen Ausblick: Sie bestätigen, dass sie zum Zeitpunkt der Veröffentlichung ihrer Arbeit noch keinen empirischen Beleg für die

Qualität und den Nutzen ihres Ansatzes vorlegen können. Sie waren aber zuversichtlich und stellten in Aussicht, einen Evaluierungsprozess in der ersten Hälfte im Jahr 2009 durchzuführen. Uns war es leider nicht möglich, Ergebnisse dieser Evaluierung oder weitere Arbeiten, die diesen Ansatz verfolgen, zu finden.

3.2.2 Bewertung

Das Problem, dass die Autoren eingangs beschreiben (und die daraus resultierenden Probleme), sind nachvollziehbar. Der Ansatz, ein strukturiertes und durchdachtes Vorgehensmodell zu schaffen, ist lobenswert. Es wäre sehr hilfreich für eine konkrete Bewertung gewesen, die Ergebnisse des - in der Arbeit angekündigten - Evaluierungsprozesses zu kennen, sofern er durchgeführt wurde. Der Titel des Papers, „Towards a Model-Driven Process for Designing ReSTful Web Services“ und der Ausblick den die Autoren darin selbst geben, legen nahe, dass diese Arbeit erst der Beginn auf dem Weg zu einem modellgetriebenen Prozess ist. Daher ist der Mangel an auffindbaren Arbeiten, die auf diesen Ergebnissen von 2009 aufbauen, verwunderlich und lässt befürchten, dass dieser Ansatz nicht konkret weiterverfolgt wurde.

3.3 An UML profile for modeling RESTful services

3.3.1 Beschreibung

In dieser Ausarbeitung [7] wird eine Erweiterung der Syntax und Semantik von Elementen des UML Metamodells vorgestellt. Das Ziel hierbei ist die Verringerung des Modellierungsaufwands von RESTful Services. Im Fokus stehen dabei RESTful Services, die als Controller von Webanwendungen dienen, die das Model-View-Controller-Pattern implementieren.

Die komplette MVC Anwendung kann dabei so modelliert werden, dass Interaktion und Struktur ersichtlich sind. Im Moment stehen die im Paper vorgestellten UML Elemente und Stereotypen nur in UML Profilen für das Programm Enterprise Architect zur Verfügung. Die Gründe für die Verwendung von Enterprise Architect sind die angebotenen Mechanismen für Metamodelle und das enthaltene Framework zum Generieren von Code. Letzteres erlaubt die Anpassung der Templates zur Codegenerierung.

Die Arbeit umfasst die Erweiterung für Controller, RESTful Services und Views.

- Ein RESTful Controller wird als Klasse mit dem Stereotypen „rest-controller“ repräsentiert, welcher das selbsterklärende Attribut „url“ zu dieser hinzufügt.
- Ein RESTful Service wird als UML Interface mit dem Stereotypen „rest-service“ modelliert, welcher Attribute für die URI und die HTTP-Methode bereitstellt. Darüber hinaus können URI Variablen und HTTP-Parameter dargestellt werden.
- Views werden ebenfalls durch stereotypisierte Klassen abstrahiert. Es können der Name, die zugehörige URL und der Status einer Sitzung ausgedrückt werden.

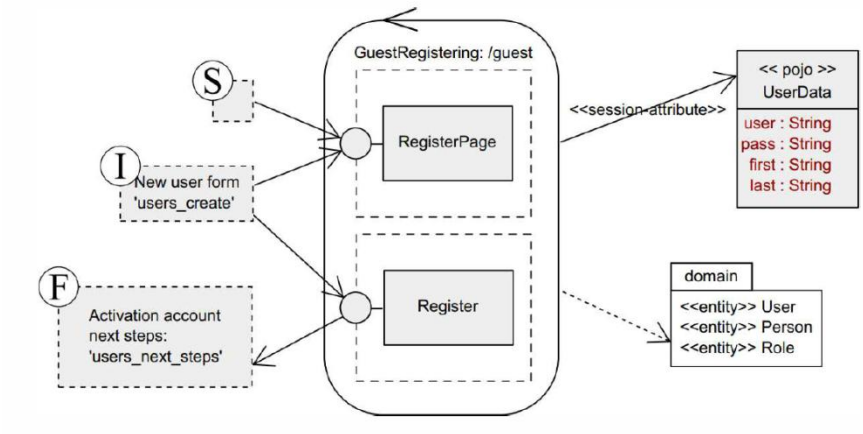


Abbildung 4: Verwendung des UML Profils [7]

Wie in obiger Abbildung zu erkennen, können damit Struktur und Interaktion eines Controllers (in diesem Fall für ein System zur Registrierung) in einem einzigen Diagramm dargestellt werden. Der Status der View wechselt dabei von Start über Intermediate zu Finish.

Zumindest für den Controller kann mit den angepassten Templates des Enterprise Architect der entsprechende Java-Code generiert werden.

```
@RequestMapping("/guest")
@Controller
@SessionAttributes({ "userData" })
public class GuestRegistering {
 @RequestMapping(value="register",
 method=RequestMethod.GET)
 public String registerPage(Model uiModel){
  // if (condition) {
  // return "users_create";
  // }
  return "";
 }

 @RequestMapping(method = RequestMethod.POST)
 public String register(Model uiModel){
  // if (condition) {
  // return "users_next_steps";
  // }
  return "";
 }
 public UserData m_UserData;
 public ActivationData m_ActivationData;
}
```

Abbildung 5: Generierter Code für den Controller [7]

3.3.2 Bewertung

Der Ansatz, ein Mittel zwischen anschaulicher Modellierung der Architektur und die Technologie berücksichtigendem Erzeugen von Code zu entwerfen, ist hier durchaus gelungen. Allerdings wirkt die beschriebene Neuerung dabei eher wie ein durchdachtes Anwenden der Möglichkeiten in Enterprise Architect, das nicht voll zur Entfaltung kommt.

Wie im obigen Schaubild zu erkennen, ist das Ergebnis - ungeachtet dessen – sehenswert. Es wurde ein sehr aussagekräftiger Typ von Diagramm entworfen, dem sich sowohl der Verlauf wie auch die Struktur eines Programms entnehmen lassen. Das Erzeugen von Quellcode ist generell sehr hilfreich, wenn auch noch ausbaufähig.

3.4 Modeling RESTful applications

3.4.1 Beschreibung

Silvia Schreier [8] stellt in ihrer Arbeit ein Metamodell vor, mit dem Schnittstellen von RESTful Webservices modelliert werden können. Sie legt dabei den Fokus auf das Design auf einer höheren Abstraktionsebene, losgelöst von technischen Details, und nennt als Ziel „model driven development".

Für die Beschreibung des Metamodells dient Ecore, ein Meta-Metamodell und Teil des „Eclipse Modeling Framework"[4].

Die Autorin gliedert die Beschreibung ihres Metamodells in zwei Teile - erstens „Structural Modeling", zweitens „Behavioral Modeling". Das strukturelle Modell umfasst ein Typensystem für Ressourcen, inklusive verschiedener Containertypen.

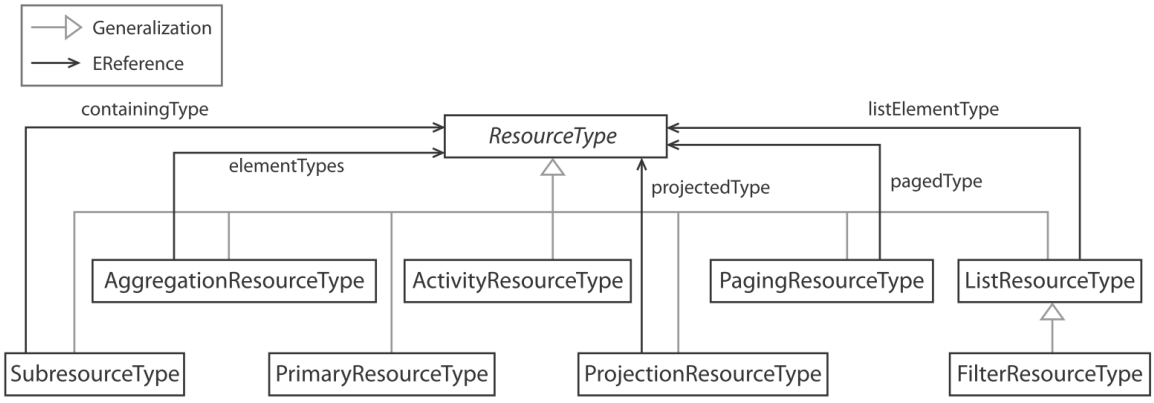


Abbildung 6: Typensystem für die Modellierung von Ressourcen [8]

Das Typensystem bietet einige Möglichkeiten, Ressourcen zu klassifizieren und miteinander in Verbindung zu setzen. Über den Typ „PrimaryResourceType" können abstrakte Vorlagen definiert werden, von denen verschiedene Instanzen erstellt werden können. Eine Hierarchie unter Ressourcen kann mit der Verwendung von „SubresourceType" ausgedrückt werden. Zusätzlich zum Typ können für Ressourcen Attribute vorgegeben werden, für die zum einen die von Java definierten primitiven Datentypen und zum anderen ein Collection Datentyp definiert werden kann.

Für das Modellieren von Methoden gibt es eine Klasse „Method", die auf eine der HTTP Methoden abbildet. Für diese kann außerdem eine Menge von „MediaTypes" und Parameter angegeben werden.

Für das „Behavioral Modeling" schlägt die Autorin vor, deterministische endliche Automaten zu verwenden. Für Methoden soll dann eine „Action" definiert werden, die Zustandsübergänge auslösen kann. Für Actions werden verschiedene weitere Typen definiert, unter anderem „CreateAction", welche neue Ressourcen erzeugt,

[4] (09.10.2013): http://www.eclipse.org/modeling/emf/

„UpdateAction“, um Properties von Ressourcen neu zu setzen, oder „ReturnAction“, um zurückgegebene Repräsentationen zu beschreiben.

3.4.2 Bewertung

Der Ansatz von Silvia Schreier behandelt eine andere Modellierungsebene als andere in dieser Fachstudie untersuchte Ansätze, unser Kriterienkatalog ließ sich deswegen hier nicht anwenden.

Uns gefällt das Metamodell bezüglich struktureller Modellierung sehr gut. Es werden verschiedene Ressourcentypen definiert und die Möglichkeit, Ressourcen zu klassifizieren, bringt eine Ausdrucksstärke mit sich, die in anderen Beschreibungssprachen nicht zu finden ist.

Der Vorschlag, deterministische endliche Automaten zu verwenden, um das Verhalten von Ressourcen zu modellieren, erscheint etwas aufwändig. Außerdem fehlen hier noch technische Details.

3.5 WRML

3.5.1 Beschreibung

WRML (Web Resource Modeling Language) wurde in dieser Fachstudie nur oberflächlich betrachtet. Zu Beginn der Fachstudie waren noch Informationen zu WRML unter www.wrml.org zu finden, jedoch war der Entwicklungsstand unklar und es gab keine Hinweise darauf, dass derzeit an WRML weitergearbeitet wird. Inzwischen verlinkt diese Seite auf ein GitHub-Projekt[5] zu WRML und die ehemals auf der Webseite zu Verfügung gestellten Informationen sind nur noch teilweise im Google-Cache zu finden. Auf GitHub gab es in den letzten zwei Monaten wieder Commits.

WRML wird als Framework beschrieben, das Tools und eine eigene Application Server Engine beinhaltet. Es führt zusätzlich einen eigenen Mediatype und Schemadefinitionen ein. Als Tools werden Wrmldoc und Werminal beschrieben. Wrmldoc ist eine webbasierte GUI, ähnlich zu Swagger (siehe Kapitel 4.6). Werminal ist, wie in unten stehender Abbildung zu erkennen, ein Command-line Tool.

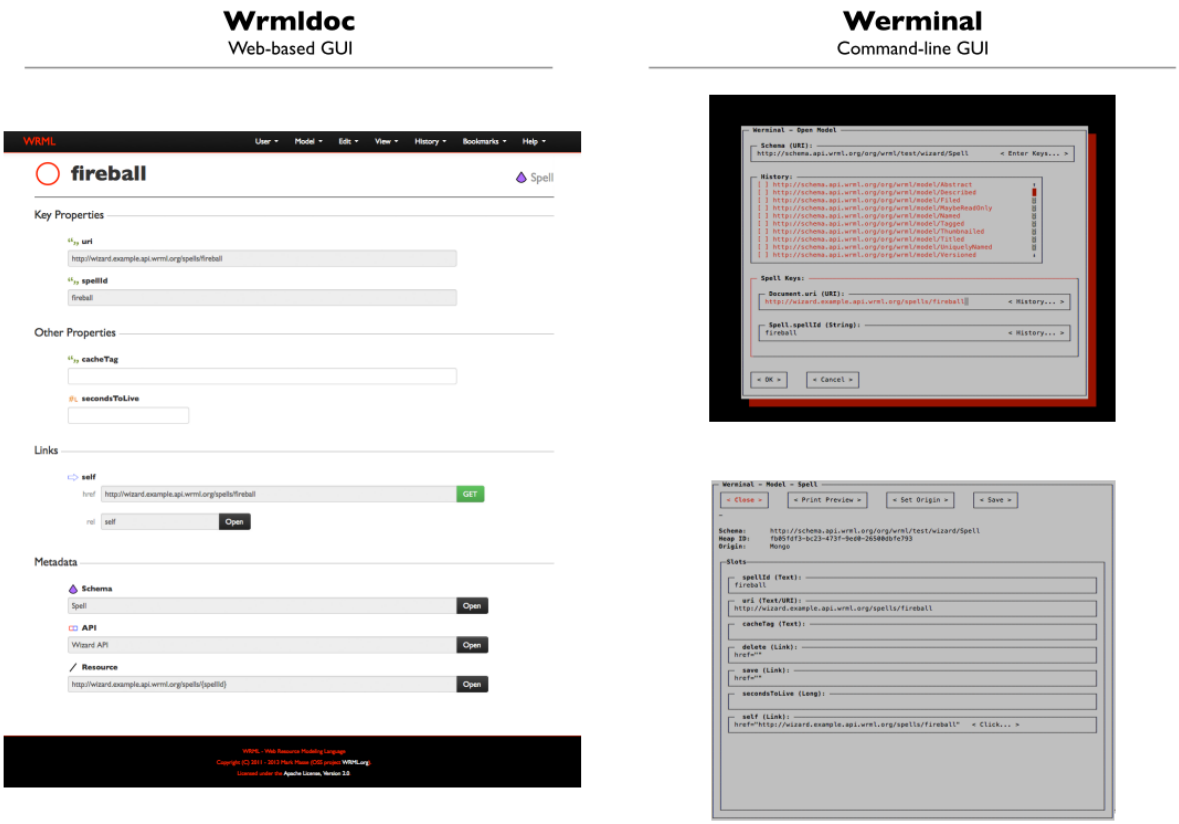


Abbildung 7: WRMLDoc und Werminal [9]

Die nächste Abbildung zeigt die Architektur des WRML Application Servers. Dessen Aufgabe ist es, HTTP-basierte Anfragen in den WRML Kontext zu übersetzen. Die

[5] (09.10.2013): https://github.com/wrml/wrml

einzelnen Komponenten (ApiLoader, SchemaLoader, etc.) kümmern sich dann um entsprechende Teile der Payload.

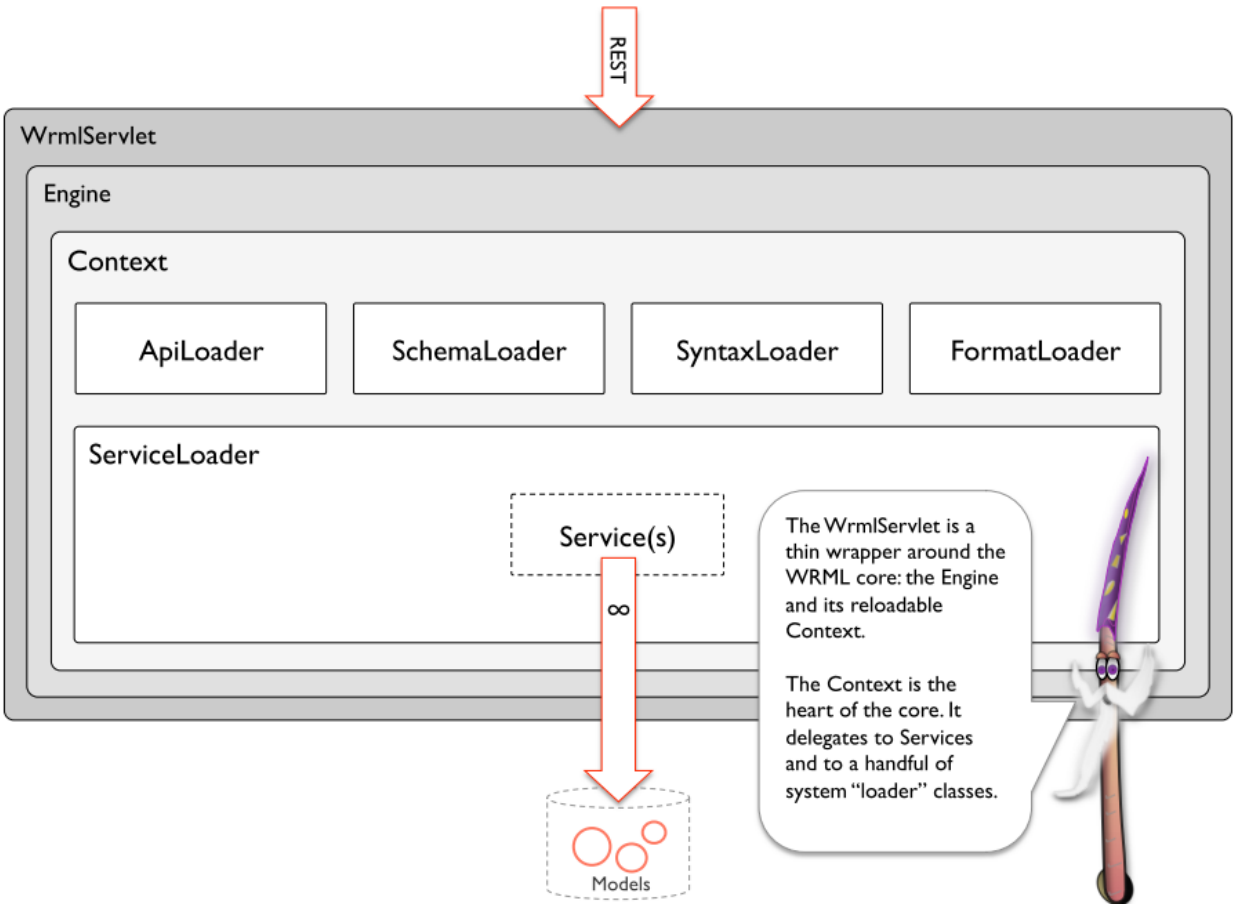


Abbildung 8: Architektur des WRML Application Servers [9]

Auf Amazon lässt sich ein Buch[6] des Autors über REST und WRML finden, in welchem er neben einigen Techniken zum Implementieren von REST Services auch WRML vorstellt. Die Rezessionen dort fallen jedoch sehr negativ aus, vor allem WRML wird in den Kommentaren stark kritisiert, weil es zu komplex sei.

3.5.2 Bewertung

WRML wurde mangels Informationen und wegen genannter Gründe in dieser Studie nicht bewertet.

[6] (09.10.2013): http://www.amazon.de/Rest-Design-Rulebook-Mark-Masse/dp/1449310508/

3.6 RESTdesc

3.6.1 Beschreibung

RESTdesc beschreibt den Ansatz, „Notation 3“ [10] zu verwenden, um semantische Aspekte von Webservices auszudrücken [11]. N3 bietet aussagenlogische Konstrukte wie Implikationen und Quantoren, sodass Vor- und Nachbedingungen für den Aufruf von Webservices formuliert werden können. Im nachfolgenden Beispiel, das die Autoren auf ihrer Homepage[7] angeben, wird ausgedrückt, dass - sofern für ein Bild ein Link vom Typ „dpedia-owl:thumbnail“ existiert - auf diesem Link eine HTTP Get Methode aufgerufen werden kann, die ein Bild der Höhe 80 Pixel zurückliefert.

```
@prefix : <http://example.org/image#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.
{
      ?image :smallThumbnail ?thumbnail.}
=>    {
      _:request http:methodName "GET";
            http:requestURI ?thumbnail;
            http:resp [ http:body ?thumbnail ].
      ?image dbpedia-owl:thumbnail ?thumbnail.
      ?thumbnail a dbpedia:Image;
            dbpedia-owl:height 80.0.}.
```

Listing 2: RESTDesc Beispielcode

Entscheidend für die Aussagekraft solcher Beschreibungen sind natürlich die Vokabulare der verwendeten Ontologien. Die Autoren schlagen die Verwendung des HTTP Vokabulars[8] vor, um alle Aspekte von HTTP Methoden beschreiben zu können. Weiterhin kann jede in RDF formulierte Ontologie verwendet werden.

Damit die Verwendung von Platzhaltern in URI Beschreibungen erklärt werden kann, wurde eine eigene Ontologie entworfen[9].

Die Autoren erklären, dass Deduktionen verwendet werden können, falls Vor- und Nachbedingungen in N3 formuliert sind [11]. Damit kann ein Clientprogramm für eine Ausgangsbedingung feststellen, welche Webservices aufzurufen sind, um ein definiertes Ziel zu erreichen, falls Ausgangsbedingung und Ziel die Ontologien verwendet, in denen auch die Beschreibungen der Webservices formuliert sind.

[7] (09.10.2013) http://restdesc.org/about/descriptions
[8] (09.10.2013) http://www.w3.org/TR/HTTP-in-RDF10/#classes
[9] (09.10.2013) http://multimedialab.elis.ugent.be/organon/ontologies/restdesc/uri-template

3.6.2 Bewertung

RESTDesc führt keine eigene Notation ein, sondern verwendet „Notation 3“, um semantische Aspekte von Webservices auszudrücken. Die Aussagekraft dieser Beschreibungen hängt stark von der verwendeten Ontologie ab, die beliebig gewählt werden kann. Aus diesen Gründen haben wir RESTDesc nicht mit unserem Kriterienkatalog bewertet.

Die N3 Notation erlaubt es über Konstrukte wie Quantoren und Implikationen und die Verwendung von beliebigen Ontologien, komplizierte Zusammenhänge formal auszudrücken. Wir glauben, dass Formalismen dieser Art unabdingbar sind, um wirklich automatisierte „service discovery“ zu entwickeln. Was RESTDesc allerdings nicht ohne weiteres ausdrücken kann, sind syntaktische Vorgaben für Interfaces von Webservices zu formulieren, beispielsweise Schemadefinitionen für erlaubte Inputs und Outputs anzugeben. Wir glauben, dass hier zusätzlich eine Möglichkeit verwendet werden muss, um die syntaktische Struktur der Interfaces zu beschreiben.

4 Mit dem Kriterienkatalog bewertete Spezifikationen und Frameworks

In diesem Kapitel wird die Bewertung einzelner Spezifikationen und Frameworks anhand des Kriterienkatalogs durchgeführt. Die Bewertung einzelner Fragen erfolgt dabei auf einer Skala von 0 bis 3. Die Punktzahl 0 bedeutet, dass die durch die Frage implizierte Anforderung nicht erfüllt wurde. Die Punktezahlen 1 (in Ansätzen erfüllt), 2 (zu Teilen erfüllt) und 3 (vollständig erfüllt) geben den Grad an, in welchem der Prüfling die Anforderung abdeckt.

Für jeden Prüfling sollen Beschreibung und Beispiel einen Überblick über dessen Eigenschaften geben und Bewertung und Fazit diese Eigenschaften evaluieren. Eine abschließende Bewertung in Relation zu anderen Prüflingen befindet sich im nächsten Kapitel.

4.1 WADL

4.1.1 Beschreibung

WADL [12] (Web Application Description Language) wird auf Wikipedia[10] als "machine-readable XML description of HTTP-based web applications (typically REST web services)" bezeichnet. Es ist also eine auf XML basierende Beschreibungssprache für HTTP-Schnittstellen. Laut dieser Quelle ist WADL das „REST equivalent of SOAP's [...] WSDL", bietet aber die entscheidende Erweiterung, dass Ressourcen eine direkte Repräsentation im XML Schema haben, an. Außerdem können diesen Ressourcen zugeordnete Methoden modelliert werden, die genau auf die HTTP Methoden abgebildet werden.

WADL ist eine „W3C Member Submission", und ist unserer Meinung nach (von WSDL abgesehen) der Ansatz mit der ausführlichsten und exaktesten Spezifikation.

4.1.2 Beispiel

Das nachfolgende Beispiel stammt aus der oben genannten Spezifikation und ist stark gekürzt:

```
<resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
  <resource path="newsSearch">
    <method name="GET" id="search">
      <request>
        <param     name="appid"     type="xsd:string"     style="query"
required="true"/>
        <param     name="query"     type="xsd:string"     style="query"
required="true"/>
        <param name="type" style="query" default="all">
          <option    value="all"/>    <option    value="any"/>    <option
value="phrase"/>
        </param>
        <param     name="results"     style="query"     type="xsd:int"
default="10"/> […]
        <response status="200">
      </response> </method>
  </resource>
</resources>
```

Listing 3: WADL Beispielcode

[10] (09.10.2013): http://en.wikipedia.org/wiki/Web_Application_Description_Language

4.1.3 Bewertung

4.1.3.1 Syntax

Können die Ressourcen der Schnittstelle definiert werden?

2 Punkte: In WADL können für die Beschreibung von Ressourcen Beschreibungen und IDs hinterlegt werden. Außerdem ist es möglich, Ressourcen hierarchisch zu schachteln.

Können pro Ressource erlaubte Operationen definiert werden?

3 Punkte: Das WADL Schema definiert das Element „method“, das über das Attribut „name“ einer HTTP Methode zugeordnet wird.

Können erwarteter Input bzw. Output des Servers definiert werden?

3 Punkte: In WADL kann unter dem „method“ Element ein „request“, bzw. „response“ Element angehängt werden, für das wiederum eine Schemadefinition hinterlegt werden kann, die mögliche Inputs, bzw. Outputs definiert.

Können Beziehungen zwischen Ressourcen definiert werden?

1 Punkt: WADL bietet die Möglichkeit, Ressourcen zu schachteln, wodurch hierarchische Beziehungen ausgedrückt werden können. Andere Formen von Beziehungen können nicht modelliert werden.

Kann für Methoden beschrieben werden, welche Statuscodes erzeugt werden können?

3 Punkte: In WADL kann pro „response“, bzw. „request“ Element eine Liste von HTTP Statuscodes angegeben werden.

Bietet die Sprache vordefinierte Ressourcentypen an?

0 Punkte: WADL bietet zwar die Möglichkeit, eigene Typen für Ressourcen zu definieren, allerdings bietet die Sprache keine vordefinierten Typen.

Können Ressourcen geschachtelt werden, sodass URIs relativ zur Eltern-Ressource definiert werden können?

3 Punkte: Ressourcen können hierarchisch geschachtelt werden, die URI Definitionen gelten dann relativ zur übergeordneten Ressource.

Können in URI-Definitionen Templates verwendet werden?

3 Punkte: WADL erlaubt es, in URI Definitionen Templates in geschweiften Klammern anzugeben, die in einem angehängten „param“ Element definiert werden können.

4.1.3.2 Semantik

Kann die Bedeutung einer Ressource definiert werden?

0 Punkte: Mit WADL kann keine Semantik ausgedrückt werden.

Kann die Bedeutung einer Operation definiert werden?

0 Punkte: Mit WADL kann keine Semantik ausgedrückt werden.

Kann die Bedeutung einer Beziehung zwischen Ressourcen definiert werden?

0 Punkte: Mit WADL kann keine Semantik ausgedrückt werden.

4.1.3.3 Lesbarkeit

Wie gering ist der Anteil an Syntax-Overhead im Code?

1 Punkt: Durch das Verwenden von XML entsteht relativ viel Overhead, vor allem durch schließende Tags.

Wie intuitiv ist den Code-Konstrukten ihre Bedeutung anzusehen?

3 Punkte: Die Verwendung von XML ist hilfreich, um hierarchische Beziehungen darzustellen. Die definierten Schlüsselworte sind treffend gewählt:

```
<resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
  <resource path="newsSearch">
    <method name="GET" id="search">
      <request>
        <param name="appid" type="xsd:string"
```

Listing 4: Schlüsselwörter in WADL

4.1.3.4 Anwendung der Sprache

In welchem Umfang wird die Sprache bereits eingesetzt?

2 Punkte: Es sind diverse Blogs, Forumsposts, und Diskussionen zu WADL zu finden. Apache bietet das Framework „CXF“, welches WADL unterstützt[11]. Außerdem bietet IBM Produkte an[12], die WADL unterstützen. Trotzdem ist der Eindruck entstanden, dass WADL bisher noch nicht in großem Rahmen unterstützt wird.

Wie gut ist die Sprache dokumentiert

2 Punkte: Die Spezifikation auf der W3-Website ist ausführlich und bietet einige Beispiele.

[11] (09.10.2013) http://cxf.apache.org/docs/jaxrs-services-description.html
[12] (09.10.2013) http://pic.dhe.ibm.com/infocenter/dmanager/v8r0m1/index.jsp?topic=%2Fcom.ibm.wodm.dserver%2Ftopics%2Fwodm_dserver.html

Wie gut sind die Tools dokumentiert, die für diese Sprache angeboten werden?

3 Punkte: Es gibt ausführliche Installationsbeschreibungen und Anleitungen, auch mit Bildern.

Einfachheit: Wie gering ist der Aufwand zur Einarbeitung in die Sprache?

2 Punkte: Auf Grund der umfangreichen Spezifikation ist einiger Aufwand erforderlich, um WADL vollständig zu erfassen.

Wie gut wird der Entwickler beim Modellieren bestehender Schnittstellen unterstützt?

1 Punkt: Es gibt ein freies Browser-Tool „REST Describe“[13], über das Schnittstellenbeschreibungen interaktiv erstellt werden können.

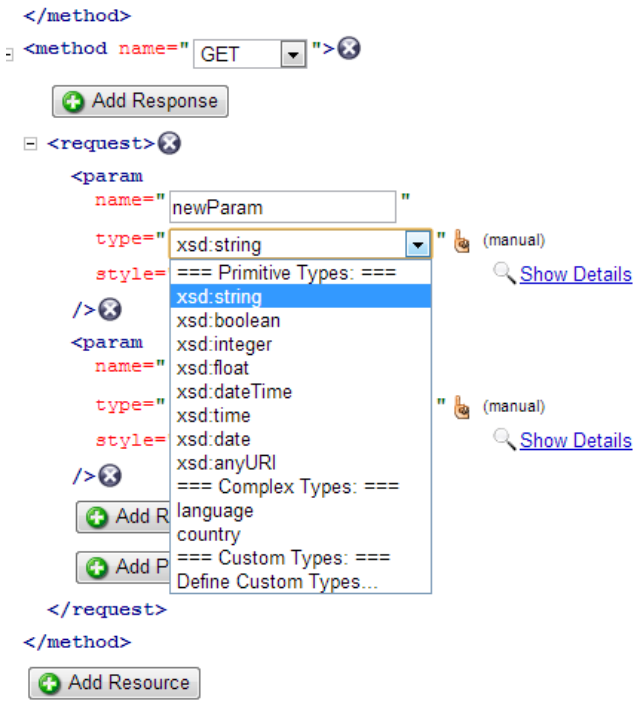


Abbildung 9: REST Describe

Außerdem erlaubt das „CFX“ Framework von Apache, Java Methoden mit Annotationen zu versehen:

[13] (09.10.2013) http://tomayac.de/rest-describe/latest/RestDescribe.html

```
@POST
@Path("books/{bookid}")
@Descriptions({
      @Description(value = "Adds a new book", target =
DocTarget.METHOD),
      @Description(value = "Requested Book", target = DocTarget.RETURN),
      @Description(value = "Request", target = DocTarget.REQUEST),
      @Description(value = "Response", target = DocTarget.RESPONSE),
      @Description(value = "Resource", target = DocTarget.RESOURCE)})
```

Listing 5: Java Annotationen für WADL

Gibt es eine hilfreiche, graphische Darstellung der Modelle?

1 Punkt: Das Browser-Tool „REST Describe“ (s.o.) kann Schnittstellenbeschreibungen hierarchisch darstellen.

Unterstützung für: Schnittstellen sollen in einem maschinenlesbaren Format beschrieben werden.

3 Punkte: Die mit dem Browser Tool „REST Describe“ (s.o.) erstellten Schnittstellenbeschreibungen können als WADL-Dokument heruntergeladen werden. Sind Java Annotationen in korrekter Syntax vorhanden, kann das „CXF“ Framework von Apache automatisiert Java Klassen erzeugen..

Unterstützung für: Aus der entworfenen Schnittstelle sollen Code-Stubs generiert werden können.

3 Punkte: Das Tool „wadl2Java“[14] kann clientseitig Codestubs generieren. Es setzt auf der Jax-RS Api auf, und erzeugt aus WADL Beschreibungen Java-Klassen. Es kann über die Kommandozeile, als Ant-Plugin, oder als Maven Plugin ausgeführt werden.

4.1.4 Fazit

WADL wirkt insgesamt „erwachsen“. Es bietet (bis auf WSDL) die ausführlichste und genaueste Spezifikation und setzt mit XML auf ein erfolgreiches und weit verbreitetes Format. Uns gefallen die Möglichkeiten, Ressourcen in Hierarchien zu modellieren, und Ressourcentypen definieren zu können.

Über WADL ist mehr Information, Dokumentation, und Diskussion zu finden, als zu jeder anderen Beschreibungssprache. Sprechend ist auch, dass sich IBM und Apache bereits damit befasst haben.

[14] (09.10.2013) https://wadl.java.net/wadl2java.html

4.2 WSDL 2.0

4.2.1 Beschreibung

WSDL 2.0 [13] (Web Service Definition Language) ist der Quasi-Nachfolger von WSDL 1.1 (Web Service Description Language) und im Unterschied zu diesem eine Recommendation des W3C. Neben einigen kosmetischen Änderungen, wie der Umbenennung von portType in Interface, ist es nun auch möglich, REST Dienste in WSDL zu beschreiben. Dies wird dadurch ermöglicht, dass in WSDL 2.0 Bindings zu allen HTTP Methoden angegeben werden können[15]. WSDL 2.0 verfolgt den service- bzw. operationenorientierten Ansatz.

4.2.2 Beispiel

Nachfolgendes Listing zeigt eine WSDL 2.0 Beispiel-Datei. Für REST Services sind vor allem der Interface- und Binding-Tag interessant, weil hier die Funktionen beschrieben werden, die durchgeführt werden können.

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
   targetNamespace="http://www.bookstore.org/booklist/wsdl"
   xmlns:tns="http://www.bookstore.org/booklist/wsdl"
   xmlns:whttp="http://www.w3.org/ns/wsdl/http"
   xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:msg="http://www.bookstore.org/booklist/xsd">
  <wsdl:documentation>
     This is a WSDL 2.0 description of a sample bookstore service
     listing for obtaining book information.
  </wsdl:documentation>

  <wsdl:types>
    <xs:import namespace="http://www.bookstore.org/booklist/xsd"
       schemaLocation="booklist.xsd"/>
  </wsdl:types>
```

[15] (09.10.2013): http://www.w3.org/TR/wsdl20-primer/#basics-interface

```
  <wsdl:interface name="BookListInterface">
    <wsdl:operation name="getBookList"
       pattern="http://www.w3.org/ns/wsdl/in-out"
       style="http://www.w3.org/ns/wsdl/style/iri"
       wsdlx:safe="true">
      <wsdl:documentation>
         This operation returns a list of books.
      </wsdl:documentation>
      <wsdl:input element="msg:getBookList"/>
      <wsdl:output element="msg:bookList"/>
    </wsdl:operation>
  </wsdl:interface>

  <wsdl:binding name="BookListHTTPBinding"
     type="http://www.w3.org/ns/wsdl/http"
     interface="tns:BookListInterface">
    <wsdl:documentation>
       The RESTful HTTP binding for the book list service.
    </wsdl:documentation>
    <wsdl:operation ref="tns:getBookList" whttp:method="GET"/>
  </wsdl:binding>

  <wsdl:service name="BookList" interface="tns:BookListInterface">
    <wsdl:documentation>
       The bookstore's book list service.
    </wsdl:documentation>
    <wsdl:endpoint name="BookListHTTPEndpoint"
       binding="tns:BookListHTTPBinding"
       address="http://www.bookstore.com/books/">
    </wsdl:endpoint>
  </wsdl:service>
</wsdl:description>
```

Listing 6: Beschreibung einer Schnittstelle mit WSDL[16]

[16] (09.10.2013): http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/

4.2.3 Bewertung

4.2.3.1 Syntax

Können die Ressourcen der Schnittstelle definiert werden?

2 Punkte: Ressourcen können nicht direkt modelliert werden, sondern lediglich Endpoints von Services mit bestimmten Interfaces, die Operationen zur Verfügung stellen. Somit müssen Ressourcen in WSDL 2.0 als Interfaces beschrieben werden.

Können pro Ressource erlaubte Operationen definiert werden?

3 Punkte: Ja, alle Operationen können beschrieben werden, sowie der Inhalt der zu erwartenden Input- bzw. Outputmessage.

Können erwarteter Input bzw. Output des Servers definiert werden?

3 Punkte: Ja, mit Hilfe von XML Schemata können diese definiert werden.

Können Beziehungen zwischen Ressourcen definiert werden?

2 Punkte: WSDL 2.0 ermöglicht es, Interfaces zu vererben. Eine konkrete Möglichkeit die Beziehungen zwischen Ressourcen zu modellieren existiert nur, wenn man das in WSDL 2.0 spezifizierte Mapping zu RDF/OWL nutzt, um seine WSDL-File in RDF zu übersetzen und diese anschließend mit anderen semantischen Informationen zusammenführt.[17]

Kann für Methoden beschrieben werden, welche Statuscodes erzeugt werden können?

1 Punkt: Es ist nur für Fehlernachrichten (Faults) möglich, einen HTTP-Statuscode anzugeben.

Bietet die Sprache vordefinierte Ressourcentypen an?

0 Punkte: Nein, dies ist in WSDL 2.0 nicht möglich.

Können Ressourcen geschachtelt werden, sodass URIs relativ zur Eltern-Ressource definiert werden können?

0 Punkte: Nein, dies ist in WSDL 2.0 nicht möglich.

Können in URI-Definitionen Templates verwendet werden?

3 Punkte: Ja, mit Hilfe von Tokens. [14]

[17] (09.10.2013): http://www.w3.org/TR/2007/NOTE-wsdl20-rdf-20070626/

4.2.3.2 Semantik

Kann die Bedeutung einer Ressource definiert werden?

1 Punkt: Nur möglich unter Zuhilfenahme des RDF-Mappings und der Definition und Nutzung weiterer RDF-Dokumente.

Kann die Bedeutung einer Operation definiert werden?

1 Punkt: Nur möglich unter Zuhilfenahme des RDF-Mappings und der Definition und Nutzung weiterer RDF-Dokumente.

Kann die Bedeutung einer Beziehung zwischen Ressourcen definiert werden?

1 Punkt: Nur möglich unter Zuhilfenahme des RDF-Mappings und der Definition und Nutzung weiterer RDF-Dokumente.

4.2.3.3 Lesbarkeit

Wie gering ist der Anteil an Syntax-Overhead im Code?

1 Punkt: Sehr großer Overhead. WSDL 2.0 nutzt XML und hat daher ein sehr verboses Erscheinungsbild. Hinzu kommt der große spezifizierte Umfang von WSDL 2.0, sodass zum Teil umständliche Konstrukte nötig sind, um vergleichsweise einfache Dinge zu modellieren.

Wie intuitiv ist den Code-Konstrukten ihre Bedeutung anzusehen?

2 Punkte: Auch für Menschen die sich bereits mit WSDL 1.1 auskennen, ist WSDL 2.0 nicht sofort zu durchschauen, weil zum Beispiel einige Namensänderungen an Tags durchgeführt wurden. Hinzu kommt, dass WSDL 2.0 nicht speziell für REST entwickelt wurde.

4.2.3.4 Anwendung der Sprache

In welchem Umfang wird die Sprache bereits eingesetzt?

1 Punkt: WSDL 2.0 ist seit 2007 eine W3C Recommendation, allerdings ist WSDL 2.0 bei weitem nicht so verbreitet wie sein Vorgänger WSDL 1.1.

Wie gut ist die Sprache dokumentiert?

2 Punkte: Die Sprache ist ausführlich dokumentiert, jedoch ohne Beispiele.[18]

Wie gut sind die Tools dokumentiert, die für diese Sprache angeboten werden?

1 Punkt: Zum Zeitpunkt der Erstellung dieser Arbeit waren nur wenige Tools im Internet vorhanden und deren Entwicklungsstatus zum Teil unbekannt (siehe Apache Woden[19]).

[18] (09.10.2013): http://www.w3.org/TR/2007/REC-wsdl20-20070626/

Einfachheit: Wie gering ist der Aufwand zur Einarbeitung in die Sprache?

1 Punkt: Aufgrund der sehr komplexen Spezifikation besteht ein relativ hoher Aufwand.

Wird der Entwickler beim Modellieren unterstützt?

1 Punkt: Aufgrund mangelnden Toolsupports, vor allem für REST/ROA, kann nur auf andere XML-Werkzeuge zurückgegriffen werden.

Gibt es eine hilfreiche, graphische Darstellung der Modelle?

0 Punkte: Nein, uns ist kein Tool bekannt.

Unterstützung für: Schnittstellen sollen in einem maschinenlesbaren Format beschrieben werden.

3 Punkte: Ja, weil sämtliche Dateien mit Hilfe von XML/XSD beschrieben werden.

Unterstützung für: Aus der entworfenen Schnittstelle sollen Code-Stubs generiert werden können.

1 Punkt: Dies ist prinzipiell möglich, aber bislang fehlen die nötigen Tools. Vor allem für das Erstellen von REST Services.

4.2.4 Fazit

Man merkt WSDL 2.0 seine SOA-Herkunft an. Im Gegensatz zu WSDL 1.1 ist es mit WSDL 2.0 inzwischen zwar möglich REST Services zu beschreiben, jedoch wirkt sich die fehlende Ressourcenorientierung und mangelnde Toolunterstützung negativ aus. Zum jetzigen Zeitpunkt lässt sich auch der Erfolg von WSDL 2.0 noch nicht absehen. Es ist fraglich ob es die sehr weite Verbreitung von WSDL 1.1 jemals erreichen wird.

[19] (09.10.2013): http://ws.apache.org/woden/

4.3 hRESTS

4.3.1 Beschreibung

„hRESTS“ [15] ist ein auf „Notation 3“[20] basierendes Mikroformat, in welchem RESTful Webservices beschrieben werden können. Das bedeutet, es ist dafür entworfen, vollständig in HTML-, oder XHTML Code eingebettet zu werden. Diese Form wird dadurch begründet, dass der größte Teil der vorhandenen Dokumentation von RESTful Webservices in Textform und für Maschinen nicht verwertbar in HTML Format vorliegt. Die Idee ist also, diese schon verfügbare Dokumentation mit maschinenverwertbaren Informationen über den Service zu erweitern. „hRESTS“ betont dabei die Einfachheit, das Modell beschränkt sich auf einige wenige Klassen.

4.3.2 Beispiel

Im Beispiel ist eine textuelle Beschreibung eines Webservice um Maschinenlesbare „hRESTS“ Beschreibungen (hervorgehoben) erweitert worden. Es kann formal ausgedrückt werden, was Beschreibung des Input, und was Beschreibung des Output ist, diese werden der HTTP Get-Methode zugeordnet.

```
<div class="service" id="svc">
<p>Description of the <span class="label">ACME Hotels</span>
service:</p>
<div class="operation" id="op1"><p>
    The operation <code class="label">getHotelDetails</code> is
    invoked using the method <span class="method">GET</span>
    at <code class="address">http://example.com/h/{id}</code>,
    with <span class="input">the ID of the particular hotel replacing
        the parameter <code>id</code></span>.
    It returns <span class="output">the hotel details in an
        <code>ex:hotelInformation</code> document.</span>
</p></div></div>
```

Listing 7: hRESTS Beispielcode

4.3.3 Bewertung

4.3.3.1 Syntax

Können die Ressourcen der Schnittstelle definiert werden?

0 Punkte: „hRESTS“ ist nicht ressourcenorientiert, sondern betrachtet Services und Operationen. Es gibt im Modell von hRESTS keine direkte Darstellung für Ressourcen.

[20] (09.10.2013): http://en.wikipedia.org/wiki/Notation3

Können pro Ressource erlaubte Operationen definiert werden?

3 Punkte: Es können erlaubte Operationen einem Service zugeordnet aufgelistet und auf HTTP-Methoden abgebildet werden.

Können erwarteter Input bzw. Output des Servers definiert werden?

1 Punkt: „hRESTS“ bietet keine Möglichkeit, Schemadefinitionen für erlaubte Ressourcentypen zu definieren. Es kann nur definiert werden, dass eine Methode einen Input, bzw. einen Output hat.

Können Beziehungen zwischen Ressourcen definiert werden?

0 Punkte: „hRESTS“ bietet keine Darstellung für Ressourcen, und damit auch keine Beziehungen zwischen Ressourcen.

Kann für Methoden beschrieben werden, welche Statuscodes erzeugt werden können?

0 Punkte: Es gibt keine Möglichkeit, mögliche Statuscodes aufzulisten.

Bietet die Sprache vordefinierte Ressourcentypen an?

0 Punkte: „hRESTS“ definiert keine Ressourcentypen.

Können Ressourcen geschachtelt werden, sodass URIs relativ zur Eltern-Ressource definiert werden können?

0 Punkte: „hRESTS“ bietet keine Darstellung für Ressourcen.

Können in URI-Definitionen Templates verwendet werden?

3 Punkte: In URI Definitionen können Platzhalter der Form „{id}“ verwendet werden.

4.3.3.2 Semantik

Kann die Bedeutung einer Ressource definiert werden?

0 Punkte: „hRESTS“ bietet keine Möglichkeit, Semantik auszudrücken.

Kann die Bedeutung einer Operation definiert werden?

0 Punkte: „hRESTS“ bietet keine Möglichkeit, Semantik auszudrücken.

Kann die Bedeutung einer Beziehung zwischen Ressourcen definiert werden?

0 Punkte: „hRESTS“ bietet keine Möglichkeit, Semantik auszudrücken.

4.3.3.3 Lesbarkeit

Wie gering ist der Anteil an Syntax-Overhead im Code?

3 Punkte: Ausgehend von bestehendem HTML-Code sind die zusätzlichen Annotationen sehr knapp, der Overhead ist minimal. (siehe Beispiel oben)

Wie intuitiv ist den Code-Konstrukten ihre Bedeutung anzusehen?

1 Punkt: Die Einbettung in HTML Code bringt es mit sich, dass die „hRESTS“ Beschreibungen zwischen HTML Tags stehen, und dadurch schwer lesbar sind, weil die Anreicherungen zwischen HTML-Konstrukten gesucht werden muss.

4.3.3.4 Anwendung der Sprache

In welchem Umfang wird die Sprache bereits eingesetzt?

0 Punkte: Uns ist kein Einsatz von „hRESTS“ bekannt.

Wie gut ist die Sprache dokumentiert

2 Punkte: „hRESTS“ wird in der Arbeit [16] vorgestellt, außerdem gibt es einen Webauftritt[21], der im Wesentlichen den Inhalt des Papers enthält, um einige Beispiele erweitert.

Wie gut sind die Tools dokumentiert, die für diese Sprache angeboten werden?

0 Punkte: Es gibt ein XSLT Style-Sheet[22] ohne nennenswerte Dokumentation, welches hREST Beschreibungen aus XHTML Code extrahieren kann. Ansonsten gibt es für „hRESTS“ keine Tools.

Einfachheit: Wie gering ist der Aufwand zur Einarbeitung in die Sprache?

3 Punkte: Die Sprache bietet eine sehr überschaubare Anzahl an Möglichkeiten an und ist deshalb leicht zu erlernen.

Wie gut wird der Entwickler beim modellieren bestehender Schnittstellen unterstützt?

0 Punkte: Es gibt keine Tools, die den Entwickler beim Erstellen der „hRESTS“ Beschreibungen unterstützen.

Gibt es eine hilfreiche, graphische Darstellung der Modelle?

0 Punkte: Es gibt keine grahische Darstellung für „hRESTS“ Beschreibungen.

[21] (09.10.2013): http://knoesis.org/research/srl/projects/hRESTs
[22] (09.10.2013): http://members.sti2.at/~jacekk/hrests/hrests.xslt

Unterstützung für: Schnittstellen sollen in einem maschinenlesbaren Format beschrieben werden.

1 Punkt: Für XHTML Code, der mit hREST angereichert ist, kann über ein XSLT Stylesheet automatisch RDF generiert werden.

Unterstützung für: Aus der entworfenen Schnittstelle sollen Code-Stubs generiert werden können.

0 Punkte: Es gibt keine Tools, die aus „hRESTS“ Beschreibungen Codestubs generieren können.

4.3.4 Fazit

„hRESTS“ bringt als Mikroformat den Vorteil, in bestehenden XHTML Code integriert werden zu können und ist durch sein schlankes Modell schnell zu erfassen. Allerdings können einige elementaren Begriffe zum Prinzip REST nicht beschrieben werden, allen voran der Begriff „Ressource“.

4.4 MicroWSMO

4.4.1 Beschreibung

Mit WSMO-Lite [17] ist eine Menge von Begriffen angeboten, die über SAWSDL (SemanticAnnotationsforWSDLandXMLSchema) Annotationen [18] verwendet werden können, um semantische Zusammenhänge im Bereich Webservices auszudrücken. Diese Ontologie wurde für die Verwendung in WSDL Beschreibungen entwickelt und eignet sich ohne Modifikation nicht für die Beschreibung von RESTful Services.

In der Ausarbeitung zu MicroWSMO [19] erarbeiten die Autoren verschiedene Servicemodelle als RDF Schemadefinitionen, die auf das WSMO-Lite Servicemodell aufbauend zusammen mit einer Erweiterung des „hRESTS“ [16] Servicemodell verwendet werden können, um RESTful Webservices zu beschreiben. Dafür wird ein „Resource oriented service model“ (ROSM) eingeführt, welches RDF-Klassen wie „ServicedResource“, „Address“, „Operation“, und „Parameter“ einführt. Für die Beschreibung von semantischen Zusammenhängen greifen die Autoren auf SAWSDL Annotationen zurück, für die sie die WSMO-Lite Ontologie verwenden.

4.4.2 Beispiel

Im Beispiel ist die Beschreibung eines Webservice in HTML um MicroWSMO Beschreibungen erweitert (hervorgehoben). [15]

```
<div class=”service” id=”svc”>
  <h1><span class=”label”>ACME Hotels</span> service API</h1>
    <p>This service is a
      <a                                                    rel=”model”
href=”http://example.com/ecommerce/hotelReservation”>
        hotel reservation</a> service. </p>
  <div class=”operation” id=”op1”>
    <h2>Operation <code class=”label”>getHotelDetails</code></h2>
    <p> Invoked using the <span class=”method”>GET</span>
      at <code class=”address”>http://example.com/h/{id}</code><br/>
      <span class=”input”>
        <strong>Parameters:</strong>
        <a   rel=”model”   href=”http://example.com/data/onto.owl#Hotel”>
[…]
```

Listing 8: MicroWSMO Beispielcode

Hier sind vor allem die semantischen Modellreferenzen zu den Klassen „hotelReservation“ und „hotel“ zu bemerken.

4.4.3 Bewertung

4.4.3.1 Syntax

Können die Ressourcen der Schnittstelle definiert werden?

2 Punkte: MicroWSMO definiert die RDF Klasse „ServicedResource“ mit Zuordnung zu einem Service, mit der Ressourcen dargestellt werden können.

Können pro Ressource erlaubte Operationen definiert werden?

3 Punkte: MicroWSMO bietet als RDF Property „supportsOperation“ eine Zuordnung von „Operation“ Klassen zu Ressourcen. Die Methoden können wiederum auf die HTTP-Methoden aus dem HTTP Namespace abgebildet werden.[23]

Können erwarteter Input bzw. Output des Servers definiert werden?

3 Punkte: An dieser Stelle sehen die Autoren vor, den SAWSDL Namespace[24] zu verwenden, damit kann als „modelReference“ eine URI zu einer Schemadefinition verlinkt werden.

Können Beziehungen zwischen Ressourcen definiert werden?

1 Punkt: Es können „ResourceCollections“ definiert werden, und Ressourcen dieser Collection zugeordnet werden, allerdings ist die Sprache auf solche hierarchischen Beziehungen beschränkt, es können keine beliebigen Referenzen definiert werden.

Kann für Methoden beschrieben werden, welche Statuscodes erzeugt werden können?

3 Punkte: Ja, MicroWSMO verwendet dafür die „ResponseCode“ Klasse aus dem HTTP Namespace.

Bietet die Sprache vordefinierte Ressourcentypen an?

1 Punkt: MicroWSMO bietet die RDF Klasse „ServicedResourceCollection“, die wiederum mehrere „ServicedResource“ Instanzen enthalten kann.

Können Ressourcen geschachtelt werden, sodass URIs relativ zur Eltern-Ressource definiert werden können?

0 Punkte: MicroWSMO erlaubt es zwar, Ressourcen hierarchisch anzuordnen, URIs müssen aber immer absolut angegeben werden.

Können in URI-Definitionen Templates verwendet werden?

3 Punkte: MicroWSMO erlaubt es, Teile der URI als Templates in der Form “{id}” anzugeben.

[23] (09.10.2013): http://www.w3.org/2006/http#
[24] (09.10.2013): http://www.w3.org/ns/sawsdl#

4.4.3.2 Semantik

Kann die Bedeutung einer Ressource definiert werden?

0 Punkte: Es können zwar für Inputs und Outputs von Methoden über eine „modelReference“ semantische Klassen angegeben werden, nicht aber direkt für Ressourcen.

Kann die Bedeutung einer Operation definiert werden?

2 Punkte: Ja, hierfür soll der SAWSDL Namespace verwendet werden, damit können über die RDF Properties „modelReference“, „liftingSchemaMapping“, und „loweringSchemaMapping“ für Methoden und deren Inputs und Outputs semantische Modellklassen angegeben werden.

Kann die Bedeutung einer Beziehung zwischen Ressourcen definiert werden?

0 Punkte: Es ist in MicroWSMO nicht möglich, Beziehungen zwischen Ressourcen semantisch zu beschreiben.

4.4.3.3 Lesbarkeit

Wie gering ist der Anteil an Syntax-Overhead im Code?

3 Punkte: Ausgehend von bestehendem HTML-Code sind die zusätzlichen Annotationen sehr knapp, der Overhead ist minimal.

Wie intuitiv ist den Code-Konstrukten ihre Bedeutung anzusehen?

1 Punkt: Wie bei „hRESTS“ finden wir die zwischen HTML Code eingebetteten Beschreibungen schwer lesbar (siehe Beispiel oben).

4.4.3.4 Anwendung der Sprache

In welchem Umfang wird die Sprache bereits eingesetzt?

0 Punkte: Die Sprache wird bisher nicht eingesetzt.

Wie gut ist die Sprache dokumentiert?

1 Punkt: MicroWSMO wird bisher nur in wissenschaftlichen Arbeiten beschrieben. Wir finden, dass das Verständnis hier durch die Vielzahl an verwendeten Sprachen, Ontologien, und Notationen erschwert wird.

Wie gut sind die Tools dokumentiert, die für diese Sprache angeboten werden?

0 Punkte: Die Arbeit verweist auf ein XSLT Style-Sheet, welches aus hREST Beschreibungen in XHTML RDF erzeugen kann. Das Sheet ist laut Paper „selbsterklärend“, es gibt also keinerlei Dokumentation.

Einfachheit: Wie gering ist der Aufwand zur Einarbeitung in die Sprache?

1 Punkt: Die Verwendung von Ontologien und verschiedenen Service-Modellen steht dem raschen Erlenen deutlich im Weg. Außerdem setzt die Verwendung anderer Sprachen wie hREST oder SAWSDL Kenntnis über diese voraus oder erfordert zusätzliche Einarbeitung.

Wie gut wird der Entwickler beim modellieren unterstützt?

0 Punkte: Es gibt keine Tools, die den Entwickler beim Erstellen von MicroWSMO Beschreibungen unterstützen.

Gibt es eine hilfreiche, graphische Darstellung der Modelle?

0 Punkte: Es gibt keine Möglichkeiten, MicroWSMO Beschreibungen graphisch darzustellen.

Unterstützung für: Schnittstellen sollen in einem maschinenlesbaren Format beschrieben werden.

3 Punkte: Über eine XSL Transformation können MicroWSMO Beschreibungen aus HTML nach RDF konvertiert werden.

Unterstützung für: Aus der entworfenen Schnittstelle sollen Code-Stubs generiert werden können.

0 Punkte: Es gibt bisher keine Tools, die aus MicroWSMO Beschreibungen Code-Stubs generieren können.

4.4.4 Fazit

Durch die Erweiterung des Servicemodells von „hRESTS“ gewinnt MicroWSMO stark an Bedeutung für das Beschreiben von Schnittstellen von RESTful Webservices. Es ist nun möglich, die wichtigsten Elemente einer Schnittstelle zu beschreiben, nämlich Ressourcen, Operationen, Inputs, und Outputs. Auch geht MicroWSMO mit der Möglichkeit, Modellreferenzen zu definieren, in die richtige Richtung für „service discovery“.

Die Einbettung in HTML Code finden wir etwas unglücklich, weil sie die MicroWSMO Beschreibungen sehr schwer lesbar macht.

4.5 SEREDASj

4.5.1 Beschreibung

SEREDASj [3] ist eine semantische Beschreibungssprache für RESTful Services. Die Abkürzung selbst steht für SEmantic REstful DAta Services, wobei das angehängte „j“ die Verwendung von JSON verdeutlichen soll. Bei der Entwicklung wurde insbesondere darauf geachtet, SEREDASj als Beschreibungssprache so einfach wie möglich zu halten.

Eine SEREDASj-Beschreibung besteht aus Metadaten und einer Beschreibung der Struktur der definierten Repräsentation. Diese Unterteilung ist im Codebeispiel gut zu erkennen.

Im Gegensatz zu anderen Ansätzen werden hier die architektonischen Eigenschaften von REST berücksichtigt. Es werden also anstatt Operationen oder In- bzw. Output die eigentlichen Repräsentationen der Ressourcen definiert, um RESTful Services angemessen beschreiben zu können.

Neben der Dokumentation des Services bzw. der verwendeten Datenformate können Elemente einer JSON-Repräsentation durch semantische Annotationen beispielsweise so beschrieben werden, dass einem Client klar ist, welches Element eine URI darstellt und darüber hinaus, was dessen Bedeutung ist.

Die automatische Erstellung einer Dokumentation aus einer SEREDASj-Beschreibung ist prinzipiell möglich.

4.5.2 Beispiel

Nachfolgend werden eine JSON Repräsentation und die zugehörige SEREDASj-Beschreibung aufgeführt. Alle Codebeispiele sind der Ausarbeitung [3] entnommen.

```
"id": 556410,
"first_name": "Markus",
"last_name": "Lanthaler",
"gender": "male",
"knows": [
      { "id": 586807, "name": "Christian Gütl" },
      { "id": 790980, "name": "John Doe" } ]
```

Listing 9: SEREDASj: JSON Repräsentation

Das Beispiel setzt ein fiktives soziales Netzwerk voraus und stellt die Repräsentation einer Person und ihrer Freunde dar.

Das folgende Codebeispiel zeigt die beiden wesentlichen Teile einer SEREDASj-Beschreibung. Die erforderlichen Metadaten bilden dabei den ersten Block. Hier werden externe Quellen angegeben und Links sowie deren Bedeutung definiert. Es wird deutlich, dass ein Client erkennen kann, dass es sich bei einer ID um einen Link zu einer anderen Person handelt.

```
"meta": {
      "prefixes": {
            "foaf": "http://xmlns.com/foaf/0.1/",
            "ex": " http://example.com/onto#",
            "owl": "http://www.w3.org/2002/07/owl#",
            "iana": "http://www.iana.org/link-relations/"},
      "links": {
            "/user/{id}": {
                  "mediaType": "application/json",
                  "seredasjDescription": "#",
                  "semantics": {
                        "owl:sameAs": "<#properties/knows>"},
                  "variables": {
                        "id": {
                              "binding": "#properties/knows/id",
                              "model": "[ex:id]"}},
                  "requestDescription": "#"
            },}}, […]
```

Listing 10: Metadaten einer SEREDASj-Beschreibung

Der zweite Teil definiert die Struktur obiger JSON-Repräsentation und gibt auch hier die Bedeutung der einzelnen Elemente an.

```
"type": "object",
"model": "[foaf:Person]",
"properties": {
      "id": {
            "type": "number", "model": "[ex:id]" },
      "first_name": {
            "type": "string","model": "[foaf:firstName]" },
      "last_name": {
            "type": "string", "model": "[foaf:surname]" },
      "gender": {
            "type": "string", "model": "[foaf:gender]" },
      "knows": {
            "type": "array",
            "model": "[foaf:knows]",
            "items": {
                  "type": "object", "model": "[foaf:Person]",
                  "properties": {
                        "id": {
                              "type": "number", "model": "[ex:id]" },
                        "name": {
                              "type": "string", "model": "[foaf:name]"
                        }}}}}
```

Listing 11: Elementdefinition einer SEREDASj-Beschreibung

4.5.3 Bewertung

4.5.3.1 Syntax

Können die Ressourcen der Schnittstelle definiert werden?

3 Punkte: Ja, es kann ein JSON Schema angegeben werden.

Können pro Ressource erlaubte Operationen definiert werden?

2 Punkte: SEREDASj verwendet ganz bewusst eine Sichtweise auf Repräsentationen, nicht auf angebotene Operationen. Erlaubte Operationen müssen durch eigene SEREDASj-Beschreibungen definiert und eingebunden werden (z.B. um Request Bodies zu beschreiben) und können nicht direkt inline beschrieben werden.

Können erwarteter Input bzw. Output des Servers definiert werden?

3 Punkte: Ja, für die Beschreibung von Inputs und Outputs kann eine SEREDASj Beschreibung hinterlegt werden. Das Modell sieht vor, zu einem Link wiederum eine SEREDASj Beschreibung anzugeben, die Request-Bodies beschreibt. Als einzige Quelle

ist das Paper selbst zu finden, in welchem hierfür allerdings kein Beispiel angegeben wird.

Können Beziehungen zwischen Ressourcen definiert werden?

3 Punkte: Das ist eine der Stärken von SEREDASj. Beziehungen zwischen Ressourcen werden über Links in der SEREDASj Beschreibung einer Repräsentation definiert.

Kann für Methoden beschrieben werden, welche Statuscodes erzeugt werden können?

0 Punkte: SEREDASj ist zur Beschreibung der Repräsentationen von Ressourcen konzipiert, nicht für die zugehöriger Methoden.

Bietet die Sprache vordefinierte Ressourcentypen an?

0 Punkte: Es sind keine Ressourcentypen in SEREDASj definiert.

Können Ressourcen geschachtelt werden, sodass URIs relativ zur Eltern-Ressource definiert werden können?

0 Punkte: Die Schachtelung von Ressourcen ist in SEREDASj so nicht möglich. Beziehungen zwischen Ressourcen werden über Links ausgedrückt. Die URIs bleiben davon sinnvoller Weise unberührt.

Können in URI-Definitionen Templates verwendet werden?

3 Punkte: URI Angaben können Platzhalter enthalten, die einem Feld in der Definition der Ressourcenrepräsentation zugeordnet sind.

4.5.3.2 Semantik

Kann die Bedeutung einer Ressource definiert werden?

3 Punkte: Die Bedeutung einer Ressource ist Teil der SEREDASj-Beschreibung und kann als Verweis auf eine Ontologie angegeben werden.

Kann die Bedeutung einer Operation definiert werden?

0 Punkte: Mit SEREDASj werden vor allem Ressourcen bzw. deren Repräsentationen beschrieben. Eine direkte Beschreibung einer Operation und deren Bedeutung ist somit nicht vorgesehen.

Kann die Bedeutung einer Beziehung zwischen Ressourcen definiert werden?

3 Punkte: Gerade hierauf wurde Wert gelegt. Die Beschreibung der Beziehung zu anderen Ressourcen gestaltet sich genauso einfach wie die semantische Annotation dieser.

4.5.3.3 Lesbarkeit

Wie gering ist der Anteil an Syntax-Overhead im Code?

3 Punkte: Bedingt durch die Darstellung in JSON und die Wahl von kurzen, prägnanten Schlüsselwörtern ist der Overhead minimal.

Wie intuitiv ist den Code-Konstrukten ihre Bedeutung anzusehen?

3 Punkte: Bedingt durch JSON ist die Darstellung übersichtlich und klar strukturiert. Außerdem lässt sich die Struktur einer SEREDASj-Beschreibung durch das definierte Modell sehr gut erfassen.

4.5.3.4 Anwendung der Sprache

In welchem Umfang wird die Sprache bereits eingesetzt?

0 Punkte: SEREDASj wird bisher nicht eingesetzt und existiert nur auf theoretischer Basis.

Wie gut ist die Sprache dokumentiert?

1 Punkt: An Dokumentation ist außer dem Paper selbst nur eine Präsentation[25] des Autors zu finden, die im Wesentlichen den gleichen Inhalt enthält. Die Spezifikation im Paper macht einen vollständigen Eindruck, ist dabei allerdings sehr kompakt und enthält nur wenig Beispielmaterial.

Wie gut sind die Tools dokumentiert, die für diese Sprache angeboten werden?

0 Punkte: Auf Grund der fehlenden Verbreitung gibt es für SEREDASj noch keine Tools.

Einfachheit: Wie gering ist der Aufwand zur Einarbeitung in die Sprache?

3 Punkte: Eine Anforderung bei der Konzeption von SEREDASj war, dass die Einarbeitungszeit auf ein Minimum reduziert wird, damit die Verwendung nicht auf Grund dieser Barriere ausgeschlossen wird. Das wird unter anderem durch die Verwendung von JSON konsequent verfolgt und ist unserer Meinung nach gelungen.

Wie gut wird der Entwickler beim Modellieren bestehender Schnittstellen unterstützt?

0 Punkte: Es gibt weder Modelle noch eine Unterstützung für deren Erstellung. Die Beschreibung der Ressourcen erfolgt manuell.

[25] (09.10.2013): http://de.slideshare.net/lanthaler/a-semantic-description-language-for-restful-data-services-to-combat-semaphobia-8064613

Gibt es eine hilfreiche, graphische Darstellung der Modelle?

0 Punkte: Es gibt keine graphische Darstellung für SEREDASj.

Unterstützung für: Schnittstellen sollen in einem maschinenlesbaren Format beschrieben werden.

1 Punkt: Im Paper beschreibt der Autor, wie SEREDASj-Beschreibungen in RDF konvertiert werden können. Die Konvertierung ist simpel und wenig fehleranfällig. Allerdings ist dieses Vorgehen bisher nicht umgesetzt bzw. die Umsetzung nicht frei zugänglich.

Unterstützung für: Aus der entworfenen Schnittstelle sollen Code-Stubs generiert werden können.

0 Punkte: Das Erstellen von Code-Stubs ist nicht Teil von SEREDASj.

4.5.4 Fazit

SEREDASj begründet eine spezielle, ressourcenorientierte Sichtweise und hält diese konsequent ein. Dabei wurde viel Wert auf Einfachheit und Verständlichkeit gelegt. Repräsentationen von Ressourcen können umfassend beschrieben und deren Bedeutung angegeben werden. Einzig der mangelnde Erfolg in Bezug auf die Verwendung trüben den sehr positiven Gesamteindruck.

4.6 Swagger

4.6.1 Beschreibung

Swagger [20] ist Spezifikation und Framework zugleich und wurde für die Dokumentation von APIs entwickelt. Der grundlegende Gedanke ist dabei, dass die Dokumentation an die API eines RESTful Service gebunden und dadurch stets aktuell ist. Im Vordergrund steht außerdem die Interaktion mit der API durch eine die Schnittstelle veranschaulichende Benutzeroberfläche.

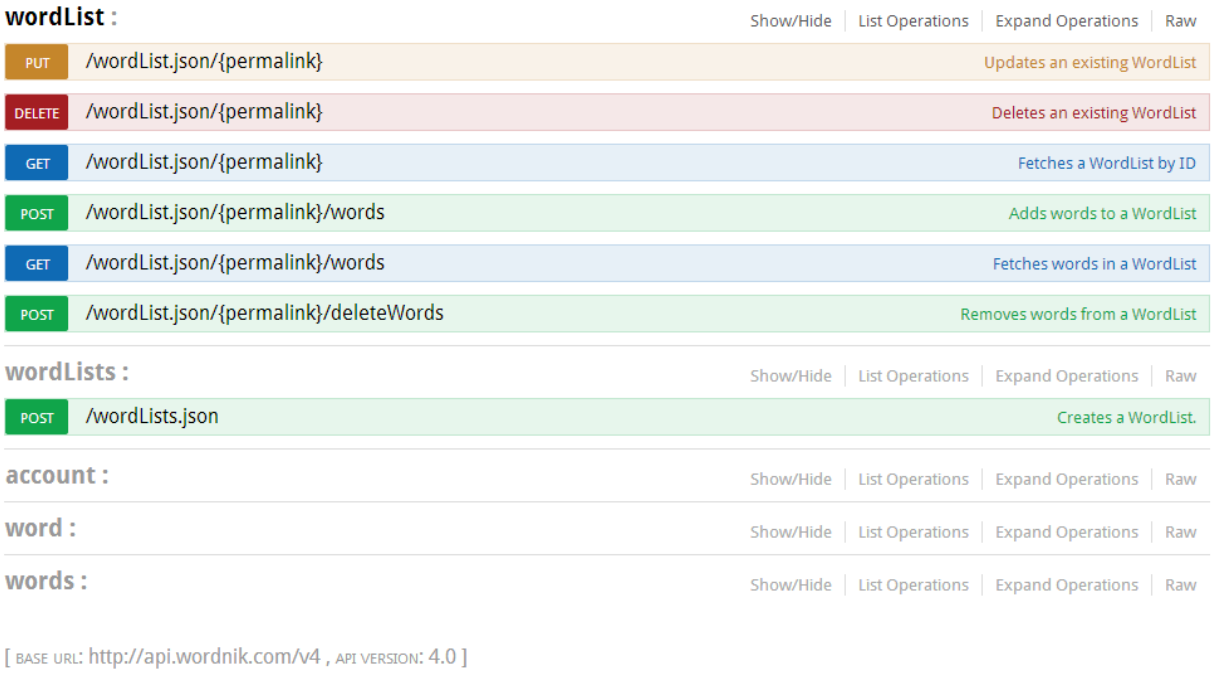


Abbildung 10: Übersicht einer API mit Swagger [21]

Obiges Beispiel ist auf http://petstore.swagger.wordnik.com/ zu finden. Die zur Verfügung stehenden HTTP-Operationen können hier durch die farbliche Hervorhebung in der Benutzeroberfläche gut unterschieden werden, bieten eine ausreichende Kurzbeschreibung und können „ausgeklappt“ werden, um Details anzuzeigen. Auch die Ressourcen lassen sich dieser Auflistung entnehmen. Die dokumentierte API lässt sich damit insgesamt sehr gut erfassen.

Auffallend ist, dass mit dem Ansatz von Swagger eigentlich Operationen beschrieben werden, anstatt Repräsentationen von Ressourcen, welche wesentlich für RESTful Services sind.

4.6.2 Beispiel

Die Benutzeroberfläche erlaubt es dem Anwender, sich die Operationen mit Hilfe von Interaktion genauer anzuschauen. Dabei sind In- bzw. Output klar zu erkennen und die Kurzbeschreibung einzelner Parameter zum besseren Verständnis ebenfalls aufgeführt.

Außerdem lassen sich der Beschreibung mögliche Error Status Codes und deren Begründung entnehmen.

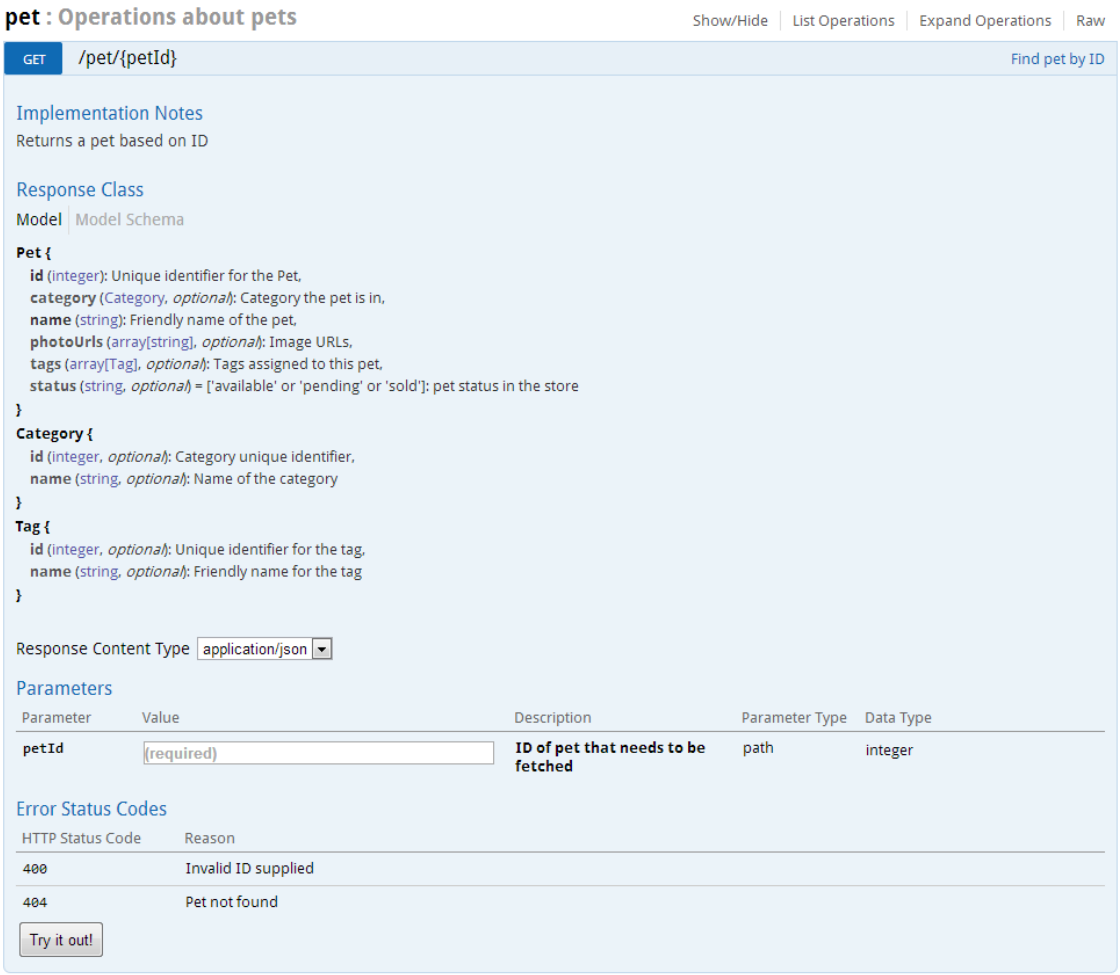


Abbildung 11: Beispiel für die Darstellung einer HTTP-Methode mit Swagger-UI [21]

Der für solche Beschreibungen erforderliche Code ist in JSON gehalten und gestaltet sich dadurch schlicht und gut lesbar. Eine Beschreibung für eine ähnliche Operation würde wie hier abgebildet aussehen.

```
"apis":[
  {
    "path":"/pet/{petId}",
    "description":"Operations about pets",
    "operations":[
      {
        "method":"GET",
        "nickname":"getPetById",
        "type":"Pet",
        "parameters":[ ... ],
        "summary":"Find pet by its unique ID",
        "notes": "Only Pets which you have permission to see will be returned",
        "responseMessages":[ ... ]
      }
    ]
  }
]
```

Abbildung 12: Beispielcode für Swagger [20]

Der Output wird dabei durch das Feld „type“ beschrieben, während „parameters“ den erforderlichen Input definiert. Die Felder „nickname“, „summary“ und „notes“ dienen dabei als Beschreibungen mit unterschiedlichem Umfang.

4.6.3 Bewertung – Swagger

4.6.3.1 Syntax

Können die Ressourcen der Schnittstelle definiert werden?

2 Punkte: Swagger stellt primär angebotene Operationen dar, Ressourcen sind im Code als „apis“ aufgeführt. Dafür können Beschreibungen hinterlegt, und Pfade definiert werden.

Können pro Ressource erlaubte Operationen definiert werden?

3 Punkte: Die Operationen eines RESTful Service können vollständig definiert werden. Das Feld „method“ erlaubt dabei die Werte GET, POST, PUT und DELETE, entsprechend der benötigten HTTP-Methode. Für Operationen können außerdem aussagekräftige Bezeichner gewählt und Kurz- sowie ausführliche Beschreibungen verfasst werden.

Können erwarteter Input bzw. Output des Servers definiert werden?

2 Punkte: In- bzw. Output einer Operation können angegeben werden. Während man für den Input erforderliche Parameter definiert, wird für den Output direkt der Datentyp angegeben. Swagger erlaubt es dafür, JSON-Schemadefinitionen zentral zu hinterlegen, die dann im gleichen Swagger-Dokument als In- bzw. Outputs referenziert werden können. Allerdings ist es nicht möglich, externe Schemadefinitionen über URLs zu verlinken.

Können Beziehungen zwischen Ressourcen definiert werden?

0 Punkte: Nein, eine solche Möglichkeit ist nicht vorgesehen.

Kann für Methoden beschrieben werden, welche Statuscodes erzeugt werden können?

3 Punkte: Ja, außerdem kann zu jedem Statuscode eine zusätzliche Beschreibungen und ggf. ein Modell (z.B. bei ungültigem Eingabeformat) angegeben werden.

Bietet die Sprache vordefinierte Ressourcentypen an?

0 Punkte: Es werden keine vordefinierten Ressourcentypen angeboten.

Können Ressourcen geschachtelt werden, sodass URIs relativ zur Eltern-Ressource definiert werden können?

0 Punkte: Diese Möglichkeit ist in Swagger nicht enthalten.

Können in URI-Definitionen Templates verwendet werden?

3 Punkte: Ja, dafür können Beschreibungen angegeben werden. Außerdem können Datentypen definiert, für numerische Datentypen Minimum und Maximum angeben, und über Enums mögliche Werte definiert werden, für Post-Methoden können komplexe Datentypen definiert werden.

4.6.3.2 Semantik

Kann die Bedeutung einer Ressource definiert werden?

0 Punkte: Semantische Annotationen werden von Swagger nicht unterstützt.

Kann die Bedeutung einer Operation definiert werden?

0 Punkte: Semantische Annotationen werden von Swagger nicht unterstützt.

Kann die Bedeutung einer Beziehung zwischen Ressourcen definiert werden

0 Punkte: Semantische Annotationen werden von Swagger nicht unterstützt.

4.6.3.3 Lesbarkeit

Wie gering ist der Anteil an Syntax-Overhead im Code?

3 Punkte: Weil Swagger JSON verwendet, ist der benötigte Code sehr schlicht. Die vorgesehenen Elemente sind dabei sinnvoll und stichhaltig benannt.

Wie intuitiv ist den Code-Konstrukten ihre Bedeutung anzusehen?

3 Punkte: Der Code ist auf Grund der Verwendung von JSON und der kompakten Konstrukte sehr verständlich. Die Gliederung dieser ist sinnvoll und kann schnell erfasst werden.

4.6.3.4 Anwendung der Sprache

In welchem Umfang wird die Sprache bereits eingesetzt?

1 Punkt: Swagger taucht immer wieder in Diskussion diverser Foren auf, und wird dabei überwiegend sehr positiv bewertet. Es sind allerdings keine Firmen angegeben oder bekannt, die Swagger einsetzen.

Wie gut ist die Sprache dokumentiert

3 Punkte: Swagger präsentiert sich mit einer übersichtlichen Dokumentation, und profitiert dabei von zur Verfügung stehenden Live-Demos, für die sogar die zugrunde liegenden Swagger-Beschreibungen ausgegeben werden können.

Wie gut sind die Tools dokumentiert, die für diese Sprache angeboten werden? – 3 Punkte

3 Punkte: Es existieren mehrere Tools, zu denen jeweils ausführliche Anleitungen für die Installation und deren Benutzung vorhanden sind.

Einfachheit: Wie gering ist der Aufwand zur Einarbeitung in die Sprache?

3 Punkte: Die anschaulichen Beispiele von Swagger-Beschreibungen zu verschiedenen APIs vermitteln durch die interaktive Benutzeroberfläche schnell ein gutes Gefühl für die Sprache, das zusammen mit der übersichtlichen Dokumentation und der Einfachheit der Sprache sehr gut zum Verständnis beiträgt.

Wie gut wird der Entwickler beim Modellieren bestehender Schnittstellen unterstützt?

1 Punkt: Das Modellieren erfolgt manuell oder, falls vorhanden, durch die automatische Interpretation der Swagger-Java-Annotationen.

Gibt es eine hilfreiche, graphische Darstellung der Modelle?

2 Punkte: Die interaktive Benutzeroberfläche (Swagger-UI) stellt auf ihre Weise ein Modell der beschriebenen Schnittstelle dar, die sich als hilfreich erweist und durch ihre gelungene Gestaltung überzeugt.

Unterstützung für: Schnittstellen sollen in einem maschinenlesbaren Format beschrieben werden.

3 Punkte: Die Entwickler definieren mit Swagger Core Java-Annotationen und die erforderliche Logik zum Generieren von Swagger-Beschreibungen an.

Unterstützung für: Aus der entworfenen Schnittstelle sollen Code-Stubs generiert werden können.

3 Punkte: Die Entwickler bieten mit Swagger Codegen die Möglichkeit Code in verschiedenen Sprachen basierend auf der entworfenen Swagger Resource Declaration zu generieren.

4.6.4 Fazit

Swagger hinterlässt einen durchweg positiven Eindruck und besticht durch die angebotene Benutzeroberfläche. Hierbei sind auch die Beschreibungsmöglichkeiten für Schnittstellen sinnvoll und auf die Anwendung in der Praxis ausgerichtet. Die kompakte Dokumentation ermöglicht in Ergänzung mit den vorhandenen Beispielen einen raschen Einstieg, der mit vielseitigen Tools belohnt wird. Wer auf semantische Annotationen angewiesen ist, sollte sich einer anderen Lösung bedienen. Auch das Entwerfen von Schnittstellen wird durch Swagger nur bedingt gefördert.

4.7 RestDoc

4.7.1 Beschreibung

RestDoc [22] bezeichnet sich selbst auf der eigenen Homepage als „language-agnostic specification for self-describing REST API's“ [23], ist also eine Beschreibungssprache für „RESTful“ Schnittstellen. RestDoc basiert auf JSON, die Spezifikation definiert also über ein JSON Schema ein Vokabular, welches sowohl Ressourcen-, als auch methodenorientiert ist.

Für den Zugriff auf RestDoc Beschreibungen sieht die Spezifikation vor, dass der Server für den Aufruf der HTTP „Options“ Methode auf eine URI eine RestDoc Beschreibung liefert, in der alle Ressourcen unter dieser URI beschrieben sind.

Es existieren Open-Source Tools für die Erstellung und Verwendung von RestDoc, sowohl serverseitig als auch clientseitig, die aber bisher eher den Status eines Prototyps haben.

4.7.2 Beispiel

Im Beispiel aus der RestDoc Spezifikation [22] ist zum einen zu sehen, dass URI Definitionen über Platzhalter angegeben werden können, zum anderen können reguläre Ausdrücke angegeben werden, um für Parameter erlaubte Werte zu definieren.

```
"resources": [ {
  "id": "LocalizedMessage",
  "description": "A localized message",
  "path": "/{locale}/{messageId}{?seasonal}",
  "params": { // URI parameters descriptions
    "locale": {
      "description": "A standard locale string, e.g. \"en_US.utf-8\"",
      "validations": [ { "type": "match", "pattern": "[a-z]+(_[A-
Z]+)[…]"}] },
    "messageId": {[…]},
    "seasonal": {[…}}
  },
  "methods": {
    "PUT": {
      "description": "Update or create a message",
      "statusCodes": { "201": "Created" },
      "accepts": {[…]} } } ]
```

Listing 12: RestDoc Codebeispiel

4.7.3 Bewertung

4.7.3.1 Syntax

Können die Ressourcen der Schnittstelle definiert werden?

2 Punkte: Restdoc bietet für die Darstellung von Ressourcen das JSON-Element „resource“, das eine ID und eine Beschreibung enthalten kann.

Können pro Ressource erlaubte Operationen definiert werden?

3 Punkte: Methoden werden in RestDoc über das JSON-Element „method“ repräsentiert, das eine Beschreibung enthalten kann.

Können erwarteter Input bzw. Output des Servers definiert werden?

3 Punkte: Für Methoden kann über das Element „accepts“, bzw. „response“ eine Menge von Schemadefinitionen angegeben werden, die die erlaubten Ressourcenrepräsentationen definiert.

Können Beziehungen zwischen Ressourcen definiert werden?

0 Punkte: Es ist in RestDoc nicht möglich, Beziehungen zwischen Ressourcen auszudrücken.

Kann für Methoden beschrieben werden, welche Statuscodes erzeugt werden können?

3 Punkte: Über das Element „statusCodes“ kann die Menge von möglichen Statuscodes definiert werden.

Bietet die Sprache vordefinierte Ressourcentypen an?

0 Punkte: RestDoc bietet darüber hinaus keine vordefinierten Typen.

Können Ressourcen geschachtelt werden, sodass URIs relativ zur Eltern-Ressource definiert werden können?

0 Punkte: RestDoc bietet keine Möglichkeit, Ressourcen hierarchisch in Beziehung zu setzen.

Können in URI-Definitionen Templates verwendet werden?

3 Punkte: Teile von URIs können über Platzhalter beschrieben werden, die von RestDoc als „Parameter“ bezeichnet werden. Für Parameter können Beschreibungen angegeben werden, und sogar erlaubte Werte über einen regulären Ausdruck definiert werden.

4.7.3.2 Semantik

Kann die Bedeutung einer Ressource definiert werden?

0 Punkte: Es ist nicht vorgesehen, mit RestDoc Semantik auszudrücken.

Kann die Bedeutung einer Operation definiert werden?

0 Punkte: Es ist nicht vorgesehen, mit RestDoc Semantik auszudrücken.

Kann die Bedeutung einer Beziehung zwischen Ressourcen definiert werden?

0 Punkte: Es ist nicht vorgesehen, mit RestDoc Semantik auszudrücken.

4.7.3.3 Lesbarkeit

Wie gering ist der Anteil an Syntax-Overhead im Code?

3 Punkte: RestDoc Beschreibungen sind in JSON geschrieben. Hier fällt kaum Overhead an. Die von RestDoc definierten Elemente sind knapp benannt, meistens nur mit einem Wort (siehe Beispiel oben).

Wie intuitiv ist den Code-Konstrukten ihre Bedeutung anzusehen?

3 Punkte: RestDoc verwendet JSON als Darstellung. Als Grundlage gibt es deswegen nur sehr wenige Konstrukte, die sehr einfach verständlich sind. Die Schlüsselwörter sind sehr kurz gehalten, aber von der Benennung her sinnvoll und leicht verständlich.

4.7.3.4 Anwendung der Sprache

In welchem Umfang wird die Sprache bereits eingesetzt?

1 Punkt: Die RestDoc Homepage nennt die „Tullius Walden Bank“ als einzigen Nutzer von RestDoc.[26]

Wie gut ist die Sprache dokumentiert?

2 Punkte: Als Dokumentation ist nur die Spezifikation auf der Homepage zu finden. Diese ist knapp, aber exakt und vollständig - außerdem mit einem kleinem Beispiel versehen. Ansonsten ist keine Dokumentation zu finden.

Wie gut sind die Tools dokumentiert, die für diese Sprache angeboten werden?

0 Punkte: Das Projekt bietet auf Github[27] eine serverseitige Implementierung, für die Dokumentation existiert. Für deren Benutzung muss der Quellcode manuell kompiliert werden.

[26] (09.10.2013) http://www.restdoc.org/refs.html
[27] (09.10.2013) https://github.com/hoegertn/restdoc-java-server

Für die clientseitige Implementierung in Python gibt es nur eine kurze Installationsanleitung, die auf Windows Systemen nicht funktioniert.[28]

Einfachheit: Wie gering ist der Aufwand zur Einarbeitung in die Sprache?

2 Punkte: Die sehr kompakte Dokumentation lässt immer wieder Fragen aufkommen, deren Antwort sich teilweise nur mühsam erarbeiten lässt, auch wenn RestDoc selbst durch seine Einfachheit punktet.

Wie gut wird der Entwickler beim Modellieren bestehender Schnittstellen unterstützt?

1 Punkt: Die serverseitige Implementierung definiert JavaDoc-Annotationen, aus denen die serverseitige Implementierung JSON RestDoc Beschreibungen erstellen kann. Für diese Annotationen ist keine Dokumentation hinterlegt außer einem kleinen Beispielprojekt als Teil des Codes. Ansonsten wird keine Unterstützung angeboten.

Beispiel für JavaDoc-Annotationen aus dem Sourcecode der serverseitigen Implementierung:

```
@POST
@RestDocIgnore
@RestDocReturnCodes(
      {@RestDocReturnCode(code = "200", description = "All went well"),
      @RestDocReturnCode(code = "403", description = "Access not allowed")})
```

Gibt es eine hilfreiche, graphische Darstellung der Modelle?

1 Punkt: RestDoc definiert, dass die Schnittstellenbeschreibungen beim Options-Aufruf auf der Ressource zurückgegeben werden. Die clientseitige Implementierung in Python kann diese in Textform auf der Konsole ausgeben.

[28] (09.10.2013) https://github.com/RestDoc/restdoc.py

```
(localhost:5000) reload
(localhost:5000) resources
+-----+-------+-----------------+-------------+
|  id |  path |     methods     | description |
+-----+-------+-----------------+-------------+
| App | /:app | ['POST', 'GET'] |     None    |
+-----+-------+-----------------+-------------+
(localhost:5000) doc App
{'id': 'App',
 'methods': {'GET': {'description': 'Gets the app'},
             'POST': {'description': 'Updates the app'}},
 'params': {'app': {'description': 'the app entry id', 'required': True}},
 'path': '/:app'}
(localhost:5000) get App -t app=foobar
200 OK
transfer-encoding: chunked
connection: keep-alive
x-powered-by: Express
```

Abbildung 13: Konsolenausgabe des RestDoc Python Clients

Unterstützung für: Schnittstellen sollen in einem maschinenlesbaren Format beschrieben werden.

3 Punkte: Sind die Schnittstellen über JavaDoc-Annotationen beschrieben, kann die serverseitige Implementierung daraus JSON RestDoc Beschreibungen erstellen. Allerdings können diese bisher nur auf der Konsole ausgegeben werden:

```
[main] INFO org.restdoc.server.impl.RestDocGenerator - Starting generation of RestDoc
[main] INFO org.restdoc.server.impl.RestDocGenerator - Searching for RestDoc API classes
[main] INFO org.restdoc.server.impl.RestDocGenerator - Scanning class: org.restdoc.server.impl.MyRSBean
[main] INFO org.restdoc.server.impl.RestDocGenerator - Class org.restdoc.server.impl.MyCrudBean provides prede
[main] INFO org.restdoc.server.impl.RestDocGenerator - Scanning class: org.restdoc.server.impl.MyResourceBean
[main] INFO org.restdoc.server.impl.RestDocGenerator - Scanning class: org.restdoc.server.impl.MyDeepRes
[main] INFO org.restdoc.server.impl.RestDocGenerator - Ignoring method: public java.lang.String org.restdoc.se
{"schemas":{"http://some.json/msg":{"type":"inline","schema":{"type":"object","properties":{"content":{"type":
{"schemas":{"http://some.json/msg":{"type":"inline","schema":{"type":"object","properties":{"content":{"type":
```

Abbildung 14: RestDoc in der Konsolenausgabe der serverseitigen Implementierung, Ausgabe abgeschnitten

Unterstützung für: Aus der entworfenen Schnittstelle sollen Code-Stubs generiert werden können.

0 Punkte: Die Generierung von Code-Stubs wird nicht angeboten.

4.7.4 Fazit

Die Sprache RestDoc wirkt gut durchdacht und setzt mit JSON auf einen bewährten Standard. Es können sowohl Ressourcen als auch Methoden dargestellt werden, was sowohl eine Ressourcen-, als auch eine methodenorientierte Modellierung erlaubt. RestDoc bietet einige gute Ideen, wie für URI-Parameter reguläre Ausdrücke, die erlaubte Werte definieren, angeben zu können. Was aber die Toolunterstützung betrifft, so sind die

auf der Homepage als „server-side implementation“, bzw. „client-side implementation“ bezeichneten Werkzeuge nicht mehr als Prototypen ohne jede graphische Oberfläche.

Die Installation der angebotenen Tools ist aufwändig und es gibt kaum Anleitungen. Tatsächlich muss für die Verwendung der serverseitigen Implementierung der Sourcecode heruntergeladen und kompiliert werden.

5 Ergebnisse

Das nachfolgende Diagramm zeigt den erreichten, prozentualen Gesamtwert jedes Ansatzes abhängig von der maximal erreichten Punktzahl in der jeweiligen Kategorie. Die Kategorie „Auswertung Gesamt“ fasst dabei die Ergebnisse der restlichen Kategorien zusammen.

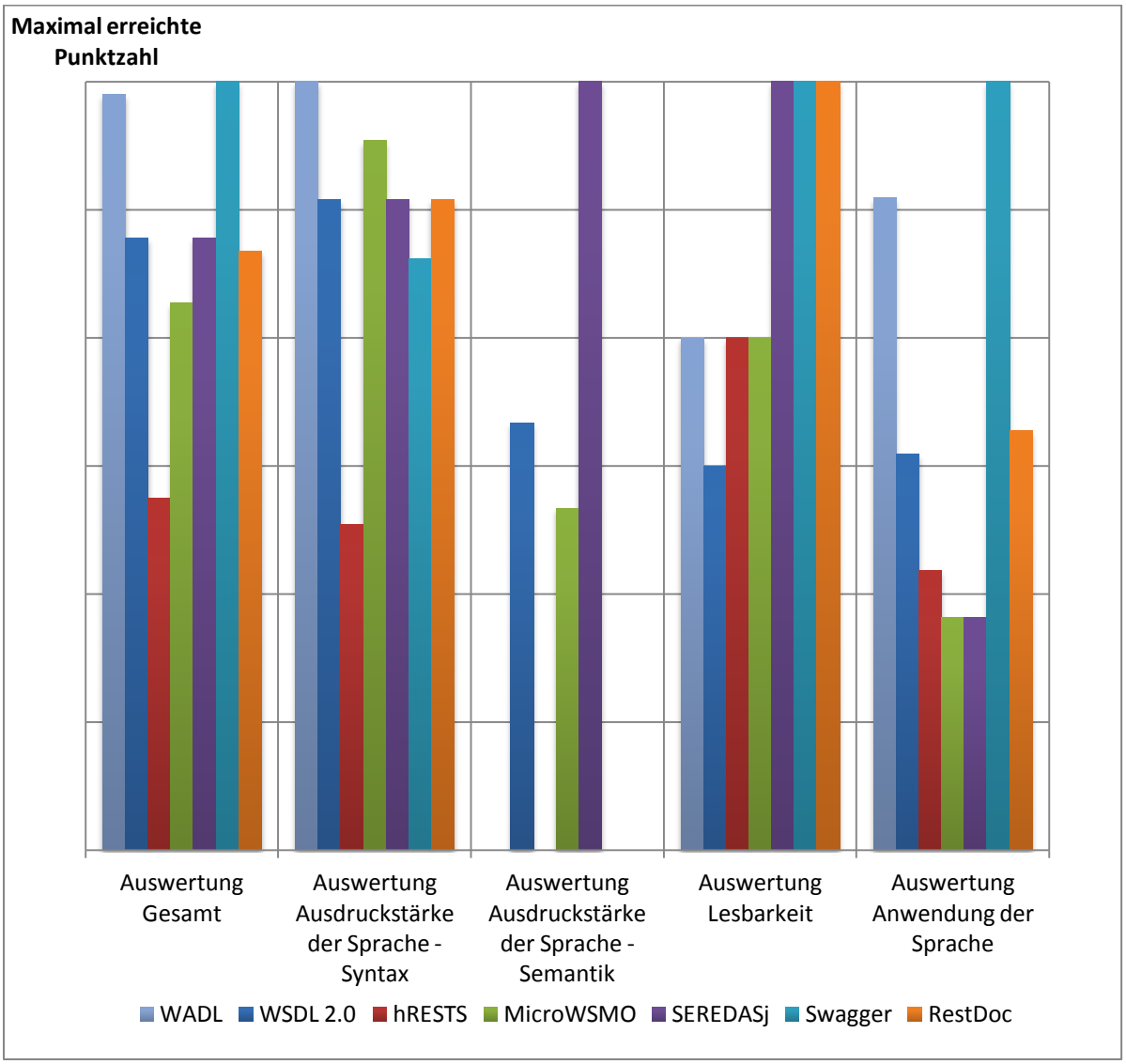


Abbildung 15: Vergleich der Ergebnisse unserer Auswertungen

Swagger und WADL haben dabei die höchste Gesamtpunktzahl erreicht. Das schließt die anfangs beschriebene Gewichtung der Fragen mit ein. WSDL 2.0, MicroWSMO, SEREDASj und RestDoc bilden das Mittelfeld: Hier würde sich eine individuelle Gewichtung der Fragen am deutlichsten zeigen.

Wir wollen an dieser Stelle auf die Vielzahl an Möglichkeiten hinweisen, die beschriebenen Modellierungssprachen zu erweitern oder zu kombinieren. Unter Zuhilfenahme von SAWSDL können beispielsweise WSDL Beschreibungen um semantische Annotationen erweitert werden. Analog gibt es Ansätze, WADL

Beschreibungen zu erweitern. Zudem ist es denkbar, zusätzlich zu einer rein syntaktischen Beschreibung eine Sprache zu verwenden, die dafür erstellt wurde, semantische Zusammenhänge zu formulieren. Wir konnten bei der Bewertung nicht alle dieser Möglichkeiten berücksichtigen, die Ergebnisse sollten also nach Anwendungsbereich differenziert betrachtet werden.

Es lässt sich beispielsweise erkennen, dass SEREDASj trotz guter Werte kaum Bedeutung in der Anwendung hat. Dieses Problem trifft wahrscheinlich viele gute Ansätze, deren Verwendung entweder durch den Mangel an Popularität und damit verbundenem Mangel an Dokumentation (Forumsdiskussionen usw.) oder an zu hoher Komplexität scheitern.

Bei Betrachtung aller Bewertungen zu den einzelnen Fragen fällt auf, dass beispielsweise nicht versucht wird, vordefinierte Ressourcentypen anzubieten. Außerdem bieten alle Sprachen die Verwendung von Templates in URI Definitionen an.

5.1 Tabellarische Übersicht der Bewertungen

Kriterien	WADL	WSDL 2.0	hRESTS	Mirco-WSMO	SEREDASj	Swagger	RestDoc
Können die Ressourcen der Schnittstelle definiert werden?	2	2	0	2	3	2	2
Können pro Ressource erlaubte Operationen definiert werden?	3	3	3	3	2	3	3
Können erwarteter Input bzw. Output des Servers definiert werden?	3	3	1	3	3	2	3
Können Beziehungen zwischen Ressourcen definiert werden?	1	2	0	1	3	0	0
Kann beschrieben werden, welche Statuscodes erzeugt werden können?	3	1	0	3	0	3	3
Bietet die Sprache vordefinierte Ressourcentypen an?	0	0	0	1	0	0	0
Können Ressourcen geschachtelt werden (relative URIs)?	3	0	0	0	0	0	0
Können in URI Definitionen Templates verwendet werden?	3	3	3	3	3	3	3
Kann die Bedeutung einer Ressource definiert werden?	0	1	0	0	3	0	0
Kann die Bedeutung einer Operation definiert werden?	0	1	0	2	0	0	0
Kann die Bedeutung einer Beziehung definiert werden?	0	1	0	0	3	0	0
Wie gering ist der Anteil an Syntax-Overhead im Code?	1	1	3	3	3	3	3
Wie intuitiv ist den Code-Konstrukten ihre Bedeutung anzusehen?	3	2	1	1	3	3	3
In welchem Umfang wird die Sprache bereits eingesetzt?	2	1	0	0	0	1	1
Wie gut ist die Sprache dokumentiert?	2	2	2	1	1	3	2
Wie gut sind die Tools für diese Sprache dokumentiert?	3	1	0	0	0	3	0
Einfachheit: Wie gering ist der Aufwand zur Einarbeitung in die Sprache?	2	1	3	1	3	3	2
Wird der Entwickler beim Modellieren unterstützt?	1	1	0	0	0	1	1
Gibt es eine hilfreiche, graphische Darstellung der Modelle?	1	0	0	0	0	2	1
Unterstützung für: Schnittstellen in maschinenlesbaren Format.	3	3	1	3	1	3	3
Unterstützung für: Code-Stubs aus Schnittstelle generieren.	3	1	0	0	0	3	0

5.2 Tabellarische Auswertung der Ergebnisse

Sprache	WADL	WSDL 2.0	hRESTS	Micro WSMO	SERE-DASj	Swagger	RestDoc
Gesamt	58	47	27	42	47	59	46
Syntax	26	22	11	24	22	20	22
Semantik	0	5	0	4	9	0	0
Lesbarkeit	4	3	4	4	6	6	6
Anwendung	28	17	12	10	10	33	18

Gesamt	0,98	0,80	0,46	0,71	0,80	1,00	0,78
Syntax	1,00	0,85	0,42	0,92	0,85	0,77	0,85
Semantik	0,00	0,56	0,00	0,44	1,00	0,00	0,00
Lesbarkeit	0,67	0,50	0,67	0,67	1,00	1,00	1,00
Sprache	0,85	0,52	0,36	0,30	0,30	1,00	0,55

Die obere Tabelle zeigt die erreichte Gesamtpunktzahl der bewerteten Spezifikationen. Hierbei wurde die Gewichtung der einzelnen Fragen bereits berücksichtigt.

Die untere Tabelle zeigt die normalisierten Werte. Eine 1,00 in der unteren Tabelle entspricht also der Höchstpunktzahl der gleichen Gruppe in der oberen Tabelle.

6 Zusammenfassung

Diese Fachstudie gibt einen Überblick über vorhandene Methoden, Sprachen und Tools zur Modellierung von REST-Schnittstellen. Die Motivation hierfür gründet in der Notwendigkeit, REST Services beschreiben zu müssen. Maschinenlesbare Beschreibungen von REST APIs existieren bis jetzt nur in überschaubarem Ausmaß. Es gibt zu diesem Thema inzwischen viele wissenschaftliche Arbeiten mit sehr verschiedenen Ansätzen, von ressourcenorientierten zu methodenorientierten und von rein syntaktischen zu semantischen Beschreibungen mit der Vision eines „semantic web“ und automatisierter „service discovery“. Keine dieser Methoden oder Sprachen hat bisher jedoch industriell große Bedeutung erlangt - ein Grund dafür könnte der Umstand sein, dass es immer noch Uneinigkeit darüber gibt, was denn überhaupt modelliert werden sollte.

Wir haben zur Ermittlung verschiedener Lösungen für diese Anforderung einen Kriterienkatalog erstellt und diesen zur Bewertung in Frage kommender Spezifikationen benutzt. Unabhängig davon wurden alternative Ansätze bewertet, die sich aus verschiedenen Gründen nicht durch den Fragenkatalog erfassen ließen.

Die Ergebnisse der bewerteten Spezifikationen wurden graphisch in einer Tabelle aufbereitet und es wurde gezeigt, dass es zum Teil erhebliche Unterschiede in den jeweiligen Stärken und Schwächen der Kandidaten gibt. So lässt sich zu diesem Zeitpunkt kein klarer „Gewinner“ feststellen und es ist fraglich ob sich in naher Zukunft ein Standard etablieren wird.

7 Literaturverzeichnis

[1] R. T. Fielding, Architectural Styles and the Design of Network-Based Software Architectures, University of California, Irvine., 2000.

[2] C. Pautasso, O. Zimmermann und F. Leymann, „Restful web services vs. "big"' web services: making the right architectural decision," *Proceedings of the 17th international conference on World Wide Web (WWW '08),* pp. 805-814, 2008.

[3] M. Lanthaler und C. Gütl, „A Semantic Description Language for RESTful Data Services to Combat Semaphobia," *Proceedings of the 5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011),* pp. 47-53, 2011.

[4] I. Porres und I. Rauf, „Modeling behavioral RESTful web service interfaces in UML," *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11),* pp. 1598-1605, 2011.

[5] M. Laitkorpi, P. Selonen und T. Systa, „Towards a Model-Driven Process for Designing ReSTful Web Services," *ICWS '09 Proceedings of the 2009 IEEE International Conference on Web Services,* pp. 173-180, 2009.

[6] L. Richardson und S. Ruby, RESTful Web Services, O'Reilly Media, 2007.

[7] E. Ormeño, M. Lund, L. Aballay und S. Aciar, „An UML profile for modeling RESTful services," *13th Argentine Symposium on Software Engineering, ASSE 2012,* pp. 119-133, 2012.

[8] S. Schreier, „Modeling RESTful applications," *WS-REST'11 Proceedings of the Second International Workshop on RESTful Design,* pp. 15-21, 2011.

[9] M. Masse, „WRML Design Notes," 2013.

[10] T. Berners-Lee und D. Connolly, „Notation3 (N3): A readable RDF syntax," W3C, [Online]. Available: http://www.w3.org/TeamSubmission/2011/SUBM-n3-20110328/. [Zugriff am 9 Oktober 2013].

[11] R. Verborgh, T. Steiner, D. Deursen, J. Roo, R. Van De Walle, J. Gabarró und Vallés, „Capturing the functionality of Web services with functional descriptions," *Multimedia Tools and Applications Volume 64 Issue 2,* pp. 365-387, May 2013.

[12] M. Hadley, „Web Application Description Language," W3C, [Online]. Available: http://www.w3.org/Submission/2009/SUBM-wadl-20090831/. [Zugriff am 7 Oktober 2013].

[13] R. Chinnici, J.-J. Moreau und S. Weerawarana, „Web Services Description Language (WSDL) Version 2.0 Part1: Core Language," W3C, [Online]. Available: http://www.w3.org/TR/2007/REC-wsdl20-20070626/. [Zugriff am 8 Oktober 2013].

[14] H. Haas, P. Le Hégaret, J.-J. Moreau, D. Orchard, J. Schlimmer und S. Weerawarana, „Web Services Description Language (WSDL) Version 2.0 Part 3: Bindings," W3C, [Online]. Available: http://www.w3.org/TR/2004/WD-wsdl20-bindings-20040803/. [Zugriff am 8 Oktober 2013].

[15] J. Kopecký, T. Vitvar, D. Fensel und K. Gomadam, „hRESTS & MicroWSMO, CMS WG Working Draft," 2009.

[16] J. Kopecký, K. Gomadam und T. Vitvar, „hRESTS: An HTML Microformat for Describing RESTful Web Services," *WI-IAT '08 Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent*

Technology - Volume 01, pp. 619-625, 2008.

[17] D. Fensel, F. Fischer, J. Kopecký, R. Krummenacher, D. Lambert und T. Vitvar, „WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web," W3C, [Online]. Available: http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/. [Zugriff am 8 Oktober 2013].

[18] J. Farrell und H. Lausen, „Semantic Annotations for WSDL and XML Schema," W3C, [Online]. Available: http://www.w3.org/TR/2007/REC-sawsdl-20070828/. [Zugriff am 8 Oktober 2013].

[19] F. Fischer und B. Norton, „D3.4.6 MicroWSMO v2 – Defining the second version of MicroWSMO as a systematic approach for rich tagging," 2009.

[20] Reverb Technologies, Inc, „Swagger: A simple, open standard for describing REST APIs with JSON | Reverb for Developers," [Online]. Available: https://developers.helloreverb.com/swagger/. [Zugriff am 8 Oktober 2013].

[21] Reverb Technologies, Inc, „Swagger Sample App," [Online]. Available: http://petstore.swagger.wordnik.com/#!/. [Zugriff am 8 Oktober 2013].

[22] RestDoc.org, „RestDoc - Documenting REST APIs Version 1 (2012-12-02)," [Online]. Available: http://www.restdoc.org/spec.html. [Zugriff am 6 Oktober 2013].

[23] T. Hoeger, „RestDoc Specification - README.md," [Online]. Available: https://github.com/RestDoc/specification/blob/master/README.md. [Zugriff am 6 Oktober 2013].

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.

Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.

Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.

Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.

Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Bruder, Leonard Harth, Fabian Karaoğuz, Nedim

Stuttgart, 10.10.2013