A Framework for the Structural Analysis of REST APIs

Florian Haupt, Frank Leymann, Anton Scherer, Karolina Vukojevic-Haupt

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{haupt, leymann, vukojevic}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{HauptLSV17,
  author    = {Florian Haupt and Frank Leymann and Anton Scherer and Karolina
               Vukojevic-Haupt},
  title     = {A Framework for the Structural Analysis of REST APIs},
  booktitle = {Proceedings of the 1st IEEE International Conference on Software
               Architecture, ICSA 2017, 3-7 April 2017, Gothenburg, Sweden},
  year      = {2017},
  pages     = {55--58},
  doi       = {10.1109/ICSA.2017.40},
  publisher = {IEEE Computer Society}
}
```

A Framework for the Structural Analysis of REST APIs

Florian Haupt, Frank Leymann, Anton Scherer, Karolina Vukojevic-Haupt
Institute of Architecture of Application Systems
University of Stuttgart
Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

Abstract— **Today, REST APIs have established as a means for realizing distributed systems and are supposed to gain even more importance in the context of Cloud Computing, Internet of Things, and Microservices. Nevertheless, many existing REST APIs are known to be not well-designed, resulting in the absence of desirable quality attributes that truly RESTful systems entail. Although existing analysis show, that many REST APIs are not fully REST compliant, it is still an open issue how to improve this deficit and where to start. In this work, we introduce a framework for the structural analysis of REST APIs based on their description documents, as this allows for a comprehensive, well-structured analysis approach that also includes analyzing the corresponding API description languages. A first validation builds on a set of 286 real world API descriptions available as Swagger documents, and comprises their transformation into a canonical metamodel for REST APIs as well as a metrics-based analysis and discussion of their structural characteristics with respect to compliance with the REST architectural style.**

Keywords- REST, interface description language, analysis

I. Introduction

The architectural style *Representational State Transfer (REST)* has become a popular choice for the realization of service-oriented architectures. Based on the core technologies of the World Wide Web (WWW), mainly the Hypertext Transfer Protocol (HTTP) together with URIs and MIME types, it promises simplicity, standards-based interoperability, and ubiquitous availability on all kind of platforms [1]. Even more important are the implications of the REST style on the quality attributes of a REST-compliant software system. Distributed software systems that follow the REST style are assumed to support inter alia software longevity, independent evolution of its components, scalability, and extensibility [2]. The main challenge in achieving these desirable quality attributes is the REST-compliant design and realization of services.

It has been shown that many APIs that claim to follow the REST style are not REST compliant at all [3][4][5]. A first step towards a REST compliant API is the correct usage of the HTTP protocol, respecting its syntactical as well as semantic specification [6]. However, being REST compliant typically requires more effort than this [7]. One of REST's core constraints is called *Hypertext as the Engine of Application State (HATEOAS)*. It demands that clients of a REST API are guided by the responses they receive from an API. Each response contains metadata like hyperlinks or forms that tell the client where it can go next and what actions are possible in the current state of its conversation with the API. Fulfilling this constraint has a major impact on the structure of a REST API, as it typically results in a graph-like structure of resources connected by hyperlinks.

In order to improve the state of the art in the design and realization of REST APIs, it is crucial to be aware of this state of the art. The goal of this work is to provide a framework for the structural analysis of REST APIs. This framework can serve as a first step in creating a current inventory of real-world REST APIs describing their structural characteristics. This inventory, comprising a set of metrics as well as a graphical representation for each API, can then be used to identify the main characteristics of today's REST APIs as well as their deficits with respect to compliance with the REST architectural style. We envision that knowing and analyzing these data in detail will allow deriving fitting approaches for resolving the identified deficits, helping to improve the state of the art with purposeful solutions.

The rest of the paper is structured as follows. In section II we give an overview about existing works on the analysis of REST APIs and position our work. Our analysis is based on a metamodel for REST APIs that we have developed in previous work. This metamodel as well as the transformation of the Swagger description languages into this metamodel are introduced in section III. The core contribution of this paper, the framework for the structural analysis of REST APIs, is presented in section IV. Section V concludes the paper with a discussion of the main results and a short outlook to future work.

II. Related Work

Several works already target the analysis of REST APIs. A first analysis of REST APIs has been conducted in [4]. The authors investigated a set of 222 Web APIs taken from ProgrammableWeb.com, a popular Web API directory. The analysis has been conducted manually and focuses on technical aspects of the selected APIs.

The work presented in [8] analyzes a set of 12 REST APIs with respect to a set of five patterns and eight anti-patterns. For each of these (anti-) patterns the authors define a corresponding heuristics and detection algorithm, both based on the observation and investigation of request and response messages exchanged with an API.

The work of [8] is continued and extended in [9]. Here, the authors focus on the analysis of the URI structure of REST APIs using a set of five linguistic patterns and anti-patterns that are applied to a set of 15 REST APIs. The general analysis approach is the same as in [8]. Each (anti-) pattern has a corresponding heuristics and detection algorithm, which are then applied to a set of previously gathered request messages.

In [10] a set of three REST APIs from three well-known cloud providers is analyzed with respect to a set of 73 best practices compiled from literature. The analysis is based on available API documentation and has been conducted manually, followed by a detailed analysis of the results.

In [11] a dataset of 78GB of HTTP traffic from an Italian mobile internet provider is analyzed with respect to REST principles and guidelines. The authors define a set of five best practices for REST APIs and a corresponding set of 18 heuristics for the compliance with these best practices. These heuristics are then implemented and applied to a representative sample of the whole dataset. In addition, the same heuristics are used to calculate the maturity level of the investigated REST APIs with respect to the maturity model by Richardson [12].

What has not been covered so far, to the best of our knowledge, is the analysis of REST APIs based on machine-readable API descriptions. REST API description languages like Swagger [13] and RAML [14] gain more and more importance, which amongst other things recently resulted in the Open API Initiative [15] as a standardization approach for API descriptions. We use this potential to allow a new perspective on REST APIs, supported by different analysis approaches. In this paper, we present a framework supporting an automated REST API analysis that is based on machine-readable API descriptions. This analysis focuses on the structure of REST APIs and can already be applied at design time, as it only requires a description (a model) of an API but no implementation.

III. A Canonical Metamodel for REST APIs

There exist many languages for the description of REST APIs from both academia as well as industry. For our analysis we are concentrating on description languages that are commonly used in real world, assuming that API descriptions based on these languages will then be available for a wide range of real-world APIs.

Before analyzing them, we will transform REST API descriptions available in different languages into a canonical metamodel. This approach has several benefits. First, most description languages do not explicitly describe the structure of an REST API, whereas our canonical metamodel does. Second, using the canonical metamodel as a common base for analysis enhances the portability of our analysis framework.

In order to get a better understanding about the structure of REST APIs and to help designers and developers to create better REST APIs, we have developed a set of metamodels for REST APIs and successfully integrated them in a model-driven approach for the design and realization of REST APIs [16][17][18]. The core model is the *atomic resource model*, which describes a REST API in terms of its basic elements like resources, methods, or representations. Another important model is the *URI model*. The HATEOAS constraint of REST demands that clients navigate through an API independent of any specific URIs by following hyperlinks. The separation of the resource model from the URI model reflects this very important aspect of REST and intends to support API designers as well as API clients in following the HATEOAS principle. A simplified version of the metamodel for the atomic resource model as well as for the URI model is shown in Fig. 1. Regarding the analysis of the structure of an REST API i.e. the analysis of the resources and their connections, the resource model contains all necessary information and we will therefore ignore the URI model in the following.

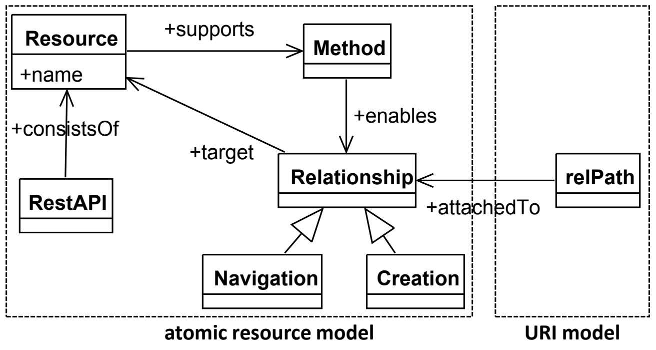


Fig. 1 Simplified metamodel for atomic resource model & URI model

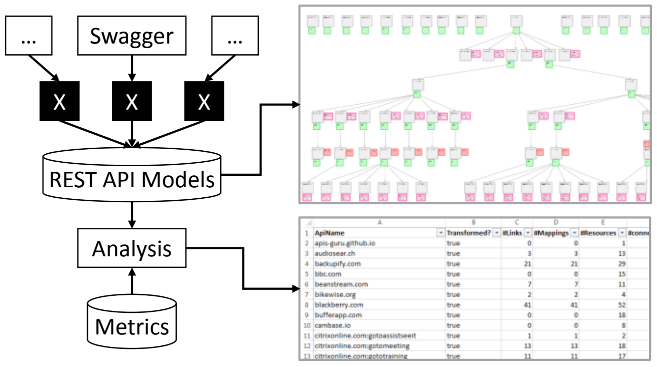
The Swagger API description language has established as the de-facto standard for modeling and describing REST APIs. Therefore, we decided to transform and analyze Swagger documents as a first validation of the feasibility of our framework. The transformation of a Swagger model comprises two phases. First, all resources are identified and transformed including their entire detailed configuration like the supported methods, representations or query parameters. Second, the relationships between the resources are identified. Unfortunately, Swagger does not provide any means to describe links between resources explicitly. Instead, the structure of a REST API is usually given by the paths the API provides. Here it is generally accepted that these paths represent hierarchical relationships. We build on this assumption for determining the relationships between resources. Consequently, the resulting resource graph is always a tree, as Swagger is inherently limited to such structures.

IV. Structural REST API Analysis Framework

The general approach of the REST API analysis framework is depicted in Fig. 2. Starting from available

REST API description documents, we transform them into our canonical metamodel and store them in a model repository. This way, the following analysis steps are easily reusable for other REST API description languages, they only need to be transformed into the canonical metamodel.

The models stored in the repository can be automatically processed by the analysis component, which in turn builds on a repository of algorithms that are able to calculate the metrics we are interested in, making this part easily extendable with new metrics as desired. The results of this analysis (i.e. a set of metric values for each REST API) are written to CSV files, allowing further analysis and processing by common office tools.



	A	B	C	D	E	
1	ApiName	Transformed?	#Links	#Mappings	#Resources	#conne
2	apis-guru.github.io	true	0	0	1	
3	audiosear.ch	true	3	3	13	
4	backupify.com	true	21	21	29	
5	bbc.com	true	0	0	15	
6	beanstream.com	true	7	7	11	
7	bikewise.org	true	2	2	4	
8	blackberry.com	true	41	41	52	
9	bufferapp.com	true	0	0	18	
10	cambase.io	true	0	0	8	
11	citrixonline.com:gotoassistseeit	true	1	1	2	
12	citrixonline.com:gotomeeting	true	13	13	18	
13	citrixonline.com:gototraining	true	11	11	17	

Fig. 2 Analysis approach

The REST API models stored in the repository can in addition be visualized using a graphical editor we developed as part of a toolchain around our REST API metamodel. We assume that the graphical representation of the resources and their relationships enables domain as well as REST experts to easily understand and assess the structure of an API.

As a first validation of our analysis approach, we performed a test run based on a set of 286 Swagger API description documents retrieved from https://apis.guru, a web page (and API) that describes itself as "Wikipedia for WEB APIs". The set includes only APIs that are publicly available (free or paid) and includes renowned providers like Microsoft Azure, Google, BBC, GitHub, Instagram, NYTimes, Spotify, and Wikimedia.

An overview about the aggregated values of all metrics that were calculated during the analysis is given in TABLE I. The first group of four metrics concentrates on the resources of an API. For the number of resources, i.e. the size of an API, the deviation between the mean value and the median indicates that the distribution is rather uneven and includes breakout values. This can in detail be seen in Fig. 3, which shows the distribution of the API size throughout the set of all APIs. The majority of APIs (53.5%) has a size of 10 or less resources (33% APIs have a size between 1 and 5 resources and 20.5% APIs have a size between 6 and 10 resources). Another 37.5% of all APIs has a size ranging between 11 and 40 resources and the remaining 9% have a size between 41 and the maximum of 264 resources (the distribution between 111 and 270 resources has been combined into one value in Fig. 3).

The set of APIs considered in the analysis comprises two noteworthy subsets, a set of 39 API models from Microsoft Azure, and a set of 105 API models from Google. As these two sets represent a significant amount of the complete API set, Fig. 3 also shows the distribution of the API size separately for the set of Azure APIs, the set of Google APIs, and the set of all remaining APIs. These three distributions vary in parts. The share of APIs with a size up to five resources is 23% and 36% for the Azure and Google APIs respectively, and nearly 45% for all remaining APIs. In contrast, the share of APIs with a size from six up to ten resources is much smaller for the remaining APIs than for Azure and Google.

TABLE I. AGGREGATED METRICS OVERVIEW

	MIN	MAX	MEAN	MEDIAN
#Resources	1	264	20,3	9
#ReadOnlyResources	0	227	10,4	4
#POST	0	93	6,5	3
#DELETE	0	40	2,6	1
#roots	1	227	8,1	4
#Links	0	248	12,2	4
maxDepth	0	7	1,8	1

The next metric, the number of read-only resources (#ReadOnlyResources in TABLE I.) counts all resources that support only the GET method but no other methods. For POST and DELETE, we count all resources that support these methods (and maybe others, as usually every resources is supposed to support GET requests).

The distribution of the share of read-only resources in an API is shown in Fig. 4, again for the whole API set (bars) as well as separately for the three subsets (curves). Looking at the whole set, it is noticeable that around 24.5% of all APIs have a share of read-only resources between 90% and 100%, i.e. these APIs focus on information retrieval rather than on content creation and manipulation. Looking at the three subsets, their distributions are rather different. The majority of Azure APIs (61.5%) has a read-only share between 30% and 60%, whereas the distribution for the Google APIs is more even. For the set of all remaining APIs, 50.5% have a read-only share of 90% or more. These differences probably

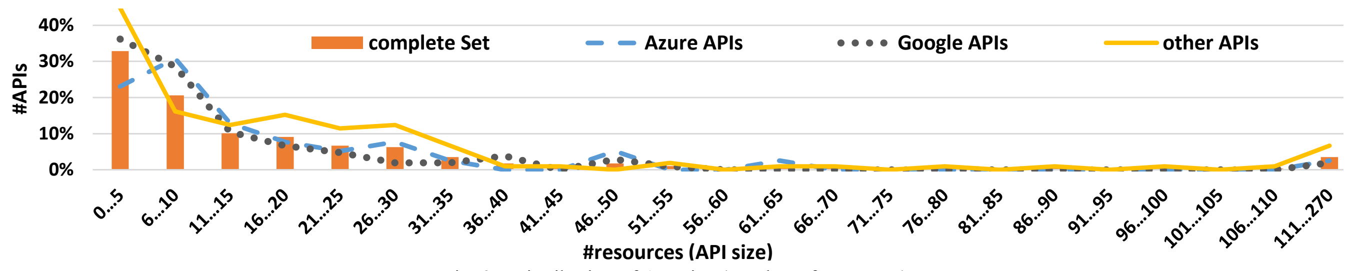


Fig. 3 Distribution of API size (number of resources)

result from the fact, that the Azure and Google APIs provide similar functionality in their APIs (both provide common cloud services) which includes not only information retrieval but also content creation and manipulation. The set of all remaining APIs however covers a much broader spectrum of services, which evidently includes a significant set of information services.

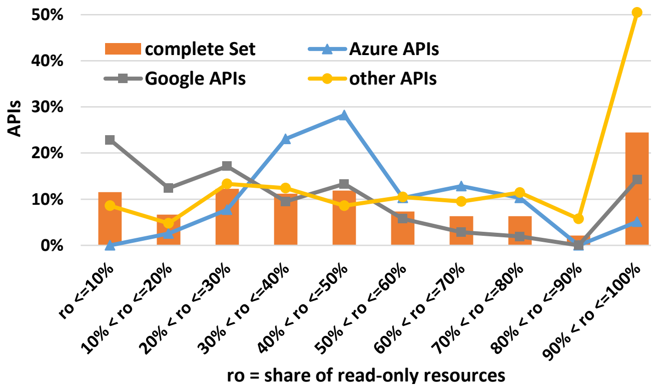


Fig. 4 Share of read-only resources in APIs

The next group of three metrics in TABLE I. adds data about links between resources to the analysis. A generally accepted best practice in REST API design, driven by the HATEOAs constraint, is that an API should have only one (or at least few) root resource [19]. However, the numbers in TABLE I. show that today's REST APIs usually have quite some root resources. The maximum depth of an API is given by the longest path of resources that are connected by links. This metric shows rather small values if we compare it to the number of resources, which indicates that APIs are in general "more wide than deep". Speaking from a client's view, this means that when navigating through an API there are comparatively little possibilities to navigate "deeper into the API" but at each of these steps, there are in average many alternatives to navigate further.

V. Discussion and Outlook

The work we presented in this paper combines two approaches that are both new to the area of REST API analysis. First, we use API description documents as starting point for the analysis, and second, we focus on analyzing the structure of REST APIs. In addition, we rely on a canonical metamodel for REST APIs, an approach that has several benefits. Our canonical metamodel explicitly describes the structure of a REST API, whereas most description languages do not. Furthermore, using the canonical metamodel as a common base for analysis enhances the portability of our analysis framework.

A first validation of the framework provided metrics for a high-level overview and characterization of a large set of real-world REST APIs. We discovered that the APIs under investigation are on average small with a median of nine resources per API, but that the distribution of the API size is rather uneven and that some APIs have more than 250 resources. Another result is that read-only resources are very common and that there is even a subset of APIs that are completely read-only.

As a next step, we plan to focus on the analysis step and the resulting metrics. The main research questions are the identification of suitable metrics reflecting quality attributes of REST APIs (e.g. complexity or REST compliance) as well as their validation.

Acknowledgment

This work has been partially funded by the German Research Foundation (DFG) within the Cluster of Excellence in Simulation Technology (EXC310/2) at the University of Stuttgart.

References

[1] J. Webber, S. Parastatidis, and I. Robinson, "REST in practice: Hypermedia and systems architecture", O'Reilly Media, 2010.

[2] R.T. Fielding and R.N. Taylor, "Principled design of the modern Web architecture", ACM Trans. Internet Technol. 2, May 2002: 115-150.

[3] D. Renzel, P. Schlebusch, and R. Klamma, "Today's top 'RESTful' services and why they are not RESTful", WISE, 2012.

[4] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web APIs on the World Wide Web", The 8th IEEE European Conference on Web Services (ECOWS 2010), 1-3 Dec 2010, Ayia Napa, Cyprus.

[5] P. Adamczyk, P.H. Smith, R.E. Johnson, and M. Hafiz, "REST and Web services: In theory and in practice", REST: from Research to Practice, Springer New York, 2011.

[6] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, 2014, http://www.ietf.org/rfc/rfc7231.txt.

[7] F. Haupt, M. Fischer, D. Karastoyanova, F. Leymann, and K. Vukojevic-Haupt, "Service composition for REST", Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International (pp. 110-119). IEEE.

[8] F. Palma, J. Dubois, N. Moha, and Y.G. Guéhéneuc, "Detection of REST patterns and antipatterns: a heuristics-based approach", ICSOC 2014, Springer Berlin Heidelberg, 2014.

[9] F. Palma, J. Gonzalez-Huerta, N. Moha, Y.G. Guéhéneuc, and G. Tremblay, "Are restful apis well-designed? detection of their linguistic (anti) patterns." International Conference on Service-Oriented Computing. Springer Berlin Heidelberg, 2015.

[10] F. Petrillo, P. Merle, N. Moha, and Y.G. Guéhéneuc, "Are REST APIs for Cloud Computing Well-Designed? An Exploratory Study." ICSOC 2016, Springer International Publishing, 2016.

[11] Rodríguez, Carlos, et al. "REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices." International Conference on Web Engineering, Springer, 2016.

[12] M. Fowler, "Richardson maturity model: steps toward the glory of rest", http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

[13] Swagger, http://swagger.io/

[14] RESTful API Modeling Language (RAML), http://raml.org/

[15] Open API Initiative, https://www.openapis.org/

[16] F. Haupt, D. Karastoyanova, F. Leymann, and B. Schroth, "A model-driven approach for REST compliant services", ICWS, 2014.

[17] F. Haupt, F. Leymann, and C. Pautasso. "A conversation based approach for modeling REST APIs." WICSA 2015 , IEEE, 2015.

[18] K. Vukojevic-Haupt, F. Haupt, F. Leymann, and L. Reinfurt, "Bootstrapping Complex Workflow Middleware Systems into the Cloud." e-Science 2015, IEEE, 2015.

[19] M. Nottingham, "Home Documents for HTTP APIs", https://tools.ietf.org/html/draft-nottingham-json-home-05

All links were last followed on 26.02.2017.