Governify for APIs: SLA-Driven Ecosystem for API Governance

Antonio Gamez-Diaz
Universidad de Sevilla
Seville, Spain
antoniogamez@us.es

Pablo Fernandez
Universidad de Sevilla
Seville, Spain
pablofm@us.es

Antonio Ruiz-Cortés
Universidad de Sevilla
Seville, Spain
aruiz@us.es

ABSTRACT

As software architecture design is evolving to a microservice paradigm, RESTful APIs are being established as the preferred choice to build applications. In such a scenario, there is a shift towards a growing market of APIs where providers offer different service levels with tailored limitations typically based on the cost. In such a context, while there are well-established standards to describe the functional elements of APIs (such as the OpenAPI Specification), having a standard model for Service Level Agreements (SLAs) for APIs may boost an open ecosystem of tools that would represent an improvement for the industry by automating certain tasks during the development.

In this paper, we introduce *Governify for APIs*, an ecosystem of tools aimed to support the user during the SLA-Driven RESTful APIs' development process. Namely, an *SLA Editor*, an *SLA Engine* and an *SLA Instrumentation Library*. We also present a fully operational *SLA-Driven API Gateway* built on the top of our ecosystem of tools. To evaluate our proposal, we used three sources for gathering validation feedback: industry, teaching and research.

- **Website**: links.governify.io/link/GovernifyForAPIs
- **Video**: links.governify.io/link/GovernifyForAPIsVideo

CCS CONCEPTS

• **Information systems → RESTful web services**; • **Software and its engineering → Extra-functional properties**; **System description languages**.

KEYWORDS

RESTful APIs, SLA, OpenAPI Specification, SLA-driven APIs, API Gateways

ACM Reference Format:
Antonio Gamez-Diaz, Pablo Fernandez, and Antonio Ruiz-Cortés. 2019. Governify for APIs: SLA-Driven Ecosystem for API Governance. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3338906.3341176

1 INTRODUCTION

In the last decade, RESTful APIs are becoming a clear trend as composable elements that can be used to build and integrate software [8]. One of the key benefits this paradigm offers is a systematic approach to information modeling leveraged by a growing set of standardized tooling stack. In this context, the term of *API Economy* is being increasingly used to describe the movement of the industries to share their internal business assets as APIs [7, 11] not only across internal organizational units but also to external third parties; in doing so, this trend has the potential of unlocking additional business value through the creation of new assets [1, 7].

In order to be competitive in such a growing market of APIs, at least two key aspects can be identified: i) *ease of use* for its potential developers; ii) a flexible usage *plan* that fits their customer's demands.

Regarding the *ease of use* perspective, third-party developers need to understand how to use the exposed APIs so it becomes necessary to provide good training material but, unfortunately, API providers do not often write good documentation of their products [2]. Notwithstanding, during the last years, the *OpenAPI Specification*[1] (OAS), formerly known as *Swagger*, has become the *de facto* standard to describe RESTful APIs from a functional perspective providing an ecosystem of tools[2] that helps the developer in several aspects of the API development lifecycle.

Concerning the usage *plans* perspective, as APIs are deployed and used in real settings, the need for non-functional aspects is becoming crucial. In particular, the adoption of Service Level Agreements (SLAs) [9] could be highly valuable to address significant challenges that industry is facing, as they provide an explicit placeholder to state the guarantees and limitations that a provider offers to its consumers. Exemplary, these limitations (such as *quotas* or *rates*) are present in most common industrial APIs [3] and both API providers and consumers need to handle how they monitor, enforce or respect them with the consequent impact in the API deployment and consumption.

However, to the best of our knowledge, there is no widely accepted and open source tool that leverages from the functional model as well as the non-functional description to create usage plans including elements such as cost, functionality restrictions or limits and performing actual API governance in production.

In this paper, we introduce the *Governify for APIs* ecosystem, a set of tools which, starting from an OAS description, assist the user during the RESTful APIs' development process for the creation of usage plans (or SLAs) and performing seamlessly SLA-Driven API governance. Ultimately, we have evaluated our proposal in three different scenarios: teaching, research and industry.

The rest of the paper is structured as follows: Section 2 analyzes different alternatives to our proposal, Section 3 presents the *Governify for APIs* ecosystem and Section 4 outlines the evaluation that

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

[1] The latest version of the OpenAPI Specification is available at https://github.com/OAI/OpenAPI-Specification
[2] https://openapi.tools

we performed. Finally, in Section 5, shows some final remarks and conclusions.

2 RELATED TOOLS

Despite ad-hoc solutions for regulating APIs have emerged, we focus on the so-called API Gateways [5], which have emerged to support API developers in the management of aspects such as consumer authentication, request throttling or billing. The increasing growth of APIs has resulted in a proliferation of API Gateways platforms that provide different levels of support for functional and non-functional aspects.

In order to spot these differences, we have considered[3] the publicly available information of 18 API Gateways. The conclusions:

1) Regarding the functional point of view: (i) Almost every API Gateway do allow the usage of the OAS for the functional definition of the API. (ii) Almost a half have an explicit (vendor-specific) model to define some parts of the configuration of the platform (e.g. endpoints, limitations, general information), but only one third allow importing/exporting. (iii) As stated by the authors in [10], the importance of OAS made it a key feature widely supported by the API Gateway provider.

Concerning the non-functional point of view: (i) Most API Gateways support quotas, against two thirds supporting rates. (ii) All API Gateways allow request as a metric, a third allows other predefined metric and only two API Gateways allow defining custom metrics. (iii) As pointed out in [3], API providers typically support limitations (quotas and/or rates) and they are usually defined over the number of requests with a minimal frequency supported that starts from minutely in the case of quotas, and secondly in the case of rates.

Based on these findings, we observe that, in spite all API Gateways spot similar features, the underlying model and concepts are different and each platform describes the configuration in a specific format, hindering, thus, the interoperability among providers. Consequently, organizations that face the transition from a certain API Gateway to a different one, they are required to perform a manual migration process and complex evaluation of the behavioral and vocabulary differences between the vendor-specific models of each API Gateway.

3 GOVERNIFY FOR APIS

3.1 Motivating Example

We will guide the explanation of the *Governify for APIs* ecosystem by means of a real-world RESTful API which needs to be governed; namely, the DBLP API, a service for retrieving bibliometric information in the Computer Science research area. It is a quite simple API that offers three GET endpoints for searching *authors*, *publications* and *venues* by introducing parameters in the query (e.g., *http://dblp.org/search/author/api?q=gamez+diaz*).

This API does not offer any explicit information of the non-functional properties or limitations besides of an entry in the FAQ[4] in natural language (sic. *you should always be fine when waiting for at least one or two seconds between two consecutive requests*). This lack hinders the creation of limitations-aware API clients because of the need for a prior human progressing of the aforementioned FAQ. Nevertheless, from the functional perspective, this API could be easily described by means of the OpenAPI Specification (c.f., resources on the website[5]). For the non-functional modeling, SLA4OAI[6] will be used. It is a vendor-neutral open-source proposal for describing APIs' aspects such as quotas and rates. It enables users to model the typical limitations that can be usually found in an API [3]. Note SLA4OAI is able to model a subset of terms that usually appear in an SLA; nevertheless, SLA can compliant with other SLA specifications, such as iAgree [6], including support for other metrics (e.g., availability) and concepts (e.g., penalties and rewards).

Starting from an API functionally defined with OAS, our goal will be to use *Governify for APIs* to regulate that API accordingly to its quota and rate limitations using a vendor-neutral specification.

3.2 Architecture

We have developed a Node.js set of tools, inspired in the microservice architectural style, packed as Docker public images[7] and deployed for a publicly online access. An overview of all the components is depicted in Figure 1. The source code and technical information are available at the supplementary website[5]. Specifically, we introduce the *SLA Editor* for hiding the complexity of the language to the end user. Next, we support two different enforces: gateway and standalone. The former, see Figure 3, is intended to be developers who want to regulate an API without modifying the source code. The latter is supposed to serve to developers who need a more fine-grained control by modifying the API code. We provide a *SLA-Driven API Gateway* for the first enforce and an *SLA Instrumentation Library* as part of a Node.js module for the latter. Both enforces are instrumented by the *SLA Engine*. Next, we describe in detail each tool.

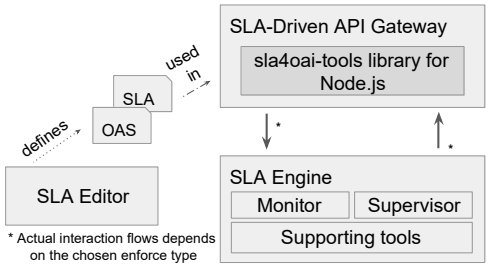

Figure 1: *Governify for APIs*' simplified architecture

3.3 SLA Editor

Governify for APIs provides an *SLA editor*[8], a user-friendly web-based text editor specifically developed for assisting the user during the modeling tasks, including auto-completion, syntax checking, and automatic binding (i.e., the changes in the UI are synchronized with the underlying SLA4OAI textual model). Precisely, Figure 2 depicts this textual-visual binding. It is possible to create different plans (e.g., *free* and *pro*) with quotas and rates over specific metrics.

[3] Avaliable at https://isa-group.github.io/2019-05-sla4oai-demo/files/api_gw.html
[4] https://dblp.org/faq/1474706
[5] https://links.governify.io/link/GovernifyForAPIs
[6] https://sla4oai.specs.governify.io
[7] https://hub.docker.com/u/isagroup
[8] https://designer.governify.io

Clicking in the + sign, the user is able to select the path and method (previously defined in the OAS document) for entering the value of the limitation. Note that other custom metrics besides *requests* can also be defined. Precisely, in the case of the DBLP API, after having modeled the functional part with OAS, the plan can be easily created. We added a made up professional plan just to show the modeling capabilities.

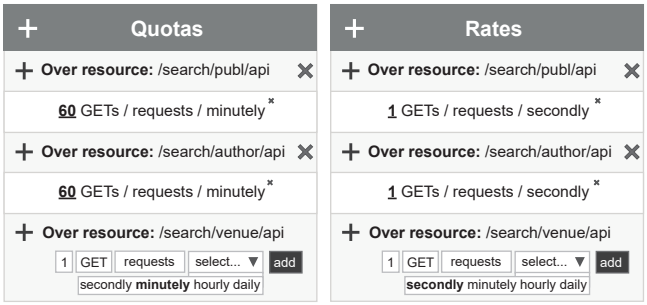

Figure 2: Editing DBLP plans in the *SLA Editor* tool

3.4 SLA Engine

The SLA4OAI specification outlines the *Basic SLA Management Service*[9] (BSMS) defining the interaction flows and the endpoints */check* and */metrics*. In Figure 3 we focus on the *Gateway* enforce as it is a more complete scenario.

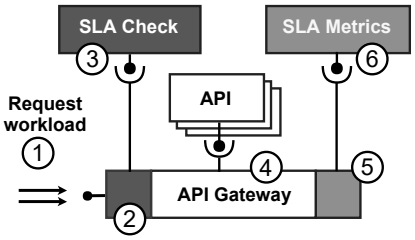

Figure 3: *Gateway* enforce defined in the SLA4OAI BSMS

First, requests will pass through the API Gateway until they are directed to the node that will serve it (step 1). Next, the API Gateway query the *SLA Check* API to determine if the request is authorized to develop the actual operation based on the appropriate SLA (step 2). Afterward, if it is authorized, the actual API is invoked and the response is returned (step 3). If it is not, a status code and a summary of the reason (as generated by the SLA check API) is returned (step 3). After the consumption ends (step 4), the metrics are sent to the *SLA Metrics* API (step 5), which is in charge of updating the status of the agreement with the new metrics introduced (step 6).

Since we stick to the SLA4OAI specification and it left open the implementation, our tooling for the*SLA Engine*, provides a concrete implementation of the BSMS, including also a particular way to handle SLA and users saving/retrieving tasks (*SLA Registry* and *Tenants*). Specifically, *Monitor*[10] is an implementation of the *Metrics* BSMS service and *Supervisor*[11], of the *Check* service.

The *Monitor* service exposes a POST operation in the route */metrics* for gathering the metrics collected from other different services. It can collect a set of basic metrics and send them to a data store for aggregation and later consumption. The metrics can be grouped in batches or sent one by one to fine-tune performance versus real-time SLA tracking.

The *Supervisor* service has a POST */check* endpoint for the verification of the current state of the SLA for a given operation in a certain scope. For each request, this service will evaluate the state of the SLA and will respond with a positive or negative response depending on whether a limitation has been overcome. In addition, this service also implements (outside the scope of the BSMS) these additional endpoints: GET/POST */tenants*, GET/POST */slas* and PUT/DELETE */slas/<id>* for managing both users (tenants and accounts) and SLA4OAI documents themselves.

3.5 SLA Instrumentation Library

Despite the fact that the BSMS defines the interaction flows between the endpoints, the concrete implementation of these interactions is left open. That is the way our aims to cover this lack. Specifically, we present an *SLA Instrumentation Library* for Node.js[12], which is a middleware (i.e., a filter that intercepts the HTTP requests and perform transformation if necessary) written for Express, the most used Node.js web application framework. This middleware intercepts all the inbound/outbound traffic to perform the BSMS flow. Throughout the Listing 1 we observe that it is necessary to import the library (line 3), to configure the endpoints of the services (lines 8 and 9) and finally register the middleware (line 12).

```
1 // Imports
2 const express = require("express");
3 const slaInstrumentationLib = require("sla4oai");
4
5 const app = express(); // Express init
6
7 // SLA4OAI init
8 const supervisorURL = {url: "supervisor.oai.governify.io"};
9 const monitorURL = {url: "monitor.oai.governify.io"};
10
11 // Express middleware registration
12 slaInstrumentationLib.register(app,supervisorURL,monitorURL);
```

Listing 1: Excerpt of the configuration of the SLA Instrumentation Library

3.6 SLA-Driven API Gateway

A more transparent way to implement the interaction flows defined is the BSMS is achieved by using a *Gateway* SLA enforce. Our tool, the *SLA-Driven API Gateway* is an open-source implementation to be deployed using any *SLA Engine*. Particularly, we provide an online preconfigured instance[13] using the aforementioned *SLA Instrumentation Library*. As depicted in Figure 4, API providers are only required to enter: (i) The real endpoint of their API; (ii) An URL pointing to the SLA4OAI document. Once an API is registered, the *SLA-Driven API Gateway* exposes a public and SLA-regulated

[9] https://sla4oai.specs.governify.io/operationalServices.html

[10] https://monitor.oai.governify.io/api/v1/docs
[11] https://supervisor.oai.governify.io/api/v1/docs
[12] https://www.npmjs.com/package/sla4oai-tools
[13] https://gateway.oai.governify.io

endpoint, as well as a */plans* endpoint for a provisioning portal. It enables customers to purchase a plan, after that, this customer will get an API-key, acting as a bearer token for HTTP authentication to consume the SLA-regulated API.

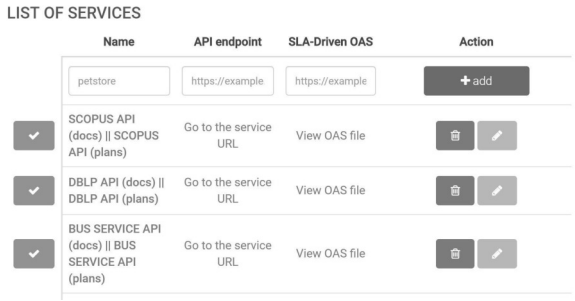

Figure 4: Configuration UI of the SLA-Driven API Gateway

4 EVALUATION

We have performed a threefold qualitative evaluation in industry, teaching and research contexts.

Concerning the industrial evaluation, some OpenAPI Initiative members have expressed its interest in SLA4OAI, the SLA modeling proposal, and in promoting a working group for evolving and extending it. Indeed, in [4], we collaborated with people from Google, Paypal, AsyncAPI Initiative and Metadev for analyzing, starting from SLA4OAI, the status of SLAs and limitations in the industry. Furthermore, in spite of the fact the SLA4OAI extension and tools have not been widely announced nor promoted, we have disclosed the tooling ecosystem into the main public Node.js artifact repository (i.e., NPM) and this platform provides a set of analytics of usage since their publishing. Specifically, based on its data we observe that *SLA Instrumentation Library* has been downloaded and installed more than 600 times[14] while the *SLA Engine* was downloaded more installed than 1900 times.

Regarding the use of *Governify for APIs* in teaching, it has been extensively used in, at least, two undergraduate service-oriented related subjects. As students were required to create their own APIs[15], they also had to set the rate and quota limitations using *Governify for APIs*. Whereas we do not have any specific usage report, we collected useful information, issues and bugs derived from running in production.

As of the research context, we are validating our proposal (language and tools) in a national research network. Several members are exposing their research results by creating an API and applying limitations using *Governify for APIs* and SLA4OAI. Then, all these artifacts are being deployed in a central publicly available catalog[16].

5 CONCLUSIONS

In this work, we have presented the *Governify for APIs* ecosystem, a set of tools integrated aimed to support the user during the SLA-Driven RESTful APIs' lifecycle. Specifically, an *SLA Editor* and an *SLA-Driven API Gateway* on the top of an *SLA Engine* composed by an *SLA Monitor* and an *SLA Check* APIs.

We have evaluated our proposal in three different scenarios: teaching, research and industry, getting, therefore, a highly valuable source of information that will be used in the upcoming improvements. With *Governify for APIs* we prove that, with state-of-the-art tools, it is possible to improve lifecycle of SLA-Driven RESTful APIs, especially those problems derived from design and operation.

Specifically, (i) Complex usage plans with quota and rate limitations can be modeled with an OAS-compliant vendor-neutral format; (ii) The support of vendor-neutral initiatives paves the way for the interoperability between API Gateway providers; (iii) *Governify for APIs* left a publicly available ecosystem of open-source tools supporting the SLA-Driven RESTful APIs' lifecycle.

[14] https://npm-stat.com/charts.html?package=sla4oai-tools
[15] https://github.com/gti-sos
[16] https://services.rcis.governify.io

ACKNOWLEDGMENTS

This work is partially supported by the European Commission (FEDER), the Spanish Government under projects BELI (TIN2015-70560-R) and HORATIO (RTI2018-101204-B-C21), and the FPU scholarship program, granted by the Spanish Ministry of Education, Culture and Sports (FPU15/02980). The authors would like to thank for their valuable technical contributions to Daniel Arteaga and Felipe Vieira da Cunha.

REFERENCES

[1] Michele Bonardi, Maurizio Brioschi, Alfonso Fuggetta, Emiliano Sergio Verga, and Maurilio Zuccalà. 2016. Fostering Collaboration Through API Economy: The E015 Digital Ecosystem. In *Proceedings of the 3rd International Workshop on Software Engineering Research and Industrial Practice (SER&IP '16)*. ACM, New York, NY, USA, 32–38. https://doi.org/10.1145/2897022.2897026

[2] Forrester. 2015. *API Management Solutions , Q3 2014.* Technical Report. Forrester.

[3] Antonio Gamez-Diaz, Pablo Fernandez, and Antonio Ruiz-Cortes. 2017. An Analysis of RESTful APIs Offerings in the Industry. In *Service-Oriented Computing*, Michael Maximilien, Antonio Vallecillo, Jianmin Wang, and Marc Oriol (Eds.). Springer International Publishing, Cham, 589–604.

[4] Antonio Gamez-Diaz, Pablo Fernandez, Antonio Ruiz-Cortes, Pedro J. Molina, Nikhil Kolekar, Prithpal Bhogill, Madhuranjan Mohaan, and Francisco Méndez. 2019. The role of limitations and SLAs in the API industry. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*. ACM, Tallin, Estonia. https://doi.org/10.1145/3338906.3340445

[5] Antonio Gámez-Díaz, Pablo Fernández-Montes, and Antonio Ruiz-Cortés. 2015. Towards SLA-Driven API Gateways. In *Actas de las XI Jornadas de Ingeniería de Ciencia e Ingeniería de Servicios*, Juan Manuel Murillo (Ed.), Vol. 201232273. Sistedes, Santander, 9. https://doi.org/10.13140/RG.2.1.4111.5609

[6] Antonio Gámez-Díaz, Pablo Fernández-Montes, and Antonio Ruiz-Cortés. 2018. Fostering SLA-Driven API Specifications. In *Actas de las XIV Jornadas de Ingeniería de Ciencia e Ingeniería de Servicios*, Manuel Lama (Ed.). Sistedes, Sevilla. https://doi.org/10.13140/RG.2.2.35748.53128

[7] Jose Maria Garcia, Pablo Fernandez, Antonio Ruiz-Cortes, Schahram Dustdar, and Miguel Toro. 2017. Edge and cloud pricing for the sharing economy. *IEEE Internet Computing* 21, 2 (3 2017), 78–84. https://doi.org/10.1109/MIC.2017.24

[8] Holger Harms, Collin Rogowski, and Luigi Lo Iacono. 2017. Guidelines for Adopting Frontend Architectures and Patterns in Microservices-based Systems. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 902–907. https://doi.org/10.1145/3106237.3117775

[9] C. Muller, A. Gutierrez Fernandez, P. Fernandez, O. Martin-Diaz, M. Resinas, and A. Ruiz-Cortes. 2018. Automated Validation of Compensable SLAs. *IEEE Transactions on Services Computing* (jan 2018), 1–1. https://doi.org/10.1109/TSC.2018.2885766

[10] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. 2018. An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing* (2018). https://doi.org/10.1109/TSC.2018.2847344

[11] W. Tan, Y. Fan, A. Ghoneim, M. A. Hossain, and S. Dustdar. 2016. From the Service-Oriented Architecture to the Web API Economy. *IEEE Internet Computing* 20, 4 (July 2016), 64–68. https://doi.org/10.1109/MIC.2016.74