Towards SLA modeling for RESTful APIs

Antonio Gámez-Díaz, Pablo Fernandez, and Antonio Ruiz-Cortés

University of Sevilla*
{agamez2,pablofm,aruiz}@us.es

Abstract. The term of *API Economy* is becoming increasingly used to describe the change of vision in how APIs can add value to the organizations. Furthermore, a greater automation of RESTful APIs management can suppose a competitive advantage for the company. New proposals are emerging in order to automatize some API governance tasks and increase the ease of use (e.g. generation of code and documentation). Despite that, the non-functional aspects are often addressed in a highly specific manner or even there not exists any solution for an automatic governance. Nevertheless, these properties are already defined in natural language at the Service Level Agreement (SLA) that both customer and provided have established.
In this paper, we carry out a study on the *aaS industry and analyze the current both API modeling and SLA modeling proposals in order to identify the open challenges for an automatic RESTful API governance.

1 Introduction

Distribution models of information systems are moving to **aaS* paradigms where customers no longer need to buy a perpetual license, host the software or maintain the infrastructure. As part of the current *Service Oriented Computing* paradigm, the micro-service architectures are rapidly emerging [5]. In this model, an application is divided into a set of small services deployed independently. They communicate each other with some lightweight protocol (e.g. HTTP) thus, each resource could be accessed by a URI following, therefore, the REST principles.

The term of *API Economy* is being increasingly used to describe the movement of service computing from academia towards the industry since people expect the technologies to be more practical [22]. Moreover, industries are considering the positive effects of exposing their business units through APIs in order that they can be used by third parties. In this context, we found marketplaces where providers get paid for the API usage that their clients made. As part of this business goals alignment, companies often identify two key aspects to be more competitive: *ease of use* and service *guarantees*.

* This work has been partially supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programs (grants TIN2015-70560-R (BELI) and P12–TIC-1867 (COPAS)) and TIN2014-53986-REDT (RCIS).

From an *ease of use* perspective, third party developers need to understand how to use the exposed APIs so it becomes necessary to provide a good documentation. Indeed, one of the main reasons for the failure of projects is the lack of good documentation. Moreover, API providers do not often make a good documentation of their products [6]. Nevertheless, as the *API Economy* model is being strengthened in the market, some industry proposals have emerged trying to model these RESTful APIs and then generate the documentation.

From a *guarantees* perspective, the *Service Level Agreements* (SLAs) have been the documents used in the industry to define the responsibilities and rights of each party, i.e. both the consumer and provider of the service, according to their business model (e.g. a free plan with a limit of 100 requests per minute and a premium plan with no request limitation) [15]. Specifically, a typical SLA introduces some guarantees over the service provision as much as some compensations in the case of the provision is under-fulfilled or over-fulfilled.

SLAs are normally included in the general service conditions in natural language [12], thus making unfeasible an automatic computational processing. Although a number of SLA modeling proposals have emerged [17], the problem has not been studied from an API perspective.

In this work, we analyze the current situation in both SLA and API modeling in order to evaluate and address a joint modeling mechanism to perform an automated management over RESTful APIs by establishing some open challenges endorsed by a study over 27 SaaS offerings.

This paper is organized as follows: Section 1 shows an analysis over some RESTful APIs in the *aaS industry. In Section 2, we describe some important proposals in API modeling area. Next, Section 3 points out the current context in SLA modeling languages. Finally, Section 4 shows some remarks and conclusions.

2 SLAs in the *aaS industry

In this section, we present a study[1] over 27 SaaS with RESTful APIs which was carried out in two different stages:

In the first stage: data acquisition by students: i) about 30 students were given two API repositories (ProgrammableWeb[2] and Mashape[3]); ii) students chose a *aaS service; iii) they filled out a form regarding some *aaS characteristics manually identified.

In the second stage: Subsequent analysis by researcher: i) data manual validation and classification, giving a a result 55 analyzed *aaS offerings (40 SaaS, 1 PaaS and 14 IaaS); ii) selection of these ones which provide a RESTful API, giving a set of 27 elements. iii) we developed a comparative framework based upon 3 different attributes:

[1] Data of this study is available at http://dx.doi.org/10.17632/ybrp8mgk96.1
[2] https://www.programmableweb.com
[3] https://market.mashape.com

Plans: *number of plans* that defines the service level delivered to the customer, *existence of custom plans*, in order that they can fit the business needs and the *existence of free plans* such a starter or a trial plan.

Billing: *maximum cost* and *minimum cost*, in US Dollars, of every set of plans; *billing period*, e.g. monthly, yearly, etc. and *billing type* that shows the manner in how a customer is billed per service usage. For instance, a pay-as-you-go plan charges the customer just for its usage whereas a flat rate plan will charge the same amount of money independently of its usage. There exist other billings models that can combine the previous ones or charge the customer once a time (usually because the offering is coupled to a hardware acquisition and it is just a value added service).

Limitations: the *existence of functionality limitations*, i.e. whether an offering enables or disables some features depending upon the acquired plan, the *existence of operation limitations*, i.e. whether the provider limits the transactions or operations of the service. This limitation can be managed by applying rates or quotas. Both of them are thresholds used to limit the operations or requests when they are reached; the difference resides in the static window for quotas (e.g. 100 requests per day) and the sliding window for rates (e.g. 100s requests per each minute).

A 49.09% of the 65 *aaS studied offerings have a RESTful API. The average number of plans that these offerings provide is 3.85, with a $\sigma = 1.81$. A 62.96% provide a free plan, whereas a 40.74% offer custom plans. Regarding the billing models, a 48.15% opt for a flat rate billing model, a third choose a pay-as-you-go model whereas a 18.51% pick out combined and other models. Since the *aaS offerings are much diverse, the average cost ranges from $44.35 to $726.56.

On the other hand, a 85.19% of these offerings limit the number of requests or operations allowed, whereas a 59.26% limit the features. It is remarkable that quotas and rates usage is not distributed homogeneously, as depicted in the Figure 1, since it is more common to have quotas in flat rate plans contrariwise to the pay-as-you-go plans, where use to limit using rates. Concerning to the periods used in both quotas and rates, a large majority opts to have a minute-based temporal window, as shown in Figure 2.

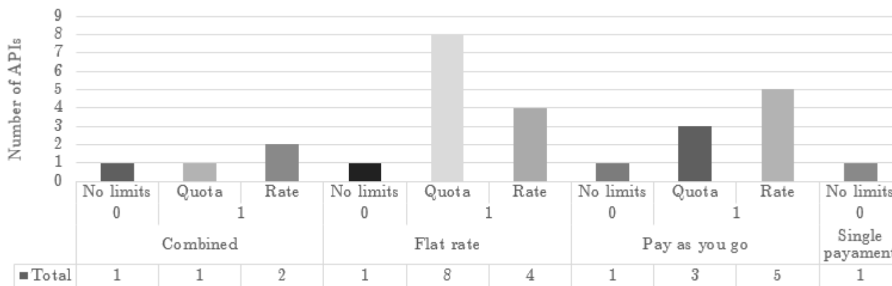

Fig. 1. Distribution of quotas and rates usage.

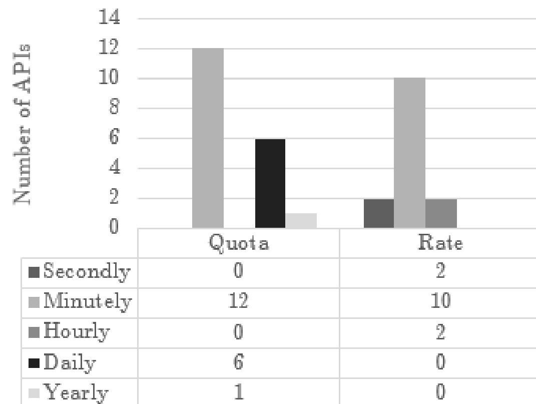

Fig. 2. Distribution of periods in quotas and rates.

After this analysis, it is possible to conclude that real-world *aaS offerings frequently have non-functional properties like billing cycles, rate and quota limitation, pricing plans, etc. In Section 2 we analyze a set of modeling techniques to study how to deal with these properties.

3 API modeling

In this section, we discuss different API modeling techniques used both in the industry and academia to define RESTful systems. API modeling languages have evolved from a classical SOA perspective, such as WSDL (2001), to become a tool to automatize the documentation generation process, such as API Blueprint (2013). Specifically, we have analyzed 6 proposals, used in both industry and academia. Finally, Table 1 depicts the differences between these languages.

WSDL [3] is a W3C recommendation since 2001 to represent the contract between the both service provider and service consumer using XML. Due to its complexity, WSDL usage is continuously decreasing in favor of more simple alternatives with tool support.

WADL [10] is a *Sun Microsystems* initiative to describe, in an XML format, web services invoked through HTTP. Although this proposal was submitted in 2009 to the W3C, it never became a standard. Its complexity may have contributed to the continuously decreasing usage.

OAI (formerly known as Swagger, initiated in 2010) [24] in an open specification created by a consortium integrated by organizations like *Google*, *IBM*, *Microsoft*, *PayPal*, *Apigee*, *Apiary*, *Restlet* and *Mashape*. OAI aims to support to the whole API life-cycle and improve the traceability between the documentation and the auto-generated code by releasing multiple open tools for both code and documentation generation.

RAML (*RESTful API Modeling Language*) [21] is language based on *YAML* syntax [19] and created in 2013 by *Mulesoft*, *AngularJS*, *Inuit*, *Airware*, *ProgrammableWeb*, *Akana*, *Cisco*, *VMware* y *Akamai*. It covers the whole API

life-cycle [4]: design, construction, testing, documentation and publishing. It is a well-assented language in the industry and some specification-complaint tools have been developed, ranging from IDE plugins to ease the modeling stage to both code and documentation generation tools.

IO Docs [25], created in 2011 by *Mashery*, is a description language oriented to the generation of documentation through a JSON Schema [7]. It defines all the necessaries resources, methods and parameters in order that an auto-generated JavaScript client could consume the API.

API Blueprint [2] is a description language for APIs, created by *Apiary* in 2013, oriented to the generation of documentation. It is made up of some semantic assumptions about Markdown syntax. It is possible to create documentation, prototype APIs and test existing APIs.

Table 1 depicts a comparative analysis considering these 6 proposals. Each column corresponds to a certain analyzed feature, so that a ✓in cell $C_{i,j}$ means the proposal i has the property j.

1. **Resources**. In a RESTful context, URIs, resource representations and API operations are all built around the concept of resources. The language has to be expressive enough in order that it can model the resources at the right granularity. Furthermore, HTTP methods (i.e. GET, POST, PUT and DELETE) should be able to be modeled.
2. **HTTP headers**. Ability to model custom headers, such as *Authorization* or *Content-Type*.
3. **Authentication**. Capability to model API security mechanisms, such as *basic authentication* or *bearer token*.
4. **Documentation generation**. Automatic generation of documentation portals.
5. **Tool support**. The existence of ecosystems of tools that allows automatizing some tasks, such as automatic testing or code generation for prototyping purposes.

	Resources	HTTP headers	Authentication	Documentation gen.	Tool support
WSDL	✓		✓		
WADL	✓	✓	✓		✓
OAI	✓	✓	✓	✓	✓
RAML	✓	✓	✓	✓	✓
IO Docs	✓	✓	✓	✓	✓
API Blueprint	✓	✓	✓	✓	✓

Table 1. Modeling capabilities of analyzed proposals.

As shown in Table 1, all the most recent proposals fully support the modeling of functional aspects of the APIs. Furthermore, they use to offer a way to generate documentation portals and code automatically. In contrast, languages such as WSDL or WADL do not have advanced tool support that allows documentation generation. Finally, we have ascertained that none of these languages allows modeling non-functional aspects, such as billing cycles or operations limitation.

4 SLA modeling

Throughout this section, we present the analysis of different academic proposals to define and model SLAs. In particular, we try to determine whether these proposals are expressive enough to deal with an API-oriented SLA scenario, as discussed in section 2.

Specifically, we have analyzed 8 languages for SLA modeling, ranging from some initial proposals such as WSLA (2003), the precursor of WS-Agreement, to others which try to improve expressiveness in some contexts, such as SLAC or CSLA (2014), including L-USDL Agreement (2016), which introduces a semantic perspective in SLA modeling area. Finally, Table 2 illustrates the main differences between these proposals.

Web Services Agreement Language (WSLA) [16] is a proposal made in 2003 by IBM. Nowadays it is no longer an active project, but it is considered as the precursor of the WS-Agreement standard. This proposal sought to provide a framework for monitoring and evaluation of SLAs. The language core offers ways to define some QoS terms and the consequences derived from them. However, unlike WS-Agreement, the publication and negotiation of contracts were not supported. An agreement in WSLA is an XML document that consists of: 1) description of the parties involved; 2) applicable parameters and guarantees; 3) metrics definition; 4) service level objectives (SLOs).

SLAng [14] is a language, created in 2003, for agreement modeling proposed by *University College London*. This language intends to associate penalties with QoS properties defined in the agreement. In addition, it allows applying these restrictions dynamically at the time or by some state variable visible by all the parties to the agreement.

Web Services Agreement (WS-Agreement) [1] is a specification presented in 2007 by the *Grid Resource Allocation* and the *Open Grid Forum*. It is based upon XML and a web service protocol to: 1) to promote through *agreement templates* an accepted set of *agreement offers* from the service responder; 2) generate a proposal of *agreement offer* from an *agreement template* of the service initiator; 3) negotiate the specific limitations at each step; 4) create an *agreement* between both the service provider and consumer with the conditions and limitations established and allow the observation of compliance. One of the main advantages of WS-Agreement is the possibility of being extended with specific domain languages. Nevertheless, it does

not support the definition of ontologies, so metrics can not be precisely defined. However, some authors have proposed semantic extensions [18] to this specification.

SLA* [11] is part of the European project *SLA@SOI*, concluded in 2011, which sought to increase the automation and predictability of all phases of the agreement life cycle. The syntax is domain independent and describes *agreement templates* documents and *agreement* documents through a generalization of other specifications such as WS-Agreement, WSLA or WSDL. It is not especially focused on web services but allows it to be extended with the requirements of each domain. As in other proposals, an *agreement template* contains five sections: 1) template attributes; 2) the parties involved in the agreement; 3) service description; 4) variable declarations; 5) terms of the agreement and guarantees.

Service-Level-Agreement for Clouds (SLAC) [26] is a language, proposed in 2014, to define SLA in a cloud environment. Since it is WS-Agreement-inspired, it shares many characteristics, structure and definitions. This language offers multiple predefined metrics, inspired by cloud computing, e.g. better multi-party support, group definitions, and parties involved for each term. Unlike other agreement models, SLAC does not differentiate between service description terms and QoS requirements.

Cloud Service Level Agreement (CSLA) [13] is a language, created in 2014, that seeks to express in greater detail the violations in SLAs in a cloud context. This proposal has been influenced by other related works like WSLA [16] and SLA@SOI [11]. This language is able to model the agreement between a consumer and a cloud provider, i.e. quality of service (QoS) or elasticity management based on service level objectives (SLOs). In addition, they also allow describing the usual concepts in WS-Agreement as the validity, the parts of the contract, the definition of the service, etc.

rSLA [23] is a language, created in 2015, for specifying, monitoring and enforcing SLAs for cloud services. The language describes basic metrics are to be obtained and how they are aggregated in terms of composite metrics. It is possible to define how to proceed if SLOs are met or violated.

Linked-USDL Agreement [9] is an extension to the Linked-USDL [20] proposed in 2016. It can capture agreement terms, business aspects, liability, compensations, and time constraints. Specifically, Linked-USDL Agreement is designed to be used to establish and share agreements among customers and providers that seek to perform automated service trading in a web context.

Table 1 depicts a comparative analysis considering these 8 proposals. Each column corresponds to a certain analyzed feature, so that a ✓in cell $C_{i,j}$ means the proposal i has the property j.

1. **Metrics**. Ability to model measuring point counters for the SLA items.
2. **Guarantees**. Capability to model a set of Service Level Objectives (SLO) that should be enforced by one party, i.e. the guarantor, to another party, i.e. the beneficiary.

3. **Compensations**. Representation of SLAs a set of compensations that represent the consequences of under-fulfilling (penalties) or over-fulfilling (rewards) the SLOs.
4. **Billing**. Support to model billing models (e.g. pay-as-you-go, flat rates or up-fronts), including billing cycles and periods.
5. **Scopes**. Ability to model and interpreter the particular element of the agreement that the scoped element applies to.
6. **Semantic approach**. Representation of the SLA concepts using semantic languages (e.g. RDF) which leverage the queries over the model to perform a question answering over SLA documents and some analysis operations.

	Metrics	Guarantees	Compensations	Billing	Scopes	Semantic apro.
WSLA	✓	✓				
SLAng	✓	✓	✓			
WS-Agreement	✓	✓	✓		*	
SLA*	✓	✓	✓			
SLAC	✓	✓	✓	✓		
CSLA	✓	✓	✓	✓	✓	
rSLA	✓	✓	✓			
L-USDL Agreement	✓	✓	✓	✓		✓

Table 2. Modeling capabilities of analyzed proposals.

As depicted in Table 2, most of the proposals cover metrics, guarantees and compensation modeling, since they are the basis of an SLA. Moreover, recent languages, such as SLAC and CSLA, tend to cover some cloud computing particularities: they offer some default metrics and make easier the definition of SLO in a cloud context, specifically in IaaS offerings. The aim of WS-Agreement scopes is just to describe what service element specifically a guarantee term applies to, hence the asterisk mark. Otherwise, CLSA try to generalize the scope usage. Finally, there exists a semantic approach to model SLAs in order to address analytics operations over the SLA. Nevertheless, none of these proposals are able to utterly model the non-functional properties shown in Section 1.

5 Conclusions

In this paper, we have studied 27 RESTful APIs of SaaS offerings and we have shown that a number of the studied APIs often define some properties such as quota/rate limitation or complex billing models. Despite of that, the current modeling languages for APIs do not take into account these non-functional properties. Furthermore, even though these properties can be contained in an SLA,

current modeling languages do not provide a full expressiveness to represent them.

This work is closely aligned to the idea of SLA-Driven API Gateways that we proposed in [8] since it is needed to have a fully expressive SLA modeling language, which can fit in the real world *aaS offerings, in order to open the door to automatic and domain-independent governance tools for RESTful APIs.

In the industry, there exists some examples of more complex SLA models and complex scopes: for instance, Amazon AWS offers multiple billing models (e.g. on-demand, reserved and spot instances). In a similar way, it is possible to find APIs which describe a metric (e.g. number of requests/min) but define multiple limit levels according to whom it may be applied to (e.g. there could be a 1000 requests/min limit for the whole organization, but a 100 requests/min for each organization user). Consequently, as future work, we plan to develop new SLA models that address the lack of expressiveness in other proposals, specifically, i) quotas and rates limitation definition; ii) support to multiple billing cycles, taking into account the derived compensations of the over-fulfillment or the under-fulfillment; iii) definition of complex scopes.

6 Acknowledgments

The authors would like to thank all the students of *Service Oriented Computation* 2015/2016 course for taking the survey, whose results have been the basis of the presented RESTful API analysis.

References

1. Alain Andrieux, Karl Czajkowski, Kate Keahey, A. Dan, Kate Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Open Grid Forum, 2004.
2. API Blueprint. API Blueprint. `https://apiblueprint.org/`, 2016.
3. E Christensen, F Curbera, G Meredith, and S Weerawarana. Web Services Description Language (WSDL). Technical Report 2008-01-07, 2001.
4. Yucong Duan. A survey on service contract. In *Proceedings - 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, SNPD 2012*, pages 805–810. IEEE, aug 2012.
5. Martin Flower. Microservices. `http://martinfowler.com/articles/microservices.html`, 2014.
6. Forrester. API Management Solutions , Q3 2014. Technical report, 2015.
7. Francis Galiegue, Kris Zyp, and Gary Court. JSON Schema. `https://tools.ietf.org/html/draft-zyp-json-schema-04`, 2013.
8. Antonio Gámez-Díaz, Pablo Fernández-Montes, and Antonio Ruiz-Cortés. Towards SLA-Driven API Gateways. In *Actas de las XI Jornadas de Ingeniería de Ciencia e Ingeniería de Servicios*, 2015.
9. J. M. Garcia, P. Fernandez, C. Pedrinaci, M. Resinas, J. Cardoso, and A. Ruiz-Cortes. Modeling Service Level Agreements with Linked USDL Agreement. *IEEE Transactions on Services Computing*, PP(99):1–1, jan 2016.

10. Marc J Hadley. Web Application Description Language (WADL). Technical Report TR-2006-153, 2006.
11. Keven T. Kearney, Francesco Torelli, and Constantinos Kotsokalis. SLA*: An Abstract Syntax for Service Level Agreements. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 217–224. IEEE, oct 2010.
12. Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.
13. Yousri Kouki, Frederico Alvares De Oliveira, Simon Dupont, and Thomas Ledoux. A language support for cloud elasticity management. In *Proceedings - 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2014*, pages 206–215. IEEE, may 2014.
14. D. D. Lamanna, J. Skene, and W. Emmerich. SLAng: A language for defining service level agreements. In *Proceedings of the IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, volume 2003-January, pages 100–106, 2003.
15. Avraham Leff, James T. Rayfield, and Daniel M. Dias. Service-level agreements and commercial grids. *IEEE Internet Computing*, 7(4):44–50, jul 2003.
16. H. Ludwig, A. Keller, A. Dan, and R. King. A service level agreement language for dynamic electronic services. In *Proceedings - 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, WECWIS 2002*, pages 25–32. IEEE Comput. Soc, 2002.
17. Adil Maarouf, Abderrahim Marzouk, and Abdelkrim Haqiq. A review of SLA specification languages in the cloud computing. *2015 10th International Conference on Intelligent Systems: Theories and Applications, SITA 2015*, pages 1–6, 2015.
18. Nicole Oldham, Kunal Verma, Amit Sheth, and Farshad Hakimpour. Semantic WS-Agreement Partner Selection. In *Proceedings of the 15th international conference on World Wide Web*, pages 697–706, 2006.
19. Ben-Kiki Oren, Clark Evans, and Ingy dot Net. Yaml specification. `http://www.yaml.org/spec/1.2/spec.html`, 2009.
20. Carlos Pedrinaci, Jorge Cardoso, and Torsten Leidig. Linked USDL: A vocabulary for web-scale service trading. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8465 LNCS, pages 68–82. Springer, Cham, 2014.
21. Uri Sarid, Misko Hevery, Ivan Lazarov, Peter Rexer, John Musser, Tony Gullotta, Jaideep Subedar, Kevin Duffey, and Rob Daigneau. RAML Version 1.0. `https://github.com/raml-org/raml-spec`, 2016.
22. Wei Tan, Yushun Fan, Ahmed Ghoneim, M. Anwar Hossain, and Schahram Dustdar. From the Service-Oriented Architecture to the Web API Economy. *IEEE Internet Computing*, 20(4):64–68, jul 2016.
23. Samir Tata, Mohamed Mohamed, Takashi Sakairi, Nagapramod Mandagere, Obinna Anya, and Heiko Ludwig. rSLA : A service level agreement language for cloud services. *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 415–422, jun 2016.
24. The Open API Initiative. OAI. `https://openapis.org/`, 2016.
25. Tibco Mashery. Mashery I/O Docs. `http://www.mashery.com/api/io-docs`, 2013.
26. Rafael Brundo Uriarte, Francesco Tiezzi, and Rocco De Nicola. SLAC: A formal service-level-agreement language for cloud computing. In *Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014*, pages 419–426. IEEE, dec 2014.