ELeCTRA: Induced Usage Limitations Calculation in RESTful APIs

Antonio Gamez-Diaz[1], Pablo Fernandez[1], Cesare Pautasso[2], Ana Ivanchikj[2], and Antonio Ruiz-Cortes[1]

[1] Universidad de Sevilla*,
{agamez2, pablofm, aruiz}@us.es,
[2] Software Institute, Faculty of Informatics, USI Lugano, Switzerland
{cesare.pautasso, ana.ivanchikj}@usi.ch

Abstract. As software architecture design is evolving to microservice paradigms, RESTful APIs become the building blocks of applications. In such a scenario, a growing market of APIs is proliferating and developers face the challenges to take advantage of this reality. For example, third-party APIs typically define different usage limitations depending on the purchased Service Level Agreement (SLA) and, consequently, performing a manual analysis of external APIs and their impact in a microservice architecture is a complex and tedious task. In this demonstration paper, we present ELeCTRA, a tool to automate the analysis of induced usage limitations in an API, derived from its usage of external APIs. This tool takes the structural, conversational and SLA specifications of the API, generates a visual dependency graph and translates the problem into a *constraint satisfaction optimization problem* (CSOP) to obtain the optimal usage limitations.

1 Motivation

In recent years, there has been a clear trend towards the micro-service architectural style where each component (i.e. micro-service) can evolve, scale and get deployed independently. This style increases the flexibility of the system and has been applied in demanding web applications such as eBay, Amazon or Netflix.

From an engineering perspective, a key element of these architectures, with respect to the modeling and implementation of microservices, is the use of the RESTful paradigm. From a business perspective, this microservice architecture tendency represents a recent shift in software engineering towards API-driven building and consumption, fostering a breeding landscape for RESTful API market in which composite service providers face new issues, such as, how to set their usage limitations accordingly to 3rd party providers' usage limitations [6].

Automatic discovery of usage limitations requires the serialization of developers' knowledge regarding the SLA usage limitations based on the API structure

* 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), P12–TIC-1867 (COPAS) and and TIN2014-53986-REDT (RCIS)) the FPU scholarship program, granted by the Spanish Ministry of Education, Culture and Sports (FPU15/02980).

and the implemented RESTful conversation [8]. This automatic discovery becomes crucial when there are frequent changes in the usage limitations of the external services or when the microservice architecture itself experiences changes. In this automatic discovery of usage limitations scenario, a key concept is the *boundary service* that corresponds to an API that has the closest interaction with the end-user by means of actions in the GUI. Specifically, each action in the GUI triggers what we coin as *root operations* (composed by a given path and an HTTP method) that are typically related to the user story. Commonly, as a facade, all the boundary services are deployed behind an API Gateway [7] that enhances the security of the infrastructure.

Given the frequent and multiple ways in which the third party APIs' usage limitations evolve and are likely to keep evolving, when building an application, we need to adapt our customer's expectations regarding the performance of the tool (i.e., how many operations can be generated over time and how long it will take to produce an operation). Therefore, to rapidly react to these changes, there is a need of automating the process to obtain the usage limitations that a provider can offer to its end-users based on certain optimal criteria in a microservice architecture, such as minimizing the requests made to the third-party providers. Specifically, the problem addressed is finding the usage limitations of a root operation induced by the rest of the services (internal and external) composing the microservice architecture. Despite the fact that knowing the usage limitations in the internal services can be useful, calculating the usage limitations in the root operations (i.e., the operations closer to the end users) is more valuable since they help the service provider to set its own usage limitations to end users and to understand the induced service level agreement he can offer to its users.

In this demonstration paper, we propose a tool that, given: i) the external usage limitations (as derived from the purchased SLA) as well as the boundary service structure and root operation conversation; it translates the problem into a *constraint satisfaction optimization problem* (CSOP) to obtain the induced usage limitations for the specified root operation.

2 Using ELeCTRA to calculate the usage limitations

In order to illustrate the problem, let us consider the following example: we have developed a report generation application for the internal evaluation of the researchers in our university. For this purpose, we needed to collect, per researcher, the quality indicators of his/her publications. After considering different bibliometric providers, we opted for the Scopus API [4] as it provides the set of publications as well as the CitesScore$^{\text{TM}}$ index, an impact factor index calculated by Elsevier. Specifically, our application depends upon four Scopus APIs. Per researcher, the following invocations are needed: i) retrieving the simple list of publications; ii) collecting the details regarding each publication; iii) gathering the details concerning each publication's venue (journal or conference). Each of these Scopus APIs present different usage limitations [6], so, depending on the input and the usage limitations of each API operation, the induced usage limitations for our end-users will vary. The question to be addressed is: *what is the maximum amount of researchers per week for which this report can be generated without over-passing the Scopus API usage limitations?*

We use ELeCTRA[1] to answer the question. First, we have to model the structural viewpoint of each Scopus RESTful service using the Open API Specification (OAS) [3], as well as the usage limitations by using the SLA4OAI Specification [5]. Each of these models should be publicly available through an URL. Finally, using the embedded editor in ELeCTRA, the x-conversation model is introduced to define the dependencies between the internal and external services, as well as the parameters needed to calculate automatically the usage limitations. This example is preloaded at ELeCTRA and can be accessed by selecting the *simple example*. ELeCTRA is composed of two different microservices (ELeCTRA-DOT and ELeCTRA-CSOP) and a user interface (ELeCTRA-UI) accessible through a web browser. This UI includes a textual editor so that a user can directly modify the *x-conversation* document and visualize the *OAS* and *SLA4OAI* models. When the user saves the model, two actions are carried out. On the one hand, an invocation to ELeCTRA-DOT, the graphical representation microservice, is performed. The *x-conversation* model is parsed and according to a set of rules, it is converted to a graph using the *dot* [1] notation. A png image is returned back to the UI. On the other hand, a request to ELeCTRA-CSOP, the CSOP model generator and solver microservice, is developed in order to calculate the usage limitations for the boundary operation.

ELeCTRA-DOT first represents all the RESTful requests (i.e., each pair of method and path) as the nodes of the graph by using the structural information present in the OAS model. Then, an edge between two operations is included if they are related in the *x-conversation* model. Next, this edge is labeled with the information about the number of invocations based on the x-conversation mode. Finally, for each operation of each service, the usage limitations information is retrieved by means of the SLA4OAI extension. ELeCTRA-CSOP, on the other hand, is the main microservice in this tool, since it is responsible for transforming the *x-conversation* model to a constraint satisfaction optimization problem in order to obtain the usage limitations for an operation. By using a set of rule constructs, the *x-conversation* is transformed into a set of parameters (e.g., the values of the usage limitations, the number of invocations from an operation to the next one), variables (e.g., the quota of the boundary operation to be maximized) and constraints (e.g., the sum of the requests made to a certain operation should not overpass the quota value for this operation). Specifically, this microservice uses the MiniZinc's [2] syntax, a language designed for specifying constrained optimization and decision problems over integers and real numbers. Once the problem has been successfully translated and validated, ELeCTRA-CSOP invokes the solver through the MiniZinc interface and gets the response back. Then, ELeCTRA-UI receives the information and sends the updated usage plans to ELeCTRA-DOT to complete the remaining usage limitations. Finally, all this information is presented to the end user.

[1] `https://electra.governify.io`

References

1. Graphviz. `https://graphviz.gitlab.io/_pages/doc/info/lang.html`.
2. Minizinc. `http://www.minizinc.org/downloads/doc-latest/minizinc-tute.pdf`.
3. Open API Specification. `https://www.openapis.org`.
4. Scopus API. `https://dev.elsevier.com/api_docs.html`.
5. SLA4OAI Specification. `https://github.com/isa-group/SLA4OAI-Specification`.
6. A. Gamez-Diaz, P. Fernandez, and A. Ruiz-Cortes. An analysis of RESTful APIs offerings in the industry. In *ICSOC*, pages 589–604, 2017.
7. A. Gámez-Díaz, P. Fernández-Montes, and A. Ruiz-Cortés. Towards SLA-Driven API Gateways. In *JCIS*, 2015.
8. A. Ivanchikj, C. Pautasso, and S. Schreier. Visual modeling of RESTful conversations with RESTalk. *Softw. Syst. Model.*, pages 1–21, may 2016.

3 Instructions for the organizers

This software tool is bundled into a v10.6.0 Node.js application, running in a Windows 10 system which should have installed (and added to the user PATH environment variable) Minizinc v2.1.7 and Graphviz v2.38.0.

Nevertheless, for the sake of simplicity and portability, authors have packed the entire app into a Docker Image[2]. Therefore, any x86_64 machine with Docker[3] will be able to run the application by typing: `docker run -p 8080:80 isagroup/governify-electra`.

Furthermore, to avoid any installation issue, the online web application is deployed at: `https://electra.governify.io`.

The demonstration video is available at: `http://youtu.be/axbkDax1N9g`.

[2] `https://hub.docker.com/r/isagroup/governify-electra`

[3] `https://docs.docker.com/install/`