RESTalk Miner: Mining RESTful Conversations, Pattern Discovery and Matching

Ana Ivanchikj, Ilija Gjorgjiev, and Cesare Pautasso

Software Institute, Faculty of Informatics, USI Lugano, Switzerland
ana.ivanchikj,ilija.gjorgjiev,cesare.pautasso@usi.ch

Abstract. REST has become the architectural style of choice for APIs, where clients need to instantiate a potentially lengthy sequence of requests to the server in order to achieve their goal, effectively leading to a RESTful conversation between clients and servers. Mining the logs of such RESTful conversations can facilitate knowledge sharing among API designers regarding design best practices as well as API usage and optimization. In this demo paper, we present the RESTalk Miner, which takes logs from RESTful services as an input and uses RESTalk, a domain specific language, to visualize them. It provides interactive coloring to facilitate graph reading, as well as statistics to compare the relative frequency of conversations performed by different clients. Furthermore, it supports searching for predefined patterns as well as pattern discovery.

Keywords: REST APIs · RESTful Conversations · Mining · Pattern Search · Visualization

1 Introduction

As the number of RESTful services is growing, with over 15'000 publicly available REST APIs [5] in the ProgrammableWeb repository[1] as of 2018, mining their logs can bring to interesting insights regarding how different clients actually use REST APIs. This can help developers detect unexpected usage patterns of their APIs by comparing different clients' conversations, or to pinpoint interactions which are worth optimizing as they are being used by most of the clients. For instance, if there is a sequence of several requests which are frequently followed, the API designer might decide to provide in the first request a direct link of the last request, thus avoiding the clients having to make the intermediary requests. Bugs might also become evident, such as unauthorized access to some resources or frequent error messages after a certain sequence of requests. Mining techniques have been successfully applied in the area of business processes for almost two decades, resulting in process discovery, conformance checking, prediction of delays, process redesign recommendation etc. [2]. Similar to business processes, the use of REST APIs also requires a particular sequence of interactions [3]. In this case they are HTTP request-response interactions between clients and

[1] http://www.programmableweb.com

servers with the goal of retrieving or modifying the state of one or more resources managed by a service provider [14]. We call the set of all possible client-server interactions, aimed at achieving a certain goal, a RESTful conversation [4, 8, 7]. As process mining builds on data mining and process model-driven approaches, mining of RESTful services also requires a model-driven approach to RESTful conversations. To that end, in [9] we have proposed RESTalk, a domain specific language for modeling and visualization of RESTful conversations and we use a simplified version of the same in the RESTalk Miner. Although different mining tools with graph visualization already exist [1, 16, 15], their visualization is not REST domain specific nor do they offer pattern searching or pattern discovery functionalities. Patterns [10] represent a systematic form of knowledge sharing as they establish a common vocabulary to describe recurring RESTful conversations [11] which is becoming increasingly important in the API-driven development [6]. Patterns can be used to pinpoint and discuss API design best practices or the absence of the same.

2 RESTalk Miner

The input to RESTalk Miner[2] is a log file from a given server containing log entries of interactions with different clients, complying to the following format:

$$\overbrace{DD/MM/YYYY}^{\text{Date}}\ \overbrace{HH:MM:SS}^{\text{Time}}\ \overbrace{3.171.112.202}^{\text{Client IP Address}}\ \overbrace{POST}^{\text{Method}}\ \overbrace{/job}^{\text{URI}}\ \overbrace{202}^{\text{Status Code}}$$

Additionally there is an optional input, i.e., a file which contains the URI templates derived from an Open API specification of the API. For instance, to abstract the following URI $/\mathrm{content}/\mathrm{serial}/\mathrm{title}/\mathrm{issn}/03029743$ this URI template can be used $/\mathrm{content}/\mathrm{serial}/\mathrm{title}/\mathrm{issn}/{:}\mathrm{id}$. Such abstraction ensures that identical method calls to the same type of resource are visualized as one request. The main default output of the RESTalk Miner is a simplified RESTalk graph showing all the conversations different clients have initiated with the server. Alternatively, the user can select to visualize only the conversations of clients of interest. Most of the nodes in the graph take the form of a juxtaposed request-response containing information about the HTTP method, URI, response status code and the number of log entries in which this request/response pair has appeared. If different log entries indicate that the server has used different responses to the same request, the request and the responses are represented as separate nodes with an exclusive gateway node in-between to emphasize the existence of alternative responses. An exclusive gateway node is also used to show alternative paths that clients have taken during their interactions with the server. A node with a round form is used to mark the start and the end of a conversation of a particular client, while edges depict the sequence flow between nodes. Based on user's preference, the graph can be flattened by abstracting from the URI

[2] https://github.com/USI-INF-Software/RESTfulConversationMining

information and showing only the methods that have been called and the response status codes. For the RESTalk visualization, dagre-d3 library [12, 13] has been used to render the internal data structure into an SVG DOM tree which is displayed by the Web browser.

RESTalk Graph and Comparative Statistics Visualization Once the above mentioned graph has been generated, the user can activate or deactivate different interactive visualizations: **node frequency coloring** which colors nodes from red to yellow depending on the number of log entries that contain the particular request/response pair; **edge frequency thickness** which adjusts the thickness of the edges based on how many clients follow the same path; **edge delay coloring** which colors the edges from red to yellow depending on the time difference between the nodes that the edge connects; **edge probability** which shows a probability of an alternative path being taken after an exclusive gateway; **status coloring** which colors responses based on their status codes; **conversation path coloring** which colors in a unique color all the requests made by the same client and in a mix of colors the nodes which are shared between clients in case multiple clients are selected. The tool also provides the user with pie chart visualization of statistical data regarding the analyzed clients of the RESTful service. The **number of nodes** pie chart shows how many request/response nodes belong to each individual client as a percentage of the total number of nodes, i.e., how lengthy each conversation is; the **uniqueness of nodes** pie chart shows how many nodes are unique to just one client, how many are shared between two, three clients etc. Clicking on a certain slice of the pie colors in the same color in the graph the nodes it refers to; the **shared nodes** pie chart shows the number of nodes shared between specific clients; while the **dynamic sharing** pie chart uses the same computation as the shared nodes pie charts, but only for the clients selected by the user.

Pattern Discovery, Matching and Visualization RESTalk Miner supports two types of pattern searches. Searching for unknown patterns, i.e., pattern discovery, and searching for known patterns, i.e., pattern matching. The pattern discovery can help identify new API design approaches and best practices, while the pattern matching can allow to search for patterns of interest. When **searching for unknown patterns** the user specifies the number of request/response nodes the pattern should contain and the minimal number of clients that must have used that pattern. If patterns that match these criteria are identified, they appear in a dropdown list and the user can decide to visualize them and/or save them. Saved patterns can be used later as **known patterns** to be searched for in other conversations. The user can also upload patterns she knows based on her experience or best practices and search for them in the given conversation. Such patterns need to be described in JSON with a log object describing the conversation pattern to be matched. Each log entry has the same structure as the logs described above, with the difference that any of the elements (Method, Status Code, URI, etc.) can be substituted by a * symbol, meaning that any value of that element will be considered a match when searching for the pattern.

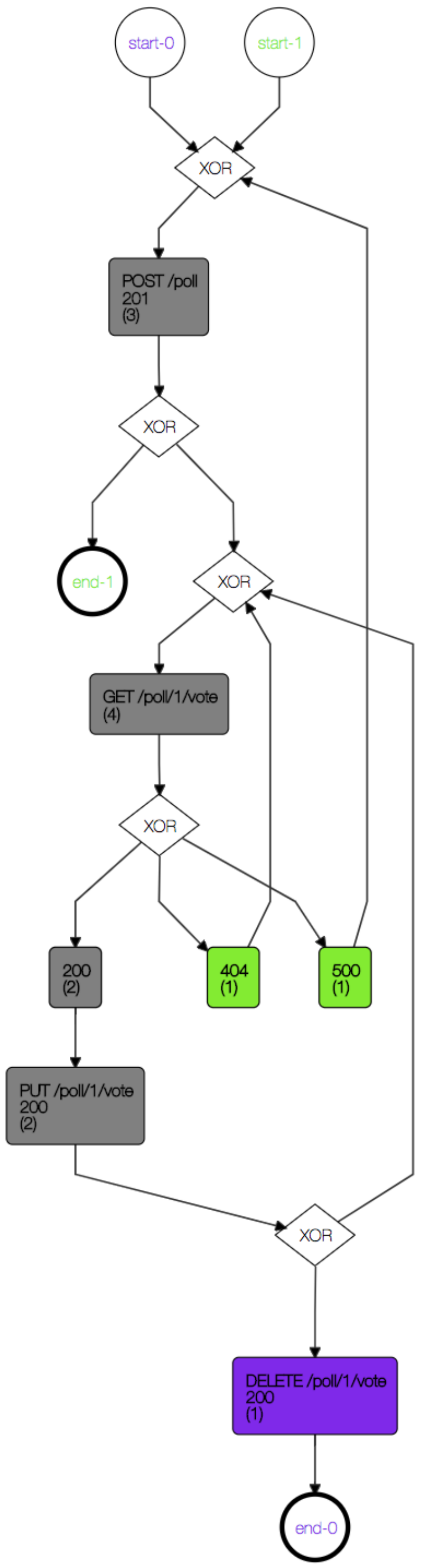
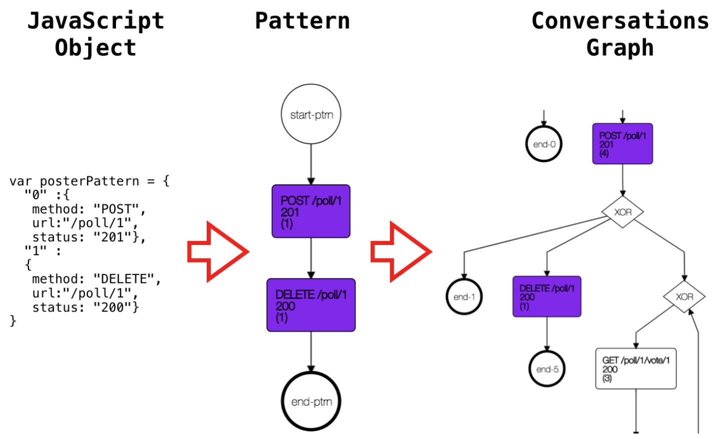


Fig. 1. Overlapping vs. unique parts of conversations



```
var posterPattern = {
  "0" :{
   method: "POST",
   url:"/poll/1",
   status: "201"},
  "1" :
  {
   method: "DELETE",
   url:"/poll/1",
   status: "200"}
}
```

Fig. 2. Pattern matching

URI values can also be used as placeholders, i.e., ensuring that the same URI is used in different requests without precisely specifying the URI value. An optional separator element in the pattern description allows for log entries not to be direct successors. For instance, if we are searching for a pattern with two log entries (POST /example/1, DELETE /example/1), if we use this separator element in the pattern description (POST /example/1 ... DELETE /example/1), an occurrence of POST /example/1 followed by PUT /example/1 followed by DELETE /example/1 will also be considered a match. Such description of the pattern we are searching for allows for greater expressiveness to match targeted patterns.

3 Conclusion

A screencast of the main functionalities of RESTalk Miner is available on YouTube[3]. Future work includes providing full support of the RESTalk constructs, such as the hyperlink flow, which will require additional input collected in the logs. We also plan to release the tool as a Web Application with user registration functionality so that the user can save the mined RESTful conversations and the discovered patterns.

References

1. Disco. https://fluxicon.com/disco/, Last accessed: 2018-08-20
2. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)

[3] https://youtu.be/N94clNa5Mlg

3. Van der Aalst, W.M., Song, M.: Mining social networks: Uncovering interaction patterns in business processes. In: Proc. of BPM. pp. 244–260. Springer (2004)
4. Benatallah, B., Casati, F., et al.: Web service conversation modeling: A cornerstone for e-business automation. Internet Computing, IEEE **8**(1), 46–54 (2004)
5. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
6. Goteti, H.: API Driven Development, Bridging the gap between Providers and Consumers. Tech. rep., CA Technologies (2015), http://rewrite.ca.com/us/articles/application-economy/apis-bridging-the-gap-between-providers-and-consumers.html
7. Haupt, F., Leymann, F., Pautasso, C.: A conversation based approach for modeling REST APIs. In: Proc. of the 12th WICSA 2015. Montreal, Canada (May 2015)
8. Hohpe, G.: Let's have a conversation. Internet Computing, IEEE **11**(3), 78–81 (2007)
9. Ivanchikj, A., Pautasso, C., Schreier, S.: Visual modeling of RESTful conversations with RESTalk. Software & Systems Modeling **17**(3), 1031–1051 (2018)
10. Meszaros, G., Doble, J.: A pattern language for pattern writing. Pattern languages of program design **3**, 529–574 (1998)
11. Pautasso, C., Ivanchikj, A., Schreier, S.: A pattern language for RESTful conversations. In: Proc.EuroPLoP. p. 4. ACM (2016)
12. Pettitt, C.: Directed graph layout for javascript. https://github.com/dagrejs/dagre (2012-2014)
13. Pettitt, C.: A d3-based renderer for dagre. https://github.com/dagrejs/dagre-d3 (2013)
14. Richardson, L., Amundsen, M., Ruby, S.: RESTful Web APIs. O'Reilly (2013)
15. Stroinski, A., et al.: RESTful web service mining: Simple algorithm supporting resource-oriented systems. In: Proc. of ICWE. pp. 694–695. IEEE (2014)
16. Verbeek, H., Buijs, J., Van Dongen, B., van der Aalst, W.M.: Prom 6: The process mining toolkit. Proc. of BPM Demonstration Track **615**, 34–39 (2010)