SpyREST: Automated RESTful API Documentation using HTTP Proxy Server

S M Sohan, Craig Anslow, and Frank Maurer
Department of Computer Science
University of Calgary
Calgary, Alberta T2N 1N4
Email: {smsohan, craig.anslow, frank.maurer}@ucalgary.ca

***Abstract*—RESTful API documentation is expensive to produce and maintain due to the lack of reusable tools and automated solutions. Most RESTful APIs are documented manually and the API developers are responsible for keeping documentation up to date as the API evolves making the process both costly and error-prone. In this paper we introduce a novel technique using an HTTP proxy server that can be used to automatically generate RESTful API documentation and demonstrate SpyREST, an example implementation of the proposed technique. SpyREST uses a proxy to intercept example API calls and intelligently produces API documentation for RESTful Web APIs by processing the request and response data. Using the proposed HTTP proxy server based technique, RESTful API developers can significantly reduce the cost of producing and maintaining API documentation by replacing a large manual process with an automated process.**

***Keywords*—RESTful API, Web API, Documentation, Automation, Example based documentation**

I. Introduction

RESTful APIs, introduced by Fielding, are used as a primary interconnection mechanism among modern day web based systems [1]. For example, the website of a restaurant can use the RESTful API from Twitter to show the latest tweets mentioning the restaurant so that prospective customers can read the experience shared by others. To allow others to use their APIs, Twitter and other RESTful API developers publish documentation describing different features of their RESTful API. The documentation of such RESTful APIs are commonly produced and maintained using a manual process that is expensive.

API documentation for library APIs, such as Java Standard Edition, commonly leverage reusable tools such as JavaDoc. The documentation produced by such tools include description of objects and methods, with custom texts primarily sourced via comments in the source code. On the other hand, RESTful APIs documentation includes additional information such as HTTP headers, request parameters, request and response data in serialized formats such as JSON, XML, etc. that cannot be easily derived from looking at the objects and methods in the source code. Using comments for these additional information also requires significant manual effort. There is a lack of reusable tools to automate the documentation process. This makes the task of RESTful API documentation a costly and error-prone one. In addition to producing the API documentation, API developers also need to publish and often maintain the documentation for multiple versions as the RESTful API evolves. This requires further manual effort because it becomes a continuous process.

To produce RESTful API documentation with information about HTTP headers, URL parameters, request and response data, the manual process is driven by example API calls. This process can be described as a six-step process as follows: 1) craft an example call to an API endpoint with required headers, URL parameters and request body, 2) make the call, 3) capture the response headers and data, 4) strip any confidential data from the captured information, 5) add custom descriptions to the captured data and 6) publish the API documentation. This six-step process is essentially repeated for all API endpoints that are documented. With *SpyREST*, we have implemented an innovative solution to largely automate the aforementioned manual process of RESTful API documentation so that all but steps 1 and 5 are automated. Steps 1 and 5 are left to a manual process to allow for a pragmatic solution that leverages computers to automate the repeated part of the process while leaving the rest to humans. Our solution relies on a lightweight HTTP proxy server to intercept example API calls so that all the HTTP traffic for an API call can be automatically gathered. The collected data is then processed to present the documentation for RESTful APIs. Sensitive data such as authentication tokens and passwords are automatically filtered and are not captured. SpyREST can generate documentation for all RESTful APIs irrespective of the technology used to implement the API since it leverages an HTTP proxy server.

The key contributions of our research are as follows: we discuss a list of requirements for tool development by analyzing related work to automate RESTful API documentation process and present a new technique based on an HTTP proxy server to meet the requirements. We demonstrate an example implementation of the proposed technique that generates automated yet customizable, version-aware, collaboration enabled and reusable API documentation software as a service platform that can be used to generate and maintain documentation for any RESTful API. We present a case study of using SpyREST to compare the advantages of our proposed technique over existing solutions for RESTful API documentation.

The remainder of this paper is organized as follows: in the next section we discuss the related work on API documentation. The requirements and implementation details of SpyREST is provided next. Then, we present a case study to demonstrate SpyREST in action. Then, we discuss the implications and limitations of our work in the discussion section.

II. Related Work

A. General API Documentation

Several papers have studied the documentation of APIs to understand and recommend best practices that are also applicable to RESTful API documentation. Robillard et al. discussed the obstacles that make APIs hard to learn by surveying API developers and users [2] [3]. They found that developers faced severe obstacles learning new APIs due to inappropriate documentation and other learning resources. Robillard recommended the following for API documentation: include good examples, be complete, support many complex usage scenarios, be conveniently organized, and include relevant design elements. Kuhn et al. discussed the importance of examples in API documentation as a key recommendation based on a survey of software developers using APIs [4]. In addition to examples, they identified trustworthiness, confidentiality, and limiting information overload as other key recommendations for API documentation. Hoffman et al. recommended making the API example scenarios to be executable test cases so that a user can execute an API and understand the related business rules [5]. We recognize all these recommendations to be equally important for RESTful APIs. SpyREST automatically generates RESTful API documents from example API calls, with information about the publisher and allows the users to execute the examples following the aforementioned recommendations.

Nasehi et al. performed a case study based on StackOverflow discussions to find out what makes good code examples [6]. They recommended API developers to include a comprehensive set of code examples in the API documentation and the use of wiki-like collaborative tools with online API documentation so that users can ask questions and get answers on officially published API documentation. Parnin et al. found that social media holds a key place in software documentation as it provides additional knowledge about APIs and gives readers a chance to engage with authors of the APIs [7]. They found active collaboration about API related questions resulted into 81% of posts receiving at least one comment with a median of 8 comments per question. Chen et al. recommended integrating crowdsourced frequently asked questions (FAQs) into API documents so that users can easily find relevant discussions when questions arise as they are browsing API documentation [8]. They presented a tool that can embed FAQs into API documents based on a user's browsing behavior. Subramanian et al. presented an automated approach to link API documentation of different Java and JavaScript libraries with code examples that are shared on StackOverflow by the API users so that the collaborative crowd documentation of APIs can be linked with official documentations [9].

Stepalina discussed the advantages of software as a service (SaaS) based solutions for API documentation systems where a web platform can be used to publish many different API documentations [10]. They identified several benefits of a SaaS platform, such as, cost effective yet powerful, platform agnostic and high accessibility, improved document quality, content re-use, automated tools, and organization of robust and scalable documentation process. SpyREST is a SaaS based tool that can be used to leverage these benefits as it allows the generation and publishing of RESTful API documentations for many different APIs under a single platform. They also identified security and reliability of such a shared platform as potentially open issues. To overcome these issues, SpyREST allows a self-hosted alternative to the SaaS model where a SaaS platform is not acceptable for RESTful API documentation.

Several tools exist that help automatic generation of API documentation for local APIs such as JavaDoc[1], RDoc[2], that convert formatted comments from source code into corresponding HTML documentation for the classes and methods. Jadeite is a tool that adds placeholder API objects for library APIs to improve the search and discovery of desired API objects [11]. While these tools have been proven to work for local library APIs, they have limited applicability for documenting RESTful APIs because HTTP specific information such as request and response headers, URL, request and response payloads are not natively supported by these tools. Hence, there exists a gap for tool support to automatically generate RESTful API documentation. SpyREST fills this gap by automatically generating RESTful API documentation from example API calls.

B. RESTful API Documentation

To address the need for a standard format to describe RESTful APIs, several related work proposed candidate specifications. Maleshkova et al. analyzed multiple Web APIs to identify the different approaches related to Web API descriptions, data formats, protocols, reusability, granularity, and authentication [12]. They found that a lack of a standard format to document Web APIs and manual documentation led to API under-specification causing confusions about how to use the APIs for different use cases. They identified a need for automated approaches. Espinha et al. observed that most RESTful APIs are documented manually by API developers making the documentation a less reliable one [13]. To standardize the terminology for RESTful API documentation, Danielsen et al. presented a vocabulary for documenting RESTful Web APIs called Web Interface Language (WIfL) [14]. The vocabulary comprises of these following objects: Resource, Request, Response, Representation and Parameters that can describe the structure and example usage of RESTful Web APIs. Generation of WIfL specific objects for RESTful APIs requires manual effort or bespoke implementation since a reusable automated approach is not presented. With SpyREST, we focus on automation so that reusable tools can be used to produce RESTful API documentation.

Verborgh et at. presented RESTdesc, a Resource oriented and Hyper-link based specification for describing RESTful APIs [15]. RESTdesc relies on pre and postconditions to describe the outcome of an API call so that a general purpose API client can parse RESTdesc formatted documentation to invoke API calls to specific RESTful APIs. Mangler et al. presented RDDL, an XML based specification for describing RESTful APIs [16]. RDDL descriptions are composed of resources, operations, parameters, and headers. Similar to WIfl, manual effort is needed to generate RESTdesc and RDDL formatted documentations.

[1] http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

[2] http://rdoc.sourceforge.net/

Kopecky et al. presented hRESTS, a machine readable micro-format to describe RESTful APIs that use an alternate representation compared to WIfL [17]. hRESTS specification comprises of Service, Operation, Address, Method, Input, Output and Label objects to describe RESTful APIs. Automated XML transformation is used to convert a formatted HTML documentation of a RESTful API into hRESTS so that the machine readable format can be used by RESTful API clients for automated invocation. Ning et al. presented OmniVoke, a RESTful API based invocation engine that provides an abstraction layer for RESTful API calls to multiple APIs that follow different conventions [18]. A general purpose RESTful API client can be used against OmniVoke since the different conventions are wrapped under a uniform RESTful API. Manual configuration is required for each existing API to wrap under OmniVoke. SpyREST relies on example API calls and does not require APIs to be described according to a specification format. Instead, the data recorded by SpyREST can be used to automatically produce API descriptions following a specification as an output that otherwise requires manual effort.

Myers et al. performed a user study based on the documentation of Web APIs used in large scale enterprises to understand the required contents for Web API documentation [19]. They recommended providing a consistent look-and-feel with explanation for the starting points and an overall map comprising of both text and diagrams, providing a browsing experience with breadcrumb trail following a hierarchy, an effective search interface, providing example code and a way to exercise the examples online without writing code. SpyREST aims to achieve requirements for RESTful API documentation using an automated yet customizable approach that can be reused across different RESTful APIs.

In addition to the research community, there are several RESTful API specification formats that are observed in the industry. Swagger is a JSON based specification that allows users to describe RESTful APIs in terms of paths, parameters, request and response headers and bodies[3]. RAML is a RESTful API documentation format that uses YAML files to describe RESTful API documentation using a similar hierarchy as Swagger[4]. Blueprint is another RESTful API specification format that uses Markdown files with additional tags to describe RESTful API objects[5]. In addition to producing RESTful API documentation, there are tools and software as a service (SaaS) providers that can be used to publish the documentation and auto generate API client code for RESTful APIs that are described using one of these formats. To use these tools to generate HTML based RESTful API documentation, API developers need to manually construct the intermediate documentation format such as Swagger, Blueprint, RAML, etc. that is required by these platforms since there is no automated tool to produce this. In contrast, SpyREST generates RESTful API documentation by synthesizing data collected from example API calls without relying on any intermediate representation.

[3]https://github.com/swagger-api/swagger-spec/blob/master/versions/2.0.md

[4]http://raml.org/spec.html

[5]https://github.com/apiaryio/api-blueprint

III. SPYREST

A. Requirements

The following list of requirements for SpyREST, R1-7 in priority order, is derived from analyzing the aforementioned related work and current API documentation practices as observed in the industry:

R1 - Automated RESTful API documentation: RESTful APIs are either documented manually or using custom tools to partially automate the process. Both processes can be expensive to maintain since API documentation is a continuous process to support the evolution of APIs. The primary requirement for SpyREST is to find a cost-effective approach to automate RESTful API documentation.

R2 - Example based: As discussed in the related work section, several authors have emphasized including example scenarios with API documentation can help users understand how to use an API [2] [4] [5] [6]. SpyREST generated API documentation needs to include example scenarios.

R3 - Executable documentation: In addition to including examples, SpyREST also needs to allow users to execute the example scenarios so that they can try the API features without having to write code [5] [19].

R4 - Version awareness: SpyREST needs to allow API developers to publish documentation for multiple versions of a Web API as it evolves. The following comment from an API user of Stripe, an online payment processing company, shows the importance of a version-aware documentation tool[6] :

> "Does the full API documentation only reflect the current version of the API? Is there a way to access the API docs for outdated versions? ...That would be very helpful. When you are trying to upgrade from one version to another it's impossible to know the implementation differences. We are currently about 4 API versions behind and are stuck behind a version that causes a significant amount of work on our end to support. I'd like to be able to upgrade incrementally through each version."

R5 - Customizable: Robillard identified the importance of customized overview information about how to use an API to reduce API learning obstacles [2]. Custom content is also required to explain business rules and overview information about APIs that may not be automatically derivable. SpyREST needs to allow users to add customized content to auto generated RESTful API documentation so that API developers can provide required information about the APIs that are not automatically generated.

R6 - Reusable: SpyREST needs to work as a reusable platform so that multiple REST API documentations can be generated and published on a single platform to get the advantages of a SaaS platform as outlined by Stepalina [10].

R7 - Collaborative: SpyREST needs to allow people to collaborate on the API documentation so that questions and answers about APIs can coexist with the API documentation. This is required to overcome the knowledge fragmentation problem when API related discussions take place on external forums such as StackOverflow.com outside the API documentation portal [8].

[6]https://groups.google.com/a/lists.stripe.com/forum/#!searchin/api-discuss/version/api-discuss/li4PyVcweiw/NT9SFTtF-vQJ

B. Design and Implementation

From our research, we identified an HTTP proxy server based solution as a novel approach for generating RESTful API documentation meeting requirements R1-7. Our approach works as follows: API developers make example API calls using a proxy ⟶ the proxy records and processes HTTP traffic ⟶ a web app generates RESTful API documentation. SpyREST, an example implementation of this approach, is composed of three main components as shown in Fig. 1.

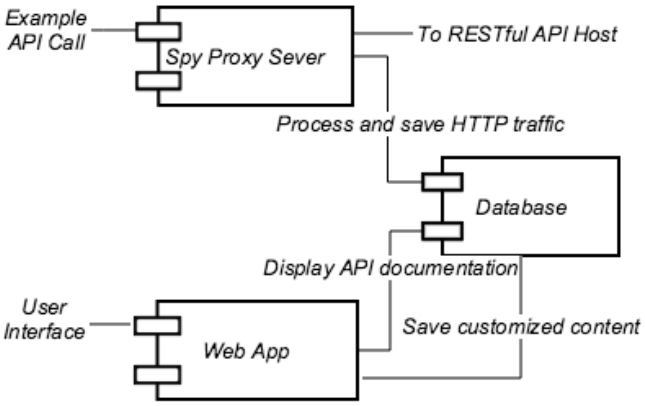


Fig. 1: SpyREST Component Diagram

The Spy: The Spy component in SpyREST is an HTTP proxy server. The Spy is named so because it records the HTTP traffic for example API calls that are made using it as a proxy server. For example, when using Spy as an HTTP proxy to make the following HTTP request:

```
Method  GET
URL     https://api.github.com/repositories?since=100
Headers accept: application/vnd.github.v3+json
```

that produces the following response:

```
Headers status: 200 OK
        content-type: application/json; charset=utf-8
        ...

Body    1 [
        2   {
        3     "id": 1,
        4     "name": "grit",
        5     "full_name": "mojombo/grit",
        6     ...
        7   }
        8 ]
```

Spy automatically saves the raw request and response data with HTTP headers. Additionally, the Spy synthesizes the data and saves the following meta data about this example API call:

version	v3
resource	repositories
action	GET /repositories
query	since: 100
strippedResponseBody	a subset of the response body
description	blank
digest	base64 hash value of the version, resource, url and description
requiresAuth	false
apiToken	blank, used when a SpyREST API token is provided
userId	blank, used when a user is found for given apiToken

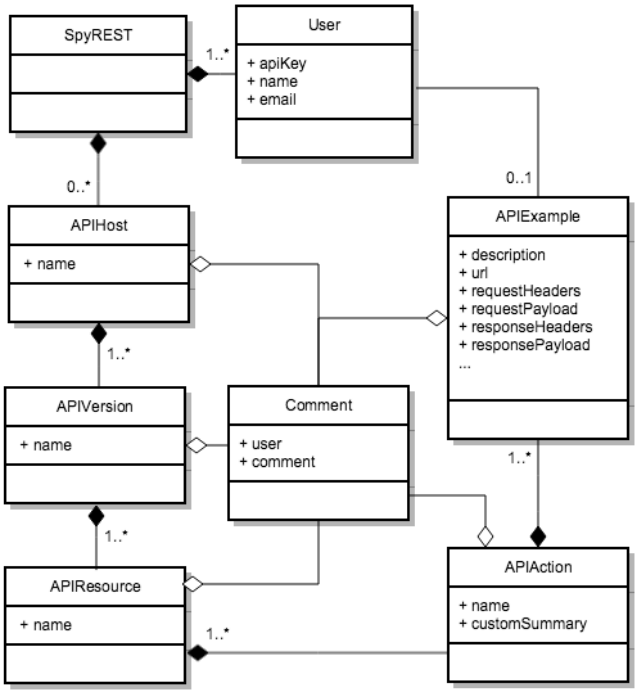
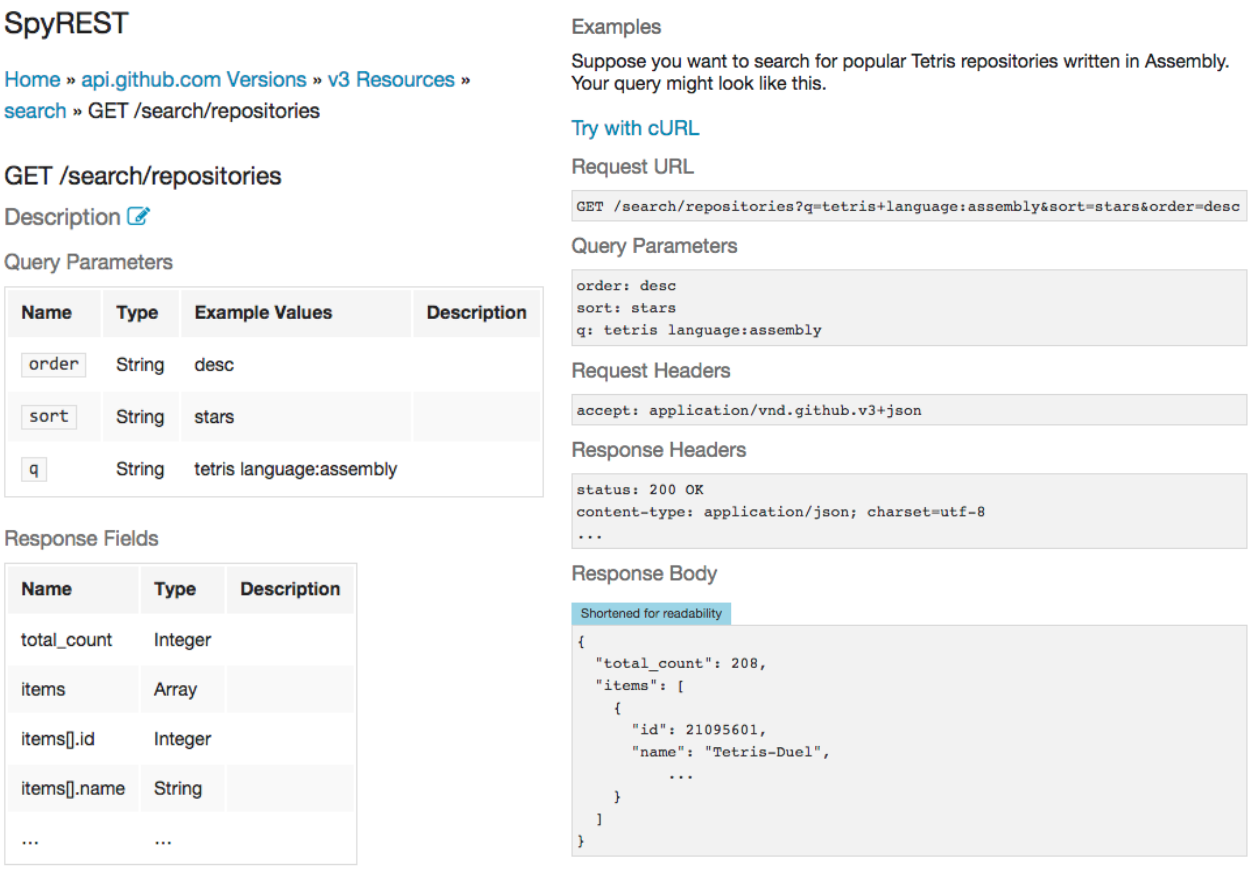


Fig. 2: SpyREST data model

As shown in this example, the version "v3" is automatically detected by parsing the accept request header. Next, the resource field is also auto detected as "repositories" by parsing the request URL. The action "GET /repositories" is automatically detected by combining the request HTTP method with request path. The strippedResponseBody field automatically saves a shorter version of the actual response where large response body is truncated to a smaller one to reduce noise from the generated documentation. These meta fields allow SpyREST to structure the example API calls for a given API host in a hierarchical model as follows: an APIHost (e.g. GitHub.com) has many APIVersions (e.g. v3), each APIVersion has many APIResources (e.g. repositories), each APIResource has many APIActions (e.g. GET /repositories), and each APIAction has many APIExamples. An API developer can sign up for a SpyREST user account to feed data to the Spy. The User object stores the SpyREST user account information. Fig. 2 shows a simplified UML diagram of SpyREST's data model.

To allow the API developers run the example API calls multiple times, the Spy computes a digest based on the version, resource, url and description fields to uniquely identify an API



(a) API documentation summary section

(b) API documentation examples section

Fig. 3: Screenshots of the web app with auto generated API documentation

example. This allows API developers to auto update the documentation when an API changes using the existing examples without creating duplicates. When saving request information, Spy automatically strips off values used in "authorization" request headers to avoid leaking confidential data in the API documentation as recommended by Stepalina [10].

SpyREST allows users to sign up and get an API key to be used for generating API documentation. The "apiToken" and "userId" fields are populated when a valid SpyREST API key is provided in the "x-spy-rest-api-key" request header. This provides credibility to the documentation generated by SpyREST since the official API developers can be identified by their SpyREST account.

To record data for an example call to an API that uses a SSL/TLS connection, the Spy needs to perform as an intentional man-in-the-middle style proxy since the data would be encrypted otherwise. We consider the security risks for this approach to be minimal when the example API calls are intended to be recorded for public viewing through the documentation. For APIs where this is not acceptable, the Spy can be installed with the required SSL certificates on their trusted infrastructure to overcome this issue.

The Database: The database component saves the data that is captured by the Spy and the web app. The Spy saves the recorded data about the API examples and the auto-computed meta data in the database. The web app saves data about SpyREST users and custom modifications on auto-generated API documentation on the same database. As shown in Fig. 2, all the data models excluding the comments are saved in the database component. The comments are saved on Disqus[7], a third-party service that features rich collaboration features for websites. MongoDB[8], a widely used document oriented database, is used to implement the database component

The Web App: The web app component further processes and presents auto generated documentation that is captured by Spy in the database. The web app also allows the API developers to edit or add custom content to the auto generated documentation. Additionally, the web app includes administration features such as user registration and SpyREST API key management. The features of the web app are discussed later in this paper in greater detail.

The source code for both the Spy and web app components are released as open source projects. The specific technologies used by SpyREST work as an example implementation of the core technique of using an HTTP proxy server to auto generate RESTful API documentation.

[7]https://disqus.com
[8]https://www.mongodb.org/

C. Features

Now that we have explained the three components of SpyREST and how they work, we discuss how SpyREST meets the aforementioned requirements R1-7.

R1 - Automated RESTful API documentation: The work flow for automated RESTful API documentation can be explained by the following steps: 1) API developer uses the Spy HTTP proxy to run example API calls, 2) The Spy records example API calls, 3) The Spy extracts meta information about the API call, 4) The Web app displays the auto generated documentation, 5) API developers can optionally customize the auto generated documentation.

Fig 3 shows some screen shots (content and presentation adapted for brevity) of fragments from SpyREST auto generated documentation that is solely based on an example API call to URL https://api.github.com/search/repositories?q=tetris+language:assembly&sort=stars&order=desc with a custom header "x-spy-rest-description" to provide the short description for the example.

To show both the structure and the example usages of the API, the auto generated API documentation features two sections, a summary section as shown in Fig. 3a and an examples section as shown in Fig. 3b. The summary section includes breadcrumb to show hierarchy of the API objects related to each API action to help API client developers easily navigate the API documentation. This section also includes three tables that display the structure of query parameters, and request and response payloads. In addition to the structure, automated type detection is used to show the data type for each field in these tables. Example values are also automatically populated for query parameters. This reduces the need for a manual effort by automating the process based.

The examples section on the API documentation shows all the recorded API examples for a given API action. For each example, it shows a description, the request URL, query parameters, and request and response headers and bodies. Because Spy filters out the "authorization" request header before saving the examples in the database, the documentation rendered by the web app displays this header with a placeholder text as "FILTERED". As discussed before, the Spy also computes a "strippedResponseBody" by truncating large arrays in API responses to contain only two samples per array. The web app displays this "strippedResponseBody" to limit the verbosity on the auto-generated API documentation. Not shown in Fig. 3 for brevity, each example in the documentation also shows the information about the API developer that ran the API example and the time when it was generated.

R2 - Example based: SpyREST generated RESTful API documentation includes both the structure of the API objects and concrete examples for different use cases. The API examples are annotated with user provided descriptions through a custom HTTP request header "x-spy-rest-description" that is otherwise hard to automatically infer. Because the Spy computes a hash digest for uniquely identifying each API example, replays of the same API example only updates the auto-generated documentation with the latest information allowing API developers to update the API documentation automatically when the API changes during development time. To record the examples in SpyREST, any REST API client can be used as long as it supports an HTTP proxy to call the API.

R3 - Executable documentation: SpyREST keeps a copy of the API request information including URL, headers and request body. As a result, SpyREST can also recreate the example API calls that can be executed by users of the generated API documentation. Currently, it automatically generates executable test cases that can be run using cURL[9]. cURL is a command line based tool that can be used as a REST API client. The examples section on Fig. 3b shows a link titled "Try with cURL", on click of which the user gets a panel with an executable API call example reconstructed from the recorded data as shown in Fig. 4.

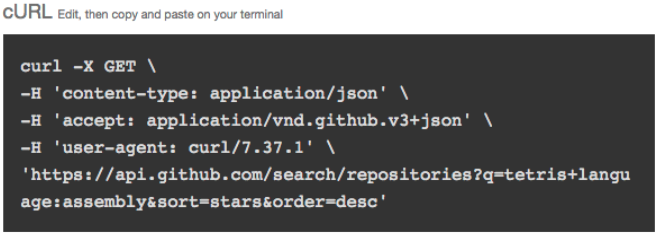


Fig. 4: Executable Examples in SpyREST using cURL

This panel containing the executable API example is editable on the user interface. The editable panel allows the API client developers to modify the API call if they need to customize the auto-generated API call. For example, to change the value of "order" query parameter from "desc" to "asc" as shown in Fig. 4, a user can edit the text in this panel before executing the API example in cURL. This also allows the API client developers to provide values for "authorization" HTTP header that is stripped by the Spy proxy server.

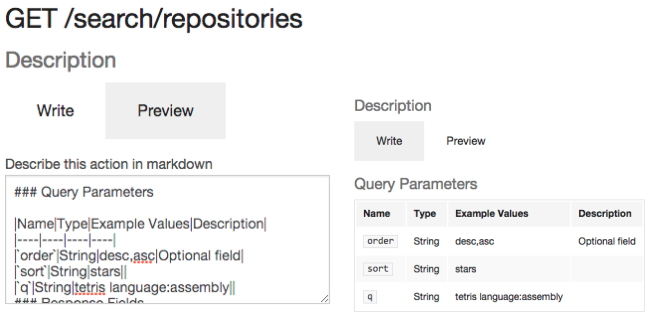
TABLE I: Examples of auto detected versions

Type	Example Value	Detected version
"accept" header	application/vnd.ex.ca.v3+json	v3
"accept" header	application/vnd.ex.ca.v3.1+json	v3.1
URL	/v2/x	v2
URL	/v2.1/x	v2.1
URL	/v2.1-pre/users	v2.1-pre

R4 - Version awareness: SpyREST allows API developers to publish the documentation for multiple versions of their RESTful APIs. To organize API documentations and examples under multiple versions, SpyREST has an automatic version detection algorithm that parses the "accept" request header or the URL. Table I shows some examples of auto-detected versions. To cover the commonly used versioning schemes, the version detection logic tokenizes the "accept" header and URL to find segments that are prefixed with a "v" followed by numbers. In case this heuristic fails to detect a version, the examples are recorded under "Default" version. If required, the auto version detection algorithm can be suppressed by specifying the version in the custom "x-spy-rest-version" header when running example API calls through the Spy in.

R5 - Customizable: SpyREST allows API developers to modify and add custom free-form contents to the auto gener-

[9]http://curl.haxx.se/



(a) Editing auto-generated summary (b) Previewing customized content

Fig. 5: Screen shots showing customization feature

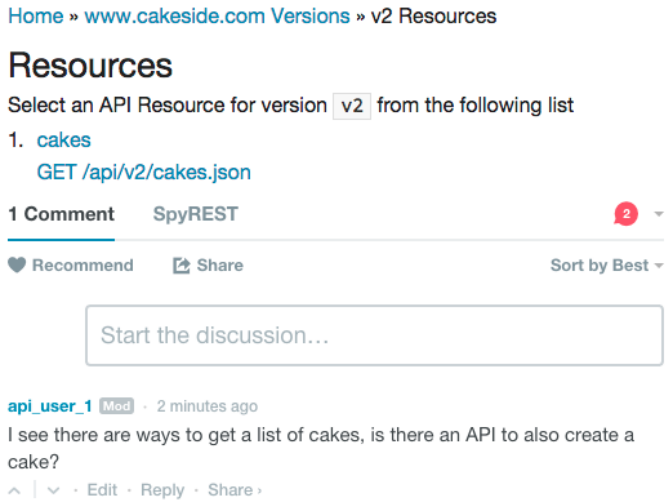


Fig. 6: SpyREST screen shot showing collaboration

ated summary section of the documentation using Markdown[10] syntax. Markdown was chosen because of its popularity as a documentation and code sharing tool among software developers on platforms such as StackOverflow and GitHub. SpyREST solely relies on example calls to APIs to auto-generate the API documentations. Custom content can be used to provide overview information and explain complicated business rules about the APIs that are not derivable from simply synthesizing the examples. We consider this as a pragmatic solution to augment manual effort with the largely automated solution to the RESTful API documentation effort. The summary section on the API documentation page has a button next to the "Description" title that opens up the Markdown editor with preview support as shown in Fig. 5. As shown in this example, a user can annotate the auto-generated documentation for the "order" query parameter as being an "optional field". The custom edits are persisted in the database and are not overridden when the API examples are replayed unless the user decides to revert back to automated summaries.

R6 - Reusable: SpyREST only relies on an HTTP proxy to auto-generate RESTful API documentation. So, it can be used to document RESTful APIs that are developed using any underlying technology. To feed SpyREST with data, any REST API client can be used as long as it supports connecting through an HTTP proxy server. This technology agnostic feature allows SpyREST to be a reusable RESTful API documentation tool. SpyREST is offered as a SaaS tool that at http://spyrest.com that can be used to auto-generate and publish documentation of multiple RESTful APIs. For APIs where a SaaS solution is not acceptable, SpyREST can also be installed and used as a self-hosted solution. Because SpyREST is an open-source application, users can modify the source to support any unique requirements that are not supported by SpyREST.

R7 - Collaborative: SpyREST allows the API developers to comment and discuss API related questions with API client developers right next to the documentation so that crowdsourced documentations can coexist with the officially published API documentation. The current implementation of commenting features on SpyREST is based on disqus[11] as shown in Fig. 6. Each API documentation page on SpyREST features its own discussion thread to help users easily locate relevant information.

IV. SpyREST Case Study

Now that we have discussed the different features offered by SpyREST, in this section we discuss a case study of using SpyREST. For this case study, we used SpyREST to auto generate documentation for 25 API actions randomly sampled from three RESTful API providers using using 272 lines of code[12]: GitHub.com, KISSMetrics.com (Online analytics tool), and LiquidPlanner.com (Online project management tool). We illustrate the process with an example usage of SpyREST to auto generate the documentation of one API action from the GitHub API.

A. GitHub Example

We start by looking at the manually written notifications list API available at https://developer.github.com/v3/activity/notifications/#list-your-notifications. Using this API, a GitHub user can fetch notifications about different activities related to their account. To auto-generate the documentation for this API using SpyREST, we need to make some example API calls via the Spy proxy. To make the example API calls, in this demonstration, we use a REST client using a Ruby program as shown in Listing 1.

To auto-generate the API documentation for notifications API in GitHub, we first setup the base URL and required HTTP request headers on lines 4-6 inside the `Github` class. Next, we setup the proxy connection to go through the Spy on line 12. The `Github` class is defined once and used for all example API calls. Then we provide an example API call describing a use case for the `/notifications` action on lines 15-22 that use the `Github` class. Additional API examples can be provided using code similar to line 17-22. This example is written as an automated test using the

[10]http://en.wikipedia.org/wiki/Markdown
[11]https://disqus.com/
[12]https://github.com/smsohan/spyrest_examples

Listing 1: Example API call using SpyREST

```
1 class Github
2   include HTTParty
3
4   base_uri 'https://api.github.com'
5   headers('Accept' => 'application/vnd.github.v3+
        json',
6     'User-Agent' => 'curl/7.37.1',
7     'content-type' => 'application/json',
8     'Authorization' => "token GITHUB_API_TOKEN'"
9   )
10
11  host, port = 'spyrest.com', 9081
12  http_proxy host, port
13 end
14
15 describe 'Notifications' do
16
17   it 'List all notifications for the current user,
         where they are participating, since a time' do
18
19     response = Github.get '/notifications?all=true&
           participating=true&since=2014-01-01T00:00:00
           Z'
20     assert_equal 200, response.code
21   end
22
23 end
```

Minitest testing framework[13] so that in addition to generating the API documentation, API developers can use the same code as functional tests for the API and vice versa, essentially performing "test driven documentation". With this setup, when the tests are run, it executes the example API calls and SpyREST automatically generates the API documentation for the `GET /notifications` action.

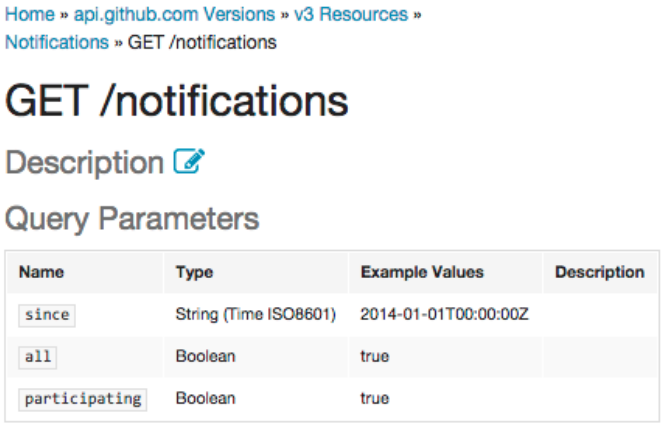


Fig. 7: Screen shot showing URL query parameters

Fig. 7 shows a screen shot of the auto generated summary table for the URL query parameters of `GET /notifications` API action. This table merges the different parameters such as `all`, `participating` and `since`, from the example API call to the `/notifications` API and displays the auto-detected data type such as ISO8601 formatted time, as well as example values that are captured.

[13]https://github.com/seattlerb/minitest

Similarly, SpyREST also automatically displays the structure of the API request and response bodies, if any, on separate tables using a similar visualization as shown in Fig. 3. A user can edit this auto generated summary information using a Markdown editor in the UI to provide further details and customize the auto generated summary.

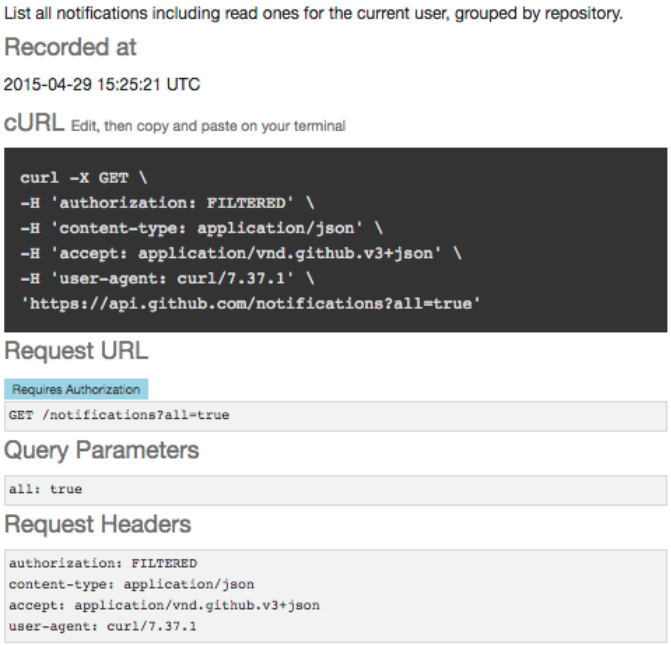


Fig. 8: Screen shot of documentation with an example use

In addition to showing the structure of the API objects, SpyREST displays the actual data collected from the example API on the auto generated documentation. Fig. 8 shows a fragment of the documentation for the example API call generated from line 15 on the test code. On line 8 of our test code, we set the `authorization` header for GitHub. The Spy does not save this secret value and the auto-generated documentation displays `authorization: FILTERED` in the Request Headers section as shown in Fig. 8. SpyREST also automatically includes a "Requires Authorization" message on the documentation because of the presence of this `authorization` header. SpyREST auto generates the cURL command to allow users to execute this example API call without writing any code. As discussed earlier, SpyREST features a commenting section on every API documentation page to allow API client developers to collaborate and find relevant information on the APIs.

In this case study, we have shown how a total of 23 lines of executable test code, has been used to auto-generate the documentation of the `/notifications` action for the Github API. We consider the effort required for writing the test code to be simpler than the effort that'd be required to manually write an API documentation by looking at the content and presentation of the auto generated documentation.

B. *SpyREST Documentation vs. Official Documentation*

We found SpyREST generated documentation for 5 of the 25 API actions from the case study included fields that are

found from actual API responses but not included in their official documentation. For example, the SpyREST generated documentation for Github `GET /notifications` API action included 34 additional fields, such as: `forks_url`, `keys_url`, `collaborators_url`, `and 29 more` that were not mentioned in the official API documentation even though actually returned as API response. Similarly, the official documentation for GitHub.com `GET /search/code` API action did not include the `releases_url` API response field. Official documentation of KISSMetrics.com `GET core/accounts` and `GET core/accounts/:account_id` API actions did not mention the `data` field that is found in the actual response as documented by SpyREST. The official documentation for `GET /api/account` action on LiquidPlanner.com did not mention the fields `workspaces`, `last_workspace_id`, and `disabled_workspaces_count` that were included in the SpyREST documentation. These examples show the error-prone nature of a manual process that requires API developers to ensure the documentation is updated to reflect any change in the API. We consider the proxy based solution to solve this problem since the API documentation can be updated by replaying the example API calls.

The official documentation of these three API providers did not include integrated collaboration features. We found 172 unanswered questions out of 662 questions with the tag `github-api` on StackOverflow.com. We found 123 questions about LiquidPlanner API on their developer forum that are not linked to officially generated documentation published as a PDF file. On StackOverflow.com, we found 4 unanswered questions out of 9 questions on KISSMetrics API. SpyREST provides the collaboration features with auto generated API documentation so that API developers and users can discuss and locate related discussions when browsing RESTful API documentation on a single web interface. SpyREST includes executable API examples, support for multiple versions and presents the auto generated documentation for different RESTful APIs using a consistent look and feel that are not provided by these studied APIs.

V. Discussion

RESTful API documentation is expensive, error-prone, and often incomplete because of the manual effort involved in the process. Our core contribution is a novel approach where an HTTP proxy server is used to largely automate the process for RESTful API documentation. There is a lack of tool support for automatically generating documentation of RESTful APIs. We presented SpyREST as an example implementation of the HTTP proxy server based solution to fill this need. SpyREST demonstrates how HTTP proxies can be used to automatically record and synthesize the recorded data from example API calls to provide a pragmatic solution for automatic RESTful API documentation. In Section IV, we demonstrated how using SpyREST to auto generate documentation from executable code can overcome the problems with a manual process.

The requirements for SpyREST are derived from analyzing the existing literature and industry practices as well as our own experiences developing RESTful APIs and their documentation. In the literature, several papers provided lists of recommendations for API documentation based on studying the existing tools and techniques and feedback collected from API developers. We found a lack of available tool support to generate RESTful API documentation following those recommendations and worked on SpyREST to provide a solution. As a result, SpyREST features such as automated documentation with executable examples, customizable contents, collaboration, version awareness and reusability provide a ready to use tool support for RESTful API documentation following the recommendations.

As demonstrated by SpyREST, the novel approach of using an HTTP proxy sever offers some unique benefits over existing tools from the academia and the industry. For example, tools that rely on user provided comments on source code and code inspection, such as JavaDoc, require the user to write formatted comments that are often applicable to a single programming language. The comments are not executable, and manual effort is required to ensure the comments and the API that it describes are kept in sync. When comments are used for documentation, the generated documentation is a static representation of the comments while SpyREST generated documentation can reflects the latest information since the documentation is generated after executing the example API calls. The verbose nature of RESTful API documentation caused by the different parts such as URL parameters, headers, request and response body, and examples makes it difficult to write and maintain the documentation as comments in source code. Using an HTTP proxy server, SpyREST offers a language agnostic solution so that RESTful APIs written in any programming language can be documented using a single tool.

Another key feature of SpyREST is the shared platform for publishing RESTful API documentation so that API developers do not need to maintain a whole website to publish their API documentation. A similar shared platform is provided by several other tools such as Swagger, Blueprint. We identify the key advantage of SpyREST over these tools to be the fact that SpyREST relies on example API calls instead of relying on a custom API specification that are required by these tools. As discussed in section II, several authors have also published specifications for describing RESTful APIs. The suggested API specifications can be used to define the structure of API objects but do not capture examples of API usage. To generate these API specifications, a manual effort is required since there is no automated tools available to do so. The work flow for using these API specification formats can be described as follows: (1) execute example API calls $\longrightarrow$ (2) manually generate API specification $\longrightarrow$ (3) automatically generate API documentation. SpyREST does not rely on any custom API specification, so the associated manual effort can be largely avoided resulting in a smaller work flow as follows: (1) execute example API calls $\longrightarrow$ (2) automatically generate API documentation. We consider this to be a cost effective approach for APIs with several endpoints or APIs that evolve.

Unlike the commercially available SaaS only solutions such as Swagger, and Blueprint, SpyREST can be used both as a SaaS and a self-hosted platform. The self-hosted mode can be used for documenting internal RESTful APIs that cannot be released on the internet for compliance reasons or modifying the open-sourced code of SpyREST to fit unique API specific needs. We have demonstrated using SpyREST with automated

test code. Using the test code helps automatically testing different use-case scenarios of the API actions in addition to generating the API documentation.

While SpyREST offers an automated solution for documenting RESTful APIs, API developers can customize the auto generated content. Several authors in the literature identified the need for rich content comprising of text and diagrams to explain complex business rules about some APIs. To achieve this goal, we consider augmenting auto generated documentation with user contributed documentation, when necessary, to provide a pragmatic solution.

To summarize, from our example implementation of SpyREST and the case-study, we have shown the advantages of the proposed HTTP proxy server based solution over the existing techniques for RESTful API documentation.

A. Threats to Validity

Internal Threats. Our proposed technique uses an HTTP proxy server that can only intercept and record the data when it is either in clear text or trusted to be decrypted by SpyREST using an intentional man-in-the-middle approach. Because SpyREST can be used as a shared platform, any secrets used in the example API calls get decrypted in memory, even though not saved by SpyREST. To overcome these threat, we recommend API developers to use disposable secrets so that the secrets used to run an API call are valid for a single API call. For use cases where this is unacceptable, the self-hosted mode can be used as an alternative. SpyREST currently only supports JSON based APIs for their widespread use among popular RESTful APIs such as Twitter, Facebook, Google Maps. A similar approach can be used but requires future work to support other formats such as XML, and CSV.

External Threats. We used SpyREST to auto generate documentation for APIs randomly sampled from three RESTful API providers as a proof of concept and found the proxy server based solution met the requirements R1-7. Further evaluation is needed involving a larger set of RESTful APIs to evaluate the effectiveness of the proposed technique of using an HTTP proxy server for RESTful API documentation.

VI. Conclusion

In this paper, we have introduced a novel technique of using an HTTP proxy server to auto generate RESTful API documentation. With SpyREST, we have shown an implementation of the technique and compared its advantages over existing techniques related to RESTful API documentation. The proxy server based solution supports integrated features such as automated RESTful API documentation with executable example API calls, support for multiple versions, customization and collaboration that are offered both as a SaaS and self-hosted platforms. These features have been recommended by existing research on the documentation of APIs. We have demonstrated using SpyREST with automated test code so that RESTful API documentation can be achieved as a positive side-effect of executing automated tests. Overall, based on the outcome of our work on SpyREST, we conclude that a proxy server based approach shows a pragmatic solution to the RESTful API documentation problem. In future, we will perform a qualitative user study involving RESTful API developers to evaluate the effectiveness about the aforementioned HTTP proxy based RESTful API documentation approach.

References

[1] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[2] M. Robillard, "What makes APIs hard to learn? the answers of developers," *Software, IEEE*, vol. PP, no. 99, pp. 1–1, 2011.

[3] M. Robillard and R. DeLine, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[4] A. Kuhn and R. DeLine, "On designing better tools for learning APIs," in *Search-Driven Development - Users, Infrastructure, Tools and Evaluation (SUITE), 2012 ICSE Workshop on*, 2012, pp. 27–30.

[5] D. Hoffman and P. Strooper, "API documentation with executable examples," *Journal of Systems and Software*, vol. 66, no. 2, pp. 143 – 156, 2003.

[6] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," in *IEEE International Conference on Software Maintenance*, 2012, pp. 25–34.

[7] C. Parnin and C. Treude, "Measuring API documentation on the web," in *Proceedings of the International Workshop on Web 2.0 for Software Engineering*, ser. Web2SE '11. ACM, 2011, pp. 25–30.

[8] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced faqs into api documentation," in *Companion Proceedings of the International Conference on Software Engineering*, ser. ICSE Companion 2014. ACM, 2014, pp. 456–459.

[9] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Proceedings of the International Conference on Software Engineering*, ser. ICSE 2014. ACM, 2014, pp. 643–652.

[10] E. Stepalina, "SaaS support in software documentation systems," in *Software Engineering Conference (CEE-SECR), 2010 6th Central and Eastern European*, 2010, pp. 192–197.

[11] J. Stylos, B. A. Myers, and Z. Yang, "Jadeite: Improving API documentation using usage information," in *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '09. ACM, 2009, pp. 4429–4434.

[12] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web APIs on the world wide web," in *Proc. of European Conference on Web Services (ECOWS)*. IEEE, 2010, pp. 107–114.

[13] T. Espinha, A. Zaidman, and H.-G. Gross, "Web API growing pains: Stories from client developers and their code," in *Proc. of Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*. IEEE, 2014, pp. 84–93.

[14] P. Danielsen and A. Jeffrey, "Validation and interactivity of web API documentation," in *International Conference on Web Services*. IEEE, 2013, pp. 523–530.

[15] R. Verborgh, T. Steiner, D. Van Deursen, S. Coppens, J. G. Vallés, and R. Van de Walle, "Functional descriptions as the bridge between hypermedia APIs and the semantic web," in *Proceedings of the Third International Workshop on RESTful Design*, ser. WS-REST '12. ACM, 2012, pp. 33–40.

[16] J. Mangler, P. Beran, and E. Schikuta, "On the origin of services using riddl for description, evolution and composition of restful services," in *Cluster, Cloud and Grid Computing (CCGrid), IEEE/ACM International Conference on*, 2010, pp. 505–508.

[17] J. Kopecky, K. Gomadam, and T. Vitvar, "hrests: An html microformat for describing restful web services," in *Proc. of International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 1. IEEE, 2008, pp. 619–625.

[18] N. Li, C. Pedrinaci, M. Maleshkova, J. Kopecky, and J. Domingue, "Omnivoke: A framework for automating the invocation of web apis," in *IEEE International Conference on Semantic Computing*, 2011, pp. 39–46.

[19] B. A. Myers, S. Y. Jeong, Y. Xie, J. Beaton, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, "Studying the documentation of an api for enterprise service-oriented architecture," *J. Organ. End User Comput.*, vol. 22, no. 1, pp. 23–51, 2010.