Computer Standards & Interfaces 83 (2023) 103657

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Improving open data web API documentation through interactivity and natural language generation

César González-Mora *, Cristina Barros, Irene Garrigós, Jose Zubcoff, Elena Lloret, Jose-Norberto Mazón

Department of Software and Computing Systems, University of Alicante Apdo. de Correos 99 Alicante, E-03080, Spain

ARTICLE INFO

Keywords:
Web API
OpenAPI documentation
Natural language processing
Natural language generation

ABSTRACT

Widely adoption of Information Technologies has resulted in the continuous growing of open data available on the Web. However, the lack of suitable mechanisms to understand open data sources hampers its reusability. One way to overcome this limitation is by means of Web Application Programming Interfaces (APIs) with proper documentation, nowadays being the existing very rudimentary, hard to follow, and sometimes incomplete or even inaccurate in most cases. In order to improve the documentation of Web APIs that access open data, this paper proposes a novel approach to automatically generate interactive Web API documentation, both machine and user readable. This process starts by analysing the documentation of an API to obtain important information, automatically constructing Natural Language descriptions of the main Web API concepts by applying Natural Language Processing (NLP), and specifically, language generation techniques. Then, the documentation is made interactive by making it available as a Web interface, offering easy access to open data provided by Web APIs. Therefore, the use and comprehension of the Web APIs is facilitated, thus promoting the reusability of open data. The feasibility of our approach is presented through a case study and an experiment with users, both showing the benefits of our approach.

1. Introduction

Nowadays, the Web has become an important information platform, thus worldwide governments and organisations have been generating and publishing open data for years [1,2], producing economic and social benefits, such as innovation and transparency. Indeed, there is a broader audience out there that is really interested in data, which are called data enthusiasts [3] (i.e. data consumers and knowledge-seekers), who are non-technical users (but domain experts) and they do not have extensive programming abilities either [4]. Therefore, data enthusiasts need easy access to open data in order to answer domain specific questions [3].

While the existence of large amounts of open data may be regarded as an advantage for these data enthusiasts, it is actually a pitfall because data can become difficult to consume [5]. Open data portals should offer users the necessary mechanisms to facilitate the discovery, extraction and usage of open data [6]. Therefore, removing barriers for data consumption processes remains a primary concern [5]. In order to do so, the most adopted approach to facilitate the access to open data to users are Web Application Programming Interfaces (APIs) [7]. Actually, they are a key feature of open data platforms, allowing data consumers (such as data enthusiasts) to access data in a convenient manner [8–10]. However, data enthusiasts may feel overwhelmed when trying to understand Web APIs and use them to access open data. One of the reasons is that using Web APIs require users to be familiar with query languages and a certain level of knowledge about programming.

The context of our research is aligned with the previous work of Abell et al. [11] and Robillard et al. [12] in the sense that a comprehensive and useful documentation is key to facilitate the wide use of APIs. This documentation will promote the reuse of open data because the value of this data is limited by our ability to interpret and comprehend it [13]. Moreover, the importance of this documentation is even more relevant regarding Web APIs that provide direct data access (query-level Web APIs) in order to facilitate the creation of third-party solutions that reuse this data [14]. Therefore, an important factor to attract users and increase the value of APIs consists of facilitating its use by an accurate, complete and interactive documentation [14,15]. Unfortunately, the available APIs to access open data are generally incomplete and difficult to use because they lack adequate documentation [11,15], so users must

* Corresponding author.
E-mail address: cgmora@ua.es (C. González-Mora).

https://doi.org/10.1016/j.csi.2022.103657
Received 14 June 2021; Received in revised form 5 April 2022; Accepted 11 May 2022
Available online 14 May 2022

dedicate more time and effort to learn how to correctly use these APIs.

In order to overcome the lack of understandable Web API documentation, Natural Language Processing (NLP) research area could help [16] to generate at the same time readable text by users and processable by machines. This NLP area is a computational subfield of Artificial Intelligence whose main purpose relies on the analysis, processing, generation and representation of natural language [17]. Within the areas enclosed in NLP, when considering the automatic generation of documentation descriptions, the area of Natural Language Generation (NLG) is essential. The main aim of this area is to develop techniques capable of generating human utterances, whether the input to these techniques is text or non-linguistic data [18]. Due to the great functionality and adaptability of NLG, our hypothesis is that the use of its techniques can be beneficial in the present research work about generation of API documentation, providing both interactive documentation and natural language descriptions at the same time.

As the main goal of this research is to provide data enthusiasts with comprehensive APIs to better access open data, we propose an approach that integrates NLP and NLG techniques. These techniques help us to generate machine and user readable documentation of open data Web APIs by including natural language descriptions, which are then easy to read and understand by data enthusiasts' users. As far as the authors are concerned, our proposal is novel for the state of the art, since the main efforts of existing related work [14,19–25] are on the generation of documentation for many different purposes in particular scenarios, but they do not generally focus on making the documentation interactive and do not take into account the positive impact of generating automatically natural language descriptions. As new approaches about the generation of API documentation are needed [11] to improve the comprehension of APIs and the reusability of data, we first proposed a preliminary version [26], and now, we propose a next step in the improvement of open data Web APIs documentation.

In order to verify that our approach provides easier to use Web APIs documentation, we perform a complete evaluation with a case study and an experiment with data enthusiasts. Indeed, in this experiment the interactive and natural language documentation generated by our approach is compared to a basic machine-readable documentation of the same APIs, so that data enthusiasts give their opinions about each documentation. With this experiment we aim to confirm that the problem associated to the difficulty of using current Web APIs by data enthusiasts because of their documentation, can be alleviated by the documentation improvement we propose.

Our contributions to this improvement of Web APIs documentation are:

- A novel freestanding approach for tackling the challenge of data enthusiasts to understand data available through Web APIs.
- A complete documentation which in addition to including easy to process specifications for machines, also adds easy to read descriptions for users, facilitating the reuse of data.
- The natural language descriptions included are generated automatically using a NLG template-based approach in conjunction with semantic knowledge, rather than documenting manually each API. It reduces the cost and effort of documenting APIs, thus enhancing the data reuse process.
- Interactive documentation which facilitates the use of Web APIs to easily obtain the desired data, reducing the human-error factor.
- An evaluation with 20 data enthusiasts, each of them accessing a Web API with a basic machine-readable documentation or an interactive natural language documentation generated by our approach, so that we can compare the results.

The novelty of our approach lies in the use of NLP techniques to address the automatic generation of machine- and human-readable documentation for Web APIs accessing open data. Consequently, our approach is relevant for open data reusers who are hampered by poorly documented APIs, which is very often the case, according to Smith and Sandberg [27]. To overcome this barrier, our approach supports open data consumers to understand Web APIs and how to use them.

This article is structured as follows: in Section 2 the related work is described; Section 3 presents the overview of the documentation generation approach, including a detailed explanation; in Section 4 the approach is evaluated with a case study and an experiment with data enthusiasts; and finally, the paper concludes in Section 5.

2. Related work

In this section, existing related work is investigated. First of all, we review existing types of Web API documentation in order to know the current situation of Web APIs to access open data. Then, we explore current solutions to support Web APIs undersandability, such as those regarding automation and NLP.

2.1. Human vs machine readable API documentation

The documentation of Web APIs is generally very rudimentary, hard to follow, and sometimes incomplete or even inaccurate [11]. Moreover, this documentation is mostly manually written and provided as plain text, preventing users to take advantage of having a machine-readable specification automatically generated [23]. Data enthusiasts that deal with Web APIs have to face these documentation obstacles. Therefore, they need to deeply evaluate APIs in order to understand and use them correctly [12], requiring human effort to create them and excessive time to understand that documentation.

In addition Web APIs usually offer documentation that is machine-readable, in other words, easy to process by computers. This documentation is usually written in a programming language following documentation standards, which may be difficult to read by data enthusiasts. To the best of the authors' knowledge, APIs that are manually documented lack machine-processable descriptions because they focus on descriptions understandable by humans [28]. On the other hand, APIs documented automatically are mostly focused on the machine specification and definition of the API rather than in user-readable descriptions, such as in [16]. Efforts have been made to facilitate the use of APIs by the citizenship by offering chatbots [29] that facilitate the user interaction. However, the generation of NL descriptions for the API functions is not currently addressed, which is necessary for fully understanding and using those APIs.

Regarding documentation types, with the information gathered from existing research, such as [11,14,30,31], some conclusions were drawn in Table 1. The existing documentation of Web APIs can be: (i) presented as plain text without following any kind of documentation standard, which promotes the readability for users but is difficult to process by machines; (ii) created manually following the OpenAPI standard, which is machine-readable but because of the human factor it is error prone and has a high cost; (iii) generated automatically in OpenAPI, which is generalisable (can be applied for any API) and machine-readable, but it is difficult to read by users; and finally, (iv) interactive OpenAPI documentation, which can be automatically generated or not, which is easy to process by computers and may be easier to use by users. Although this

Table 1
Advantages and disadvantages of different API documentation types.

Documentation type	Advantages	Disadvantages
Plain text (not following a standard)	User readable	Difficult to process by machines
OpenAPI manually created	Machine processable	Error prone and higher cost
OpenAPI automatically generated	Processable and generalisable	Difficult to read by users
Interactive OpenAPI	Processable and easy to use	Still difficult to read

interactive documentation helps users to query Web APIs, the information about what the API offers may still be difficult to read by users, hindering the use of this API despite its interactivity. This kind of documentation is better compared to those in OpenAPI without interactivity, but is not generally available on current Web APIs. Therefore, there could be a further step in the way of providing documentation of an API: interactive OpenAPI documentation with NL descriptions generated automatically. This interactivity means that users would be able to query the data from the API just by clicking the documentation, which works as a Web interface. In this case, in addition to being easy to process by computers, it would also be easy for users to use the API by this kind of interactive documentation.

As seen in Table 1, the different types of Web API documentation not only offer advantages such as being easy to use or to process, but also disadvantages such as difficulty in their use. There may seem that there is no adequate documentation that offers all the benefits but without the drawbacks. However, we propose to offer a documentation easy to read for users because it includes natural language descriptions, easy to process by machines as long as it follows the OpenAPI standard, easy to obtain since it is generated automatically and not by hand, and easy to use without errors due to its interactivity.

2.2. Supporting web APIs understandability

In this section, related work regarding the comprehension of Web APIs is reviewed: efforts regarding API documentation are first discussed, and then, the generation of text from the NLP perspective is analysed.

From the point of view of the API documentation generation, the main efforts have been focused on the creation of basic documentation (i.e., a machine-readable specification), leaving the interactivity in a secondary plane. In general, the documentation is presented as plain text or following an open standard, but they are not presented as interactive (i.e., a Web interface to interact with the documentation itself). One example that considers the interactivity of API documentation addresses the generation and design of Web frontends based on OpenAPI documentation [32]. However, the tool presented in [32] needs to be handled manually by developers in order to improve the documentation for end users, but the enhancement of the documentation for non-technical users is not addressed and thus they may still have difficulties in understanding and using APIs.

On the other hand, as the documentation can be in natural language, or at least the descriptions of each method or parameter of the API, the use of natural language generation techniques is needed. Indeed, NLG has been used in many applications, such as text summarisation [33], dialog systems [34] or the generation of simplified texts [35]. Moreover, it has been also widely employed and integrated in different research areas, such as in computer vision, for the generation of textual descriptions for human activities in videos [36]; or in business intelligence, for the generation of reports and real time notifications about the state of a company's information technology services [37]. However, the automatic generation of natural language descriptions is left in a secondary plane by related work. In general, related research approaches provide the documentation of Web APIs with manually written descriptions, including methods and parameters. In other cases, these descriptions do not include natural language as they are generated as a rough draft containing a few keywords related to the overall behaviour of the method, or they are not generated at all. Some examples of this type of approaches can be found in [19], where a set of techniques for generating structured documentation of Web APIs from usage examples are detailed. The authors propose a first step towards automatically learning complete service descriptions. However, the generation of methods' descriptions is not tackled. Also, the authors of this paper proposed an automatic API generation process [38,39] which also generates API documentation, but it includes very simple descriptions with keywords extracted from API methods, without exploiting the potentials that the integration of NLP techniques could provide. In [20], a new framework for generating titles for Web tables is presented. This is accomplished by extracting keywords that are relevant to the table. The proposed technique is the first to consider text generation methods for table titles. However, NLG is not applied to generate documentation and it is based on existing table descriptions in plain text. Another paper [21] presents a technique to automatically generate human readable summaries for Java classes. The proposed tool determines the class and method stereotypes and uses them to select the information to be included in the documentation. However, this text generation approach is only valid for programming code documentation and it does not address the description of APIs in natural language. Moreover, the documentation generated is only about a general vision of the Java class, but the explanation of their methods is missing in contrast to our proposal. There are also some research works [14,22–24] that deal with automatically inferring API specifications from manually written documentation. In [22], NLP techniques are used to infer an API specification from existing natural language documentation of the API, and in [14], HTTP requests are employed to generate machine-readable documentation and combine it with existing human-readable information in order to provide complete API documentation. In [23], the authors present an approach for automatically transform HTML documentations into OpenAPI specifications. Moreover, in [24] API documents using semantics based on word embeddings for code migration purposes is analysed. However, in these investigations the problem is handled differently to ours because they assume handwritten API descriptions and then they generate machine-readable documentation based on specific keywords. But the important problem of generating complex descriptions of the API in natural language is not addressed.

The importance and usefulness of API documentation, especially in OpenAPI format is emphasised in [40] and [41]. They deal with generating models from API documentations, but the problem of API documentation descriptions' quality is not addressed and they do not address the generation of these descriptions in natural language. Moreover, the research presented in [11] is focused on improving existing data-intensive APIs and their maintenance through the analysis of their usage by users. From this analysis, the documentation of the API can then be improved. Related to the readability of Web Services, in [25] the authors propose a practical metric to quantify readability in WSDL documents and best practices to improve their readability.

From the NLP point of view, the NLG task is essential since it allows the automatic generation of text regardless of the type of input (e.g., text or non-linguistic data). This task has been commonly addressed through the use of knowledge-based approaches. This type of approaches commonly rely on the use of templates and rules for generating text [42]. For example, PASS [43] is a system that describes non-linguistic data, which generates soccer reports. This system creates a summary of a specific match employing a template-based approach. The input to this system are the match statistics, as well as heterogeneous data such as the league, the date, the match events, the players, the total number of shots or the accuracy of the passes. The templates used by the system PASS were manually derived from sentences in the MeMo FC corpus [44]. Another example is the work presented in [45], where a computational system for generating linguistic descriptions from video camera images, in the context of traffic in a roundabout, is described. In this sense, the authors first analyse the image to obtain information about the vehicles in the roundabout entering lanes and then generate a description of the overall roundabout status using fuzzy logic and templates. Furthermore, within the NLG field the interest in the generation of reasoning explanations has increased. This type of systems generates the explanation about the decisions made in order to achieve a goal, such as in the steps that a system followed in the execution of an algorithm, or the decisions made in the resolution of a mathematical problem. Although this research line could not be exactly the same as our goal of generating descriptive documentation in natural language, it is closely related since this type of explanations often include the

description of variables and terms, and the techniques used could be also appropriate for the purpose of our research. Some examples of reasoning explanation systems can be found in [46], where a system that generated explanations for a machine learning algorithm decision is presented; or in [47], where a semi-automatic process to analyse business models and generate a requirement documents that describes the models is proposed.

As seen in the existing related work, dealing with the automatic generation of both machine and human documentation for Web APIs is barely addressed. There are some research that deals with similar problems, but in some cases the problem arises in other scenarios, such as Web services rather than APIs. Therefore, to the best of our knowledge, the solution proposed in this paper is novel and goes beyond the state of the art in providing the suitable and interactive documentation for Web APIs, in a fully automatic manner, by integrating NLP techniques into the API documentation generation process.

3. NLP For improving API documentation

In this section, our approach[1] for the documentation generation process is described. This constitutes a further step on the documentation of data Web APIs, by improving and enriching current documentation with suitable descriptions in natural language (NL). These descriptions are automatically generated and included in an interactive adaptation of the OpenAPI documentation. Specifically, the process of improving the documentation is able to add NL descriptions to any existing OpenAPI documentation of a Web API. With this incorporation of descriptions, the documentation of Web APIs will be not only machine-readable but also understandable by users. Moreover, this documentation of the API is made interactive, which means that it is presented through a Web interface so that users can directly query the underlying API to easily get the data. Therefore, the enhancement of the comprehension and use of Web APIs will promote as a result the reuse of data.

The process of improving open data Web API documentation starts with an existing OpenAPI documentation of an open API. After that, the generation of natural language descriptions are automatically generated and integrated in the existing initial documentation. In this manner, a more detailed and comprehensive documentation is obtained. Finally, this documentation is made interactive by creating a Web interface version with the help of Swagger[2] open source specification and tools. This whole process is now explained step by step in the following subsections.

3.1. Starting point: An existing openAPI documentation

The process starts with a basic machine-readable API documentation (Fig. 1) following the OpenAPI standard in JSON format. Then, using the proposed NL descriptions generator approach, a set of descriptions in natural language is created and later appended to the documentation of the API, facilitating the understanding of the Web API.

Therefore, the first step of the process (Step 1 in Fig. 1) begins with a Web API including an OpenAPI documentation which is only machine-readable. This documentation is not easy to read by human users, so generating NL descriptions of the API is essential to better understand and access it to get the underlying data.

In order to generate these NL descriptions, the automatic generator process first gathers the following information about the API from this starting OpenAPI documentation: the title given to the API; the name of each method; all the parameters from each method, with name, type and example value; and all the properties given as a result of the API, also with name, type and example data. This information is extracted from the input documentation by analysing the JSON object that contains all the components of the API documentation. In order to do so, the process iterates through the JSON objects and arrays that the OpenAPI standard specifies, such as API "title", "paths", "components", "parameters" and "properties".

Once the aforementioned information is gathered, the process continues with the next step in order to automatically generate the general description of the API as well as a description for each of the API methods, parameters and properties. This generation process will be further explained in the next subsection.

3.2. Natural language descriptions generator

Taking as input the information extracted in the first step of our approach's architecture, the second step (Step 2 in Fig. 1) is the generation of descriptions using NLP techniques. From this information, the generation of NL descriptions is performed using and integrating different NLP techniques. Specifically, tokenisation, word sense disambiguation and a NLG template-based approach are employed to generate the descriptions that will be added to the OpenAPI specification. An overview of these steps is shown in Fig. 2.

Template-based approaches have proven to generate relatively high quality texts and faster than other NLG approaches [43], being suitable in the case of the generation of descriptive API documentation since its integration with other systems or approaches would not affect their performance. The type of templates used in this approach is usually a text with gaps that must be filled with specific information in order to complete its semantic meaning. The information for generating the method's descriptions within the API documentation is provided by the input API specification. In this regard, the name of the methods or the parameters/properties of these methods will be used for producing semantically enriched descriptions.

The templates to be used in this proposed approach are designed considering as reference the generic CKAN API[3], which is used by many open data platforms such as Data.gov (the U.S. Government's open data platform) and Data.gov.uk (the U.K. Government's open data platform). Since the text needed to describe the different parts of the API must be different, a variety of 3 different generic templates were hand-crafted (as shown in Fig. 3): (i) general API description; (ii) API method description; and (iii) parameter/property description. Within these templates, the most important information is shown in bold in order to highlight it to the readers. In addition, there are different variables used along the templates to include the information coming from the API: ${fileName} (name of the dataset to access through the API), ${params} (list of available parameters to filter data), ${methodName} and ${methodName2} (names of two methods of the API as example), ${methodEx} and ${methodEx2} (example values for two methods of the API), ${recordExample} (API response example in JSON format), ${ptype} (indicates if it corresponds to a parameter or a property of the API), ${pname} (name of the parameter or property), ${datatype} (parameter or property data type such as string), ${pexample} (example of parameter or property value), ${pdefinition} (definition of the parameter or property name), ${required} (if the parameter is required in the method call) and ${pin} (if the parameter must be in query or in the path of the API request).

As illustrated in Fig. 3, the variables ${fileName}, ${params}, ${methodName}, ${methodName2},${methodEx},${methodEx2} and ${recordExample} are employed in the first and second templates (i.e., the general API and method descriptions) to fill the available gaps and create an understandable and well-structured text. In this case, the information contained in these variables is solely provided by the API specification. However, concerning the third template (i.e., the generation of parameter/property descriptions), more information is required

[1] https://github.com/cgmora12/NL4OpenAPI
[2] https://swagger.io/
[3] https://docs.ckan.org/en/latest/api/index.html

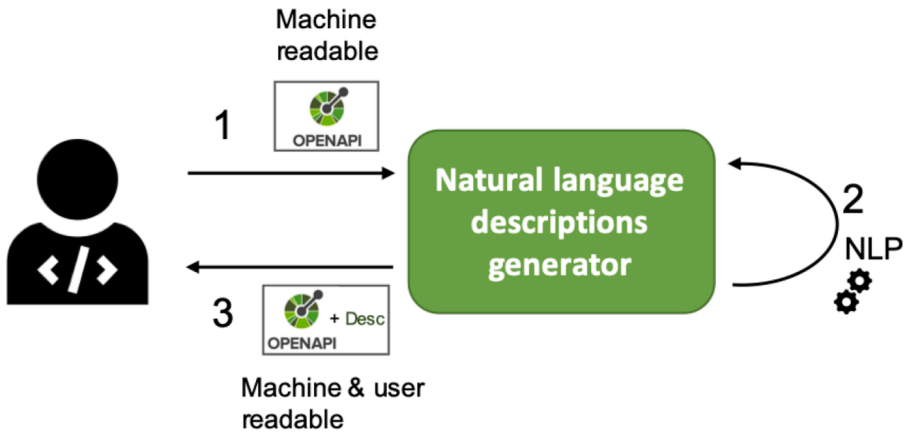

Fig. 1. Overview of the automatic generation of API documentation.

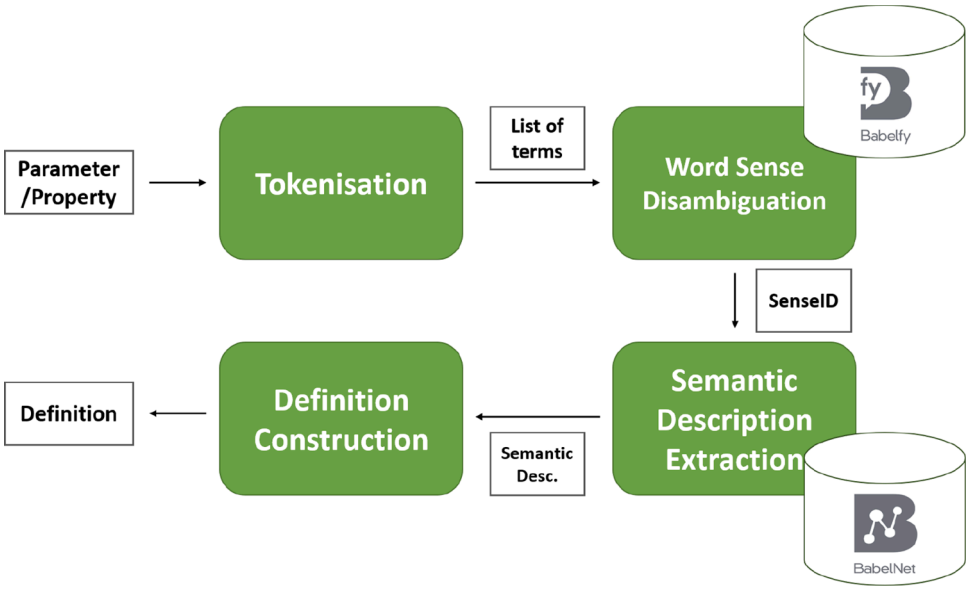

Fig. 2. Overview of our NLP approach for generating API documentation.

in order to generate a description that helps the data enthusiasts users to better understand the parameters to be used in the API. Therefore, in the proposed approach, the information coming from the API specification is enriched with the application of NLP techniques, such as word sense disambiguation, in conjunction to the use of knowledge-bases and semantic resources. The use of this type of techniques and resources may be helpful in situations where is difficult to discern the meaning of a parameter/property that correctly fits the context of the API documentation. For example, in the context of an API for obtaining employment data, the word "mean" would adopt the meaning of average instead of other meanings such as a "stingy person". Concerning the type of resources mentioned above, Babelfy [48] and BabelNet [49] are specifically used for the generation of the parameter/property descriptions. These resources provide semantic knowledge to the generated description, thus enriching the API documentation and providing the user with definitions and examples of the data queried by the API. The former performs word sense disambiguation, using a semantic network. The latter is a multilingual encyclopedic dictionary and a semantic network that integrates information from several sources such as WordNet [50] or Wikipedia. Babelfy allows the disambiguation of a specific term (either by using only this specific term or employing it together with an example), obtaining the sense of the term in the form of an ID representing a set of synonyms. Then, searching this ID in BabelNet, the semantic description of the term is obtained. In this preliminary version of the approach, in case that a term has more than one sense, we choose the one with the highest disambiguation score[4] provided by Babelfy. Since in many cases, the parameter/property are not a single token, an intermediate processing step is needed before using BabelNet/Babelfy resources. This is due to the fact that these resources need as input a sentence or a term with the words to disambiguate correctly spelled. Therefore, in the case that a parameter/property contains more than one word in a single token (e.g., when several words are separated by a underscore: *Country_Code*), tokenisation[5], via regular expressions, is employed in order to separate these words. In this sense, the following cases are considered: (i) when the words are separated by non-alpha numeric characters, such as "&","_" or "$"; and, (ii) when the words are in camel case format, such as "*FlagsCode*" or "*CountryCode*". Ultimately, if Babelfy is not able to disambiguate the parameter/property as a whole, or any of its components, its description will not appear in the final API documentation. Once the enriched description is generated, as

[4] A value between 0 and 1 to indicate the confidence degree of the algorithm for the disambiguation of the term [51].

[5] The process of splitting a stream of text into more basic units such as words, phrases or tokens (elements with an identified meaning).

General API template

This **interactive documentation** allows to extract information from **data about ${fileName}** in a simple way without having to be an expert in APIs. This API allows to filter by different characteristics, for example: ${params}.
The API response will be in **JSON format**, containing **a list of objects with properties**: ${params}.
An **example** of a query would be the following: localhost:8080/v1/${methodName}/${methEx}
You can also **query the API by different parameters at the same time** using the **API main method** as follows
localhost:8080/v1/?${methodName}=${methEx}&${methodName2}=${methEx2}

Method template

Find ${fileName} by ${methodName}.
This **API method can be used to find ${fileName} data by ${methodName}**, and you **must filter the results by a specific ${methodName}**.
These results will contain the following fields: ${params}.
An **example** of using this method is "/${methodName}/${methEx}" or "/?${methodName}=${methEx}", whose result would include at least the following record: ${recordExample}

Parameter/property template

The ${ptype} ${pname} is of type ${datatype} and an example is ${pexample}. It indicates the ${pdefinition}.
This parameter is ${required} and must be in ${pin}.

Fig. 3. Templates to generate API documentation NL descriptions.

a result of the outlined procedure, the information in the third template is completed with the variables ${ptype}, ${pname}, ${datatype}, ${pexample}, ${required} and ${pin}; thus producing a well-structured text.

3.3. Generating an interactive openapi documentation

Finally, the last step consists of creating an interactive documentation which is easy to process by machines and easy to read by data enthusiasts at the same time (Step 3 in Fig. 1). First, the generated descriptions in natural language of the API are integrated in the existing (machine-readable) documentation of the API. Then, this documentation is going to be made interactive by automatically creating a Web interface. In order to do so, the necessary Web programming code (in HTML, CSS and Javascript languages) is appended to the documentation, which is incorporated in a Web server in the NodeJS environment to allow interactivity. Therefore, this documentation of the API will be now both machine and user readable, all in a single and interactive OpenAPI documentation (shown in Fig. 4).

In order to perform this integration of descriptions, the input OpenAPI is analysed to locate where these generated NL descriptions have to be placed within the OpenAPI specification: the general description of the API is placed in the "info" component of the OpenAPI documentation; the description of each method of the API is placed in the corresponding OpenAPI "path"; the description of each parameter is placed in the corresponding "parameter" of each "path"; and finally, the description of each property is placed in the corresponding "property" of the "components" object.

When this process ends, the OpenAPI documentation including descriptions in natural language is included in an automatically generated server to offer the documentation as a Web interface. It consists in a NodeJS[6] server that is created with the help of the Swagger Codegen tool[7], which creates the structure of the server and manages the calls to the underlying API. An example of this generated interactive documentation is shown and explained in detail in Section 4.1.1. One of the main benefits of having an interactive documentation is the consistency of the results, which means that the possibility of introducing errors such as writing mistakes is severely reduced, and therefore there is less chance of obtaining erroneous data. For example, if users want to filter the data by a parameter called "Area_Type", they have to write the parameter exactly taking into account symbols and uppercase. However, with our interactive approach users do not need to write down the parameter and they can just click on the specific method for filtering.

4. Evaluation

In this section the proposed approach is first evaluated with different examples. Then, a specific case study is introduced, and finally, an experiment with real users is presented. To ensure the correct performance of the proposed approach, a set of 5 different OpenAPI documentations have been used. Each documentation is related to one Web API that provides direct access to specific data about different topics.

Table 2 shows a brief summary of the time spent by the proposed approach for generating the OpenAPI documentation with natural language descriptions. As can be seen, the time required to generate the API documentation by our proposed approach is affected by the number of API methods'. In this sense, the disambiguation process would introduce a delay in generation time for each method. Taking into account that the Web APIs contain between 7 and 46 methods, the time to generate the

[6] https://nodejs.org
[7] https://swagger.io/tools/swagger-codegen/

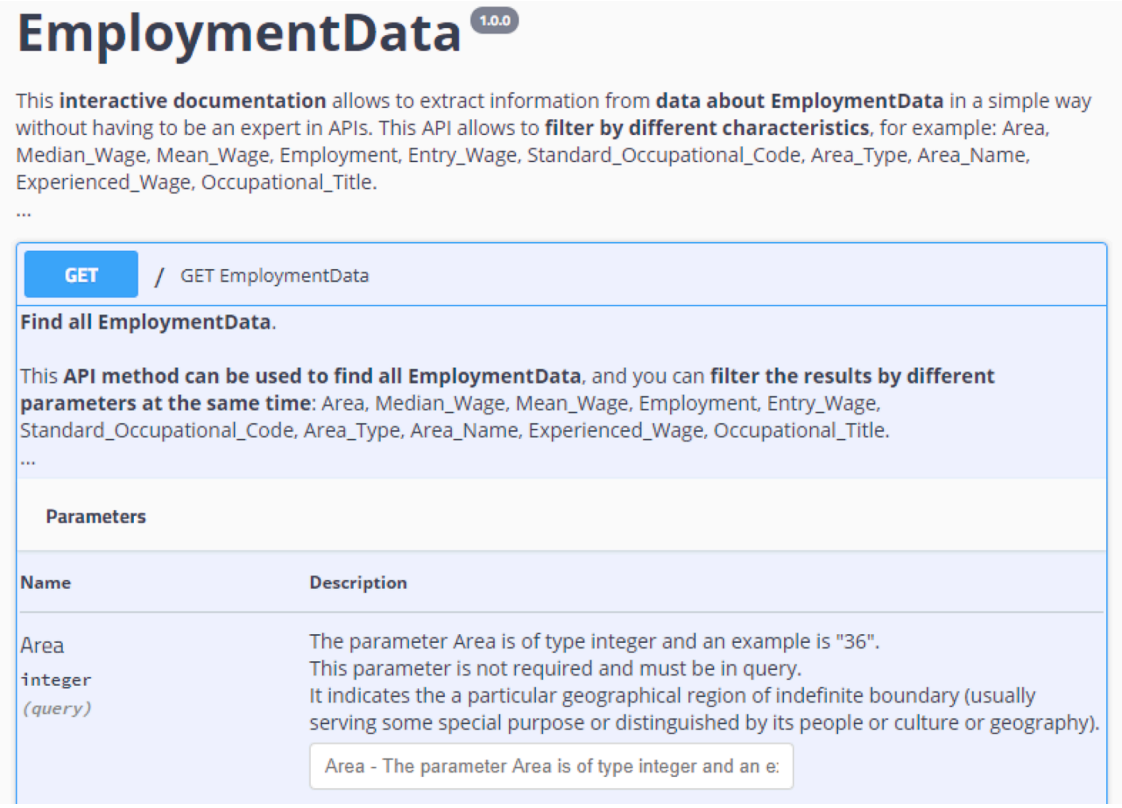

Fig. 4. Extract of the obtained interactive documentation.

Table 2
Generation of NL descriptions for different datasets.

Web API	# of methods	Generation time
Voter History Data	20	43.02 s
Biodiversity by County	13	36.54 s
Employment Statistics	10	25.88 s
Leading Causes of Death	7	24.18 s
Demographic Statistics	46	214.89 s

Table 3
Extract of the dataset used as example in the case study.

Area Name	Occupational Title	Employment	Mean Wage	Median Wage
NY	All Occupations	9,385,620	63,270	46,010
NY	Budget Analysts	3150	80,060	76,720
NY	Credit Analysts	7440	121,490	104,600
NY	Financial Analysts	53,250	139,930	109,230
NY	Sales Managers	23,120	202,230	183,850

related API documentation including natural language descriptions is between 24 and 215 seconds.

4.1. Case study

A case study is now introduced to illustrate the whole process and show the feasibility and usefulness of our proposal. It consists of applying the proposed approach described in Section 3 to an existing data Web API. With this example we attempt to demonstrate that data enthusiasts need to make a great effort to interpret and understand the available third-party Web APIs because of the lack of suitable human-readable API documentation, which also hampers the reusability of data.

Our scenario includes open data from a Web API providing employment statistics, from which we are going to create an interactive OpenAPI documentation with natural language descriptions.

4.1.1. Generating documentation for employment data

An API providing employment statistics has been chosen to illustrate this process. The information provided by the API is about employment in a wide variety of professions in New York. Table 3 shows an extract of employment data from the dataset, which contains information such as occupation, area name or wage. This data originally comes from the U.S. Government's open data website (data.gov).

This "*Employment Statistics*" dataset is used as example with its Web API[8] and its OpenAPI documentation[9]. This documentation includes information about a set of methods: a general method to retrieve all the data from the source; and one method to filter by the values contained in each column of the data source. However, this OpenAPI documentation does not include NL descriptions, which makes the application of our approach to generate NL descriptions very valuable in this context.

In order to improve the existing documentation, the generation of NL descriptions is performed to create an API general description, an explanation of each API method and also a definition of each parameter of the API. After that, the documentation is made interactive by providing the documentation as a Web interface. Finally, this complete documentation is exposed online[10] so that we can just test it.

4.1.2. API Documentation comparison

The objective of this comparison is to visually contrast how our descriptions have benefited and increased the quality of the basic documentation we initially had.

As shown in Fig. 5, the API general description in the original

[8] https://wake.dlsi.ua.es/EmploymentAPI/
[9] https://wake.dlsi.ua.es/EmploymentAPI/docs/openapi.json
[10] https://wake.dlsi.ua.es/EmploymentAPI/docs/

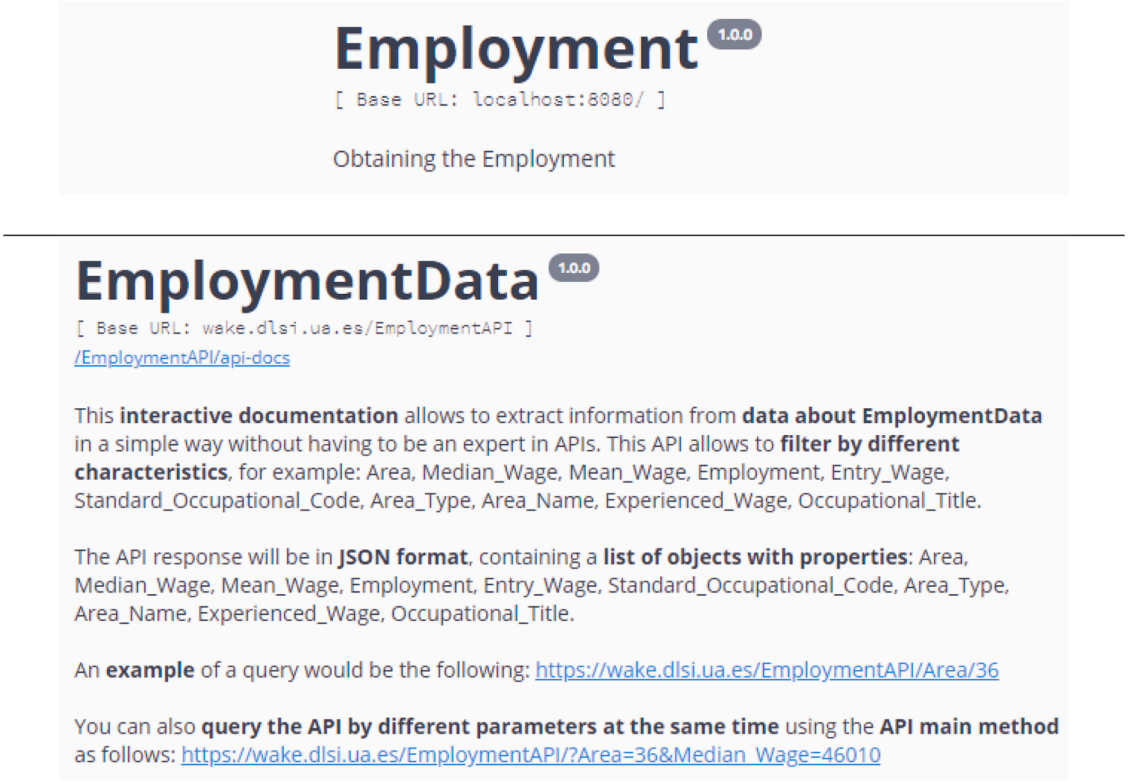

Fig. 5. Description of the API in the basic OpenAPI documentation (above) and the interactive one generated by our approach that uses NLP (below).

documentation[9] is almost empty (Fig. 5 above), whereas the documentation[10] generated using NLP (Fig. 5 below) includes a complete description about the data offered in the API, the filters that can be applied, the result format and an example about how to query the Web API.

Moreover, the comparison of the API main method description is shown in Fig. 6. The original API main method documentation (Fig. 6 above) only includes a short summary which is equal to the description of other methods of the API. However, the documentation of the API main method generated by our approach (Fig. 6 below) is interactive as it allows you to easily query the API just clicking in the button provided by the Web interface. It also includes an example about how to query this API method and a complete description about the data offered by this concrete method, which can be filtered by different parameters.

Finally, the documentation about the parameters of an API method are compared in Fig. 7. The description of these parameters is almost empty in the original documentation, meanwhile the documentation of each parameter generated by the proposed approach includes a complete description in natural language about the type of data, an example of use, if it is required or not, how to pass this parameter to the API, and finally, a semantic definition about the meaning of the parameter itself.

Therefore, from the comparison of a basic open data Web API documentation without integrating NLP approach previously presented[11], we have seen that the documentation generated by our approach contains more information for users to understand and query the API to get the desired data. With this help, data enthusiasts can easily promote the use of open data available through Web APIs.

4.2. Experiment with users

To evaluate our approach with real users, an experiment comparing the use of OpenAPI documentation with the use of interactive OpenAPI documentation with NL descriptions created by our approach was conducted. This experiment was carried out with data enthusiasts using the original basic documentation and the generated by our approach for 3 Web APIs selected from Table 2, which makes a total of 6 API documentations. For each documentation, we supplied a different survey. In this experiment with users, 20 participants from Alicante (Spain) accessed specific data using the related Web API and its documentation. These participants were data enthusiasts without programming skills which attended a big data seminar oriented to take advantage of data in different scenarios, which was conducted by the Employment Centre of the University of Alicante[12].

4.2.1. Experiment setup

First of all, we provide an explanation of the experiment for the participants in a website[13] we created specifically for the experiment. This website includes a link to one of the 6 available surveys, randomly for each participant to take a different survey (only one). These 6 surveys are different because there is 3 available Web APIs (their structure is the same but they are about different data), and each one has the option to use the original documentation or the documentation generated using our NL documentation approach. However, the difficulty of the surveys remains the same and also the questions for participants are equal despite the documentation they are using, the only changes affect the data asked depending on the API they have to query (because the data offered by the APIs is different).

A preliminary training phase was conducted in order to help data enthusiasts to know about APIs and OpenAPI documentation. Therefore, we provided in the experiment website[13] an explanatory video about what is a Web API, how to access data and how to read an OpenAPI documentation.

Then, users were asked to fill in an online survey where they have to

[11] https://github.com/cgmora12/AG

[12] https://web.ua.es/centro-empleo/

[13] https://wake.dlsi.ua.es/ExperimentDocumentation

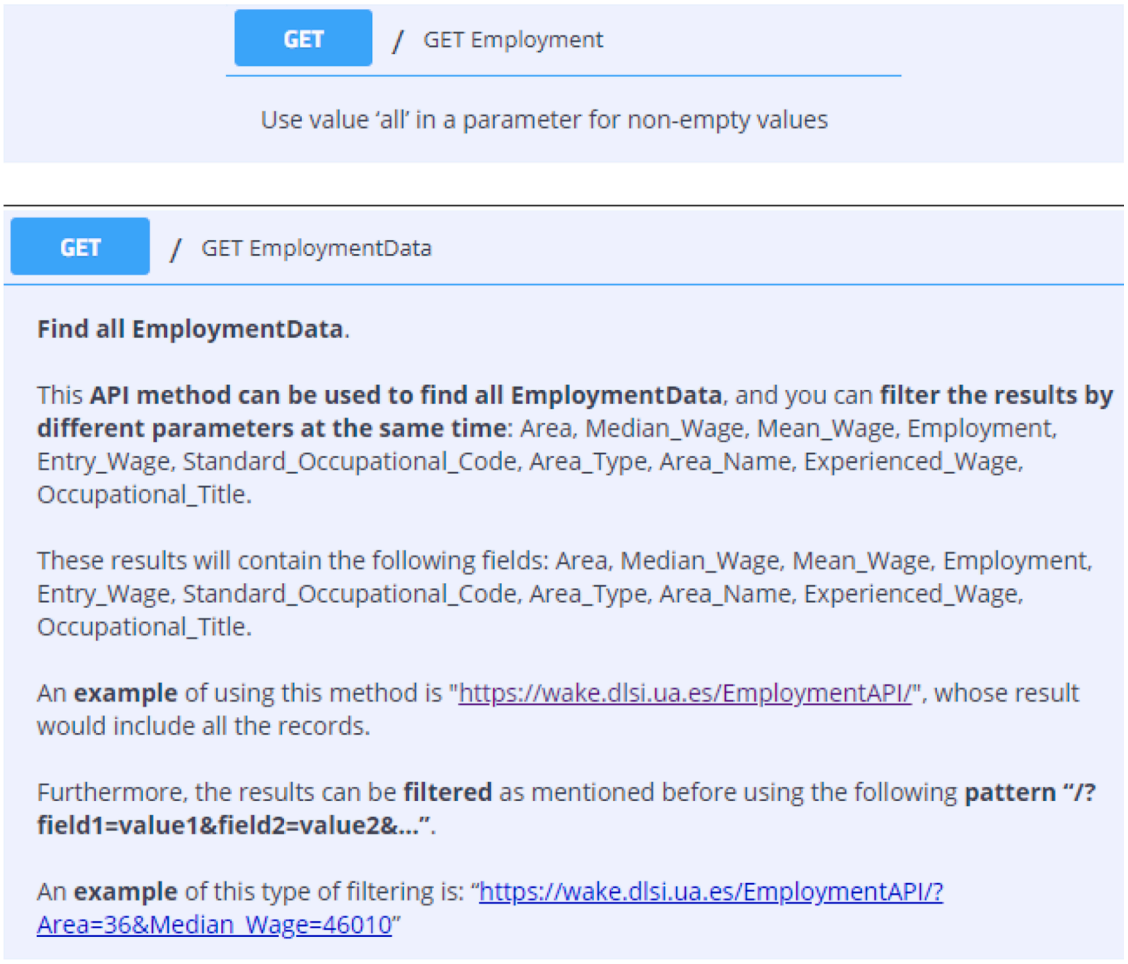

Fig. 6. Comparison of the API main method description in the original OpenAPI documentation without applying our NLP approach (above) and integrating NLP (below).

perform the following tasks: (i) read the provided documentation; (ii) query the specified API to get specific data; (iii) answer some questions about the retrieved data; and finally, (iv) answer general questions about the documentation used and their satisfaction about the process of accessing data using the API and its documentation.

In this evaluation we recorded the time spent to answer the questions and the correctness of answers in order to know the effort they needed to query the API using the provided documentation and their efficiency. Therefore, the data analysis used for this experiment consisted of analysing the time in seconds for the correct answers with ANOVA, using the type of documentation and the specific API as fixed factors. It is also important to consider that the time has been transformed with square root in order to avoid heteroscedasticity (circumstance in which the standard deviations of a predicted variable are non-constant).

4.2.2. Online surveys

There is a different survey for each type of documentation and each different Web API, so that we can compare the use of the documentation in different scenarios. All the surveys are available online: 1) Employment API with original basic documentation survey[14], 2) Employment API with complete NL documentation survey[15], 3) Biodiversity API with original basic documentation survey[16], 4) Biodiversity API with complete NL documentation survey[17], 5) Death Causes API with original basic documentation survey[18], 6) Death Causes API with complete NL documentation survey[19].

Next, an example of survey for the Employment API with interactive NL documentation is going to be explained in detail. This example is shown in the Appendix Appendix B.

First of all, in the introduction of the survey we offer information about the API and we encourage users to carefully read the provided documentation before answering the survey. Then, users are asked to perform 6 different queries to the API using the documentation. One of the queries consists of obtaining the mean wage of financial managers employees from "Long Island Region". We also asked the user about the query used to obtain this required data.

Finally, the general questions were related to gather information about previous experience in the use of Web APIs, difficulty in the queries performed, usefulness of the documentation, satisfaction with the documentation and possible improvements or changes for the documentation used.

4.2.3. Experiment results

An extract of the results of the evaluation are shown in the Appendix A (Tables A.4–A.9), presenting also the average of the results in the last row of each table. As can be seen, although users in the

[14] https://wake.limequery.com/982922?lang=es
[15] https://cgm119.limequery.com/837582?lang=es
[16] https://wake.limequery.com/574386?lang=es
[17] https://cgm119.limequery.com/123695?lang=es
[18] https://wake.limequery.com/591598?lang=es
[19] https://cgm119.limequery.com/923551?lang=es

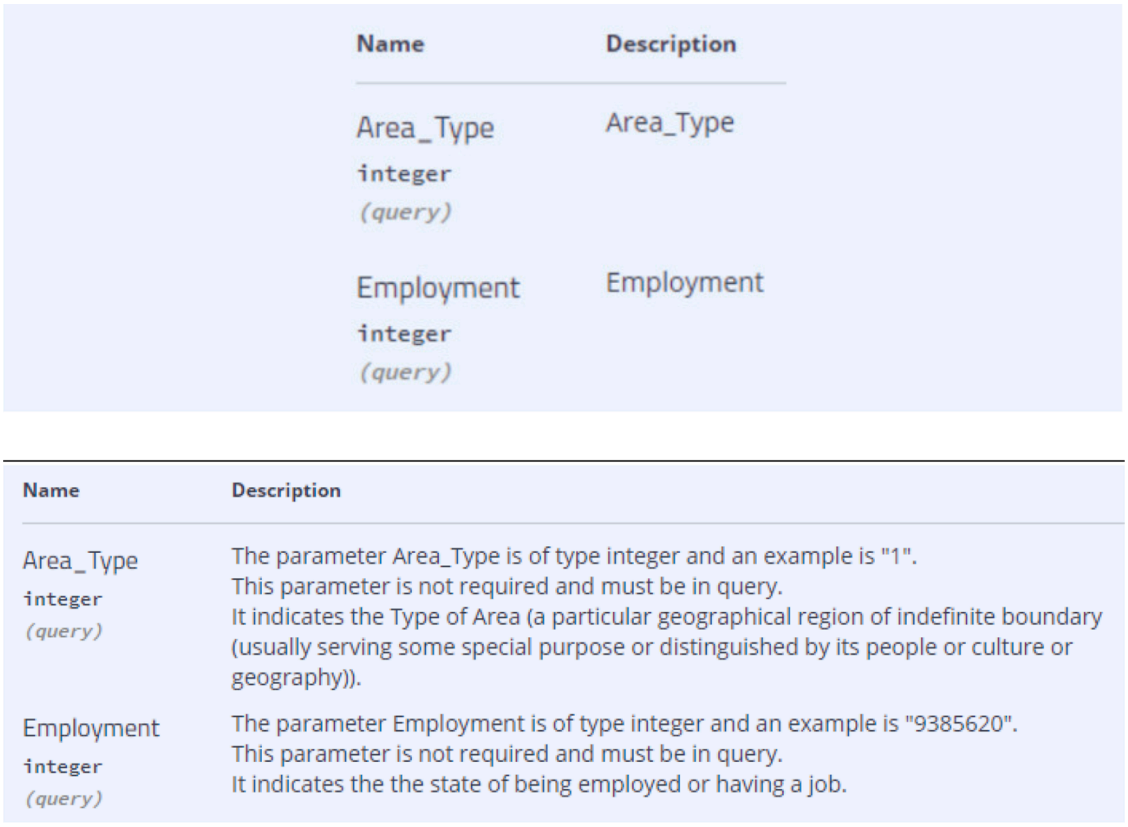

Fig. 7. Description of the API main method parameters in the basic original OpenAPI documentation (above) and integrating NLP (below).

Table A1
Results of survey about employment API with original OpenAPI documentation. Average results in the last row.

Correct answers	Correct queries	Previous experience with APIs	Usefulness of documentation	Clear documentation	Time (s)	General satisfaction
2/6	0/6	No	5 of 5 stars	No	2183.64	3 of 5 stars
5/6	6/6	No	5 of 5 stars	Yes	620.15	4 of 5 stars
5/6	5/6	Yes	3 of 5 stars	Yes	2383.56	2 of 5 stars
5/6	1/6	No	4 of 5 stars	Yes	2580	4 of 5 stars
6/6	6/6	No	1 of 5 stars	No	4790	3 of 5 stars
0/6	0/6	No	1 of 5 stars	No	-	1 of 5 stars
4/6	0/6	No	2 of 5 stars	No	1655	2 of 5 stars
3.86/6	2.57/6	No	3 of 5 stars	No	2368,73	2.71 of 5 stars

Table A2
Results of survey about employment API with NL OpenAPI documentation. Average results in the last row.

Correct answers	Correct queries	Previous experience with APIs	Usefulness of documentation	Clear documentation	Time (s)	General satisfaction
3/6	1/6	No	4 of 5 stars	Yes	1601.12	4 of 5 stars
3/6	3/6	No	3 of 5 stars	No	8137.8	3 of 5 stars
5/6	6/6	Yes	5 of 5 stars	Yes	6739.7	5 of 5 stars
4/6	4/6	No	3 of 5 stars	No	9947.26	4 of 5 stars
5/6	6/6	No	4 of 5 stars	Yes	1339.14	4 of 5 stars
4/6	4/6	No	3.8 of 5 stars	Yes	5553	4 of 5 stars

Table A3
Results of survey about biodiversity API with original OpenAPI documentation. Average results in the last row.

Correct answers	Correct queries	Previous experience with APIs	Usefulness of documentation	Clear documentation	Time (s)	General satisfaction
2/6	0/6	No	1 of 5 stars	No	2120.57	1 of 5 stars
5/6	5/6	No	2 of 5 stars	Yes	2616.04	3 of 5 stars
5/6	6/6	No	4 of 5 stars	No	2179.35	4 of 5 stars
0/6	0/6	No	3 of 5 stars	No	5132.28	3 of 5 stars
1/6	0/6	No	3 of 5 stars	No	873.38	3 of 5 stars
2.6/6	2.2/6	No	2.6 of 5 stars	No	2584.32	2.8 of 5 stars

Table A4
Results of survey about biodiversity API with NL OpenAPI documentation. Average results in the last row.

Correct answers	Correct queries	Previous experience with APIs	Usefulness of documentation	Clear documentation	Time (s)	General satisfaction
6/6	5/6	No	5 of 5 stars	Yes	909.59	4 of 5 stars
5/6	4/6	No	2 of 5 stars	No	1652.84	4 of 5 stars
1/6	0/6	No	3 of 5 stars	Yes	1583.71	3 of 5 stars
0/6	0/6	No	4 of 5 stars	Yes	6213.67	4 of 5 stars
4/6	3/6	No	4 of 5 stars	No	1724.81	4 of 5 stars
5/6	5/6	No	5 of 5 stars	Yes	1430.28	4 of 5 stars
3.5/6	2.83/6	No	3.83 of 5 stars	Yes	2252.48	3.83 of 5 stars

Table A5
Results of survey about death causes API with original OpenAPI documentation. Average results in the last row.

Correct answers	Correct queries	Previous experience with APIs	Usefulness of documentation	Clear documentation	Time (s)	General satisfaction
1/6	0/6	No	4 of 5 stars	Yes	1274.77	3 of 5 stars
5/6	5/6	Yes	4 of 5 stars	Yes	2352.65	4 of 5 stars
3/6	0/6	No	3 of 5 stars	No	1895.64	3 of 5 stars
4/6	2/6	No	2 of 5 stars	Yes	8242.53	3 of 5 stars
2/6	0/6	No	3 of 5 stars	Yes	1290.4	3 of 5 stars
0/6	3/6	Yes	3 of 5 stars	No	1231.86	5 of 5 stars
6/6	1/6	No	3 of 5 stars	Yes	1292.26	4 of 5 stars
4/6	0/6	No	2 of 5 stars	No	931.98	3 of 5 stars
3.13/6	1.38/6	No	3 of 5 stars	Yes	2314.01	3.5 of 5 stars

Table A6
Results of survey about death causes API with NL OpenAPI documentation. Average results in the last row.

Correct answers	Correct queries	Previous experience with APIs	Usefulness of documentation	Clear documentation	Time (s)	General satisfaction
4/6	1/6	No	4 of 5 stars	No	1341.82	4 of 5 stars
6/6	1/6	Yes	4 of 5 stars	Yes	741.67	4 of 5 stars
6/6	6/6	Yes	4 of 5 stars	Yes	1546.19	4 of 5 stars
6/6	2/6	Yes	1 of 5 stars	No	1931.3	1 of 5 stars
6/6	6/6	Yes	4 of 5 stars	Yes	1234.56	4 of 5 stars
0/6	1/6	No	2 of 5 stars	No	1355.11	2 of 5 stars
6/6	5/6	No	5 of 5 stars	Yes	737.5	5 of 5 stars
0/6	1/6	No	2 of 5 stars	No	875.5	2 of 5 stars
4.25/6	2.875/6	Yes/No	3.25 of 5 stars	Yes/No	1220.45	3.25 of 5 stars

surveys with documentation in NL can take more time in some cases to answer the questions, they generally take less time to achieve correct answers than the users with the original machine-readable OpenAPI documentation. In general, the usefulness of the documentation for obtaining the required data was higher using our proposed approach, meanwhile the general satisfaction was also greater using documentation with natural language descriptions. It is also important to note that, in general, users consider the documentation in NL more clear than the original basic documentation, and also around the 90% of participants recognise that a documentation in NL is important to know how to use APIs and then access relevant data, as shown in Fig. 8.

For participants with original documentation, the number of correct answers and correct queries to get the data are significantly lower than for participants with NL documentation generated by our approach (Fig. 9). Moreover, as shown in Fig. 10, users considered this NL documentation as highly useful rather than the original documentation, and in some cases, users of machine-readable documentation preferred not using the documentation because of its format.

As shown in Fig. 11, the time needed to answer each question of the survey was generally lower than 10 minutes and in average around 4 minutes. Moreover, the number of questions that surpassed 20 seconds to answer them is low, but they introduce a variability in the data observed. Since the time measured can induce to errors because users can stay in the same question for a long time without even looking at the survey, we considered important for the correctness of the results not to take into account those question's responses that took more than 20 minutes. With this consideration, the time spent on (correctly) resolving a question by performing a query to the API is higher with original documentation than with NL documentation. Moreover, the density of the response time with the complete documentation (in NL) is concentrated around the average (3 minutes, line in red). However, with original documentation more dispersion of the response times is observed, as there are more responses with longer times (line in blue). Also, Fig. 12 shows that in all types of surveys (biodiversity, deathCauses, employment) it always takes longer to respond with original documentation than with NL documentation ($F_{1,128}=6.192$; p-value=0.0141). There is no interaction between the two factors ($F_{2,128}=0.443$; p-value=0.1538) and the pattern is the same for all surveys, with more time spent on original documentation.

In addition, in all surveys the time was less when the documentation was in NL rather than the original machine-readable OpenAPI documentation. Then, the effect of the type of documentation is significant (p-value=0.0141).

Regarding the general satisfaction of participants (Fig. 13, using the NL documentation they reported a higher rating (general average of 7.2 out of 10) compared to the satisfaction when using the original documentation (general average of 6 out of 10).

Therefore, this experiment shows that our approach successfully achieves the objective of helping users to access data through APIs by improving existing documentation. Comparing the documentation created by our approach with the existing machine-readable, we can conclude that users easily find the information they are requested to query, with fewer effort, higher user satisfaction and less errors.

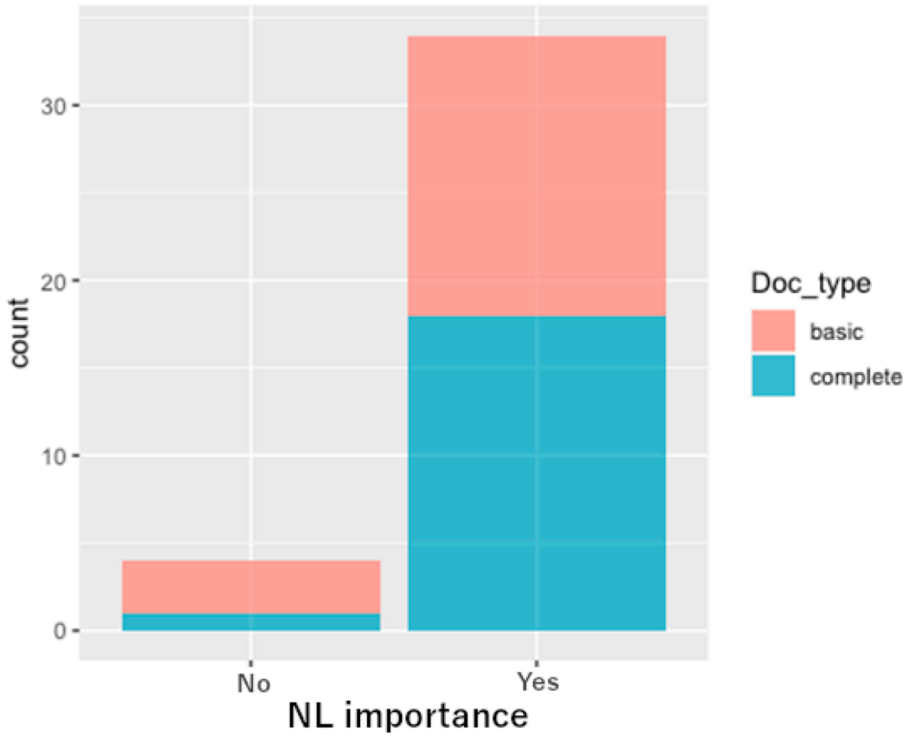

Fig. 8. Participants consideration about if a documentation in NL is important for Web APIs, being basic the documentation without NLP and complete the documentation generated with NLP by our approach.

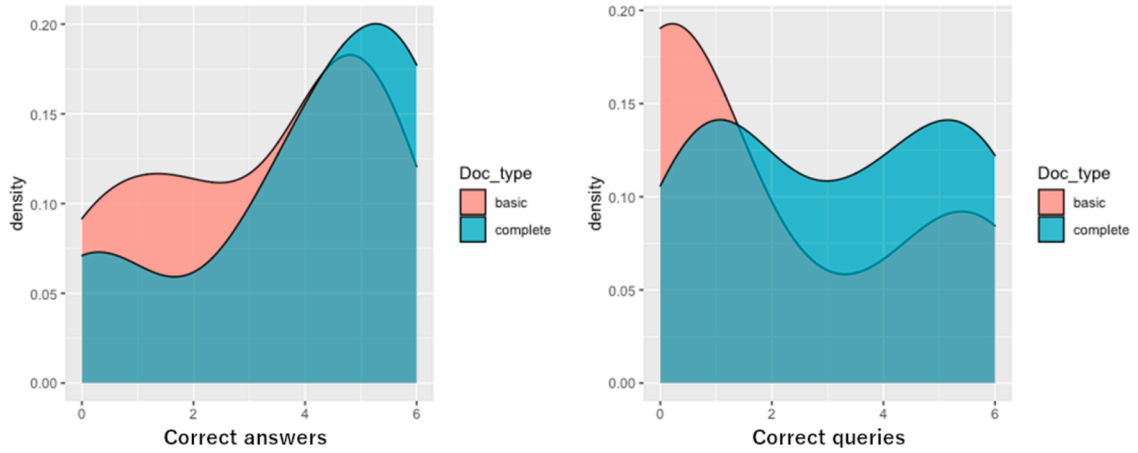

Fig. 9. Correctness of answers (left) and queries (right) differentiated by participants using NL documentation vs using basic original documentation.

4.3. Discussion

From this evaluation we can state that the inclusion of interactivity and NLP techniques improve the documentation of Web APIs, which can now be easily understood by users and processed by machines at the same time. Compared to not having any documentation, or having only a machine-readable documentation that only includes the names of the API methods, our approach contributes to the existing related work by providing both human and machine-readable documentation, which simplifies the comprehension and reusability of the data. Furthermore, the generated documentation allows users to query the underlying API through different methods by the documentation Web interface, thus facilitating the access to the data.

These results can be justified as it is normal that with the original basic documentation, users take less time to answer the surveys than with the generated NL documentation because they do not understand machine-readable documentation and thus they have less to read. However, the main problem is that with the basic documentation there are many more errors because data enthusiasts can be confused with invalid results that they believe to be true. Instead, when using our generated documentation it is advantageous against the misunderstanding as well as the misleading use of the Web API.

In comparison with existing related work on the generation of API documentation [14,16,22–24,29], our approach provides NL descriptions automatically generated, which is not addressed by those related works. As demonstrated in the evaluation with users, these NL descriptions improve the understanding and thus the correct use of the existing APIs comparing with APIs whose documentation does not include NL descriptions for their functions. Also, in other works such [32], the interactivity of the documentation has been addressed. However, since manual interaction is needed and NLG is not addressed in their approach, our approaches cannot be compared in terms of

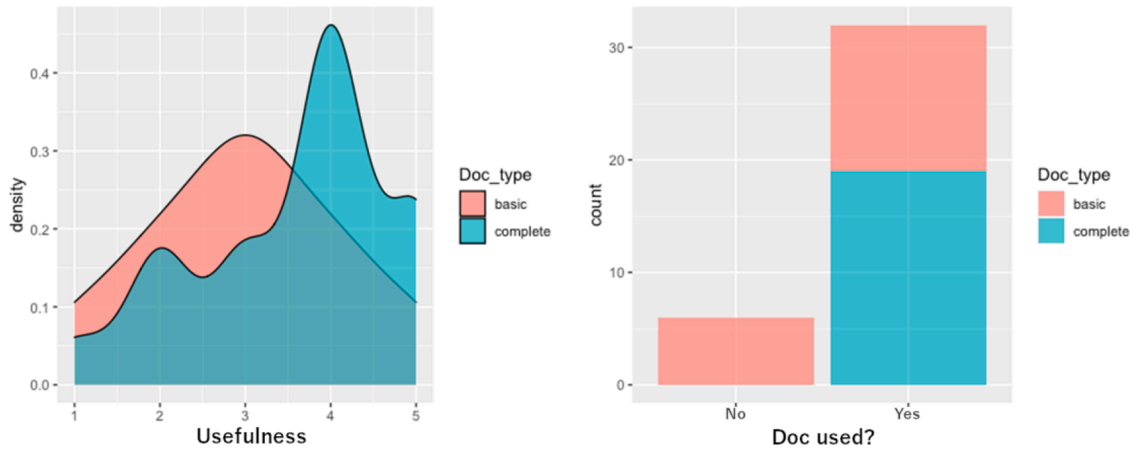

Fig. 10. Usefulness of documentation for participants (in the left), and number of users using the documentation for the experiment (in the right).

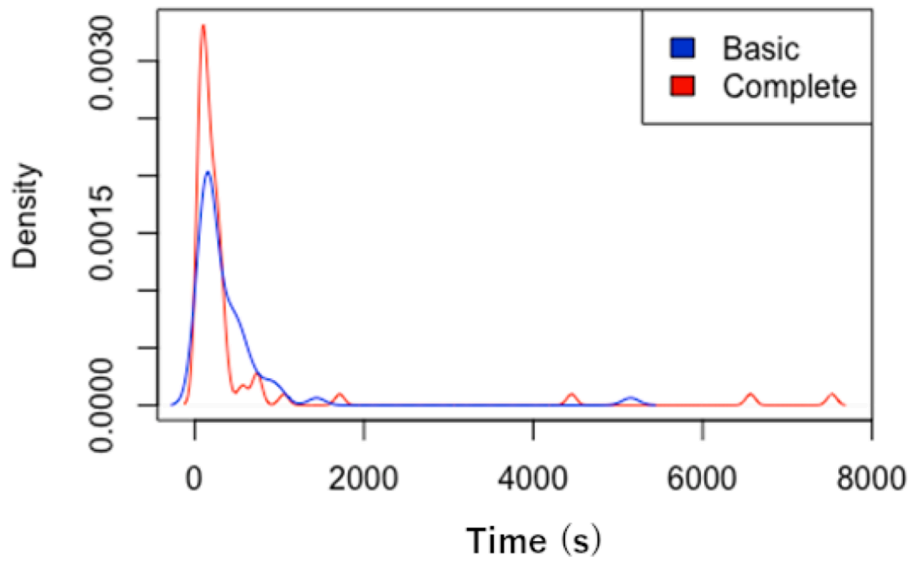

Fig. 11. Time needed to answer the surveys using basic (without NLP) and complete documentation (generated with NLP by our approach).

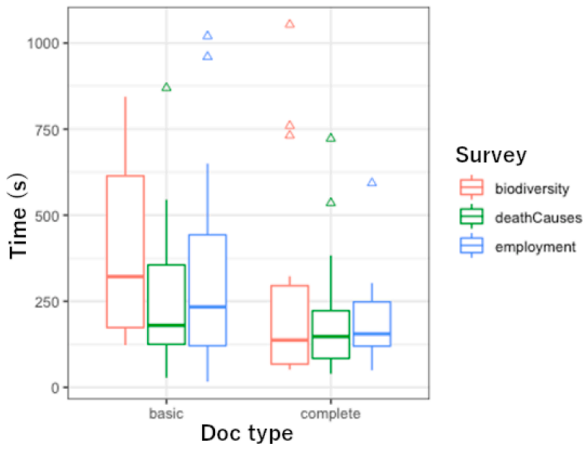

Fig. 12. Density plot of time to answer different surveys with basic and complete documentation (generated with NLP by our approach).

performance. Finally, other related works that use NLG [20,21,33–37] do not address the improvement of Web APIs and their documentation with descriptions understandable by humans.

Therefore, we have evaluated with examples and users that the proposal successfully achieves the objective of generating the suitable Web API descriptive documentation in different situations. The importance of including natural language descriptions in OpenAPI documentation is that it actually helps users to reuse existing data and citizens to be able to access the data offered on the Web.

5. Conclusions and future work

In this paper we have presented an approach that integrates NLP techniques to generate interactive documentation of open data Web APIs, easy to read and understand by data enthusiasts users. Our approach starts with a basic machine-readable API documentation in the OpenAPI standard, which is easy to process by machines but difficult to understand by common users, hindering the reuse of data. From this documentation, we propose a natural language description generator to create a set of descriptions in natural language and append them to the existing machine-readable documentation of the API. This process is based in NLP and specifically in NLG techniques that allow us to create NL descriptions from important concepts of the initial documentation. Moreover, this documentation is made interactive by presenting it as a Web interface to facilitate even more the task of querying the API. For evaluating the proposed approach, we tested the automatic generation

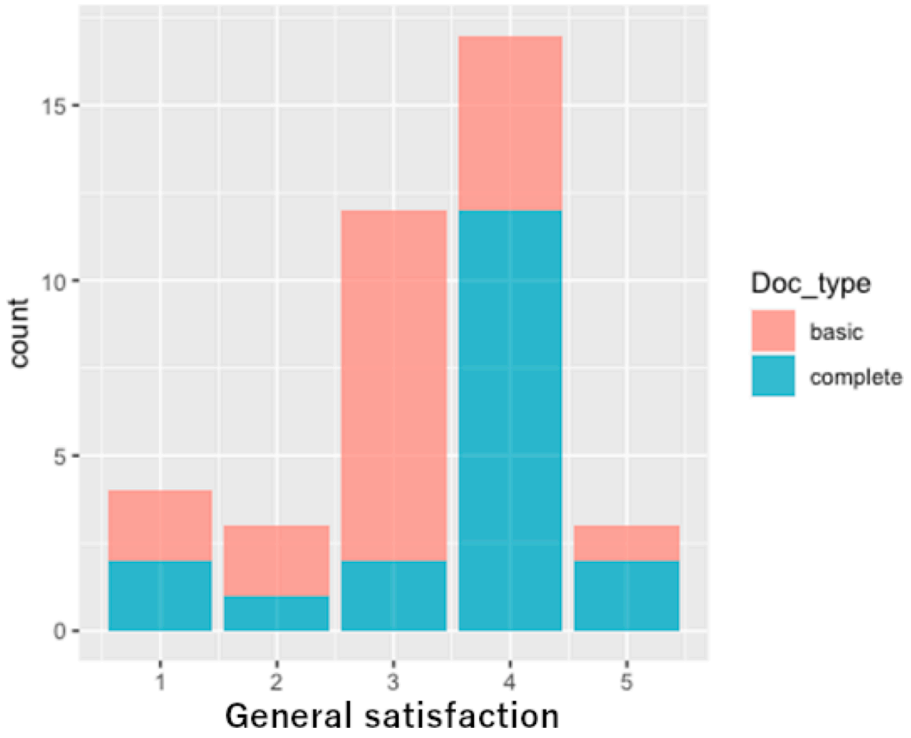

Fig. 13. General satisfaction of participants using basic original and complete NL documentation.

of documentation process with several machine-readable documentations of datasets from the Data.gov open data portal. After that, we have presented a case study in which our approach is applied within a specific scenario and then evaluated with data enthusiasts in an experiment. In this sense, we illustrated and described a real-based situation where users are interested in obtaining specific data from an API. The proposed approach (publicly available at GitHub[1]) is a key element for improving data management and analysis. This is because enhanced API documentation would lead to a better understanding of the API by users, facilitating the access and handling of open data available online. Regarding this implementation of the approach, the generation of enriched descriptions is independent of the API generation, being this step easily applied to existing APIs from different contexts, as well as the API generation process can be applied from different data sources available online. Moreover, the interactivity of the proposed approach allows users to easily access the documentation's webpage and directly query the underlying API by performing a few clicks. Therefore, the implementation allows to easily use and extrapolate the solution for existing problems regarding documentation of APIs.

Limitations of our approach mainly come from the challenges that NLG embraces. As stated in [52], it is necessary not only to describe how the NLG is used for a specific task, but also to discuss its limitations to show the complexity of the task and the challenges that need to be addressed. The first limitation of our approach is related to the unavailability of metadata found in open data sources [53]. This may prevent NLG from generating satisfactory sentences for useful Web API documentation. The use of external knowledge resources could improve this limitation. Another important limitation of NLG is related to hallucination from the data [54], i.e., the generation of texts that are apparently well written but they are unsubstantiated and not faithful to the provided data. In our approach, templates are used to try to overcome this limitation when documentation of Web APIs is generated from open data sources.

As future work, the generation process will be extended by using semantic Web technologies to apply data integration mechanisms. With regard to the generation of descriptions within the API documentation, the NLP area also provides with techniques that allow the adaptation or customisation of the generated descriptions depending on the user needs. In this sense, the descriptions could be simplified according to a specific linguistic level or could also include more technical terms if required. In addition to this, this approach could be easily extended to other languages (i.e., multilingual) since the semantic resources employed (i.e., BabelNet and Babelfy) are linked to many languages, which would facilitate the reuse of code. Finally, studying how our proposal could impact the software industry is an interesting avenue for future work, as according to [27] the poor documentation of APIs for accessing open data causes two main problems for entrepreneurs and companies: on the one hand, the lack of documentation hinders the effort to design and develop open data-based services and software products; and on the other hand, even if the documentation exists, software developers have difficulties in understanding and seeing what open data the APIs can access.

CRediT authorship contribution statement

César González-Mora: Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Cristina Barros:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Irene Garrigós:** Supervision, Project administration, Funding acquisition, Conceptualization, Investigation, Writing – original draft, Writing – review & editing. **Jose Zubcoff:** Supervision, Formal analysis, Conceptualization, Investigation, Visualization, Data curation, Writing – original draft, Writing – review & editing. **Elena Lloret:** Supervision, Funding acquisition, Conceptualization, Investigation, Writing – original draft, Writing – review & editing. **Jose-Norberto Mazón:** Supervision, Funding acquisition, Conceptualization, Investigation, Writing – original draft, Writing – review & editing.

Declaration of Competing Interest

None.

Acknowledgment

This work has been partially funded by the following projects: GVA-COVID19/2021/103, TIN2016-78103-C2-2-R, PROMETEU/2018/089, RTI2018-094653-B-C22, RTI2018-094649-B-I00, TIN2017-90773-REDT and COST Action CA18231.

Appendix A. Result tables

This appendix shows an extract of the evaluation results.

Appendix B. Survey example used in the experiment (API employment survey)

This appendix corresponds to an example of survey for the Employment API with interactive NL documentation.

This survey will evaluate the use of an API to obtain employment data for New York City. Before starting the survey, it is recommended that you go to the following address to read and use the API documentation: https://wake.dlsi.ua.es/EmploymentAPI/docs/.

Please read this documentation carefully before starting the survey in order to be able to answer the questions asked in the survey.

Thank you very much in advance for your cooperation.

Specific questions:

1. Using the documentation provided (https://wake.dlsi.ua.es/EmploymentAPI/docs/): Get the name associated with area 2.
   Please enter the query used to obtain the answer to the above question.
2. Using the documentation provided (https://wake.dlsi.ua.es/EmploymentAPI/docs/): Get the median salary for workers whose professional title is ”Financial Managers” and the name of their area is ”Long Island Region”.
   Please enter the query used to obtain the answer to the above question.
3. Using the documentation provided (https://wake.dlsi.ua.es/EmploymentAPI/docs/): Obtain the professional title of workers from ”Western New York Region”, whose occupational code is 29–2071 and earn on average 45080.
   Please enter the query used to obtain the answer to the above question.
4. Using the documentation provided (https://wake.dlsi.ua.es/EmploymentAPI/docs/): Get the number of ”Computer Programmers” from ”Long Island Region”.
   Please enter the query used to obtain the answer to the above question.
5. Using the documentation provided (https://wake.dlsi.ua.es/EmploymentAPI/docs/): Get the type of area where the average salary for ”Natural Sciences Managers” is 144370.
   Please enter the query used to obtain the answer to the above question.
6. Using the documentation provided (https://wake.dlsi.ua.es/EmploymentAPI/docs/): Get the name of the job whose area is 1 and has 890 employees.
   Please enter the query used to obtain the answer to the above question.

General questions

1. Did you have previous experience in using REST APIs? (Yes or No)
2. Difficulty of the consultations carried out in this experiment (from 1 to 5, being 1easy and 5 difficult).
3. Do you think you have answered the questions correctly? Please tick the options that apply (Yes, No, Some, Don’t know)
4. Has the documentation provided helped you to better understand the API and the data it provides? (from 1 to 5, being 1 not at all helpful and 5 very helpful).
5. Have you used the documentation provided to perform the API queries? (Yes or No)
6. Does the documentation provided include sufficient and clear information? (Yes or No)
7. Is the format of the documentation provided clear? (Yes or No)
8. Do you consider it necessary to have interactive and natural language documentation, such as the one provided for this survey? (Yes or No)
9. In general, are you satisfied with the information provided by the API documentation? (from 1 to 5, being 1 not at all and 5 fully satisfied)
10. What improvements or changes would you propose? (optional)

References

[1] F.T. Neves, M. de Castro Neto, M. Aparicio, The impacts of open data initiatives on smart cities: a framework for evaluation and monitoring, Cities 106 (2020).
[2] M.S. Altayar, Motivations for open data adoption: an institutional theory perspective, Gov. Inf. Q. 35 (4) (2018) 633–643.
[3] P. Hanrahan, Analytic database technologies for a new kind of user: The data enthusiast. Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 2012, pp. 577–578.
[4] K. Morton, M. Balazinska, D. Grossman, R. Kosara, J. Mackinlay, Public data and visualizations: how are many eyes and tableau public used for collaborative analytics? SIGMOD Rec. 43 (2) (2014) 17–22.
[5] D. Roman, N. Nikolov, A. Putlier, D. Sukhobok, B. Elvesæter, A. Berre, X. Ye, M. Dimitrov, A. Simov, M. Zarev, et al., Datagraft: one-stop-shop for open data management, Semant. Web 9 (4) (2018) 393–411.
[6] M. Lnenicka, A. Nikiforova, Transparency-by-design: what is the role of open data portals? Telemat. Informat. 61 (2021) 101605.
[7] E. Daga, L. Panziera, C. Pedrinaci, A BASILar approach for building Web APIs on top of SPARQL endpoints. Proceedings of the 3rd Workshop on Services and Applications over Linked APIs and Data volume 1359, 2015, pp. 22–32.
[8] R. Mchov, M. Lnnika, Evaluating the quality of open data portals on the national level, J. Theoret. Appl. Electron. Commer. Res. 12 (1) (2017) 21–41.
[9] S. Neumaier, J. Umbrich, A. Polleres, Automated quality assessment of metadata across open data portals, J. Data Inform. Qual. 8 (1) (2016).
[10] K. Braunschweig, J. Eberius, M. Thiele, W. Lehner, The state of open data - limits of current open data platforms. Proceedings of the 21st WWW Conference, 2012.
[11] A. Abelló Gamazo, C.P. Ayala Martínez, C. Farré Tost, C. Gómez Seoane, M. Oriol Hilari, Ó. Romero Moral, A Data-driven approach to improve the process of data-intensive API creation and evolution. Proceedings of the Forum and Doctoral Consortium Papers Presented at CAiSE 2017, 2017, pp. 1–8.
[12] M.P. Robillard, R. DeLine, A field study of API learning obstacles, Empirical Software Engineering 16 (6) (2011) 703–732.
[13] D.A. Keim, Information visualization and visual data mining, IEEE Trans. Vis. Comput. Graph. 8 (1) (2002) 1–8.
[14] P.J. Danielsen, A. Jeffrey, Validation and interactivity of web API documentation. IEEE ICWS, 2013, pp. 523–530.
[15] R. Koi, X. Franch, P. Jovanovic, A. Abell, A data-driven approach to measure the usability of web apis. 2020 46th Euromicro Conference on SEAA, 2020, pp. 64–71.
[16] H. Ed-douibi, J.L. Cánovas Izquierdo, J. Cabot, Example-driven web API specification discovery. Modelling Foundations and Applications, 2017, pp. 267–284.
[17] Survey of the State of the Art in Human Language Technology, in: R. Cole (Ed.), Cambridge University Press, New York, USA, 1997.
[18] J. Bateman, M. Zoch, Natural Language Generation, Oxford University Press, 2003.
[19] P. Suter, E. Wittern, Inferring web api descriptions from usage data. 3rd IEEE Workshop on Hot Topics in Web Systems and Technologies, 2015, pp. 7–12.
[20] B. Hancock, H. Lee, C. Yu, Generating titles for web tables. The World Wide Web Conference, 2019, pp. 638–647.
[21] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, K. Vijay-Shanker, Automatic generation of natural language summaries for Java classes. 21st International Conference on Program Comprehension, 2013, pp. 23–32.
[22] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, A. Paradkar, Inferring method specifications from natural language API descriptions. Proceedings of the 34th ICSE, 2012, pp. 815–825.
[23] H. Cao, J.-R. Falleri, X. Blanc, Automated generation of REST API specification from plain HTML documentation, in: M. Maximilien, A. Vallecillo, J. Wang, M. Oriol (Eds.), Service-Oriented Computing, 2017, pp. 453–461.
[24] Y. Lu, G. Li, Z. Zhao, L. Wen, Z. Jin, Learning to infer API mappings from API documents, in: G. Li, Y. Ge, Z. Zhang, Z. Jin, M. Blumenstein (Eds.), Knowledge Science, Engineering and Management, 2017, pp. 237–248.
[25] A. De Renzis, M. Garriga, A. Flores, A. Cechich, C. Mateos, A. Zunino, A domain independent readability metric for web service descriptions, Comput. Stand. Interface. 50 (2017) 124–141.

[26] C. González-Mora, C. Barros, I. Garrigós, J. Zubcoff, E. Lloret, J.-N. Mazón, Applying natural language processing techniques to generate open data web APIs documentation. ICWE, 2020, pp. 416–432.
[27] G. Smith, J. Sandberg, Barriers to innovating with open government data: exploring experiences across service phases and user types, Inf. Polity 23 (3) (2018) 249–265, https://doi.org/10.3233/IP-170045.
[28] J. Kopecký, T. Vitvar, C. Pedrinaci, M. Maleshkova, RESTful Services with Lightweight Machine-readable Descriptions and Semantic Annotations, 2011, pp. 473–506.
[29] H. Ed-douibi, J.L. Cánovas Izquierdo, G. Daniel, J. Cabot, A model-based chatbot generation approach to converse with open data sources. Web Engineering, 2021, pp. 440–455.
[30] G. Uddin, M.P. Robillard, How API documentation fails, IEEE Software 32 (4) (2015) 68–75.
[31] M. Maleshkova, C. Pedrinaci, J. Domingue, Investigating Web APIs on the World Wide Web. ECOWS, 2010, pp. 107–114.
[32] I. Koren, R. Klamma, The exploitation of openapi documentation for the generation of web frontends. Companion Proceedings of the The Web Conference 2018, 2018, pp. 781–787.
[33] H. Hardy, A. Vlachos, Guided neural language generation for abstractive summarization using abstract meaning representation. Proceedings of the EMNLP Confrerence, 2018, pp. 768–773.
[34] C. Huang, O. Zaiane, A. Trabelsi, N. Dziri, Automatic dialogue generation with expressed emotions. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies volume 2, 2018, pp. 49–54.
[35] I. Macdonald, A. Siddharthan, Summarising news stories for children. Proceedings of the 9th INLG Conference, 2016, pp. 1–10.
[36] N. Alharbi, Y. Gotoh, Natural language descriptions for human activities in video streams. Proceedings of the 10th INLG conference, 2017, pp. 85–94.
[37] A. Ramos-Soto, J. Janeiro, J. Alonso, A. Bugarin, D. Berea-Cabaleiro, Using fuzzy sets in a data-to-text system for business service intelligence. EUSFLAT Conference, 2017, pp. 220–231.
[38] C. Gonzlez-Mora, D. Toms, I. Garrigs, J.J. Zubcoff, J.N. Mazn, Model-driven development of web APIS to access integrated tabular open data, IEEE Access 8 (2020) 202669–202686.
[39] C. González-Mora, I. Garrigós, J. Zubcoff, J.-N. Mazón, Model-based generation of web application programming interfaces to access open data, Journal of Web Engineering (2020) 194–217.
[40] H. Ed-Douibi, J.L.C. Izquierdo, J. Cabot, OpenAPItoUML: a tool to generate UML models from OpenAPI definitions. International Conference on Web Engineering, 2018, pp. 487–491.
[41] R. Rodríguez, R. Espinosa, D. Bianchini, I. Garrigós, J.-N. Mazón, J.J. Zubcoff, Extracting models from web API documentation. International Conference on Web Engineering, 2012, pp. 134–145.
[42] A. Gatt, E. Krahmer, Survey of the state of the art in natural language generation: core tasks, applications and evaluation, J. Artif. Int. Res. 61 (1) (2018) 65–170.
[43] C. Van der Lee, E. Krahmer, S. Wubben, PASS: a Dutch data-to-text system for soccer, targeted towards specific audiences. Proceedings of the 10th INLG Conference, 2017, pp. 95–104.
[44] N. Braun, M. Goudbeek, E. Krahmer, The multilingual affective soccer corpus (MASC): Compiling a biased parallel corpus on soccer reportage in English, German and Dutch. Proceedings of the 9th INLG Conference, 2016, pp. 74–78.
[45] G. Trivino, A. Sanchez, A.S. Montemayor, J.J. Pantrigo, R. Cabido, E.G. Pardo, Linguistic description of traffic in a roundabout. International Conference on Fuzzy Systems, 2010, pp. 1–8.
[46] J.M. Alonso, A. Ramos-Soto, C. Castiello, C. Mencar, Explainable AI beer style classifier. The SICSA Reasoning, Learning and Explainability Workshop, 2018.
[47] B. Aysolmaz, H. Leopold, H.A. Reijers, O. Demirörs, A semi-automated approach for generating natural language requirements documents based on business process models, Inf. Softw. Technol. 93 (2018) 14–29.
[48] A. Moro, A. Raganato, R. Navigli, Entity linking meets word sense disambiguation: a unified approach, Trans. Assoc. Comput. Linguist. 2 (2014) 231–244.
[49] R. Navigli, S.P. Ponzetto, Babelnet: the automatic construction, evaluation and application of a wide-Coverage multilingual semantic network, AIJ 193 (2012) 217–250.
[50] C. Fellbaum, WordNet: An Electronic Lexical Database (Language, Speech, and Communication), MIT Press, 1998.
[51] I. Iacobacci. Neural-grounded semantic representations and word sense disambiguation: a mutually beneficial relationship, 2018. Ph.D. thesis.
[52] E. van Miltenburg, M. Clinciu, O. Dusek, D. Gkatzia, S. Inglis, L. Leppänen, S. Mahamood, E. Manning, S. Schoch, C. Thomson, L. Wen, Underreporting of errors in NLG output, and what to do about it, in: A. Belz, A. Fan, E. Reiter, Y. Sripada (Eds.), Proceedings of the 14th International Conference on Natural Language Generation, INLG 2021, Aberdeen, Scotland, UK, 20–24 September, 2021, Association for Computational Linguistics, 2021, pp. 140–153.
[53] C. Dong, Y. Li, H. Gong, M. Chen, J. Li, Y. Shen, M. Yang, A survey of natural language generation, 2021. https://arxiv.org/abs/2112.11739. 10.48550/ARXIV.2112.11739.
[54] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, P. Fung, Survey of hallucination in natural language generation, CoRR abs/2202.03629 (2022).