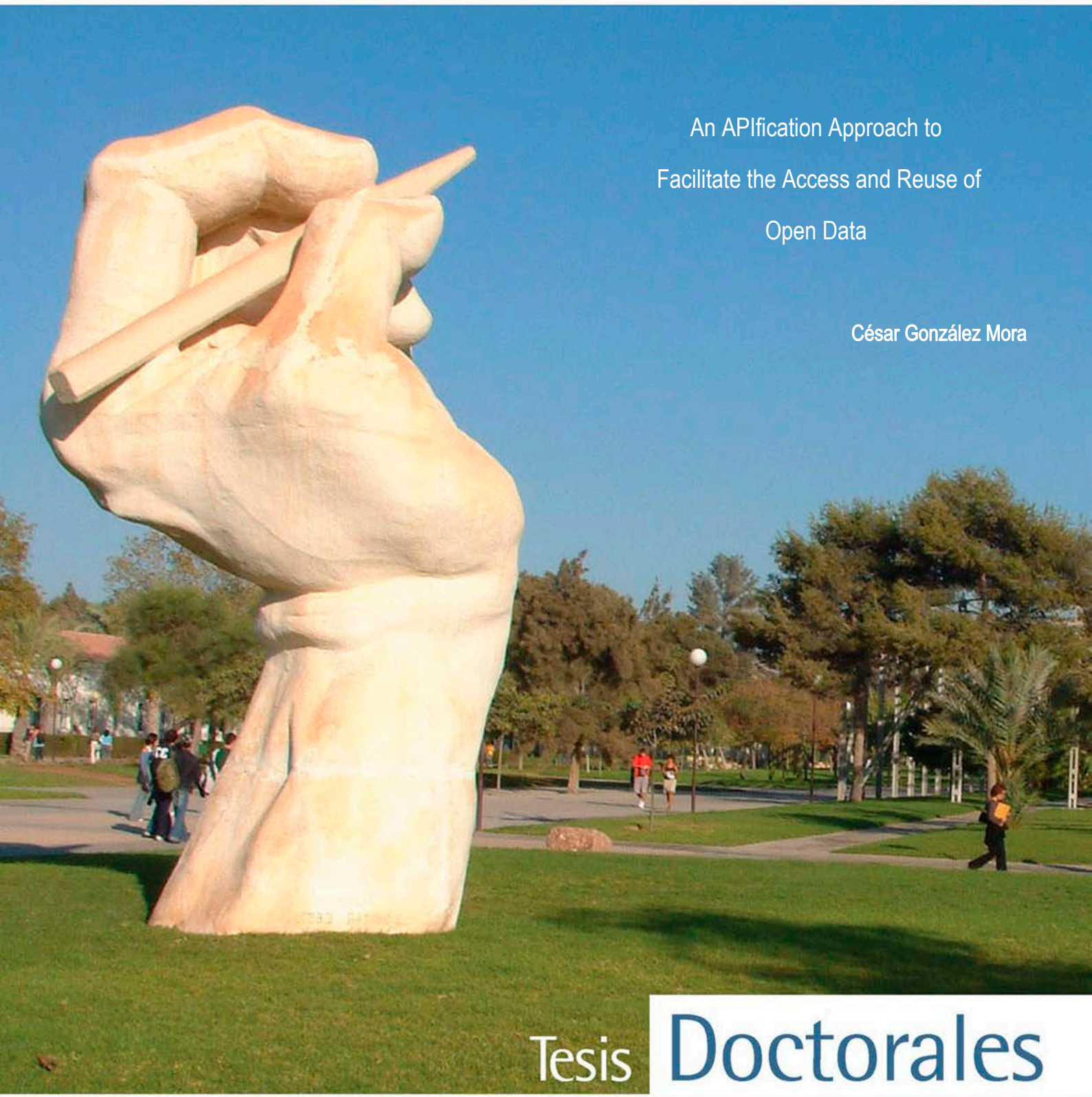
Universitat d'Alacant
Universidad de Alicante



An APIfication Approach to Facilitate the Access and Reuse of Open Data

César González Mora

Tesis Doctorales
UNIVERSIDAD de ALICANTE

Unitat de Digitalització UA
Unidad de Digitalización UA

Universitat d'Alacant
Universidad de Alicante

Departamento de Lenguajes y Sistemas Informáticos
Escuela Politécnica Superior

An APIfication Approach to Facilitate the Access and Reuse of Open Data

César González Mora

Tesis presentada para aspirar al grado de

DOCTOR O DOCTORA POR LA UNIVERSIDAD DE ALICANTE
MENCIÓN DE DOCTOR O DOCTORA INTERNACIONAL
DOCTORADO EN INFORMÁTICA

Dirigida por

Dr. Irene Garrigós Fernández
Dr. Jose Jacobo Zubcoff Vallejo

Esta tesis ha sido financiada por la Universidad de Alicante mediante un contrato destinado a la formación predoctoral, y por la Generalitat Valenciana mediante una subvención para la contratación de personal investigador de carácter predoctoral (ACIF2019).

Para mi familia
por su apoyo
incondicional.

Agradecimientos

En primer lugar, gracias a mis directores de tesis Irene Garrigós y Jose Jacobo Zubcoff por su esfuerzo y dedicación durante esta aventura que ha sido el doctorado, espero que este no sea el final del camino. Gracias a Irene por ser primero mi tutora del Trabajo Final de Grado, que fue uno de los motivos principales que me llevaron a empezar este doctorado, y por su inestimable ayuda y oportunidades que me ha dado todo este tiempo. Y gracias a Jacobo por unirse a este largo recorrido que me ha ayudado a realizar, como si de la escalada de una montaña se tratase, paso a paso.

Todo el trabajo de investigación desarrollado ha sido como parte del grupo de investigación WaKe (Web and Knowledge research group), junto a mis directores de tesis y Jose-Norberto, quien también me ha ayudado siempre que lo he necesitado y le agradezco su enorme aportación a esta tesis doctoral. Destacar también al Departamento de Lenguajes y Sistemas Informáticos y a la Universidad de Alicante que me han facilitado la realización del doctorado.

A todos los coautores que han aportado su granito de arena en este doctorado, con los que es un placer trabajar e investigar, como David, Elena, Sven, Sergio, Diego, Manuel y un largo etcétera.

Al Instituto Tecnológico de Karlsruhe por permitirme realizar una estancia de 3 meses, a York Sure-Vetter por su amable invitación dentro de su grupo de investigación, y a todos sus miembros que me acogieron como uno más. Además, a todos los amigos con los que compartí tiempo allí, como Max, Matthias Frank y el grupo de ESN.

A todos mis amigos que han vivido conmigo este duro proceso, especialmente Álvaro y Campos. Y a mis colegas del Grado en Ingeniería Informática del grupo ARA y del Máster, especialmente al Team Rocket, Carlos, Javi y el resto que me han ayudado. Además, a todos los compañeros de laboratorio que me han apoyado, especialmente Cristina con la que más tiempo he compartido.

A mi familia que siempre me apoya: mis tías y tíos, abuelos, primos, mi hermano Andrés y cuñada, mis suegros y May. A Roy y a Nico. A los que están y los que no han podido estar, pero que sé que me han acompañado.

A mis padres, Luis y Ana, que han estado pendientes de mi progreso y que se alegran mucho de mis éxitos. Siempre me han ayudado, no solo a superar cada obstáculo del doctorado, sino también de la vida.

Y por último, quiero agradecerle especialmente a mi pareja Paula que ha sido una partícipe más de mi doctorado, mejorando siempre todo lo que hago.

Contents

Part I

Preamble

Chapter 1

Preface

Given that the research conducted during the PhD has been published in international peer-reviewed journals, this dissertation is configured as a thesis by publication, which means that the main part of the work (published works part) presents the papers as reprints of such publications in their original format. The set of publications in chronological order are:

1. César González Mora, Irene Garrigós, Jose Zubcoff, and Jose-Norberto Mazón. Model-based Generation of Web Application Programming Interfaces to Access Open Data. Journal of Web Engineering, pages 194–217, 2020.

2. César González-Mora, David Tomás, Irene Garrigós, José Jacobo Zubcoff, and Jose-Norberto Mazón. Model-Driven Development of Web APIs to Access Integrated Tabular Open Data. IEEE Access, 8:202669–202686, 2020.

3. Paloma Cáceres García De Marina, José María Cavero Barca, Carlos E. Cuesta, Miguel Ángel Garrido, Irene Garrigós, César González-Mora, Jose-Norberto Mazón, Almudena Sierra-Alonso, Belén Vela, and José Jacobo Zubcoff. Open Data Consumption Through the Generation of Disposable Web APIs. IEEE Access, 9:76354–76363, 2021.

The complete structure of this dissertation is organised as follows:

- **Contents**: the index of the thesis, with the suitable links to the different parts and the number of pages.

- **Preamble**: includes this preface with initial explanations of the thesis, then a summary of the research is presented, and finally, more details are given in the introduction and proposals chapters.

- **Published works**: the compilation of papers that have been published to conform this thesis.
- **Closure**: part that presents the conclusions of the thesis with a discussion of the research done, including ongoing and future work.
- **Appendices**: Spanish summary of the presented thesis, including the introduction, proposals, published works summary and conclusions.
- **References**: the bibliography used in Preamble and Closure parts of this thesis, as the references of each published work are contained in their related section within the corresponding paper.

Chapter 2

Summary

Nowadays, there is a tendency to publish data on the Web, due to the benefits it brings to the society and the new legislation that encourage the opening of data. These collections of open data, also known as datasets, are typically published in open data portals by governments and institutions around the world in order to make it open – available on the Web in a free and reusable manner. The common behaviour tends to be that publishers expose their data as individual tabular datasets.

Open data is considered highly valuable because promoting the use of public information produces transparency, innovation and other social, political and economic benefits. Especially, this importance is also considerable in situational scenarios, where a small group of consumers (developers or data scientists) with specific needs require thematic data for a short life cycle. In order that these data consumers easily assess whether the data is adequate for their purpose, there are different mechanisms. For example, SPARQL endpoints have become very useful for the consumption of open data, and particularly, Linked Open Data (LOD). Moreover, in order to access open data in a straightforward manner, Web Application Programming Interfaces (APIs) are also a highly recommended feature of open data portals.

However, accessing open data is a difficult task since current open data platforms do not generally provide suitable strategies to access their data. On the one hand, accessing open data through SPARQL endpoints is a difficult task because it requires knowledge in different technologies, which is challenging especially for novice developers. Moreover, LOD is not usually available since most used formats in open data portals are tabular. On the other hand, although providing Web APIs would facilitate developers to easily access open data for reusing, there is a lack of suitable Web APIs in open data portals. Moreover, in most cases, the currently available APIs only allow to access catalog's metadata or to download entire data resources (i.e. coarse-grain access to data), hampering the reuse of data. In addition, as open data is commonly

published individually, without considering potential relationships with other datasets, reusing several open datasets together is not a trivial task. Thus, it requires mechanisms that allow data consumers to integrate and access tabular open data published on the Web. Therefore, open data is not being fully exploited because of its difficult access.

As the access to open data is thus still limited for end-users, particularly those without programming skills, we propose a model-based approach to automatically generate Web APIs from open data. This proposal, which we have entitled "APIfication", takes into account the access to multiple integrated tabular datasets and the consumption of data in situational scenarios. Firstly, we focus on data that can be integrated by means of join and union operations. Then, we coin the term disposable Web APIs as an alternative mechanism for the consumption of open data in situational scenarios. These disposable Web APIs are created on-the-fly to be used temporarily by a user to consume specific open data.

Accordingly, the main objective is to provide suitable mechanisms to easily access and reuse open data on the fly and in an integrated manner, solving the problem of difficult access through SPARQL endpoints for most data consumers and the lack of suitable Web APIs with easy access to open data. With this approach, we address both open data publishers and consumers, as long as the publishers will be able to include a Web API within their data, and data consumers or reusers will be benefited in those cases that a Web API pointing to the open data is missing. The results of the experiments conducted led us to conclude that users consider our generated Web APIs as easy to use, providing the desired open data, even though coming from different datasets and especially in situational scenarios.

Chapter 3

Introduction

In this chapter, the context is first presented regarding the whole research performed during the PhD. This context involves the tendency of opening data by global governments, the benefits that open data brings to the society, the availability of open data and its access through Web APIs or Semantic Web technologies. In addition to the context, the problems related to the access to open data are then presented. These problems are detailed to point out the lack of Web APIs, the difficulty in using Semantic Web Technologies, the low availability of SPARQL endpoints in open data portals, the shortage of mechanisms to access integrated data and the specific problems in situational scenarios.

3.1 Context

Worldwide governments and organisations are nowadays making their data available online [5, 8]. This open data is published in Web portals to be freely accessible and reusable [11], as long as it is considered highly valuable and there is a great deal of awareness in most countries about open data [36, 6].

The adoption of open data initiatives promotes the use of public information, which mainly leads to significant economic benefits according to several studies [43, 38, 44]. Besides these economic benefits, the opening of data also produces transparency, innovation and other social and political benefits [27]. The increase of open data initiatives is also motivated by the growing pressure imposed by governments [42] with new legislation [1], which force public administrations to offer data to citizens. The potential beneficiaries of the open data are those citizens which are provided with large amounts of data, but also data reusers (individuals and companies) that reuse data to create useful applications, fostering the economy and benefiting the citizenship [46, 38].

In order to publish open data, there are two main options: (i) in open data portals that offer a Web platform with a catalog of mostly tabular-form data, and (ii) as Linked Open Data (LOD) generally available through SPARQL endpoints. On the one hand, as an example of open data portals, among the most popular open data repositories are *data.gov.uk* and *data.gov*, which provide a catalog of data resources from the UK and the USA governments, and the *European Data Portal*[1] that combines several data catalogs from the European Commission. On the other hand, as an example of LOD, the LOD cloud[2] includes the datasets that have been published in the Linked Data format (available altogether through the "LOD-a-lot" queryable dump [21]), and the DBpedia platform that offers structured data from Wikipedia by a SPARQL endpoint[3].

In order to take advantage of open data, as many people as possible should be able to easily access this data. Among the most adopted approaches to access open data are the Web APIs [13], as they are an important and recommended feature of open data platforms, allowing developers to make open data accessible to citizens [11] by building their own applications based on this open data [30]. Regarding LOD, it allows users to access data, in the same way as a database management system is used, by means of query languages such as SPARQL [17]. These interfaces to query data on the Web involve powerful query capabilities such as SPARQL endpoints or direct download of data in RDF format such as data dumps.

[1] `https://data.europa.eu/en`
[2] `https://lod-cloud.net/`
[3] `https://dbpedia.org/sparql/`

3.2 Problems to solve

However, there is a large gap between open data and its access by the society. Although several studies [40, 12] suggest the creation of Web APIs to fill this gap, there is still a lack of suitable APIs to access data from open data platforms around the world [22]. Indeed, only about the 6.6% of datasets on average include an API with direct access to open data, that is, query-level APIs that allow to consult directly the data. In most cases the platforms include an API, such as CKAN-based APIs[4] that follow the DCAT[5] standard, which only provides data catalog metadata or a download link for the entire dataset (coarse-grain access to data). A query-level API with fine-grain access to data makes it easier for developers to provide specific data according to user needs [11], enhancing the process of data reuse. In order to promote the use of these APIs to access open data, a key factor is providing relevant and useful documentation [41, 14], following popular standards such as OpenAPI that allows to easily understand and reuse open data [18]. However, existing APIs do not typically include any proper documentation or any precise specification of the functionality and data they offer [28, 4]. Therefore, understanding how to use these APIs and consequently reusing data is a difficult task.

Additionally, exploring LOD by structured query languages such as SPARQL can be challenging and error-prone, especially for novice developers and end-users [24]. Therefore, data reusers are generally more familiar with REST-like APIs than SPARQL [13]. Average open data consumers mainly lack the skills required to use SPARQL queries and the underlying RDF (Resource Description Framework) data model since they both have a steep learning curve [13].

Furthermore, another problem regarding the use of SPARQL is that LOD is not usually available in open data portals, representing only a 0.5% of the total [34]. Actually, as highlighted in recent and relevant studies [20, 37], these portals are more concerned about tabular formats over LOD, being the most used format with a representation of 46.4% (either direct tabular formats or embedded tabular data).

Per contra, most publishers expose their tabular data as separate datasets. Therefore, publishers share their data on the Web without considering potential relationships with other open data [23]. This scenario provokes that data reusers are required to make an extra effort to integrate different data. Consequently, there is a great need for a mechanism that allows data consumers to integrate and access open data published on the Web.

There are also concrete scenarios where a quick and easy access to open data is especially important. This case is the scenario of situational data, which consists of thematic data required for a short life cycle by a small group of consumers with specific needs [3]. Consequently, in a situational scenario, alter-

[4] `https://docs.ckan.org/en/2.6/api/`
[5] `https://www.w3.org/TR/vocab-dcat/`

native mechanisms are necessary to provide access to open data for data reusers without any knowledge regarding Semantic Web technologies [9, 29]. Thus, it is easier to have a Web API with simple access to open data without wasting time processing tabular data sources or learning how to query a SPARQL endpoint [15].

3.3 Related Works

There is a variety of related research that deal with the topic of open data, LOD and facilitating its access for developers. Firstly, to solve problems regarding the access and reuse of data, considering the difficulty of accessing SPARQL endpoints, the creation of APIs is proposed. However, related works such as [40, 26, 35] generally propose the manual creation of APIs to access specific open and linked data platforms, which is time-consuming. The automatic generation of APIs is also addressed [39, 19], but fine-grain access to open data platforms is not considered, they require specific artefacts which are not commonly available and they do not provide suitable documentation.

Moreover, to the best of our knowledge, there is no solution that tackles situational scenarios in which thematic open data is temporarily used for specific needs. Research [45, 32] that address the exploration of Linked Open Data facilitate the access through different interfaces, but they generally require knowledge in RDF or SPARQL technologies or they only provide access to specific endpoints.

Regarding the integration of data, although data integration challenges have been researched for years with significant progress [25], efforts have been made only on solving specific problems. However, according to Abadi et al. [2] more work is required on researching how to pipeline data integration to cover all the way from raw data to an end-user's desired outcome. For instance, additional metadata is required for developers to know how datasets can be related to each other. Although several recent approaches proposed adding this kind of metadata to tabular datasets by means of annotations [7, 16], this methodology has not being widely adopted by publishers.

Accordingly, as far as the authors are aware, although related works help data reusers to overcome the different problems to access open data, they do not offer a flexible and complete solution that really facilitates developers to obtain open data from different sources to improve the open data reuse process.

Chapter 4

Proposals

One of the most adopted practices to facilitate the access to open data is the deployment of Web APIs on top of open data portals and Linked Data sources [31]. In this sense, we propose an approach to grant access to open data at the query-level (i.e., fine-grain access), including the access to integrated data and in situational scenarios. Therefore, this thesis is focused on facilitating the access to open data, considering a data integration process together with the generation of APIs to simplify the consumption of open data from different datasets in particular scenarios. The Web APIs we propose are created on-the-fly to be also used temporarily by users to consume open data as situational data. These APIs have been denominated as disposable Web APIs because they are suddenly created to allow data consumers to assess whether the data is adequate for their purpose, thus avoiding the complexity and learning curve of SPARQL and the effort of manually processing the data.

The main objective is twofold: (i) encourage open data publishers to include an API within the data to be opened and (ii) help developers to create value from open data, that is, facilitate the reuse of open data and thus promote citizens to access it. In order to fulfil those objectives, we are going to explain the different proposals separately: first the APIfication approach from tabular open data is introduced; then, this APIfication approach is extended with the proposal of integrating open data and accessing it altogether; and finally, the proposal to take advantage of semantic information is applied to the initial APIfication approach to generate disposable Web APIs.

4.1 APIfication approach

First of all, we propose a model-driven APIfication approach in order to achieve the automatic generation of query-level Web APIs with fine-grain access to data. These Web APIs help developers to access and reuse open data, promoting then the usage of open data by the citizenship.

This APIfication process [47] consists of a set of transformations in order to auto-generate Web APIs for open datasets. The process is based on automatic, generic and standardised mechanisms to generate Web APIs with machine-readable documentation following the most popular open source standard: OpenAPI 3.0. Following this standard helps users to understand the functioning of the API and to work with the API as a Web interface. Also, model-driven mechanisms [10] allow us to face up with the heterogeneity of the existing open data sources, integrating the API and its documentation in Model Driven Development processes to standardise the way of creating and defining APIs.

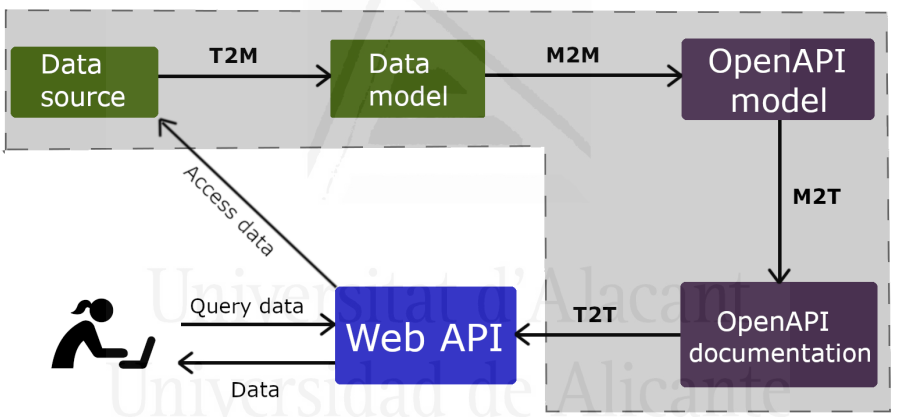


Figure 4.1: Automatic API generation process.

The transformation process for automatically generating Web APIs starts with an open data source. These Web APIs help users to access and reuse the initial data. Moreover, the process also creates OpenAPI documentation to help Web API users, and a model of the data and the documentation in order to take advantage of the large number of existing modeling tools for the integration of these artefacts in model-driven development processes. The whole transformation process from the data source to the Web API - shown in Figure 4.1 - is launched by the automatic generator program, including the following steps: a text to model (T2M) transformation from the data source to the data model, a model to model (M2M) transformation from the data model to the OpenAPI model, a model to text (M2T) transformation from the OpenAPI model to its OpenAPI documentation, and finally a text to text (T2T) transformation from

the OpenAPI documentation to the Web API. When the process has finished, users are able to query the generated Web API, so that the API access the data from the source, and finally, the queried data is returned to the users. Therefore, starting from a dataset containing rows and cells, the system performs a direct transformation to construct a data model object with row and cell objects. Then, the data model is transformed to an OpenAPI model using each cell of the first row as an API component (method, parameter, and property), parsing this OpenAPI model object to a standard format for API documentation. Finally, the complete Web API is automatically created according to the components detailed in its documentation.

With this approach users are able to obtain a Web API from any open data source to easily access the dataset with suitable OpenAPI documentation. More details are presented in Chapter 5.

4.2 Integration approach

In addition to the APIfication approach presented previously, it is worth noting that there can be a previous step in this APIfication process in case data needs to be integrated. This new step consists of integrating similar data so that the generated Web API is able to access open data in an integrated manner.

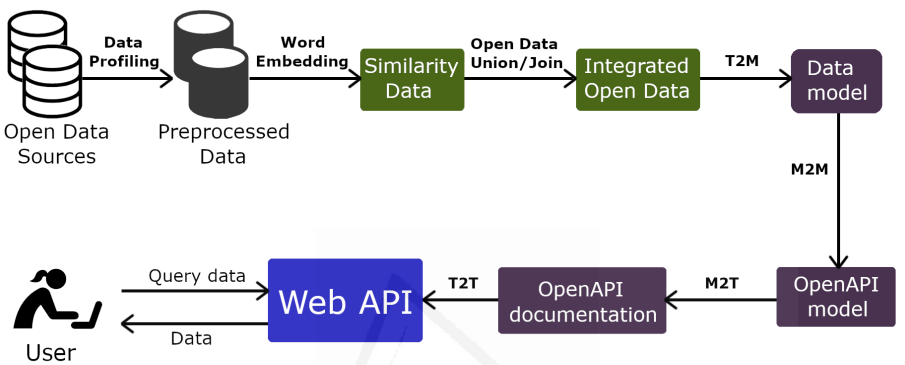


Figure 4.2: Automatic open data integration and API generation process.

The process considering both steps (integration and APIfication) is shown in Figure 4.2. The first step implies detecting unionable and joinable tabular datasets using word embeddings techniques [33] to identify which columns from different input tabular datasets are more likely to be integrated. The next step consists of integrating the datasets by applying join or union operations. Finally, from the previously integrated dataset, a Web API to access this data is generated by means of the previously explained APIfication transformations.

The operators considered for integrating open data are union and join from relational algebra, working as follows:

- Union operator aims to get a unique dataset from two tabular datasets (A and B), containing rows that are in A or in B. A and B must have share columns referring to the same concept.
- Join operator aims to get a unique dataset from two tabular datasets (A and B), including every column from A and B, and containing rows that fulfil a matching condition.

The operation is chosen between union and join, depending on the similarity found between the datasets to integrate. For example, if two datasets are highly similar in only a subset of columns, they are likely to be integrated by means of a join operation. However, if two datasets are similar in almost all the columns, they are more likely to be integrated by a union operation. More details are presented in Chapter 6.

4.3 Semantic approach

Finally, another previous step in the APIfication approach can be added to leverage semantic information from data sources in order to automatically generate easy-to-use disposable Web APIs, which can be used to access open data in situational scenarios.

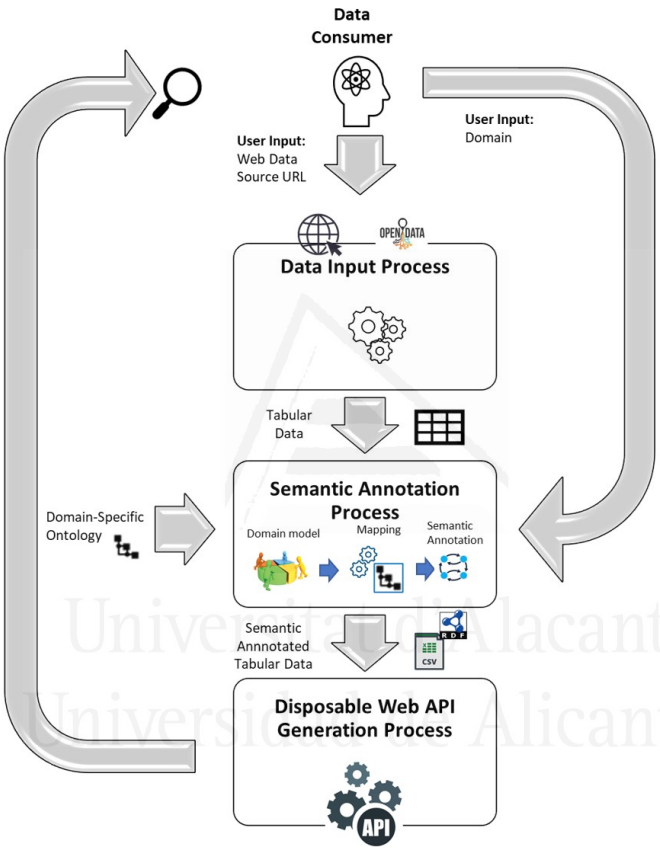


Figure 4.3: Open data consumption process for situational scenarios.

The proposed process for open data consumption through the use of automatically generated disposable Web APIs comprises three stages:

1. Data Input Process: in this stage, the Web data is selected to satisfy data consumer needs in a situational scenario.
2. Semantic Annotation Process: the input data is annotated using a domain-specific ontology, obtaining semantically annotated data.
3. Disposable Web APIs Generation Process: the input of this process is the

previously semantically annotated tabular data, from which disposable Web APIs are automatically generated in order to allow the consumption of data in a situational scenario.

The complete open data consumption process is shown in Figure 4.3, including the inputs and outputs of each process. Regarding the APIfication approach, the new step of annotating the data is added in order to improve the quality of the available data. This step first starts by studying the information contained in the data source and important features of the selected domain. It is then necessary to create a glossary of terms concerning the terminology used in the data source, describing each one, after which a domain model with which to represent these terms and their relationships must be defined in a domain model by using a UML class diagram. After that, it is important to identify the semantics of the information in order to align them with the terms of a reference vocabulary, that is, a vocabulary that the developer needs to choose (e.g. domain-specific ontology). The mappings required in order to establish the correspondence between different information elements from the data source (glossary of terms) and the reference vocabulary (domain-specific ontology) are defined by means of a manual inspection of the input data source. We specifically identify the subjects, predicates and objects of a tabular input file and map them onto elements from the reference vocabulary (e.g. a domain-specific ontology). This mapping is manually defined and added to a configuration file. Once all the mappings have been defined, a custom script is programmed in order to transform the input tabular source data into the semantically annotated data, according to the mappings defined in the configuration file. Then, if the alignment is possible, it is necessary to use the original data, and semantically annotate them using the existing terms from the reference vocabulary (domain-specific ontology) by means of the RDF language using the previously defined mappings. If the alignment is not possible, then we should analyse the difference between the original data and the reference vocabulary in order to extend it. Finally, the last step consists of integrating the semantic dataset with other sources. This process is done by relating elements from the different semantic datasets and discarding duplicated information in the datasets when necessary.

More details are presented in Chapter 7.

4.4 Contributions

In order to reduce the gap between open data and its consumption by the users, we proposed an APIfication approach with the suitable mechanisms to consider open data, data integration and situational data.

Among the contributions of this APIfication approach are:

- The creation of a generic, automatic and model-based process for the generation of Web APIs. This approach aims thus to directly simplify the open data reuse process, which will result in economic benefits to developers and the infomediary sector.
- The implementation of this automatic Web API generation process, which is available online at GitHub[6].
- The allowance for developers to easily create Web APIs by their own, releasing to them the management on how to provide data to the citizenship, such as creating mobile applications that access open data through the automatically generated APIs. Even if they have to publish the API on a server, they still have control over both the data and its access, which also has its benefits. For example, these APIs can be easily customised by developers, if needed, to improve the experience of obtaining data.
- The provision of Web API documentation following the most popular standard (OpenAPI), which facilitates the understanding of the APIs and consequently promoting their use.
- The definition of a word embedding-based similarity measure between tabular datasets. This measure allows identifying joinable and unionable tabular open data in order to facilitate the process of integrating data to be accessed altogether.
- The implementation of the integration and Web API generation processes consecutively in the same pipeline, which is available online at GitHub[7].
- A process for the semantic annotation of tabular data sources from the Web.
- A model-driven approach that can be employed to ingest our semantic annotation process and then generate disposable Web APIs to access data sources.
- The evaluation of our approach through suitable experiments with developers and real tabular open data, generating Web APIs to access specific data, comparing its use with the access by Semantic Web technologies such as SPARQL.

[6] `https://github.com/cgmora12/AG`
[7] `https://github.com/cgmora12/DataIntegration2API`

Part II

Published works

Chapter 5

Model-based Generation of Web Application Programming Interfaces to Access Open Data

César González Mora, Irene Garrigós, Jose Zubcoff, and Jose-Norberto Mazón. Model-based Generation of Web Application Programming Interfaces to Access Open Data. Journal of Web Engineering, pages 194–217, 2020.

Model-based Generation of Web Application Programming Interfaces to Access Open Data

César González-Mora*, Irene Garrigós, Jose Zubcoff and Jose-Norberto Mazón

Department of Languages and Informatics Systems, University of Alicante, Spain
E-mail: cgmora@ua.es; igarrigos@ua.es; jose.zubcoff@ua.es; jnmazon@ua.es
**Corresponding Author*

Received 28 February 2020; Accepted 04 October 2020;
Publication 22 December 2020

Abstract

In order to facilitate the reusing of open data from open data platforms' catalogs, Web Application Programming Interfaces (APIs) are an important mechanism for reusers. However, there is a lack of suitable Web APIs to access data from open data platforms. Moreover, in most cases, the currently available APIs only allow to access catalog's metadata or to download entire data resources (i.e. coarse-grain access to data), hampering the reuse of data. Therefore, we propose a model-based approach to automatically generate Web APIs from open data. Our generated Web APIs facilitate the access and reuse of specific data (i.e., providing fine-grain or query-level access to data), which will result in many societal and economic benefits such as transparency and innovation. With this approach we address open data publishers which will be able to include a Web API within their data, but also open data reusers in case of missing APIs. This APIfication process, which means the creation of APIs for every available dataset, is based on automatic, generic and standardised generation mechanisms. The performance and functioning of this

Journal of Web Engineering, Vol. 19_7–8, 1147–1172.
doi: 10.13052/jwe1540-9589.197810

approach is validated with different datasets, which successfully generates Web APIs that facilitate the reuse of data.

Keywords: Web APIs, open data, data access, data reuse.

1 Introduction

As the World Wide Web has become an important information platform, many organisations are interested in providing information via Internet [2]. Therefore, worldwide governments and organisations are increasingly generating data and making it available online [14], which contributes to the globalisation of information. Promoting the use of public information produces transparency, innovation and other social, political and economic benefits [14]. The potential beneficiaries of these open data initiatives are data reusers (individuals and companies), which can use data to create useful applications or services to grow in economic terms [20] and benefit the wider society [29].

Several studies [20, 27, 28] indicate that the economic potential of open data is significant: it is estimated [27] that public information could generate more than $3 trillion a year of value as a result of open data in different areas of the global economy, such as Education and Transportation. It is also stated [27] that direct and indirect economic benefits for the whole EU economy were about the order of 200 billion euros in 2008. For example, in the United Kingdom the open data program estimated [27] the direct economic benefits of public sector information at around 1.8 billion pounds a year, with an overall impact including direct and indirect benefits of around 6.8 billion pounds. Also, in Spain it is estimated from the Aporta project [20] that there are over 150 companies that work solely on the infomediary sector, generating around 500 million euros annually that can be directly attributed to open data reuse [28]. Besides these economic benefits, the increase of open data initiatives is motivated by the growing pressure imposed by governments [14, 25] with new legislation [1], which force public administrations to offer data to citizens. This open data is commonly offered in catalogs within Web platforms. For instance, *data.gov.uk* provides a catalog of data resources from the UK Government or the European Data Portal provides data catalogs from the European Commission, which are among the most popular open data repositories.

In order to handle open data, Web APIs are a recommended feature of open data platforms [5], allowing developers to build their own applications and bring open data to citizens. Therefore, Web APIs not only enable retrieval

Table 1 Current amount of data and APIs in open data platforms

Open Data Portal	Datasets	Datasets with query-level API	% of datasets with query-level API	Generic Web API
europeandataportal.eu (Europe)	860,000	60,000	7%	No
data.gov (US)	301,000	20,000	6.6%	CKAN API
data.gov.uk (UK)	46,000	200	0.4%	CKAN API
datos.gob.es (Spain)	20,000	2,000	10%	API to download resources
open.canada.ca (Canada)	80,000	40	0.05%	CKAN API
data.gov.au (Australia)	30,700	6,300	20.5%	CKAN API
data.gov.sg (Singapore)	1,500	13	0.9%	CKAN API
govdata.de (Germany)	21,000	200	1%	No
datos.gob.cl (Chile)	3,500	3	0.1%	CKAN API
Average results	150,522	9,862	6.6%	CKAN API

of information, but also facilitate building of versatile applications based on online data [17].

Although organisations and other organisms under the umbrella of smart cities are starting to create public data catalogues [22], the absence of suitable Web APIs to access online data is a common problem in open data platforms around the world, as shown in Table 1. The amount of data offered on the most important data platforms varies between more than a thousand and almost a million datasets. However, only about the 6.6% of datasets, on average, include a query-level API to ease the access to the data, which means a Web API to access directly that data. In most cases, the platforms include an API following the DCAT[1] standard, such as a CKAN API[2], which is oriented to access only data catalog metadata. At most, a download link for entire datasets is also provided (i.e., coarse-grain access to data). A query-level API with fine-grain access to data allows to filter and project data sources in order to get specific data. This type of access makes it easier for developers to provide specific data according to user needs [5] and improve the process of

[1]https://www.w3.org/TR/vocab-dcat/

[2]https://docs.ckan.org/en/2.6/api/

data reuse, but there is still a lack of these kind of query-level APIs in current open data platforms.

Other problem arisen from open data platforms is that their APIs do not typically include any proper documentation or any precise specification of the functionality and data they offer. Therefore, understanding how to use these APIs and consequently reusing data is a difficult task. Moreover, aiming at standardising the way in which Web APIs are specified and documented, the OpenAPI Initiative has been announced by several vendors, such as Google and SmartBear [9]. However, most documentation does not follow a standard that allow to easily understand and reuse open data [9].

In order to tackle these problems, we propose an approach to grant access to open data at the query-level (i.e., fine-grain access). The main objective is to help open data publishers to include an API within the data to be opened. It is also very useful for helping developers to create value from open data, that is, facilitate the reuse of open data and thus promote citizens to access it. With our approach, open data reusers will be able to manage how they provide data to the citizenship, such as creating mobile applications that access open data through the automatically generated APIs. Therefore, the aim is to facilitate access to the data, helping data publishers to provide the API for already published data or for new data, and otherwise for developers who would use the APIs to reuse data from apps or give access to third parties. This gives developers an easy way to reuse data as long as they have not been provided with an API, which usually happens. Even if they have to publish the API on a server, but they have control over both the data and its access, which also has its benefits. For example, these APIs can be easily customised by developers, if needed, to improve the experience of obtaining data.

This approach consists of an APIfication process [30], which means the automatic creation of APIs for every dataset, based on model-driven approach mechanisms [4] which allow us to face up with the heterogeneity of the existing open data sources. Also, model-driven mechanisms allow us to more easily include APIs corresponding definition and documentation following the most popular open source standard: OpenAPI 3.0. Following this standard helps understanding the functioning of the API and supposes an added value since it can be used for other purposes, such as the generation of models that represent the API [10]. The main advantage of using models and metamodels is the integration of the API and its documentation in Model Driven Development processes, which consist of important development artifacts [15] that help in standardising the way of defining APIs, improving the visualisation of such structures using easy to read

interfaces. Using models in software development is highly recommended because models help us understand a complex problem and its potential solutions through abstraction [24]. Therefore, we can take advantage of using models and modeling techniques because they are one of the most fundamental techniques to address challenges in software development such as problem understanding, balancing time and effort, dynamic changes in the development and the management of large projects [3]. It consists of a way to increase the quality, efficiency and predictability of large-scale software development [3].

This article is structured as follows. In Section 2 it is presented the running example used to illustrate the approach explained in Section 3, in which the overview of the APIfication approach is detailed. Then, in Section 4, the approach is validated to test its correct functioning and analyse its performance. Finally, related work is described in Section 5 and the paper concludes in Section 6.

2 Running Example

This section introduces a running example about accessibility in cities, which is used along the paper to illustrate the proposal. One of the most important things that smart cities want to achieve is to optimise the urban accessibility for people with disabilities, which would improve the quality of people's life [18]. For that reason, it is used as a case of study of the automatic Web API generation process.

This running example exposes a situation where a developer wants to create value from open data. In this case, providing this open data through a mobile application, containing information about local businesses that can be accessed by people with disabilities. However, when searching on the Internet for open data about accessible businesses to reuse them in the application, the developer discovers different problems. Firstly, datasets that contain the required data do not include the suitable Web APIs to reuse that data directly (i.e. fine-grain access). And secondly, in case there is an API, it is difficult to know how to use the API because proper documentation following a standard such as OpenAPI is not included.

The example would be equivalent to approaching it from the point of view of the data publisher, which aims to improve access to its information by offering a Web API with query-level capabilities and documentation. For creating such infrastructure, the data publisher can use our proposal that facilitate this task, especially if data publisher lacks programming skills.

Table 2 Extract of businesses accessibility data in CSV format

Decal Recipient	Closed/ Moved	Location	Year	Category	Sub-Category
AMC Showplace 11		1351 S College Mall Rd	1996	Entertainment Venue	Cinema
American Eagle Outfitters		College Mall	1996	Retail	Clothing
Andrew Davis Mens Wear		101 W Kirkwood Ave	2011	Retail	
Applebees Neighborhood Grill and Bar		College Mall	1996	Restaurant/Bar	American

Thus, developers who want to create value from open data will take advantage of the direct access and reuse of data through the auto-generated API.

In this example, the suitable data is offered in the data.gov Portal, where only a CKAN API is available to access metadata or to download entire datasets (i.e. coarse-grain access). The *Accessibility Decal Recipients* dataset[3] contains information about the *AccessAbility Decal program* in order to recognise businesses in the city of Bloomington (United States) that are accessible to people with disabilities. This dataset is available in CSV, which is a simple and well table-structured data format commonly used in open data platforms, with 3 stars in the 5-star open data model[4]. In Table 2 there is an extract of this accessibility data, which includes information such as the name, place and category of the different businesses that are accessible by people with disabilities. The column "Decal Recipient" is the name of the accessible business, the "Closed/Moved" column is filled when a business is closed or moved to another location, the next column "Location" specifies the current place of the business, the opening year is detailed in column "Year", and finally, the "Category" and "Sub-Category" columns consist of a classification of the business area of the establishment, which can be empty.

The application created by the developer should offer a list of accessible businesses, being able to filter by the name of the business, the current situation (if closed or moved), the location within the city of Bloomington, the year when the business opened, the category of the business and the

[3]https://catalog.data.gov/dataset/accessibility-decal-recipients

[4]https://5stardata.info/en/

subcategory. All these filters should be able to be applied both individually and together (combined according to user needs).

This example attempts to demonstrate that developers need query-level Web APIs to easily access and reuse the desired data. We provide an approach to generate these APIs, which could also be used by the publisher of the data to provide also the missing Web APIs. This approach is explained in the next Section.

3 A Model-based APIfication Approach to Access Open Data

In this section a model-driven APIfication approach is presented in order to achieve the automatic generation of query-level Web APIs with fine-grain access to data.

An overview of the automatic generation process is shown in Figure 1. This APIfication process is able to automatically generate a Web API for any dataset chosen by a developer. This transformation process starts with an open data source, from which a complete Web API is generated, including interactive OpenAPI documentation of the API. This auto-generated Web API helps users to access and reuse the initial data, and the generated documentation can be helpful to know how to use the API and even to try it through a Web interface. The automatic generator creates a model of the data and the OpenAPI documentation in order to take advantage of the large number

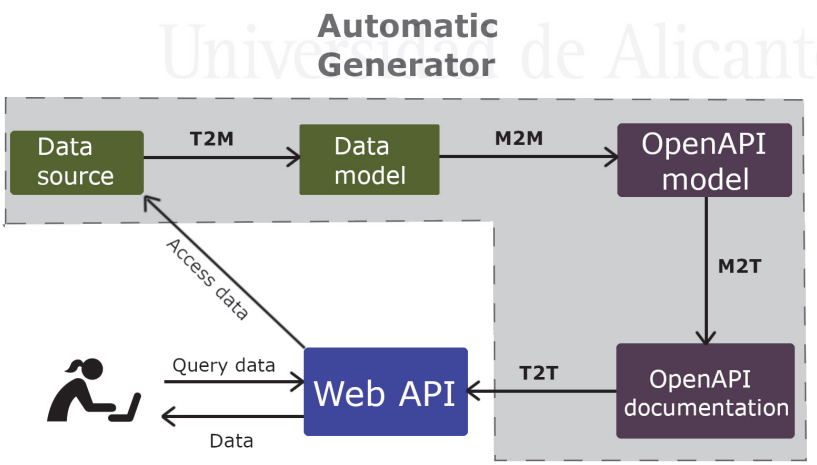


Figure 1 Automatic API generation process.

of existing modeling tools for the integration of these artefacts in model-driven development processes. The whole transformation process from the data source to the Web API is launched by the automatic generator program[5], including the following steps: a text to model (T2M) transformation from the data source to the data model, a model to model (M2M) transformation from the data model to the OpenAPI model, a model to text (M2T) transformation from the OpenAPI model to its OpenAPI documentation, and finally a text to text (T2T) transformation from the OpenAPI documentation to the Web API. When the process has finished, users are able to query the generated Web API, then the API access the data from the source and finally the queried data is returned to the users.

The automatic generator can be used by users without programming skills, such as an open data publisher or an open data reuser. In order to launch this generator, the only requirements are having installed java and nodejs. An example of launching the generator consist of using the following command in the computer's terminal: *java -jar ag.jar csv2api filename*, where "filename" can be the link to an online dataset or the name of a previously downloaded dataset. More information is available at the GitHub page[6].

The transformation process is explained in detail in the following subsections, describing the different stages that are part of the process.

3.1 From Data Source to Data Model (T2M)

The first stage of the transformation process starts from a specific data source, from which a data model is inferred.

The data source is converted into the data model through a text to model (T2M) transformation in order to represent the data and proceed with the model-based transformation approach. This data model consists of a MOF[7]-based model in XMI format according to its metamodel defined by the authors (Figure 2), similar to the one specified in the Eclipse model transformations scenarios[8], which is implemented in the Ecore format from the Eclipse Modeling Framework (EMF). In the metamodel, the relation between the different objects is specified: a CSV file with its filename contains a set of rows, and a row with its position contains a set of cells with value and type. Each cell of the model contains the information of each cell from the

[5] https://github.com/cgmora12/AG/blob/master/ag.jar
[6] https://github.com/cgmora12/AG
[7] https://www.omg.org/mof/
[8] http://www.eclipse.org/atl/atlTransformations/

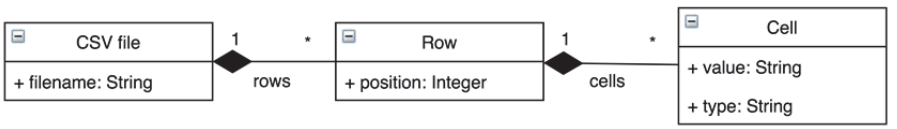


Figure 2 CSV datafile metamodel.

first rows of the CSV file: the first row of the data source will be used by the automatic generator for creating the API methods, properties and parameters; and the second row will be used as example data and also for type inference. This type inference is performed by analysing the data types of a set of values from the dataset second row. We do not analyse the whole data due to performance issues, but in case the type inference is not correct, the data types specified in the API can be changed by developers after the generation process is completed.

In order to work properly, the dataset to apply the automatic generation of API must be in CSV format, which has been chosen in this example because it is one of the most popular formats for publishing data. In case the dataset is in other format the only effort to properly generate the API can be to parse it to CSV, which can be performed by external tools such as Convertio[9]. This CSV must have the information stored in a tabular form separated by commas or semicolons, avoiding strange characters and metadata embedded in the file itself. The first row of the CSV must contain the names of the columns, whereas the other rows contain the different values for each column, always using the same separator between those columns. This first row will be used by the automatic generator to create, for each column name, a method of the API, a parameter for this API method and a property defined in the API documentation. The values obtained from the second row of the dataset will be used by the automatic generator for example values of the API methods, parameters and properties.

Particularly, in the running example the automatic generation program reads the *Accessibility Decal Recipients* dataset, which is a CSV file, and processes it by rows and columns. The data in the first couple of rows is analysed, creating a data model with table, row, and cell objects, as shown in Figure 3. This generated model contains an object “Table”, in which we can find a set of “Rows” containing many “Cells”. The information contained in the first row cells consists of the column names, while the second row cells contain data examples about accessible businesses. An example of

[9]https://convertio.co/en/

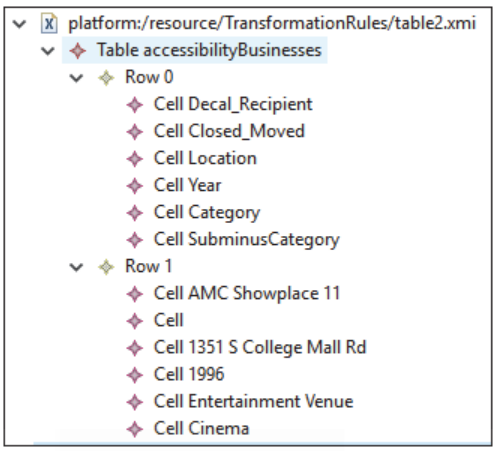


Figure 3 Datafile model in XMI format.

column name is "Decal_Recipient", which is converted into a cell in the first row (Row 0 in Figure 3). This cell and all cells from the first row will be then used as the name of an API method, parameter and property in the following transformation steps. On the other hand, an example of value from of "Decal_Recipient" is "AMC Showplace11", which is converted into a cell in the second row (Row 1 in Figure 3), which will be used as example value in the corresponding API method, parameter and property.

3.2 From Data Model to OpenAPI Model (M2M) and Documentation (M2T)

The second stage in the automatic API generation process is about creating the documentation of the API, which follows the OpenAPI standard and it is based in models.

Once created the data model in the previous stage, a model to model (M2M) transformation between the data model and the OpenAPI model is performed. After that, a model to text (M2T) transformation between the OpenAPI model and the OpenAPI documentation is carried out, obtaining at the end of this stage a complete documentation of the API and its related model.

First, the automatic generation program launches the M2M transformation defined in ATL language. ATL is one of the most widely used model transformation languages, backed by a mature and efficient execution runtime [15], which is used currently [8, 26]. For this reason, a set of transformation

```
-- @atlcompiler emftvm
-- @path Table=/TransformationRules/Table.ecore
-- @path Openapi=/TransformationRules/Openapi.ecore
module Table2Openapi;

create OUT: Openapi from IN: Table;
rule Main {
    from
        s: Table!Table
    using {
            firstTableRow : Sequence(Table!Cell) = s.rows->first().cells;
            lastTableRow : Sequence(Table!Cell) = s.rows->last().cells;
    }
    to
        t: Openapi!API (
            openapi <- '3.0.0',
            info <- openapi_info,
            servers <- Sequence{openapi_servers},
            paths <- Sequence{openapi_basic_path}.union
                (firstTableRow -> collect(e | thisModule.OpenapiPaths(e, s))),
            components <- Sequence{openapi_components}
        ),
        openapi_info: Openapi!Info (
            title <- s.filename,
            version <- '1.0.0',
            description <- 'Obtaining the ' + s.filename
        ),
        openapi_servers: Openapi!Server (
            url <- 'http://www.urlprueba.com/v1'
        ),
```

Figure 4 ATL transformation rules extract.

rules between the data model and the OpenAPI model have been defined using the ATL language, as shown in the extract of the code in Figure 4. The ATL transformation rules start from the Table object defined in its Ecore metamodel, and its rows and cells are used to generate the OpenAPI model and all the different objects contained in its OpenAPI metamodel. Basically, from a data model containing the first rows of a CSV file, the OpenAPI model is generated by creating the different elements of the OpenAPI documentation from these set of cells.

The generated OpenAPI model is in XMI format, as shown in Figure 5. It is based on an OpenAPI metamodel (Figure 6) defined by the authors, in Ecore format. It has been created by updating an existing OpenAPI metamodel[10] from Swagger 2.0 to OpenAPI 3.0 specification. This OpenAPI metamodel contains all the related objects required by OpenAPI. The main parts of this metamodel are: the "API" object containing OpenAPI information about the version and the related "Server" object which specifies the URL

[10]https://github.com/SOM-Research/APIDiscoverer/tree/master/metamodel

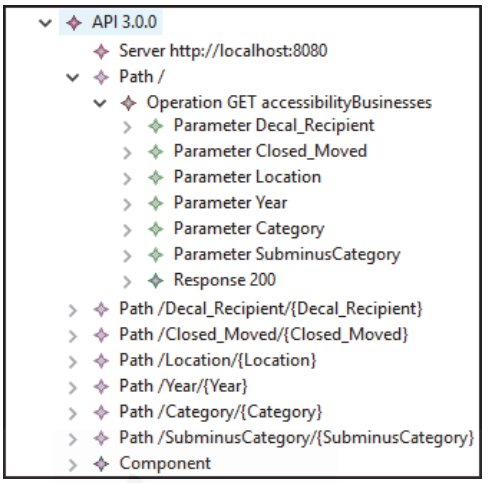


Figure 5 Extract of OpenAPI model (XMI).

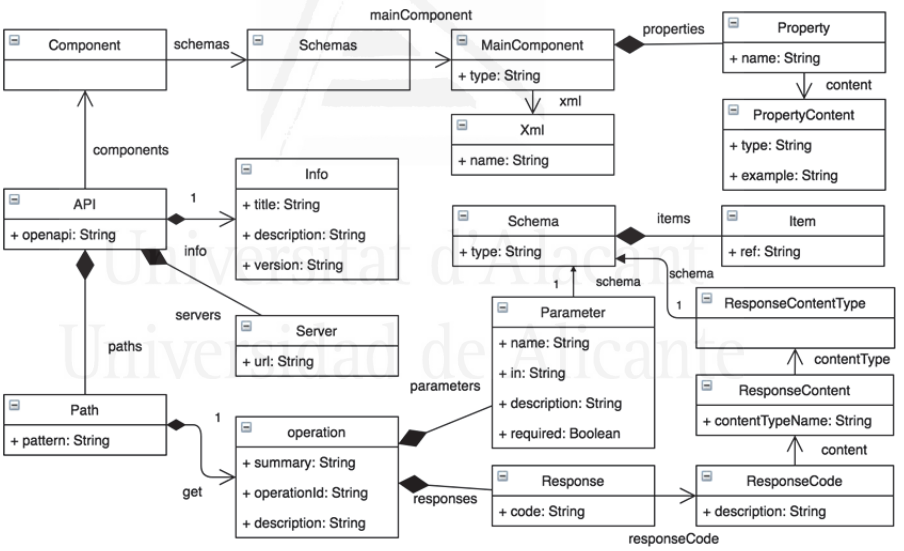


Figure 6 OpenAPI metamodel.

to the Web API; the "Path" object which includes a set of API operations with their parameters and specific response with its code, content and type; and finally, the "Component" object to define the properties of the API within the "Schemas" and "MainComponent" elements. In this case, the automatic

generator defines in ATL a transformation from each cell of the first row (which is the column names of the CSV file) to an operation of the API, and also each cell of this first row is transformed into a property and a parameter of the API, using each cell of the second row as example value for the corresponding property/parameter. The OpenAPI model contains all of these objects with specific information about the running example, that is, the different paths and components of the API to retrieve the accessible businesses data.

Considering the "Decal_Recipient" column name from the running example, it is converted into a "Path" object with the pattern "/Decal_Recipient/Decal_Recipient" that includes a get operation, and also to a Parameter object with name "Decal_Recipient" that can be used in this operation and a "Property" also with name "Decal_Recipient".

The definition and documentation of the Web API is represented by a JSON file (Figure 7) according to the standards of Swagger,[11] because it helps us to design, build, document and test the API. Therefore, by a model to text

Figure 7 Extract of OpenAPI JSON file.

[11] https://swagger.io

(M2T) transformation, the API documentation JSON file is directly inferred from the OpenAPI model in XMI format. It consists of a simple element to element transformation since the OpenAPI model contains the same elements than the API documentation but in different format (JSON rather than XMI).

3.3 From OpenAPI Documentation to Web API

Finally, in this stage the complete Web API is generated from its OpenAPI documentation.

The automatic generator creates the API represented by a server in NodeJS,[12] a simple and efficient runtime environment for network applications. This process is accomplished with the help of the Swagger Codegen tool, which creates the structure of the server and manages the calls to the API redirecting them to the corresponding method in the NodeJS code. It also creates an interactive documentation of the API from the existing documentation generated in the previous step. After that, the automatic generator completes the server with the needed features to return the asked data retrieved from the data source, such as filtering by the required parameters. This code added automatically contains functions to read the source file, searching for the desired data and returning it to the user, which is independent from the data source and equal in each API generated.

Between the OpenAPI and the API to generate there is a direct relationship: each "Path" specified in the OpenAPI documentation will result in a different method of the API, and each parameter of the API will be used by the API for filtering and returning the results. The structure of the API also contains: an "api" folder, which includes a swagger file defining structure of the API; the folder "controllers", which includes the main controller to manage the queries and redirect them to the default controller to execute the query, get the information and return it to the user; a "node_modules" folder with the required libraries to implement the NodeJS server; the file "data.csv" containing the CSV input data; and finally a set of additional files in which there is API information.

This generated Web API with query-level capabilities can be published in an online server so that open data reusers can query the data with the desired parameters to filter the information. As the generated API is managed by the user, it can be easily customised, allowing developers with programming skills to add new queries, change the existing ones, modify the filters and

[12] https://nodejs.org

personalise the data they provide. The available queries and filters of the Web API are specified in the interactive OpenAPI documentation, and because of its simplicity they can be executed by non-experts in query languages. All these queries will be of type HTTP GET, specifying the value of the different columns for filtering the results. The queries that this auto-generated API offers came from the dataset itself, so that each column of the dataset is used to create one API method, which can be used to filter the information with values from this specific column. Moreover, to improve the performance of the APIs pagination of the results is available through using limit and offset as parameters of the query. When a user queries the generated API, this query is analysed in order to provide the suitable information. The information to be returned to the user is extracted directly from the dataset, which is analysed row by row by the API to check whether the row fulfils the query filters. Once all the rows that fulfil the query are gathered together by the API, they are sent together parsed as JSON[13] format because it is easy for humans to read and write and it is easy for machines to parse and generate.

For the running example, the Web API generated is available online[14] and it contains a set of methods to query the accessible businesses. All the operations available in the generated API as example are available in the OpenAPI documentation online,[15] which is shown in Figure 8. Considering the

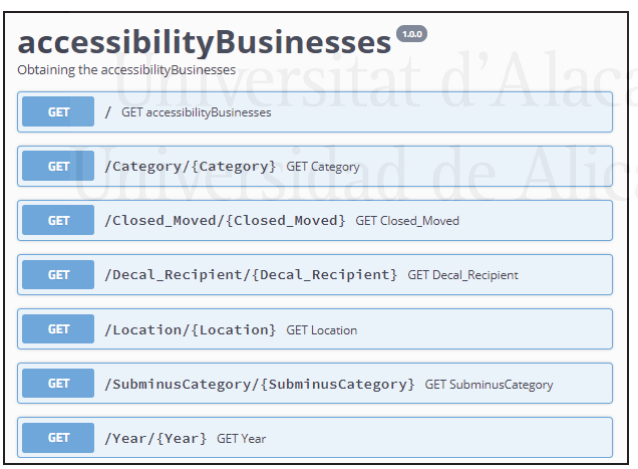


Figure 8 Interactive OpenAPI documentation of the API.

[13]https://www.json.org
[14]https://wake.dlsi.ua.es/AG/RunningExample/
[15]https://wake.dlsi.ua.es/AG/docs/

"Decal_Recipient" column name from the original dataset, it is now a method of the API which can be queried using the GET operation: https://wake.dlsi.ua.es/AG/RunningExample/Decal_Recipient/AMCShowplace11, which uses "AMC Showplace 11" as value of the "Decal_Recipient" parameter defined also as property in the API documentation.

When a user queries this API, a list of accessible businesses that fulfil the specified parameters (column values) is retrieved. For instance, a query that requests the banks that are accessible for people with disabilities is: https://wake.dlsi.ua.es/AG/RunningExample/Category/Bank. From that request, the Web API will response with the result obtained from the CSV dataset. In this case, the result for the query example is the data about accessible banks. An extract of API output in JSON format which contains information about an accessible bank is:

```
{
    'Decal_Recipient': 'BloomBank',
    'ClosedMoved': '',
    'Location': '1301 N Walnut Street',
    'Year': '2009',
    'Category': 'Bank',
    'Sub-Category': ''
}
```

The application created by the developer specified in the running example (Section 2) will access this Web API to reuse data according to the application requirements. Therefore, with the help of this API it will be able to provide a list of accessible businesses, which can be filtered by the name of the business ("Decal_Recipient"), the current situation ("ClosedMoved"), the location within the city of Bloomington ("Location"), the opening year ("Year"), and the category ("Category") and subcategory of the business ("Sub-Category").

The demonstration example, including the dataset used and all the auto-generated files, is publicly available online.[16]

4 Validation of the Approach

In order to evaluate the correctness and performance of the automatic generation process, an experiment was carried out with 20 datasets in a Windows

[16]https://github.com/cgmora12/AG/tree/master/RunningExample/AG_accessibilityBusinesses

Table 3 Validation results of the automatic generation process

CSV Title	Topic	Open Data Platform	# of Rows	# of columns	Generation Time
Traffic state	Transport	datos.gob.es	3,565,683	4	13.4 s
Bikes usage	Transport	datos.gob.es	5,185	5	10.2 s
School grants	Education	Data.gov	2,494	7	10.2 s
Demographic statistics	Government	Data.gov	237	46	10 s
Voter data	Government	Data.gov	7,517,745	46	29.2 s
Biodiversity	Environment	Data.gov	20,017	12	10.3 s
Road safety	Transport	datos.alcobendas.org	113	4	9.7 s
Metro stops	Transport	datos.alcobendas.org	44	12	9.8 s
Train stops	Transport	datos.alcobendas.org	19	12	9.7 s
Population	Government	datos.alcobendas.org	441	5	9.7 s
Salmonella tests	Government	data.gov.uk	13	3	9.7 s
Radioactivity	Environment	data.gov.uk	794	57	10.1 s
Schools list	Education	data.gov.uk	102	16	9.7 s
Street lights	Government	data.gov.uk	27,409	15	10.5 s
Travel data	Government	data.gov.uk	1,853	20	10.3 s
British behaviour	Government	data.gov.uk	168	10	9.7 s
Innovation	Government	open.canada.ca	2,413	17	10.2 s
Wholesale trade sales	Economy	open.canada.ca	8,752,568	17	35.1 s
International payments	Economy	open.canada.ca	245,107	17	10.9 s
Employee earnings	Economy	open.canada.ca	21,072,480	18	80 s

10 computer with an Intel i5 processor and 8GB of RAM memory. The list of CSV data sources analysed in this experiment is shown in Table 3. These online datasets are from different open data platforms: *datos.gob.es* from the Spanish goverment, *data.gov* from U.S. Government, *datos.alcobendas.org* from a city in Madrid, *data.gov.uk* from UK and *open.canada.ca* from Government of Canada. They contain information about education, environment, transport, government and economy. The performance of the approach has been evaluated by analysing the API generation time for each data source with a variable number of rows and columns, as shown in Table 3. The range of file sizes covers from just 10 rows to over 20 million and from 3 columns to 57, resulting in a maximum of 379,304,640 cells and a minimum of 39 cells.

For each dataset, the automatic generation process creates the corresponding Web API with OpenAPI documentation. The time calculated considers the process of copying the whole file without non-alphanumeric characters in the first row (which is dangerous for the API routes), but not the time to download the resource because it depends on network issues, so that the process starts with an already downloaded file. Moreover, in order to get proper results the processing time of the automatic generator has been calculated 5 times for each dataset, thus the generation time is an average of all these 5 different executions.

4.1 Results

The results obtained (Table 3) show that the automatic generation process barely takes between 9 and 80 seconds (Figure 9), with source files of up to more than 20 million of rows, to generate from any dataset a complete Web API, including related models and interactive OpenAPI documentation.

Figure 9 shows a linear behaviour depending on the size of the dataset with a significant positive trend. Thus, as the number of records increases, the time increases in a fraction of seconds (3.1e-06 seconds/record or 3 seconds per million records). The determination coefficient indicates that 98.18% of the variability of the measured time is explained by the number of records. No significant relationship has been found with the number of columns observed.

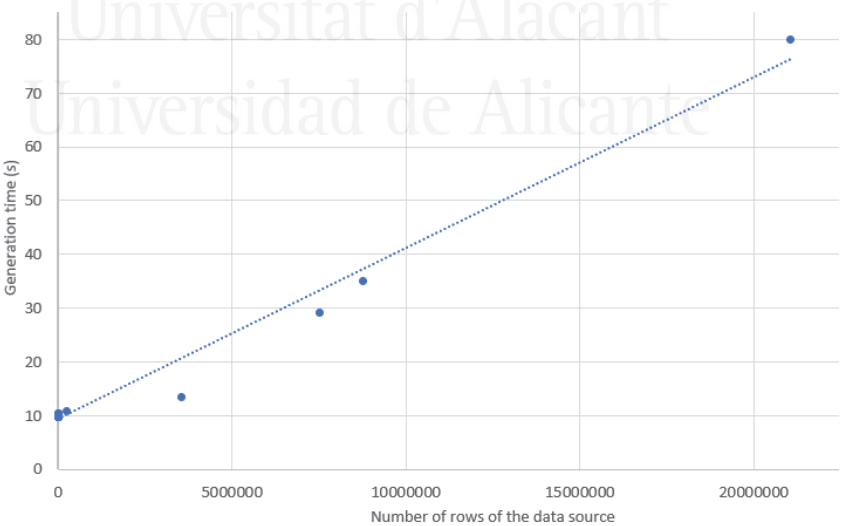


Figure 9 Generation time of the approach depending on the evaluated source file size.

On the other hand, the performance of the Web APIs is also taken into account. When the number of rows is really high, such as 21 million rows, it can take an amount of time similar to the generation time shown in Table 3. This situation can be overcome with the help of pagination of the results (using "limit" and "offset" parameters), optimising the performance of the API and thus significantly reducing the time to get the desired results. Therefore, although the APIs still access data from original CSV file, with this performance improvement it can take less than 4 seconds to return requested data from datasets of up to 21 million rows. For example, a query without filters that specifies the limit of 10.000 results to the Web API that manages 21 million rows takes barely 3 seconds to bring the results to the browser. This validation has been performed using an API generated using our approach from the dataset "Employee earnings" (last row in Table 3).

Consequently, from this validation we can state that the proposal is functional and useful, since it successfully achieves the objective of efficiently performing the APIfication process in an average time of 16 seconds. From the situation where open data platforms do not provide suitable mechanisms to reuse data, such as Web APIs at the query level (i.e. fine-grain access to data), our approach contributes towards the generation of these APIs, which not only eases the task of accessing and reusing open data by developers, but also allows open data publishers to facilitate the reuse of their open data by these developers. Furthermore, the automatic generator successfully provides a modeling environment around open data, helping reusers and citizens to understand better the information offered on the Internet.

5 Related Work

There is a variety of related research that deal with the topic of open data and accessing it using Web APIs, which makes it easier for the developers. In [6], it is supported the idea that the society is opening its data, but there is still the need of building the technology required to enable the citizens to access it. The main goal of this project, which is in an early stage, is giving the citizenship unrestricted access to the open data available online. In [29] it is detailed how open data is mostly not readily usable from the citizen, so it is important to facilitate the access to encourage software developers to use the open data, which is essential in smart cities. Also [22] proposes the development of an open data API which supplies tools that will help the general public to access the data about their own cities. This creation of APIs has also been addressed by other research [13], which describes a

simple query-level API which is used to provide access to a semantic Web to developers unfamiliar with the RDF and SPARQL complex technologies. Additionally, the approach described in [19] is addressing the query of RDF data and convert it to structures and formats which can be processed by data mining tools, because accessing the RDF over numerous SPARQL endpoints supposes a challenging task. In particular, it targets the retrieval and integration of RDF data into the processes designed using RapidMiner, a data mining environment widely used in the industry and research. However, these proposals require the manual creation of the APIs, which is time-consuming.

The automatic generation of APIs has also been proposed [12, 21]: the EMF-REST framework for generating Web APIs [12] needs a model of the API to perform the creation of APIs, thus requiring users to create this model by themselves because they are not generally available; and [21] is focused in helping developers to create automatically Web APIs. However, these approaches are not targeting the access and reuse of data from open data platforms. In addition, csv-to-api[17] dynamically generates RESTful APIs from static CSVs, allowing users to interact with that CSV as if it was a native API. However, this process does not use a model-based approach for the transformations and API documentation is not provided, so that the generated APIs are difficult to use and integrate in model-based scenarios.

Other works propose the use of transformations between metamodels of APIs and other related elements. The paper [15] describes ATL, a domain-specific language for specifying model-to-model transformations, because models are the main development artefacts and model transformations are among the most important operations applied to models, in the context of Model Driven Engineering. In [10, 11] models are used to represent the Web API definition, offering a better visualisation of the API operations. Also, in [9] the metamodel of the API definition is used to simplify the transformation between the API and its definition, to include the API in the OpenAPI initiative. In [23] they use a metamodel for standardising the information extracted from Web APIs documentation; and a method for the extraction of models, discovering useful data, and automatically generating the corresponding models that conform to the defined metamodel. These metamodels help designers to better understanding of each Web API they are using. The tool Direwolf Interaction Flow Designer [16] generates Web frontends from API definitions, using as an intermediary step an API definition model in IFML. The API2MoL engine [7] creates bridges between APIs

[17] https://github.com/project-open-data/csv-to-api

and model-driven engineering, with the objective of creating models from the APIs for facilitating the management of a plethora of APIs. As we have seen, model-based approaches can be used to represent and generate Web API documentation, but they are not used to generate the whole Web API as proposed by the authors in this paper.

As seen in the existing research, the creation of APIs is proposed to solve problems regarding the access and reuse of data. However, they generally propose the manual creation of APIs to access specific open and linked data platforms which is time-consuming. The automatic generation of APIs is also addressed, but fine-grain access to open data platforms is not addressed and they require specific artefacts which are not generally available. Instead, our approach starts directly from the data source, which would avoid the need for users to create these artefacts manually. Therefore, as far as the authors are aware, although related works help data reusers to create APIs and documentation, they do not offer a flexible solution for generating Web APIs from existing open data sources that provide query-level access (i.e., fine-grain access). To do so, our research proposes a model-based approach to automatically generate this kind of Web APIs. Furthermore, we also address open data publishers to easily provide Web APIs, unlike the studied related work which only focus on end users.

6 Conclusions and Future Work

In this paper we have presented an approach that addresses the problem related to accessing and reusing open data available online due to the shortage of query-level Web APIs. In order to solve this problem, we have proposed a model-driven APIfication approach which aims to make open data easily reusable for open data reusers. This process, based on automatic, generic and standardised generation mechanisms, is made up of model-based transformation rules to generate Web APIs with documentation following the OpenAPI 3.0 standard. With this approach we address at first to help open data publishers, so that they can include a Web API that facilitates the reuse of data. In case that an API is missing to reuse open data, we address to help open data reusers such as developers that aim to create value from open data.

The evaluation of the approach with different datasets demonstrates that the generator performs efficiently: it is able to auto-generate successfully a complete Web API for any dataset that did not came up with an API before. Accordingly, the main contribution of this research is the creation of an automatic API generation process to facilitate the access and reuse of

open data. Therefore, the approach aims to directly simplify the open data reuse process, which will result in economic benefits to developers and the infomediary sector.

As future work, the transformation process needs to be extended to work with different formats of open data, considering performance of Web APIs, data integration and semantics. Regarding performance, we plan to keep on working on improving it by considering to include a data stage layer. This layer would include required functionality borrowed from a NoSQL DBMS such as MongoDB to store processed open data coming from CSV. Also, we plan to explore how to use in-memory databases in our approach. Regarding data integration, we plan to consider open data that is split in several datasets and provide unique access to them through a Web API.

Acknowledgements

This work has been funded by the National Foundation for Research, Technology and Development and the project TIN2016-78103-C2-2-R of the Spanish Ministry of Economy, Industry and Competitiveness. César González-Mora has a contract for predoctoral training with the Generalitat Valenciana and the European Social Fund by the grant ACIF/2019/044.

References

[1] State official newsletter of Spain (BOE). Law 19/2013 of December 9, transparency, access to public information and good governance. BOE number 295 of 10-12, 2013.

[2] P. Atzeni, P. Merialdo, and G. Mecca. Data-Intensive Web Sites: Design and Maintenance. *World Wide Web*, 4(1):21–47, 2001.

[3] Sami Beydeda, Matthias Book, Volker Gruhn, et al. *Model-driven software development*, volume 15. Springer, 2005.

[4] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

[5] K. Braunschweig, J. Eberius, M. Thiele, and W. Lehner. The State of Open Data Limits of Current Open Data Platforms. Proceedings of the 21st WWW Conference, 2012.

[6] J. Cabot. Open data for all: an API-based approach, 2016. https://modeling-languages.com/open-data-for-all-api/. Accessed July 31, 2019.

[7] J. L. Cánovas, F. Jouault, J. Cabot, and J. García. API2MoL: Automating the building of bridges between APIs and Model-Driven Engineering. *Information and Software Technology*, 54(3):257 – 273, 2012.

[8] J. S. Cuadrado, E. Guerra, and J. de Lara. AnATLyzer: An Advanced IDE for ATL Model Transformations. pages 85–88. Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, 2018.

[9] H. Ed-douibi, J. L. Cánovas, and J. Cabot. Example-Driven Web API Specification Discovery. pages 267–284. Modelling Foundations and Applications - Springer International Publishing, 2017.

[10] H. Ed-douibi, J. L. Cánovas, and J. Cabot. OpenAPItoUML: A Tool to Generate UML Models from OpenAPI Definitions. pages 487–491, Cham, 2018. Web Engineering - Springer International Publishing.

[11] H. Ed-douibi, J. L. Cánovas Izquierdo, F. Bordeleau, and J. Cabot. Wapiml: Towards a modeling infrastructure for web apis. In *ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 748–752, 2019.

[12] Hamza Ed-douibi, Javier Luis Cánovas Izquierdo, Abel Gómez, Massimo Tisi, and Jordi Cabot. EMF-REST: Generation of RESTful APIs from Models. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1446—-1453. Association for Computing Machinery, 2016.

[13] I. Hopkinson, S. Maude, and M. Rospocher. A Simple API to the Knowledgestore. In *Proceedings of the International Conference on Developers*, volume 1268, pages 7–12. CEUR-WS.org, 2014.

[14] M. Janssen, Y. Charalabidis, and A. Zuiderwijk. Benefits, Adoption Barriers and Myths of Open Data and Open Government. *Information Systems Management*, 29(4):258–268, 2012.

[15] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1):31–39, 2008. Special Issue on Second issue of experimental software and toolkits (EST).

[16] I. Koren and R. Klamma. Generation of Web Frontends from API Documentation with Direwolf Interaction Flow Designer. pages 492–495, Cham, 2018. Web Engineering - Springer International Publishing.

[17] Maria Maleshkova, Lukas Zilka, Petr Knoth, and Carlos Pedrinaci. Cross-lingual web API classification and annotation. In *Proceedings of the 2nd International Conference on Multilingual Semantic Web-Volume 775*, pages 1–12. CEUR-WS. org, 2011.

[18] P. Neirotti, A. De Marco, A. C. Cagliano, G. Mangano, and F. Scorrano. Current trends in Smart City initiatives: Some stylised facts. *Cities*, 38:25–36, 2014.

[19] A. Nolle, G. Nemirovski, A. Sicilia, and J. Pleguezuelos. An Approach for Accessing Linked Open Data for Data Mining Purposes. Proceedings of RapidMiner Community Meeting and Conference, 2013.

[20] Aporta Project. Characterization study of the infomediary sector in Spain, 2012. https://www.ontsi.red.es/ontsi/sites/ontsi/files/121001_red_007_final_report_2012_edition__vf_en_1.pdf.

[21] Ricardo Queirós. Kaang: A RESTful API Generator for the Modern Web. In *7th Symposium on Languages, Applications and Technologies*, volume 62 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:15, 2018.

[22] M. Rittenbruch, M. Foth, R. Robinson, and D. Filonik. Program your city: designing an urban integrated open data API. pages 24–28. Proceedings of Cumulus Conference: Open Helsinki–Embedding Design in Life, 2012.

[23] R. Rodríguez-Echeverría, J. M. Conejero, P. J. Clemente, M. D. Villalobos, and F. Sánchez-Figueroa. Extracting Navigational Models from Struts-Based Web Applications. pages 419–426, 2012.

[24] B. Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, 2003.

[25] U. Sivarajah, V. Weerakkody, P. Waller, H. Lee, Z. Irani, Y. Choi, R. Morgan, and Y. Glikman. The role of e-participation and open data in evidence-based policy decision making in local government. *Journal of Organizational Computing and Electronic Commerce*, 26(1-2):64–79, 2016.

[26] A. Srai, F. Guerouate, N. Berbiche, and H. Drissi. An MDA approach for the development of data warehouses from relational databases using ATL transformation language. *International Journal of Applied Engineering Research*, 12:3532–3538, 2017.

[27] A. Stott. Open data for economic growth. *Washington DC: World Bank*, 2014.

[28] B. Ubaldi. Open government data: Towards empirical analysis of open government data initiatives. *OECD Working Papers on Public Governance*, (22), 2013.

[29] V. Weerakkody, Z. Irani, K. Kapoor, U. Sivarajah, and Y. K. Dwivedi. Open data and its usability: an empirical view from the Citizen's perspective. *Information Systems Frontiers*, 19(2):285–300, 2017.

[30] J. Wettinger, U. Breitenbücher, and F. Leymann. ANY2API – Automated APIfication – Generating APIs for Executables to Ease their Integration and Orchestration for Cloud Application Deployment Automation. pages 475–486. Proceedings of the 5th International Conference on Cloud Computing and Services Science, 2015.

Biographies

César González-Mora is a PhD student in the Web and Knowledge research group from the Department of Software at University of Alicante, Spain. His research interests include open data, web augmentation, the semantic web and application programming interfaces. His work is funded by a contract with the Generalitat Valenciana of Spain and the European Social Fund for predoctoral training (ACIF/2019/044).

Irene Garrigós (PhD.) is Associate Professor in the Department of Software and Computing Systems at the University of Alicante, Spain. Her research interests include open data, web augmentation, web modeling languages, personalization and application programming interfaces. She is the head of the Web and Knowledge research group of the University of Alicante.

José Jacobo Zubcoff (PhD.) presents a wide teaching and research experience in the field of Statistics, Data Mining and its application to Biology. He has more than 100 publications in which he has dealt with obtaining knowledge from a data source. He has done research in various fields of science, both in computing, biology, medicine, education and social sciences. In addition, he has directed and participated in more than 20 competitive public projects financed by the Ministry of Economy and Competitiveness, the Generalitat Valenciana, the University of Alicante and European and private projects, all of them contributing his knowledge about data analysis, data mining and aiming at the democratization of knowledge.

Jose-Norberto Mazón (PhD.) is Associate Professor in the Department of Software and Computing Systems at the University of Alicante, Spain. His research interests include open data, business intelligence in big data scenario, design of data-intensive web applications, smart cities and smart tourism destinations. He is the author of more than hundred scientific publications in international conferences and journals. He is currently Chair of the Torrevieja's Venue of the University of Alicante.

Chapter 6

Model-Driven Development of Web APIs to Access Integrated Tabular Open Data

César González-Mora, David Tomás, Irene Garrigós, José Jacobo Zubcoff, and Jose-Norberto Mazón. Model-Driven Development of Web APIs to Access Integrated Tabular Open Data. IEEE Access, 8:202669–202686, 2020.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 16, 2020, accepted October 28, 2020, date of publication November 6, 2020, date of current version November 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3036462

Model-Driven Development of Web APIs to Access Integrated Tabular Open Data

CÉSAR GONZÁLEZ-MORA[1], DAVID TOMÁS[1], IRENE GARRIGÓS[1], JOSÉ JACOBO ZUBCOFF[2], AND JOSE-NORBERTO MAZÓN[1]

[1]Department of Software and Computing Systems, University of Alicante, 03690 Alicante, Spain
[2]Department of Sea Sciences and Applied Biology, University of Alicante, 03690 Alicante, Spain

Corresponding author: César González-Mora (cgmora@ua.es)

This work was supported by the National Foundation for Research, Technology and Development of the Spanish Ministry of Economy, Industry and Competitiveness under Project TIN2016-78103-C2-2-R and Project RTI2018-094653-B-C22. The work of César González-Mora was supported by a contract for predoctoral training with the Generalitat Valenciana and the European Social Fund under Grant ACIF/2019/044.

ABSTRACT More and more governments around the world are publishing tabular open data, mainly in formats such as CSV or XLS(X). These datasets are mostly individually published, i.e. each publisher exposes its data on the Web without considering potential relationships with other datasets (from its own or from other publishers). As a result, reusing several open datasets together is not a trivial task, thus requiring mechanisms that allow data consumers (as software developers or data scientists) to integrate and access tabular open data published on the Web. In this paper, we propose a model-driven approach to automatically generate Web APIs that homogeneously access multiple integrated tabular open datasets. This work focuses on data that can be integrated by means of join and union operations. As a first step, our approach detects unionable and joinable tabular open data by using a table similarity measure based on word embeddings. Then, an APIfication process is developed to create APIs that access the previously integrated datasets through a single endpoint. A running example is presented throughout the article, as well as a set of experiments for performance evaluation to show the feasibility of our approach.

INDEX TERMS Data integration, join, union, open data, data access, Web APIs, word embeddings.

I. INTRODUCTION

Nowadays the amount of open data available on the Web is increasing due to the great interest of governments and institutions around the world in adopting open data initiatives [1]. A good example is found in the smart city arena, where open data has attracted great attention for local and regional governments as the best way of publishing the big data they are producing [2].

This open data is commonly offered in catalogues within Web platforms, named *open data portals*. For instance, *https://www.data.gov/*, which provides a catalogue of data resources from the USA Government (at national level), or *https://data.cityofchicago.org/*, which gathers open data from Chicago (at local level).

The ultimate goal of open data portals is to provide Linked Open Data (LOD) that allows consumers to use Semantic Web technologies to identify relationships among data [3]. LOD is easy to integrate and access by means of query languages such as SPARQL. Unfortunately, LOD is not usually available since most used formats in open data government portals are tabular (46.4%), either direct tabular formats such as CSV (9.1%) and XLS(X) (6.1%), or embedded tabular data such as HTML (25.0%) and PDF (9.2%). Meanwhile, LOD formats such as RDF only represent 0.5% of the total [4] (even though there exist proposals to transform from CSV to RDF [5]). The prevalence and priority of tabular formats over LOD in open data government portals has been highlighted in recent studies by the European Comission [6] and the Organisation for Economic Co-operation and Development (OECD) [7]. In this scenario, accessing and manipulating tabular datasets remains a relevant research topic in the field of open data.

When it comes to integrating tabular open data, additional metadata is required for developers to know how datasets can be related to each other (such as relational-like primary keys

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

or foreign keys, as stated by the "Model for Tabular Data and Metadata on the Web"[1] developed by the W3C CSV on the Web Working Group). Although several recent approaches proposed adding these kind of metadata to tabular datasets by means of annotations [8], [9], this methodology has not being widely adopted by publishers. Therefore, open data is mostly individually published, i.e. each publisher shares their data on the Web without considering potential relationships with other open data. This scenario hampers users of open data government portals (such as software developers or data scientists) in reusing open data, since additional effort must be done to successfully integrate this data. Thus, mechanisms that allow data consumers to integrate and access tabular open data published on the Web (through open data government portals) are highly required.

To this end, in this paper we propose a model-driven APIfication process to define transformations for automatically generating Web APIs that access integrated tabular open data. Data integration is a complex process, involving several kinds of transformations (such as cleansing, combination, normalisation, etc.) to offer a unified view of a set of heterogeneous data from different sources. In this paper, we focus on join and union operators since they are specially relevant for open data integration [10]. In addition, it should be noted that considering these operators provides the basis for including other ones (such as filtering and sorting).

Although data integration challenges have been researched for years with significant progress [11], efforts have been made only on solving specific problems. However, according to Abadi *et al.* [12] more work is required on researching how to pipeline data integration to cover all the way from raw data to an end-user's desired outcome. This could be achieved, for instance, by means of generating mechanisms that support developers and data scientists in consuming the right data for their purposes by using external programming languages such as Java, Python, and R. In this sense our work is aligned with Abadi *et al.*, since the approach we propose focuses on pipelining the data integration process together with an API generation in order to simplify consumption of integrated open data.

The first step in the approach proposed consists of detecting the tabular datasets that are more likely to be integrated by means of join and union operations.[2] For this purpose, we defined a similarity measure between tabular data based on word embeddings [13]. Afterwards, in a second step a Web API is automatically generated to directly access the integrated datasets. Web APIs are a recommended feature of open data portals [14], allowing data consumers to build data-intensive applications and bring open data to citizens. Moreover, the documentation of the Web API is also automatically generated (i.e. its interactive OpenAPI[3] documentation), helping data consumers to better understand how to access and reuse integrated tabular open data coming from different sources.

It is worth noting that both steps in the APIfication process are decoupled and can be used independently, while at the same time relationships among steps (e.g. passing information from integrated data to a data model in order to initiate the Web API generation) can be easily considered by following a model-driven development approach.

In summary, the contributions of this article are as follows:

- The definition of a word embedding-based similarity measure between tabular datasets to identify joinable and unionable tabular open data.
- A set of model-driven transformations to automatically generate a Web API to easily access previously integrated data (by applying the corresponding join and union operations).
- Evaluation of our approach with real tabular open data.
- The implementation of our approach for automatically data integration and the corresponding Web API generation, which is available online at GitHub.[4]

This article is structured as follows. Section II presents the running example used to illustrate the approach explained in Section III. That section describes our model-driven approach for automatically generating Web APIs to access integrated tabular open data. Then, Section IV presents the evaluation of the approach. Finally, related work is described in Section V and conclusions and future work are sketched out in Section VI.

II. RUNNING EXAMPLE

This section introduces a running example which is used throughout the article to illustrate our approach. The considered scenario is related to available open data regarding the COVID-19 pandemic coming from different open data portals, and how they can be integrated and accessed through an automatically generated Web API. All the files related to the running example are publicly available online.[5]

This running example describes a situation where a data scientist is willing to use available open data to create a dashboard to analyse the COVID-19 pandemic evolution along different cities in USA. However, the data scientist has to address a data integration problem, since open data about COVID-19 is published at different portals, by different local or regional governments (USA cities or states), and usually federated in a national open data portal. This data may rely on different schemas (i.e. data structures), being necessary to integrate them first in order to successfully analyse them together.

In our running example, data comes from USA Government's open data portal.[6] On one hand, data from Chicago is

[1] https://www.w3.org/TR/tabular-data-model/

[2] It is worth noting that our approach could be used for any tabular format, although we focus on CSV files in this article.

[3] https://www.openapis.org/

[4] https://github.com/cgmora12/DataIntegration2API

[5] https://github.com/cgmora12/DataIntegration2API/tree/master/runningExample

[6] https://www.data.gov/

gathered from two different datasets. The first one[7] provides figures of positive COVID-19 cases by day, filtering by gender, race, and different ranges of age. It contains 148 rows and 39 columns. An excerpt of this data (called *dataset 1*) is shown in Table 1. The second one[8] (*dataset 2*) provides hospital capacity metrics during COVID-19 period. It contains 132 rows and 33 columns. See Table 2 for a sample data.

TABLE 1. Excerpt of data from COVID-19 cases in Chicago (*dataset 1*).

date	cases - total	deaths - total	cases - age 0-17	cases - male	cases - female
03/25/2020	367	5	2	196	170
03/26/2020	416	2	2	230	184
03/27/2020	404	8	4	192	210

TABLE 2. Excerpt of hospital capacity data from Chicago (*dataset 2*).

date	ventilators total capacity	ventilators in use COVID-19 patients	ventilators in use non-COVID-19
03/25/2020	1,048	79	343
03/26/2020	1,053	88	295
03/27/2020	1,042	108	309

On the other hand, open data from New York City is gathered from a unique dataset[9] (*dataset 3*) that includes daily counts of cases, hospitalisations, and deaths from COVID-19. It contains 151 rows and 4 columns. An excerpt is shown in Table 3.

TABLE 3. Excerpt of data from COVID-19 cases in New York City (dataset 3).

date	case count	hospitalization count	death count
2020 Mar 22 12:00:00 AM	2580	725	50
2020 Mar 23 12:00:00 AM	3568	1037	82
2020 Mar 24 12:00:00 AM	4504	1153	94

In order to consider data provenance, we automatically add a new column to each dataset containing its publisher before processing the data: in dataset 1 and dataset 2 the publisher is "City of Chicago", whereas "City of New York" is the publisher of dataset 3. This provenance data comes from the datasets' metadata (USA Government's open data portal) and can be easily obtained through its API.

[7]https://catalog.data.gov/dataset/covid-19-daily-cases-and-deaths, last accessed October 2020

[8]https://catalog.data.gov/dataset/covid-19-hospital-capacity-metrics, last accessed October 2020

[9]https://catalog.data.gov/dataset/covid-19-daily-counts-of-cases-hospitalizations-and-deaths, last accessed October 2020

If a data scientist wants to use this data to get the COVID-19 positive cases, deaths, and people hospitalised by day using ventilators, three datasets must be downloaded and integrated by applying join and union operators (a more detailed explanation about these operators is provided in the next section) as follows (see also Fig. 1):

- Dataset 1 and dataset 2 contain data with different measures that can be joined by the column "date" (resulting in a new dataset) in order to have positive cases, deaths, and hospitalised people using ventilators by day in Chicago. For example, the 25th of March, there were 367 cases and 5 deaths in Chicago (dataset 1), and also 79 hospitalised people that used ventilators due to COVID-19 (dataset 2). All this information is joined in the same row because they share the same "date", which is used as a condition for the join operator.
- Union of the previously joined dataset and dataset 3 can be applied to get the positive cases, deaths, and hospitalised persons using ventilators by day in both Chicago and New York. For example, in order to perform the union operation, the columns are matched as follows: "date" with "date" (previously considering that a date format conversion must be done), "case count" with "cases", "death count" with "deaths" and "hospitalized count" with "ventilators". Matching these columns and applying a union operator results in a new table containing all the rows from the input tables. Fig. 1 shows that the first row of the integrated data comes from dataset 3, whereas the second row comes from the previously joined datasets. Additionally, a new column has been included with information about the publisher of the data.

Manually obtaining the integrated dataset required in this example is a time-consuming task for the data scientist. In the following section, we use this running example to show how our approach can be useful to save time and effort in integrating and accessing tabular open data.

III. A MODEL-DRIVEN APIfication APPROACH TO ACCESS INTEGRATED TABULAR OPEN DATA

This section presents our model-driven APIfication approach to automatically generate Web APIs to access previously integrated tabular open data.

Operators considered for handling input tabular data are union and join from relational algebra. For the sake of simplicity, in this article we borrow these operators from SQL (the well-known implementation of the relational algebra and a recognised standard for querying and handling tabular data):

- Union operator is denoted by $\cup$ symbol in relational algebra. Given two tabular datasets (A and B), union operator aims to get a unique dataset that contains rows that are in A or in B (denoted as A $\cup$ B). A and B must have the same columns (number, order, and datatype) to be computed. Also, each column of each dataset must refer to the same concept to be meaningful.

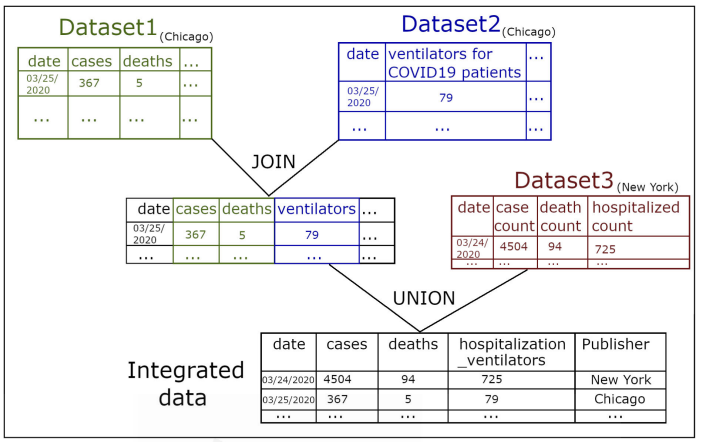


FIGURE 1. Example of integrating COVID-19 datasets.

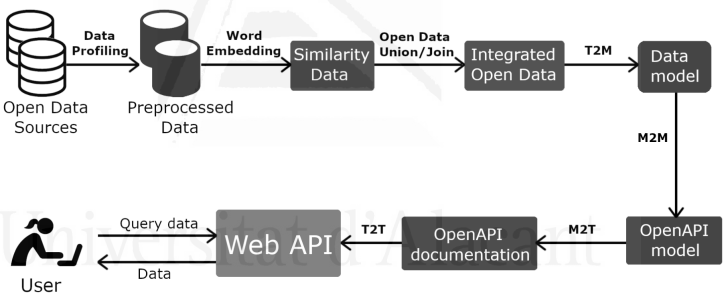


FIGURE 2. Automatic open data integration and API generation process.

- Join operator is denoted by ⋈ symbol in relational algebra. Given two tabular datasets A and B, join operator aims to get a unique dataset that includes every column from A and B (denoted as A ⋈ B) and contains rows that fulfil a matching condition (applied to values of some columns).

An overview of our approach is shown in Fig. 2. It consists of two parts:

- Data integration (“Similarity Data” and “Integrated Open Data” boxes in Fig. 2). The first step implies detecting unionable and joinable tabular datasets from an open data portal. Our approach uses word embeddings [13] to detect which columns from different input tabular datasets are more likely to be integrated by using join and union operators. The next step consists of applying the required join and union operators to get an integrated tabular dataset. More details are given in Section III-A. As shown in Fig. 2, prior to this data integration phase a data profiling is performed in order to discover data types and to apply type conversions to homogenise them (e.g. conversion of date formats by using the “mm/dd/yyyy” pattern), as well as to detect keys that will be used later to apply the join operator.
- Model-driven transformations to obtain a Web API to access integrated data, including its interactive OpenAPI documentation: “Data model”, “OpenAPI model”, “OpenAPI documentation”, and “Web API” boxes in Fig. 2. From the previously integrated dataset, a Web API to access this data is generated by means of the following model-driven transformations: (i) a text to model (T2M) transformation from the integrated data to the data model; (ii) a model to model (M2M) transformation

from the data model to the OpenAPI model; (iii) a model to text (M2T) transformation from the OpenAPI model to its OpenAPI documentation; and finally (iv) a text to text (T2T) transformation from the OpenAPI documentation to the Web API. Therefore, starting from a dataset containing rows and cells, the system performs a direct transformation to construct a data model object with row and cell objects. Then, the data model is transformed to an OpenAPI model using each cell of the first row as an API component (method, parameter, and property), parsing this OpenAPI model object to a standard format for API documentation. Finally, the complete Web API is automatically created according to the methods detailed in its documentation. These model-driven transformations are explained in detail in Section III-B.

A. INTEGRATION OF TABULAR OPEN DATA

As mentioned before, the first part of the process consists of integrating datasets by means of join and union operators. The initial step is to determine which columns of the datasets are similar enough to apply these operations. The following sections explain in detail how similar columns are detected, as well as how union and join operators are applied for integrating tabular data (we specifically focus on CSV files).

1) MEASURING COLUMN SIMILARITY FOR DETECTING UNIONABLE AND JOINABLE TABULAR DATA

Word embeddings is a Natural Language Processing (NLP) technique in which words or phrases are mapped to vectors of real numbers. This mapping is based on the distributional hypothesis, which states that words that occur in the same contexts tend to have similar meanings [15]. Word embeddings have been shown to capture semantic regularities in vector space, since the relationship between two word vectors mirrors the linguistic relationship between those words [13].

In order to determine whether union or join operations between tabular data can be performed, we have established a mechanism based on word embeddings to calculate the (semantic) similarity of two tabular datasets. Using word embeddings overcomes the problems of lexical approaches based on string similarity: terms such as ''city'' and ''location'' could be considered as being very different in terms of string matching, but in a word embedding space these two terms may be closely related and considered as highly similar. The similarity mechanism takes as an input a set of tabular datasets to be compared, returning as a result a JSON file where all the columns of each dataset are compared with each other, obtaining a similarity measure for each pair of columns belonging to different datasets. These column pairs are sorted in descending order of similarity. This measure serves as the input to decide whether union or join operators can be applied to integrate different datasets (see Section III-A2 for additional information). In order to calculate the similarity between columns, we take into account two elements of the tabular datasets: name of the columns and their content (values) for each row.

The first step consists of normalising these values by splitting CamelCase and hyphenated words (very common in column names), removing punctuation, and converting text to lowercase. After that, the word embedding model is used to extract two vectors for each column: one represents the name of the column and the other the content of the column for each row of data. In those situations where the name of the column includes more than one word, the vectors representing each word are averaged to get a single vector. Averaging word embeddings is one of the most popular methods of combining embedding vectors, outperforming more complex techniques especially in out-of-domain scenarios [16]. The same strategy is applied to the content of the column, where the final vector is the result of calculating the mean between the vectors of each of the values contained.

As in previous works [13], we use the cosine similarity to compute the distance between vectors in the embedding space:

$$sim(v_1, v_2) = \frac{v_1 v_2}{\|v_1\| \|v_2\|} = \frac{\sum_{i=1}^{n} v_{1i} v_{2i}}{\sqrt{\sum_{i=1}^{n} (v_{1i})^2} \sqrt{\sum_{i=1}^{n} (v_{2i})^2}},$$

where v_1 and v_2 represent the word embedding vectors of the name of the columns or the content of the column for each row, and $sim(v_1, v_2)$ is a float value in the range [0, 1], where 0 means no similarity and 1 means maximum similarity between the vectors considered.

If the word embedding model does not provide coverage for the name of the column or its content (i.e. their tokens are not in the vocabulary of the model), the Levenshtein distance [17] is used as a backup strategy to ensure that the system always returns a similarity value between columns. This string metric is based on the number of single-character edits (insertions, deletions or substitutions) required to change one string into the other. We applied the normalised edit distance to obtain values in the range [0, 1], computed as $(length - distance)/length$, where *distance* is the Levenshtein distance and *length* is the sum of the lengths of the two strings to compare.

For each two columns compared, we obtain a similarity value of the name of the column and a similarity value of its content. To obtain a single final similarity score of two columns c_1 and c_2, we perform a linear combination of these two values:

$$sim(c_1, c_2) = \alpha \cdot sim(c_{n1}, c_{n2}) + (1 - \alpha) \cdot sim(c_{c1}, c_{c2}),$$

where $sim(c_{n1}, c_{n2})$ is the similarity of the names of the columns, $sim(c_{c1}, c_{c2})$ is the similarity of their contents, and α is a parameter in the range [0, 1] that weights the relevance of the two similarity scores in the final result.

2) APPLYING UNION AND JOIN OPERATIONS FOR DATA INTEGRATION

The similarity measure computed in the previous step is used to decide whether union or join operations can be applied for integrating tabular data. Two rules apply:

- A join operator can be applied to datasets that have at least one column (detected as candidate key in the data profiling) with similarity value higher than a specific threshold (as explained below). These columns are considered as key columns in the join operation (several columns can be detected as key for joining).
- A union operator can be applied to datasets that have all columns with similarity value higher than a specific threshold. Columns with similarity below this threshold can be removed previously. Also, if the majority of columns are similar, the union operation can be applied only on these columns omitting the rest of non-similar columns.

The value of the thresholds mentioned are obtained empirically considering that: (i) join operation requires key columns, which are a small subset of columns from the input datasets (usually one or two); (ii) union operation requires the same columns to be present in both input datasets. In the running example, the threshold for join operator was set to 0.95, whereas for union operator was set to 0.75. However, our approach is flexible and thresholds can be adapted at any time.

Join and union operators between datasets are automatically applied by dynamically developing and executing transformations with Pentaho Data Integration Java libraries.[10] Therefore, after analysing the similarity calculations we can automatically create a transformation using these libraries and execute it to obtain the integrated data. This generated transformation includes the following operations: (i) reading the datasets to be integrated; (ii) sorting data; (iii) computing similarity measures to apply join/union operations; and (iv) save the integrated data to an output file. An excerpt of the code required is shown in Fig. 3 (the full code is available at the GitHub repository[11]).

Regarding the running example, a join operation is performed automatically on the two datasets from Chicago (dataset 1 and dataset 2) because the similarity of "date" column from COVID-19 cases dataset and the "date" column from Hospital Capacity Metrics is the highest computed (0.9988) and above the threshold of 0.95. Both "date" columns are selected as key columns for the join operator. After joining, we obtain a single tabular dataset with data coming from both datasets (see Table 4). Moreover, after calculating the similarity between columns from the previously integrated datasets and dataset 3 from New York, the similarity obtained is higher than the 0.75 threshold for the following matching columns: "case count" with "cases total" (0.8847), "death count" with "deaths total" (0.8724), "date of interest" with "date" (0.8607), and "hospitalized count" with "ventilators in use covid 19 patients" (0.7793).

[10]https://www.hitachivantara.com/en-us/products/data-management-analytics/pentaho-platform/pentaho-data-integration.html

[11]https://github.com/cgmora12/DataIntegration2API/blob/7cca3fb915f933a315858ac7ae6e0e1358f7df33/src/table/union/TableUnion2API.java#L341

```
// Create transoformation
StepLoader.init();
TransMeta transMeta = new TransMeta();
transMeta.setName("Join Transformation");

// First step: read CSV
CsvInputMeta csvinput = new CsvInputMeta();
csvinput.setFilename(csvFilename1);
csvinput.setDelimiter(separator1);
csvinput.setHeaderPresent(true);
csvinput.setEnclosure("\"");
csvinput.setBufferSize("50000");
csvinput.setInputFields(configureInputFields());

StepMeta csvinputStepMeta = new StepMeta("CsvInput", csvinput);
csvinputStepMeta.setDraw(true);
csvinputStepMeta.setLocation(100, 100);
transMeta.addStep(csvinputStepMeta);
```

FIGURE 3. Excerpt of Java code for generating a join transformation.

TABLE 4. Excerpt of joined dataset from COVID-19 cases in Chicago.

date	cases - total	deaths - total	ventilators in use COVID-19 patients
03/25/2020	367	5	476
03/26/2020	416	2	88
03/27/2020	404	8	479

Therefore, a union operation is performed to integrate both datasets based on these matching columns.

Although a Pentaho Data Integration transformation is created and executed dynamically by using its corresponding Java libraries, the transformation for the running example can be accessed by using the Pentaho Data Integration GUI, named Spoon, as shown in Fig. 4. It is worth noting that, by doing this, Spoon capabilities for debugging transformations can be easily used if required. For example, users could use Spoon for manually editing joining keys automatically selected previously by our approach.

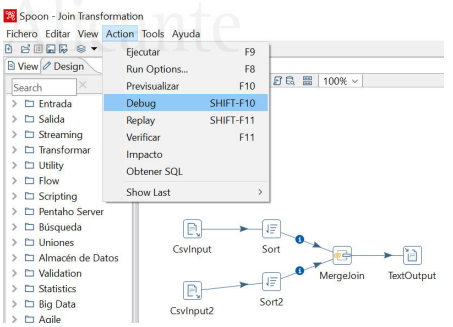


FIGURE 4. Example of join transformation auto-generated dynamically.

Once the data is integrated by applying join and union operators, a Web API is automatically generated to access these data (Table 5) as shown in the following section. This process of generating the Web API from the integrated data

TABLE 5. Excerpt of the dataset obtained after union operation of the datasets from COVID-19 cases in Chicago and New York.

date	cases - total	deaths - total	hospitalized (with ventilators)	publisher
03/22/2020	2580	50	725	City of New York
03/23/2020	3568	82	1037	City of New York
03/24/2020	4504	94	1153	City of New York
03/25/2020	367	5	476	City of Chicago
03/26/2020	416	2	295	City of Chicago
03/27/2020	404	8	479	City of Chicago

is decoupled from data integration itself and can be used independently. However, at the same time, the required information is easily transferred throughout the process thanks to the model-driven development principles. In particular, after obtaining the similarity results, the decision on which tables to perform the union/join operation depends on a threshold. Although there is a default threshold, users could change it at the beginning of the data integration process. This decision does not affect the process of API generation since only already integrated data, together with other information stored in the models, are required as an input.

B. MODEL-DRIVEN TRANSFORMATIONS FROM INTEGRATED DATA TO WEB API

After generating the integrated data, the transformation process continues in order to generate a Web API that offers easy access to this data. As shown in Fig. 2, this transformation from the integrated data to the Web API contains a text to model (T2M) transformation to create a data model, a model to model (M2M) transformation to generate the documentation model in OpenAPI, a model to text (M2T) transformation to obtain the complete OpenAPI documentation, and finally a text to text (T2T) transformation that completes the process by producing the Web API.

The T2M transformation starts from the integrated open data, generating the related data model. It consists of a MOF-based[12] model in XMI format according to its metamodel defined in Fig. 5 by using the Ecore format (actually, the *de facto* reference implementation of MOF) from the Eclipse Modeling Framework (EMF). This metamodel specifies that tabular data as CSV file contains a name and a set of rows, which include a position and a set of cells with value and type. The first row of a tabular dataset, which represents the column names, is used for the API method, properties, and parameters; the second row, which contains the data, is used as example values for each column. Therefore, the integrated data in a CSV file is processed by rows, analysing the first two rows for creating the data model with row and cell objects.

Particularly, in the running example the generated data model shown in Fig. 6 contains an object "Table", formed by a set of "Rows" containing many "Cells". The information

[12]https://www.omg.org/mof/

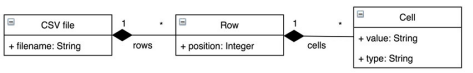


FIGURE 5. Tabular dataset metamodel.

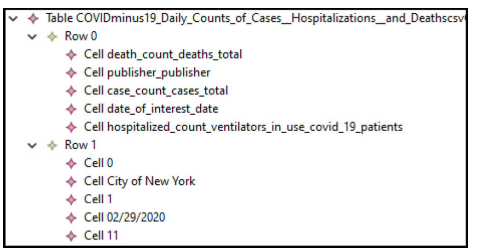


FIGURE 6. Datafile model in XMI format.

contained in the cells of the first row consists of the column names, while the cells in the second row contain integrated data examples about COVID-19. An example of column name is "date_of_interest_date", which is converted into a cell in the first row (Row 0 in Fig. 6). This cell, and all the cells from the first row, will then be used as the name of an API method, parameter, and property in the subsequent transformation steps. On the other hand, an example value of "date_of_interest_date" is "02/29/2020", which is converted into a cell in the second row (Row 1 in Fig. 6) and will be used as an example value in the corresponding API method, parameter, and property.

Once the data model is created, the M2M transformation is performed. In this step an OpenAPI model is created from the data model. This transformation is defined using ATL, one of the most used languages for model transformations [18], [19]. Using the ATL language we defined a set of transformation rules (see Fig. 7) that automatically convert the data model

```
-- @atlcompiler emftvm
-- @path Table=/TransformationRules/Table.ecore
-- @path Openapi=/TransformationRules/Openapi.ecore
module Table2Openapi;

create OUT: Openapi from IN: Table;
rule Main {
    from
        s: Table!Table
    using {
            firstTableRow : Sequence(Table!Cell) = s.rows->first().cells;
            lastTableRow : Sequence(Table!Cell) = s.rows->last().cells;
    }
    to
        t: Openapi!API (
            openapi <- '3.0.0',
            info <- openapi_info,
            servers <- Sequence{openapi_servers},
            paths <- Sequence{openapi_basic_path}.union
                (firstTableRow -> collect(e | thisModule.OpenapiPaths(e, s))),
            components <- Sequence{openapi_components}
        ),
        openapi_info: Openapi!Info (
            title <- s.filename,
            version <- '1.0.0',
            description <- 'Obtaining the ' + s.filename
        ),
        openapi_servers: Openapi!Server (
            url <- 'http://www.urlprueba.com/v1'
        ),
```

FIGURE 7. Excerpt of ATL transformation rules.

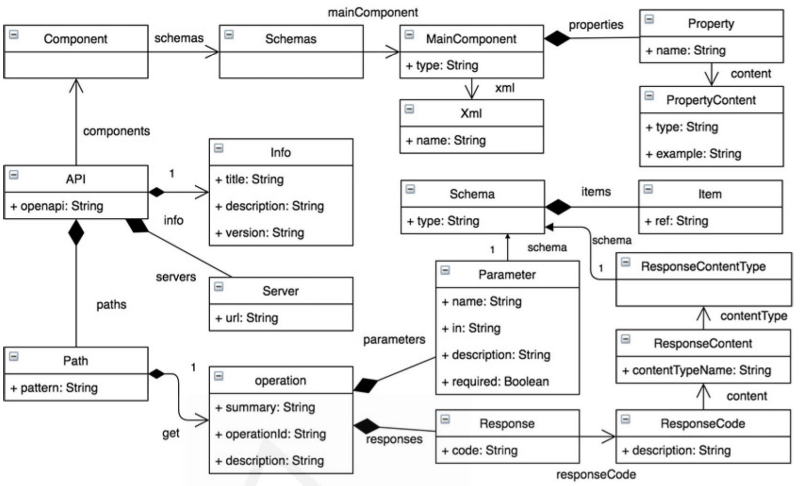


FIGURE 8. OpenAPI metamodel.

to the OpenAPI model. Starting with the table object from the data model, we are able to generate the OpenAPI model and all the different objects contained in its OpenAPI metamodel, which has been created by the authors according to the OpenAPI structure (Fig. 8). The main elements of this OpenAPI metamodel are: the `API` object containing OpenAPI information about the version and the URL of the Web API; the `Path` object which includes a set of API operations with their parameters; and the `Component` object to define the properties of the API.

The generated OpenAPI model for the running example is shown in Fig. 9. The "date_of_interest_date" column name from the running example is converted into a "Path" object with the pattern "/date_of_interest_date/ date_of_interest_date" including a GET operation, a "Parameter" object "date_of_interest_date" that can be used in this operation, and a "Property" named "date_of_interest_date".

After that, the M2T transformation between the OpenAPI model and the OpenAPI documentation is carried out. This OpenAPI documentation of the Web API is represented by a JSON file according to the standards of Swagger.[13] This JSON file is directly inferred from the OpenAPI model in XMI format by a simple element to element transformation, since the OpenAPI model contains the same elements than the API documentation. An excerpt of the OpenAPI documentation in JSON format generated for the running example is shown in Fig. 10. It contains important elements such as API information, server URL, components of the API, and paths to obtain the COVID-19 data filtering by different parameters and properties. The "Path" object "date_of_interest_date"

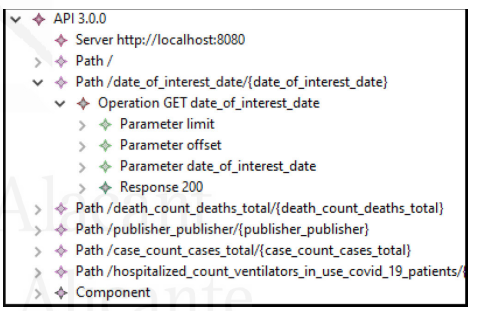


FIGURE 9. Excerpt of the OpenAPI model (XMI).

from the OpenAPI model is now converted into a JSON object inside the "Path" JSON array. This "date_of_interest_date" JSON object also contains more details about the GET operation and the parameter to filter the information.

Finally, the complete Web API is generated by the T2T transformation from the OpenAPI documentation. The T2T process is able to automatically generate the whole Web API represented by a server in NodeJS.[14] This API generation is performed with the help of the Swagger Codegen tool,[15] which helps in the creation of the structure required for the API, managing the calls to its different methods. Next, the code for the methods of the API is provided by our approach, which includes the required features to retrieve and

[13] https://swagger.io

[14] https://nodejs.org

[15] https://swagger.io/tools/swagger-codegen/

FIGURE 10. Excerpt of the OpenAPI JSON file.

return the integrated data requested by the data consumers. To this end, we first developed a base programming code which is then automatically added to each generated API. This code is then adapted to the method of the API and the source dataset, being able to perform the operations to read the dataset, search and filter specific information, and return it to the users.

There is a direct relationship between the OpenAPI and the API to generate: each "Path" specified in the OpenAPI documentation will result in a different method of the API, and each parameter will be used by the API for filtering and returning the results. The structure of the API also contains: an "api" folder, which includes a Swagger file defining the structure of the API; the folder "controllers", which includes the main controller to manage the queries and redirect them to the default controller to execute the query, get the information and return it to the user; a "node_modules" folder with the required libraries to implement the NodeJS server; the file "data.csv" containing the CSV input data; and a set of additional files including API information.

The generated Web API can be published in an online server so that users can query and filter the integrated data with the desired parameters. Data consumers are also able to perform API queries using the interactive OpenAPI documentation. This interactive documentation reduces the possibility of introducing errors such as writing mistakes. For example, if data consumers want to query the API directly and filter the data by a parameter called `case_count_cases_total`, they have to specify the exact parameter taking into account symbols and lowercase. However, with our interactive documentation data consumers do not need to write down the parameter since they can just click on the specific method to filter by the desired parameter.

With regard to the running example, the OpenAPI model generated contains all the objects with the specific information. That is, the different paths and components of the API to retrieve the integrated COVID-19 data filtering by different parameters, which are indeed the columns contained in the integrated data. The API of the running example is available at https://wake.dlsi.ua.es/IntegrationAPI/, whereas the corresponding documentation (see Fig. 11) is at https://wake.dlsi.ua.es/IntegrationAPI/docs/.

When the API from the running example is queried, the COVID-19 data that fulfils the specified parameters (column values) is retrieved. For instance, a query that requests the COVID-19 data in date 03/25/2020 is: *https://wake.dlsi.ua.es/IntegrationAPI/?date_of_interest_date=03/25/2020*. From this request, the Web API returns the required information in JSON format. The result obtained from this query example is the data about COVID-19 in the date specified, containing the following information:


```
{
 "date_of_interest_date": "03/25/2020",
 "publisher_publisher":
    "City of New York",
 "death_count_deaths_total": "123",
 "case_count_cases_total": "4864",
 "hospitalized_count_ventilators_
    in_use_covid_19_patients": "1303"
},
{
 "date_of_interest_date": "03/25/2020",
 "publisher_publisher":
    "City of Chicago",
 "death_count_deaths_total": "5",
 "case_count_cases_total": "368",
 "hospitalized_count_ventilators_
    in_use_covid_19_patients": "79"
}
```


IV. EVALUATION

This section describes the evaluation carried out on our approach. We evaluated three aspects of the system: the word embeddings-based similarity approach for tabular data integration, the automatic generation of Web APIs to access integrated data, and the execution time performance of the entire pipeline.

A. WORD-EMBEDDINGS FOR DISCOVERING JOINABLE AND UNIONABLE TABLES

This section presents an intrinsic evaluation of the word embeddings-based similarity approach for tabular dataset integration. The goal of these experiments is not only to evaluate the performance of the similarity approach, but also to identify the best model and parameters to be used in the subsequent API generation stage (described in Section IV-B).

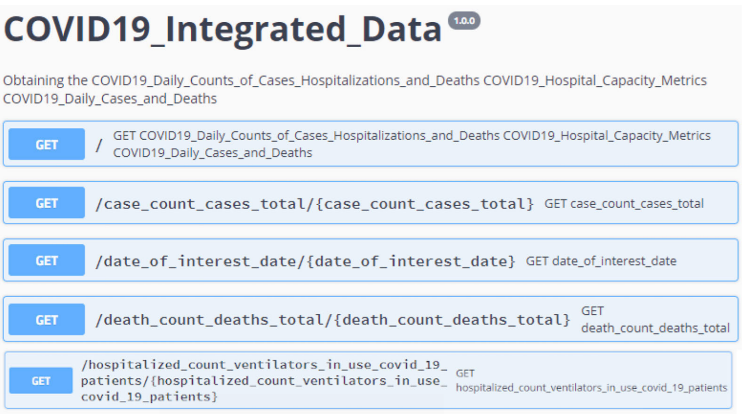


FIGURE 11. Interactive OpenAPI documentation of the API.

1) EXPERIMENTAL SETUP

In the evaluation carried out, we assimilated the task of computing similarity of tabular datasets to the task of *ad hoc* table retrieval: answering a search query with a ranked list of tables [20]. In this context, the search query is not a sequence of keywords but a table itself [21].

Given a set of tables T, the similarity $sim(t_1, t_2)$ for every pair $t_1, t_2 \in T$ is computed as:

$$sim(t_1, t_2) = \frac{\sum_{i=1,j=1}^{i\leq n, j\leq m} sim(c_{1i}, c_{2j})}{|C_1||C_2|},$$

where $C_1 = \{c_{11}, c_{12} \ldots, c_{1n}\}$ and $C_2 = \{c_{21}, c_{22} \ldots, c_{2m}\}$ are the set of columns of t_1 and t_2 respectively. Thus, the similarity between two tables is computed as the average similarity between their columns (calculated using the formula described in Section III-A1). For a given table, the system will return a ranked list of related tables based on this similarity measure.

The dataset used to test the approach was developed by Nargesian *et al.* [22] and it is publicly available for the evaluation of approaches related to tabular data integration. This database consists of more than 5,000 tables in CSV format extracted from USA, Canada, and UK open data portals, providing a ground truth that identifies which columns of a table match the columns of the other tables. The dataset was built starting with 32 base tables, which were manually aligned to identify matching columns. The final set was created by first issuing a projection on a random subset of columns of a base table, and then a selection with some limit and offset on the projected table. The tables contain the name of the columns and the corresponding content of the cells, comprising text, numeric, and date values.

Although the word embedding approach is specially suitable for textual data, our proposal also provides coverage for columns containing other data types. First, the name of the columns are textual data, even if its contents are numbers or dates. Thus, the system can return a similarity value based solely on the column names. Secondly, if the content of the columns is considered, even though all the possible numeric values are not represented in the embedding space, the word embedding models still provide coverage for many of them. For instance, the fastText model described below covers 99.90% of the numbers ranging from 0 to 10,000. This implies that any numeric value used to represent days, months, or years has a vector representation in the model.

In order to perform the experiments, a subset of 1,000 tables was randomly selected. Every table in this subset was used as a query to the system and compared with all the other tables in it, obtaining a ranked list of the most similar tables according to the measure given above. In our experiments, we consider a returned table to be relevant to a query table if there is at least one matching column between them in the ground truth provided.

In this evaluation, we use two pre-trained word embedding models and one task-specific model in order to decide the best configuration to incorporate in our pipeline:

- Word2vec:[16] embedding vectors pre-trained on part of Google News dataset, comprising about 100 billion words. The model contains 300-dimensional vectors for 3 million words and phrases [13], although in the experiments presented here only words are considered.
- fastText:[17] embedding vectors pre-trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset, comprising about 16 billion words [23].

[16] https://code.google.com/archive/p/word2vec/, accessed October 2020.
[17] https://github.com/facebookresearch/fastText, accessed October 2020.

As in the previous case, the vectors are composed of 300 dimensions.

- WikiTables: task-specific model using skip-gram Word2vec [24] trained on the Wikipedia Tables corpus, containing 1.6 million Wikipedia relational tables [25]. The corpus was pre-processed as mentioned in Section III-A1, splitting CamelCase and hyphenated words, removing punctuation, and converting text to lowercase. For every table in this corpus, all the names of the columns were extracted and treated as an input document to train Word2vec.[18] A second model was created for the content of the cells. In this case, all the attribute values in a column were considered as an input document to train the model. Thus, we have two separate word embedding models to calculate the similarity between names of the columns and the content of the cells. Again, vectors are composed of 300 dimensions.

In the case of fastText, we also conducted experiments with a pre-trained model containing subword information, but the results obtained were worse than the model presented above. We decided to remove these results from the following section for the sake of simplicity.

We computed the coverage of each of these models, calculated as the percentage of tokens in the 1,000 tables bechmark dataset that has a representation in the model. In the case of Google Word2vec, the coverage for names of the columns and content of the cells is 83.02% and 78.18% respectively. fastText provides a coverage of 87.69% and 80.91% for columns and cells. Finally, the coverage of WikiTables is 71.54% for columns and 78.19% for cells. These values reflect that fastText is the model offering the largest coverage, whereas WikiTables provides the lowest results. As mentioned in Section III-A1, the Levenshtein distance is used as a backup strategy when a word is not represented in the model.

2) RESULTS AND DISCUSSION

This section reports the precision of top-k table searches at different k. More precisely, P@10 and P@50 where computed, corresponding to the number of relevant results among the top 10 and top 50 documents retrieved respectively. This measure is in accordance with web-scale information retrieval systems, where thousands of relevant documents are available but no user will be interested in reading all of them. The final score is computed as the average precision for the 1,000 queries carried out, one for each table.

The experiments include the three word embeddings models mentioned above. In addition, we tested a baseline using BM25 [26], a classical ranking function widely employed by search engines to estimate the relevance of documents to a given search query. The implementation of the baseline was carried out using Apache Solr.[19]

In order to identify the best value for α, we tested each embedding model with different values for this parameter, from 0 to 1 inclusive, in 0.1 increments. In the case of the baseline, three different queries were employed in Apache Solr to simulate the ranking experiment done with the embedding vectors: a query that uses only the content of the cells (equivalent to $\alpha = 0.0$), a query that uses only the names of the columns (equivalent to $\alpha = 1.0$) and, finally, a query containing both (equivalent to $\alpha = 0.5$). Fig. 12 and Fig. 13 show the results for Google pre-trained Word2vec (*gl*), pre-trained fastText (*ft*), task-specific Word2vec trained on Wikipedia Tables (*wt*), and BM25 (*bm*).

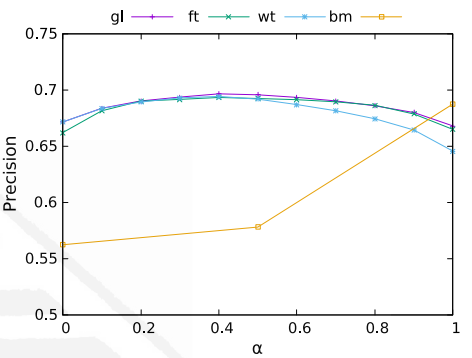


FIGURE 12. P@50 for each of the embedding models (*gl*, *ft*, and *wt*) and the baseline (*bm*).

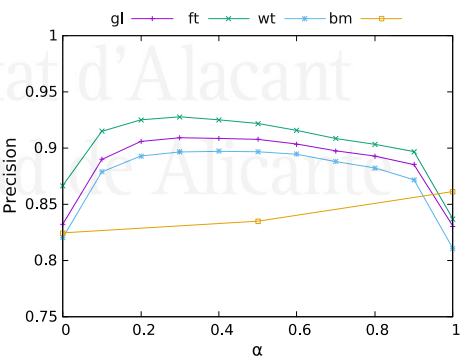


FIGURE 13. P@10 for each of the embedding models (*gl*, *ft*, and *wt*) and the baseline (*bm*).

The results at P@50 show that the three word embeddings models perform very close for every α value. The best result is obtained by Google pre-trained embeddings with P@50=0.6964 and $\alpha = 0.4$, which means that the content of the cells have a higher relevance in the final similarity score. The other two embedding models reach also its best performance with $\alpha = 0.4$. In the case of the BM25 baseline, it is worth noting that the results using only the content

[18] https://radimrehurek.com/gensim/models/word2vec.html
[19] https://lucene.apache.org/solr/.

of the cells are significantly lower than the word embeddings approaches. These results slightly improve when both names of columns and content of cells are taken into account ($\alpha = 0.5$), but it achieves the best results when only the name of the columns are considered ($\alpha = 1.0$), outperforming the other models.

Results at P@10 are encouraging in terms of the performance of the word embeddings models. The best results (0.9276) are obtained by the fastText model with $\alpha = 0.3$. This result improves 7.71% the performance of the baseline. The comparison between the three embedding models shows that its performance correlates with the coverage previously described. This highlights the importance of having a large enough corpus to properly build an effective word embeddings model for this task.

For the end user, the most important aspect is to improve the precision of the first results obtained by our approach. Therefore, P@10 is the measure that we should try to maximise. For this reason, we choose the best performing configuration in this evaluation (fastText model with $\alpha = 0.3$) to tune the system for the API generation experiments described in the following section.

The retrieval phase described here was omitted in the running example, since we started directly having the tables we wanted to integrate. Nevertheless, we can compute the similarity $sim(t_1, t_2)$ between these tables to see how they would rank in an hypothetical retrieval task such as the one presented in this section. The similarity between the two tables from Chicago used in the join operation is 0.2872. This value reveals that the tables present an average low similarity, which is an expected result for join operations since most of their columns are not related. It should be noted that the formula is computing the similarity between all possible pairs of columns, which in this case involves 33 and 39 columns for a total of 1,287 pairs. Following our approach the tables were considered as joinable since one pair of columns achieved a similarity value over 0.95 as described in Section III-A2. The similarity between the resulting joined table (71 columns) and the New York table (4 columns) is 0.6304, showing significantly higher similarity than the previous case. The reason is that the number of column pairs to compare is lower (284) and there are more related pairs of columns. Indeed, all the columns in the New York city table presented a similarity over 0.75 with the corresponding columns in the Chicago table, resulting in the union of both tables as established in our methodology.

Although the task proposed in this section is similar to that defined by Nargesian *et al.* [22], the results are not directly comparable. While they try to maximise the performance on their own definition of "unionability" (involving all the unionable columns of a table), our goal is to address both union and join operations where tables have at least one matching column in common. Taking into account these differences, the authors obtained a P@10 value of around 0.8 with an approach based solely in word embeddings, which improved to about 0.95 when combined with semantic information from YAGO [27] and the actual set of values from the tables. Thus, our best result (0.9276) is in a similar range to those obtained by Nargesian *et al.* with their best performing ensemble of features, whereas we are using only embedding information at this stage.

B. GENERATING WEB APIs FOR ACCESSING INTEGRATED OPEN DATA

In this section, we provide an evaluation of the approach for automatically generating Web APIs to access integrated data. This evaluation measures the quality by considering the number of correctly generated APIs for a set of input tabular datasets.

1) EXPERIMENTAL SETUP

We use again the benchmark described by Nargesian *et al.* for this evaluation. As mentioned before, this dataset was generated starting from a set of 32 base tables on which the authors performed different selections and projections on various sizes to obtain the final set of 5,000 tables. From this set of 32 base tables, only 27 were involved in the generation of the 1,000 random subset used in the previous evaluation (see Section IV-A). With the aim of aligning both experiments, we focused on these 27 base tables for the evaluation done in this section.

In order to test the performance of our approach on both union and join operations, each table was divided into three subtables by using projection and filtering as reverse operations of join and union, respectively. The goal of the system in this experiment is to automatically generate a Web API for accessing the integration of these subtables by using first join and then union operations.

Two experiments were conducted either considering columns names or not when calculating the similarity between tables. The rationale behind this is to avoid bias regarding the fact that subtables coming from the same table have the same column names, and this would not be the usual situation in a real scenario. Thus, the first experiment considers only the content of the cells, and the second experiment uses both content and column names for computing the similarity between tables.

2) RESULTS AND DISCUSSION

The results of the experiments for evaluating the generated Web APIs are shown in Table 6. The column *Original Web API functions* represents the expected functions to be generated for each of the 27 base tables in the benchmark. The columns *Correctly generated Web API functions* indicates the number of functions that were correctly generated by our proposal in the two settings proposed: with and without column names. *Precision* is defined in this case as the number of functions of the API generated from the integrated tables compared to the number of functions offered by the API from the original tables. In the final results, we consider both micro-averaged precision (i.e. aggregating the functions of all the tables to compute the average) and macro-averaged

TABLE 6. Results of the Web API generation experiments.

Table	Original Web API functions	Correctly generated Web API functions (without column names)	Precision	Correctly generated Web API functions (with column names)	Precision
1	12	10	0.83	11	0.92
2	7	6	0.86	7	1.00
3	7	3	0.43	7	1.00
4	13	3	0.23	9	0.69
5	6	6	1.00	6	1.00
6	22	21	0.96	22	1.00
7	6	6	1.00	6	1.00
8	13	13	1.00	13	1.00
9	13	4	0.31	11	0.85
10	14	14	1.00	14	1.00
11	19	16	0.84	17	0.90
12	23	23	1.00	14	0.61
13	20	20	1.00	19	0.95
14	21	21	1.00	17	0.81
15	23	13	0.57	23	1.00
16	13	8	0.62	8	0.62
17	16	3	0.19	3	0.19
18	9	9	1.00	9	1.00
19	28	24	0.86	25	0.89
20	13	1	0.08	4	0.31
21	19	8	0.42	19	1.00
22	26	24	0.92	26	1.00
23	11	6	0.55	9	0.82
24	19	16	0.84	16	0.84
25	17	13	0.77	17	1.00
26	22	22	1.00	22	1.00
27	12	10	0.83	12	1.00
All	424	323		366	
	Micro-averaged precision		0.76		0.86
	Macro-averaged precision		0.74		0.87

precision (i.e. precision is computed for every table and then averaged). Macro-averaged precision allows treating all the APIs generated equally regardless of the number of functions they have.

The value of micro-averaged precision obtained is 0.76 when column names are not consider. This value rises to 0.86 when the column names are taken into account in the similarity measure formula. Macro-averaged precision is 0.74 if column names are not considered and 0.87 otherwise. In both cases, the use of column names improves the results as expected, since they are clear cues to identify whether two columns match in the similarity algorithm proposed. It is worth noting that using only the content of the cells allows generating the correct API functions about 75% of the time.

In order to clarify why the system fails to generate the correct API functions in some situations, we analysed in more detail the specific case of table 3, where our approach obtained perfect precision using the names of the columns but dropped to 0.43 when using only the content of the cells. This table contains data about ridership from Chicago Transit Authority.[20] The API generated for the original dataset has to include the 7 functions that are shown in Fig. 14.

FIGURE 14. Functions of the Web API generated to access data from table 3.

The API automatically generated considering column names has the same 7 functions, whereas the API generated using only content of the cells fails to produce the following 4 functions:

```
/alightings/{alightings}
/boardings/{boardings}
/cross_street/{cross_street}
/on_street/{on_street}
```

When only the content of the table is considered, the similarity algorithm incorrectly matches the columns "alightings" and "boardings" with a value of 0.9996, while the similarity with the correct column "alightings" is 0.9991. The reason is that both columns have very similar values as the concepts are closely related ("boarding" means going into a bus, while "alighting" means going out of the bus), making it difficult for the algorithm to discriminate between the two cases.

[20] https://www.transitchicago.com/

TABLE 7. Description of the dataset and execution time performance (in seconds) for the integration and generation processes.

Table name	Rows	Columns	Rows selected		Integration time	Generation time
Salmonella tests	13	3	13	(100%)	0.0028	9.7
Population	441	5	30	(6.8027%)	0.0059	9.7
Travel data	1,853	20	70	(3.7776%)	0.0608	10.3
Biodiversity	20,017	12	30	(0.1499%)	0.0184	10.3
Street lights	27,409	15	20	(0.0730%)	0.0138	10.5
International payments	245,107	17	20	(0.0082%)	0.0133	10.9
Traffic state	3,565,683	4	20	(0.0006%)	0.0030	13.4
Voter data	7,517,745	46	700	(0.0093%)	0.6762	29.2
Wholesale trade sales	8,752,568	17	450	(0.0051%)	0.3176	35.1
Employee earnings	21,072,480	18	150	(0.0007%)	0.1318	80.0

Another example is the column ''cross street'', which is incorrectly matched to ''on street'' with a similarity of 0.9914, while the similarity with the correct column ''cross street'' is 0.9833. Again, the reason is that both columns have similar values as the concepts are related (''on street'' refers to the street where the bus stop is, while ''cross street'' refers to the nearest crossing street).

In this situation, when the contents of the cells are very similar, it is required the use of additional metadata such as the names of the columns to achieve a good performance in the automatic generation of the API.

In the case of the running example, using the best configuration (fastText model with $\alpha = 0.3$) all the API functions expected were correctly generated for the resulting tables after join (72 functions, including the *publisher* column) and union (5 functions, including also the *publisher* column) operations, achieving a perfect precision of 1.00 in this experiment.

C. EXECUTION TIME PERFORMANCE

This section analyses the execution time performance of the system proposed, providing a reference on the time required to carry out the integration and generation processes from a CSV file, and discussing their implications in a production environment.

1) EXPERIMENTAL SETUP

In order to evaluate the execution time performance of the integration and API generation processes, two experiments were carried out using 10 different CSV datasets.[21] These files were retrieved from different open data platforms, representing an heterogeneous set in terms of their size and nature. The content of the tables includes texts, numbers, and dates. The amount of rows in the tables ranges from 13 (table ''Salmonella tests'') to over 21 millions (table ''Emloyee earnings''), and from 3 columns (table ''Salmonella tests'') to 20 (tables 'Travel data'' and ''Voter data''), resulting in a minimum of 39 cells and a maximum of 379,304,640. The specific number of rows and columns for each CSV file is shown in Table 7.

[21]Datasets available in GitHub repository: https://github.com/cgmora12/DataIntegration2API#example-datasets.

The measurement of the execution time of the experiments was carried out in a desktop computer running Windows 10, equipped with an Intel i7 processor and 16 GB of RAM.

2) RESULTS AND DISCUSSION

First we evaluated the time elapsed during the integration process. In this phase calculating the embeddings for each column of a table is the bottleneck of the task. Once the embeddings are obtained, computing the similarity between two tables is reduced to calculate the inner product between 300-dimensional vectors representing their columns, which can be computed in a minimal time. Thus, in this first experiment we have focused on determining the time taken to calculate the embedding vectors for each table.

This task is time consuming when the tables have a large number or rows, since the embedding of each column is computed as the average of the embeddings of its content. For instance, in the example table ''Traffic state'' containing 3,565,683 rows and 4 columns, the process of obtaining the embedding vectors takes 423.41 seconds in the hardware setting mentioned above, which can be unsuitable for a production environment.

To reduce the time required for this process, we first analysed to what extent the number of rows could be reduced without affecting the embedding representation of the table, taking into account that in many cases columns contain repeated values that could be removed seamlessly. For each table we randomly choose samples of rows of different sizes[22] (ranging from 1 to 20,000 in increments of 10) and obtained the corresponding embeddings. Then, the similarity measure $sim(t_1, t_2)$ defined in Section IV-A1 was used to compare how similar were the reduced version of the table and the full version containing all the rows. The results of this experiment revealed that, in all the cases, we were able to obtain a reduced version of the table that was 99% similar to the original one by considering a small sample of the rows.

Table 7 shows the number of rows and the percentage it represents from the original (column *Rows selected*) that was required for each table to exceed the 99% similarity threshold. For instance, table ''Traffic state'' contains over 3 million rows, but the reduced version has only 20 rows as

[22]Except for ''Salmonella tests'', which contained only 13 rows.

a result of having a small number of columns (only 4) and a large number of repeated values in them. The table that required a larger number of samples was "Voter data", with 700 rows. In all the cases, the time taken to obtain the embedding representation of the reduced table was below one second (column *Integration time*). For the largest tables (above 200,000 rows), the percentage of rows kept from the original was below 0.01%. This value could be established as a fixed threshold in a production environment to ensure that the tables can be processed in real time while retaining (at 99%) their original representation in the word embedding space.

Regarding the evaluation of the API generation process, the results presented in Table 7 (column *Generation time*) show that the automatic process takes between 9.7 and 80.0 seconds to generate a complete Web API, including the related models and interactive OpenAPI documentation. This evaluation was carried out using all the rows in the tables to identify how this number affects the execution time of the generation module. If we took the reduced tables of the previous experiment, where the largest table contains 700 rows, the generation time would be comprised between 9.7 and 10.3 seconds in all cases.

Fig. 15 shows a linear behaviour considering the number of rows of the tables. The process is able to generate the Web API and its related components (models and documentation) in an average time of 16 seconds. The results indicate that, as the number of rows in the source file increases, the time to generate the Web API also increases in a ratio of 3 seconds per million rows.

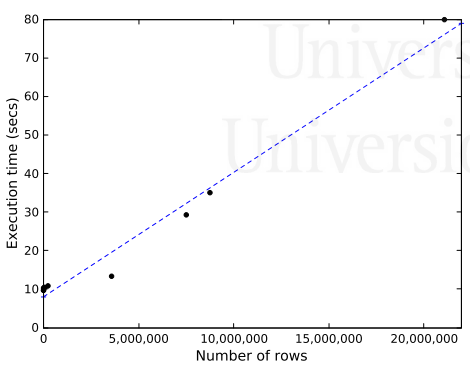


FIGURE 15. Generation time of the API depending on the size of the tables.

Regarding the performance of the generated APIs in returning the results, when the number of rows is really high (such in the case of "Employee earnings") the API takes a time similar to the generation performance shown in Table 7. This situation can be overcome by the pagination of the results using "limit" and "offset" parameters: "limit" indicates the maximum results to show, whereas "offset" indicates the starting point of results bypassing the records until the specified offset. This pagination auto-generated by our APIfication approach optimises the performance of the API and reduces the time to get the results. With this mechanism, APIs are able to return results from datasets of up to 21 million rows in less than 4 seconds. For example, a query that specifies a limit of 10,000 results to the Web API that manages the largest dataset ("Employee earnings") takes around 3 seconds to return the results to the browser.

V. RELATED WORK

The following paragraphs summarise the existing related work. Specifically, we focus on two areas: (i) word embeddings for data integration; and (ii) Web API generation for open data access.

A. WORD EMBEDDINGS AND TABULAR DATA INTEGRATION

In recent years, word embeddings have enjoyed widespread use in a variety of semantic tasks in the field of Natural Language Processing, such as sentiment analysis [28], machine translation [29], text classification [30], and dialog systems [31].

Prior works related to tabular data considered word embeddings as a means to represent the content of the tables. In the task of table retrieval, consisting on answering a search query with a ranked list of tables, Zhang *et al.* [32] used pre-trained word and entity embeddings combined with different similarity measures to beat a strong learning to rank baseline. The work by Deng *et al.* [20] considered different table elements (caption, column headings, and cells) to train word embeddings that were utilised in three table-related tasks: row population, column population, and table retrieval. In a similar vein, Nargesian *et al.* [22] defined a semantic measure based on word embeddings that were trained on Wikipedia documents. The authors defined natural language domains and statistical tests between the vectors that were used to evaluate the likelihood that two attributes were from the same domain.

In the work presented in this paper, we follow a similar approach to previous research using pre-trained word embeddings, but introducing as a novelty the inclusion of task-specific embeddings based on a large dataset of tabular data. More importantly, unlike previous works that treat data integration as an isolated process, we include this procedure in a pipeline to automatically generate Web APIs for the integrated data.

B. WEB APIs FOR OPEN DATA ACCESS

Web APIs are created to make accessing open data easier for developers. However, this process has been usually done manually [33], [34] as a time-consuming task.

The automatic generation of APIs has been addressed in recent studies. EMF-REST [35] is a framework for generating Web APIs that needs a model of the API to create it, requiring users to build the model by themselves since it is not generally

available. Another work [36] focused on guiding developers in creating Web APIs for accessing required data. However, these approaches are not targeting the access and reuse of integrated open data coming from different sources.

Other works propose the use of model-driven approaches for API definitions. The models are used to represent the Web API definition, offering a better visualisation of the API operations [37], [38]. Also, the metamodel of the API definition is used to simplify the transformation between the API and its definition to include it in the OpenAPI initiative [39]. The API2MoL engine [40] creates bridges between APIs and model-driven engineering, with the objective of creating models from the APIs for facilitating the management of a plethora of APIs.

These model-based approaches for generating Web APIs can be used to represent and generate the Web API documentation, but they are not employed to generate the whole Web API to access integrated data as proposed in this paper.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented an approach to address the problem of accessing integrated open data from different sources by using a unique end point. Specifically, we have proposed an APIfication approach which aims to facilitate the integration and access to tabular datasets. Our APIfication approach has two parts: (i) a word embeddings-based approach that uses column similarity to determine which datasets can be integrated by using union and join operators; and (ii) a model-driven approach for automatically generating a Web API to access and reuse open data in an integrated manner, as well as the documentation of the Web API to support users in consuming the integrated data easily through a Web interface.

In the experiments conducted, the proposed similarity measure based on word embeddings achieved precision values over 0.92 in the task of retrieving unionable/joinable tables given a query table. When this measure was included in the API generation pipeline, our approach was able to automatically integrate the tables and generate the expected functions with a micro-averaged precision of 0.76 using only the content of the cells, and 0.86 precision when the names of the columns were also taken into account. These results are promising, even more so considering that the whole data integration and API generation process is fully automated. The execution time performance experiments revealed that selecting a small number of rows allowed to obtain the word embeddings representation for large tables in real time, while keeping the table representation almost unchanged in the vector space (99% similarity with respect to the original table). Additionally, the API generation procedure for tables of more than one million rows took around 10 seconds to complete, which makes the whole pipeline suitable for a production scenario.

As a future work, we plan to extend our approach by considering more operators for data integration in addition to union and join. We also plan to develop mechanisms to guide users in the application of the approach, for example, by defining GUIs to allow developers to provide semantic hints about the open data to be integrated. Moreover, we plan to investigate how metadata coming from the W3C CSV on the Web Working Group[23] can improve our approach, as well as whether it is a motivation for publishers to properly describe their tabular open data.

Regarding the similarity module, future experiments are required in order to determine the best performing combination of models, i.e. using one model (e.g. Word2vec) to calculate the similarity between column names and another one (e.g. fastText) for the content values. The backup strategy based on Levenshtein distance can also be improved by using subword embeddings that can handle the problem of out-of-vocabulary terms, such as the subword version of fastText mentioned before, or models using Byte-Pair Encoding (BPE) [41]. This approach could benefit the system when the content of the tables include numbers and dates, as it provides better coverage than word-based models that could offer limited representation of numeric values. An interesting path for further research is the use of contextual word embedding models such as BERT [42] and its derivatives (e.g. ALBERT [43], RoBERTa [44], and DistilBERT [45]). These models provide different vector representations for a term in the embedding space depending on the context (surrounding terms) where it occurs. They have demonstrated to achieve state-of-the-art results on a number of language understanding tasks, including question answering and natural language inference. Another issue that deserves attention is the improvement of the WikiTables task-specific model by gathering additional tabular examples. In the experimental evaluation we showed that this model offered a lower coverage than the other pre-trained models, which could be affecting its performance. Finally, another line of future work is the combination of the similarity functions provided by the word embeddings models with traditional information retrieval ranking functions such as BM25, which demonstrated to obtain better results than the word embedding models in those experiments where only the content of the tables was taken into account.

REFERENCES

[1] M. S. Altayar, "Motivations for open data adoption: An institutional theory perspective," *Government Inf. Quart.*, vol. 35, no. 4, pp. 633–643, Oct. 2018.

[2] H. Dong, G. Singh, A. Attri, and A. El Saddik, "Open data-set of seven canadian cities," *IEEE Access*, vol. 5, pp. 529–543, 2017.

[3] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data: The story so far," in *Linked Data: The Story so Far, in: Semantic Services, Interoperability and Web Applications: Emerging Concepts*. Hershey, PA, USA: IGI Global, 2011, pp. 205–227.

[4] S. Neumaier, J. Umbrich, and A. Polleres, "Automated quality assessment of metadata across open data portals," *J. Data Inf. Qual.*, vol. 8, no. 1, pp. 1–29, Nov. 2016, doi: 10.1145/2964909.

[5] E. Muñoz, A. Hogan, and A. Mileo, "Using linked data to mine RDF from wikipedia's tables," in *Proc. 7th ACM Int. Conf. Web Search Data Mining - WSDM*, 2014, pp. 533–542, doi: 10.1145/2556195.2556266.

[23] https://www.w3.org/TR/tabular-data-model/

[6] *Open Data Maturity Report 2019*, Publications Office of the European Union, European Commission, Brussels, Belgium, 2019.
[7] *Open Government Data Report: Enhancing Policy Maturity for Sustainable Impact, OECD Digital Government Studies (2018)*, Organisation for Economic Co-operation and Development, OECD, Paris, France, 2018.
[8] M. Arenas, F. Maturana, C. Riveros, and D. Vrgoč, "A framework for annotating CSV-like data," *Proc. VLDB Endowment*, vol. 9, no. 11, pp. 876–887, Jul. 2016.
[9] Y. Doi and M. Toyama, "ToT for CSV: Accessing open data CSV files through SQL," in *Proc. 21st Int. Conf. Inf. Integr. Web-based Appl. Services*, Dec. 2019, pp. 423–429.
[10] R. J. Miller, "Open data integration," *Proc. VLDB Endowment*, vol. 11, no. 12, pp. 2130–2139, 2018.
[11] A. Halevy, A. Rajaraman, and J. Ordille, "Data integration: The teenage years," in *Proc. 32nd Int. Conf. Very large data bases*, 2006, pp. 9–16.
[12] D. Abadi *et al.*, "The seattle report on database research," *ACM SIGMOD Rec.*, vol. 48, no. 4, pp. 44–53, Feb. 2020, doi: 10.1145/3385658.3385668.
[13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, Lake Tahoe, NV, USA, vol. 2, 2013, pp. 3111–3119.
[14] K. Braunschweig, J. Eberius, M. Thiele, and W. Lehner, "The state of open data limits of current open data platforms," in *Proc. 21st. WWW Conf.*, 2012, pp. 1–6.
[15] Z. S. Harris, "Distributional structure," *Word*, vol. 10, nos. 2–3, pp. 146–162, Aug. 1954.
[16] S. Gupta, T. Kanchinadam, D. Conathan, and G. Fung, "Task-optimized word embeddings for text classification representations," *Frontiers Appl. Math. Statist.*, vol. 5, p. 67, Jan. 2020.
[17] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Sov. Phys. Doklady*, vol. 10, no. 8, pp. 707–710, 1966.
[18] J. S. Cuadrado, E. Guerra, and J. de Lara, "AnATLyzer: An advanced IDE for ATL model transformations," in *Proc. 40th Int. Conf. Softw. Eng., Companion Proceeedings*, May 2018, pp. 85–88, doi: 10.1145/3183440.3183479.
[19] A. Srai, F. Guerouate, N. Berbiche, and H. Drissi, "An MDA approach for the development of data warehouses from relational databases using ATL transformation language," *Int. J. Appl. Eng. Res.*, vol. 12, pp. 3532–3538, 2017.
[20] L. Zhang, S. Zhang, and K. Balog, "Table2 Vec: Neural word and entity embeddings for table population and retrieval," in *Proc. 42nd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Jul. 2019, pp. 1029–1032.
[21] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu, "Finding related tables," in *Proc. Int. Conf. Manage. Data - SIGMOD*, 2012, pp. 817–828.
[22] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," *Proc. VLDB Endowment*, vol. 11, no. 7, pp. 813–825, Mar. 2018.
[23] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. for Comput. Linguistics*, vol. 5, pp. 135–146, Dec. 2017.
[24] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. 1st Int. Conf. Learn. Represent.*, 2013, pp. 1–12.
[25] C. S. Bhagavatula, T. Noraset, and D. Downey, "Tabel: Entity linking in Web tables," in *The Semantic Web—ISWC*. Cham, Switzerland: Springer, 2015, pp. 425–441.
[26] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford, *Okapi at TREC-3*, vol. 109. Gaithersburg, MD, USA: NIST Special Publication Sp, 1995.
[27] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proc. 16th Int. Conf. World Wide Web - WWW*, 2007, pp. 697–706.
[28] M. Giatsoglou, M. G. Vozalis, K. Diamantaras, A. Vakali, G. Sarigiannidis, and K. C. Chatzisavvas, "Sentiment analysis leveraging emotions and word embeddings," *Expert Syst. Appl.*, vol. 69, pp. 214–224, Mar. 2017.
[29] W. Y. Zou, R. Socher, D. Cer, and C. D. Manning, "Bilingual word embeddings for phrase-based machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2013, pp. 1393–1398.
[30] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, "From word embeddings to document distances," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 957–966.
[31] G. Forgues, J. Pineau, J.-M. Larchevêque, and R. Tremblay, "Bootstrapping dialog systems with word embeddings," in *Proc. NIPS Workshop Modern Mach. Learn. Natural Lang. Process.*, vol. 2, 2014, pp. 1–5.
[32] S. Zhang and K. Balog, "Ad hoc table retrieval using semantic similarity," in *Proc. World Wide Web Conf. World Wide Web - WWW*, 2018, pp. 1553–1562.
[33] I. Hopkinson, S. Maude, and M. Rospocher, "A simple API to the knowledgestore," in *Proc. Int. Conf. Developers*, vol. 1268, 2014, pp. 7–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2878379.2878381
[34] M. Rittenbruch, M. Foth, R. Robinson, and D. Filonik, "Program your city: Designing an urban integrated open data API," in *Proc. Cumulus Conf., Open Helsinki–Embedding Design Life*, 2012, pp. 24–28.
[35] H. Ed-douibi, J. L. C. Izquierdo, A. Gómez, M. Tisi, and J. Cabot, "EMF-REST: Generation of RESTful APIs from models," in *Proc. 31st Annu. ACM Symp. Appl. Comput. SAC*, 2016, pp. 1446–1453.
[36] R. Queirós, "Kaang: A RESTful API Generator for the Modern Web," in *Proc. 7th Symp. Lang., Appl. Technol.*, vol. 62, 2018, pp. 1:1–1:15.
[37] H. Ed-douibi, J. L. C. Izquierdo, F. Bordeleau, and J. Cabot, "WAPIml: Towards a modeling infrastructure for Web APIs," in *Proc. ACM/IEEE 22nd Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Sep. 2019, pp. 748–752.
[38] H. Ed-douibi, J. L. Cánovas, and J. Cabot, "OpenAPItoUML: A tool to generate UML models from OpenAPI definitions," in *Web Engineering*. Cham, Switzerland: Springer, 2018, pp. 487–491.
[39] H. Ed-douibi, J. L. Cánovas, and J. Cabot, "Example-driven Web API specification discovery," in *Modelling Foundations and Applications*. Cham, Switzerland: Springer, 2017, pp. 267–284.
[40] J. L. Cánovas Izquierdo, F. Jouault, J. Cabot, and J. García Molina, "API2MoL: Automating the building of bridges between APIs and model-driven engineering," *Inf. Softw. Technol.*, vol. 54, no. 3, pp. 257–273, Mar. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584911001984
[41] B. Heinzerling and M. Strube, "Bpemb: Tokenization-free pre-trained subword embeddings in 275 languages," in *Proc. 11th Int. Conf. Lang. Resour. Eval. (LREC)*, Miyazaki, Japan, 2018, pp. 2989–2993.
[42] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proc. 2019 Conf. North Amer. Chapter Assoc. Comput. Linguistics*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 4171–4186.
[43] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," *CoRR*, vol. abs/1909.11942, 2019. [Online]. Available: http://arxiv.org/abs/1909.11942
[44] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*. [Online]. Available: https://arxiv.org/abs/1907.11692
[45] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," *CoRR*, vol. abs/1910.01108, 2019. [Online]. Available: http://arxiv.org/abs/1910.01108

CÉSAR GONZÁLEZ-MORA is currently pursuing the Ph.D. degree with the Web and Knowledge Research Group, Department of Software, University of Alicante, Spain. His research interests include open data, Web augmentation, and the semantic Web and application programming interfaces. His work is funded by a contract with the Generalitat Valenciana of Spain and the European Social Fund for predoctoral training.

DAVID TOMÁS is currently a Lecturer with the Department of Software and Computing Systems, University of Alicante, Spain. He is the author of more than 70 scientific publications in international conferences and journals. He has participated in over 20 public and private projects, including EU-funded QALL-ME, FIRST, and SAM projects. His research interests include information retrieval, knowledge representation, information extraction, question answering, sentiment analysis, recommender systems, log analysis, and machine learning approaches to text categorisation.

IRENE GARRIGÓS is currently an Associate Professor with the Department of Software and Computing Systems, University of Alicante, Spain. She is also the Head of the Web and Knowledge Research Group, University of Alicante. Her research interests include open data, Web augmentation, Web modeling languages, personalization, and application programming interfaces.

JOSÉ JACOBO ZUBCOFF presents a wide teaching and research experience in the field of statistics, data mining, and its application to biology. He has more than 100 publications in which he has dealt with obtaining knowledge from a data source. He has done research in various fields of science, both in computing, biology, medicine, education, and social sciences. In addition, he has directed and participated in more than 20 competitive public projects financed by the Ministry of Economy and Competitiveness, the Generalitat Valenciana, the University of Alicante, and European and private projects, all of them contributing his knowledge about data analysis, data mining, and aiming at the democratization of knowledge.

JOSE-NORBERTO MAZÓN is currently an Associate Professor with the Department of Software and Computing Systems, University of Alicante, Spain. He is the author of more than 100 scientific publications in international conferences and journals. His research interests include open data, business intelligence in big data scenario, design of data-intensive Web applications, smart cities, and smart tourism destinations. He is also the Chair of the Torrevieja's Venue of the University of Alicante.

Chapter 7

Open Data Consumption Through the Generation of Disposable Web APIs

Paloma Cáceres García De Marina, José María Cavero Barca, Carlos E. Cuesta, Miguel Ángel Garrido, Irene Garrigós, César González-Mora, Jose-Norberto Mazón, Almudena Sierra-Alonso, Belén Vela, and José Jacobo Zubcoff. Open Data Consumption Through the Generation of Disposable Web APIs. IEEE Access, 9:76354–76363, 2021.

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 15, 2021, accepted May 15, 2021, date of publication May 20, 2021, date of current version June 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3082182

Open Data Consumption Through the Generation of Disposable Web APIs

PALOMA CÁCERES GARCÍA DE MARINA[1], JOSÉ MARÍA CAVERO BARCA[1], CARLOS E. CUESTA[1], MIGUEL ÁNGEL GARRIDO[1], IRENE GARRIGÓS[2], CÉSAR GONZÁLEZ-MORA[2], JOSE-NORBERTO MAZÓN[2], ALMUDENA SIERRA-ALONSO[1], BELÉN VELA[1], AND JOSÉ JACOBO ZUBCOFF[2]

[1]Department of Computer Science and Statistics, Rey Juan Carlos University, 28933 Madrid, Spain
[2]Department of Software and Computing Systems, University of Alicante, 03690 Alicante, Spain

Corresponding author: Irene Garrigós (igarrigos@ua.es)

This work was supported in part by the Access@City coordinated Research Project through the Spanish Ministry of Science, Innovation and Universities under Grant TIN2016-78103-C2-1-R and Grant TIN2016-78103-C2-2-R; in part by the Plataforma intensiva en datos proveedora de servicios inteligentes de movilidad (MoviDA) Project through Rey Juan Carlos University; and in part by the Recolección y publicación de datos abiertos para la reactivación del sector turístico postCOVID-19 (UAPOSTCOVID19-10) Project through the Consejo Social of the University of Alicante. The work of César González-Mora was supported in part by the Generalitat Valenciana, and in part by the European Social Fund under Grant ACIF/2019/044.

ABSTRACT The ever-growing amount of information in today's world has led to the publication of more and more open data, i.e., that which is available in a free and reusable manner, on the Web. Open data is considered highly valuable in situational scenarios, in which thematic data is required for a short life cycle by a small group of consumers with specific needs. In this context, data consumers (developers or data scientists) need mechanisms with which to easily assess whether the data is adequate for their purpose. SPARQL endpoints have become very useful for the consumption of open data, but we argue that its steep learning curve hampers open data reuse in situational scenarios. In order to overcome this pitfall, in this paper, we coin the term disposable Web APIs as an alternative mechanism for the consumption of open data in situational scenarios. Disposable Web APIs are created on-the-fly to be used temporarily by a user to consume open data. In this paper we specifically describe an approach with which to leverage semantic information from data sources so as to automatically generate easy-to-use disposable Web APIs that can be used to access open data in a situational scenario, thus avoiding the complexity and learning curve of SPARQL and the effort of manually processing the data. We have conducted several experiments to discover whether non-experienced users find it easier to use our disposable Web API or a SPARQL endpoint to access open data. The results of the experiments led us to conclude that, in a situational scenario, it is easier and faster to use the Web API than the corresponding SPARQL endpoint in order to consume open data.

INDEX TERMS Disposable Web APIs, open data, semantic annotation, SPARQL.

I. INTRODUCTION

The volume of data on the Web has, with the advent of the open data movement, exploded in the last few years. Open data is published in order to be freely accessible and reusable (without copyright restrictions) and is, therefore, usually considered highly valuable as situational data. Situational data [1] has a narrow focus on a specific area and, often, a short lifespan so as to add value to data owned (and controlled) by a small group of consumers with a unique set of needs. Examples of this might be data scientists who wish to analyze a company's sales with respect to weather conditions within a period of time, or a Web developer who needs to implement a prototype of an envisioned smart city app that employs traffic data.

There are roughly two ways in which to publish open data: (i) open data portals that focus on published tabular-form data (such as CSV files), and (ii) Linked Open Data that allow users to use Semantic Web technologies to access data on the Web in the same way as a database management system is used, i.e., by means of query languages such as SPARQL [2]. The current goal of any open data project is

The associate editor coordinating the review of this manuscript and approving it for publication was M. Anwar Hossain.

to provide a SPARQL endpoint that will enable the powerful and seamless querying of data. However, average open data consumers (such as data scientists or developers) lack the skills required to smoothly use the SPARQL query language and the underlying RDF (Resource Description Framework) data model, since they both have a steep learning curve. Moreover, recent surveys [4] conclude that no usable tool that could be used to support the whole Linked Data consumption process currently exists. Consequently, in a situational scenario, alternative mechanisms that will allow powerful open data consumption without any knowledge of Semantic Web technologies are, therefore, required [3], [5].

Our hypothesis is that, in situational scenarios, it is easier to have a Web API access to open data than a SPARQL endpoint. These Web APIs must specifically allow data consumers to access data easily and rapidly without wasting time processing tabular data sources (from CSV files or from Web sites) or learning how to query a SPARQL endpoint. They have been denominated as disposable Web APIs because they are created on-the-fly to be used temporarily by a user to consume open data as situational data. Moreover, the use of a disposable Web API allows a data consumer (data scientist or developer) to easily assess whether the data is adequate for his/her purpose, thus avoiding the complexity and learning curve of SPARQL and the effort of manually processing the data. However, developing disposable Web APIs requires to enrich tabular data sources with data semantics in the sense of the ''Model for Tabular Data and Metadata on the Web'' developed by the W3C CSV on the Web Working Group[i1] which proposes to add semantic annotations by means of separated metadata file to supplement tabular data. In this paper we, therefore, describe an approach with which to leverage semantic information from data sources so as to automatically generate easy-to-use disposable Web APIs that can be used to access open data in a situational scenario. Our specific contributions are the following:

- A process for the semantic annotation of tabular data sources from the Web.
- A model-driven approach that can be employed to ingest our semantic annotation and generate disposable Web APIs with which to query data sources.
- A controlled experiment carried out in order to compare the process of accessing data from the SPARQL endpoints with that of using our equivalent generated disposable Web APIs.

Throughout this paper, we use a real situational case study (i.e., a running example) based on data obtained from the public transport sector in Madrid (Spain), namely Metro Madrid data, in order to exemplify our approach. A data scientist working at a real estate company was willing to analyze the company's internal data regarding housing rental and incomes in Madrid by considering additional open data related to the accessibility of public transport, which helps create a more inclusive society that provides the same opportunities for all [11]. This would, therefore, allow the real estate company to attain new insights and make informed decisions with which to, e.g., provide adapted solutions to those of its customers with special mobility needs during a short-term marketing campaign. A disposable Web API with which to consume public transport accessibility data for Madrid was, therefore, required.

The remainder of the paper is as follows: Section II presents some related work, while Section III describes the process of automatically generating disposable Web APIs using semantic-annotated open data sources. Section IV provides a validation of the proposal by describing an experiment in which data consumption from a Web API and the use of SPARQL were compared, and finally, Section V shows a summary of our main conclusions and future work.

II. RELATED WORK

This section reviews some research works related to how Web APIs are useful as regards providing easier data consumption.

The starting point for our current work was the research described in [11], which was focused on adding semantics to existing data. In the present paper, however, we have used this step as a part of the whole process carried out to make it easier for users to access open data on the Web.

In [6], a simple API, which is used to provide access to the KnowledgeStore, a semantic web storage, is described. It is used by many developers that are unfamiliar with RDF and SPARQL technologies to build web applications that use this data. The developers have to compose a query using simple API methods, which is then converted into a SPARQL query so as to access the endpoint and obtain the data required. However, this API was created manually and is used only to access specific data from specific semantic web storage.

The authors of [19], [20] propose an automatic query-centric API for routine access to Linked Data. In these papers, they present and extend ''grlc'', a generic Linked Data gateway. The papers propose the generation of APIs that provide uniform API access to any Linked Data published in SPARQL endpoints, Linked Data Fragments servers, RDF dumps, or RDFa embedded in HTML pages. Unfortunately, a Github repository containing all possible SPARQL queries to the endpoint is required, signifying that a developer must manually write down all possible queries that might be unfamiliar with RDF and SPARQL queries.

Rather than using SPARQL queries, users can also use GraphQL-LD [21] queries to access RDF data, which are GraphQL queries enhanced with a JSON-LD context. This GraphQL-LD approach is a developer-friendly alternative to SPARQL, but only for experts in GraphQL APIs, signifying that developers must be familiar with this programming language. Instead, users employing our approach are provided with an easy-to-use standard API based on OpenAPI principles.

An approach to help Web developers access Linked data is proposed in [22]. The objective is to transform ontologies into

[1] https://www.w3.org/TR/tabular-data-model/

Ontology-Based APIs such that developers can easily query Linked Data. However, our approach is directly addressed toward developers who wish to access open data available on the Web, signifying that the starting point is not an ontology or an SPARQL endpoint from the Linked Data cloud.

In [7], a RESTful API with which to semantically augment the data that is being published by a sensor is proposed, but it is not useful as regards retrieving data from the semantic web. The authors have manually created a system that transforms a set of inter-linked JSON documents into an RDF graph in order to obtain a semantically-enriched dataset and have the full query capability provided by SPARQL. However, an API that facilitates access to the semantic data without advanced knowledge of the SPARQL query language is still required.

The authors of [8] describe how cities are starting to release their public information and create public data catalogues, but there is still a lack of open APIs, which is restricting the full potential of the open data. The paper in question therefore proposes the development of an open data API, which provides tools that will help users to access the data concerning their own cities.

In [9], a framework called Prov4J is described, which uses Semantic Web tools and standards. It provides to developers a provenance management mechanism with which to build provenance-aware applications. Tracking the origin of information currently plays a fundamental role on the Web because it enables users to determine the suitability and quality of the data. The Prov4J framework includes a Client API that contains key interface methods for a set of provenance queries, thus minimizing the users' interactions with SPARQL. However, this API was created manually for data provenance purposes only.

Finally, mapping strategies have been investigated to a significant extent, and work researching conversion from other formats to RDF has also been carried out for static [23] and live conversion [24]. Other solutions [3], [5] aim to reduce the difficulty involved in using SPARQL and RDF technologies. However, they still lack the support specifically required by data scientists and open data reusers to overcome the difficulty of accessing RDF data using SPARQL. According to experts (as detailed in [3]), the problem does not reside in RDF, and these solutions may not be suitable for non-experts in this Semantic Web environment. It is even more necessary in a situational scenario, where data consumers need to temporarily access data easily and rapidly without spending too much time learning how to query the SPARQL endpoint.

In our approach we, therefore, provide an alternative through the use of standard and easy-to-query APIs rather than SPARQL endpoints that access RDF data.

In summary, there is much research on the creation of Web APIs with which to access data, but there is, to the best of our knowledge, no solution that tackles situational scenarios in which thematic open data is temporarily used for specific needs.

III. OPEN DATA CONSUMPTION PROCESS FOR SITUATIONAL SCENARIOS

The proposed process for open data consumption through the use of automatically generated disposable Web APIs comprises three stages:

1) **Data Input Process:** in this stage, it is necessary to select the Web data from either open data sources or data embedded in the HTML code of a Web site (specifying the Web data source URL), required to satisfy data consumer needs according to a situational scenario. This data must be processed (e.g., by programming a Python script in order to obtain the data by means of web scraping techniques). We would like to point out that we consider, as input of this process, Web data sources with no semantic information.

2) **Semantic Annotation Process:** the input tabular data (e.g. a CSV file) is annotated using a domain-specific ontology, obtaining semantically annotated data. The output of this process is a CSV file with semantic annotations as RDF triples.

3) **Disposable Web APIs Generation Process:** the input of this process is the previously semantically annotated tabular data, from which disposable Web APIs are automatically generated in order to allow the consumption of data in a situational scenario.

The complete open data consumption process is shown in Fig. 1, including the inputs and outputs of each process.

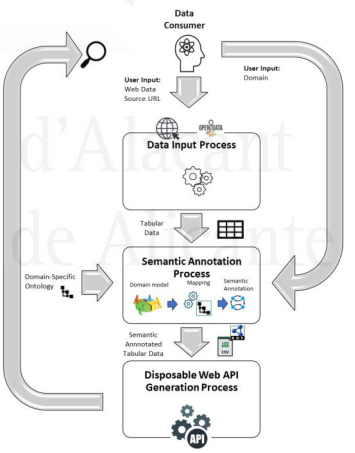


FIGURE 1. Open data consumption process for situational scenarios.

The Semantic Annotation of the Structured Open Data Process (subsection III.A) and the Automatic Generation of Disposable Web APIs (subsection III.B) are described in the following subsections.

A. SEMANTIC ANNOTATION OF STRUCTURED OPEN DATA

In this section, we describe the systematic process employed to generate a semantically annotated dataset from the

structured open data on the Web. This method makes it possible to enrich data by adding new semantics.

The process entails the following steps:

Step 1. First, information contained in the data source and important features of the selected domain must be studied. It is then necessary to create a glossary of terms concerning the terminology used in the data source, describing each one, after which a domain model with which to represent these terms and their relationships must be defined in a domain model by using a UML class diagram.

Step 2. Identify the semantics of the information in order to align them with the terms of a reference vocabulary, that is, a vocabulary that the developer needs to choose (e.g. domain-specific ontology). At this point, when necessary, auxiliary vocabularies can be chosen so as to cover additional properties. The mappings required in order to establish the correspondence between different information elements from the data source (glossary of terms) and the reference vocabulary (domain-specific ontology) are defined by means of a manual inspection of the input data source. We specifically identify the subjects, predicates and objects of a CSV input file and map them onto elements from the reference vocabulary (e.g. a domain-specific ontology, such as MAnto). This mapping is manually defined and added to a configuration file. Once all the mappings have been defined, a custom script is programmed in order to transform the input tabular source data (CSV file) into the semantically annotated data, according to the mappings defined in the configuration file. This data annotation process is described in a detailed manner by means of a case study shown in our previous work [11].

Step 3. If the alignment is possible, then it is necessary to use the original data, and semantically annotate them using the existing terms from the reference vocabulary (domain-specific ontology) by means of the RDF language using the previously defined mappings. If the alignment is not possible, then we should analyze the difference between the original data and the reference vocabulary in order to extend it and, in this case, return to the second step.

Step 4. Integrate the semantic dataset with other sources. This is done by relating elements from the different semantic datasets and discarding duplicated information in the datasets, when necessary. At this point, it is also possible to add new properties to the final annotated dataset. Finally, a single annotated dataset is obtained as the output of Step 4. It is worth mentioning, in the case of integrating with non-semantic data sources, that the semantic annotation process from step 1 to step 3 has to be performed beforehand.

This generic process has been applied to our case study about the public transport domain, and specifically to Metro Madrid.

During the **Data Input Process**, we first selected the open datasets available in the Madrid Public Transport open data portal. We then developed a Python script in order to obtain the Metro Madrid accessibility data by means of web scraping techniques, thus completing our input dataset.

We subsequently semantically annotated the real Metro Madrid public transport company data (by applying aforementioned steps 1, 2 and 3); in this case, there is no previously annotated data and we, therefore, omit step 4.

An explanation of each of the steps in the Semantic Annotation Process applied to our case study is provided as follows.

In **Step 1** of the semantic annotation of structured open data, we created a glossary of terms concerning the terminology used in the data source, describing each one. We then defined the domain model using a UML class diagram with which to represent these terms and their relationships (see Fig. 2).

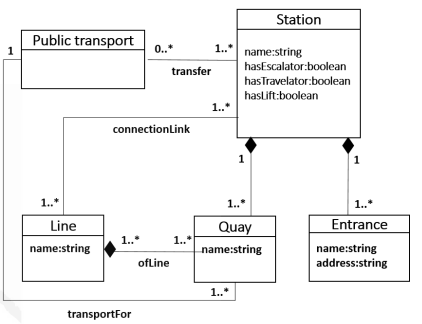


FIGURE 2. Domain model of metro Madrid.

In **Step 2**, we first choose our reference vocabulary, which is denominated as MAnto [12], [13]. MAnto[2] is a light ontology based on the Identification of Fixed Objects in the Public Transport (IFOPT) reference data model (CEN/TC278 2012)[14], included in the European Reference Data Model for Public Transport Information (Transmodel) [15]. The reason for using the MAnto ontology as a reference vocabulary in our case study is twofold: (i) first, we wished to describe a real scenario, and what is more important, (ii) we required a well-known ontology with which to conduct the experiments in a real setting. We then established the mappings between the elements of the Metro Madrid and the chosen reference vocabulary, MAnto. For example, we manually identify Sol as a station (mao:StopPlace subject) and 1, 2 and 3 as lines (mao:Line objects), and there are three connection links (mao:hasConnectionLink properties) that relate the subject (Sol) with the objects (lines 1,2 and 3).

In **Step 3**, we annotated the data semantically using the terms from our reference vocabulary (MAnto ontology) as follows: if the alignment is possible: (a) we first extract the data from the data source by means of a Python scraper; (b) we semantically annotate them (for example, each quay belonging to Line 1 of Metro Madrid has been annotated as `?quay mao:ofLine "1"`) and (c) we reuse terms from

[2]https://github.com/vortic3/LinkedUnifiedDataset/blob/master/MAnto_Lite_ontology.rdf

other vocabularies only when necessary (for example, if the name term exists in the `schema` ontology, we do not create the same term in MAnto -mao:name metadata-, but use that which already exists: in the RDF triple "`43566 sch:name Sol`", we assign the name of Sol to station 43566 using a metadata property that already exists in the schema ontology).

The output of the Semantic Annotation Process will be a CSV file with semantic annotations as RDF triples, which will be used as input to the Disposable Web APIs Generation Process.

B. AUTOMATIC GENERATION OF DISPOSABLE WEB APIs

In this section, a complete model-based transformation process is proposed in order to achieve the automatic generation of a disposable Web API from the previously semantic-annotated data sources (an overview of this process is shown in Fig. 3). The transformation process includes (apart from the RDF annotation using the semantic engine, as explained above), the following steps: (a) a text to model (T2M) transformation from the annotated data source to a specific data model (specific to the source format) that we have defined specifically for tabular data in CSV format; (b) a model to model (M2M) transformation from this data model to an OpenAPI model that we have also defined; (c) a model to text (M2T) transformation from the OpenAPI model to its OpenAPI specification, and finally, (d) a text to text (T2T) transformation from the OpenAPI specification to the Web API that makes it possible to access open data.

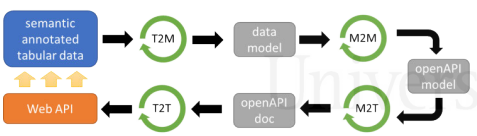


FIGURE 3. Automatic API generation process.

1) FROM DATA SOURCE TO DATA MODEL (T2M)

The first stage of the transformation process is a text to model (T2M) transformation. The data source is converted into the data model in order to represent the data and proceed with the model-based transformation approach. This data model is based on a metamodel based on MOF0F[3] (shown in Fig. 4) which is implemented in the Ecore format from the Eclipse Modeling Framework (EMF).

The input semantic annotated tabular data in our case study consists of: (a) the Madrid metro accessibility data source, which is a CSV file, and (b) the semantic annotations as RDF triples. Our approach processes the CSV by rows and columns, injecting the semantic information into the columns names. The data is analyzed, and a data model with table, row, and cell objects is created. This generated model (as shown in Fig. 5) contains an object 'Table', in which there is a set

[3]https://www.omg.org/mof/

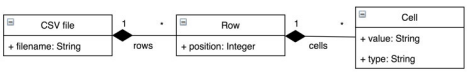


FIGURE 4. CSV datafile metamodel.

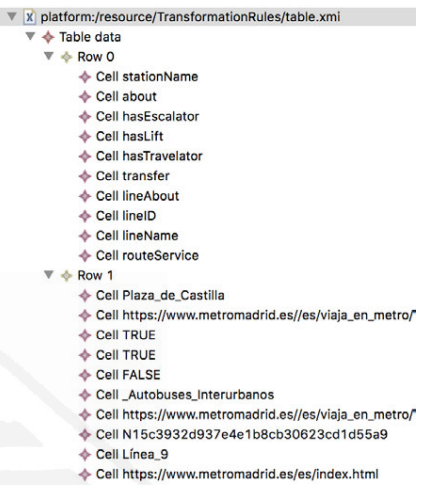


FIGURE 5. Datafile model in XMI format.

of 'Rows' containing many 'Cells'. Each cell in the model contains the information regarding each cell from the CSV file, which is the information concerning metro stations and their accessibility.

2) FROM DATA MODEL TO OpenAPI MODEL (M2M) AND SPECIFICATION (M2T)

Once the data model has been populated from the data file, the second stage involves a model to model (M2M) transformation from the CSV data model to the OpenAPI model, followed by a model to text (M2T) transformation between the OpenAPI model and the OpenAPI specification.

First, an M2M transformation defined in ATL language [16]–[18] is launched. ATL is one of the most widely used model transformation languages currently employed, and is backed by a mature and efficient execution runtime. A set of transformation rules between the data model and the OpenAPI model has, therefore, been defined using the ATL language, as shown in the extract of the code in Fig. 6. The ATL transformation rules start from the Table object defined in its Ecore metamodel, and its rows and cells are used to generate the OpenAPI model and all the different objects contained in its metamodel.

The OpenAPI model generated is in XMI format, as shown in Fig. 7.

It is based on its OpenAPI metamodel in Ecore format (Fig. 8), which has been created by updating an existing

```
-- @atlcompiler emftvm
-- @path Table=/TransformationRules/Table.ecore
-- @path Openapi=/TransformationRules/Openapi.ecore
module Table2Openapi;

create OUT: Openapi from IN: Table;
rule Main {
    from
        s: Table!Table
    using {
            firstTableRow : Sequence(Table!Cell) = s.rows->first().cells;
            lastTableRow : Sequence(Table!Cell) = s.rows->last().cells;
    }
    to
        t: Openapi!API (
            openapi <- '3.0.0',
            info <- openapi_info,
            servers <- Sequence{openapi_servers},
            paths <- Sequence{openapi_basic_path}.union
                (firstTableRow -> collect(e | thisModule.OpenapiPaths(e, s))),
            components <- Sequence{openapi_components}
        ),
        openapi_info: Openapi!Info (
            title <- s.filename,
            version <- '1.0.0',
            description <- 'Obtaining the ' + s.filename
        ),
        openapi_servers: Openapi!Server (
            url <- 'http://www.urlprueba.com/v1'
        ),
```

FIGURE 6. ATL transformation rules.

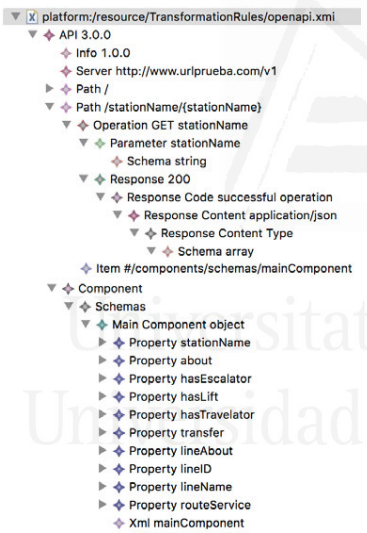


FIGURE 7. OpenAPI model (XMI).

OpenAPI metamodel2F[4] from Swagger 2.0 to OpenAPI 3.0 specification.

The definition and documentation of the Web API are represented by a JSON file (Fig. 9) in accordance with the Swagger standards3F,[5] because this helps design, build, document and test the API. The API specification JSON file is, therefore, directly generated from the OpenAPI model in XMI format by using a model to text (M2T) transformation. It consists of a straightforward transformation, since the

[4]https://github.com/SOM-Research/APIDiscoverer/tree/master/metamodel
[5]https://swagger.io

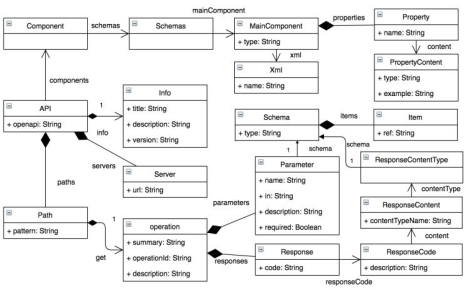


FIGURE 8. OpenAPI metamodel.

```
"openapi":"3.0.0",
"info":{
   "description":"Obtaining the metro stops",
   "version":"1.0.0",
   "title":"stopsmetro"
},
"servers":[
   "https://wake.dlsi.ua.es/madrid/metro"
],
"paths":{
   "/":{ },
   "/stationName/{stationName}":{ },
   "/about/{about}":{ },
   "/hasEscalator/{hasEscalator}":{ },
   "/hasLift/{hasLift}":{ },
   "/hasTravelator/{hasTravelator}":{ },
   "/lineAbout/{lineAbout}":{ },
   "/lineID/{lineID}":{ },
   "/lineName/{lineName}":{ },
   "/routeService/{routeService}":{ },
   "/transfer/{transfer}":{ }
}
```

FIGURE 9. Extract of OpenAPI JSON file.

OpenAPI model has elements that are equivalent to the API specification, but in a different structure because a different format is used. We, therefore, perform this M2T transformation programmatically from the OpenAPI model in XMI format to the OpenAPI specification of the API, which is in JSON format. It is easy to perform the transformation because we have constructed the OpenAPI model according to the OpenAPI specification, such that each object from the model corresponds to a JSON object in the file. For example, each path of the API is transformed from the "Path" object in the OpenAPI model to the "paths" object in the JSON file (e.g., Path /stationName/{stationName} in Figure 7 is transformed in the element "/stationName/{stationName}" from attribute path in JSON file of Figure 8). The complete API specification in Swagger 2.0 format is available online.[6]

3) FROM OpenAPI SPECIFICATION TO WEB API

In this stage, the complete Web API is generated from its OpenAPI specification. The automatic process creates the

[6]https://wake.dlsi.ua.es/madrid/api-docs

API code represented by a server in NodeJS4F,[7] a simple and efficient runtime environment for network applications. The automatic generation process is accomplished with the help of the Swagger Codegen tool, which creates the structure of the server with the files and folders required. It also manages the calls to the API and redirects them to the corresponding method in NodeJS code. The automatic generator then completes the server with the features required to return the data requested, which are retrieved from the data source.

This generated Web API has been published in an online server, thus allowing open data reusers to query the data with the parameters desired to filter the information. The queries to the Web API are specified in the OpenAPI documentation, whose simplicity makes it possible for them to be executed by non-experts in query languages in a situational scenario. The API available online contains a method with which to query the metro stops filtered by many parameters. When a user queries this information, the list of metro stops is returned, including the stops that fulfill the specified query parameters. For instance, a query that requests the metro stops that are be accessible via a lift and escalator is: https://wake.dlsi.ua.es/madrid/metro/?hasLift=TRUE&hasEscalator=TRUE

The Web API will respond to this request with the required information in JSON format, because it is easy for humans to read and write and it is easy for machines to parse and generate5F.[8] The result obtained from the query example is the data concerning metro stops that are accessible by lift and escalator. In this case, many stations fulfill these conditions and we consequently show only the example of the "Islas Filipinas" station. The extract of the API output in JSON format, therefore, contains information regarding the stations filtered as:


```
{
    "stationName": " Islas Filipina ",
    "about":
  "https://www.metromadrid.es//es/
viaja_en_metro/red_de_metro/estaciones/
IslasFilipinas.html",
    "hasEscalator": "TRUE",
    "hasLift": "TRUE",
    "hasTravelator": "FALSE",
    "transfer": "",
"lineAbout":
  "https://www.metromadrid.es//es/
viaja_en_metro/red_de_metro/lineas_y
_horarios/linea07.html",
    "lineID": "N0d6e5c75dec24ce9a67f74
     c725894bd0",
    "lineName": "Línea 7",
    "routeService": "https://www.
     metromadrid.es/es/index.html"}
```


[7] https://nodejs.org
[8] https://www.json.org/

IV. VALIDATION

In order to validate our approach, a controlled experiment was conducted in which we compared two means of consuming open data: (i) by using the SPARQL access point of RDFs or (ii) by using our equivalent generated disposable Web API.

A. DESCRIPTION OF THE EXPERIMENT

The objective of the experiment was to study whether there were any differences as regards consuming data via the Web API vs. via SPARQL queries. In order to conduct the experiment we, therefore, planned to carry out a set of surveys (concerning Web APIs and SPARQL) with Data Science Master's Degree students at the Universidad Rey Juan Carlos (Madrid, Spain). One group of 15 users was instructed in the use of APIs, while the other group of 17 students was instructed in the use of SPARQL, such that each of the groups would answer the surveys by using the API or SPARQL, respectively. It is worth noting that all the participants had similar previous knowledge of the technologies used in the experiment, and they were also simultaneously instructed in the use of APIs and SPARQL. Each survey, i.e., that concerning the use of the Web API and the other concerning the use of SPARQL, had to be answered by the participants using the correct queries in each case.

The surveys were composed of five data requests (using a SPARQL endpoint of the corresponding Web API), i.e., the participants were given five queries and requested to find the right data. We also made a note of the time it took to answer each question, in addition to whether the query was correct and the number of attempts made until the correct query was attained. These variables helped us assess how difficult it was for the users to obtain the correct data and whether the Web API access (generated by using the approach proposed in this paper) is better than the use of the corresponding SPARQL endpoint.

The specific queries were related to the Metro Madrid transport data (i.e., the running example used throughout this paper):

1. Set of stations on a given subway line (Line 1).
2. Subway lines that pass through a certain station ("Canal").
3. Stations that have correspondence with other means of transportation.
4. Correspondence of a station ("Plaza de Castilla") with a specific means of transport ("Cercanías Renfe").
5. Determining whether a given station had an escalator, elevator or moving walkway.

All of these questions were marked as having a basic level of complexity (considered as simple), with the exception of the last (considered as complex), because, unlike the others, it was necessary to use the SPARQL FILTER command to correctly perform the query. These simple and complex queries allowed us to assess whether our approach for generating Web APIs improves SPARQL data access in any particular way.

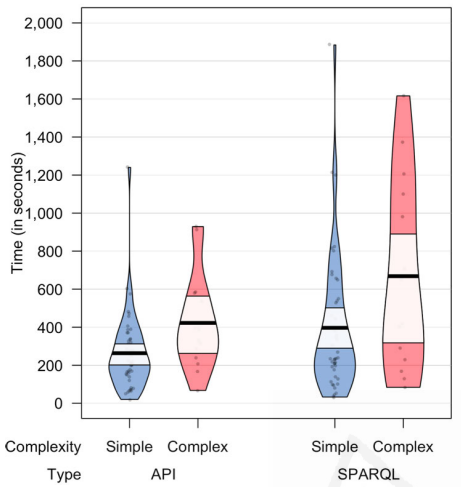


FIGURE 10. Time by type and complexity.

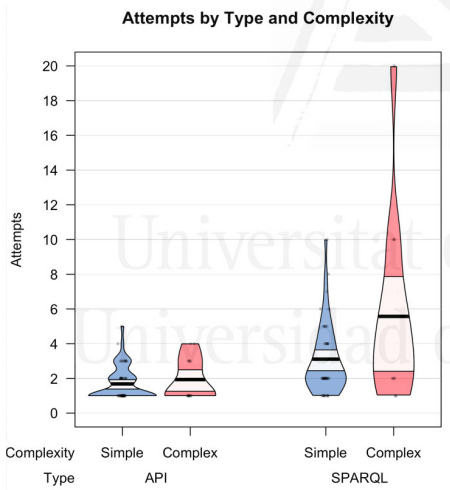


FIGURE 11. Attempts by type and complexity.

The analysis was focused on studying the time it took the participants to answer the query and the number of attempts needed to find the correct answer. We used a two-factor ANOVA with interaction to analyze the effect on the response time for each type of tool (API/SPARQL) and the complexity (simple/complex). The time variable was transformed with the double square root in order to comply with homocedasticity. The number of attempts was analyzed by using Poisson's GLM approach, taking into account the type and complexity factors.

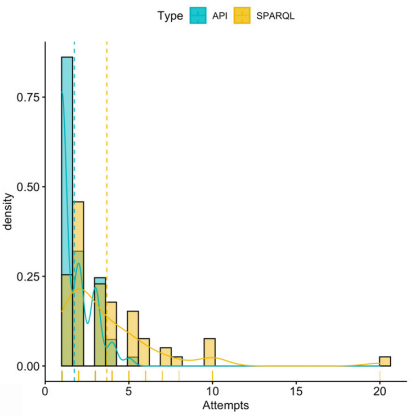


FIGURE 12. Histogram for attempts by type.

B. RESULTS

The results obtained from the analysis of the time that the participants took to respond with the correct answers have a different pattern according to whether they were attained using API or SPARQL (Fig. 10). No interaction between the two factors was detected ($F_{1,118}$, p-value = 0.8450), signifying that the pattern observed in both types is the same: more complex queries take more time, regardless of the type of approach (SPARQL or Web APIs). However, both factors separately had significant differences in their levels. Users spent more response time when using SPARQL than when using Web APIs for both simple and complex queries ($F_{1,118}$, p-value = 0.0018). The complexity was also significant: the simpler queries required less time than the complex ones ($F_{1,118}$, p-value = 0.0324).

The number of attempts was higher when using SPARQL (p-value = 1.42e-10) (Fig. 11). It was also higher for the more complex queries (p-value = 0.0001). In the case of SPARQL, there were up to 20 attempts, the average being around 4 attempts (Fig. 12). In the case of API, the average number of attempts was about 2.

It is, therefore, possible to conclude that the use of Web API attains better results than the use of SPARQL as regards both the number of attempts and the response time.

C. DISCUSSION

When employing Web APIs, the users generally responded with correct answers in less time than when employing SPARQL. If we consider the complexity of the queries, the more complex the query was, the more time they spent resolving them by using SPARQL. The response time for complex queries using SPARQL has greater variance than the other cases.

With regard to the number of attempts, the pattern is similar: the more complex the query, the longer it took, regardless

of using Web APIs or SPARQL. In addition, the number of attempts almost doubles when using SPARQL, despite the complexity of the query. This is particularly relevant in the case of situational scenarios, in which the data lifetime is very short. We do not normally know the structure of situational data, signifying that it will be more complex to manage.

The results of the experiment have allowed us to verify our previous hypothesis: that it is easier and faster in a situational scenario to use a Web API than the corresponding SPARQL endpoint in order to consume open data. Using disposable Web APIs, therefore, makes sense if the data to be consumed has a short lifespan and the objective is to solving specific one-time problems. When compared to a SPARQL endpoint, this could, therefore, be more useful in other more stable scenarios.

V. CONCLUSION

In this paper, we present an approach whose objective is to generate disposable Web APIS for open data consumption. Disposable Web APIs will likely be used to consume situational data, i.e., those data that have a narrow focus on a specific area and, often, a short lifespan in order to add value to data owned (and controlled) by a small group of consumers with a unique set of needs. This was the case in our situational scenario, in which a data scientist working at a real estate company was willing to analyze its internal data regarding housing rental and incomes in Madrid by adding open data from the Madrid public transport so as to attain new insights and make informed decisions in order to provide adapted solutions to customers with special mobility needs. Our approach first proposes a process with which to acquire structured open data. A semantic annotation process is then carried out to obtain semantically annotated data (i.e., RDF triples). Finally, this annotated data is used as a starting point for a model-driven process that automatically generates the disposable Web API for open data consumption in a situational scenario.

Our approach makes use of some semantic information, but we do not rely on SPARQL endpoints. Indeed, we have also conducted a controlled experiment to show that our Web APIs approach is more convenient for accessing data than a SPARQL endpoint.

One piece of immediate future work is that of carrying out more detailed experimentation in order to consider a wider range of situational scenarios. Our long-term future work consists of extending our disposable Web APIs generation process in order to consider open data consumers' personalization requirements.

ACKNOWLEDGMENT

The authors would like to thank to those students who agreed to participate in the experiment.

REFERENCES

[1] A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J.-N. Mazón, F. Naumann, T. Pedersen, S. B. Rizzi, J. Trujillo, P. Vassiliadis, and G. Vossen, "Fusion cubes: Towards self-service business intelligence," *Int. J. Data Warehousing Mining*, vol. 9, no. 2, pp. 66–88, Apr. 2013.

[2] B. DuCharme, *Learning SPARQL: Querying and updating with SPARQL 1.1*. Newton, MA, UAS: O'Reilly Media, 2013.

[3] D. Booth, C. G. Chute, H. Glaser, and H. Solbrig, "Toward easier RDF," in *Proc. W3C Workshop Standardization Graph Data*, Berlin, Germany, 2013, pp. 1–5.

[4] J. Klímek, P. Škoda, and M. Neăaský, "Survey of tools for linked data consumption," *Semantic Web*, vol. 10, no. 4, pp. 665–720, May 2019.

[5] P. Lisena, A. Meroáo-Peáuela, T. Kuhn, and R. Troncy, "Easy Web API development with SPARQL transformer," in *Proc. Int. Semantic Web Conf.* Cham, Switzerland: Springer, 2019, pp. 454–470.

[6] I. M. S. Hopkinson and M. Rospocher, "A simple API to the knowledge-store," in *Proc. Int. Conf. Developers*, Oct. 2014, pp. 1–5.

[7] H. Müller, L. Cabral, A. Morshed, and Y. Shu, "From RESTful to SPARQL: A case study on generating semantic sensor data," in *Proc. 6th Int. Conf. Semantic Sensor Netw.*, vol. 1063, Oct. 2013, pp. 51–66.

[8] M. Rittenbruch, M. Foth, R. Robinson, and D. Filonik, "Program your city: Designing an urban integrated open data API," in *Proc. Conf., Open Helsinki-Embedding Design Life*, 2012, pp. 24–28.

[9] A. Freitas, A. Legendre, S. O'Riain, and E. Curry, "Prov4j: A semantic Web framework for generic provenance management," in *Proc. 2nd Int. Workshop Role Semantic Provenance Manage. (SWPM*, 2010, pp. 17–22.

[10] J. Cabot. (2018). *Open Data for All: An API-Based Approach Interested Modeling Languages*. [Online]. Available: https://modeling-languages.com/open-data-for-all-api/

[11] P. Cáceres, A. Sierra-Alonso, B. Vela, J. M. Cavero, M. A. Garrido, and C. E. Cuesta, "Adding semantics to enrich public transport and accessibility data from the Web," *Open J. Web Technol.*, vol. 7, no. 1, pp. 1–18, 2020.

[12] P. Ceres, A. Sierra-Alonso, B. Vela, J. M. Cavero, and C. E. Cuesta, "Towards smart public transport data: A specific process to generate datasets containing public transport accessibility information," in *Proc. 3rd Int. Conf. Universal Accessibility Internet Things Smart Environments*, Rome, Italy, 2018, pp. 66–71.

[13] P. Cáceres, A. Sierra-Alonso, C. E. Cuesta, B. Vela, and J. M. Cavero, "Modelling and linking accessibility data in the public bus network," *J. Universal Comput. Sci.*, vol. 21, no. 6, pp. 777–795, 2015.

[14] CEN/TC 278. (2012). *Intelligent Transport Systems—Public Transport—Identification of Fixed Objects In Public Transport*. [Online]. Available: http://www.normes-donnees-tc.org/wp-content/uploads/2014/05/IFOPT-FR.pdf

[15] Transmodel. (2016). *Transmodel, Road Transport and Traffic Telematics*. [Online]. Available: http://www.transmodel.org/en/cadre1.html

[16] F. Jouault, F. Allilaire, J. Bázivin, and I. Kurtev, "ATL: A model transformation tool," *Sci. Comput. Program.*, vol. 72, pp. 1–2, May 2008, doi: 10.1016/j.scico.2007.08.002.

[17] J. S. Cuadrado, E. Guerra, and J. de Lara, "AnATLyzer: An advanced IDE for ATL model transformations," in *Proc. 40th Int. Conf. Softw. Eng., Companion Proceeedings*, May 2018, pp. 85–88, doi: 10.1145/3183440.3183479.

[18] A. Srai, F. Guerouate, N. Berbiche, and H. Drissi, "An MDA approach for the development of data warehouses from relational databases using ATL transformation language," *Int. J. Appl. Eng. Res.*, vol. 12, pp. 3532–3538, 2017.

[19] A. Meroáo-Peuela and R. Hoekstra, "GRLC makes Github taste like," in *Proc. Eur. Semantic Web Conf.*, 2016, pp. 342–353.

[20] A. Meroáo-Peáuela and R. Hoekstra, "Automatic Query-centric API for routine access to linked data," in *Proc. Int. Semantic Web Conf.*, 2017, pp. 334–349.

[21] R. Taelman, S. M. Vander, and R. Verborgh, "GraphQL-LD: Linked data Querying with GraphQL," in *Proc. 17th Int. Semantic Web Conf.*, 2018, pp. 1–4.

[22] D. Garijo and M. Osorio, "OBA: An ontology-based framework for creating REST APIs for knowledge graphs," in *Proc. Int. Semantic Web Conf.* Cham, Switzerland: Springer, Nov. 2020, pp. 48–64.

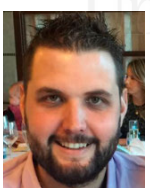
[23] A. Dimou, S. M. Vander, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle, "RML: A generic language for integrated RDF mappings of heterogeneous data," in *Proc. LDOW*, Jan. 2014, pp. 1–5.

[24] M. Lefranáois, A. Zimmermann, and N. Bakerally, "A SPARQL extension for generating RDF from heterogeneous formats," in *Proc. Eur. Semantic Web Conf.* Cham, Switzerland: Springer, May 2017, pp. 35–50.

PALOMA CÁCERES GARCÍA DE MARINA received the M.Sc. degree in computer science from the Polytechnic University of Madrid, Spain, in 1993, and the Ph.D. degree in computer science from Rey Juan Carlos University, Madrid, Spain, in 2006. She is currently an Associate Professor with Rey Juan Carlos University. Her research interests include software engineering, Web engineering, model-driven development, data science, open and linked data, the semantic Web, and transport and accessibility. She is the author or coauthor of several national and international articles related to these areas.

JOSÉ MARÍA CAVERO BARCA received the M.Sc. degree in computer science from the Polytechnic University of Madrid and the Ph.D. degree in computer science from Rey Juan Carlos University. He is currently an Associate Professor with the School of Computer Science Engineering, Rey Juan Carlos University. His research interests include databases, open data, transport, accessibility, and scientometrics.

CARLOS E. CUESTA received the Ph.D. degree in information technologies from the University of Valladolid, in 2002. He is currently an Associate Professor of software engineering with Rey Juan Carlos University, Madrid, Spain. He has had work published in premier conferences and journals, such as the *International Journal of Information Technology & Decision Making* and *Future Generation Computer Systems*. His main research interest includes software architecture, including such topics as self-adaptive systems and systems-of-systems, and with relation to concurrency theory, formal methods, distributed systems, or data engineering. He has been the Program Chair of the Sixth and 12th European Conferences on Software Architecture (ECSA 2012 & 2018).

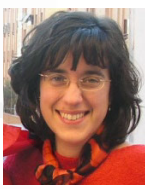


MIGUEL ÁNGEL GARRIDO received the M.Sc. degree in computer science from Rey Juan Carlos University, Madrid, Spain, in 2014, where he is currently pursuing the Ph.D. degree in computer science. For ten years, he has worked as a computer technician at several entities. He is also an Assistant Professor with Rey Juan Carlos University. His research interests include Web engineering, model driven development, semantic Web, linked open data, transport, and accessibility.

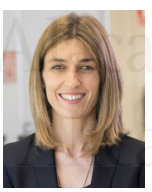
IRENE GARRIGÓS received the Ph.D. degree. She is currently an Associate Professor with the Department of Software and Computing Systems and the Head of the Web and Knowledge Research Group, University of Alicante, Spain. Her research interests include open data, Web augmentation, Web modeling languages, and personalization and application programming interfaces.

CÉSAR GONZÁLEZ-MORA is currently pursuing the Ph.D. degree with the Web and Knowledge Research Group, Department of Software, University of Alicante, Spain. His research interests include open data, Web augmentation, semantic Web, and application programming interfaces. His work is funded by a contract with the Generalitat Valenciana of Spain and the European Social Fund for predoctoral training.

JOSE-NORBERTO MAZÓN received the Ph.D. degree. He is currently an Associate Professor with the Department of Software and Computing Systems, University of Alicante, Spain. He is also the Chair of the Torrevieja Venue at the University of Alicante. He is the author of more than 100 scientific works published in international conferences and journals. His research interests include open data, business intelligence in the big data scenario, the design of data-intensive Web applications, smart cities, and smart tourism destinations.



ALMUDENA SIERRA-ALONSO received the Ph.D. degree in computer science from the Universidad Politécnica de Madrid, in 2000. She is currently an Associate Professor with the Department of Computer Science and Statistics, Rey Juan Carlos University. She is the author or coauthor of several articles related to software engineering, the semantic web, and teaching in engineering education (in areas, such as operating systems and software engineering). Her research interests include software engineering, data science, open and linked data, the semantic Web, and transport and accessibility.



BELÉN VELA received the M.Sc. degree in computer science from Carlos III University and the Ph.D. degree in computer science from Rey Juan Carlos University. She is currently an Associate Professor with the Department of Computing Science, Computer Architecture, Programming Languages and Systems and Statistics and Operative Investigation, Rey Juan Carlos University, Madrid, Spain. She also leads the Vortic3 Research Group. She has participated in and led several research projects and has had numerous articles published in prestigious journals and conferences. Her research interests include software engineering, information system engineering, data science, open data, DB (NoSQL), model driven development, transport, accessibility, and scientometrics.

JOSÉ JACOBO ZUBCOFF received the Ph.D. degree. He has a wide range of teaching and research experience in the field of statistics, and data mining and its application to biology. He has more than 100 publications in which he has dealt with obtaining knowledge from a data source. He has carried out research in various fields of science, such as computing, biology, medicine, education, and social sciences. He has additionally directed and participated in more than 20 competitive public projects financed by the Ministry of Economy and Competitiveness, the Generalitat Valenciana, the University of Alicante, and European and private projects, all of which have contributed to his knowledge of data analysis, data mining, and the attempt to democratize knowledge.

Part III

Closure

Chapter 8

Conclusions

This chapter includes the conclusions drawn from the whole research presented previously in this thesis: first, a discussion of the research carried is presented; and finally, the ongoing and future work we plan to perform are detailed.

8.1 Discussion

In this thesis, an APIfication approach is proposed in order to solve the problems related to access open data. This new process allows users to leverage from the auto-generated Web APIs, which provides access to open data not only from specific datasets, but also to integrated datasets and situational data. The APIfication approach arisen during the PhD is exposed in 3 different papers presented in Part II. These papers have been published in international journals indexed in the "Journal Citation Reports" (JCR), located within the first quartiles (two in the first quartil and the other in the fourth quartil). These publications present the contributions made by our research: an APIfication approach for open data, integrated data and situational data, with the proper experiments to analyse their suitability.

In order to evaluate the correctness and performance of the APIfication process, as explained in detail in Chapter 5, an experiment was carried out with 20 datasets of different size, generating the corresponding Web API with OpenAPI documentation and related models. The results obtained show that the automatic generation process barely takes between 9 and 80 seconds, with source files of up to more than 20 million of rows. Accordingly, the evaluation of the APIfication approach with different datasets demonstrates that the generator performs efficiently: it is able to auto-generate successfully a complete Web API for any dataset from scratch.

Moreover, as presented in Chapter 6, regarding the access to integrated data, three aspects of the system were evaluated: the word embeddings-based similarity approach for tabular data integration, with precision values over 0.92; the automatic generation of Web APIs to access integrated data, with precision of 0.76 using only the content of the cells and 0.86 precision when using also the names of the columns; and the execution time performance of the entire pipeline, taking around 10 seconds to complete the API generation procedure for tables of more than one million of rows. Consequently, the proposed similarity measure based on word embeddings achieved high precision values in the task of retrieving unionable/joinable tables given a query table. Moreover, when this measure was included in the APIfication pipeline, our approach was able to automatically and efficiently integrate the tables and generate the expected functions, which makes the whole pipeline suitable for a production scenario.

Finally, a controlled experiment was conducted, with additional details given in Chapter 7, in which we compared two means of consuming open data: (i) by using the SPARQL access point of RDFs or (ii) by using our equivalent generated disposable Web API. In order to conduct the experiment we, therefore, planned to carry out a set of surveys (concerning Web APIs and SPARQL) with Data Science Master's Degree students. The surveys were composed of five data requests (using a SPARQL endpoint or the corresponding Web API), i.e., the participants were given five queries and requested to find the right data. In this experiment, users spent more response time when using SPARQL than

when using the Web APIs. When employing Web APIs, the users generally answered with correct answers in less time than when employing SPARQL. The results of the experiment have allowed us to verify that it is easier and faster in a situational scenario to use a Web API than the corresponding SPARQL endpoint in order to consume open data. Thus, the experiment shows that the Web API generated by our APIfication approach is more convenient for accessing data than a SPARQL endpoints for average open data consumers (such as data scientists or developers).

Therefore, the contributions presented by our APIfication approach ensure the fulfilment of the objectives regarding the encouragement to include an API in the publishing process and also helping developers to reuse open data, whose suitability has been evaluated in detail through the different experiments.

8.2 Ongoing and future work

On the one hand, regarding the APIfication approach, the transformation process is going to be extended to work with different formats of open data. Also, we are exploring how to integrate directly these APIs in open data Web portals. In this sense, Web Augmentation techniques open up a new line of research which we are currently working on to facilitate the access to open data, addressing also visually impaired users, which can interact with the portal through voice interfaces. Moreover, the generated Web APIs are being improved also with a new line of research related to their OpenAPI documentation with the addition of natural language descriptions.

On the other hand, regarding the integration of open datasets, we plan to extend our approach by considering more operators in addition to union and join. Finally, a next step consists of carrying out more detailed experimentation to consider a wider range of the citizenship and the applicability of the approach under different conditions, such as in situational scenarios.

Therefore, as there is ongoing work to finish and future work to perform, the research continues in order to achieve new interesting research to be published on international journals.

Part IV

Síntesis (Spanish summary)

Appendix A

Resumen

Hoy en día, hay una tendencia a publicar datos en la web, debido a los beneficios que aporta a la sociedad y a la nueva legislación que fomenta la apertura de datos. Estos conjuntos de datos suelen ser publicados en portales de datos abiertos por gobiernos e instituciones de todo el mundo. El objetivo principal de abrir los datos es hacerlos disponibles en la web de forma gratuita y reutilizable. El comportamiento habitual suele ser que los publicadores de datos los expongan como conjuntos individuales de datos tabulares.

Los datos abiertos se consideran muy valiosos porque promueven el uso de la información pública, lo cual genera beneficios como la transparencia, innovación y otros aportes sociales, políticos y económicos. Especialmente, esta importancia es también considerable en escenarios situacionales, donde un pequeño grupo de consumidores (desarrolladores o científicos de datos) con necesidades específicas requieren datos temáticos para un ciclo de vida corto. Para que estos consumidores de datos evalúen fácilmente si los datos son adecuados para su propósito existen diferentes mecanismos. Por ejemplo, los endpoints SPARQL se han vuelto muy útiles para el consumo de datos abiertos y, en particular, de Linked Open Data (LOD). Además, para acceder a los datos abiertos de forma sencilla, las interfaces de programación de aplicaciones (APIs) web son también una característica muy recomendable de los portales de datos abiertos.

Sin embargo, acceder a estos datos abiertos es una tarea difícil, ya que las actuales plataformas de datos abiertos no suelen ofrecer estrategias adecuadas para acceder a sus datos. Por un lado, acceder a los datos abiertos a través de endpoints SPARQL es una tarea difícil porque requiere conocimientos en diferentes tecnologías, lo que supone un reto especialmente para los desarrolladores novatos. Además, los LOD no suelen estar disponibles, siendo los formatos más utilizados en los portales gubernamentales de datos abiertos los tabulares. Por otra parte, aunque la provisión de APIs web facilitaría a los desarrolladores el acceso a los datos abiertos para su reutilización, existe una falta de APIs web adecuadas en los portales de datos abiertos. Además, en la mayoría de los ca-

sos, las APIs disponibles actualmente solo permiten acceder a los metadatos del catálogo o descargar conjuntos de datos completos (es decir, acceso de grano grueso a los datos), lo que dificulta la reutilización de los datos. Además, como los datos abiertos suelen publicarse individualmente sin tener en cuenta las posibles relaciones con otros conjuntos de datos, la reutilización en grupo de varios conjuntos de datos abiertos no es una tarea trivial, por lo que se necesitan mecanismos que permitan a los consumidores de datos integrar y acceder a los datos abiertos tabulares publicados en la web. Por lo tanto, todo el potencial de los datos abiertos no se está aprovechando debido a su difícil acceso.

Dado que el acceso a los datos abiertos sigue siendo limitado para los usuarios finales, especialmente para aquellos que no tienen conocimientos de programación, proponemos un enfoque basado en modelos para generar automáticamente APIs web a partir de datos abiertos. Este enfoque, al que hemos llamado "APIficación", facilita el acceso a los datos abiertos, teniendo en cuenta el acceso a múltiples conjuntos de datos tabulares integrados y el consumo de datos en escenarios situacionales. En primer lugar, nos centramos en los datos que pueden integrarse al estar estrechamente relacionados. A continuación, acuñamos el término APIs web desechables como mecanismo alternativo para el consumo de datos abiertos en escenarios situacionales. Estas APIs web desechables se crean sobre la marcha para ser utilizadas temporalmente por un usuario para consumir datos abiertos específicos.

En consecuencia, el objetivo principal es proporcionar mecanismos adecuados para acceder y reutilizar fácilmente los datos abiertos sobre la marcha y de forma integrada, resolviendo el problema de la dificultad de acceso a través de endpoints SPARQL para la mayoría de los consumidores de datos, y la falta de APIs web adecuadas con acceso sencillo a los datos abiertos. Con este enfoque, nos dirigimos tanto a los publicadores como a los consumidores de datos abiertos, así pues, los publicadores podrán incluir una API web junto con sus datos, y los consumidores o reutilizadores de datos se verán beneficiados en aquellos casos en los que falte una API web que apunte a los datos abiertos. Los resultados de los experimentos realizados nos han llevado a concluir que los usuarios consideran que nuestras APIs web generadas automáticamente son fáciles de usar y proporcionan los datos abiertos deseados, aunque provengan de diferentes conjuntos de datos y especialmente en escenarios situacionales.

Appendix B

Introducción

En este capítulo se presenta, en primer lugar, el contexto de toda la investigación realizada durante el doctorado. Este contexto incluye la tendencia de apertura de datos por parte de los gobiernos mundiales, los beneficios que los datos abiertos aportan a la sociedad, la disponibilidad de los datos abiertos y su acceso a través de las APIs web o de tecnologías de la web semántica. Además del contexto, se presentan los problemas relacionados con el acceso a los datos abiertos. Estos problemas se detallan para señalar la falta de APIs web, la dificultad en el uso de las tecnologías de la web semántica, la baja disponibilidad de endpoints SPARQL en los portales de datos abiertos, la escasez de mecanismos para acceder a los datos integrados y el problema específico en los escenarios situacionales.

B.1 Contexto

Hoy en día, los gobiernos y las organizaciones de todo el mundo ponen sus datos en línea [5, 8]. Estos datos abiertos se publican en portales web para que sean libremente accesibles y reutilizables [11], ya que se consideran de gran valor y existe una gran concienciación en la mayoría de los países sobre los datos abiertos [36, 6].

La adopción de iniciativas de datos abiertos fomenta el uso de la información pública, lo que conlleva principalmente importantes beneficios económicos según varios estudios [43, 38, 44]. Además de estos beneficios económicos, la apertura de datos también produce transparencia, innovación y otros beneficios sociales y políticos [27]. El aumento de las iniciativas de datos abiertos también está motivado por la creciente presión impuesta por los gobiernos [42] con nuevas legislaciones [1] que obligan a las administraciones públicas a ofrecer sus datos a los ciudadanos. Los potenciales beneficiarios de los datos abiertos son los ciudadanos, a los que se les facilitan grandes cantidades de datos, pero también los reutilizadores de datos (particulares y empresas) que reutilizan los datos para crear aplicaciones útiles, fomentando la economía y beneficiando a la ciudadanía [46, 38].

Para publicar datos abiertos hay dos formas principales: (i) en portales de datos abiertos que ofrecen una plataforma web con un catálogo de datos en su mayoría de forma tabular, y (ii) como Linked Open Data (LOD) generalmente disponible a través de endpoints SPARQL. Por un lado, respecto a los portales de datos abiertos, *data.gov.uk* y *data.gov* se encuentran entre los repositorios de datos abiertos más populares, que ofrecen un catálogo de recursos de datos de los gobiernos del Reino Unido y de los Estados Unidos; y el *Portal de Datos Europeos*[1] que combina varios catálogos de datos de la Comisión Europea. Por otro lado, la nube LOD[2] incluye los conjuntos de datos que han sido publicados en el formato Linked Data (datos enlazados que están disponibles en su totalidad a través del volcado consultable "LOD-a-lot" [21]); y la plataforma DBpedia, que ofrece datos estructurados de Wikipedia mediante un endpoint SPARQL[3].

Para aprovechar las ventajas de los datos abiertos, el mayor número posible de personas debe poder acceder fácilmente a estos datos. Entre los enfoques más adoptados para acceder a los datos abiertos están las APIs web [13], ya que son una característica importante y recomendada de las plataformas de datos abiertos, permitiendo a los desarrolladores hacer que los datos abiertos sean accesibles a los ciudadanos [11] construyendo sus propias aplicaciones basadas en estos datos abiertos [30]. En cuanto a LOD, permite a los usuarios acceder del mismo modo que se utiliza un sistema de gestión de bases de datos, pero mediante lenguajes de consulta como SPARQL [17]. Estas interfaces de consulta de datos en la web implican potentes capacidades, como los endpoints SPARQL o la descarga directa de datos en formato RDF (volcados de datos).

B.2 Problemas a resolver

Sin embargo, existe una gran brecha entre los datos abiertos y su acceso por parte de la sociedad. Aunque varios estudios [40, 12] sugieren la creación de APIs web para llenar este vacío, todavía hay una falta de APIs adecuadas para acceder a los datos de las plataformas de datos abiertos en todo el mundo [22]. De hecho, solo alrededor del 6,6% de los conjuntos de datos incluyen una API con acceso directo a los datos abiertos, es decir, una API a nivel de consulta que permita acceder directamente a los datos. En la mayoría de los casos, las plataformas incluyen una API, como las APIs basadas en CKAN que siguen el estándar DCAT, que solamente proporciona los metadatos del catálogo de datos o un enlace de descarga para el conjunto de datos (acceso de grano grueso a los datos). Una API de nivel de consulta con acceso de grano fino a los datos facilita a los desarrolladores la provisión de datos específicos según las necesidades de los usuarios [11], mejorando el proceso de reutilización de los datos. Además, para promover el uso de estas APIs para acceder a los datos abiertos, un factor clave es proporcionar una documentación relevante y útil [41, 14], siguiendo estándares populares, como OpenAPI, que permiten entender y reutilizar fácilmente los datos abiertos [18]. Sin embargo, las APIs existentes no suelen incluir una documentación adecuada ni una especificación precisa de la funcionalidad y de los datos que ofrecen [28, 4]. Por lo tanto, entender cómo utilizar estas APIs y, en consecuencia, reutilizar los datos resulta una tarea compleja.

Además, la exploración de LOD mediante lenguajes de consulta estructurados como SPARQL puede ser desafiante y propensa a errores, especialmente para los desarrolladores y usuarios finales novatos [24]. Por lo tanto, los reutilizadores de datos suelen estar más familiarizados con las APIs de tipo REST que con SPARQL [13]. Los consumidores habituales de datos abiertos carecen principalmente de los conocimientos necesarios para realizar consultas SPARQL y acerca del modelo de datos RDF (Resource Description Framework) subyacente, ya que ambos tienen una pronunciada curva de aprendizaje [13].

Además, otro problema relacionado con el uso de SPARQL es que los datos en formato LOD no suelen estar disponibles en los portales de datos abiertos, representando solamente un 0,5% del total [34]. En realidad, tal y como se destaca en recientes y relevantes estudios [20, 37], estos portales se preocupan más por los formatos tabulares, siendo el formato más utilizado con una representación del 46,4%.

Por contra, la mayoría de los publicadores exponen sus datos tabulares como conjuntos de datos separados, es decir, sin tener en cuenta las posibles relaciones con otros conjuntos de datos abiertos [23]. Este escenario provoca que los reutilizadores de datos tengan que hacer un esfuerzo adicional para integrarlos. En consecuencia, existe una gran necesidad de un mecanismo que permita a los consumidores de datos integrar y acceder a los datos abiertos disponibles.

También hay escenarios concretos en los que un acceso rápido y fácil a los datos abiertos es especialmente importante. Este es el caso de los datos situa-

cionales, que consisten en datos temáticos requeridos durante un ciclo de vida corto por un grupo reducido de consumidores con necesidades específicas [3]. En consecuencia, en un escenario situacional, son necesarios mecanismos alternativos para proporcionar acceso a los datos abiertos para los reutilizadores de datos, sin requerir ningún conocimiento de las tecnologías de la web semántica [9, 29]. Así, es más fácil disponer de una API web con acceso sencillo a los datos abiertos, sin perder tiempo procesando fuentes de datos tabulares o aprendiendo a consultar un endpoint SPARQL [15].

B.3 Trabajos relacionados

Existe una gran variedad de investigaciones relacionadas con el tema de los datos abiertos, LOD y la simplificación de su acceso para los desarrolladores. En primer lugar, para resolver los problemas de acceso y reutilización de los datos, teniendo en cuenta la dificultad de acceso a los endpoints SPARQL, se propone la creación de APIs web. Sin embargo, trabajos como [40, 26, 35] proponen, en general, la creación manual de APIs para acceder a plataformas específicas de datos abiertos y enlazados, lo cual supone mucho tiempo y no es escalable. También se aborda la generación automática de APIs [39, 19], pero no se considera el acceso de grano fino a las plataformas de datos abiertos, requiriendo artefactos específicos que no están comúnmente disponibles y sin proporcionar una documentación adecuada para entender dichas APIs y promover su uso.

Además, hasta donde sabemos, no existe ninguna solución que aborde los escenarios situacionales en los que los datos abiertos temáticos se utilizan temporalmente para necesidades específicas. Las investigaciones [45, 32] que abordan la exploración de Linked Open Data facilitan su acceso a través de diferentes interfaces, pero generalmente requieren conocimientos en tecnologías RDF o SPARQL, o solo proporcionan acceso a endpoints específicos lo cual también requiere de mucho tiempo y persiste el problema de escalabilidad.

En cuanto a la integración de datos, aunque los retos de la integración de datos se han investigado durante años con importantes avances [25], los esfuerzos se han centrado únicamente en resolver problemas específicos. Sin embargo, según Abadi et al. [2] se necesita más trabajo para investigar cómo canalizar la integración de datos para cubrir todo el camino desde los propios datos hasta el resultado deseado por el usuario final. Por ejemplo, se necesitan metadatos adicionales para que los desarrolladores sepan cómo se pueden relacionar los conjuntos de datos entre sí. Aunque varios enfoques recientes han propuesto añadir este tipo de metadatos a los conjuntos de datos tabulares mediante anotaciones, esta metodología no ha sido ampliamente adoptada por los publicadores de datos.

En consecuencia, aunque los trabajos relacionados ayudan a los reutilizadores de datos a superar los diferentes problemas para acceder a los datos abiertos, no consideramos que ofrezcan una solución flexible y completa que realmente facilite a los desarrolladores la obtención de datos abiertos de diferentes fuentes para mejorar el proceso de reutilización de dichos datos.

Appendix C

Propuestas

Una de las prácticas más adoptadas para facilitar el acceso a los datos abiertos es el despliegue de APIs web sobre portales de datos abiertos y fuentes de Linked Data [31]. En este sentido, esta tesis está enfocada en garantizar el acceso a los datos abiertos a nivel de consulta (es decir, el acceso de grano fino), incluyendo el acceso a datos integrados y en escenarios situacionales. Nuestra propuesta se centra, por tanto, en facilitar el acceso a los datos abiertos considerando también un proceso de integración de datos junto con la generación de APIs para simplificar el consumo de datos abiertos de diferentes conjuntos de datos en escenarios particulares. Las APIs web que proponemos se crean sobre la marcha para ser utilizadas temporalmente por los usuarios para consumir datos abiertos como datos situacionales. Estas APIs han sido denominadas como APIs web desechables porque se crean sobre la marcha para permitir a los consumidores de datos evaluar si los datos son adecuados para su propósito, evitando así la complejidad y la curva de aprendizaje de SPARQL y el esfuerzo de procesar manualmente los datos.

El objetivo principal es doble: (i) animar a los publicadores de datos abiertos a incluir una API junto con los datos que se van a abrir y (ii) ayudar a los desarrolladores a crear valor a partir de los datos abiertos, es decir, facilitar la reutilización de dichos datos y promover así su acceso por los ciudadanos. Para cumplir estos objetivos, vamos a explicar las diferentes propuestas por separado: primero se introduce el enfoque de APIficación a partir de datos abiertos tabulares; después, se amplía este enfoque de APIficación con la propuesta de integrar los datos abiertos y acceder a ellos de forma conjunta; y, por último, se aplica al enfoque de APIficación la propuesta de aprovechar la información semántica para generar APIs web desechables en escenarios situacionales.

C.1 APIficación

En primer lugar, proponemos un enfoque de APIficación basado en modelos para lograr la generación automática de APIs web a nivel de consulta con acceso de grano fino a los datos. Estas APIs web ayudan a los desarrolladores a acceder y reutilizar los datos abiertos, promoviendo así el uso de los dichos datos por parte de la ciudadanía.

Este proceso de APIficación [47] consiste en un conjunto de transformaciones para autogenerar APIs web con acceso a los conjuntos de datos abiertos. El proceso se basa en mecanismos automáticos, genéricos y estandarizados para generar APIs web con documentación procesable por ordenador siguiendo el estándar de código abierto más popular: OpenAPI 3.0. Seguir este estándar ayuda a los usuarios a entender el funcionamiento de la API y a trabajar con ella como interfaz web. Además, los mecanismos dirigidos por modelos [10] permiten hacer frente a la heterogeneidad de las fuentes de datos abiertas existentes, integrando la API y su documentación en procesos de desarrollo dirigido por modelos para estandarizar la forma de crear y definir las APIs.

El proceso de transformación para la generación automática de APIs web comienza con una fuente de datos abiertos. Estas APIs web ayudan a los usuarios a acceder y reutilizar unos datos iniciales seleccionados de una fuente de datos. Además, el proceso también crea la documentación OpenAPI para ayudar a los usuarios de la API web, y un modelo de los datos y de la documentación con el fin de aprovechar el gran número de herramientas de modelado existentes para la integración de estos artefactos en los procesos de desarrollo dirigidos por modelos. Todo el proceso de transformación de la fuente de datos a la API web - que se muestra en la figura 4.1 - se pone en marcha mediante el programa generador automático, incluyendo los siguientes pasos: una transformación de texto a modelo (T2M) desde la fuente de datos al modelo de datos, una transformación de modelo a modelo (M2M) desde el modelo de datos al modelo OpenAPI, una transformación de modelo a texto (M2T) desde el modelo OpenAPI a su documentación OpenAPI, y finalmente una transformación de texto a texto (T2T) desde la documentación OpenAPI a la API web. Una vez finalizado el proceso, los usuarios pueden consultar los datos a través de la API web generada, de forma que la API accede a los datos desde la fuente y los devuelve a los usuarios. Por tanto, partiendo de un conjunto de datos que contiene filas y celdas, el sistema realiza una transformación directa para construir un objeto de modelo de datos con objetos de fila y celda. A continuación, el modelo de datos se transforma automáticamente en un modelo OpenAPI utilizando cada celda de la primera fila como un componente de la API (método, parámetro y propiedad), convirtiendo este objeto del modelo OpenAPI en un formato estándar para la documentación de la API. Por último, la API web completa se crea automáticamente incluyendo los métodos, parámetros y propiedades detallados en su documentación.

Con este enfoque, los usuarios son capaces de obtener una API web de

cualquier fuente de datos abierta para acceder fácilmente al conjunto de datos con la documentación OpenAPI adecuada. Se presentan más detalles en el capítulo 5.

C.2 Integración

Además del enfoque de APIficación presentado anteriormente, cabe destacar que puede haber un paso previo en este proceso de APIficación en caso de que sea necesario integrar datos. Este nuevo paso consiste en integrar datos similares para que la API web generada sea capaz de acceder a los datos abiertos de forma integrada.

El proceso que considera ambos pasos (integración y APIficación) se muestra en la figura 4.2. El primer paso implica la detección de conjuntos de datos tabulares unibles utilizando técnicas de "word embeddings" [33] para identificar qué columnas de diferentes conjuntos de datos tabulares de entrada tienen más probabilidades de integrarse. El siguiente paso consiste en integrar los conjuntos de datos aplicando operaciones de "union" o "join". Finalmente, a partir del conjunto de datos previamente integrado, se genera una API web para acceder a estos datos mediante las transformaciones de APIficación anteriormente explicadas.

Los operadores considerados para la integración de los datos abiertos son "union" y "join" del álgebra relacional que funcionan de la siguiente forma:

- El operador "union" tiene como objetivo obtener un único conjunto de datos a partir de dos conjuntos de datos tabulares (A y B), conteniendo filas que estén en A o en B. Para ello, A y B deben compartir columnas que se refieran al mismo concepto.
- El operador "join" también tiene como objetivo obtener un único conjunto de datos a partir de dos conjuntos de datos tabulares (A y B), pero que incluya todas las columnas de A y B y que contenga filas que cumplan una condición.

La operación se elige entre "union" y "join" en función de la similitud encontrada entre los conjuntos de datos a integrar. Por ejemplo, si dos conjuntos de datos son muy similares solo en un subconjunto de columnas, es probable que se integren mediante una operación "join". Sin embargo, si dos conjuntos de datos son similares en casi todas las columnas, es más probable que se integren mediante una operación de unión. En el capítulo 6 se presentan más detalles.

C.3 Enfoque semántico

Por último, en el enfoque de APIficación se puede añadir también un proceso previo para aprovechar la información semántica de las fuentes de datos. El objetivo principal es el de generar automáticamente APIs web desechables y fáciles de usar, las cuales pueden utilizarse para acceder a datos abiertos en escenarios situacionales.

El proceso propuesto para el consumo de datos abiertos mediante el uso de APIs web desechables generadas automáticamente comprende tres etapas:

1. Proceso de entrada de datos: en esta etapa se seleccionan los datos web para satisfacer las necesidades de los consumidores de datos en un escenario situacional.

2. Proceso de anotación semántica: los datos de entrada se anotan utilizando una ontología específica del dominio, obteniendo datos anotados semánticamente.

3. Proceso de generación de APIs web desechables: la entrada de este proceso son los datos tabulares previamente anotados semánticamente, a partir de los cuales se generan automáticamente APIs web desechables para permitir el consumo de datos en un escenario situacional.

El proceso completo de consumo de datos abiertos se muestra en la figura 4.3, incluyendo las entradas y salidas de cada proceso. En cuanto al enfoque de APIficación, se añade el nuevo paso de anotar previamente los datos para mejorar la calidad de los datos disponibles. Este paso comienza por estudiar la información contenida en la fuente de datos y las características importantes del dominio seleccionado. A continuación, es necesario crear un glosario de términos relativos a la terminología utilizada en la fuente de datos, describiendo cada uno de ellos, tras lo cual se debe definir un modelo de dominio con el que representar estos términos y sus relaciones mediante un diagrama de clases UML. Después, es importante identificar la semántica de la información para alinearla con los términos de un vocabulario de referencia, es decir, un vocabulario que el desarrollador debe elegir (por ejemplo, una ontología específica del dominio). Los mapeos necesarios para establecer la correspondencia entre los distintos elementos de información de la fuente de datos (glosario de términos) y el vocabulario de referencia (ontología específica del dominio) se definen mediante una inspección manual de la fuente de datos de entrada. En concreto, identificamos los sujetos, predicados y objetos de un archivo de entrada tabular y los asignamos a elementos del vocabulario de referencia. Este mapeo se define manualmente y se añade a un archivo de configuración. Una vez definidos todos los mapeos, se programa un script personalizado para transformar los datos tabulares de entrada en los datos anotados semánticamente, de acuerdo con los mapeos definidos en el archivo de configuración. A continuación, si la alineación es posible, es necesario utilizar los datos originales, y anotarlos semánticamente

utilizando los términos existentes del vocabulario de referencia (ontología específica del dominio) mediante el lenguaje RDF, utilizando los mapeos previamente definidos. Si la alineación no es posible, hay que analizar la diferencia entre los datos originales y el vocabulario de referencia para ampliarlo. Finalmente, el último paso consiste en integrar el conjunto de datos semánticos con otras fuentes. Esto se hace relacionando elementos de los distintos conjuntos de datos semánticos y descartando la información duplicada en los conjuntos de datos, cuando sea necesario.

En el capítulo 7 se presentan más detalles.

C.4 Contribuciones

Para reducir la brecha entre los datos abiertos y su acceso por parte de los usuarios, hemos propuesto un enfoque de APIficación con los mecanismos adecuados para considerar los datos abiertos, la integración de datos y los datos situacionales.

Entre las aportaciones de este enfoque de APIficación se encuentran:

- La creación de un proceso genérico, automático y basado en modelos para la generación de APIs web. El enfoque pretende así simplificar directamente el proceso de reutilización de datos abiertos, lo que aportará beneficios, no solo económicos, para los desarrolladores y el sector infomediario.
- La implementación de este proceso automático de generación de APIs web, que está disponible en línea en GitHub[6].
- La posibilidad de que los desarrolladores creen fácilmente APIs web por su cuenta, liberando a éstos de la gestión sobre cómo proporcionar datos a la ciudadanía, como por ejemplo creando aplicaciones móviles que accedan a los datos abiertos a través de las APIs generadas automáticamente. Aunque tengan que publicar la API en un servidor, tienen el control tanto de los datos como de su acceso, lo que también tiene sus ventajas. Por ejemplo, estas API pueden ser fácilmente personalizadas por los desarrolladores, si es necesario, para mejorar la experiencia de obtención de datos.
- La puesta a disposición de la documentación de las APIs web siguiendo estándares populares (OpenAPI), lo que facilita la comprensión de estas APIs y, en consecuencia, la promoción de su uso.
- La definición de una medida de similitud entre conjuntos de datos tabulares basada en "word embeddings". Esta medida permite identificar los datos abiertos tabulares unibles con el fin de facilitar el proceso de integración de los datos a los que se accede de forma conjunta.
- La implementación de los procesos de integración y generación de la API web de forma sucesiva, que está disponible en línea en GitHub[7].
- Un proceso para la anotación semántica de fuentes de datos tabulares de la web.
- Un enfoque basado en modelos que se emplea para incorporar nuestro proceso de anotación semántica y luego generar APIs web desechables para acceder a las fuentes de datos.
- La evaluación de nuestro enfoque con desarrolladores y datos abiertos tabulares reales, generando las APIs web adecuadas para acceder a datos específicos, comparando su uso con el acceso mediante tecnologías de la web semántica como SPARQL.

Appendix D

Trabajos publicados

En este capítulo se presenta el título y resumen breve de los trabajos que se han publicado a lo largo de todo el programa de doctorado (el contenido completo de las publicaciones se muestra en la parte II de esta tesis):

- Generación de interfaces de programación de aplicaciones web basada en modelos para acceder a datos abiertos ("Model-based Generation of Web Application Programming Interfaces to Access Open Data" [22]).
- Desarrollo de APIs web basado en modelos para acceder de forma integrada a datos abiertos tabulares ("Model-Driven Development of Web APIs to Access Integrated Tabular Open Data" [23]).
- Consumo de datos abiertos mediante la generación de APIs web desechables ("Open Data Consumption Through the Generation of Disposable Web APIs" [15]).

D.1 Generación de interfaces de programación de aplicaciones web basada en modelos para acceder a datos abiertos

Para facilitar la reutilización de los datos abiertos de los catálogos de las plataformas de datos abiertos, las interfaces de programación de aplicaciones (APIs) web son un mecanismo importante para los reutilizadores. Sin embargo, faltan APIs web adecuadas para acceder a los datos de las plataformas de datos abiertos. Además, en la mayoría de los casos, las APIs disponibles actualmente solo permiten acceder a los metadatos del catálogo o descargar recursos de datos completos (es decir, acceso de grano grueso a los datos), lo que dificulta la reutilización de los datos. Por ello, proponemos un enfoque basado en modelos para generar automáticamente APIs web a partir de datos abiertos. Nuestras APIs web generadas facilitan el acceso y la reutilización de datos específicos (es decir, proporcionan un acceso de grano fino o de nivel de consulta a los datos), lo que dará lugar a importantes beneficios sociales y económicos, como la transparencia y la innovación. Con este enfoque nos dirigimos a los publicadores de datos abiertos que podrán incluir una API web junto con sus datos, pero también a los reutilizadores de datos abiertos en caso de que falten dichas APIs. Este proceso de APIficación, que supone la creación de APIs para cada conjunto de datos disponible, se basa en mecanismos de generación automáticos, genéricos y estandarizados. El rendimiento y funcionamiento de este enfoque se ha validado con diferentes conjuntos de datos, generando con éxito APIs web que facilitan la reutilización de los datos.

D.2 Desarrollo de APIs web basado en modelos para acceder de forma integrada a datos abiertos tabulares

Cada vez son más los gobiernos de todo el mundo que publican datos abiertos, principalmente en formatos tabulares como CSV o XLS(X). Estos conjuntos de datos se publican en su mayoría de forma individual, es decir, cada publicador expone sus datos en la web sin tener en cuenta las posibles relaciones con otros conjuntos de datos (propios o de terceros). En consecuencia, la reutilización en grupo de varios conjuntos de datos abiertos no es una tarea trivial, por lo que se requieren mecanismos que permitan a los consumidores de datos (como desarrolladores de software o científicos de datos) integrar y acceder a los datos abiertos tabulares publicados en la web. En este trabajo, proponemos un enfoque basado en modelos para generar automáticamente APIs web que accedan de forma homogénea a múltiples conjuntos de datos abiertos tabulares e integrados. Este trabajo se centra en los datos que pueden integrarse mediante operaciones de "union" y "join". Como primer paso, nuestro enfoque detecta los datos abiertos tabulares que se pueden unir mediante una medida de similitud de tablas basada en la técnica "word embeddings". A continuación, se desarrolla un proceso de APIficación para crear APIs que accedan a los conjuntos de datos previamente integrados a través de un único punto de acceso. A lo largo del artículo se presenta un ejemplo de funcionamiento, así como un conjunto de experimentos para la evaluación del rendimiento que demuestran la viabilidad de nuestro enfoque.

D.3 Consumo de datos abiertos mediante la generación de APIs web desechables

La creciente cantidad de información en el mundo actual ha llevado a la publicación de cada vez más datos abiertos, que son aquellos que están disponibles de forma gratuita y reutilizable en la web. Los datos abiertos se consideran muy valiosos en escenarios situacionales, en los que los datos temáticos son requeridos para un ciclo de vida corto por un pequeño grupo de consumidores con necesidades específicas. En este contexto, los consumidores de datos (desarrolladores o científicos de datos) necesitan mecanismos con los que evaluar fácilmente si los datos son adecuados para su propósito. Los endpoints SPARQL resultan muy útiles para el consumo de datos abiertos, pero argumentamos que su pronunciada curva de aprendizaje dificulta la reutilización de los datos abiertos en escenarios situacionales. Para superar este problema, en este artículo acuñamos el término APIs web desechables como mecanismo alternativo para el consumo de datos abiertos en escenarios situacionales. Las APIs web desechables se crean sobre la marcha para que un usuario las utilice temporalmente para consumir datos abiertos. En este trabajo describimos específicamente un enfoque con el que aprovechar la información semántica de las fuentes de datos para generar automáticamente APIs web desechables fáciles de usar que permitan acceder a datos abiertos en un escenario situacional, evitando así la complejidad y la curva de aprendizaje de SPARQL y el esfuerzo de procesar manualmente los datos. Hemos llevado a cabo varios experimentos para descubrir si a los usuarios no experimentados les resulta más fácil utilizar nuestra API web desechable o un endpoint SPARQL para acceder a los datos abiertos. Los resultados de los experimentos nos han llevado a concluir que, en un escenario situacional, es más fácil y rápido utilizar la API web que el correspondiente endpoint SPARQL para consumir datos abiertos.

Appendix E

Conclusiones

En este capítulo se recogen las conclusiones extraídas de toda la investigación presentada anteriormente en esta tesis: en primer lugar, una discusión sobre la investigación y la tesis; y por último, el trabajo en curso y futuro que tenemos previsto realizar de aquí en adelante.

E.1 Discusión

En esta tesis se propone un enfoque de APIficación para resolver los problemas relacionados con el acceso a datos abiertos. Este nuevo proceso permite a los usuarios aprovechar las APIs web autogeneradas, lo que proporciona acceso a los datos abiertos no solamente a conjuntos de datos específicos, sino también a conjuntos de datos integrados y datos situacionales. El enfoque de la APIficación surgido durante el doctorado se expone en 3 artículos diferentes presentados en la Parte II. Estos trabajos han sido publicados en revistas internacionales indexadas en el "Journal Citation Reports" (JCR), situadas dentro de los primeros cuartiles (dos en el primer cuartil y el otro en el cuarto cuartil). Estas publicaciones presentan las aportaciones de nuestra investigación: un enfoque de APIficación para datos abiertos, datos integrados y datos situacionales, con los experimentos adecuados para analizar su idoneidad.

Para evaluar la corrección y el rendimiento del proceso de APIficación, como se presenta en el capítulo 5, se realizó un experimento con 20 conjuntos de datos de diferente tamaño, generando la correspondiente API web con la documentación OpenAPI y los modelos relacionados. Los resultados obtenidos muestran que el proceso de generación automática apenas tarda entre 9 y 80 segundos, con archivos fuente de hasta más de 20 millones de filas. En consecuencia, la evaluación del enfoque de APIficación con diferentes conjuntos de datos demuestra que el generador se comporta de forma eficiente: es capaz de generar automáticamente y con éxito una API web completa para cualquier conjunto de datos desde cero.

Además, en lo que respecta al acceso a los datos integrados y tal y como hemos visto en el capítulo 6, se evaluaron tres aspectos del sistema: el enfoque de similitud basado en "word embeddings" para la integración de datos tabulares, con valores de precisión superiores a 0,92; la generación automática de APIs web para acceder a los datos integrados, con una precisión de 0,76 utilizando solo el contenido de las celdas y de 0,86 cuando se utilizan también los nombres de las columnas; y el rendimiento del tiempo de ejecución de todo el proceso, que tarda unos 10 segundos en completar la generación de la API para tablas de más de un millón de filas. En consecuencia, la medida de similitud propuesta basada en "word embeddings" alcanzó altos valores de precisión en la tarea de recuperación de tablas unibles dada una tabla de consulta. Además, cuando esta medida se incluyó en el proceso de APIficación, nuestro enfoque fue capaz de integrar automática y eficientemente las tablas y generar las funciones esperadas, lo que hace que todo el proceso sea adecuado para un escenario de producción.

Por último, se llevó a cabo un experimento (presentado en detalle en el capítulo 7) en el que comparamos dos formas de consumir datos abiertos (i) utilizando el punto de acceso SPARQL o (ii) utilizando nuestra API web desechable generada de forma automática. Para llevar a cabo el experimento, planeamos realizar un conjunto de encuestas (relativas a las APIs web y a SPARQL) con estudiantes de un Máster de Ciencia de los Datos. Las encuestas se componían

de cinco peticiones de datos (utilizando un endpoint SPARQL o la correspondiente API web), es decir, se les daba a los participantes cinco consultas y se les pedía que encontraran los datos adecuados. En este experimento, los usuarios emplearon más tiempo de respuesta cuando utilizaron SPARQL que cuando utilizaron las APIs web. Al emplear las APIs, los usuarios respondieron generalmente con respuestas correctas en menos tiempo que al emplear SPARQL. Los resultados del experimento nos han permitido comprobar que es más fácil y rápido en un escenario situacional utilizar una API web que un endpoint SPARQL para consumir datos abiertos. Así, el experimento muestra que la API web generada por nuestro enfoque de APIficación es más conveniente para acceder a los datos que un endpoint SPARQL para los consumidores habituales de datos abiertos.

Por lo tanto, las contribuciones presentadas por nuestro enfoque de APIficación garantizan el cumplimiento de los objetivos en cuanto al fomento de la inclusión de una API en el proceso de publicación y también ayudando a los desarrolladores a reutilizar los datos abiertos, cuya idoneidad ha sido evaluada en detalle a través de los diferentes experimentos.

E.2 Trabajo en curso y futuro

Por un lado, en cuanto al enfoque de la APIficación, se va a ampliar el proceso de transformación para trabajar con diferentes formatos de datos abiertos. Además, estamos explorando cómo integrar directamente estas APIs en portales web de datos abiertos. En este sentido, las técnicas de aumento de la web ("Web Augmentation") abren una nueva línea de investigación en la que estamos trabajando actualmente para facilitar el acceso a los datos abiertos, dirigiéndose también a los usuarios con discapacidad visual, que pueden interactuar con el portal a través de interfaces de voz que proponemos. Además, las APIs web generadas se están mejorando también con una nueva línea de investigación relacionada con su documentación OpenAPI, incorporando descripciones en lenguaje natural.

Por otro lado, en lo que respecta a la integración de conjuntos de datos abiertos, tenemos previsto ampliar nuestro enfoque considerando más operadores además de "union" y "join". Por último, un próximo paso consiste en llevar a cabo una experimentación más detallada con el fin de considerar una gama más amplia de la ciudadanía y la aplicabilidad del enfoque en diferentes escenarios como los escenarios situacionales.

Por lo tanto, como hay trabajo en curso para terminar y trabajo futuro para realizar, la investigación continúa con el fin de lograr nuevas investigaciones interesantes para ser publicadas en revistas internacionales.

Part V

References

Bibliography

[1] State official newsletter of Spain (BOE). Law 19/2013 of December 9, transparency, access to public information and good governance. BOE number 295 of 10-12, 2013.

[2] Daniel Abadi, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, Philip A. Bernstein, Peter A. Boncz, Surajit Chaudhuri, Alvin Cheung, AnHai Doan, Luna Dong, Michael J. Franklin, Juliana Freire, Alon Y. Halevy, Joseph M. Hellerstein, Stratos Idreos, Donald Kossmann, Tim Kraska, Sailesh Krishnamurthy, Volker Markl, Sergey Melnik, Tova Milo, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Jignesh Patel, Andrew Pavlo, Raluca A. Popa, Raghu Ramakrishnan, Christopher Ré, Michael Stonebraker, and Dan Suciu. The seattle report on database research. *SIGMOD Rec.*, 48(4):44–53, 2019.

[3] Alberto Abelló, Jérôme Darmont, Lorena Etcheverry, Matteo Golfarelli, Jose-Norberto Mazón, Felix Naumann, Torben Pedersen, Stefano Bach Rizzi, Juan Trujillo, Panos Vassiliadis, et al. Fusion cubes: Towards self-service business intelligence. *International Journal of Data Warehousing and Mining (IJDWM)*, 9(2):66–88, 2013.

[4] Alberto Abelló Gamazo, Claudia Patricia Ayala Martínez, Carles Farré Tost, Cristina Gómez Seoane, Marc Oriol Hilari, and Óscar Romero Moral. A Data-driven approach to improve the process of data-intensive API creation and evolution. In *Proceedings of the Forum and Doctoral Consortium Papers Presented at the 29th CAiSE*, pages 1–8, 2017.

[5] Mohammed Saleh Altayar. Motivations for open data adoption: An institutional theory perspective. *Government Information Quarterly*, 35(4):633–643, 2018.

[6] Lameck M Amugongo, Hippolyte N Muyingi, and Jürgen Sieck. Increasing open data awareness and consumption in namibia: A hackathon approach. In *13th Culture and Computer Science-Cross Media conference*, pages 187–198, 2015.

[7] Marcelo Arenas, Francisco Maturana, Cristian Riveros, and Domagoj Vrgoč. A framework for annotating csv-like data. *Proceedings of the VLDB Endowment*, 9(11):876–887, 2016.

[8] Judie Attard, Fabrizio Orlandi, Simon Scerri, and Sören Auer. A systematic review of open government data initiatives. *Government Information Quarterly*, 32(4):399 – 418, 2015.

[9] David Booth, CG Chute, H Glaser, and H Solbrig. Toward easier rdf. In *W3C Workshop on Web Standardization for Graph Data. Berlin, Germany*, 2019.

[10] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

[11] K. Braunschweig, J. Eberius, M. Thiele, and W. Lehner. The State of Open Data - Limits of Current Open Data Platforms. In *Proceedings of the 21st World Wide Web Conference*, 2012.

[12] J. Cabot. Open data for all: an API-based approach, 2016. `https://modeling-languages.com/open-data-for-all-api/`. Accessed July 31, 2019.

[13] Enrico Daga, Luca Panziera, and Carlos Pedrinaci. A BASILar approach for building web APIs on top of SPARQL endpoints. In *CEUR Workshop Proceedings*, volume 1359, pages 22–32, 2015.

[14] P. J. Danielsen and A. Jeffrey. Validation and Interactivity of Web API Documentation. In *IEEE 20th International Conference on Web Services*, pages 523–530, 2013.

[15] Paloma Cáceres García De Marina, José María Cavero Barca, Carlos E. Cuesta, Miguel Ángel Garrido, Irene Garrigós, César González-Mora, Jose-Norberto Mazón, Almudena Sierra-Alonso, Belén Vela, and José Jacobo Zubcoff. Open Data Consumption Through the Generation of Disposable Web APIs. *IEEE Access*, 9:76354–76363, 2021.

[16] Yasushi Doi and Motomichi Toyama. Tot for csv: accessing open data csv files through sql. In *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services*, pages 423–429, 2019.

[17] Bob DuCharme. *Learning SPARQL: querying and updating with SPARQL 1.1.* O'Reilly Media, Inc., 2013.

[18] H. Ed-douibi, J. L. Cánovas, and J. Cabot. Example-Driven Web API Specification Discovery. In *Modelling Foundations and Applications - Springer International Publishing*, pages 267–284, 2017.

[19] Hamza Ed-douibi, Javier Luis Cánovas Izquierdo, Abel Gómez, Massimo Tisi, and Jordi Cabot. EMF-REST: Generation of RESTful APIs from Models. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, page 1446–1453. Association for Computing Machinery, 2016.

[20] European Commission. Open data maturity report 2019. *Publications Office of the European Union*, 2019.

[21] Javier D. Fernández, Wouter Beek, Miguel A. Martínez-Prieto, and Mario Arias. Lod-a-lot. In *The Semantic Web - ISWC 2017*, pages 75–83. Springer International Publishing, 2017.

[22] César González Mora, Irene Garrigós, Jose Zubcoff, and Jose-Norberto Mazón. Model-based Generation of Web Application Programming Interfaces to Access Open Data. *Journal of Web Engineering*, pages 194–217, 2020.

[23] César González-Mora, David Tomás, Irene Garrigós, José Jacobo Zubcoff, and Jose-Norberto Mazón. Model-Driven Development of Web APIs to Access Integrated Tabular Open Data. *IEEE Access*, 8:202669–202686, 2020.

[24] Pavel Grafkin, Mikhail Mironov, Michael Fellmann, Birger Lantow, Kurt Sandkuhl, and Alexander V. Smirnov. SPARQL Query Builders: Overview and Comparison. In *BIR Workshops*, 2016.

[25] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: The teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, pages 9–16, 2006.

[26] I. Hopkinson, S. Maude, and M. Rospocher. A Simple API to the Knowledgestore. In *Proceedings of the 2014 International Conference on Developers - Volume 1268*, pages 7–12, 2014.

[27] M. Janssen, Y. Charalabidis, and A. Zuiderwijk. Benefits, Adoption Barriers and Myths of Open Data and Open Government. *Information Systems Management*, 29(4):258–268, 2012.

[28] Rediana Koçi, Xavier Franch, Petar Jovanovic, and Alberto Abelló. A data-driven approach to measure the usability of web apis. In *2020 46th Euromicro Conference on SEAA*, pages 64–71, 2020.

[29] Pasquale Lisena, Albert Meroño-Peñuela, Tobias Kuhn, and Raphaël Troncy. Easy web api development with sparql transformer. In *International Semantic Web Conference*, pages 454–470. Springer, 2019.

[30] Maria Maleshkova, Lukas Zilka, Petr Knoth, and Carlos Pedrinaci. Cross-lingual web API classification and annotation. In *Proceedings of the 2nd International Conference on Multilingual Semantic Web-Volume 775*, pages 1–12. CEUR-WS. org, 2011.

[31] Albert Meroño-Peñuela and Rinke Hoekstra. grlc makes GitHub taste like linked data APIs. In *European Semantic Web Conference*, pages 342–353. Springer, 2016.

[32] Albert Meroño-Peñuela and Rinke Hoekstra. Automatic query-centric API for routine access to Linked Data. In *International Semantic Web Conference*, pages 334–349. Springer, 2017.

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc.

[34] Sebastian Neumaier, Jürgen Umbrich, and Axel Polleres. Automated quality assessment of metadata across open data portals. *J. Data and Information Quality*, 8(1):2:1–2:29, 2016.

[35] Andreas Nolle, German Nemirovski, Alvaro Sicilia, and J Pleguezuelos. An approach for accessing linked open data for data mining purposes. In *Proceedings of RCOMM*, 2013.

[36] Great Britain. Cabinet Office. *Open data white paper: unleashing the potential*, volume 8353. The Stationery Office, 2012.

[37] Organisation for Economic Co-operation and Development, OECD. Open government data report: Enhancing policy maturity for sustainable impact. *OECD Digital Government Studies*, 2018.

[38] Aporta Project. Characterization study of the infomediary sector in Spain, 2012. `https://www.ontsi.red.es/ontsi/sites/ontsi/files/121001_red_007_final_report_2012_edition__vf_en_1.pdf`.

[39] Ricardo Queirós. Kaang: A RESTful API Generator for the Modern Web. In *7th Symposium on Languages, Applications and Technologies*, volume 62 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:15, 2018.

[40] M. Rittenbruch, M. Foth, R. Robinson, and D. Filonik. Program your city: designing an urban integrated open data API. In *Proceedings of Cumulus Helsinki 2012 Conference: Open Helsinki-Embedding Design in Life*, pages 24–28, 2012.

[41] Martin P. Robillard and Robert DeLine. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011.

[42] U. Sivarajah, V. Weerakkody, P. Waller, H. Lee, Z. Irani, Y. Choi, R. Morgan, and Y. Glikman. The role of e-participation and open data in evidence-based policy decision making in local government. *Journal of Organizational Computing and Electronic Commerce*, 26(1-2):64–79, 2016.

[43] A. Stott. Open data for economic growth. *Washington DC: World Bank*, 2014.

[44] B. Ubaldi. Open government data: Towards empirical analysis of open government data initiatives. *OECD Working Papers on Public Governance*, 22, 2013.

[45] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. *Journal of Web Semantics*, 37:184–206, 2016.

[46] V. Weerakkody, Z. Irani, K. Kapoor, U. Sivarajah, and Y. K. Dwivedi. Open data and its usability: an empirical view from the Citizen's perspective. *Information Systems Frontiers*, 19(2):285–300, 2017.

[47] J. Wettinger, U. Breitenbücher, and F. Leymann. ANY2API - Automated APIfication - Generating APIs for Executables to Ease their Integration and Orchestration for Cloud Application Deployment Automation. In *Proceedings of the 5th International Conference on Cloud Computing and Services Science*, pages 475–486, 2015.