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

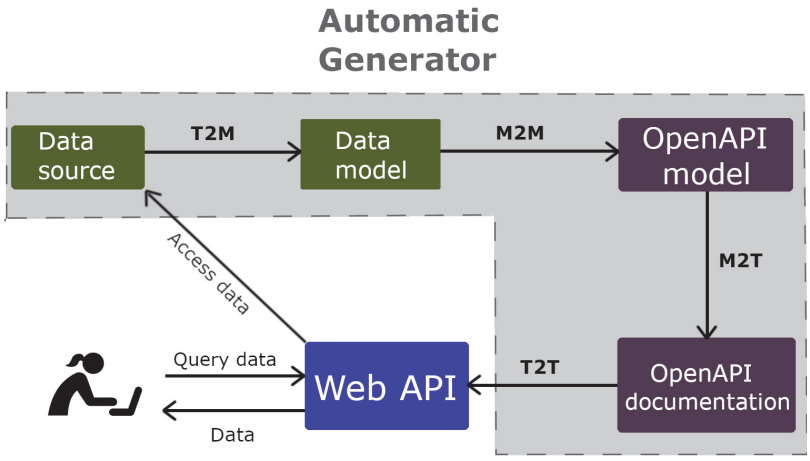

Figure 1 Automatic API generation process.

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/

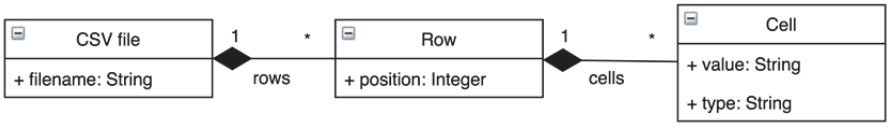

Figure 2 CSV datafile metamodel.

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/

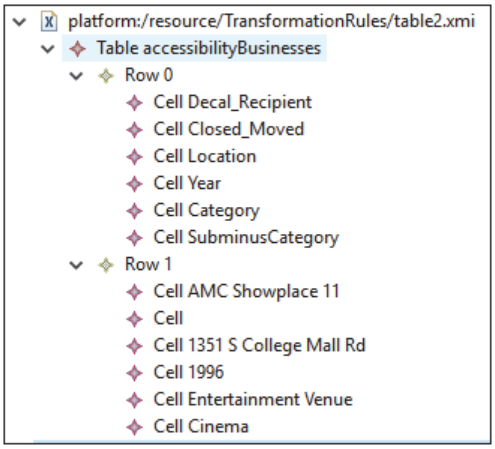

Figure 3 Datafile model in XMI format.

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

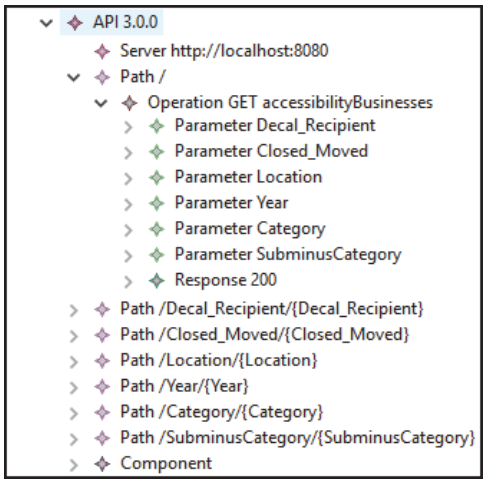

Figure 5 Extract of OpenAPI model (XMI).

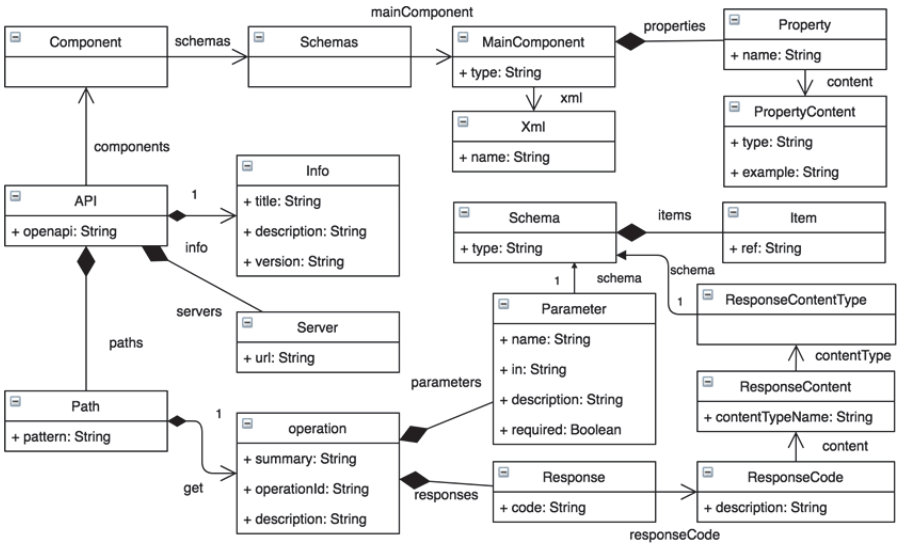

Figure 6 OpenAPI metamodel.

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

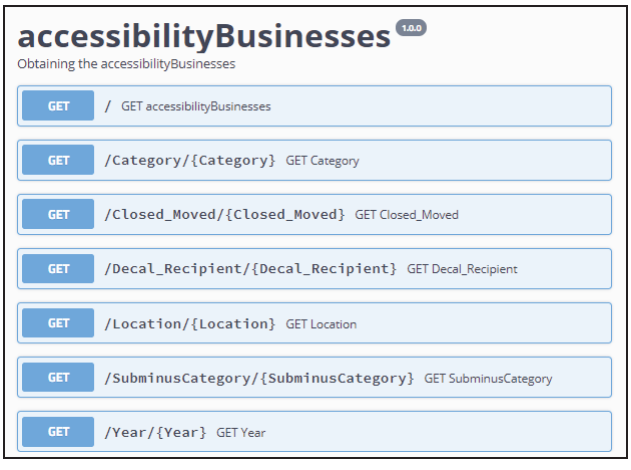

Figure 8 Interactive OpenAPI documentation of the API.

[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.

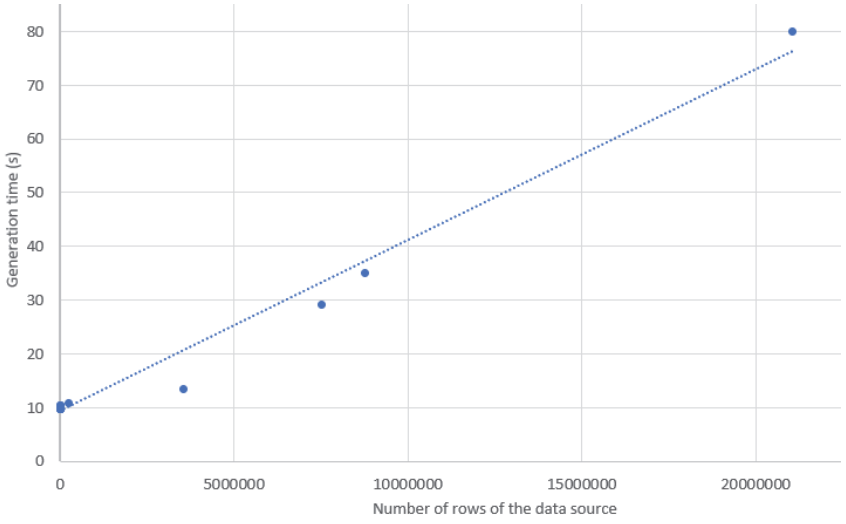

Figure 9 Generation time of the approach depending on the evaluated source file size.

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.