Managing Open Data Evolution through bi-dimensional mappings

Henrique Varella Ehrenfried*, Eduardo Todt,†, Daniel Weingaertner, ‡, Luis C. E. Bona§,
Fabiano Silva¶, Marcos Didonet Del Fabro‖
Center of Scientific Computing and Free Software (C3SL Labs)
Department of Informatics
Federal University of Paraná (UFPR)
R. Cel. Francisco H. dos Santos, 100 – Curitiba – PR – Brasil
Email: *hvehrenfried, †todt, ‡danielw, §bona, ¶fabiano, ‖marcos.ddf@inf.ufpr.br

***Abstract*—The availability of large Open Data sources creates opportunities for data analytics on different domains. But in order to be effectively used, the data needs to be correctly extracted, formatted and integrated, which is a specially challenging task on Open Data sources, since there is usually less rigour in standardizing subsequent data releases. This means Open Data evolution must be handled. A domain specific solution, taking stock of existing approaches, but with delimited kinds of operations and mappings, would be useful for providing coarse-grained management of data evolution operations throughout time. In this paper, we present an Open Data Evolution managing solution, aiming to integrate periodically released data sets. We define a set of operations acting over the instances, schema and mappings, which are executed after each new data release. These operations rely on the existence of a *time dimension* in the input mappings. The approach is validated on a real-world case study, which is being currently used to integrate and access a large Brazilian educational Open Data source, with billions of records and hundreds of columns evolving over many years. The proposed solution is used to process this data source, successfully integrating more than 90 data releases from 2012 to 2018.**

I. Introduction

The diversity and volume of Open Data sources has grown rapidly in recent years, partly due to policies of governmental data disclosure and transparency adopted by many public institutions. The information contained in these data sources has great potential to promote social improvements and enhance the knowledge of citizens about their own society. But these data sets are intrinsically difficult to process and combine because they are very heterogeneous, both, in the form of their availability and in their format/structure. In order to be usable, the data needs to be correctly extracted and integrated throughout many time-bound releases to enable data science.

Open Data sources are often made available in a denormalized format, e.g., in large CSV (Comma Separated Values) files representing a fixed period of time. Extracting and integrating data from a specific release usually involves manual adjustments to the existing schema, and this needs to be periodically redone for every new data release in response to changes or evolutions of the data, in what can be categorized as an Open Data evolution problem.

We illustrate this problem though an example. Consider the necessity of extracting an indicator containing the number of undergraduate teachers from all Brazilian schools for a period of 5 years. The information can be extracted from an Open Data source produced by a Brazilian Educational Agency. While this is a particular indicator, the scenario can be generalized as "*periodically extracting indicators from public Open Data sources*". The indicator must be correctly extracted from this source and can be aggregated in different dimensions. However, the data is not provided on a single formatted source, but on a yearly basis, which means that the extraction operations needs to be performed for every year. Besides, there are changes on the data definition and format from one period to another, making it difficult to integrate the Open Data.

There are different solutions/tools that could be used to handle these issues. The Clio tool [1] is one of the best known, being developed before the advent of Open Data and a precursor to many others. More recently the *Polystore*-like approaches, such as *BigDawg* [2], ESTOCADA [3] or MISO [4] focus on the data integration issues, as does the Data Civilizer system [5]. However, periodically released sources need to be integrated using ad-hoc scripts. Concerning this last issue, there are solutions concentrating on schema and instances changes, such as CoDEL [6] or BiDEL/InVerDa[7], providing a set of element-based operations to handle evolution. They provide means to keep track of evolution, and to have multiple concurrent versions of the schema. Model management solutions [8] first presented operators to manage metadata, i.e., schemas and mappings, then were extended to manipulate data [9] [10]. In order to take stock of these solutions in an envolving Open Data scenario, it would be important to define a domain specific approach, adapted to the management of the operations handling the constant and periodic evolution of the Open Data sources along time.

In this paper, we present an approach for the management of Open Data evolution. We define a set of operations that may be executed periodically, which handle data and schema insertions, updates and deletions. We provide a set of formal definitions for each operation, which are technology independent. The operations use a set of mappings as input, defining how the input data and schema are integrated into a single data source. The mapping definitions have a *time* dimension

attribute, enabling their clear periodical organization. The proposed mapping format is defined in CSV files, thus being an Open Data source itself, and allows to specify complex source-to-target translations

We validate our approach through a real-world case study where periodically released data sets about schools and Universities in Brazil were extracted and integrated into a single data source. This also publicly available data source contains hundreds of attributes and billions of records, spanning over more than 90 data releases since 2012. The specific tooling aspects of this approach have been presented as a Demo at EDBT 2019 [11]. In this paper, we focus on the methodology, the operations' flow and their definitions in a more general approach.

The paper is organized as follows: section II presents our approach for Open Data evolution management; section III presents a validation through a case study; section IV is the related work and section V presents the conclusions.

II. Managing Open Data Evolution

In this section we describe our approach for managing Open Data Evolution using bi-dimensional mappings. It has three main artifacts: 1) the *input data*, often extracted from CSV files; 2) the *target database*, where the data is integrated and 3) the *bi-dimensional mappings*, which specify how the data is translated into the target database. These artifacts are manipulated by a set of *operations*, following specific execution flows.

Definition 1 (Target database)**.** *The target database $D = \{R_1, ..., R_n\}$ is a database composed by a set of n tables. Each table R_i is a set of k instance tuples $R_i = \{(r_{1,1}, r_{1,2}, ..., r_{1,m}), ..., (r_{k,1}, r_{k,2}, ..., r_{k,m})\}$, where m is the number of columns of R_i. If the table is empty, $k = 0$. Each table has a header $H(R_i) = (h_1, h_2, ..., h_m)$ and $H_j(R_i) = h_j$ corresponds to the header of column j of R_i. If x is a column header of R, then $R(x) = \Pi_x(R)$ is the column x of R.*

The database stores sets of tabular structures, where each table has one header and its corresponding instances. For the remaining of the paper, we assume a relational database management system as a concrete implementation to illustrate the operations.

The $create$ operation is used to create a new empty table R' in the target database, according to the provided description. It is defined as follows:

$$create(R') : \quad D_{new} = D \ \cup \ \{R'\}$$

The create operation is implemented following the specification of the target database. When considering relational databases, it generates a SQL create table expression based on the table specification, with its corresponding columns and types. If necessary, other database-specific elements could be implemented, such as primary or foreign keys.

Tables can be deleted, whether they have instances or not, by executing the $drop$ operation defined as:

$$drop(R) : \quad D_{new} = D \setminus \{R\}$$

The drop operation generates a SQL drop statement to remove a table from the database, with all its corresponding data. This operation may be frequently used in scenarios with frequent changes or tests on the data extraction. Once the database structure is created, it is necessary to include and maintain the Open Data sources.

Definition 2 (Input file)**.** *The input file is defined as $F = \{(f_{1,1}, f_{1,2}, ..., f_{1,m}), ..., (f_{k,1}, f_{k,2}, ..., f_{k,m})\}$, where each $f_{i,j}$ is a cell that contains data. The file F contains a header $H(F)$ that can be defined as $H(F) = (h_1, h_2, ..., h_m)$ where each $H_j(F) = h_j$ is a column header. If x is a column header of F, then $F(x) = \Pi_x(F)$ is the column x of F.*

This definition can represent CSV (Comma Separated Value) files, with a set of columns separated by headers and its corresponding data. CSV files are one of the most used formats in Open Data sources, because it is simple and easy to share and to process.

Definition 3 (Mapping table)**.** *A mapping table MT_R is a table that contains the mappings between each column of a target table R and the corresponding column of the input file F_y of the year y. Let Y be the set of years for which there are input files related to table R. The mapping table is defined as:*

$$MT_R = \{ (t, f_1, \ldots, f_y, \ldots, f_{|Y|}) \mid \ \forall t \ \exists j \ H_j(R) = t, \ \exists i \ H_i(F_y) = f_y, \ f_y \mapsto t \ \text{ where } f_y \mapsto t \text{ is the mapping between the input column } f_y \text{ associated with the year } y \text{ and the column } t \text{ of the target table } R.$$

Definition 4 (Transformation)**.** *A transformation is a function that translates data from one or more input columns into a column in the target database.*

Definition 5 (Bi-dimensional mapping table)**.** *A bi-dimensional mapping table B_R is an extension of a mapping table MT_R where, for each year, a target column is associated to either a transformation function or to an input column.*

The year information is a separate dimension, specified as *column headers* from the B_R. The execution of all mappings is defined by the combination of three functions: $MapTo$, $exec$ and Map. Function $MapTo$ is a higher-order function that, given a specific year y and the target column z of table R, returns the mapping:

$$MapTo(y, z, R) = \Pi_y(\sigma_{H_t(B_R)=z}(\Pi_{H_t(B_R),y}(B_R)))$$

Function $exec$ returns the column values of applying the $MapTo(y, z, R)$ function to R, which is either a column stored in F_y, either the result of the transformation function returned

by $MapTo$. It takes four parameters: the input file F_y, the year y and the target column z of R:

$$exec(F_y, y, z, R) = \begin{cases} apply(MapTo(y, z, R)), \\ \text{if } MapTo \text{ is a transformation,} \\ \\ F_y(MapTo(y, z, R)), \\ \text{otherwise} \end{cases}$$

where *apply* generates the column values by applying the transformation returned by $MapTo$ to the input file (and other tables of the target database if necessary).

Finally, Map is the complete bi-dimensional mapping execution, i.e., it return the target tuples, which calls $exec$ for each line of the mapping table B_R:

$$\textbf{\textit{Map}}(F_y, y, B_R) = \{\ exec(F_y, y, z, R) \mid \forall z, z \in B_R(H_t(B_R))\ \}$$

This execution is used during the insert operation, which is defined as:

$$insert(F_y, y, R) : R_{new} = R \cup \textbf{\textit{Map}}(F_y, y, B_R)$$

Figure 1 illustrates the insert operation, using as input a CSV file F containing country names and a corresponding code. The mapping table (MT in the figure) contains the target column names in the *DB_Name* column, and two sets of mappings for each column, for years 2018 and 2019, specifying the time dimension. The columns *code_country* and *name_country* are directly copied from the CSV into the target database. The column *continent* is calculated using a transformation function that generates a continent name, using the data being inserted. The functions are implemented as SQL *CASE* statements, which are injected into a SQL bulk insert command (a bulk insert command in SQL insert the whole table in a single operation).

The *remap* operation is responsible for implementing modifications on the table structure and mapping values in cases when a new release of an Open Data source differs from already inserted data. After the creation of a table, a simplified version of the mapping table, called the *MS*, is stored in the database and used by the $remap$ operation. The $remap$ operation changes the table structure, by creating, removing or renaming columns:

$$remap(C, R) : \quad R = \textit{create(C,R) or delete(C,R) or rename(C,R)}$$

Given a set of columns $C = \{c_1, c_2, ..., c_m\}$ and the table R, the *create* function modifies the table structure of R by adding the set of columns into R, with the corresponding data left blank. The *delete* function removes the data and the column definition from table R.

The *rename* function updates the name of the columns in the target table, preserving the existing data. To achieve this, it creates an auxiliary table R_a, containing the current set of columns to be renamed and its corresponding data. Then, the function deletes the original columns from the target table, creates the new columns given by C and inserts the data stored in the auxiliary table R_a.

Figure 2 illustrates the $remap$ operation when adding a new column. It is an extension of the scenario shown in the $insert$ operation (Figure 1), to keep a country table up to date. The operation adds a new column called *calling_code*. The mapping could as well specify the dropping or renaming of an existing column. This synchronization helps the curator of the data to achieve consistency, improving the database's schema and its documentation. The operations can be implemented as SQL DML (Data Manipulation) and DDL (Data Definition) statements.

In order to modify existing data without modifying the table structure, it is necessary to execute the *update* operation. The operation receives as parameter the target table R, the columns to be updated C, and a temporary table R_t, which is created combining operations $create(R_t)$ and $insert(F_y, y, R_t)$. The operation updates the columns of the target table R, from a given year y, with the data from R_t, as defined below:

$$update(R, C, R_t) : \ R = \forall c, c \in C,\ \forall z, z \in R_t(c),\ \forall r, r \in R(c),\ r \leftarrow z$$

where $r \leftarrow z$ is the update of the field r with the corresponding data in z.

Figure 3 shows the $update$ operation. It creates a temporary table whose structure is a copy of the target table's structure. The new data is inserted into the temporary table and then each column that should be updated is copied form the temporary table into the target table. Note that it is executed for a specific year, meaning that only the columns of a given year are updated.

The set of defined operations (*create, drop, insert, remap, update*) with the mapping table and the dimension-based mappings enables managing the extraction and integration of periodically released Open Data sources, which can be further analyzed through the implementation of individual indicators or using specific data science approaches. A possible workflow for these operations is shown in Figure 4, were all possible flows are presented.

All operations start by verifying the existence of a table passed by parameter, triggering the next step, except when creating a table. If the next step is triggered, each operation modifies the database following its definition. The update operation changes the values of the table; the remap operation updates the table schema and data to match the mapping the insert operation adds more data into the table; the create operation adds a new table into the database; the drop operation removes a table from the database. If there is any problem in any operation, an error message is triggered and the operation is completely un-done. Once an operation cycle finishes, other operations can be executed, for instance, we could have the following flow of operations: 1) create table; 2) insert data; 3) update data (N-times); 4) remap; 4) update data. The operation that is constantly called for each new periodical release is the update operation.

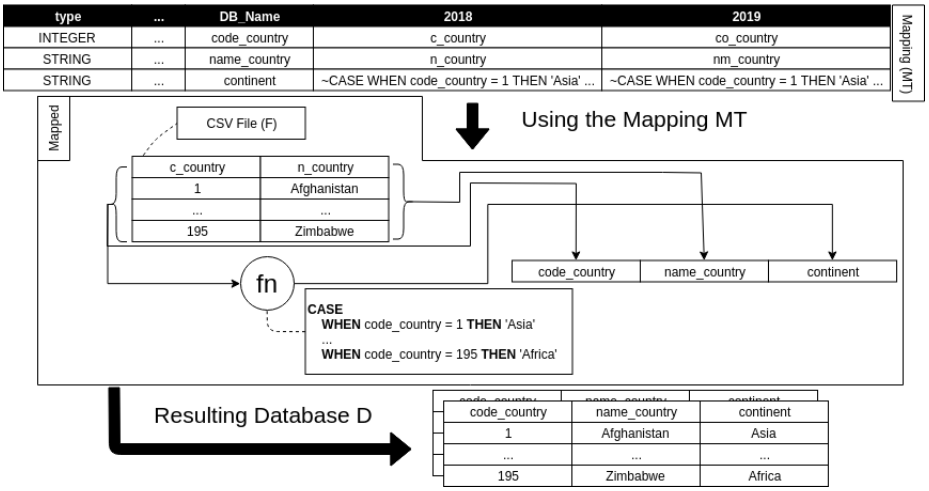

Fig. 1. Using a mapping table and an input CSV to insert data about countries

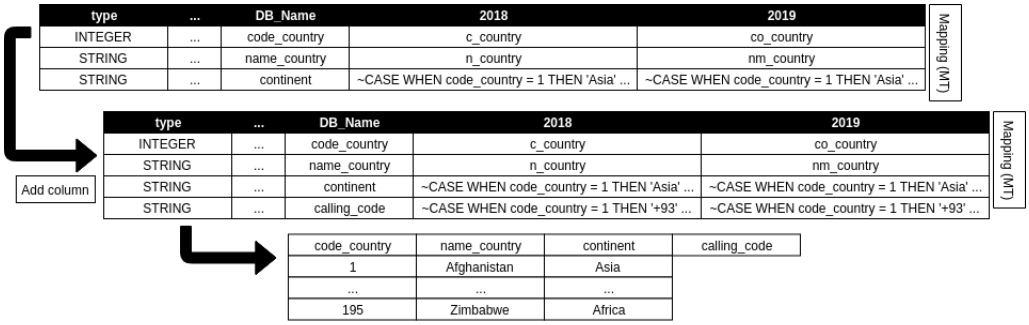

Fig. 2. Adding a new 'calling_code' column into an existing table using the remap operation

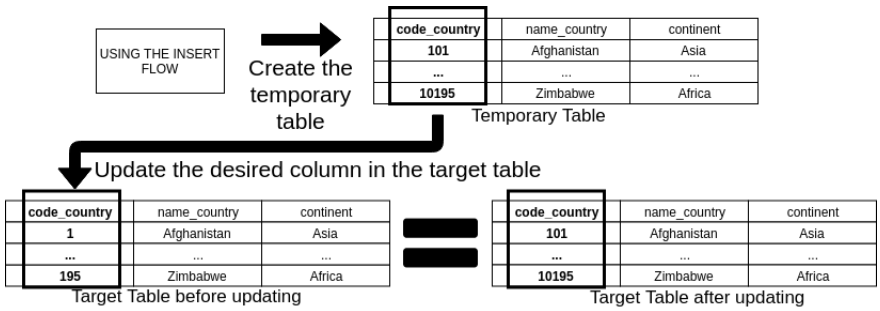

Fig. 3. Updating the country code using the update operation

III. Experimental validation

In this section, we validate our domain-specific approach through a case study, to show the utilization of bi-dimensional mappings and a set of operations for managing Open Data evolution. We first define three research questions (**RQs**) to be answered:

RQ1: Management operations – Do the proposed operations improve manageability of Open Data evolution? This question is relevant for data scientists and developers maintaining the periodical releases.

RQ2: Bi-dimensional mapping table – Is there a difference in having a bi-dimensional mapping table, to understand the mappings categorized by year and to maintain them? This question is relevant for the mapping developers and data scientics that need to update the sources after every new release.

RQ3: Transformation functions – Is there an advantage to be able to choose between an input column of the CSV or to use transformations over existing data? This question is of particular relevance to mapping developers.

We implemented the concepts presented in this paper in the *HOTMapper* tool[1]. It is currently being actively used by

[1]The HOTMapper tool code is available at https://github.com/C3SL/hotmapper

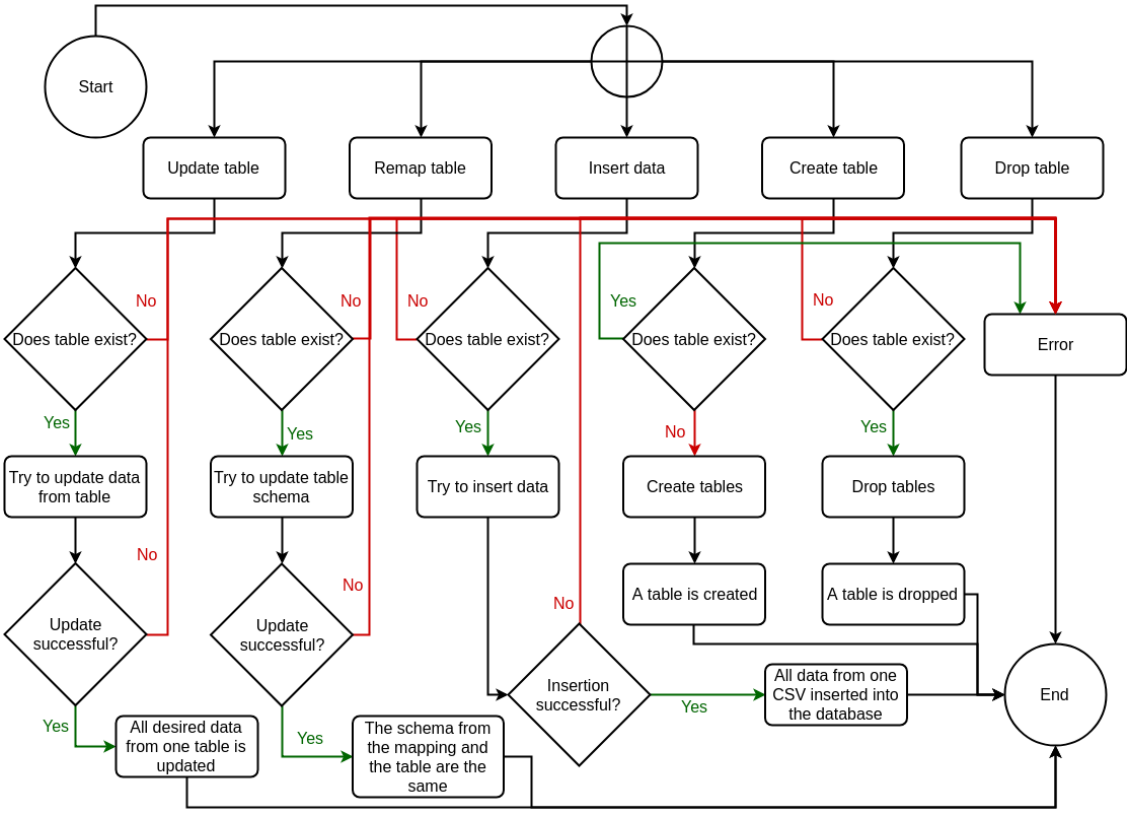

Fig. 4. Execution flow of the operations

two real-world Open Data applications. The first one, called SMPPIR[2], helps to monitor social inclusion indicators on higher-education. The second project, LDE [3], helps to monitor key indicators about the Brazilian educational system. Both web portals use our approach to manage the periodical Open Data releases.

The information is released periodically as Open Data sources, in a yearly basis. These sources are de-normalized tables in CSV files, with hundreds of columns each. Table I shows the number of records and tables processed and inserted into the target database, which is a column store called *MonetDb*[4].

TABLE I
OPEN DATA SOURCES PROCESSED

Year	Input Tables	Tuples
2018	8	76,779,455
2017	16	114,339,874
2016	15	109,803,647
2015	15	117,044,046
2014	15	120,084,335
2013	15	112,408,839
2012	11	36,258,716

[2]Accessible at https://seppirhomologa.c3sl.ufpr.br - In portuguese
[3]Accessible at https://dadoseducacionais.c3sl.ufpr.br/ - In portuguese
[4]MonetDb database: http://www.monetdb.org

Using the SMPPIR application, we describe one possible execution flow for a new set of data that is made available and needs to be inserted into the target database. The first step is to create the mapping file MT. We use a simplified version of the real world application with a small subset of columns (excerpt shown below[5]). It contains the mapping table that maps a code of specific institutions (such as universities) and the institution creation date in four lines: (1) the headers, where the first four fields are fixed, and the remaining fields are the time dimension added for each released year; (2) a complete 1-to-1 mapping with the institution code; (3) a mapping with the institution start date. The data for this specific field is not available for all years, so the corresponding column for non-existent years is left blank; (4) a complex mapping using a transformation function, only for year 2016 (notice the blank fields for previous years).

1) `Std.Label,New Label,DB Name,Type,2010,2011,2012,2013,2014,2015,2016`
2) `COD_INST,Institution code,cod_inst, INTEGER, CO_INST, CO_INST, CO_INST, CO_INST, CO_INST, CO_INST,CO_INST`
3) `START_DATE,Start date,start_date,VARCHAR(255), , , ,START_DATE,START_DATE, START_DATE, START_DATE`

[5]The complete mappings used in both scenarios are available in the following Git repositories: https://gitlab.c3sl.ufpr.br/simcaq/mapping_protocols and https://gitlab.c3sl.ufpr.br/SMPPIR/SMPPIR-Mapping-Protocols/tree/master/Protocols.

```
4) PROF_STUDIES,Prof. Studies,
   professional_studies,BOOLEAN, , , ,
   , , ,CASE WHEN ("TEACHING_LEVEL"=1 OR
   "TEACHING_LEVEL"=2) THEN 1 ELSE 0 END
```

The transformation function in line 4 maps information about professional studies (PROF_STUDIES), which is the combination of values of an existing teaching level (TEACHING_LEVEL). This is a more complex mapping, showing the possibility to calculate values from existing columns in the source database.

The workflow uses these mappings and the definition file to create a new table followed by the insertion of its data. First, it executes the $create$ operation, which produces two new tables: An auxiliary table containing the mappings and the target table that stores the data.

During the *insert* operations two types of mapping are possible: 1) a 1-to-1 mapping between input and target columns; 2) a mapping that calls a transformation function, which is evaluated for each column for which it is specified. The transformation function must be specified in a language that is compatible with the target database. In our scenario, we use SQL CASE statements.

Every time a new data source is released, this mapping is passed as parameter to the management operations, and with new columns for each year. To update it, the data scientist updates the mapping, runs the $remap$ function if there are changes in the mapping and inserts the new data. The operations need to be re-executed, with new mappings, for every new release. The number of tables from Table I indicates that we executed the *create* and *insert* operations for at least 95 times to populate the tables, at least one *update* or *remap* per new data source that is updated. The *drop* operation is used when the developed transformations are not correct and the insertion need to be re-executed from scratch.

The research questions were answered as follows. For **RQ1**, the set of operations narrows the scope of possible operations on the data sources, making it clear what needs to be done to create a new data source and to update it periodically, so the question is positively answered. This is an advantage over generic frameworks, which have a higher expressive power, but where the operations are not explicit. For **RQ2**, the bi-dimensional mapping table enables having explicit mappings for each new release, even in cases when they are copies of the previous year. The mappings are centralized, and it is possible to have a general view of the evolution, so this question is answered positively as well. In addition, the mappings themselves can be published as Open Data sources, enabling to keep track of the implemented operations. This traceability is important for data scientists to assess the validity of the data transformations. Finally, to answer **RQ3**, the possibility to write a transformation over the existing data is important to enable the generation of new values, when only copying the input data is not possible. In addition, the simplicity of the format of the mapping table enables sharing it with other researchers from similar domains, so they can validate and improve the mapping definitions. The utilization of a tool implementing such concepts in two real-world scenarios for releases from 7 years of educational data are a strong indication of the success of the approach.

To summarize, the approach creates a domain specific solution, which narrows the set of possible operations and mappings that can be done for managing Open Data evolution. This assumption makes the solution less expressive, for instance, it is not targeted to complex and complete ETL or data exchange workflows. The specificity of the approach enables to concentrate in a diminished and well stablished operations and simple mappings.

IV. Related Work

Managing the evolution of Open Data sources often span over different research subjects. In Data Integration solutions [12], it is necessary to uniformly access different source schemes through an integrated source [12]. The approach from [13] presents a brief history on data integration solutions and the current issues when having multiple Open Data sources. Their solution for creating union-able tables [14] could be valuable for our framework to create the initial mapping table, but they do not focus on maintenance and updates of the data.

Data exchange approaches enable the translation of source data into a target data, where one of the most known solutions are the Clio tool [1]. It could be used for defining the mappings to be used in an yearly basis. Business Intelligence frameworks, such as *Saiku*, *IBM Cognos* or *QLikView*, or many others, could be used as well. They are very expressive, providing complete frameworks accessible for the developers or data analysts, often providing graphical interfaces or specialized languages. However, due to its genericity, there is no design indications on which operations could be used on evolution. This means new open data evolution projects need to be defined from scratch. The periodically released sources need to be integrated using ad-hoc scripts, or the queries need to be manually updated, which means there is not a specific workflow for handling evolution. In addition, the correspondences between the evolving data is not explicitly published as Open Data sources as well.

There has been extensive work on schema and data evolution, such as the surveys from [15] and [16]. The solutions are based on element-to-element evolution, where each operation, often called SMO (Schema Modification Operation), are handled using different techniques. Two recent approaches are CoDEL[6] and BiDEL/InVerDa [7]/[17]. BiDEL is a language to specify bi-directional SMOs, while InVerDa generates code to handle evolution; it also enables having multiple schemas evolving concurrently. In addition, the updates are processed ”online”, i.e., after each single element update. Our approach has a different main goal, since we do not have specific SMOs to handle each modification, but we have a full copy of the columns and transformations for each new period, to integrate into a single source. This limits the possible modifications, and it does not allow multiple concurrent versions, but the management through time is simpler, together with the set

of operations. In addition, the transformations enable data evolution.

Model management solutions [8] first presented the idea of using operators to handle metadata and data. They provide operators such as *Merge*, *Match*, *Extract*, *Apply*, which can be used in an algebra and combined to handle schema evolution, among other issues. As in our approach, the evolutions are handled "off-line'.These approaches evolved to express richer mappings, manipulating data as well [9] [10]. Our operations could be considered as extensions of the *Apply* operator, targeted to Open Data management workflow, by handling both schema and data evolution.

V. Conclusions

We have presented a domain-specific approach for management of Open Data evolution based on a set of operations and bi-dimensional mappings. We narrow the scope of possible operations and mappings, so it can be used as a starting point for data scientists who do not need a complete framework for defining any kind of ad-hoc transformations over data. We advocate that a domain specific framework with such characteristics can be used in many Open Data evolution scenarios, because there is a large availability of data in denormalized CSVs.

The operations *create*, *update*, *insert*, *remap* and *drop* facilitates the management of the evolution of Open Data sources in order to create integrated data sources. The execution flow of the operations has been shown effective, being validated through a case study in a real-world scenario, covering more than 90 data releases. The operations take as input a mapping table, which contains information on how to translate data and schema of the input sources into a target integrated database. The mapping table stores the mappings with columns categorized by the release period, which we call *time dimension*. This enables maintaining the mappings from one release to another, without loosing track of the transformations executed.

The mappings are stored in a tabular (CSV) format, thus being an Open Data source themselves, making it possible to audit them. This is an important issue to enable third parties to assess the correctness of the mappings, and also its validity with respect to a specific domain analysis, i.e., if a given indicator is really applicable and important, for instance, to drive a given public policy. This assessment can only be done by specialists.

As future work, we plan to develop a graphical interface to ease the task of creating the mappings and managing the proposed execution flow. We also plan to make broader experiments about the usability of the tool.

References

[1] R. Fagin, L. M. Haas, M. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis, *Clio: Schema Mapping Creation and Data Exchange*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 198–236.

[2] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. Zdonik, "The bigdawg polystore system," *ACM Sigmod Record*, vol. 44, no. 2, pp. 11–16, 2015.

[3] F. Bugiotti, D. Bursztyn, A. Deutsch, I. Ileana, and I. Manolescu, "Invisible glue: scalable self-tuning multi-stores," in *Conference on Innovative Data Systems Research (CIDR)*, 2015.

[4] J. LeFevre, J. Sankaranarayanan, H. Hacigumus, J. Tatemura, N. Polyzotis, and M. J. Carey, "Miso: souping up big data query processing with a multistore system," in *Proc. of the 2014 SIGMOD*, 2014, pp. 1591–1602.

[5] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang, "The data civilizer system." in *CIDR*, 2017.

[6] K. Herrmann, H. Voigt, A. Behrend, and W. Lehner, "Codel – a relationally complete language for database evolution," in *Advances in Databases and Information Systems*, M. Tadeusz, P. Valduriez, and L. Bellatreche, Eds. Cham: Springer, 2015, pp. 63–76.

[7] K. Herrmann, H. Voigt, T. B. Pedersen, and W. Lehner, "Multi-schema-version data management: Data independence in the twenty-first century," *The VLDB Journal*, vol. 27, no. 4, pp. 547–571, Aug. 2018.

[8] P. A. Bernstein, "Applying model management to classical meta data problems," in *CIDR 2003, First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 5-8, 2003, Online Proceedings*, 2003.

[9] P. A. Bernstein and S. Melnik, "Model management 2.0: Manipulating richer mappings," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07. New York, NY, USA: ACM, 2007, pp. 1–12.

[10] S. Melnik, P. A. Bernstein, A. Halevy, and E. Rahm, "Supporting executable mappings in model management," in *Proceedings of the 2005 ACM SIGMOD*. New York, NY, USA: ACM, 2005, pp. 167–178.

[11] H. V. Ehrenfried, R. Eckelberg, H. Iboshi, E. Todt, D. Weingaertner, and M. D. D. Fabro, "Hotmapper: Historical open data table mapper," in *22nd EDBT, 2019, Lisbon, Portugal, March 26-29, 2019*, 2019, pp. 550–553.

[12] C. Yu and L. Popa, "Constraint-based XML query rewriting for data integration," *Proceedings of the 2004 ACM SIGMOD international conference on Management of data - SIGMOD '04*, p. 371, 2004.

[13] R. J. Miller, "Open data integration," *Proc. VLDB Endowment*, vol. 11, no. 12, pp. 2130–2139, Aug. 2018.

[14] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," *Proceedings of the VLDB Endowment*, vol. 11, no. 7, pp. 813–825, 2018.

[15] E. Rahm and P. A. Bernstein, "An online bibliography on schema evolution," *SIGMOD Rec.*, vol. 35, no. 4, pp. 30–31, Dec. 2006.

[16] P. Manousis, P. Vassiliadis, A. Zarras, and G. Papastefanatos, "Schema evolution for databases and data warehouses," in *Business Intelligence*, E. Zimányi and A. Abelló, Eds. Cham: Springer International Publishing, 2016, pp. 1–31.

[17] K. Herrmann, H. Voigt, T. Seyschab, and W. Lehner, "Inverda - coexisting schema versions made foolproof," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, May 2016, pp. 1362–1365.