Reusable transformations of Data Cube Vocabulary datasets from the fiscal domain

Jindřich Mynarz[1], Jakub Klímek[123], Marek Dudáš[1], Petr Škoda[12], Christiane Engels[4], Fathoni A. Musyaffa[5], and Vojtěch Svátek[1]

[1] University of Economics, Prague
Nám. W. Churchilla 4, 130 67 Praha 3, Czech Republic
jindrich.mynarz|marek.dudas|svatek@vse.cz
[2] Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
klimek|skoda@ksi.mff.cuni.cz
[3] Czech Technical University in Prague, Faculty of Information Technology
Thákurova 9, 160 00 Praha 6, Czech Republic
jakub.klimek@fit.cvut.cz
[4] Fraunhofer IAIS
Schloss Birlinghoven, 53757 Sankt Augustin, Germany
christiane.engels@iais.fraunhofer.de
[5] University of Bonn, Institute for Computer Sciences
Römerstraße 164, 53117 Bonn, Germany
musyaffa@cs.uni-bonn.de

Abstract. Shared data models provide leverage for reusable data transformations. Common modelling patterns and data structures can make data transformations applicable to diverse datasets. Similarly to data models, reusable data transformations promote separation of concerns, prevent duplication of effort, and reduce the time spent processing data. However, unlike data models, which can be shared as RDF vocabularies or ontologies, there is no well-established way of sharing data transformations. We propose a way to share data transformations as 'pipeline fragments' for LinkedPipes ETL (LP-ETL), which is an RDF-based data processing tool focused on RDF data. We describe the features of LP-ETL that enable development of reusable transformations as pipeline fragments. Pipeline fragments are represented in RDF as JSON-LD files that can be shared directly or via dereferenceable IRIs. We demonstrate the use of pipeline fragments on data transformations for fiscal data described by the Data Cube Vocabulary (DCV). We cover both generic transformations for any DCV-compliant data, such as DCV validation or DCV to CSV conversion, and transformations specific for the fiscal data used in the OpenBudgets.eu (OBEU) project, including conversion of Fiscal Data Package to RDF or normalization of monetary values. The applicability of these transformations is shown on concrete use cases serving the goals of the OBEU project.

1 Introduction

Many data processing tasks share the same requirements on data transformations. For example, generic cleaning and enrichment tasks apply to a variety of datasets. The com-

monality of applicable data transformations can be based on the common data models used to describe the transformed datasets. In such case, common data models provide affordances for common data transformations. However, while data models can be shared as RDF vocabularies or ontologies, there is no well-established way how to share data transformations.

It is desirable that common data transformations can be separated into reusable components. Componentization into modular transformations endowed with single responsibility promotes separation of concerns, avoids duplication of code, and makes data transformations more maintainable. It reduces the development time needed for data transformations, thus decreasing the effort spent on pre-processing data.

In this paper we describe a way to share data transformations as 'pipeline fragments' for LinkedPipes ETL (LP-ETL) [5]. The pipeline fragments are represented in RDF and can be shared either as files or via dereferenceable IRIs. We show several reusable transformations built on top of the Data Cube Vocabulary (DCV) [1] that can be shared as pipeline fragments. We cover both generic transformations applicable to any DCV data and domain-specific transformations for fiscal data, comprising both budget and spending data, that are motivated by the use cases in the OpenBudgets.eu (OBEU)[6] project. More details about the presented transformations can be found in an OBEU deliverable on data optimisation, enrichment, and preparation for analysis [4].

OBEU is an EU-funded H2020 project devoted to advancing analyses of fiscal open data. In order to support the goals of the project we developed the OBEU data model, which is built on DCV. The data model is based on reusable component properties (i.e. instances of `qb:ComponentProperty`) that can be composed into dataset-specific data structure definitions (DSDs). We chose to adopt DCV because it provides the OBEU data model with a uniform way to represent heterogeneous fiscal datasets, which makes them better combinable and comparable. Moreover, DCV simplifies combining fiscal data described using the data model with other statistical datasets, such as macroeconomic indicators, which can put the data into a broader context useful for data analyses.

We start this paper with a description of the features of LP-ETL that enable building reusable data transformations. Subsequently, we cover reusable transformations built for DCV data for the purposes of the OBEU project, and we conclude with a discussion of how these transformations are used in the project. Before we proceed with the paper, we review related work in technologies for reusable transformations of RDF data.

1.1 Related work

While there are many ETL tools, only few allow reusing partial transformations of RDF data. Thanks to its declarative nature, SPARQL 1.1 Update [2] operations can be used to formalize reusable transformations of RDF. SPARQL allows to bundle several update operations separated by semicolons in a single request, so that it can encapsulate more complex transformations requiring multiple steps; e.g., create, populate, and delete a temporary named graph. However, SPARQL Update is restricted to transformations of RDF data, and as such it cannot incorporate tasks involving other data formats; e.g.,

[6] `http://openbudgets.eu`

decompressing a ZIP file. Moreover, SPARQL Update operations must be static. If they contain dynamic parts, another mechanism is needed to generate them.

Besides the standard SPARQL, there are bespoke tools for ETL of RDF data that allow reusing transformations. DataGraft [7] provides reusable components and transformations focusing on tabular data. UnifiedViews [6] features templates[7] that enable to pre-configure individual data processing units; e.g., with a specific regular expression. If we extend our scope beyond RDF, we can find Pentaho, a widely used ETL tool, which features mapping steps that can encapsulate reusable transformations.[8] The mapping steps, also known as sub-transformations, use placeholders for their inputs that they expect to be provided from their parent transformation and for the outputs that they provide back to their parent transformation to read. The mapping steps can be stored in separate files and imported into other transformations.

However, none of these tools satisfies all requirements for easy reuse of data transformations. DataGraft focuses only on tabular data, while there are other formats such as XML, JSON and RDF which need to be processed. UnifiedViews is not very user-friendly and has issues with sharing pipelines between multiple versions of the software. Pentaho does not support RDF transformations. This is why we chose LP-ETL, described in the next section, as the most suitable candidate for our use case.

2 LinkedPipes ETL

LP-ETL is a data processing tool focused on producing RDF data from various data sources. A data processing task in LP-ETL is defined as a pipeline. Pipeline is a repeatable process that consists of configurable components, each responsible for an atomic data transformation task, such as a SPARQL query or a CSV to RDF transformation. ETL (extract, transform, load) denotes a data processing task in which data is first extracted, then transformed, and finally loaded to a database or a file system. LP-ETL consists of a back-end, which runs the data transformations and exposes APIs, and a front-end, which is a responsive web application that provides a pipeline editor and an execution monitor to the user. One of the distinguishing features of LP-ETL, when compared to other ETL tools, is the fact that the pipelines and all component configurations are themselves stored as RDF, which enables novel data processing workflows. Consequently, the use of the tool requires the knowledge of RDF, SPARQL, and related technologies. In addition, its use for statistical data processing requires the knowledge of DCV and the Simple Knowledge Organization System (SKOS)[9] vocabulary.

2.1 Runtime configuration

All component configurations in LP-ETL are stored as RDF. The configurations range from simple, such as a SPARQL query to be executed or a URL to download, to complex, as is the case of the component that transforms tabular data to RDF and contains

[7] `https://grips.semantic-web.at/display/UDDOC/3.+DPU+Templates+Section`
[8] `http://wiki.pentaho.com/display/EAI/Mapping`
[9] `https://www.w3.org/TR/skos-reference/`

a full mapping of the CSV columns to RDF. A key feature of LP-ETL is the possibility to provide a configuration to the component at runtime. In this way, a pipeline can create configurations for its components based on its input data. For example, this feature can be used to download multiple files specified in a list. First, a download component downloads the list, passes it to another component that parses the list and transforms it into a runtime configuration for another download component that subsequently downloads the files in the list. A more complex use of this feature is generating a SPARQL query based on the pipeline's input data and then executing it.

2.2 Pipeline fragments

Since LP-ETL pipelines themselves are stored using RDF as JSON-LD files, they can be easily shared on the Web. When exporting a pipeline from LP-ETL, the user can either download the pipeline as is or download it without potentially sensitive information, such as user names and passwords for uploading to a server, so that it can be shared safely. The exported pipeline can be then shared publicly, such as by making it available at a public URL.

Users of the shared pipeline can either upload the pipeline into their LP-ETL instance from a JSON-LD file or from an IRI that dereferences to the pipeline's representation in JSON-LD. The shared pipelines can be incomplete and may need to be provided with input or configuration to work. Incomplete pipelines constitute the so-called pipeline fragments, which can be imported into existing pipelines to reuse common transformations. Instead of referencing pipeline fragments directly, they are reused as copies to enable local modifications, so they are not automatically updated if the 'master' copy changes. In particular, this feature is useful for sharing transformations that require no or only minor adjustments to serve their purpose, such as having provided a URL of the input file. Pipeline fragments can enforce the contract of their interface, i.e. necessary configuration or requisite input, by using the *SPARQL ASK* component that allows to check if the contract's preconditions are satisfied by querying the fragment's input and asserting the expected boolean result. If the expectations are not met, the component makes the pipeline execution fail.

2.3 Sending input data to pipelines using HTTP API

Execution of LP-ETL pipelines can be triggered by sending an HTTP POST request containing input data for the pipelines. In this case, a component that receives the posted input data is placed at the start of a pipeline. Any file can be provided as input data. This feature allows the pipeline designers to pass data into a pipeline without needing to specify the source of the data directly in the pipeline. It promotes modularization of pipelines, so that a complex pipeline can be split into several simpler, possibly reusable pipelines that trigger each other.

2.4 Typical pipeline shape

In order to provide a concrete illustration of LP-ETL pipelines we discuss their typical shape. A typical shape of a pipeline transforming source tabular files into RDF using DCV and the OBEU data model can be seen in Figure 1.

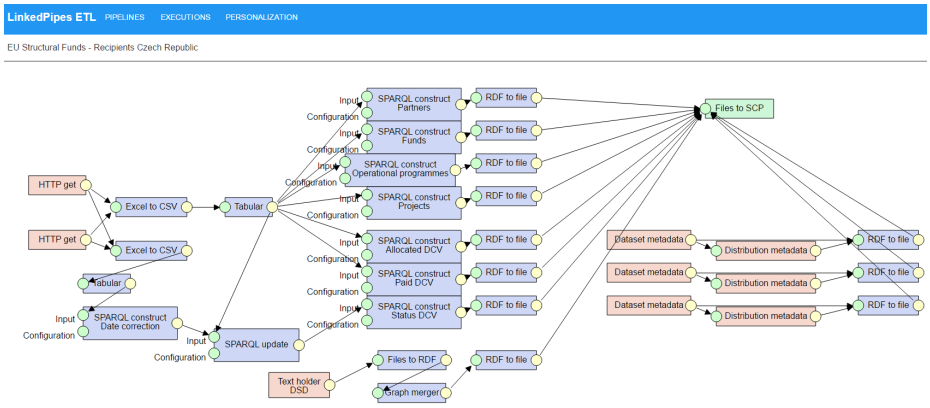

Fig. 1. An example pipeline in LP-ETL

First, the source tabular data in the Excel format is downloaded by the *HTTP Get* component. In the next step, it is converted to CSV files using the *Excel to CSV* component, which can also do some basic operations like sheet selection and rows and columns selection. CSVs are in turn transformed to raw RDF using the *Tabular* component. This component is based on the CSV on the Web W3C Recommendation[10] with some technical adjustments to better serve the ETL process. For example, we adjusted the way property IRIs are generated, as in the specification they are based on the CSV file name, which makes it impractical for batch processing of multiple files, where the property IRI needs to be fixed across files. The raw RDF data is subsequently mapped to the desired vocabularies, typically via a series of *SPARQL CONSTRUCT* or *SPARQL Update* components. The resulting RDF data is then serialized to dumps and uploaded to a target server. During the process, one source table can be split into multiple RDF datasets (three are used in Figure 1) due to data modeling reasons. One example of such a reason can be an input table, which is formatted for printing and besides individual transactions contains quarterly sums of the transactions. In this case, we may decide to transform it as two datasets, one containing the individual transactions and another one containing the quarterly sums. In addition to the data itself, metadata can be also added in the pipeline, providing important information about the datasets themselves.

2.5 Performance and scaling

Given the nature of the sequential pipeline processing, the size of the input data and the complexity of the pipeline are the key factors affecting the overall performance of the transformation. So far, every component has all of its input and output data materialized in an in-memory RDF store, which poses restrictions on the size of data. The number of components affects the transformation runtime as the data is copied in each step. The upside of this approach is the availability of intermediate data for debugging purposes.

[10] `https://www.w3.org/TR/csv2rdf/`

There are multiple ways of optimization being developed, but those are out of the scope of this paper.

3 Collection of reusable transformations

We created several reusable transformations to support the goals of the OBEU project. The transformations are implemented as LP-ETL pipeline fragments and are available at `https://github.com/openbudgets/pipeline-fragments`. While the individual technologies used in these pipelines are not new, the way they are combined is novel. We can split the transformations into generic ones that are applicable to any DCV-compliant data and ones that are applicable only to the fiscal data represented with the OBEU data model.

3.1 Transformations of DCV-compliant data

The DCV specification [1] provides two standard transformations: DCV normalization and validation using integrity constraints. Both these transformations are encoded in SPARQL. We reused these formulations and extended them into pipeline fragments. Additionally, we developed a pipeline fragment for DCV to CSV conversion.

DCV normalization DCV data can be coerced to a regular structure using the DCV normalization algorithm,[11] which allows to simplify queries on the data. The algorithm is expressed via SPARQL Update operations, which makes it simple to wrap in a LP-ETL pipeline fragment via its *SPARQL Update* component. DCV normalization can serve for pre-processing input data for subsequent transformations. Since the normalized DCV data follows a regular structure, transformations of such data are simpler due to a less heterogeneous input.

DCV validation DCV defines 22 integrity constraints[12] that formalize some of the assumptions about well-formed DCV-compliant data. Apart from one constraint that tests datatype consistency, the remaining 21 constraints are implemented via SPARQL ASK queries that evaluate to `true` when a constraint violation is found. While 19 constraints are static, two constraints must be generated dynamically based on the dataset to be validated. We reformulated the constraints as SPARQL CONSTRUCT queries that produce descriptions of the errors found in the validated data. The errors are represented by the SPIN RDF[13] vocabulary, which allows to pinpoint the RDF resources causing the errors and provide explanations to users to help them fixing the errors. In this way, instead of boolean answers from ASK queries, users receive richer descriptions of the detected constraint violations. We generate the dynamic integrity constraints

[11] `https://www.w3.org/TR/vocab-data-cube/#normalize-algorithm`
[12] `https://www.w3.org/TR/vocab-data-cube/#wf-rules`
[13] `http://spinrdf.org/spin.html`

using Mustache[14] templates and pass the generated queries further as runtime configuration. Apart from these constraints we generate the query for the integrity constraint 12 that detects duplicate observations. Compared to the generic query for this constraint in the DCV specification, we observed approximately 100× speed-up for the generated dataset-specific query. We implemented DCV validation as a pipeline fragment that executes the integrity constraints, merges their resulting RDF graphs, and outputs them both as RDF, which allows further automated processing, and as an HTML report for quick visual inspection by users.

DCV to CSV conversion Many data mining tools cannot handle RDF and instead require input data to be provided in a single propositional table. The required tabular structure can be serialized in CSV, which is considerably less expressive than RDF. However, DCV constrains RDF to follow a more regular structure. DCV dimensions and measures must adhere to 1..1 cardinality and attributes to 0..1 cardinality. Since multi-valued properties are missing from the DCV model, producing a single CSV table out of DCV data is simpler than doing so out of arbitrary RDF data. RDF can be transformed to tabular data using SPARQL SELECT queries [3], the results of which can be serialized to CSV.

We developed a pipeline fragment for DCV to CSV conversion, which is based on the data structure definition (DSD) of the transformed dataset. The columns of the CSV output are derived from the components composing the DSD. We extract the component specifications from the DSD, including their type, attachment, and order. The extracted data is used in a Mustache template to generate a SPARQL SELECT query that transforms the datasets conforming to the DSD to CSV.

3.2 Transformations of OBEU-compliant data

Besides the generic transformations applicable to any DCV data we also developed transformations that can be reused for any data described using the OBEU data model. In this section we describe the pipeline fragments for conversion from Fiscal Data Package (FDP) to RDF, validation of integrity constraints of the OBEU data model, and normalization of monetary amounts using exchange rates and GDP deflators.

FDP to RDF transformation FDP[15] is a data format used in the OpenSpending project.[16] Since the OpenSpending project and OBEU share some goals and FDP exhibits a similar structure to the OBEU data model,[17] it is straightforward to provide a transformation between the formats to enable to use FDP data in OBEU and vice versa. Therefore we have developed an FDP to RDF transformation pipeline in LP-ETL. FDP and OBEU data models cover the same domain and are semantically almost identical. FDP consists of CSV files accompanied by metadata in a JSON descriptor file

[14] `https://mustache.github.io`
[15] `http://fiscal.dataprotocols.org/spec/`
[16] `http://openspending.org`
[17] `https://github.com/openbudgets/data-model`

describing the meaning of CSV columns and possible relationships between them. The description is based on *dimensions*, which have a similar interpretation to the OBEU dimension properties. The FDP dimensions are mapped to CSV columns. The goal of the pipeline is then first to transform the metadata into RDF and then to transform the CSV records according to the metadata.

The input of the pipeline is a single JSON file – an *FDP descriptor* – containing references to CSV files with the actual data. The input CSV files are first transformed to RDF via the *Tabular* component to enable to manipulate them via SPARQL queries. The rest of the transformation is implemented via SPARQL CONSTRUCT queries. The FDP descriptor in JSON is reinterpreted as JSON-LD[18] and transformed to RDF. The output of the FDP to RDF pipeline is an RDF graph compliant with the OBEU data model. The pipeline is available at `https://github.com/openbudgets/pipeline-fragments/tree/master/FDPtoRDF`.[19]

The implementation in SPARQL was guided by several principles. We split the transformation into atomic steps, each having a single responsibility. This leads to better code organization that eases iterative development and improves maintenance. The decomposition of the transformation process simplified debugging it, because the intermediate outputs of each step were available to scrutiny, albeit it complicated the pipeline's structure. The transformation's state is propagated through the pipeline as auxiliary RDF annotations, which are read and updated by the transformation steps. The advantages of the declarative formulation of the pipeline are that it is easier to understand and maintain and it can be configured directly through the UI of LP-ETL. Conversely, the downside of this approach is the duplication of transformation code due to the granularity of SPARQL queries. It is often the case that transformation queries differ only slightly; e.g., transformation of different FDP dimensions to OBEU dimension properties, in which only the source and target dimension/property differ. It is possible that modularization of SPARQL queries could avoid such duplication and ease their maintenance.

Validation of the OBEU integrity constraints In a similar fashion to DCV, the OBEU data model provides integrity constraints that can detect violations of some assumptions behind the data model. The constraints cover the recurrent errors made by the users of the data model. As is the case in our implementation of DCV validation, these integrity constraints are formalized as SPARQL CONSTRUCT queries producing SPIN RDF data. Besides the patterns expressed in the SPARQL queries, these constraints leverage background knowledge encoded in the DCV and the OBEU data model. We implemented six integrity constraints in total, mostly testing the assumptions about datasets' DSDs.

1. **Redefinition of component property's code list:** Detects if the validated dataset redefines the code list for a coded component property reused from the OBEU data model. The validated datasets should instead define the custom code lists for a derived subproperty.

[18] `http://json-ld.org/`

[19] See also the pipeline's overview at `https://goo.gl/H6SSiE`.

2. **Hijacked core namespace:** The validated dataset must not invent terms in the namespace of the OBEU data model.[20] A different namespace should be used instead.
3. **Missing mandatory component property:** The validated dataset must contain mandatory component properties (or their subproperties) from the OBEU data model: `obeu-attribute:currency`, `obeu-dimension:fiscalPeriod`, `obeu-dimension:operationCharacter`, `obeu-dimension:organization`, and `obeu-measure:amount`.
4. **Property instantiation:** A common error we discovered was instantiation of properties. Since RDF only allows to instantiate classes, instantiating properties is incorrect. In the context of DCV, this error may be caused by typos in class IRIs that differ from property IRIs only in character case (e.g., `qb:DataSet` vs. `qb:dataSet`).
5. **Use of abstract property:** Properties marked as abstract in the OBEU data model (e.g., `obeu-dimension:classification`) should not be directly reused. Users should mint subproperties of the abstract properties and use these instead.
6. **Wrong character case in DCV:** The constraint detects if the validated dataset contains a non-existent term from the DCV namespace that differs from an existing DCV term only in character case. Apart from reporting the invalid term, the constraint suggests a valid substitute.

Normalization of monetary amounts In order to support comparative analysis combining fiscal data from different times and areas, we developed a pipeline fragment for normalization of monetary amounts using exchange rates and GDP deflators. This normalization allows to compare amounts of money not only as nominal values, but also in terms of their real value. Among others, monetary amounts can differ in currency, country, and time when they were spent. The amounts can be converted to a single currency, such as euro, using exchange rates and adjusted for different price levels depending on country and time using GDP deflators. The GDP deflator is a price index based on the gross domestic product (GDP) of a country that measures changes in price levels with respect to a specific base year. We can normalize an amount Q using the following calculation, in which Q' is the normalized monetary amount, $I_{p,t}$ is the price index for the target year to which we normalize, $I_{p,0}$ is the price index for the original year when Q was spent, and E_t is the exchange rate to euro for the target year:

$$Q' = \frac{I_{p,t} Q}{I_{p,0} E_t} \tag{1}$$

The normalization requires the monetary amounts to be enriched with two datasets: one containing exchange rates and the other containing GDP deflators. Eurostat[21] pro-

[20] `http://data.openbudgets.eu/ontology/`
[21] `http://ec.europa.eu/eurostat`

vides both these datasets in tabular format,[22] but thanks to Linked Statistics[23] they are also available in RDF modelled with DCV, which makes it easier to combine them with other data. We developed a pipeline fragment that uses these datasets to calculate normalized values of monetary amounts. The datasets are joined via country (and in turn by the country's currency) and year. The normalized datasets can either reuse Eurostat's resources for countries and years or provide explicit links to them. The above-described calculation for normalization is implemented in a dynamically generated SPARQL query.

4 OBEU use case

The pipeline fragments introduced in the previous section are used at several points in the OBEU project. The transformations of OBEU datasets take advantage of the two validation fragments as well as of the reusability of the datasets transformations themselves. In between the integration of datasets and the analytics part of OBEU, the DCV to CSV pipeline serves as connection point. We present two selected use cases to show how these transformations are applied.

4.1 Validation of OBEU datasets

An important part of the OBEU project is the integration of datasets from different sources, structures, and formats into one platform using the uniform OBEU data model. To this end, the datasets have to be transformed to this data model. For datasets in other formats than FDP, such as raw datasets from different governments' publication offices, we developed separate pipelines using LP-ETL.[24] Due to the huge diversity in the structure of the raw datasets and different data formats, the pipelines for transforming these datasets differ and have to be developed or at least adapted manually. Consequently, this is an error-prone process, for which an automated way to validate the transformed datasets with minimum effort is essential. Using the pipeline fragments described above, such a validation pipeline can be set up with a few clicks.

A validation pipeline for an OBEU dataset checking both the DCV and OBEU validity can be composed as follows (cf. Figure 2). First, the transformed dataset to be validated is loaded into the pipeline (e.g., downloaded from GitHub via the *HTTP Get* component) and normalized using the DCV normalization pipeline fragment since the following DCV validation step requires its input data to be in the DCV normal form. Then the two different validation steps, DCV and OBEU, are performed over

[22] Exchange rates: `http://ec.europa.eu/eurostat/web/products-datasets/-/tec00033`
GDP deflators: `http://ec.europa.eu/eurostat/web/products-datasets/-/nama_10_gdp`

[23] `http://eurostat.linked-statistics.org`

[24] Pipelines for EU-level datasets (e.g., the budget of European Structural Investment Funds) and various EU member states' regions and municipalities (e.g., Thessaloniki and Athens in Greece, Aragon in Spain, or Bonn in Germany) are available at `https://github.com/openbudgets/datasets`.

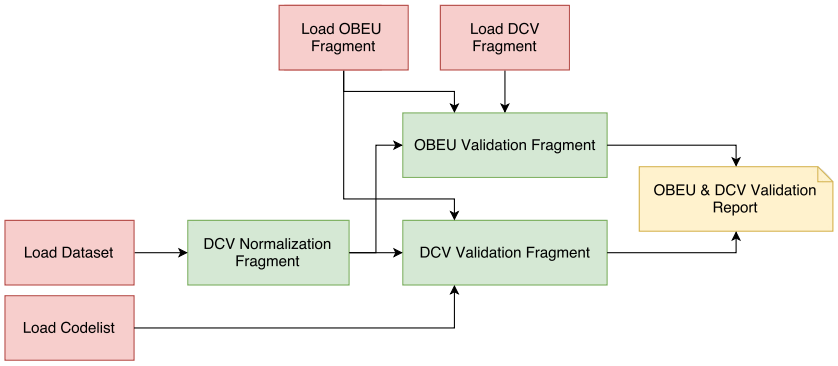

Fig. 2. Overview of the OBEU validation pipeline checking both the DCV and OBEU validity.

the normalized dataset. Finally, the resulting validation reports are combined to a joint output.

Iterating over this pipeline supports debugging of OBEU transformation pipelines and bug fixing in the corresponding datasets. An example of such a validation pipeline for the 2016 budget of the city of Bonn (Germany) is available on GitHub.[25] To reuse the pipeline for another dataset, only the URLs of the dataset to be validated need to be changed.

Thanks to their modularity, the validation and normalization pipelines are also being used for the FDP2RDF pipeline by simply linking them to its output: the pipeline fragment can be imported directly into the FDP2RDF pipeline and connected to its end, so that the validation is run every time the FDP2RDF pipeline runs. As the FDP2RDF pipeline is still in development, the validation pipelines are used mainly for checking the FDP2RDF pipeline itself and have already helped to discover and fix bugs.

4.2 Comparative analysis on normalized values

Several datasets have been transformed to the OBEU data model to be used in the OBEU project. They vary with respect to time, country, and also administrative level (i.e., local, regional, national, and EU level).

To perform a proper analysis of budget and spending datasets from different years and regions it is essential that the analysed monetary values are comparable. More concretely, the currency should be the same, the inflation rate has to be taken care of as well as different price levels among countries. Combining the pipeline fragment for normalizing monetary amounts with the pipeline fragment for DCV to CSV conversion, we are able to produce a CSV file with comparable amounts. This can be used as input for a comparative analysis across different regions or a time series analysis along different years with state-of-the-art analysis tools to address questions such as how Thessaloniki in Greece and Bonn in Germany budget their expenditures for public transportation or

[25] `https://github.com/openbudgets/datasets/blob/master/Bonn/pipelines/validation.jsonld`

to what extent the crisis in Greece has influenced the municipalities' budgets compared over time.

5 Conclusion

The need to transform fiscal or similar data from heterogeneous source formats to RDF is growing widespread and reusing the solutions for recurring data transformation tasks thus becomes vital. We proposed a technological solution for such reuse, based on the state-of-the-art LP-ETL tool, presented a concrete collection of reusable transformations and demonstrated it on use cases in the OBEU project. Pipelines combining reusable fragments of different functionality, ranging from validation through structural normalization to complete transformation between formats, have been composed. The utilization of those significantly reduces the efforts to perform repetitive steps required in the dataset transformation cycle.

Our future work will focus primarily on the application of the pipelines in the context of the OpenBudgets.eu use case while extending their collection further and testing to what degree it could be ported to other fields that exploit multidimensional data.

Acknowledgements: The presented research has been supported by the H2020 project no. 645833 (OpenBudgets.eu).

References

1. Cyganiak, R., Reynolds, D.: The RDF Data Cube Vocabulary. W3C recommendation, W3C (2014), https://www.w3.org/TR/vocab-data-cube/
2. Gearon, P., Passant, A., Polleres, A.: SPARQL 1.1 Update. W3C recommendation, W3C (2013), https://www.w3.org/TR/sparql11-update/
3. Hausenblas, M., Villazón-Terrazas, B., Cyganiak, R.: Data shapes and data transformations. Tech. rep. (2012), http://arxiv.org/abs/1211.1565
4. Klímek, J., Mynarz, J., Škoda, P., Zbranek, J., Zeman, V.: Deliverable 2.2: Data optimisation, enrichment, and preparation for analysis. Tech. rep. (2016), http://openbudgets.eu/assets/deliverables/D2.2.pdf
5. Klímek, J., Škoda, P., Nečaský, M.: LinkedPipes ETL: Evolved linked data preparation. In: The Semantic Web: ESWC 2016 Satellite Events - ESWC 2016 Satellite Events, Anissaras, Crete, Greece, May 29-June 2, 2016, Revised Selected Papers, to appear (2016)
6. Knap, T., Kukhar, M., Macháč, B., Škoda, P., Tomeš, J., Vojt, J.: UnifiedViews: An ETL framework for sustainable RDF data processing. In: ESWC (2014)
7. Roman, D., Dimitrov, M., Nikolov, N., Putlier, A., Sukhobok, D., Elvesæter, B., Berre, A.J., Ye, X., Simov, A., Petkov, Y.: Datagraft: Simplifying open data publishing. In: Posters & Demos of the 13th European Semantic Web Conference (2016), http://2016.eswc-conferences.org/sites/default/files/papers/Accepted%20Posters%20and%20Demos/ESWC2016_DEMO_DataGraft.pdf