OpenBudgets.eu: Fighting Corruption with Fiscal Transparency

Project Number: 645833 Start Date of Project: 01.05.2015 Duration: 30 months

Deliverable 2.2

Data optimisation, enrichment, and preparation for analysis

Dissemination Level	Public
Due Date of Deliverable	Month 12, 30.4.2016
Actual Submission Date	30.05.2016
Work Package	WP 2, Data Collection and Mining
Task	T 2.2
Type	Demonstrator
Approval Status	Final
Version	1.0
Number of Pages	17
Filename	D2.2 Data optimisation, enrichment, and preparation for analysis.docx

Abstract: The intermediate data produced using various ETL pipelines during their development may not be consistent with the RDF Data Cube Vocabulary and the OpenBudgets.eu data model. This is why we developed several techniques for data optimisation, detecting such inconsistencies and reporting them to the pipeline developers so that they can fix them. Once the data is in a consistent state, it may be enriched with other data using the Linked Data principles and finally prepared for further analysis using data mining tools. In this deliverable, we demonstrate our approach to these tasks using LinkedPipes ETL, which we use for data preparation in the project.

Project funded by the European Union's Horizon 2020 Research and Innovation Programme (2014 – 2020)

History

Version	Date	Reason	Revised by
0.1	16.05.2016	Version for internal review	Jindřich Mynarz
0.2	30.05.2016	Version for external review	Christiane Engles
1.0	30.05.2016	Final version for submission	Jakub Klímek

Author List

Organisation	Name	Contact Information
UEP	Jakub Klímek	klimek@opendata.cz
UEP	Jindřich Mynarz	mynarzjindrich@gmail.com
UEP	Petr Škoda	skodapetr@gmail.com
UEP	Jaroslav Zbranek	zbranek.jaroslav@gmail.com
UEP	Václav Zeman	prozeman@gmail.com

Executive Summary

This demonstrator deliverable contains a description of the extensions needed to implement various data optimisation, enrichment, and pre-processing tasks, which were developed for LinkedPipes ETL (LP-ETL), the ETL tool used in the OpenBudgets.eu project, and then a demonstration of processes implemented in LP-ETL, using these extensions. The data optimisation processes detect and report inconsistencies in data used in the project with the RDF Data Cube Vocabulary and the OpenBudgets.eu data model using six specific rules. The data enrichment processes demonstrate how data can be combined with other relevant data using the Linked Data principles to add significant value. Finally, the preparation for analysis process describes, how the RDF data used in the project is transformed to the less expressive CSV format required by many data mining tools, which are used to discover additional facts from the data. The created pipeline fragments are available in the repository at https://github.com/openbudgets/pipeline-fragments.

Abbreviations and Acronyms

CSV	Comma Separated Values
DCV	The RDF Data Cube Vocabulary
DSD	Data Structure Definition
ETL	Extract Transform Load
IRI	Internationalized Resource Identifier
LP-ETL	LinkedPipes ETL
NUTS	Nomenclature of Territorial Units for Statistics
RDF	Resource Description Framework
TSV	Tab-separated values

Table of Contents

1 Introduction

The well-known 80:20 rule of data science (Lohr, 2014) states that while analyses of data take 20 % of a data scientist's time, 80 % of the time is spent pre-processing data into a form suitable for analysis. In this deliverable we demonstrate several possible data pre-processing steps that can decrease the pre-processing effort needed to enable analysis of fiscal data in OpenBudgets.eu.

We first document several features of LinkedPipes ETL (LP-ETL), the ETL framework described in deliverable 2.1 (Engels et al., 2016), that were developed as prerequisites for data pre-processing tasks described further on. In order to streamline pre-processing of fiscal data we created reusable pipeline fragments for LP-ETL that automate common tasks of data optimisation, enrichment, and preparation for analysis. These tasks include validation of integrity constraints imposed by the Data Cube Vocabulary and the OpenBudgets.eu data model, value normalization of monetary amounts using enriched data, or pre-processing for data analysis via propositionalization of RDF data into a single CSV table. We start by describing the newly developed features of LP-ETL, using which the pipeline fragments described later are built.

2 Newly developed features of LinkedPipes ETL

LinkedPipes ETL is the data processing tool we use in OpenBudgets.eu to ingest fiscal data in various formats and transform it to RDF, modelled according to the data model documented in deliverable 1.4 (Dudáš et al., 2015). The tool itself was introduced in deliverable 2.1 and is further described on its web page.[1]

To be able to accomplish our goals in this deliverable, we needed to develop a few improvements to LP-ETL. The documentation here is nearly identical to the one we put in the LP-ETL web documentation.[2] First, we introduce a mechanism for development and sharing of pieces of ETL pipelines called pipeline fragments. Then we describe our new Mustache template component, which allows us to dynamically generate SPARQL queries based on the data flowing through the pipeline. In line with that goes the improvement of the existing SPARQL components giving them the ability to execute the dynamically generated queries. The final addition is the ability to share pipelines without login credentials, which are specific to the specific development environment.

2.1 Pipeline fragments

Support for reusable pipeline fragments was developed for LP-ETL. It enables developers to share and reuse parts of their ETL pipelines. LP-ETL pipelines are represented as RDF in JSON-LD, which is a text format that allows to publish the pipelines easily on the Web. Pipelines can be thus used as parts of documentation of the ETL processes and also as directly usable examples of individual component's usage. What a developer needs to do to reuse a published pipeline fragment is to import it from its URL either as a new pipeline, or as a part of an existing pipeline. This speeds up the development process in cases where similar pieces of pipelines need to be reused and flattens the learning curve both for experienced and novice developers when using new components. An example of a published pipeline fragment can be seen in the documentation of the Mustache component.[3]

[1] http://etl.linkedpipes.com

[2] http://etl.linkedpipes.com/documentation

[3] http://etl.linkedpipes.com/components/t-mustache

2.2 Mustache template component

We developed a component for LP-ETL implementing Mustache, a library for rendering text templates. First, we recommend the potential user of this component to get to know the library itself[4] with its demo.[5] There, a template and a sample JSON file with data are shown and the resulting text can be generated. In LP-ETL, we work with RDF instead of JSON, therefore, the template placeholders will be IRIs instead of JSON attributes and the template data will be stored as RDF. In our work presented in this deliverable, we use Mustache templates for generating SPARQL queries based on data and also for rendering HTML reports from RDF data.

2.2.1 Input data in JSON and in RDF

Below, you can see the original Mustache input data from the demo in JSON (Figure 1) and the same input data in RDF (Figure 2). This can be used for comparison, because the structure and meaning remains the same. Note that in Figure 2 the order of the items has to be specified explicitly, because the RDF data model is a set of triples without ordering, which is in contrast to the tree-shaped JSON.

```
{
  "header": "Colors",
  "items": [
      {"name": "red", "first": true, "url": "#Red"},
      {"name": "green", "link": true, "url": "#Green"},
      {"name": "blue", "link": true, "url": "#Blue"}
  ],
  "empty": false
}
```

Figure 1 - Example of Mustache input data in JSON

```
@prefix :         <http://localhost/ontology/> .
@prefix ex:       <http://example.com/> .
@prefix mustache: <http://plugins.linkedpipes.com/ontology/t-mustache#> .

ex:1 a :OutputClass ;
  :header "Colors" ;
  :items ex:1-1, ex:1-2, ex:1-3 ;
  :empty false ;
  mustache:fileName "file.html" .
```

[4] http://mustache.github.io

[5] http://mustache.github.io/#demo

```
ex:1-1 :name "blue" ;
  :link true ;
  :url "#Blue" ;
  mustache:order 3 .

ex:1-2 :name "red" ;
  :first true ;
  :url "#Red" ;
  mustache:order 1 .

ex:1-3 :name "green" ;
  :link true ;
  :url "#Green" ;
  mustache:order 2 .
```

Figure 2 - Example of Mustache input data in RDF (Turtle)

2.2.2 Templates for JSON and for RDF

Below, you can see the original Mustache template from the demo (Figure 3) for the JSON input data in Figure 1. For comparison, the template in Figure 4 is usable for RDF data from Figure 2. The structure of the data and the meaning of the Mustache constructs stays the same.

```
<h1>{{header}}</h1>
{{#bug}}
{{/bug}}

{{#items}}
  {{#first}}
    <li><strong>{{name}}</strong></li>
  {{/first}}
  {{#link}}
    <li><a href="{{url}}">{{name}}</a></li>
  {{/link}}
{{/items}}

{{#empty}}
  <p>The list is empty.</p>
{{/empty}}
```

Figure 3 - Example Mustache template for JSON

```
<h1>{{http://localhost/ontology/header}}</h1>
{{#bug}}
{{/bug}}

<ol>
{{#http://localhost/ontology/items}}
  {{#http://localhost/ontology/first}}
    <li><strong>{{http://localhost/ontology/name}}</strong></li>
  {{/http://localhost/ontology/first}}
  {{#http://localhost/ontology/link}}
    <li><a
href="{{http://localhost/ontology/url}}">{{http://localhost/ontology/nam
e}}</a></li>
  {{/http://localhost/ontology/link}}
{{/http://localhost/ontology/items}}
</ol>

{{#http://localhost/ontology/empty}}
  <p>The list is empty.</p>
{{/http://localhost/ontology/empty}}
```

Figure 4 - Example Mustache template for RDF usable in LP-ETL

2.2.3 Output file

Below in the Figure 5, the output file for our example is shown. It is a simple HTML markup, which can be generated both with the original Mustache JSON input data and JSON-based template, and the RDF input data and the new Mustache LP-ETL component.

```
<h1>Colors</h1>


<ol>
  <li><strong>red</strong></li>
  <li><a href="#Green">green</a></li>
   <li><a href="#Blue">blue</a></li>
</ol>
```

Figure 5 - Example Mustache output

2.2.4 LP-ETL template specifics

The component looks in the input data for the instances of the output class specified in the configuration and executes the template on each one. In our example, the entity class was `http://localhost/ontology/OutputClass`. Note that only literals can be used by the template because IRIs are used to connect one object to another. In the case that you need to output the IRI itself, you can generate its literal version, e.g., using the SPARQL update component[6] before passing the data to the Mustache component.

[6] http://etl.linkedpipes.com/components/t-sparqlupdate

In addition to the standard Mustache, LP-ETL templates support special properties that can further customize the outputs.

The output file name property, denoted by the `http://plugins.linkedpipes.com/ontology/t-mustache#fileName` IRI, which can be attached to the entity class instance, can be used to generate a different file for each entity class instance.

One of the key Mustache concepts is a list of items. In JSON, the order of the items is given implicitly, because JSON is a tree and therefore each node has an ordered sequence of children. In RDF, the order of the list items needs to be specified explicitly, because the RDF data model is a generic graph. We could have used the RDF list (i.e. `rdf:Seq`), however, it seemed that the explicit order will be more usable. The property is denoted by the `http://plugins.linkedpipes.com/ontology/t-mustache#order` property that can be attached to a list item.

A sample pipeline fragment showing the component usage in LP-ETL is available.[7]

2.3 Configurable SPARQL components

The SPARQL query handling components of LP-ETL were improved so that they accept runtime configuration via RDF. This means that the developers can generate the SPARQL queries to be executed using other LP-ETL components based on input data, which improves the capabilities of LP-ETL pipelines. Specifically, this feature is used together with the Mustache component, e.g., to transform RDF data to CSV using a SPARQL SELECT query generated based on a Data Cube Vocabulary (DCV) data structure definition. Pipeline fragments described in the following sections use these components for adaptive SPARQL queries based on input data. In Figure 6, a simple example of the SPARQL CONSTRUCT component configuration is given. It is made of two triples, one stating the resource type and the other containing the query.

```
@prefix sc: <http://plugins.linkedpipes.com/ontology/t-sparqlConstruct#>
.

<http://localhost/resources/configuration> a sc:Configuration ;
    sc:query "CONSTRUCT { ?s ?p ?o } WHERE { ?s ?p ?o}" .
```

Figure 6 - Sample SPARQL Construct RDF configuration

2.4 Removing credentials for sharing LP-ETL pipelines

A common problem identified when sharing LP-ETL pipelines was that environment-specific, and therefore sensitive credentials for handling data were shared along with the pipeline and had to be removed manually. This process was tedious, especially for frequently updated pipelines. Therefore, we implemented an extension to LP-ETL which adds an option to remove these credentials automatically when downloading a pipeline. This was done by tagging component configuration fields, which can potentially contain such sensitive information. These may include the username and password, but also the hostname and port number of the RDF stores, and local paths to files on servers. The configuration in these fields is then removed during download, so that the resulting file can be shared safely.

[7] http://etl.linkedpipes.com/assets/pipelines/t-mustache-1.jsonld

3 Optimisation

We developed pipeline fragments for data normalization and validation. The validation pipeline fragments test if datasets conform to the Data Cube Vocabulary and the OpenBudgets.eu data model. The detected errors are reported to users who can proceed to fix them, typically by correcting the ETL pipelines for processing data or by extending the pipelines with additional transformations.

3.1 DCV normalization

The normal form of DCV data[8] has all component properties attached on the level of observations. When data adheres to the normal form, its regularity allows to simplify queries on the data. The specification of DCV provides a normalization algorithm implemented as a series of SPARQL Update operations.[9] We used these operations to develop a pipeline fragment[10] that transforms DCV data into the normal form. DCV normalization is used as a pre-processing step for all of the following pipeline fragments.

3.2 DCV validation

Data optimisation must be preceded by data validation that discovers problems that need to be fixed. Besides syntax validation, RDF allows to validate semantics given by the vocabularies used to describe data. The data model of OpenBudgets.eu (Dudáš et al., 2015) is based on the Data Cube Vocabulary (DCV). Consequently, there are 2 principal sources of integrity constraints that the data processed in OpenBudgets.eu needs to conform to. There are specific constraints for budget and spending data defined by the OpenBudgets.eu data model and generic constraints that all DCV datasets must adhere to.

DCV defines 21 integrity constraints as a part of its specification.[11] The constraints are formulated as SPARQL ASK queries that evaluate to `true` if a constraint violation is found. Use of SPARQL makes the constraints easy to implement. For example, the SPARQL rules are used in the Data Cube Validator.[12] Alternatively, the NoSPA-RDF-Data-Cube-Validator[13] avoids SPARQL to achieve performance gains, and instead implements the integrity constraints directly in Java. In particular, the improved runtime is apparent in the integrity constraint 12[14] that checks for duplicate observations. However, the validator produces results formatted in Markdown, which is suitable for reading by humans, but not for further automated processing.

DCV integrity constraints expect the datasets to be validated to be in the normal form described above and available in the default graph of the queried RDF store. In order to transform a dataset to be validated into the normal form required by the validation, the DCV normalization pipeline fragments that we described above can be used. In addition to the dataset the default graph must contain its data structure definition (DSD), definitions of dimensions used in the DSD, and code lists used by coded properties in the DSD. Since DSDs may include reused

[8] https://www.w3.org/TR/vocab-data-cube/#h2_normalize

[9] https://www.w3.org/TR/vocab-data-cube/#normalize-algorithm

[10] https://github.com/openbudgets/pipeline-fragments/tree/master/dcv/dcv-normalization

[11] https://www.w3.org/TR/vocab-data-cube/#wf-rules

[12] Source code of the Data Cube Validator is bundled with the validator for the Organization Ontology in https://github.com/epimorphics/org-verification.

[13] https://github.com/yyz1989/NoSPA-RDF-Data-Cube-Validator

[14] https://www.w3.org/TR/vocab-data-cube/#ic-12

component properties, gathering data for validation requires either to collect descriptions of the properties manually or dereference their IRIs and harvest their descriptions automatically.

We developed a pipeline fragment for DCV validation.[15] The required input of the validation is an RDF dataset that contains its DSD, descriptions of the DSD's components, and their code lists. These requirements are tested via a SPARQL ASK component that stops the pipeline execution and raises an error if the constraints are not satisfied. The pipeline then executes the DCV integrity constraints on the provided dataset. The constraints are reformulated from SPARQL ASK queries in the DCV specification to SPARQL CONSTRUCT queries that output descriptions of the detected constraint violations. We implemented the integrity constraint 12 by generating its query based on the dimensions in the dataset's DSD. Compared with the constraint's query defined in the DCV specification, the generated query achieves approximately 100× speed-up. The violations are described using the SPIN vocabulary.[16] Unlike simple SPARQL ASK queries that have boolean results SPIN vocabulary allows to describe constraint violations and thus provide helpful information to users who can fix the invalid data. Validation results are thus available in RDF, which makes them amenable to further processing. Additionally, the validation results are rendered into an HTML report via the Mustache component for their quick visual inspection by users.

3.3 OpenBudgets.eu data model validation

The data model of OpenBudgets.eu imposes additional constraints on top of DCV. Correctly modelled OpenBudgets.eu datasets adhere to the data model as described in (Dudáš et al., 2015). Testing compliance with the OpenBudgets.eu data model can be automated to some degree as we show in this deliverable. However, there are many constraints that either cannot be validated automatically or the effort to do so is greater than the gains from automated validation (e.g., infrequent issue). We observed common errors appearing in the datasets modelled using the OpenBudgets.eu data model. To address these errors we developed several SPARQL rules that are able to detect some of the common errors. These rules mostly test the assumptions about the DSDs of OpenBudgets.eu datasets. Testing compliance between observations and DSDs is covered by generic DCV validation. In total, six validation rules were implemented:

1. **Redefinition of component property's code list:** This rule detects if the validated dataset redefines a code list for a coded component property from the core OpenBudgets.eu data model. Instead of defining a different code list a subproperty of the core component property should be derived and used with a specific code list.
2. **Hijacked[17] core namespace:** This rule tests if the dataset to be validated defines a term in the namespace of the core OpenBudgets.eu data model (i.e. `http://data.openbudgets.eu/ontology/`) that is not defined by the core data model itself. The core namespace should be used only for the terms in the core OpenBudgets.eu data model. New terms should be defined in a different namespace.
3. **Missing mandatory component property:** Each dataset that adheres to the OpenBudgets.eu data model must contain the following component properties (or their subproperties): `obeu-attribute:currency`, `obeu-dimension:fiscalPeriod`, `obeu-dimension:operationCharacter`, `obeu-dimension:organization`, and `obeu-measure:amount`. This rule tests if these properties are explicitly provided in the dataset's DSD.

[15] https://github.com/openbudgets/pipeline-fragments/tree/master/dcv/dcv-validation

[16] http://spinrdf.org/spin.html

[17] https://www.w3.org/wiki/NamespaceHijacking

4. **Property instantiation:** We discovered that it is a common error to instantiate an RDF property. RDF only allows to instantiate classes, so instantiating properties is incorrect. We assume this error may be often caused by typos in IRIs of classes that differ from property IRIs only in character case (e.g., `qb:DataSet` and `qb:dataSet`).
5. **Use of abstract property:** Several properties in the core OpenBudgets.eu data model are defined as abstract (e.g., `obeu-dimension:classification`); i.e. they should not be used directly and instead their subproperties should be minted. This rule test if the dataset to be validated is free from use of these abstract properties.
6. **Wrong character case in DCV:** This rule detects if non-existent terms from the DCV namespace that differ only in character case from the terms in this namespace are used. Besides reporting the non-existent DCV term, the rule suggests an existing term with correct character case that may be used instead

The pipeline fragment[18] we developed for validation of OpenBudgets.eu data model requires the data model of OpenBudgets.eu to be available in a specific named graph. In order to automate population of this graph we created a pipeline[19] that merges all data about the data model. Besides the OpenBudgets.eu data model it requires the description of the DCV to be in the named graph `http://purl.org/linked-data/cube`. We prepared a pipeline for loading DCV too.[20] As is the case for DCV validation, the dataset to be validated must be in the normal form.

LP-ETL does not support named graphs for internal data storage, which complicates the implementation of this pipeline fragment. As a work-around we load the validated dataset into an external RDF store and execute the validation rules using SPARQL extractor component on this endpoint. When the validation is finished, the pipeline automatically cleans the validated dataset from the RDF store. Similarly to the DCV validation, the validation results are available both in RDF described using the SPIN vocabulary and in an HTML report that is better suitable for human users.

4 Enrichment

Following the linked data approach, the way to solve problems is often to add more data. This is also the case in analyses of fiscal data that frequently require additional data. In particular, data enrichment is regularly needed for comparative analysis. Absolute values can be compared directly only in some cases, but comparison becomes feasible if we convert the absolute values to relative values using contextual data, such as population counts to compute per capita spending, that we add during data enrichment. In this deliverable, we show an enrichment that can be used to normalize monetary values. We also suggest several ways in which fiscal data can be enriched with data describing its context.

Linking is a necessary prerequisite for enrichment. In our previous work, we demonstrated the benefits of linking code lists for enrichment of fiscal data (Ioannidis, 2016). To ease linking of large datasets, we developed a patch for the Silk link discovery framework that we describe further on.

4.1 Value normalization of monetary amounts

A basic requirement for comparative analysis of fiscal data is being able to compare monetary amounts in terms of their value. Instead of comparing nominal values of the amounts, we often

[18] https://github.com/openbudgets/pipeline-fragments/tree/master/obeu/obeu-model-integrity-constraints

[19] https://github.com/openbudgets/pipeline-fragments/tree/master/obeu/load-obeu

[20] https://github.com/openbudgets/pipeline-fragments/tree/master/dcv/load-dcv

need to compare their real values. Suppose you have two amounts from different times, such as different fiscal years, and places, such as different EU member states. As European System of Accounts 2010 says: *"The fact that countries have different price levels and currencies poses a challenge to interspatial comparisons of prices and volumes"* (European Union, 2013, p. 303). How can you tell which of the amounts has a greater value?

When it comes to money, space corresponds to currency and time corresponds to changes in the price level. In order to normalize currency it can be converted to a single currency, such as euro. Currency can be converted using exchange rates. Values can be adjusted for changing price levels by using price indices. Value normalization thus requires enrichment with two datasets, one about exchange rates, the other about a selected price index. Exchange rates for national currencies of EU member states averaged over a year are provided by Eurostat.[21] In order to normalize price levels, several coefficients can be used, such as implicit deflator based on gross domestic product (GDP), which is a measure of price level change with respect to a specific base year. Eurostat provides implicit deflators in euro for the EU member states based in every 5th year (e.g., 2005, 2010 etc.).[22]

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 expended, and$E_t$is the exchange rate to euro for the target year:

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

Equation 1: Value normalization using deflators and exchange rate

The target year should be chosen as the most recent year in the normalized data. In our case, the employed price index in the implicit deflator. Deflator's base year should be chosen as the nearest preceding year to the target year. For example, if the amounts to be normalized are from 2008 and 2014 and deflator's base years are 2005, 2010, and 2015, we choose 2010. Eurostat offers the deflators either in national currencies or in euro. We chose to use deflators in national currencies because it is a generically applicable method. However, it is also possible to start by converting currency to euro and then apply a deflator in euro, which gives almost identical results if one of the deflated currencies is euro.

In order to implement the normalization we reused the above-mentioned Eurostat datasets converted to RDF and modelled using DCV by the LATC project.[23] An issue of the data is that it represents measures as strings instead of numbers. Moreover, as is often the case with EU-funded projects, the LATC project ended and the data was not updated since 2014, so deflators are available only up to 2013. Another minor issue of the deflators datasets is the IRIs of the code list concepts for price indices lack description, so we can only guess their semantics from their IRIs. For example, `http://eurostat.linked-statistics.org/dic/unit#PD10_NAC` is *Price index (implicit deflator), 2010=100, national currency*.

We developed a pipeline fragment[24] that produces normalized monetary measures for input fiscal datasets. The pipeline requires input data to be in the DCV normal form. Moreover, we encountered an issue that makes the pipeline require manual configuration. Country is often not explicit in data, but instead it is implicitly linked via the organization that spent the normalized amount. In most datasets, there is a single organization operating in a single country, so it is possible to provide the country manually. In cross-country datasets links to

[21] http://ec.europa.eu/eurostat/web/products-datasets/-/tec00033

[22] http://ec.europa.eu/eurostat/web/products-datasets/-/nama_10_gdp

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

[24] https://github.com/openbudgets/pipeline-fragments/tree/master/monetary_value_normalization

countries need to be provided explicitly in data and the pipeline fragment must be amended. Since there are multiple ways to represent years in fiscal data, the pipeline is limited to support only two common ways found in OpenBudgets.eu datasets (via `obeu-dimension:fiscalYear` and `obeu-dimension:date` and the subproperties thereof).

4.2 Contextual normalization

Absolute monetary amounts can be converted to relative amounts by using data about the context in which the amounts are spent. For example, population counts can be used to compute per capita spending. We can enrich budget and spending data with data about the area in which it is spent. The typical contextual data about administrative areas includes demographic statistics and geospatial data. A prime source of such data in the context of the European Union is Eurostat.[25] For each NUTS region[26] it offers population counts,[27] densities,[28] or area sizes.[29] While Eurostat provides the data in TSV format, the LATC project exposed the data in RDF using DCV.[30] Alternatively, higher-level indicators can be used to relate monetary amounts to relevant objectives, such as unemployment rate. For example, relevant indicators can be found in data by OECD,[31] such as annual average wages. Such contextual data is typically available for administrative areas, such as NUTS regions, which can be linked straightforwardly via their codes.

4.3 Patch for Silk link discovery framework

In order to enable interlinking large datasets available via SPARQL endpoints we implemented a patch for the Silk link discovery framework.[32] The patch fixes a common problem that Silk has with loading large datasets that need to be split into multiple chunks. The paging implemented by Silk required the queried RDF stores to sort the whole result set for each request, which is a computationally demanding operation that either slows down query execution or makes the RDF store reject the query completely. We leveraged the scrollable cursors functionality exposed by the Virtuoso RDF store that enables to prevent these effects by using a nested query. The patch was submitted to the Silk code repository (https://github.com/silk-framework/silk/pull/59), but to the date of writing has not been merged in the repository.

5 Preparation for analysis

The data model of OpenBudgets.eu is based on RDF, but RDF cannot be directly processed by many analytical tools. Instead, CSV appears to be the lowest common denominator of data that is accepted by most tools. Therefore, we developed an automated conversion from RDF to CSV to ease processing the OpenBudgets.eu data in analytical tools.

[25] http://ec.europa.eu/eurostat

[26] http://ec.europa.eu/eurostat/web/nuts/overview

[27] http://ec.europa.eu/eurostat/web/products-datasets/-/demo_r_pjanaggr3

[28] http://ec.europa.eu/eurostat/web/products-datasets/-/demo_r_d3dens

[29] http://ec.europa.eu/eurostat/web/products-datasets/-/demo_r_d3area

[30] http://eurostat.linked-statistics.org

[31] http://oecd.270a.info

[32] http://silkframework.org

5.1 Conversion of DCV to CSV

A common feature of data mining tools is that they require data in a single table with a set of propositions in the form of attribute-value pairs (Lachiche, 2013). The tabular data model expected by these tools can be serialized in CSV. Data model used by OpenBudgets.eu is based on RDF and is consequently more expressive than CSV can be. Nevertheless, the model is constrained by the Data Cube Vocabulary (DCV). DCV constrains dimensions and measures to 1..1 cardinality and attributes to 0..1 cardinality. Absence of multi-valued properties makes it considerably simpler to produce a single table out of a DCV dataset than out of arbitrary RDF data. Columns of the output CSV can be derived from the components explicitly described in a dataset's DSD. Graph-shaped RDF can be transformed into the tabular format using SPARQL SELECT queries (Hausenblas et al., 2012). Finally, results of SPARQL SELECT queries can be serialized directly into CSV.

We developed a pipeline fragment for LP-ETL[33] that transforms DCV data into CSV. Its input consists of a DCV dataset in RDF with its DSD and its output is the dataset in CSV. The pipeline fragment extracts data about dataset's components from its DSD, including the components' type, attachment, and order. The extracted data is used to render a Mustache template that generates a SPARQL SELECT query to transform the dataset from RDF to CSV.

he developed transformation has several limitations. In order to produce readable column names in the CSV output, it assumes that local names of the components properties used in a DSD are unique. If this assumption is broken, hashed IRIs (e.g., MD5 hashes) of the component properties can be appended to their local names to make them unique. The transformation does not support components attached to measure properties. Nevertheless, if component properties are attached to a measure property, they become its integral part, so their semantics is embodied in the measure. Additionally, such components can have only one value per dataset, so they give no distinguishing power to data mining.

6 Conclusion

We created data pre-processing pipeline fragments that automate several common tasks of optimisation, enrichment, and preparation for analysis. Reusability of the fragments is based on shared data model of DCV and the OpenBudgets.eu data model they operate on. The DCV basis of the OpenBudgets.eu data model allowed us to leverage standard operations specified for DCV data, such as normalization or validation of integrity constraints. We demonstrated the benefit of data enrichment on the conversion nominal monetary values to real values using exchange rates and price indices, which is a common task preceding comparative analyses. We aimed for maximum reuse also in the case of preparation for data analysis, for which we created a pipeline fragment that transforms RDF DCV data into CSV, a typical format used by data mining tools.

The developed pipeline fragments will be used in the following data analyses in OpenBudgets.eu. They are available from the repository at https://github.com/openbudgets/pipeline-fragments. Wider application of the fragments will provide feedback on how to improve their usability and robustness. Data pre-processing work will continue with development of additional pipeline fragments as required by the OpenBudgets.eu use cases.

7 References

- Dudáš, M.; Horáková, L.; Klímek J., Kučera J., Mynarz J., Sedmihradská L., Zbranek J. (2015): *OpenBudgets.eu - Deliverable D1.4 - User documentation*, 2015, http://openbudgets.eu/assets/deliverables/D1.4.pdf.

[33] https://github.com/openbudgets/pipeline-fragments/tree/master/dcv/dcv-to-csv

- Engels, C.; Musyaffa, F; Dong, T.; Klímek, J.; Mynarz, J.; Orlandi, F.; Auer, S. (2016): *Deliverable 2.1 - Tools for semantic lifting of multiformat budgetary data*. http://openbudgets.eu/assets/deliverables/D2.1.pdf
- European Union (2013): *European System of Accounts 2010.* http://ec.europa.eu/eurostat/documents/3859598/5925693/KS-02-13-269-EN.PDF/44cd9d01-bc64-40e5-bd40-d17df0c69334. ISBN 978-92-79-31242-7. DOI 10.2785/16644.
- Hausenblas, M., Villazón-Terrazas, B., Cyganiak, R. (2012): *Data Shapes and Data Transformations*. CoRR. http://arxiv.org/abs/1211.1565.
- Ioannidis, L.; Klímek. J.; Musyaffa, F.; Mynarz, J.; Sedmihradská, L.; Zbranek, J. (2016): *OpenBudgets.eu - Deliverable D1.9 - Linking code lists to external datasets*. http://openbudgets.eu/assets/deliverables/D1.9.pdf
- Lachiche, N. (2013): Propositionalization. *Encyclopedia of Machine Learning.* Springer. http://link.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_680.
- Lohr, S. (2014): For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights. *New York Times*. http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html