Application of the Linked Data Visualization Model on Real World Data from the Czech LOD Cloud

Jakub Klímek
Czech Technical University in Prague
Faculty of Information Technology
klimek@fit.cvut.cz

Jiří Helmich
Charles University in Prague
Faculty of Mathematics and Physics
helmich@ksi.mff.cuni.cz
http://xrg.cz/

Martin Nečaský
Charles University in Prague
Faculty of Mathematics and Physics
necasky@ksi.mff.cuni.cz
http://xrg.cz/

ABSTRACT

In the recent years the Linked Open Data phenomenon has gained a substantial traction. This has lead to a vast amount of data being available on the Web in what is known as the LOD cloud. While the potential of this linked data space is huge, it fails to reach the non-expert users so far. At the same time there is even larger amount of data that is so far not open yet, often because its owners are not convinced of its usefulness. In this paper we refine our Linked Data Visualization Model (LDVM) and show its application via its implementation Payola. On a real-world scenario built on real-world Linked Open Data created from Czech open data sources we show how end-user friendly visualizations can be easily achieved. Our first goal is to show that using Payola, existing Linked Open Data can be easily mashed up and visualized using an extensible library of analyzers, transformers and visualizers. Our second goal is to give potential publishers of (Linked) Open Data a proof that simply by publishing their data in a right way can bring them powerful visualizations at virtually no additional cost.

Categories and Subject Descriptors

H.5.2 [**User interfaces**]: GUIs, Interaction styles; H.3.5 [**Online Information Services**]: Data sharing; H.3.5 [**Online Information Services**]: Web-based services

Keywords

Linked Data, Visualization, Semantic Web

1. INTRODUCTION

Recently, vast amount of data represented in a form of Linked Open Data (LOD) has appeared on the Web. The key LOD principle is linking data entities in a machine interpretable way so that they form a huge data space distributed across the Web: the *LOD cloud*. The LOD cloud is not interesting for end-users until there are useful tools available built on top of it. Very important are tools which are able to present various kinds of LOD to users who want to explore the data. This includes LOD browsers and visualization tools. An end-user often does not know more about datasets than that there are some data structures contained which could be visualized. For example, entities in a dataset can be geocoded. In that case, the user may require to start the exploration on a visualization of those entities on a map. Or the dataset may contain a hierarchical structure and various techniques for hierarchy visualizations can be used.

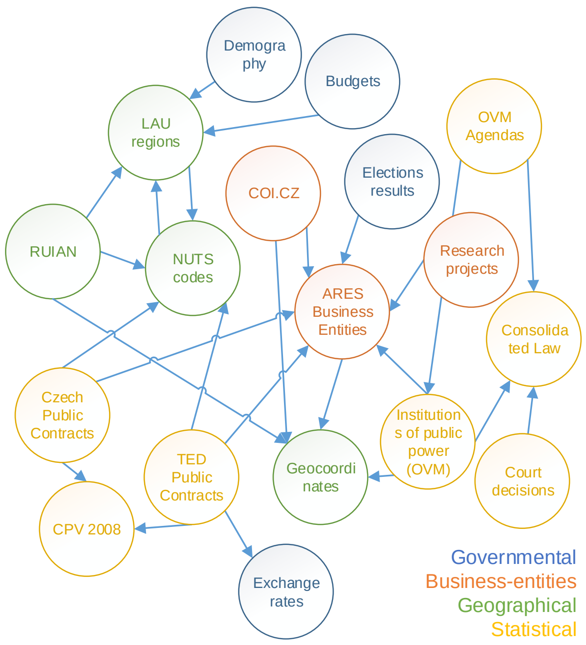


Figure 1: Czech Linked Open Data (CzLOD) Cloud

The reader may argue that this does not depend on the fact that we work with LOD, which is true. Any non-LOD dataset can also be explored this way. However, LOD brings an important new dimension (besides the uniform data model – RDF) to the problem of data presentation, especially when we talk about visualization. Suppose that we have a dataset with addresses, we geocoded them and display them on a map. Suppose now that this is a LOD dataset and that we have other LOD datasets without GPS coordinates, but linked to the geocoded entities of the former one. We can build a tool which displays any of these linked datasets on a map as well. This, of course, applies only when the links make sense in terms of location, but the point is that compared to non-LOD datasets, it is easy to create and use links in LOD.

In our previous work, we presented the *Linked Data Visualisation Model (LDVM)* [4]. It enables us to combine various LOD datasets and visualize them with various kinds of visualizations. It separates the part which prepares the data for visualizations from the actual visualizers. Visualizers then specify their expected input data structure (e.g., hierarchical with labels, geo-coordinates, etc.) in RDF using broadly accepted vocabularies. This allows reuse of visualizers for a broad range of LOD datasets. We focus on two groups of users and their cooperation. First are expert users who can easily prepare analyses and visualization components and the second group are lay users. They can use and combine components prepared for them by the experts using a LDVM implementation without extensive knowledge of RDF and SPARQL. An example of such a use by a lay user can be visualizing a given analysis using various visualizers or running the same analysis on various data sources.

Contributions. The primary purpose of this paper is to demonstrate the benefits that LDVM brings to users on real-world data. We show that our implementation *Payola* allows any expert user to easily access his SPARQL endpoint of choice or upload his RDF file, perform analyses using SPARQL queries and visualize the results using a library of visualizers. At the same time, the components created by experts can be also easily used and reused by lay users. We present several visualization scenarios of real-world data from the *Czech LOD (CzLOD) cloud*. The Czech LOD cloud contains various datasets we publish in our research group. We describe these datasets briefly in this paper as well. Each scenario takes several datasets from the CzLOD cloud, combines them together, extracts a data structure which can be visualized and offers appropriate visualizers to the user.

Outline. The rest of this article is organized as follows: In Section 2 we survey the CzLOD cloud and describe the currently available datasets. In Section 3 we present the Linked Data Visualization Model (LDVM), a simple yet powerful formalism for building analyses, transformations and visualizations of Linked Data. In Section 4 we briefly describe Payola, our implementation of LDVM. In Section 5 we present our real world examples of analyzers, transformers and visualizers on our running LDVM instance and put our contributions in a perspective of publishing data of public administration bodies. In Section 6 we briefly survey related work and finally, in Section 7 we conclude.

2. CZECH LOD CLOUD

In this section, we introduce a survey of the Czech Linked Open Data (CzLOD) cloud. As the data itself is not the main contribution of this paper, we will not go into too much detail. We are working on the cloud continuously in the OpenData.cz initiative since 2012 to show the owners of the data, who are mainly public bodies, what benefits can be gained from proper publication. The cloud is accessible at `http://linked.opendata.cz/sparql` and runs on OpenLink Virtuoso 7 Open-Source triplestore [1] and currently contains approximately 100 million triples not counting the largest dataset, RUIAN, which is described later. Figure 1 contains a map of the CzLOD cloud similar to the global LOD cloud. It is also color coded, red are the datasets about Czech business entities, green are the geographical datasets, yellow are the governmental datasets and blue are the statistical datasets. In addition, the CzLOD cloud also includes various e-Health datasets, which are, however, beyond the scope of this paper.

Business entities datasets. In the heart of the CzLOD cloud is the *ARES* dataset. It is data from various Czech registries and mainly the Business registry. In the Czech Republic, every business entity has its unique 8-digit identification number. Based on this number, it is easy to devise a rule for unique business entity URI creation. For example, the Czech Ministry of Interior is identified by `http://linked.opendata.cz/resource/business-entity/CZ00007064`. For each business entity in the Business registry, the dataset contains its official name, type, address of headquarters, kinds of its activities, names of shareholders, etc.

Another dataset about business entities is *COI.CZ*, which contains data about inspections, resulting fines, and bans issued by the Czech Trade Inspection Agency. Each inspection record contains the business entity identification number, location, region (NUTS code and LAU code) and information about the resulting fine or ban. Again, this data links easily to our other datasets about business entities via the URL based on the identification number. The source data is published as 3 star open data[2] (CSV files) by the agency[3].

Our *Research projects* dataset contains information about research grants funded by various Czech grant agencies. For each project there is data about amounts of money paid to each of the participants for each year of the project as well as identification numbers of all participants and additional information about the projects. The source data can be exported as Excel files from a web portal maintained by the Research, Development and Innovation Council [4].

Geographical datasets. Our newest and biggest geographical dataset is *RUIAN* - register of territorial identification, addresses and real estates. It has more than 600 million triples and contains data about all address points, streets, parts of towns, towns, cities, various levels of regions and also about every building and every lot in the Czech Republic including the hierarchy. Each object in RUIAN has assigned geocoordinates, which can be transformed to GPS coordinates. This creates a powerful base for geocoding in the Czech Republic. RUIAN is also linked to NUTS and LAU codes, which are 5-level European codes for towns, regions etc. The source data is in XML and freely accessible[5]. Other geographical datasets contain the already mentioned NUTS and LAU codes hierarchies. Additionally, the (*Geocoordinates*) dataset contains geocoordinates for each address found in our datasets created by geocoding.

Governmental datasets. Currently, there are three kinds of governmental datasets in the CzLOD cloud. The first kind contains information about institutions of public power (*OVM*), e.g., ministries, cities, but even public notaries, etc. For each institution, that is also a business entity, there is its identification number, address, type and also information about its offices and their opening hours. In addition, there is a dataset with *agendas* of these institutions, that are also linked to *laws* according to which they are established. This data gives a good base for, e.g., mobile applications that give the user his location and opening hours of the nearest notary, etc. The second kind of our governmental datasets

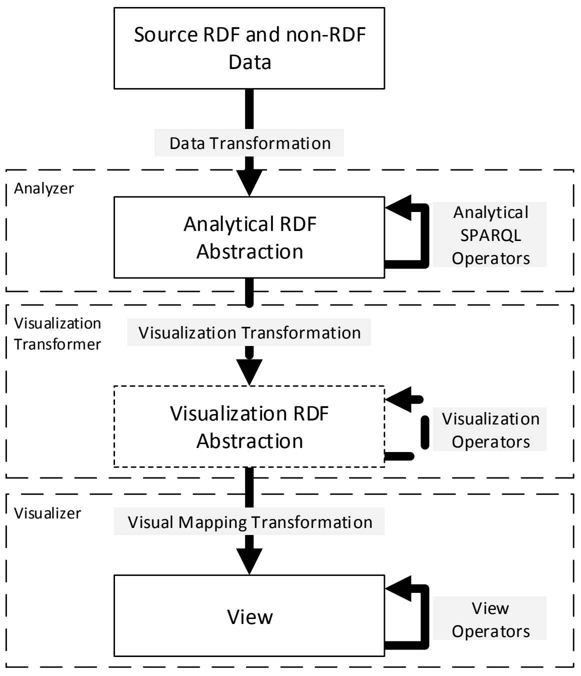
[1] `https://github.com/openlink/virtuoso-opensource`

[2] `http://5stardata.info/`

[3] `http://www.coi.cz/cz/spotrebitel/open-data-databaze-kontrol-sankci-a-zakazu/`

[4] `http://www.isvav.cz`

[5] `http://vdp.cuzk.cz/`



Source RDF and non-RDF Data
Data Transformation
Analyzer
Analytical RDF Abstraction
Analytical SPARQL Operators
Visualization Transformer
Visualization Transformation
Visualization RDF Abstraction
Visualization Operators
Visualizer
Visual Mapping Transformation
View
View Operators

Figure 2: High level LDVM overview.

are *law* datasets. The main part consists of *consolidated laws* of the Czech Republic. The other part consists of *decisions of Czech Supreme court* linked to laws. In addition, there are datasets with information about *public contracts*.

Statistical datasets. Our statistical datasets include data about demography and budgets of cities linked to the cloud via NUTS and LAU codes. We also have exchange rates of all currencies to Euro from the European Central Bank. Finally, there are results of elections to the Czech parliament.

3. LINKED DATA VISUALIZATION MODEL

In this section we briefly go through the Linked Data Visualization Model (LDVM), which we defined in our previous work [4]. First, we give an overview of the model and then we formalize its key elements.

3.1 Overview of LDVM

The Linked Data Visualization Model (LDVM) is an adaptation of the general *Data State Reference Model* (DSRM) [5] for the specifics of the visualization of RDF and Linked Data. It is an abstract data process inspired by a typical Knowledge Discovery Process [10]. We extend DSRM with three additional concepts – *analyzers*, *transformers* and *visualizers*. They denote reusable software components that can be chained to form an LDVM instance. Figure 2 shows an overview of the LDVM. The names of the stages, transformations and operators proposed by DSRM have been slightly adapted to the context of RDF and Linked Data. LDVM resembles a pipeline starting with raw source data (not necessarily RDF) and results with a visualization of the source data. It is organized into 4 stages that source data needs to pass through:

1. *Source RDF and non-RDF data:* raw data that can be RDF or adhering to other data models and formats (e.g. XML, CSV) as well as semi-structured or even non-structured data (e.g. HTML pages or raw text).
2. *Analytical abstraction:* extraction and representation of relevant data in RDF obtained from source data.
3. *Visualization abstraction:* preparation of an RDF data structure required by a particular visualization technique (e.g., 1D, 2D, 3D or multi-dimensional data, tree data, etc.)
4. *View:* creation of a visualization for the end user.

Data is propagated through the LDVM pipeline by applying 3 types of transformation operators:

1. *Data transformation:* transforms the raw data represented in a source data model or format into a representation in the RDF data model; the result forms the base for creating the analytical RDF abstraction.
2. *Visualization transformation:* transforms the obtained analytical abstraction into a visualization abstraction.
3. *Visual mapping transformation:* maps the visualization abstraction data structure to a concrete visual structure on the screen using a particular visualization technique specified using a set of parameters.

There are operators within the stages that allow for in-stage data transformations:

1. *Analytical SPARQL operators:* transform the output of the data transformation to the final analytical abstraction (e.g. aggregations, enrichment from LOD).
2. *Visualization operators:* further refine the visualization abstraction data structure (e.g., its condensation if it is too large for a clear visualization).
3. *View operators:* allow a user to interact with the view (e.g., rotate, scale, zoom, etc.).

3.2 LDVM stages

Source RDF and non-RDF Data Stage. The first stage considers RDF as well as non-RDF data sources. The data transformation transforms the source data to an RDF representation that forms a base for creating an analytical abstraction. If the source RDF data does not have a suitable structure for the following analysis, the transformation can be a sequence of one or more SPARQL queries that map the source data to the required structure.

Analytical RDF Abstraction Stage. The output of the second stage (*analytical RDF abstraction*) is produced by applying a sequence of various analytical SPARQL operators on the RDF output produced by the data transformation. We call the sequence an *analyzer* (see Figure 2). Our goal is to enable users to reuse existing analyzers for analyzing various datasets. We want to enable users to find analyzers that can be applied for analyzing a given data set and, vice versa, to find datasets that may be analyzed by a given analyzer automatically. Therefore, it is necessary to be able to decide whether an analyzer can be applied on a given dataset, i.e. whether the analyzer is *compatible* with the dataset. We formalize the notion of compatibility later in Section 3.3.

Visualization Abstraction Stage. We want to ensure that visualizers are reusable for different analytical abstractions. However, building specific visualizers for particular analytical abstractions would not enable such reuse. This is because each visualization tool visualizes particular generic characteristics captured by analytical abstractions. For example, there can be a visualizer of tree structures using the TreeMap technique or another visualizer of the same structures using the SunBurst technique. And, another visualizer may visu-

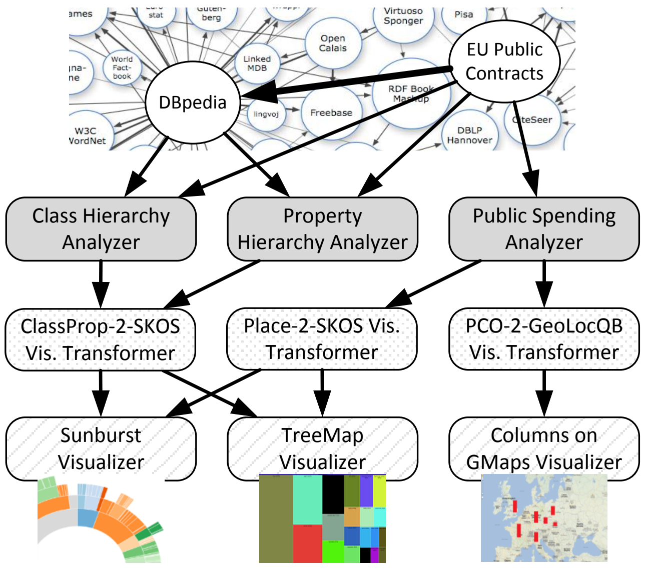


Figure 3: Sample analyzers and visualizers

alize 2-dimensional structures on Google Maps. An analytical abstraction may contain encoded both the tree structure as well as the 2-dimensional structure. All three mentioned visualizers can be applied to the analytical abstraction as well as on any other abstraction which contains the same structures encoded. Therefore, we need to transform the analytical abstraction into the form accepted by the desired visualizers. An example can be that we have a visualizer for a tree-like structure which accepts the SKOS[6] vocabulary with its `skos:broader` property for the hierarchy. The analytical abstraction might already contain this hierarchy, then no visualization transformation is required. Or, the analytical abstraction might contain a tree-like hierarchy modeled using `rdfs:subClassOf` property and it needs to be transformed here. This transformation is performed by the visualization abstraction stage. We call the transformation a *visualization transformer*. Again, a user can reuse various transformers for extracting visualization abstractions of the desired kind from compatible analytical abstractions.

View Stage. The output of the (*view*) stage is produced by a component called a *visualizer*. A view is a visual representation of a visualization abstraction on the screen. A visualizer performs *visual mapping transformation* that may be configured by a user using various parameters, e.g. visualization technique, colors and shapes. The user can also manipulate the final view using the view in-stage operators such as zoom and move. A visualizer can be reused for visualizing various visualization abstractions that contain data structures accepted by the visualizer.

3.3 Formalization

The core concepts of LDVM are reusable components, i.e. analyzers, visualization transformers and visualizers. *Analyzers* and *visualization transformers* consume RDF data via their input interfaces and produce RDF data as their output. *Visualizers* consume RDF data and produce a visualization a user can interact with. The goal is to formally introduce the concept of compatibility of a component with its input RDF data. We formalized the model in our previous work [4]. However, since then as the implementation progressed, we have simplified the formalization for it to be more practical and with no effect on its power. Given the formalization, we are then able to decide whether a given analyzer can be applied on a given RDF dataset. Similarly, we can decide whether a visualization transformer can be applied on a given analytical abstraction, etc. Our approach is based on the idea to describe the expected input of a LDVM component with an *input signature* and the expected output with an *output data sample*. The signature and the data sample are provided by the creator of the component. Each component can then check whether its input signature is compatible with the output sample of the previous component. The input signature comprises a set of SPARQL ASK queries which should be inexpensive so that they can be evaluated quickly on a number of datasets. The output data sample is a small RDF data sample that shows the format of the output of the component. The input signature of one component is then compatible with the output data sample of another component when all the SPARQL ASK queries of the signature are evaluated on the data sample as *true*. Our rationale is to provide a simple and lightweight solution, which allows to check the compatibility of a number of components without complex reasoning.

DEFINITION 1 (INPUT SIGNATURE). *A set of SPARQL ASK queries $\mathcal{S}_\mathcal{C} = \{Q_1, Q_2, \ldots, Q_n\}$ is an* input signature *of a LDVM component C.*

Note that an analyzer can potentially extract data from multiple data sources, e.g., SPARQL endpoints or RDF files. Then the analyzer would have to have a separate input signature for each data source. However, for simplicity, we omit this slight extension. Analyzers and visualization transformers provide an output data sample, against which an input signature of another LDVM component can be checked.

DEFINITION 2 (OUTPUT DATA SAMPLE). *RDF data $\mathcal{D}_\mathcal{C}$ representing the maximum possible structure of the output data format produced by a LDVM component C using minimum amount of triples is an* output data sample *of C. This only applies to analyzers and visualization transformers.*

DEFINITION 3 (COMPATIBILITY). *We say that LDVM component C with input signature S_C is* compatible *with LDVM component D with output data sample D_D iff each $Q_i \in \mathcal{S}_\mathcal{C} = \{Q_1, Q_2, \ldots, Q_n\}$ returns* true *when executed on $\mathcal{D}_\mathcal{D}$, i.e., $\prod_{i=1}^{n} E(Q_i, D_D) = 1$ where $E(Q_i, D_D) \in \{0, 1\}$ is the evaluation of SPARQL ASK query Q_i against data D_D.*

Given the output data samples are small and the SPARQL ASK queries are inexpensive, we can, for a given SPARQL endpoint, automatically offer all possible visualizations using available LDVM components to our lay users. The process for checking of available visualizations using LDVM starts with the analyzers. Each analyzer performs SPARQL ASK queries from its input signature. If it is compatible (all ASKs return `true`), it is marked as available. Next are the visualization transformers. Because they are optional and also can be chained, they need to perform their checks in iterations. In the first iteration, all transformers perform their ASKs from their input signatures on the output data samples of available analyzers. Those who succeed are marked

[6]`http://www.w3.org/TR/skos-reference/`

available. In the next iteration, all transformers that are not available perform their ASKs on the output data samples of the newly available transformers. This ends when there is no new available transformer. Finally, all visualizers perform their ASKs on all available analyzers and visualization transformers. The result is a set of all possible combinations of what can be visualized in the given SPARQL endpoint. See Figure 3 for illustration.

4. IMPLEMENTATION: PAYOLA

Payola[7] is a web framework for analyzing and visualizing Linked Data [12]. It enables users to build their own instances of LDVM pipelines. Payola provides an LDVM analyzer editor in which SPARQL queries and custom plugins can be combined. Firstly, the user defines a set of data sources such as SPARQL endpoints or RDF files as input data and then connects other plugins to them. Some of the plugins are designed to provide simple SPARQL constructs. Join and Union plugins enable users to analyze a dataset created from multiple datasets stored in separate SPARQL endpoints. It is also possible to transform results of an analyzer with a custom transformer. When the pipeline is evaluated, the user can choose a visualizer to see the results in various forms. Throughout the LDVM pipeline all data is RDF and the user can download the results in a form of an RDF file.

Payola also offers collaborative features. A user is able to create an analyzer and share it with the rest of the Payola users. That enables them to run such an analyzer as well as to create a new analytical plugin, which is based on that analyzer. As analytical plugins have parameters that affect their behavior, a new analyzer–based plugin may also have parameters, which can be chosen from the parameters of the plugins of the original analyzer. This feature supports formation of an ecosystem where expert users create analyzers for those who are less experienced. Combining those analyzers into new ones enables even inexperienced users to create a complex analyzer with less effort.

It is possible to extend Payola with custom plugins for analysis and visualization. For instance, a user is allowed to upload a code snippet of a new analytical plugin via our web interface. The framework compiles the code and integrates the created plugin immediately into the application.

Let us briefly describe some of the latest Payola features. Based on the previous user evaluation presented in [4], we focused our work on improving the user experience. We introduced changes to make it even easier for non–expert users to browse LDVM pipeline results without extensive knowledge of LOD principles or Payola itself.

The latest Payola version offers a one–click solution for presenting results of an LDVM pipeline in a chosen visualizer. When an LDVM pipeline is created, it is assigned a unique URL. When a user accesses such a URL, Payola automatically loads the pipeline and creates the desired visualization (see Section 5.3.2). To speed things up, we also implemented caching of analyzer results so that we can serve more users in a shorter time without repeated analysis evaluation. This brings us very close to what we see as a final stage of delivering a visualization to a non–expert user – embedding an LDVM visualization based on an LDVM pipeline into an external website. That is a part of our future work.

[7]`http://payola.cz`

5. DEMONSTRATION OF LDVM

In this section, we present our real world example of implementation and usage of LDVM. We present various analyzers, visualization transformers and visualizers, which are actual LDVM components with input signatures and output data samples. The examples run in Payola, our LDVM implementation (see Section 4).

5.1 Analyzers

In this section, we describe two analyzers that create analytical abstractions from the CzLOD cloud, their input signatures and output data samples. An analyzer is a software component that produces RDF data and for a given data source (or possibly more data sources) can say whether it can extract data from this data source or not. It can, for instance, represent a complex computation over simple data or it can simply be a SPARQL query, which is a case of our two examples. Note that when an analyzer is in a form of a SPARQL CONSTRUCT query, its input signature corresponds to its WHERE clause and its output data sample is an instance of its CONSTRUCT clause.

5.1.1 A1: Institutions of public power

The first analyzer A_1 takes data from 2 datasets: Institutions of public power (OVM) and Geocoordinates. From the OVM dataset, it extracts the institutions with their types and addresses[8]. The types of the institutions are expected to be `skos:Concept`s and the labels of the types are expected to be `skos:prefLabel`s. From the Geocoordinates dataset, the analyzer extracts geocoordinates gained by geocoding the OVM addresses. The input signature of the analyzer consists of one ASK query $\mathcal{S}_{A_1} = \{Q^{A_1}\}$:

```
# Q of A1
[] s:name [] ;
   ovm:typSubjektu ?type ;
   s:address ?address .
?address s:streetAddress [];
         s:postalCode [];
         s:addressRegion [];
         s:addressLocality [];
         s:geo ?g.
?type skos:prefLabel [] .
?g s:longitude [];
   s:latitude [] .
```

And an example of its output data sample $\mathcal{D}_{A_1}$ is:

```
# D of A1
<ovm> s:geo <geo> ;
      s:title "title";
      s:description "desc" ;
      ovm:typSubjektu <type>.
<type> skos:prefLabel "Type" .
<geo> s:latitude "50.088289";
      s:longitude "14.404446".
```

Our SPARQL endpoint contains this kind of data and therefore Q^{A_1} returns `true`, which means that A_1 is compatible with our SPARQL endpoint.

5.1.2 A2: Inspections of COI.CZ

The second analyzer A_2 takes data from 5 datasets: Inspections of the Czech Trade Inspection Agency (COI.CZ), ARES (Business Registry), NUTS codes hierarchy, LAU codes hierarchy and also Geocoordinates. From COI.CZ it extracts information about inspections which resulted into

[8]`s` is a prefix for `http://schema.org/`

sanctions. Specifically, it extracts their dates, places, resulting fines, links to business entities inspected and links to LAU regions in which the inspection took place. From LAU regions, the analyzer takes names of the regions and links to broader NUTS codes. From NUTS codes, the analyzer takes names of the regions and their hierarchy.

From ARES, the analyzer extracts names of inspected business entities. Finally, from Geocoordinates, it extracts the geocoordinates of addresses found in COI.CZ. The input signature of this analyzer consists of 2 SPARQL ASK queries $\mathcal{S}_{A_2} = \{Q_1^{A_2}, Q_2^{A_2}\}$:

```
# Q1 of A2
[] a s:CheckAction;
   s:location/s:location ?region;
   s:location/s:geo ?geo;
   s:object ?object;
   dcterms:date ?date ;
   s:result ?result.
?result a coicz:Sanction;
        s:result/gr:hasCurrencyValue [] .
?object gr:legalName [] .
?region a ec:LAURegion;
        ec:level 2 .
?geo s:latitude [];
     s:longitude [].
FILTER(datatype(?date) = xsd:date)
```

$Q_1^{A_2}$ checks for the inspections `s:CheckAction`, its region (LAU), geocoordinates, business entity, date and fine. The fine has to have an amount, the business entity has to have a legal name, the region must be LAU level 2. $Q_2^{A_2}$ checks the LAU and NUTS datasets whether there is the region hierarchy present and whether the regions have their names.

```
# Q2 of A2
[] a s:CheckAction;
   s:location/s:location ?region;
   s:result/s:result [] .
?region a ec:LAURegion;
   ec:level 2 ;
   dcterms:title [] ;
   ec:hasParentRegion ?lau1.
?lau1 dcterms:title [] ;
      ec:hasParentRegion ?nuts3 .
?nuts3 rdfs:label [] ;
       ec:hasParentRegion ?nuts2 .
?nuts2 rdfs:label [] ;
       ec:hasParentRegion ?nuts1 .
?nuts1 rdfs:label [].
```

This data is present in our SPARQL endpoint so both the queries return `true`. Therefore, A_2 is compatible with our data source. An example of the output data sample $\mathcal{D}_{A_2}$ is:

```
# D of A2
<ca> a s:CheckAction;
     s:location <region> ;
     s:geo <geo>;
     s:title "title";
     s:description "description";
     dcterms:date "2014-02-16"^^xsd:date ;
     rdf:value 2 .
<geo> s:latitude "50.088289";
      s:longitude "14.404446".
<region> a ec:LAURegion;
         ec:level 2 ;
         rdfs:label "label" ;
         ec:hasParentRegion <lau1>.
<lau1> rdfs:label "label" ;
       ec:hasParentRegion <nuts3> .
<nuts3> rdfs:label "label" ;
         ec:hasParentRegion <nuts2> .
<nuts2> rdfs:label "label" ;
         ec:hasParentRegion <nuts1> .
<nuts1> rdfs:label "label".
```

5.2 Visualization transformers

In this section we describe a sample visualization transformer. It can be used to connect output RDF data from our analyzers or any other compatible analyzers to the inputs of our visualizers. A visualization transformer can be any software component that transforms data between different formats or performs aggregations for better visualization. Note that because we use RDF, the visualization transformers are in fact SPARQL CONSTRUCT queries. Again, their input signatures correspond to their FROM clauses and their output data samples correspond to their CONSTRUCT clauses.

5.2.1 T1: Region hierarchy to SKOS hierarchy

Because we have various tree structure visualizers that accept tree structures using `skos:Concept`s for nodes with `skos:prefLabel` for labels and `skos:broader` properties for edges and also accept optional `rdf:value` for the size of a leaf, we need to transform the hierarchy extracted in analyzer A_2 (see Section 5.1.2) accordingly. The region hierarchy, that is in the output data sample of analyzer A_2 consists of `ec:LAURegion`s for regions, `s:CheckAction` for the inspections made by COI.CZ. In addition, the inspections have their sanction amounts in `rdf:value`, which we want to visualize as sizes of their corresponding leaves in the resulting tree visualization. Therefore, the input signature of T_1 consists of one SPARQL ASK query $\mathcal{S}_{T_1} = \{Q^{T_1}\}$ which corresponds to the output data sample of A_2:

```
# Q of T1
[] a s:CheckAction;
   s:location ?region;
   s:title [] ;
   rdf:value [] .
?region a ec:LAURegion;
        ec:level 2 ;
        rdfs:label [] ;
        ec:hasParentRegion ?lau1.
?lau1 rdfs:label [] ;
      ec:hasParentRegion ?nuts3 .
?nuts3 rdfs:label [] ;
       ec:hasParentRegion ?nuts2 .
?nuts2 rdfs:label [] ;
       ec:hasParentRegion ?nuts1 .
?nuts1 rdfs:label [] .
```

An example of its output data sample $\mathcal{D}_{T_1}$ will correspond to the input signature of visualizer V_1 (see Section 5.3.2):

```
# D of T1
<ca> a skos:Concept;
    skos:prefLabel "title";
    rdf:value 100 ;
    skos:broader <region> .
<region> a skos:Concept;
    skos:prefLabel "label" ;
    skos:broader <lau1> .
<lau1> a skos:Concept;
    skos:prefLabel "label" ;
    skos:broader <nuts3>.
<nuts3> a skos:Concept;
    skos:prefLabel "label" ;
    skos:broader <nuts2>.
<nuts2> a skos:Concept;
    skos:prefLabel "label" ;
    skos:broader <nuts1>.
<nuts1> a skos:Concept;
    skos:prefLabel "label".
```

5.3 Visualizers

In this section, we present sample visualizers which visualize the results of the aforementioned analyzers. Moreover,

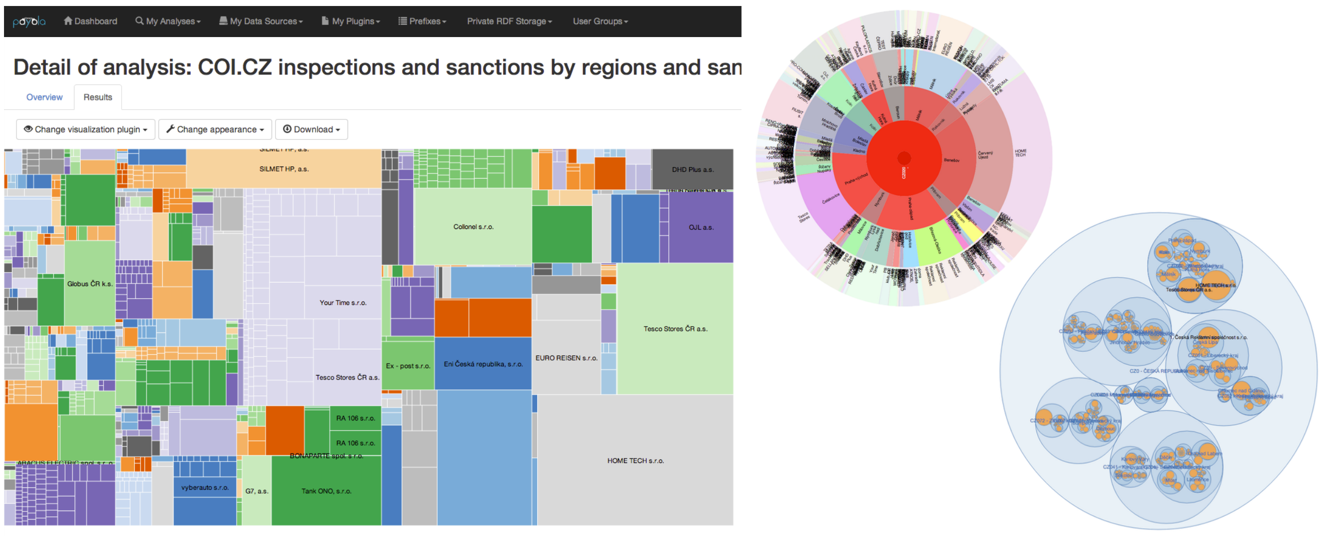


Figure 4: Tree hierarchy visualizations of a pipeline based on analyzer A_2 and transformer T_1. A treemap on the left side, sunburst and circle layout packing on the right side.

they demonstrate how visualizers benefit from the concept of input signatures and compatibility checks. For each of visualizers, we will describe its input signature. Since a product of a visualizer is not a dataset, but a visualization, there is no specification of an output data sample. The compatibility check is, once again, a SPARQL ASK query which is, in the case of a visualizer, executed against an output data sample of the last transformer in a given LDVM pipeline.

Since one of the main reasons why the LDVM was proposed is to facilitate the process of LOD exploration, we have chosen to utilize some well-known visualization techniques to present a dataset in a form, which is understandable by non-expert users. We have experimented with two commonly used visualization techniques: a tree hierarchy visualization and a map visualization. One of the goals of our experiments was to show that it is possible to integrate well-known visualization libraries into an application, which works with RDF and is based on the LDVM.

5.3.1 V1: Tree hierarchy visualizers

Tree hierarchy visualization is a commonly used visualization technique. The results of the analyzers A_1 and A_2 (followed by the transformer T_1) contain hierarchical data structures which can be visualized in this way. As described before, we chose the SKOS vocabulary as a format for tree visualizations and therefore we present Q^{V_1} as the input signature for a tree hierarchy visualizer V_1:

```
# Q of V1
[] a skos:Concept ;
    skos:prefLabel [] ;
    rdf:value [] ;
    skos:broader ?b .

?b a skos:Concept ;
    skos:prefLabel [] .
```

Query Q^{V_1} enforces that the visualized dataset contains a leaf node with a value specified, as well as a reference to its parent. To traverse the hierarchy, we use the skos:broader property, which stands for the *has parent* relationship. It is now easy to check compatibility of Q^{V_1} with D^{A_1} and D^{T_1}.

We chose to implement 4 different tree hierarchy visualizers. To demonstrate the flexibility of a LDVM-compliant framework, we decided to use a freely available visualization techniques based on a well-known and commonly used document manipulation library *D3.js*[9]. Specifically, we introduce the following visualizers: Zoomable Treemap, Sunburst, Zoomable Sunburst, and Layout Packing. The library provides a module which produces adjacency diagrams or a hierarchical layout using recursive circle-packing for a given tree structure. It is not a hard task to build the expected tree structure of JavaScript objects based on the data that conforms the described input signature. Among others, we use Apache Jena[10] to serialize the results of an analyzer or a transformer into *RDF/JSON*[11]. The serialization is transferred to a user's web browser, processed by a visualizer and passed to the visualization library, which computes the visualization itself. Note that LDVM does not specify implementation details, we could use JSON-LD, RDF/XML, Turtle or any other serialization format, moreover an arbitrary non-RDF format.

We present some visualizations based on aforementioned analyzers in Figure 4 (live demos [12] [13] [14]).

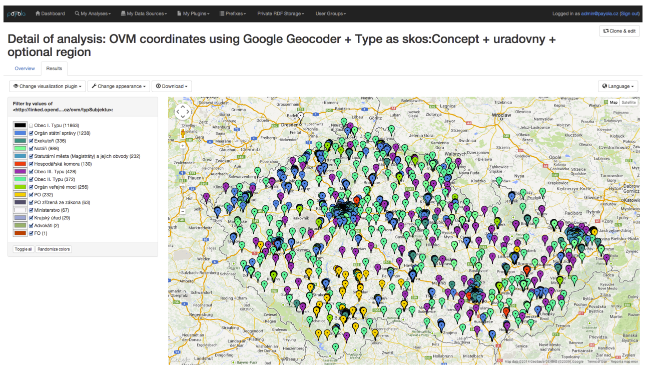
5.3.2 V2: Geo data visualizers

Geo data visualization is another example of commonly used visualization techniques. There are many Open Data mashups that integrate map visualizations in order to provide an eye–catching presentation of arbitrary datasets.

The input signature of a map visualizer can be actually very simple. We define Q^{V_2} to be the only query of an input signature of the visualizer V_2:

```
# Q of V2
?[] s:geo ?c ;
    s:description [] ;
```

[9] http://d3js.org
[10] https://jena.apache.org/
[11] https://dvcs.w3.org/hg/rdf/raw-file/default/rdf-json/index.html
[12] http://vis.payola.cz/coi-treemap
[13] http://vis.payola.cz/coi-z-sunburst
[14] http://vis.payola.cz/coi-packed


Detail of analysis: OVM coordinates using Google Geocoder + Type as skos:Concept + uradovny + optional region

Filter by values of <http://linked.opend....cz/ovm/typSubjektu>:

- ■ ☐ Obec I. Typu (11863)
- ■ ✓ Orgán státní správy (1238)
- ■ ✓ Exekutoři (336)
- ■ ✓ Notáři (988)
- ■ ✓ Statutární města (Magistráty) a jejich obvody (232)
- ■ ✓ Hospodářská komora (130)
- ■ ✓ Obec III. Typu (428)
- ■ ✓ Obec II. Typu (372)
- ■ ✓ Orgán veřejné moci (256)
- ■ ✓ PO (232)
- ■ ✓ PO zřízená ze zákona (63)
- □ ✓ Ministerstvo (67)
- ■ ✓ Krajský úřad (29)
- ■ ✓ Advokáti (2)
- ■ ✓ FO (1)

Figure 5: Map visualization with the faceted browsing feature enabled. It presents data of a pipeline based on analyzer A_1.

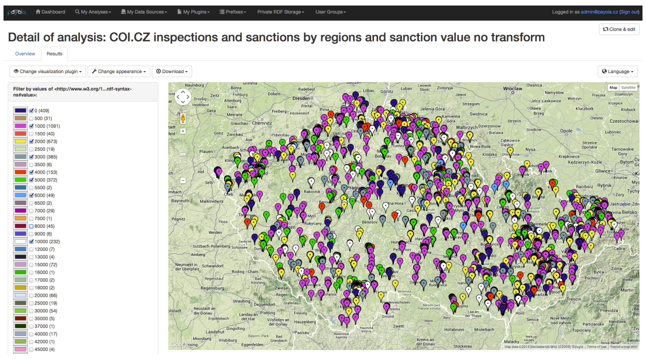
```
    s:title   [] .

?c  s:geoCoordinates [
    s:latitude  [] ;
    s:longitude [] .
] .
```

Again, it is easy to see that such a query would return *true* when matched against D^{A_1} or D^{A_2}. Hence, V_2 is compatible with A_1 and A_2.

We integrated two different map visualization libraries in order to provide three different visualizers. Two of them are based on the *Google Maps JavaScript API*[15] and the third one utilizes the *ArcGIS API for JavaScript*[16]. The former stands for a classic map visualization where a resource with geo data is represented on a map by a single marker. The other one generates a heatmap layer. The signature does not contain any additional values. Each resource contributes to the generated heatmap layer equally, while locally increasing the intensity of the heatmap by one. The third visualizer utilizes a clustering layer which automatically groups markers that are close to each other. When zooming in, markers are getting further apart. Therefore, the layer dissolves clusters into smaller ones or reveals single markers.

These map visualizers clearly motivate the notion of input signatures. We have three different software components doing the same task so it is very natural to unify their input format. As in the case of hierarchies, we could utilize other vocabularies with properties such as `wgs84:lat` or `geo:point`, which, in fact, have the same semantic meaning. That is also one of the reasons for the support of transformer chaining in LDVM. We could have a LDVM pipeline, where an analyzer outputs data in a proprietary geographic coordinate system, followed by a transformer, which converts such a system to WGS84 using the `wgs84` ontology, followed by a transformer, which converts `wgs84` to `s:geoCoordinates`.

To make the basic map visualizer more usable, we extended it to provide a faceted browsing capability. Let us use the results of analyzer A_1 for demonstration. Consider input signature Q^{V_2} and the output data sample D_{A_1}. A visualizer with this signature ignores other properties, such as `ovm:typSubjektu` (type of institution). The types of institutions are instances of `skos:Concept`, which is often used as a type. This might suggest that the property links the resource (institution) to its type. In our case, the institution type can be a notary, a municipality, a ministry, etc. We allow the users to use these properties as facets to customize the visualization when exploring the dataset by letting them, e.g., to change a color of a specific type of institutions or even hide them.


Detail of analysis: COI.CZ inspections and sanctions by regions and sanction value no transform

Filter by values of <http://www.w3.org/1...rdf-syntax-ns#value>:

- ■ ✓ 0 (409)
- ■ ☐ 500 (31)
- ■ ✓ 1000 (1091)
- ■ ☐ 1500 (40)
- ■ ✓ 2000 (673)
- ■ ☐ 2500 (19)
- ■ ✓ 3000 (385)
- ■ ☐ 3500 (6)
- ■ ✓ 4000 (153)
- ■ ✓ 5000 (372)
- ■ ☐ 5500 (2)
- ■ ✓ 6000 (49)
- ■ ☐ 6500 (2)
- ■ ☐ 7000 (29)
- ■ ☐ 7500 (1)
- ■ ☐ 8000 (45)
- ■ ☐ 9000 (6)
- □ ✓ 10000 (232)
- ■ ☐ 12000 (7)
- ■ ☐ 13000 (4)
- ■ ☐ 15000 (72)
- ■ ☐ 16000 (1)
- ■ ☐ 17000 (2)
- ■ ☐ 18000 (2)
- ■ ☐ 20000 (66)
- ■ ☐ 25000 (19)
- ■ ☐ 30000 (54)
- ■ ☐ 35000 (5)
- ■ ☐ 37000 (1)
- ■ ☐ 40000 (17)
- ■ ☐ 42000 (1)
- ■ ☐ 45000 (4)

Figure 6: Map visualization of the data produced by analyzer A_2.

To detect this type of properties, the following query is executed against the visualized dataset:

```
select distinct ?p where
{
    [] <http://schema.org/geo> [];
       ?p [] .
}
```

It gives us a list of properties that might be used for marker grouping. Since such a list contains also properties defined by the input signature, we need to involve property blacklisting to exclude properties that would probably create a group for each marker separately (titles, descriptions, etc.).

Examples of visualizations with faceted browsing can be seen in Figures 5[17] and 6[18] (see footnotes for links to live demos). When multiple properties are matched, we need to solve some minor issues such as compute visibility based on all filters or how to apply multiple color settings to a single marker. In the case of the color, we let the user decide which property is to be used to change colors of markers.

5.4 Evaluation

One of the main aims of LOD is to enable data reuse in unforeseen ways by third parties. Specifically, public bodies representatives often state that they do not want to publish raw data, because it does not have a nice visualization for the public. And then they spend large amounts of money to build custom portals with functionality that has been implemented many times before that visualize their unpublished data. They are hard to convince that publishing the data itself, not to mention in some standardized format or even as LOD, can bring them benefits. This is because for the general public the data is useless without interpretation in a form of an application. What the public bodies do not realize is that the development of those applications could be left to third-parties. With these facts in mind we can say that with Payola framework and LDVM as its formal

[15]`https://developers.google.com/maps/documentation/javascript/`
[16]`https://developers.arcgis.com/javascript/`

[17]`http://vis.payola.cz/ovm-gmaps`
[18]`http://vis.payola.cz/coi-gmaps`

background, we can show a library of analyzers, transformers and visualizers that can be easily used and reused for all Linked Data. In addition, we have concrete examples as evidence of feasibility of this approach as shown in this paper. We also showed that implementation of open source visualizers such as those from D3js.org as plugins to Payola can be done easily. The data from COI.CZ, that are presented in this paper, are one of the COMSODE datasets. The publisher of this data in COI.CZ now gets a free and powerful visualization of its data and integration with other datasets and all he had to do was to publish a CSV file.

6. RELATED WORK

More and more projects are focused on analyzing, exploring and visualizing Linked Data. The most sophisticated survey to date has been presented by Dadzie and Rowe [6]. They concluded that most of the tools were not suitable to be used by lay users and the situation has not significantly improved since. One is still required to understand the basics of the Semantic Web while using Linked Data browsers such as *Tabulator* [2] and *Explorator* [1]. The user is expected to navigate a graph through tabular views displaying property–value pairs of explored resources. However, they do not offer features that would enable a user to overview a whole dataset. At the time of writing of this paper, Tabulator did not support current versions of web browsers and therefore it was not possible to completely check up on its progress. Compared to our one-click pipeline execution, which enables an expert user to prepare a visualization and share it with a non-expert one, we find Explorator a rather complicated tool. It is not very easy even for an expert user to start using it. Another exploration tool is Freebase Parallax [19], which offers advanced visualizations like timelines, maps and other rich snippets, but works with a fixed data source – Freebase. Semaplorer [15] is an exploration mashup based on multiple large datasets. It demonstrates the power of Linked Data while being focused on the tourism data domain. It provides faceted browsing capabilities for 4 different types of facets (person, time, tag and location).

There are several tools that visualize data based on vocabularies in a similar way that our new visualizers do. Let us start with visualizers like *map4rdf*[20], *LinkedGeoData browser* [16] and *Facete*[21], which understands spatial data. The first two focus on geographical data visualizations, but both are built on top of specific datasets. That means that compared to Payola, the user is not able to apply the visualizer to his own dataset. map4rdf supports faceted discovery of Spanish institutions and enables the user to add a specific overlay containing statistical SCOVO-based data in a form of a timeline visualization. According to [7], the authors are currently working on DataCube vocabulary support. Its most interesting feature is filtering of values by choosing an arbitrary region on a map. LinkedGeoData browser enables its users to explore POIs all over the world. Facete is a JavaScript SPARQL-based Faceted Search library. It enables users to explore an arbitrary dataset stored on an arbitrary SPARQL endpoint. Using facets a user is able to narrow down the volume of the explored data. Facete offers a basic table view, but it also provides some more sophisticated visualization widgets. One of them is focused on visualization of spatial data. Since Facete is an exploration tool, it completely lacks features that would provide data analysis or transformation like Payola does. It just enables a user to explore and filter data from a chosen SPARQL endpoint. Another group are vocabulary-based visualizers. *CubeViz* [9] offers the same circle packing layout visualization as Payola does, but based on the DataCube vocabulary. Payola also offers an experimental version of a DataCube visualizer, but is not limited to it. *FoaF Explorer*[22] is focused on visualizing FOAF profiles. One can also mention *ViDaX* [8], a Java desktop Linked Data visualizer based on the Prefuse [23] visualization library. Based on ontologies and property types, it suggest suitable visualizations to its users. However, we did not find a copy of the tool anywhere so we were unable to experiment with it. *Rhizomer* [3] offers various types of visualizations for different datasets. It suggests a reasonable workflow for datasets exploration: 1) Overview, 2) Filter, 3) Visualize. It includes the treemap, timeline, map and chart visualizers. However, it focuses just on the visualization stage of a LDVM pipeline.

There are also tools which let the user to build a custom analyzer as Payola does. The best known is *Deri Pipes* [13], which is a platform that enables a user to create mashups and perform data transformations. However, it is focused just on data analysis which means that there could be a Payola analytical plugin which would use a pipeline produced by Deri Pipes as another analyzer data source. *Open Data Mashup* [24] provides a very similar functionality, but it also offers visualizations based on vocabularies, including map visualizations. It is based on two types of widgets a user is able to combine together. The first one is a data source, the second one is a visualizer. However, it distinguishes only two dataset types - statistical and spatial and lacks flexibility since a visualizer receives data a widget which combines a data source, an analyzer and a transformer.

We have also seen some generic graph visualizers like *VisualRDF*[25], which is a work in progress and is being currently developed while utilizing the D3.js library. Tools like *IsaViz* [14], *Fenfire* [11] and *RDF–Gravity*[26] use the well-known node-link visualization technique to represent a dataset. Payola also offers generic graph visualizations, but on top of that, it provides a way of customizing the visualization based on ontologies and user-defined vocabularies. Using an extensible library of visualizers, Payola is able to visualize an arbitrary dataset.

IsaVis also belongs to a group of tools implementing Fresnel - Display Vocabulary for RDF [27], which specifies how a resource should be visually represented by Fresnel-compliant tools like LENA [28] and Longwell [29]. Those are also focused only on the visualization stage of LDVM. Fresnel vocabulary could be perceived as a LDVM visualization abstraction.

We have already mentioned Facete, which is a SPARQL based JavaScript library. There are also other similar li-

[19]http://parallax.freebaseapps.com
[20]http://oeg-dev.dia.fi.upm.es/map4rdf/
[21]http://cstadler.aksw.org/facete/
[22]http://xml.mfd-consult.dk/foaf/explorer/
[23]http://prefuse.org/
[24]http://ogd.ifs.tuwien.ac.at/mashup/
[25]https://github.com/alangrafu/visualRDF
[26]http://semweb.salzburgresearch.at/apps/rdf-gravity/
[27]http://www.w3.org/2005/04/fresnel-info/
[28]https://code.google.com/p/lena/
[29]http://simile.mit.edu/issues/browse/LONGWELL

braries like *Sgvizler*[30] or *Visualbox*[31], which enables a user to embed a dataset visualization into their website. Unlike Facete, they require a user to have a deep knowledge of SPARQL language, since that is the only possible way of using those tools. Last but not least, we mention a publishing framework *Exhibit*[32]. It enables the user to create web pages with advanced search and filtering features providing visualizations like maps, timelines or charts. However, it requires the input data to be in a form of JSON and recommends using *Babel*[33] service to transform RDF and other data formats into the desired JSON variant.

7. CONCLUSIONS

In this paper, we presented the *Czech LOD cloud* – a set of interlinked LOD datasets we have published in our research group and we used it for demonstration of the benefits of the *Linked Data Visualization Model* (LDVM) for LOD visualization. We briefly recapitulated the basic principles of LDVM, updated its formalization and shortly described our own implementation of LDVM - *Payola*. Then we presented several visualization scenarios of datasets from the Czech LOD cloud. The scenarios demonstrated benefits of LDVM for users – how they can combine various LDVM components to extract required data structures from their datasets (with so called *analyzers* and *visualization transformers*) and how even lay users can easily reuse suitable *visualizers* to visualize the extracted structures.

8. ACKNOWLEDGMENTS

This work was partially supported by a grant from the European Union's 7th Framework Programme number 611358 provided for the project COMSODE and also partially by the TAČR grant no. TA02010182.

9. REFERENCES

[1] S. Araujo, D. Shwabe, and S. Barbosa. Experimenting with Explorator: a Direct Manipulation Generic RDF Browser and Querying Tool. In *WS on Visual Interfaces to the Social and the Semantic Web (VISSW2009)*, 2009.

[2] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *3rd Int. Semantic Web User Interaction WS*, 2006.

[3] J. Brunetti, R. Gil, and R. Garcia. Facets and Pivoting for Flexible and Usable Linked Data Exploration. In *Interacting with Linked Data Workshop, ILD'12*, Crete, Greece, May 2012.

[4] J. M. Brunetti, S. Auer, R. García, J. Klímek, and M. Nečaský. Formal Linked Data Visualization Model. In *Proceedings of the 15th International Conference on Information Integration and Web-based Applications & Services (IIWAS'13)*, pages 309–318, 2013.

[5] E. H. Chi. A Taxonomy of Visualization Techniques Using the Data State Reference Model. In *IEEE Symposium on Information Vizualization 2000*, INFOVIS '00, Washington, DC, USA, 2000. IEEE.

[6] A.-S. Dadzie and M. Rowe. Approaches to visualising Linked Data. *Semantic Web*, 2(2):89–124, 2011.

[7] A. de León, F. Wisniewki, B. Villazón-Terrazas, and O. Corcho. Map4rdf - Faceted Browser for Geospatial Datasets. In *Proceedings of the First Workshop on USING OPEN DATA*. W3C, June 2012.

[8] B. Dumas, T. Broché, L. Hoste, and B. Signer. Vidax: An interactive semantic data visualisation and exploration tool. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 757–760, New York, NY, USA, 2012. ACM.

[9] I. Ermilov, M. Martin, J. Lehmann, and S. Auer. Linked open data statistics: Collection and exploitation. In P. Klinov and D. Mouromtsev, editors, *Knowledge Engineering and the Semantic Web*, volume 394 of *Communications in Computer and Information Science*, pages 242–249. Springer Berlin Heidelberg, 2013.

[10] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[11] T. Hastrup, R. Cyganiak, and U. Bojars. Browsing Linked Data with Fenfire. In *Linked Data on the Web (LDOW2008) workshop, in conjunction with WWW 2008 conference*, 2008.

[12] J. Klímek, J. Helmich, and M. Nečaský. Payola: Collaborative Linked Data Analysis and Visualization Framework. In *10th Extended Semantic Web Conference (ESWC 2013)*, pages 147–151. Springer, 2013.

[13] D. Le-Phuoc, A. Polleres, M. Hauswirth, G. Tummarello, and C. Morbidoni. Rapid prototyping of semantic mash-ups through semantic web pipes. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 581–590, New York, NY, USA, 2009. ACM.

[14] E. Pietriga. IsaViz: a Visual Environment for Browsing and Authoring RDF Models. In *WWW 2002, the 11th World Wide Web Conference*, Honolulu, Hawaii, USA, 2002. World Wide Web Consortium.

[15] S. Schenk, C. Saathoff, S. Staab, and A. Scherp. SemaPlorer—interactive semantic exploration of data and media based on a federated cloud infrastructure. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):298–304, 2009.

[16] C. Stadler, J. Lehmann, K. Höffner, and S. Auer. LinkedGeoData: A Core for a Web of Spatial Open Data. *Semantic Web Journal*, 2011.

[30] http://dev.data2000.no/sgvizler/

[31] http://alangrafu.github.io/visualbox/

[32] http://www.simile-widgets.org/exhibit/

[33] http://service.simile-widgets.org/babel/