Formal Linked Data Visualization Model [⋆]

Josep Maria Brunetti[1], Sören Auer[2], Roberto García[1], Jakub Klímek[3], and Martin Nečaský[3]

[1] GRIHO, Universitat de Lleida
{josepmbrunetti, rgarcia}@diei.udl.cat, http://griho.udl.cat/

[2] AKSW, Universität Leipzig
auer@uni-leipzig.de, http://aksw.org/

[3] Charles University in Prague
Faculty of Mathematics and Physics
{klimek, necasky}@ksi.mff.cuni.cz, http://xrg.cz/

Abstract. In the last years, the amount of semantic data available in the Web has increased dramatically. The potential of this vast amount of data is enormous but in most cases it is very difficult for users to explore and use this data, especially for those without experience with Semantic Web technologies. Applying information visualization techniques to the Semantic Web helps users to easily explore large amounts of data and interact with them. In this article we devise a formal Linked Data Visualization Model (LDVM), which allows to *dynamically* connect data with visualizations. We report about our comprehensive implementation of the LDVM comprising a library of generic visualizations that enable both users and data analysts to get an overview on, visualize and explore the Data Web and perform detailed analyzes on Linked Data.

Keywords: Semantic Web, Linked Data, Visualization, Interaction

1 Introduction

In the last years, the amount of semantic data available on the Web has increased dramatically, especially thanks to initiatives like Linked Open Data (LOD). The potential of this vast amount of data is enormous but in most cases it is very difficult *and* cumbersome for users to visualize, explore and use this data, especially for lay-users [8] without experience with Semantic Web technologies. Visualizing and interacting with Linked Data is an issue that has been recognized from the beginning of the Semantic Web (cf. [11]). Applying information visualization techniques to the Semantic Web helps users to explore large amounts of data and interact with them. The main objectives of information visualization are to transform and present data into a visual representation, in such a way that users can obtain a better understanding of the data [5]. Visualizations

[⋆] We would like to thank our colleagues from the GRIHO, XRG and AKSW research groups for their helpful comments and inspiring discussions during the development of LDVM. This work was partially supported by a grant from the European Union's 7th Framework Programme provided for the project LOD2 (GA no. 257943).

are useful for obtaining an overview of the datasets, their main types, properties and the relationships between them.

Compared to prior information visualization strategies, we have a unique opportunity on the Data Web. The unified RDF data model being prevalent on the Data Web enables us to bind data to visualizations in an *unforeseen* and *dynamic* way. An information visualization technique requires certain data structures to be present. When we can derive and generate these data structures automatically from reused vocabularies or semantic representations, we are able to realize a largely automatic visualization workflow. Ultimately, we aim to realize an ecosystem of data extractions and visualizations, which can be bound together in a dynamic and unforeseen way. This will enable users to explore datasets even if the publisher of the data does not provide any exploration or visualization means.

Most existing work related to visualizing RDF is focused on concrete domains and concrete datatypes. The *Linked Data Visualization Model* (LDVM) we propose in this paper allows to connect different datasets with different visualizations in a dynamic way. LDVM balances between flexibility of visualization options and efficiency of implementation or configuration. Ultimately we aim to contribute with the LDVM to the creation of an ecosystem of data publication and data visualization approaches, which can co-exist and evolve independently. Our main contributions are in particular:

1. The adoption of the Data State Reference Model [6] for RDF through the creation of a formal Linked Data Visualization Model, which allows to *dynamically* connect data with visualizations.
2. A comprehensive, scalable implementation of the Linked Data Visualization Model comprising a library of generic data extractions and visualizations
3. An evaluation of our LDVM implementation using a benchmark consisting of 16 visualization tasks

The remainder of this article is organized as follows: Section 2 discusses related work. Section 3 introduces the Linked Data Visualization Model. Section 4 describes an implementation of the model and Section 5 presents its evaluation with different datasets and visualizations. Finally, Section 6 contains conclusions and future work.

2 Related Work

Exploring and visualizing Linked Data is a problem that has been addressed by several projects. Dadzie and Rowe [8] present the most exhaustive and comprehensive survey to date of existing approaches to visualising and exploring Linked Data. They conclude that most of the tools are designed only for tech-users and do not provide overviews on the data. *Linked Data browsers* such as *Tabulator* [2] or *Explorator* [1] allow users to navigate the graph structures and usually display property-value pairs in tables, but no broader view of the dataset. *Rhizomer* [3] provides an overview of the datasets and allows to interact with data through Information Architecture components such as navigation menus, breadcrumbs and facets. It also provides visualizations such as maps and timelines. *DERI Pipes*[4] is an engine and graphical environment for general Web Data transformations and Mashup. However, it is not intended for lay-users and requires SW

[4] `http://pipes.deri.org/`

Tool	Overview	Detail View	Automation	Lay users	Specific visualizations	Collaboration
Tabulator	-	✔	-	✔	M, T	-
Explorator	-	✔	✔	-	-	-
DERI Pipes	-	✔	-	-	-	✔
Fresnel	-	✔	-	-	-	-
Exhibit	-	✔	-	✔	M, T, C	-
Fenfire	-	✔	-	-	G	-
Sgvizler	✔	✔	-	-	C, G, T, M, R	-
Rhizomer	✔	✔	✔	✔	M, T, C	-
LODVis	✔	✔	✔	✔	R, M, E, B	-
Payola	✔	✔	-	-	C, E, O	✔

Table 1. Comparison of generic Linked Data visualization approaches. M - Map, T - Timeline, C - Chart, B - Bubble Chart, O - Circles, R - Treemap, G - Graph, E - Tree.)

expertise. *Graph-based tools* such as *Fenfire* [12], *RDF-Gravity*[5], *IsaViz*[6] provide node-link visualizations of the datasets and the relationships between them. Although this approach can help obtaining a better understanding of the data structure, in some cases graph visualization does not scale well to large datasets [10]. There are also *JavaScript libraries* for visualizing RDF. *Sgvizler*[7] renders the results of SPARQL queries into HTML visualizations such as charts, maps, treemaps, etc. It requires SPARQL knowledge in order to create RDF visualizations. *Exhibit*[8] helps users to create interactive sites with advanced text searching and filtering functionality. Other tools are restricted to visualizing and browsing concrete domains, e.g. *LinkedGeoData browser* [15] or *map4rdf*[9] for spatial data or *FoaF Explorer* for FOAF profiles. Table 1 shows a summary of generic Linked Data visualization approaches. To summarize, most existing tools make it difficult for non-technical users to explore linked data or are restricted to concrete domains. Very few of them provide generic visualizations for RDF data combined with high-automation and overview visualizations.

Regarding data visualization, most of the existing work is focused on categorizing visualization techniques and systems [9]. [4] and [7] propose various taxonomies of visualization techniques. However, most visualization techniques are only compatible with concrete domains or data types [5]. Chi's *Data State Reference Model* [6] defines the visualization process in a generic way. In Section 3 we describe how this model serves as a starting point for our Linked Data Visualization Model.

3 Linked Data Visualization Model

In this section we present the Linked Data Visualization Model (LDVM), which is based on a preliminary version [**?**]. First, we give an overview of the model and then we formalize its key elements. We also provide examples to explain the principles.

[5] `http://semweb.salzburgresearch.at/apps/rdf-gravity`
[6] `http://www.w3.org/2001/11/IsaViz`
[7] `http://code.google.com/p/sgvizler/`
[8] `http://simile-widgets.org/exhibit3/`
[9] `http://oegdev.dia.fi.upm.es/map4rdf/`

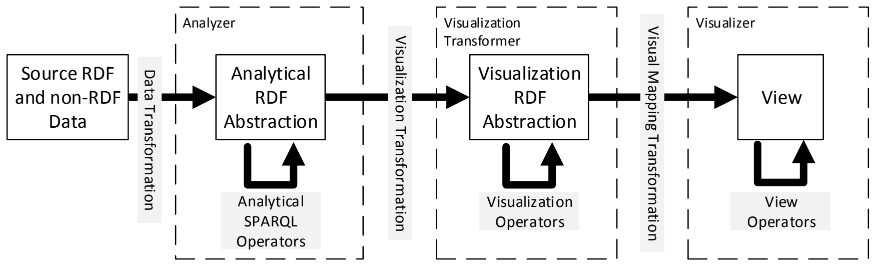
Source RDF and non-RDF Data
Data Transformation
Analyzer
Analytical RDF Abstraction
Analytical SPARQL Operators
Visualization Transformation
Visualization Transformer
Visualization RDF Abstraction
Visualization Operators
Visual Mapping Transformation
Visualizer
View
View Operators

Fig. 1. High level LDVM overview.

3.1 Overview of LDVM

We use the *Data State Reference Model* (DSRM) proposed by Chi [6] as a conceptual framework for our *Linked Data Visualization Model (LDVM)*. DSRM describes the visualization process in a generic way. Our LDVM is an adaptation of this generic model for the specifics of the visualization of RDF and Linked Data. The main difference is that in certain parts, LDVM works solely with RDF data model for increased automation while DSRM is generic in each of its parts and does not constraint the applied data models. We also 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 1 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 four 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; the data structure is based on generic data types for visual analysis proposed by Shneiderman [14] (i.e., 1D, 2D, 3D or multi-dimensional data, temporal data (as a special case of 1D), tree data, or network data)
4. *View:* creation of a visualization for the end user.

Data is propagated through the LDVM pipeline from one stage to another by applying three 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 transparent 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 as many data sources are currently not RDF. 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. Since data extraction is a vast research field on its own, we will not consider non-RDF data sources in this paper and refer the reader to [16] for a survey on knowledge extraction approaches.

Analytical RDF Abstraction Stage. The output of the second stage (*analytical RDF abstraction*) is produced by applying a sequence of various in-stage analytical SPARQL operators on the RDF output produced by the data transformation. We call the sequence an *analyzer* (see Figure 1). Our goal is to enable users to reuse existing analyzers (possibly created by other users) 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 3.4.

Visualization Abstraction Stage. An analytical abstraction is not a suitable data structure for visualization. We want to ensure that a visualization tool is reusable for different analytical abstractions. Building specific visualizers for particular analytical abstractions would not enable such reuse. This is because each visualization tool visualizes particular generic characteristics captured by the analytical abstraction. For example, there can be a tool that visualizes tree structures using the TreeMap technique or another tool that visualizes the same structures using the SunBurst technique. And, another tool may visualize 2-dimensional structures on Google Maps. An analytical abstraction may contain encoded both tree structure as well as 2-dimensional structure. All three mentioned tools 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 a more generic data structure based on Generic Visualization Data Types (GVDTs), listed in Table 2. The GVDTs are inspired by Shneiderman [14] data types. This structure is then what is visualized by visualization tools. This transformation is performed by the visualization abstraction stage of LDVM. Its output (*visualization abstraction*) is produced by a component that performs visualization transformation followed by a set of in-stage visualization operators. We

RDF Vocabulary	**Data Type**	**Visualization Tool**
xsd:int, dc:subject,... (count)	1D	Histogram
wgs84:lat, geo:point,...	2D	Map
visko:3DPointPlot,...	3D	3D Rendering
qb:Observation, scovo:Item,...	Multidimensional	Chart
xsd:date, ical:dtstart,...	Temporal	Timeline, Calendar,...
rdfs:subClassOf, skos:narrower,...	Tree	Treemap, SunBurst,...
foaf:knows,...	Network	Graph,...

Table 2. Generic visualization data types.

call it a *visualization transformer*. To facilitate associating visualization tools to analytical abstractions, we provide mappings from GVDTs to visualization tools but also to RDF vocabularies. This way it is possible to associate input analytical RDF abstractions to GVDTs and then provide or recommend the more suitable visualization tools to deal with them. The mappings are summarized in Table 2. For example, an observation modeled using the the Data Cube Vocabulary (QB) for dimensional data and part of an analytical abstraction can be modeled using the generic Multidimensional datatype and then visualized by any charting tool, which will be based on this visualization abstraction. A visualization transformer can be reused for extracting visualization abstractions from various analytical abstractions. The only requirement is that *the transformer must be compatible with the analytical abstraction*.

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 use GVDTs supported by the visualizer. In other words, *the visualizer must be compatible with a visualization abstraction*. It is necessary to ensure compatibility between analyzers and RDF datasets, between visualization transformers and analytical RDF abstractions, and between visualizers and visualization RDF abstractions. As we show later in Section 3.4, the notion of compatibility is based on ontologies. For example, an analytical RDF abstraction adhering to a given ontology may be processed only by a visualization transformer that supports this ontology. In addition, a visualization transformer is also able to process all analytical abstractions adhering to any other ontology mapped to the supported one thanks to the Linked Data principles, which also enable analyzers, visualization transformers and even visualizers to enrich the processed data with other data from the LOD cloud.

3.3 Sample LDVM Pipeline Instance

Figure 2 shows several sample instances of LDVM. There are 3 analyzers available to users. 2 of them can work with any data source and extract its class hierarchy or property hierarchy (expressed with `rdfs:subClassOf` or `rdfs:subPropertyOf` properties, respectively). The third one, *public spending analyzer*, analyzes the amount of money spent on public contracts of public authorities grouped by EU regions at various levels (countries, regions, municipalities etc.). It requires the data to be represented

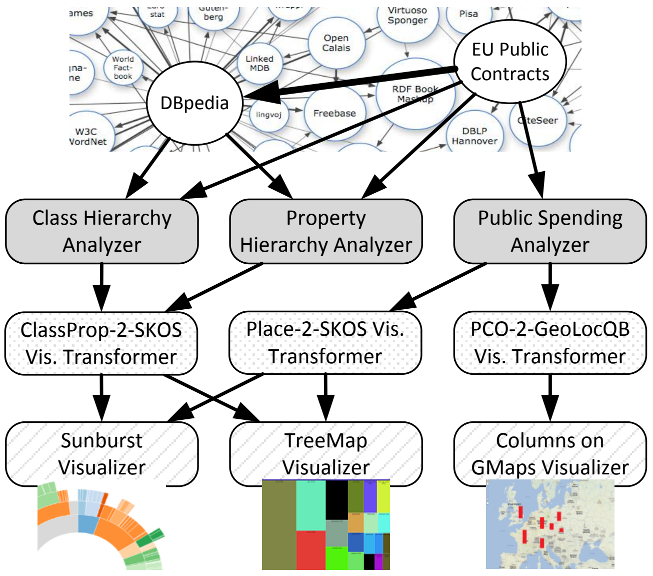
DBpedia
EU Public Contracts
Class Hierarchy Analyzer
Property Hierarchy Analyzer
Public Spending Analyzer
ClassProp-2-SKOS Vis. Transformer
Place-2-SKOS Vis. Transformer
PCO-2-GeoLocQB Vis. Transformer
Sunburst Visualizer
TreeMap Visualizer
Columns on GMaps Visualizer

Fig. 2. Sample analyzers and visualizers using LOD.

according to the *Public Contracts Ontology*[10]. The output contains EU regions represented as instances `s:AdministrativeArea` from `http://schema.org` name space. Regions are organized into a hierarchy using `s:containedIn`.

Further, there are three visualization transformers. *ClassProp-2-SKOS Visualization Transformer* transforms a class or property hierarchy to a visualization abstraction adhering to tree GVDT modeled with SKOS. Each node of the tree is associated with the name of the class or property, respectively, and the number of instances. *Place-2-SKOS Visualization Transformer* transforms a hierarchy of places expressed using `http://schema.org` constructs to tree GVDT modeled using SKOS as well. Each node of the tree is enriched with the name of the respective region and the money spent in the region. The last *PCO-2-GeoLocQB Visualization Transformer* transforms the analytical abstraction containing EU regions each complemented with the measured money spent to 2-dimensional GVDT modeled with QB. The dimensions are GPS longitude and latitude of the capital cities of each region. The items of the 2-dimensional structure are the capital cities with their names and the money spent. It is not necessary that the GPS coordinates are in the analytical abstraction. The transformer enriches the abstraction with GPS coordinates from the LOD cloud (DBpedia or other source) using links that must be present.

Finally, there are three different visualizers. *Sunburst Visualizer* shows a tree GVDT modeled using SKOS as a sunburst view. Similarly, *TreeMap Visualizer* shows a tree GVDT modeled with SKOS as a TreeMap view. *Columns on GMaps Visualizer* shows a 2-dimensional GVDT modeled with QB with two dimensions representing GPS coordinates on a Google Map. Each 2-dimensional item with a label and number is repre-

[10] `http://purl.org/procurement/public-contracts#`

sented as a column placed on the given coordinates. The height of the column depends on the number. The column is enriched with the label.

Various instances of LDVM pipelines can be created using these analyzers, visualization transformers, and visualizers. A sample instance may combine *Class Hierarchy Analyzer*, *ClassProp-2-SKOS Visualization Transformer*, and *TreeMap Visualizer* and can be applied on *DBpedia*. The data flows through the particular LDVM stages are:

- *Source Data:* raw RDF statements from DBpedia including the DBpedia ontology.
- *Analytical RDF Abstraction:* RDF statements stating a label, children and number of instances for each class in the DBpedia ontology.
- *Visual RDF Abstraction:* RDF statements adhering to SKOS representing a tree GVDT of DBpedia classes and their hierarchy.
- *View:* a TreeMap visualization of the DBpedia class hierarchy created by the visualizer.

The data is transformed between various stages as follows. The *Data Transformation* is an identity function since we use RDF data. The *Visualization Transformation* transforms the class hierarchy to a SKOS representation. The *Visual Mapping Transformation* is a function that maps the SKOS hierarchy to a TreeMap The in-stage operators are applied within each stage:

- *Analytical RDF Abstraction:* SPARQL queries that compute the numbers of instances of each class and represent the numbers as RDF statements with classes as subjects (the SPARQL queries form the analyzer).
- *Visual Abstraction Operators:* if there are more sibling nodes with 0 assigned, they are condensed to a single node.
- *View:* zoom in, zoom out.

3.4 Formalization

The core concept of LDVM are reusable components, i.e. analyzers, visualization transformers, and visualizers. They are software components that consume RDF data via their input interfaces. An *analyzer* consumes source RDF data. A *visualization transformer* consumes an analytical RDF abstraction. A *visualizer* consumes a visualization RDF abstraction. The goal is to formally introduce the concept of compatibility of a component with its input RDF data. This formalization can then be easily applied on analyzers, visualization transformers and visualizers. 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*. The signature comprises an ontology, which describes the classes and properties expected in the processed RDF data, and a set of SPARQL queries that further restrict the data a component is able to process. The input signature is then compatible with the RDF data when the ontology of the signature is semantically compatible with the ontology of the RDF data and the SPARQL queries of the signature are evaluated on the RDF data to a non-empty result. As we show later, we define the semantic compatibility of two ontologies only on the base of simple equivalence and sub-type-of mappings between their classes and properties. Our rationale is to provide a simple and lightweight solution, which allows to resolve the compatibility without complex reasoning. In addition, we allow to specify more complex rules of compatibility using SPARQL queries defined by the input signature. While the ontologies can be used to resolve the static compatibility at design-time of a LDVM pipeline instance, SPARQL queries can be evaluated at run-time. To define the concept of compatibility based on input signatures we first introduce a formal simplified representation of an ontology.

Definition 1 (Ontology). *An ontology $\mathcal{O}$ is a triple $(\mathcal{C}, \mathcal{P}, \mathcal{M})$ where $\mathcal{C}$ is a set of* classes*, $\mathcal{P}$ is a set of* predicates*, and $\mathcal{M}$ is a set of* RDF statements *(mappings of the classes and properties to other ontologies). Both classes and predicates are specified using their unique URIs.*

Example 1 (Sample Ontology). Let us demonstrate the definition on two simple ontologies $\mathcal{O}_{pc}$ and $\mathcal{O}_{vcard}$. $\mathcal{O}_{pc} = (\{\texttt{pc:Location}\}, \{\texttt{geo:long}, \texttt{geo:lat}\}, \mathcal{M}_{pc})$. $\mathcal{M}_{pc}$ contains the following statement:

– `pc:Location rdfs:subClassOf geo:SpatialThing`

As we can see, the ontology contains a class for a geographical location and two properties representing GPS coordinates. Moreover, the ontology contains an equivalence mapping of its class to a class in another ontology. $\mathcal{O}_{vcard} = (\{\texttt{vcard:Location}\}$, $\{\texttt{vcard:longitude}$, $\texttt{vcard:latitude}\}$, $\mathcal{M}_v)$ where $\mathcal{M}_v$ contains following statements:

– `vcard:Location rdfs:equivalentClass geo:SpatialThing`
– `vcard:longitude rdfs:equivalentProperty geo:long`
– `vcard:latitude rdfs:equivalentProperty geo:lat`

This ontology definition allows us to define the concept of compatibility of two given ontologies. We first need to define compatibility of two classes and compatibility of two properties.

Definition 2 (Class Compatibility). *Let $\mathcal{O}_s = (\mathcal{C}_s, \mathcal{P}_s, \mathcal{M}_s)$ and $\mathcal{O}_t = (\mathcal{C}_t, \mathcal{P}_t, \mathcal{M}_t)$ be two ontologies. Let $\mathcal{M} = \mathcal{M}_s \cup \mathcal{M}_t$. Let $\mathcal{C} = \mathcal{C}_s \cup \mathcal{C}_t \cup \{$classes used as subjects or objects in statements from $\mathcal{M}\}$. A class $C_t \in \mathcal{C}_t$* is compatible with *a class $C_s \in \mathcal{C}_s$ iff*

– *$C_t = C_s$, OR*
– *$\exists C \in \mathcal{C}$ s.t. C_t is compatible with C and*
  - *$(C_s, \texttt{rdfs:equivalentClass}, C)) \in \mathcal{M}$,*
  - *$(C, \texttt{rdfs:equivalentClass}, C_s)) \in \mathcal{M}$, or*
  - *$(C_s, \texttt{rdfs:subClassOf}, C)) \in \mathcal{M}$.*

A class C_s is compatible with class C_t when they are the same class or there exists a chain of equivalence or sub-class-of mappings between both classes. Therefore, the relation *being compatible with* between two classes is reflexive and transitive but not symmetrical.

Definition 3 (Property Compatibility). *Let $\mathcal{O}_s = (\mathcal{C}_s, \mathcal{P}_s, \mathcal{M}_s)$ and $\mathcal{O}_t = (\mathcal{C}_t, \mathcal{P}_t, \mathcal{M}_t)$ be two ontologies. Let $\mathcal{O}$ denote the union of both. A property $P_t \in \mathcal{P}_t$* is compatible with *a property $P_s \in \mathcal{P}_s$ iff*

– *$P_t = P_s$, OR*

– $\exists P \in \mathcal{P}$ *s.t.* P_t *is compatible with* P *and*
  - $(P_s, \texttt{rdfs:equivalentProperty}, P) \in \mathcal{M}$*,*
  - $(P, \texttt{rdfs:equivalentProperty}, P_s) \in \mathcal{M}$*, or*
  - $(P_s, \texttt{rdfs:subPropertyOf}, P) \in \mathcal{M}$

Compatibility of properties is similar to the compatibility of classes. Again, the relationship is a reflexive and transitive but not symmetrical. Based on the compatibility of classes and properties we can define the compatibility of two ontologies. The definition says that an ontology $\mathcal{O}_t$ is compatible with $\mathcal{O}_s$ iff for each component of $\mathcal{O}_t$ (i.e. each class or property) there is a compatible property in $\mathcal{O}_s$.

Definition 4 (Ontology Compatibility). *An ontology* $\mathcal{O}_t = (\mathcal{C}_t, \mathcal{P}_t, \mathcal{M}_t)$ *is* compatible with an ontology $\mathcal{O}_s = (\mathcal{C}_s, \mathcal{P}_s, \mathcal{M}_s)$ *iff*
– $(\forall C_t \in \mathcal{C}_t)(\exists C_s \in \mathcal{C}_s)(C_t$ *is compatible with* $C_s)$
– $(\forall P_t \in \mathcal{P}_t)(\exists P_s \in \mathcal{P}_s)(P_t$ *is compatible with* $P_s)$

As we can easily prove, again ontology compatibility is reflexive and transitive but not symmetrical.

Example 2 (Compatible Ontologies). In our previous example, `vcard:Location` from the ontology $\mathcal{O}_{vcard}$ is compatible with the class `pc:Location` from the ontology $\mathcal{O}_{pc}$. Also, properties `vcard:longitude` and `vcard:latitude` from $\mathcal{O}_{vcard}$ are compatible with `geo:long` and `geo:lat` from $\mathcal{O}_{pc}$, respectively. Therefore, according to the previous definition, the ontology $\mathcal{O}_{vcard}$ is compatible with $\mathcal{O}_{pc}$.

We further need to formalize a dataset. Basically, a dataset comprises a set of RDF statements, an ontology that describes the structure of the RDF statements and, optionally, their semantics in a form of mappings to other ontologies.

Definition 5 (Dataset). *A dataset* $\mathcal{DS}$ *is a pair* $(\mathcal{D}, \mathcal{O})$ *where* $\mathcal{D}$ *is a set of RDF statements and* $\mathcal{O}$ *is an ontology with classes and properties used as types in* $\mathcal{D}$*.*

We now define *input signatures*. Each LDVM component has an input signature that comprises an ontology and a set of SPARQL queries. Both are optional. The ontology describes the basic required structure of the data that may be processed by the component. The SPARQL queries specify further restrictions on the data.

Definition 6 (Input Signature and Compatibility). *An* input signature $\mathcal{S}$ *is a pair* $(\mathcal{O}, \mathcal{Q})$ *where* $\mathcal{O}$ *is an ontology and* $\mathcal{Q}$ *is a set of SPARQL queries.* $\mathcal{O}$ *specifies the basic required structure of the input RDF data.* $\mathcal{Q}$ *specify further more complex restrictions. We say that* $\mathcal{S}$ is compatible with *a dataset* $\mathcal{DS} = (\mathcal{S}_{DS}, \mathcal{O}_{DS})$ *iff*
– $\mathcal{O}$ *is compatible with* $\mathcal{O}_{DS}$
– *each* Q *in* $\mathcal{Q}$ *returns a non-empty result when executed on* $\mathcal{S}_{DS}$

Example 3 (Analyzer Compatibility with a Dataset). Based on Figure 2 we demonstrate how compatibility of *PCO-2-GeoLocQB Visualization Transformer* with the analytical abstraction created by *Public Spending Analyzer* can be checked. The visualization transformer has an input signature $\mathcal{S}_{cgmap} = (\mathcal{O}_{vcard}, \mathcal{Q}_{cgmap})$. The analytical abstraction created by the analyzer on the base of the *EU Public Contracts Dataset* is a dataset

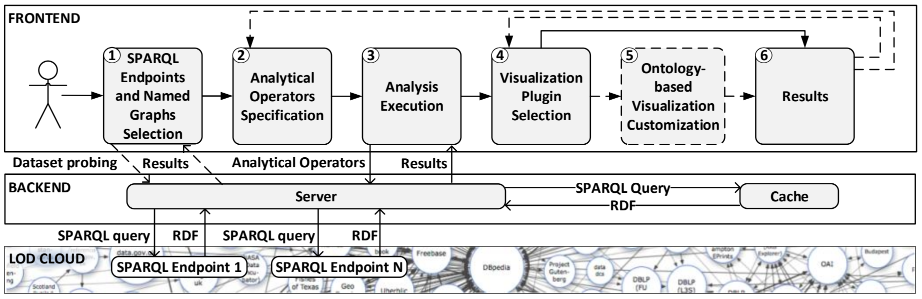

Fig. 3. High-level Linked Data Visualization architecture.

$\mathcal{DS}_{spend} = (\mathcal{D}_{spend}, \mathcal{O}_{pc})$. To check the compatibility we first need to check the compatibility of $\mathcal{O}_{vcard}$ with $\mathcal{O}_{pc}$ (as performed in Example 2). Then we execute queries in $\mathcal{Q}_{cgmap}$ against $\mathcal{D}_{spend}$. Our sample query checks whether there are instances of `vcard:Location` in the dataset that have the values of `vcard:longitude` and `vcard:latitude` and also other properties that are the same for all the instances and have integer values, which can then be used for rendering by a visualizer. If the query returns a non-empty result, the visualization transformer is compatible with the analytical abstraction and and the visualizer can be used.

Let us note that we could replace the ontology with corresponding SPARQL queries. However, this would prevent us from resolving a part of the compatibility statically, which could be a discomfort for users who built a LDVM pipeline instance.

4 Implementation

Based on LDVM, we implemented a comprehensive two-level prototype. The two levels correspond to two levels of visualization detail from Shneiderman's visual information seeking mantra: "*overview first, zoom and filter, then details on demand*" [14]. Figure 3 shows the general architecture of our LDVM implementation.

Step 1 corresponds to the LDVM *source RDF and non-RDF data* stage, in which a user can enter SPARQL endpoints and select graphs to visualize. The LDVM implementation may probe the selected SPARQL endpoints based on the formalism of compatibility introduced in Definition 6 to determine which analyzers can be offered to the user in the next step. In *step 2* the user selects an analyzer from the offered list. (S)he can also customize the analyzer or create a completely new one. *Step 3* performs the data analysis specified by the analyzer. The analysis is sent to the server, executed and the LDVM *analytical RDF abstraction* is returned. This involves querying of the specified SPARQL endpoints, which may involve SPARQL query caching mechanism for performance optimization, and synthesis of data gathered from the individual SPARQL endpoints involved. In *Step 4* the user chooses a visualizer to visualize the obtained analytical abstraction. First, the user chooses what structure encoded in the analytical abstraction should be visualized. This corresponds to choosing a suitable *visualization transformer*. Then, he chooses a suitable *visualizer* to visualize the *visualization abstraction* created by the transformer. The optional *step 5* involves optional customization of the specified visualization (e.g. shapes, colors, etc.). In other words,

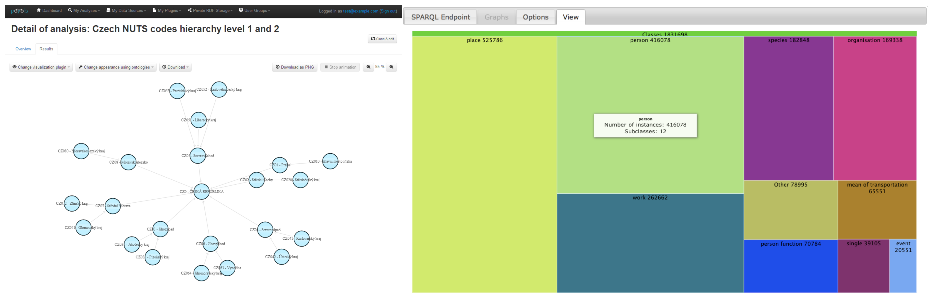

Fig. 4. Implementation screenshots: On the left Payola graph visualization an analysis result, on the right LODVisualization TreeMap of DBPedia class hierarchy

the user specifies the *visual mapping transformation* in this step. Finally, in *step 6* our LDVM implementation runs the chosen visualizer (i.e. creates a *view*) and presents it to the user. The user can then execute LDVM *view operators*, which correspond to zooming, scrolling etc. The arrow from step 6 to step 4 represents a possible change or customization of the used visualizer over the same data. The arrow from step 6 to step 2 represents a change of level of detail. This may be the zoom-in from Shneiderman's mantra from the *overview* level to the *detail on demand* level. Also, the change of detail can be in the opposite direction, i.e. from a detailed view to its context in the overview. In our case, this means transition between our two parts of our LDVM implementation. Let us now introduce our implementations on the respective levels.

Overview – LODVisualization[11] allows to connect different datasets, different data analysis and different visualizations according to the LDVM in a dynamic way. These visualizations allow users to obtain an overview of RDF datasets and realize what the data is about: their main types, properties, etc. In LODVisualization users can enter or select a SPARQL endpoint and select the graphs to visualize (first step of LDVM model). Then, the compatibility between analyzers and datasets is checked in order to determine which of them are available (second step). Once the analyzer has been executed, the results are stored into a visual abstraction, which corresponds to the third step of the model. Finally, users can visualize the results using different visualizers depending on their compatibility (fourth step). Our implementation includes analysis such as the class hierarchy, property hierarchy, SKOS concepts hierarchy, properties connecting two classes, etc. The results can be visualized using techniques such as tables, treemaps, charts, maps, etc. Since being based on the LDVM, LODVisualization is easy to extend with additional analysis and visualization techniques. Figure 4 shows a treemap with the DBpedia class hierarchy generated with LODVisualization. LODVisualization was integrated with Payola to provide users with easy transition between the two levels of visualization details. When exploring a dataset, there is a link provided that transfers all relevant information (SPARQL endpoint, graphs, classes, properties) to Payola and lets users continue analyzing data in Payola.

Details on demand – Payola[12] [13] is a general framework for analysis and visualization of RDF data and in contrast to *LOD visualization* focuses on the *details on*

[11] `http://lodvisualization.appspot.com/`
[12] `http://live.payola.cz`

demand part of Shneiderman's mantra. Its workflow consists of two main parts. *First*, a user creates an analysis. An analysis starts with SPARQL endpoints selection and then consists of individual connected *analytical SPARQL operators*, which have an RDF graph on their input, do a transformation and send the transformed RDF graph to their output. Payola is a framework, so the concrete operator functionality and usability is the responsibility of plugins creators. We created a library of default operators that represent typical parts of SPARQL queries. These operators are integrated into SPARQL queries and executed on the selected SPARQL endpoints. Once an analysis is defined, it can be executed in a *second* part creating the analytical abstraction, which can be then visualized using different visualizers. The analytical abstraction in Payola is always an RDF graph. Concrete visualizations are again implemented via visualization plugins. An example can be seen on the left side of Figure 4. While the graph visualization is generically applicable, the chart visualization has certain requirements on the input RDF graph (i.e. LDVM visualization abstraction). It has to contain resources of a certain type, each one equipped with a label and property with a value. When the input graph meets this criteria, i.e. it is a visualization RDF abstraction compatible with the chart visualizer, it can be visualized as a chart. The vision for Payola is that for each problem domain, a domain expert creates custom LDVM analyzers for that domain and also custom LDVM visualizers appropriate for that domain. Payola was integrated with LODVisualization to allow an easy transition from the details on demand visualization level back to the overview level. Whenever a SPARQL endpoint and a graph is selected in Payola, a link to LODVisualization enables the user to explore the selected dataset on the overview level.

5 Evaluation

Our evaluation consists of two parts. First, we evaluated our implementation with end users performing several tasks. Second, we have compared our prototypes with the other approaches described in the Related Work section.

User evaluation We asked our test users to perform tasks using our prototype LDVM implementation – *overview* implemented by LODVisualization and *details on demand* implemented by Payola. Expert users, who know the RDF and SPARQL, can create visualizations on both levels of detail and implement them in the tools directly. However, our goal was to show that Linked Data visualization does not have to require such expertise. This is consistent with our vision where expert users create complex configurable visualizations with user-friendly parameters, which are used as black boxes by lay users. Therefore, we created some configurable sample analyzers and visualizers and we let lay users use and customize them to perform various tasks. We asked 16 users to assess the difficulty of the tasks on a scale from 0 – not able to solve, 1 – difficult to 5 – easy and to record the time required to complete each task. The following scenarios and tasks were defined for the user evaluation.

Scenario 1: Generic tasks consists of 8 tasks than can be done using DBPedia and LODVisualization: (1) five most frequently used classes, (2) five most frequently used properties, (3) five least frequently used classes, (4) five least frequently used properties (5) most generic classes in the class hierarchy, (6) most generic concepts in the SKOS

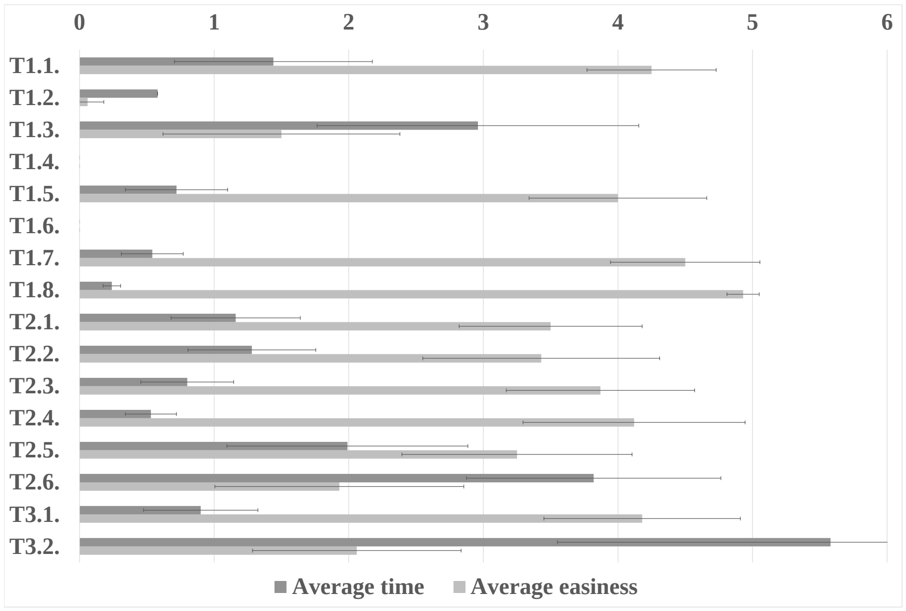

Fig. 5. Average easiness (0-5, higher is better) and time in minutes (lower is better) including standard deviations for each task (N = 16)

concept hierarchy, (7) five instances of class `dbo:Village` with the highest indegree, (8) five instances with the highest outdegree. Next, we prepared a general scenario using DBpedia and a domain specific one regarding public procurement.
Scenario 2: DBpedia The overview level tasks are: (1) Explore city and country classes in the class hierarchy. (2) Properties connecting cities and countries. (3) Person classes in the class hierarchy and all subclasses. (4) Properties connecting persons and cities. The details on demand level tasks are: (5) Cities with a population of more than 2 million and their countries. Use the prepared analysis in Payola and visualize using Gravity visualizer. (6) Cities in the Czech republic with a population of more than 50.000 and soccer players born there. A similar analysis is available in Payola showing cities in Germany with population more than one million and artists born there. Clone this analysis and change it accordingly. Use the circle visualizer.
Scenario 3: Public procurement Here, only details on demand level tasks are performed: (1) Really expensive Czech public contracts (i.e. cost more than 20 billion CZK). Use the prepared analysis in Payola. Use the triple table visualizer. (2) Hierarchy of Germany NUTS regions of level 1 and 2. Clone and change a similar analysis in Payola for Czech NUTS regions. Use the Gravity visualizer.

The results are summarized in Figure 5.Users were not able to perform tasks 1.2, 1.4 and 1.6. They knew how to perform these tasks with LODVisualization but the SPARQL queries to obtain the property hierarchy and the SKOS hierarchy timed out. These analyses can be executed correctly over smaller datasets, but they do not scale yet

Tool	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**2.1**	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**3.1**	**3.2**
Tabulator	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Explorator	-	-	-	-	-	-	-	-	-	✔	-	✔	✔	✔	✔	✔
D. Pipes	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Fresnel	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Exhibit	-	-	-	-	-	-	-	-	-	-	-	-	✔	-	✔	✔
Fenfire	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sgvizler	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Rhizomer	✔	-	-	-	✔	✔	-	-	✔	✔	✔	✔	-	-	-	✔
LODVis	✔	-	✔	-	✔	-	✔	✔	✔	✔	✔	✔	-	-	-	-
Payola	-	-	-	-	-	-	-	-	-	-	-	-	✔	✔	✔	✔

Table 3. Visualization capability comparison with other tools.

to large datasets such as DBpedia. It is important to note that the execution times depend on the dataset size as well as on the availability of SPARQL endpoints or servers, and therefore they can produce a timeout. The average easiness rating for overview tasks is 3.67 and 2.82 for details on demand tasks. Complex tasks such as 2.6 and 3.2 were difficult for users and still required much time. However, tasks 2.5 and 3.1 were easier and quicker to solve. Overall, 12 out of the 16 partially quite complex tasks can be solved in reasonable time (only thee task require more than 3 minutes).

Comparison with other tools With Tabulator, Fenfire or Fresnel it is not possible to perform any of the tasks because they only allow to display a concrete resource or a set of resources and their properties. Explorator can perform details on demand tasks by combining subsets of resources and filtering them. However, it is not possible to perform any overview task. Some of the details on demand tasks can be performed with Exhibit and Rhizomer through their facet navigation. Rhizomer also provides class and SKOS concepts overviews that allow to perform some of the overview tasks. Finally, using Semantic Pipes or Sgvizler it is possible to perform all tasks, but SPARQL and programming skills as well as extensive domain knowledge are necessary. Table 3 shows the comparison with other tools. In summary, this evaluation shows, that our LDVM implementation allows users to solve the by far largest variety of visualization tasks without programming or domain knowledge.

6 Conclusions and Future Work

We presented the Linked Data Visualization Model (LDVM) that can be applied to rapidly create visualizations of RDF data. It allows to connect different datasets, data extractions and visualizations in a dynamic way. We have implemented the model in two complementary applications which allow to create generic visualizations for RDF. In our implementations we provide visualizations that support the overview and details-on-demand tasks proposed by Shneiderman.

The LDVM is the first step on a larger research agenda aiming at automatizing the visualization of semantic data. In future work we focus on complementing the Linked Data Visualization Model with an ontology that can help during the matching process between data and visualizations, capture the intermediate data structures that can be generated and choose the visualizations more suitable for each data structure. Regarding the implementations of the model, we plan to improve the analyizer component to make it possible to execute complex SPARQL queries without timing out, e.g. using a cache memory or decomposing the query into simpler sub-queries. We also focus on extending the library of data analyzers and visualizations. We aim to integrate these components into a general dashboard for visualizing and interacting with Linked Data through different visualizations as well as other exploration and authoring components.

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, and R. Garcia. The linked data visualization model. In *International Semantic Web Conference (Posters & Demos)*, 2012.
5. S. K. Card and J. Mackinlay. The structure of the information visualization design space. In *IEEE Symp. on Information Visualization*, INFOVIS '97, 1997.
6. S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Academic Press, London, 1999.
7. 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.
8. E. H. H. Chi, P. Barry, J. Riedl, and J. Konstan. A spreadsheet approach to information visualization. In *IEEE Symposium on Information Visualization '97*, 1997.
9. A.-S. Dadzie and M. Rowe. Approaches to visualising Linked Data. *Semantic Web*, 2(2):89–124, 2011.
10. M. C. F. de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 9:378–394, 2003.
11. F. Frasincar, R. Telea, and G.-J. Houben. Adapting graph visualization techniques for the visualization of rdf data. In *Visualizing the Semantic Web*, 2006.
12. V. Geroimenko and C. Chen, editors. *Visualizing the Semantic Web*. Springer, 2002.
13. T. Hastrup, R. Cyganiak, and U. Bojars. Browsing linked data with fenfire, 2008.
14. J. Klímek, J. Helmich, and M. Nečaský. Payola: Collaborative Linked Data Analysis and Visualization Framework. In *10th Extended Semantic Web Conference (ESWC 2013)*. Springer, 2013.
15. B. Shneiderman. The eyes have it. In *IEEE Symposium on Visual Languages*, 1996.
16. C. Stadler, J. Lehmann, K. Höffner, and S. Auer. LinkedGeoData: A Core for a Web of Spatial Open Data. *Semantic Web Journal*, 2011.
17. J. Unbehauen, S. Hellmann, S. Auer, and C. Stadler. Knowledge extraction from structured sources. In S. Ceri and M. Brambilla, editors, *SeCO Book*, volume 7538 of *Lecture Notes in Computer Science*, pages 34–52. Springer, 2012.