Customising LOD views: a declarative approach

Alice Graziosi
DASPLab - Digital and Semantic Publishing Laboratory
DISI - Department of Computer Science and Engineering, University of Bologna
Bologna, Italy
alice.graziosi@studio.unibo.it

Angelo Di Iorio
DASPLab - Digital and Semantic Publishing Laboratory
DISI - Department of Computer Science and Engineering, University of Bologna
Bologna, Italy
angelo.diiorio@unibo.it@unibo.it

Francesco Poggi
DASPLab - Digital and Semantic Publishing Laboratory
DISI - Department of Computer Science and Engineering, University of Bologna
Bologna, Italy
francesco.poggi5@unibo.it

Silvio Peroni
DASPLab - Digital and Semantic Publishing Laboratory
DISI - Department of Computer Science and Engineering, University of Bologna
Bologna, Italy
silvio.peroni@unibo.it

Luca Bonini
DASPLab - Digital and Semantic Publishing Laboratory
DISI - Department of Computer Science and Engineering, University of Bologna
Bologna, Italy
luca.bonini@studio.unibo.it

ABSTRACT

This paper is about web applications to browse and efficiently visualise large Linked Open Dataset (LOD). The focus is on the customisation of LOD views over semantic datasets also for non expert users. The paper presents the motivation and the details of a visual data format and a chain of tools to easily produce and customize such visualisations. Two proofs of concepts are also presented in order to demonstrate the feasibility and flexibility of our approach.

CCS CONCEPTS

•Information systems → Resource Description Framework (RDF); Web applications; **•Human-centered computing →** *Graph drawings; Information visualization;*

KEYWORDS

Semantic Web, RDF, SPARQL, Linked Open Data visualisation

ACM Reference format:
Alice Graziosi, Angelo Di Iorio, Francesco Poggi, Silvio Peroni, and Luca Bonini. 2018. Customising LOD views: a declarative approach. In *Proceedings of ACM SAC Conference, Pau,France, April 9-13, 2018 (SAC'18),* 8 pages.
DOI: xx.xxx/xxx_x

1 INTRODUCTION

The Web is continuously changing: from a Web of Documents, in which the content is primarily consumed by *human readers*, we are moving towards a Web of Data, in which the information is published in machine-readable format so that *software agents* can collect, query and merge data embedded in Web pages [6]. The Linked Open Data (LOD) initiative[1] played, and still plays, an important role in such a shift. Linked Data are structured and interlinked data that can be consulted through semantic queries: data are stored as RDF, released under an open license and interconnected by typed links. While important issues such as the quality of LOD are debated and studied [9], the amount of data available as LOD[2] is already huge and constantly increasing [5]. Moreover, many tools for converting structured datasets in RDF and automatically linking them to existing LOD have been developed [15, 20]

At the end of the day, the LOD datasets are meant to be consumed by *human readers* too. Mash-ups and domain-specific applications require data, and show them to the final consumers in a way that hides the complexity of the underlying data model. Though, there is still room for improvement in the processes for producing such visualizations.

Users have two options. The first one is using tools for building visualisations directly, for instance LD-VOWL[3] [17] [22]. These tools are very easy to use, take raw data as input and produce user-friendly visualization; on the other hand, they provide designers a limited set of options and very low flexibility. The second approach consists of coding the visualizations by exploiting graphic libraries and modules, such as D3.js[4] and Cytoscape.js[5]. These libraries are very powerful but technical competencies and programming skills are still required. A rather deep knowledge of Semantic Web and LOD technologies is also needed to collect and combine data. Reusability is another critical issue: many optimal visualizations of LOD exist, but they are designed as task-specific solutions so they are not easily reusable in other contexts.

SAC'18, Pau,France

DOI: xx.xxx/xxx_x

[1] https://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[2] Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/.
[3] http://vowl.visualdataweb.org/ldvowl.html
[4] https://d3js.org/
[5] http://js.cytoscape.org//

In this paper we investigate trade-offs between these two approaches. We are seeking solutions for supporting non-programmers to build and customise LOD visualizations, and here we present a conceptual model to build such visualizations and two proof-of-concept implementations built on top of that model. A JSON-based data format, which makes the overall implementation modular and easily reconfigurable, is introduced too. Some preliminary analysis showed that our approach is promising.

The paper is then structured as follows. We review the most powerful Web-based tools for building LOD visualizations in Section 2. Then we introduce our approach in Section 3; some details about our data model and format are provided in Section 4. Section 5 shows our systems, while some discussion and future works conclude the paper in Section 6.

2 RELATED WORKS

In this section we present a brief summary of the state of the art of Linked Open Data (LOD) visualisation and exploration tools. Building systems to generate LOD visualisation allow easy access to the content of datasets and information extraction through visual data exploration. Unfortunately, most of the existing tools are intended for Semantic Web expert users. Usually, visualisations offered by those systems are difficult to customise or not customisable at all. In order to facilitate the user to consult data in a simple and user-friendly way, the tools should provide novel methods to filter, cluster and, in wider terms, to personalise data visualisation by having it tailored to user preferences. Further details concerning existing visualisation tools are discussed in a comprehensive study [12].

Before going on, it is worth underlining that we take into consideration only web-based tools that provide interesting interactivity and/or customisation features for visualising RDF datasets. The tools that concern solely ontologies (or born as plug-in for desktop applications such as ontology editors) are not taken into account in this work. It is also interesting to point out that the types of statements managed by the evaluated tools are ABox information. A knowledge base can be represented conceptually as a combination of terminologies (TBox) and assertions (ABox). TBox statements describe a set of concepts and their properties, while ABox statement describe entities and values.

One of the most important tools is LodLive[6] [8]. LodLive is a web application for browsing the Web of Data. This open source tool is entirely developed with JavaScript and it is freely embeddable on other applications. LodLive allows the user to pass from one endpoint to another because different resources from different SPARQL endpoints are interconnected to each other exploiting the interconnection capability intrinsic in the nature of LOD. LodLive displays data in an excellent graph visualisation which can be expanded incrementally to explore the desired resource in details.

The VOWL-based toolset is one the most representative and powerful. VOWL (Visual Notation for OWL Ontologies)[7] [18] is a visual language that uses graphical primitives in combination with color scheme to define graphical objects for most of the OWL elements. Those objects are combined together in a force-directed graph view to represent ontologies. VOWL-based tools like WebVOWL [16] and LD-VOWL [17] provide users with interfaces and wizards to build visualisations incrementally, starting from a publicly available data source or a local dataset.

WebVOWL [16] offers sophisticated views of data and users can choose among a set of graphic objects and properties. They are also able to customise a limited set of properties of these objects. LD-VOWL[8] [17] [22] is another implementation of VOWL that extracts schema information from SPARQL endpoint's data and display it in an overview graph visualisation. LD-VOWL allows personalization in terms of Class-node/Class-node and Class-node/Type-node distance, layout and external elements colours. The classes and properties to be rendered can be filtered too (as shown in the first column of the table).

GraphVizDb[9] [3] [4] is a scalable tool for efficient LOD visualization and graph exploration over very large RDF graphs. GraphVizDb also offers some degree of personalisation. Users can select if they want to show edge labels or node values; they can also set which type of zoom or focus apply to the final visualisation.

RDF-GL [13] is a SPARQL-based graphical query language for RDF models that enables users that are not Semantic Web experts to create queries visually. In particular, they arrange and connect symbols on a virtual canvas, so that complex queries over large and distributed databases can be performed quickly and efficiently, without requiring any knowledge of a formal query language.

Efforts have also been made to create visual query languages in the context of ontologies. GLOO [11] is a visual query language for OWL-DL ontologies that hides the complexity of a DL-based query languages (e.g. DIG, nRQL) to non-expert users and supports them to query OWL ontologies.

Other two tools are worth mentioning: UDUVUDU and Linked Data Reactor. UDUVUDU[10] [19] provides users with a template-based language to match patterns in input datasets and generate HTML-based visualisations. The language is still quite difficult for non-programmers, but it guarantees high flexibility and power. In fact, it can be used to produce views with customised static and dynamic objects, as well as for views of aggregated data.

Linked Data Reactor[11] [14] is equally powerful. It is a framework to develop flexible and reusable widgets for Linked Data applications. Some coding is still required for the users but the system comes with a wide range of reusable UI web components and widgets which can be assembled in new visualisations. New widgets are also continuously added by the developers.

3 GENERATING LOD VIEWS: ENOUGH FLEXIBILITY?

The tools surveyed in the previous section, together with many others not listed here, provide simple interfaces to load LOD datasets from external sources (SPARQL endpoints or dumps) and select one visualisation among a set of predefined options and widgets. The process is simple and fast and the results are effective.

[6] http://lodlive.it/
[7] http://vowl.visualdataweb.org/
[8] http://vowl.visualdataweb.org/ldvowl.html
[9] http://83.212.97.26:8080/graphVizdb/
[10] http://dbpedia.exascale.info/
[11] http://ld-r.org/

However these solutions are not fully flexible. Mastering technologies is still required if the users want to create personalised views and have full control on the final output.

Such impression was confirmed by some preliminary interviews we conducted to a group of eight people involved in two projects we are participating to.

Our first case study was the Semantic Lancet Project[2, 10]. The goal of the project was to create an experimental LOD about scholarly publications. In particular, it took as input all papers published in the Journal of Web Semantics by Elsevier and produced a dataset that contains, for each paper, bibliography, abstract and citations compliant with the Semantic Publishing and Referencing (SPAR) Ontologies[12]. These ontologies describe the publishing domain in detail and allow designers to build complex and expressive structures. One of the main issues faced in this project is making the resulting dataset understandable to the average user. For instance, a typical problem is producing a bird-eye overview of the dataset that just shows a summary of the whole citation network, and that can be expanded to show details on user's demand.

Similar problems were also evident in another project promoted by the Italian Academic And Research Network (GARR Consortium[13]). The objective of this project is investigating the availability of semantically enriched datasets that describe computers networks, and developing mechanisms to display such datasets.

These two projects gave us a chance to collect some issues related to building visualisations and to communicating data. These issues have been collected and analysed as described in [12], and are summarised here as a set of requirements that will be used in the rest of the paper:

- **R1: Easy Data Selection**. The user should be provided with an easy and possibly graphical way to build SPARQL queries and execute them against SPARQL endpoints in order to choose the desired data to be shown. Strategies are needed to help users to explore large and complex datasets. An incremental approach, in which only part of the dataset is shown to the users so they can select a subset of the entities, and in case move to others, seems to be a viable and effective solution. The key aspect is that users should not have to know neither semantic query language nor the details concerning the underlying dataset structure.
- **R2: Data Simplification**. The users also expressed a strong need for tools that make visualisations simple, even when built on top of complex data models. The complexity of the ontological structures in LOD datasets is often directly reflected on the visual representations of such datasets. For instance, two entities in a graph may be linked by a long chain of properties, which is not readable nor easily understandable by the users. Note that such property chains can be simplified in SPARQL (with schema reduction techniques[14]), but still users need support while performing this complex task.
- **R3: Easy View Selection**. Once data are chosen, users requires simple interfaces for associating visualizations to these data. In particular, they noted that multiple coordinated visualizations are effective, and expressed the need to easily combine atomic visual artifacts into even more complex views. In fact, the users should be able to associate graphic objects (i.e. the primitives and properties in our terminology) to each element selected from the dataset.
- **R4: Static Objects Customisation**. These objects composing the view should be customisable, allowing users to manipulate their aspect (e.g. the shape, colour, size, etc.) and, consequently, the way they convey the meaning. Surprisingly enough, such a customisation is not supported by most of the surveyed tools: our testers confirmed that most tools don't meet such requirement. We were also suggested to extend the set of basic 2D shapes to include also generic images (for instance dynamically loaded from DBpedia[15] dataset).
- **R5: Dynamic Objects Customisation**. The ability to change *dynamically* the graphical properties of the objects according to the properties of the entities they represent was also outlined in our interviews. Most existing solutions implement a static mapping instead, in which all the instances of the same class (in the input dataset) are mapped to the same graphical object. Providing users a simple way to customise objects dynamically, and making views automatically updatable, would be very effective.
- **R6: Support for alternative clustering**. Grouping related items in distinct clusters is fundamental for LOD datasets. Our interviews also suggested that tools should support alternative ways of organising groups of entities. The 'traditional' way of showing clusters in a LOD is to have one central node representing the cluster connected to one node for each item in the cluster. A more intuitive visualisation is, for example, to group all elements close to each other, in a well identified area that contains all nodes representing the members. To the best of our knowledge, such clusters have not been used in combination with graph-based views to display LODs. Having the chance of interactively group data into cluster by selecting different properties is a further customisation that users could benefit from.

The focus of this paper is on data selection, view selection and static object personalisation. In fact, in next Section a customisable and flexible data format is presented. By having an editable data format users can decide on their own how to graphically display semantic data.

4 A DECLARATIVE APPROACH TO BUILD LOD VIEWS

Starting from the requirements discussed so far we designed a modular and declarative approach, on top of which we built some prototypes able to build LOD visualizations automatically and to support users in the customisation of such visualizations.

Figure 1 shows our solution. The process takes as input an RDF dataset and produces a visualisation in two phases: data selection and view generation. In the *data selection* phase, the user can select

[12] http://www.sparontologies.net/
[13] https://www.garr.it/en
[14] SPARQL property path feature deals exactly with that issue of property summarization; https://www.w3.org/TR/sparql11-query/#propertypaths.
[15] http://dbpedia.org

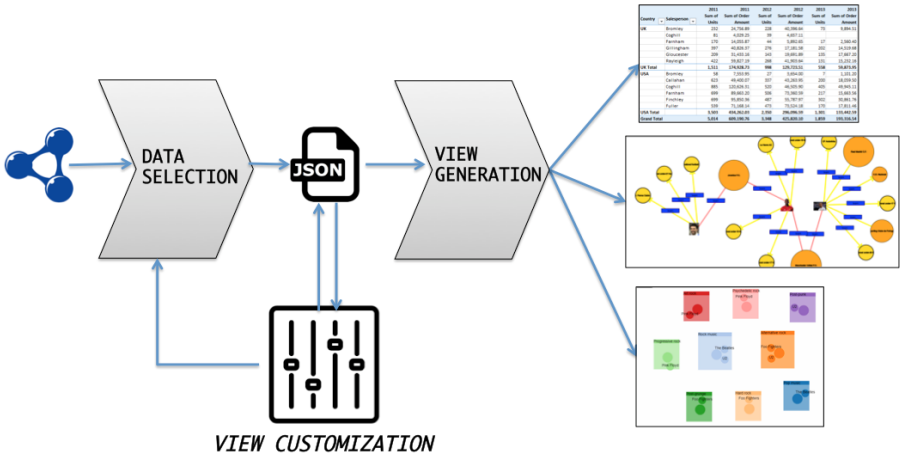

Figure 1: An overview of our declarative approach. The process for generating visualisations is organised in two main phases: data selection and view generation. A JSON-based document is used to specify how the selected data should be rendered visually.

the input data, filter them and reduce them to a simplified form (requirement R1 and R2). The latter step is actually not implemented yet.

The selected data are eventually converted in multiple alternative views, that users can select though a simple interface (R3). The *view generation* phase consists of actually generating the output for the final readers.

The overall process is driven by a declarative manifest, shown in the central part of the picture and serialised in JSON format, which describes both the data and the visualization. In particular the manifest lists all properties of all objects in the visualization, both static and dynamic (requirements R4 and R5), and how they are organised and clustered (requirement R6). The user can adjust the manifest and run the *view generation* process again to easily produce new visualizations.

In this section we go into the details of the manifest, presenting a visual data format called JLO (Json Layout for querying Ontology), which is one of the contribution of this work.

JLO has been designed in order to be easy to understand and machine readable. The structure of a JLO instance can be described in details and validated with a JSON Schema[16] vocabulary[17].

The most important thing to point out is that JLO is in effect a personalisable template for the visualisation of entities and properties. The current JLO version is focused on graphs. The outcome of the visualisation based on a JLO file is a graph composed by nodes (for RDF entities) and links (for RDF properties). However, it has been designed to be extensible. For example, new layouts, style attributes and behaviours can be added to the current format.

A JLO file is composed by four arrays: one for describing nodes, one for links, one for templates (optional) and one for CSS classes (optional). Nodes array refers to all of the instances of all type of classes in a result set of a query. Links array contains all the labelled arcs that connect the resources mentioned above.

[16] http://json-schema.org
[17] https://github.com/LucaBonini/JLO/blob/master/README.md

According to JLO template specification, the shape of nodes can be chosen among the following six options: circle, rectangle, ellipse, polygon, image (JPEG or PNG) and a personalised one. For circle, rectangle, ellipse and polygon it is needed to indicate a size and a colour. Also other graphical aspects can be personalised: for instance, the border of the shapes - except for images - can be personalised in thickness, colour, etc.. Also some interactive behaviour can be personalised. For example, the mouseover event can trigger a change in some graphical properties of a node, such as colour and thickness. Similarly, also the links are customisable in colour, border, arrow direction, etc..

Let's go into some details about the JLO specification. Each node item in the nodes array contains several fields:

- *id*: a textual identifier for the source node. Usually it corresponds to the URI of the represented resource
- *label*: a label for the node
- *size*: an integer that specifies the size of shape for the current node. Allowed values range from 1 to 100.
- *popup_mo*: a string (among "resource", "label" and "comment") that specifies what textual field must be shown in the tooltip
- *mouseover*: a string that specifies a behaviour for the mouseover event
- *color_mo*: a string that associates a colour to the mouseover event
- *cssclass (optional)*: a CSS class for the node from CSS class array. This is the templating mechanism used by JLO, which is borrowed from CSS. Multiple properties can be specified as a CSS class array and referenced by nodes
- *comment*: a comment about the node
- *shape*: a string among "circle", "rectangle", "ellipse", "polygon", "image" and a personalised one (from the template array), used to specify the shape that should be used to draw the current node

- *sstyle (optional)*: an array that contains all the style properties of the node such as *fill* (i.e. a string for the node colour), *stroke* (i.e. a string for the border color), *stroke_width* (i.e. an integer for the border thickness), *stroke_dasharray* (i.e. an integer for dotted line thickness) and *text_color* (i.e. a string for label color).

Each link item in the links array contains several fields:

- *source*: the id of the source node (i.e. the URI of the source)
- *target*: the id of the target node (i.e. the URI of the target)
- *cssclass (optional)*: a the of CSS class from the CSS classes array
- *sstyle (optional)*: an array that contains all the style properties of the link such as *fill* (i.e. e string for link colour), *stroke_width* (i.e. an integer for border thickness), *stroke_dasharray* (i.e. an integer for dotted line thickness) and *direction* (i.e. "forward", "backward" or "both")
- *label*: a label for the link

Templates can be specified and tailored to user preferences. They can be used to customise the style and the shape of the nodes. A template item in the templates array is structured as follows:

- *name*: a string for the template name
- *shape*: a string among "circle", "rectangle", "ellipse", "polygon" or "image" to specify the shape to display
- *fill*: a string for the node colour
- *size*: an integer that specifies the shape size
- *stroke*: a string for the border color
- *stroke_width*: an integer for the border thickness
- *stroke_dasharray*: an integer for the dotted line thickness
- *text_color*: a string for the colour of the label

CSS classes can be tailored to user preferences and used to customise the visualisation of nodes and links. A CSS class item within the classes array is structured as follows:

- *classname*: a string for the class name
- *fill*: a string for the node colour
- *stroke*: a string for the border color
- *stroke_width*: an integer for the border thickness
- *stroke_dasharray*: an integer for the thickness of the dotted line
- *text_color*: a string for the color of the label

5 FROM MODEL TO PROTOTYPES

In this section we present two projects as a proof of concept of our declarative approach: a JLO - based web engine (that is able to process JLO documents and create visualisations) and GIG - Generating interfaces for RDF graphs (an interactive web-based tool based for creating visualisations from LOD)

5.1 JLO - based web application

A web application[18] has been developed in order to display visualisations based on semantic data and expressed in the JLO format. The app provides a graphical way to render JLO documents, which have been created on the basis of a series of SPARQL [21] query to the DBpedia [1] endpoint. The web app is aimed at the visualisation of semantic data intended to non technical audience.

[18]Demo is online at http://eelst.cs.unibo.it/lbonini/.

The idea behind the web application is made of four steps:

- **Query creation and execution**: users can build a SPARQL query using a visual query builder and then run it against a semantic dataset in order to get the desired data. In such a way no knowledge of the SPARQL query language is required to the final users.
- **Layout personalisation**: the customisation of the visualisation can be performed modifying the JLO template by using a graphic interface. This step has not been implemented yet but it is worth noting that the JLO format has been set up to be easily editable. It is important to notice that this feature will definitely help non expert user to make customisable visualisation over semantic dataset, so we intend to develop it as the next step of our project.
- **From result set to JLO file**: the translation of the SPARQL query result set into a JLO format file is made by using a Python script.
- **From JLO file to visualisation**: a JavaScript script takes as input the JLO file and displays the contents as a force-directed graph layout exploiting the extremely powerful D3.js[19] framework [7].

The demo web application implements only the last two steps. Nevertheless, three example queries are *hardcoded* and the three respective JLO files are parsed and displayed. Two of those examples come with the demo and are shown in Figure 2 and in Figure 3 above.

5.2 GIG - Generating interfaces for RDF graphs

GIG[20] [21] is an interactive web-based application for the visualisation of RDF dataset.

The purpose of this web application is to create a domain independent environment that allows users to make customisable visualizations of LODs, facilitating the access to semantic-enriched information for non-expert users.

The data visualisation workflow is based on four steps:

- **Select a data source**: user can select a knowledge base -a SPARQL endpoint (like the well known DBpedia[22]) and a graph- from a dropdown list menu.
- **Query building**: users can filter the desired data from the knowledge base using a visual SPARQL query builder, without having to know neither semantic query language nor the dataset structure. Users start exploring the selected dataset by choosing the class of entities (e.g. Band) to display. Then they select one among its datatype property (e.g. years active) and one of its object property (e.g. the membership, which connects to individuals of another class). Multiple properties can be selected. In the faceted visual query builder shown in Figure 6, for example, users also selected other object properties (e.g. the musical genre).

[19]D3 is a JavaScript library for visualising data with HTML, SVG, and CSS, https://d3js.org/

[20]A demo of the tool is available online at http://eelst.cs.unibo.it:8092/.

[21]An extended abstract can be found at: https://womencourage.acm.org/wp-content/uploads/2017/07/womENcourage_2017_paper_55.pdf?189db0

[22]http://wiki.dbpedia.org/

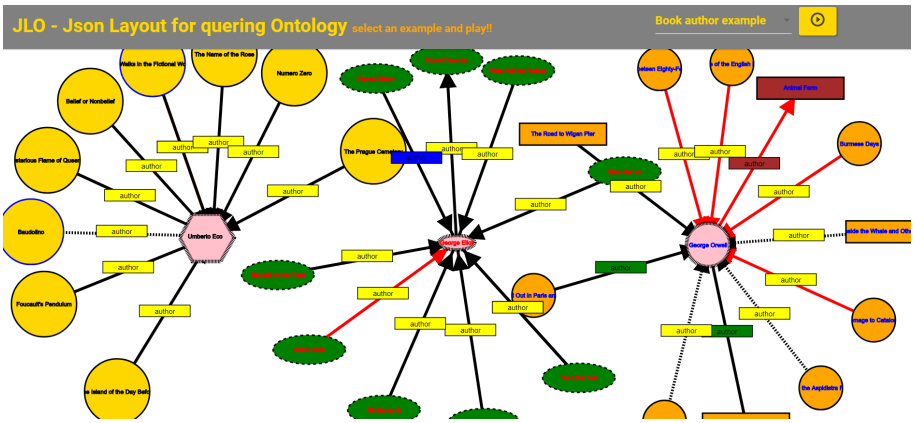

Figure 2: An example of visualisation of instances and properties based on JLO. In this case, three authors (i.e. Umberto Eco, George Elliot and George Orwell) are connected to their main plays. Many features of JLO are tested in this example, as the ability of using different shapes, colors, style for arcs and labels, etc.

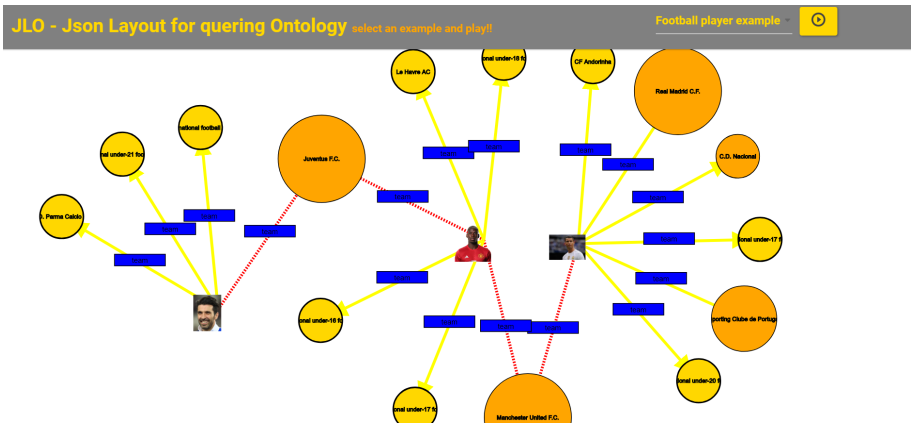

Figure 3: An example of visualisation of instances and properties based on JLO. Three soccer players are connected to the clubs they played for. Images (instead of shapes) are used to represent the footbal players. The size of the circles representing the teams are proportional to the number of international awards held by the teams.

- **Data visualisation**: the visualisation is automatically generated and updated after each change in the query. Many interactive behaviours are implemented. For instance, users can move the elements, navigate the visualization, zoom in/out, etc..

Two data visualisations have been implemented so far:

- **Graphs**: Data can be displayed as a force-directed graph layout. Due to the graph-like structure of the RDF data model, the node-link visualisations are considered the more relevant for the LOD domain. An example of graph visualisation is presented in Figure 4.
- **Clusters**: Data can be grouped together into visually distinct groups by selecting certain properties for a quick overview. To the best of our knowledge, clusters have not been used in combination with graph-based views to display LODs. Another innovative aspect concerns the possibility of interactively selecting the properties used to cluster items in GIG. An example of clustered data in presented in Figure 5.

GIG application is build with the AngularJS[23] framework, the powerful D3.js library for dynamic and interactive data visualisation and the SemanticUI[24] components framework for graphic interface.
As a future work, we plan to add new customisation features and static/dynamic graphic objects, and to develop more infoview technques (such as dendrogram, circle packing, radial node-link tree) that are eligible to be used to represent RDF data. Moreover, other data processing frameworks could be included if needed.

[23] AngularJS is a JavaScript-based open-source front-end web application framework, https://angularjs.org/.
[24] https://semantic-ui.com/

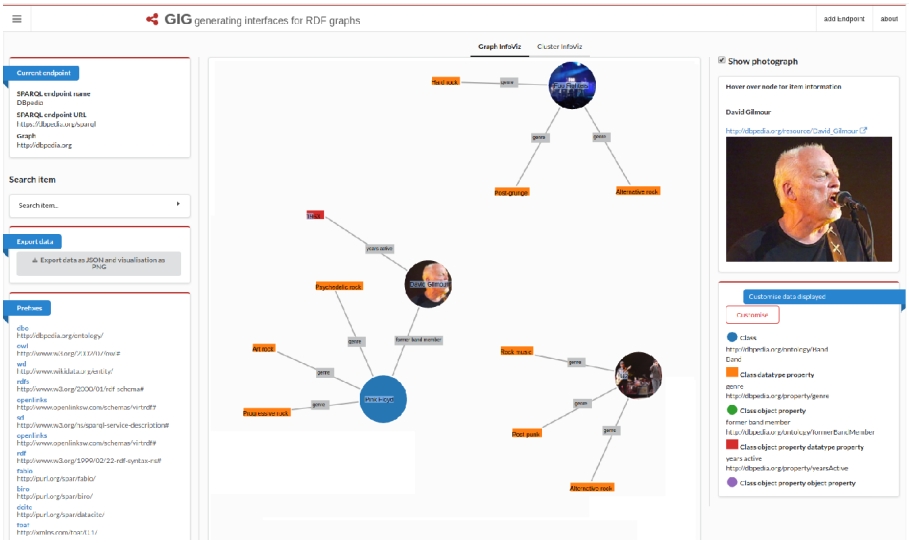

Figure 4: An example of graph view in GIG representing data about bands, band members and musical genres. Information about the source knowledge base (DBpedia in the example) and the used prefixes is shown on the left side of the interface, while the query editor is on the right side.

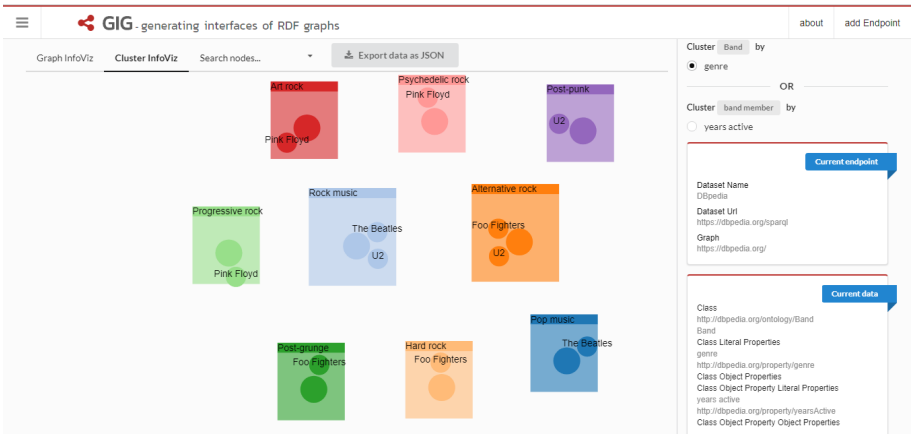

Figure 5: An example of cluster visualisation in GIG showing an alternative view of the same data represented in Fig. 4. In this case, the bands are clustered with respect of the musical genres.

6 DISCUSSION AND CONCLUSIONS

Preliminary results are promising and show that this is viable approach towards the exploration of semantic datasets for non-technical audience without Semantic Web domain knowledge.

Table 1 below shows if the JLO based web app and/or the GIG web app fulfill the requirements discussed in Section 3. As can be seen, both cases do not fully implement mechanisms for customising the final visualisation of the filtered data.

The JLO based web app and GIG web app are currently being developed but not completed yet. In the near future we plan to integrate these two platforms in a single workspace and to extend the set of primitives, widgets and options they provide to the users. The objective is to cover all the missing requirements and to investigate new ones.

The users will play a crucial role in such a process. Our next step is to validate the requirements discussed so far with a larger set of users, together with the JLO manifest and format.

From the implementation point of view, we also need to run some end-user tests on the current prototypes and their integration.

The ability to produce interactive visualisations is a further topic we plan to investigate. The challenge is again to specify the behaviours of the visualisation with a declarative approach, by providing a set of clear options to the final users, without filling too much the interface and without asking them to know the details of the data model and the dataset structure.

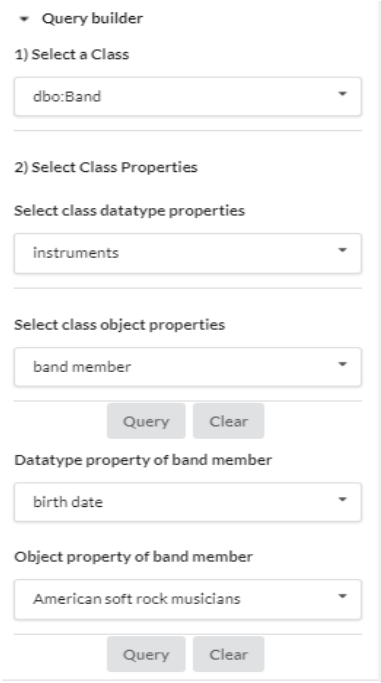

Figure 6: Query building with GIG. Multiple Classes and properties can be selected and filtered.

Table 1: JLO based web app and GIG web app comparison for a quick overview.

	JLO based web app	GIG web app
Easy Data Selection	hardcoded	✓
Data Simplification	✗	✗
Easy View Selection	✓	✓
Static Objects Customisation	✓	ongoing
Dynamic Objects Customisation	✗	✗
Support for alternative clustering	✗	✓

ACKNOWLEDGMENTS

The work is supported by a grant of the Consortium GARR - Italian Academic And Research Network[25] and by MIUR PRIN 2015 GAUSS Project.

REFERENCES

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. *The semantic web* (2007), 722–735.

[2] Andrea Bagnacani, Paolo Ciancarini, Angelo Di Iorio, Andrea Giovanni Nuzzolese, Silvio Peroni, and Fabio Vitali. 2014. The semantic Lancet project: a linked open dataset for scholarly publishing. In *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 101–105.

[3] Nikos Bikakis, John Liagouris, Maria Kromida, George Papastefanatos, and Timos K. Sellis. 2015. Towards Scalable Visual Exploration of Very Large RDF Graphs. In *ESWC*.

[4] Nikos Bikakis, John Liagouris, Maria Krommyda, George Papastefanatos, and Timos K. Sellis. 2016. graphVizdb: A scalable platform for interactive large graph visualization. *2016 IEEE 32nd International Conference on Data Engineering (ICDE)* (2016), 1342–1345.

[5] Christian Bizer, Tom Heath, and Tim Berners-Lee. 2009. Linked data-the story so far. *Semantic services, interoperability and web applications: emerging concepts* (2009), 205–227.

[6] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. 2008. Linked data on the web (LDOW2008). In *Proceedings of the 17th international conference on World Wide Web*. ACM, 1265–1266.

[7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics* 17, 12 (2011), 2301–2309.

[8] Diego Valerio Camarda, Silvia Mazzini, and Alessandro Antonuccio. 2012. LodLive, exploring the web of data. In *I-SEMANTICS*.

[9] Paolo Ciancarini, Francesco Poggi, and Daniel Russo. 2016. Big data quality: a roadmap for open data. In *Big Data Computing Service and Applications (BigDataService), 2016 IEEE Second International Conference on*. IEEE, 210–215.

[10] Angelo Di Iorio, Andrea Giovanni Nuzzolese, Silvio Peroni, Francesco Poggi, Fabio Vitali, and Paolo Ciancarini. 2017. Analysing and Discovering Semantic Relations in Scholarly Data. In *Italian Research Conference on Digital Libraries*. Springer, 3–19.

[11] Amineh Fadhil and Volker Haarslev. 2006. GLOO: A Graphical Query Language for OWL Ontologies. In *OWLED*.

[12] Alice Graziosi, Angelo Di Iorio, Francesco Poggi, and Silvio Peroni. 2017. Customised Visualisation of Linked Open Data. In *3rd International Workshop on Visualization and Interaction for Ontologies and Linked Data (VOILA! 2017) @ISWC 2017*. Springer. http://ceur-ws.org/Vol-1947/paper03.pdf

[13] Frederik Hogenboom, Viorel Milea, Flavius Frasincar, and Uzay Kaymak. 2009. RDF-GL: A SPARQL-Based Graphical Query Language for RDF.

[14] Ali Khalili, Antonis Loizou, and Frank van Harmelen. 2016. Adaptive Linked Data-Driven Web Components: Building Flexible and Reusable Semantic Web Interfaces - Building Flexible and Reusable Semantic Web Interfaces. In *ESWC*.

[15] Oliver Lehmberg, Alexander Brinkmann, and Christian Bizer. 2017. WInte. r-a web data integration framework. In *CEUR workshop proceedings*, Vol. 1963. RWTH, Paper–506.

[16] Steffen Lohmann, Vincent Link, Eduard Marbach, and Stefan Negru. 2014. WebVOWL: Web-based visualization of ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 154–158.

[17] Steffen Lohmann, Vincent Link, Eduard Marbach, and Stefan Negru. 2016. Extraction and Visualization of TBox Information from SPARQL Endpoints. In *Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2016) (LNAI)*, Vol. 10024. Springer, 713–728.

[18] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. 2016. Visualizing ontologies with VOWL. *Semantic Web* 7, 4 (2016), 399–419.

[19] Michael Luggen, Adrian Gschwend, Bernhard Anrig, and Philippe Cudré-Mauroux. 2015. Uduvudu: a Graph-Aware and Adaptive UI Engine for Linked Data. In *LDOW@WWW*.

[20] Francesco Poggi, Gabriele Cigna, and Andrea Giovanni Nuzzolese. 2016. Enhancing Open Data to Linked Open Data with ODMiner.. In *LD4IE@ ISWC*. 44–50.

[21] Eric Prud'Hommeaux, Andy Seaborne, and others. 2008. SPARQL query language for RDF. (2008). https://www.w3.org/TR/rdf-sparql-query/

[22] Marc Weise, Steffen Lohmann, and Florian Haag. 2016. LD-VOWL: Extracting and Visualizing Schema Information for Linked Data Endpoints. In *Proceedings of the 2nd International Workshop on Visualization and Interaction for Ontologies and Linked Data (VOILA 2016) (CEUR-WS)*, Vol. 1704. CEUR-WS.org, 120–127. http://ceur-ws.org/Vol-1704/paper11.pdf

[25] http://www.garr.it/en