Visual Query System for Analyzing Social Semantic Web

Jinghua Groppe
Institute of Information Systems (IFIS), University of Lübeck
Ratzeburger Allee 160
D-23538 Lübeck, Germany
+49 451 500 5706
groppej@ifis.uni-luebeck.de

Sven Groppe
Institute of Information Systems (IFIS), University of Lübeck
Ratzeburger Allee 160
D-23538 Lübeck, Germany
+49 451 500 5705
groppe@ifis.uni-luebeck.de

Andreas Schleifer
Institute of Information Systems (IFIS), University of Lübeck
Ratzeburger Allee 160
D-23538 Lübeck, Germany
+49 451 500 5701
schleifer@ifis.uni-luebeck.de

ABSTRACT

The social web is becoming increasingly popular and important, because it creates the collective intelligence, which can produce more value than the sum of individuals. The social web uses the Semantic Web technology RDF to describe the social data in a machine-readable way. RDF query languages play certainly an important role in the social data analysis for extracting the collective intelligence. However, constructing such queries is not trivial since the social data is often quite large and assembled from a large number of different sources. In order to solve these challenges, we develop a Visual Query System (VQS) for helping the analysts of social data and other semantic data to formulate such queries easily and exactly. In this VQS, we suggest a condensed data view, a browser-like query creation system for absolute beginners and a Visual Query Language (VQL) for beginners and experienced users. Using the browser-like query creation or the VQL, the analysts of social data and other semantic data can construct queries with no or little syntax knowledge; using the condensed view, they can determine easily what queries should to be used. Furthermore, our system also supports precise suggestions to extend and refine existing queries.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems – *Query processing.*

General Terms

Languages.

Keywords

Semantic Web, SPARQL, RDF, Visual Query Languages.

1. INTRODUCTION

The Social Web fosters collaboration and knowledge sharing, boots the collective intelligence of the society, organizations and individual people. The significance of the Social Web is already beyond the field of computer science. The Semantic Web promises a machine-understandable web, and provides the capabilities for finding, sorting and classifying web information, and for inferring new knowledge. The Semantic Web hence offers a promising and potential solution to mining and analyzing the social web.

Therefore, the emerging Social Semantic Web, the application of semantic web technologies to the social web, seems to be a certain evolution. In the Social Semantic Web, the social data is described using the Semantic Web data format RDF, and the social data becomes hence machine-understandable.

Analysis of social data plays a critical role in e.g. extracting the collective intelligence, obtaining insights, making decisions and solving problems. The Social Semantic Web gathers a huge amount of semantic data. For example, DBpedia (http://wiki.dbpedia.org/About) has extracted 479 million RDF triples from the social website Wikipedia (http://en.wikipedia.org/wiki/Main_Page). Analyzing such a large amount of data by either browsing the datasets or their visualizations is impractical. RDF query languages like SPARQL [7] are obviously an important tool for the data analysis.

Ontologies are used to describe the structure of the Semantic Web data. Due to the open world assumption of the Semantic Web, the data may contain structures, which are not described by the given ontologies. Ontologies are also often not given for considered data. This is especially true for the social data, which is contributed by a huge number of individual participants.

In order to formulate SPARQL queries for analyzing social data or other semantic data, the analysts have to hence look into the data as well as its ontology, besides having the knowledge of the SPARQL language. Looking into a large amount of social data is obviously inefficient and impractical. Therefore, in this work, we propose a condensed data view, which describes the structure of the initial large data, but uses a compact representation.

We also develop a browser-like query creation system and a visual editor of SPARQL queries, which allows analysts to formulate SPARQL queries with little syntax knowledge of SPARQL. Using different frames for the browser-like creation and the visual editor within the same window, the user can use both approaches in parallel for query creation, i.e., all effects during query editing are visible in the browser-like query creation as well as in the visual editor and the user can manipulate the query either in the browser-like query creation frame or in the visual editor. Furthermore, we support to extend and refine queries based on the concrete social data for constructing exacter queries for more precise and efficient data analysis during browser-like query creation as well as visual editing.

In order to demonstrate these approaches to facilitating the social data (and arbitrary other data) analysis, we develop a Visual Query System (VQS), which includes:

- a publicly available online demonstration at http://www.ifis.uni-luebeck.de/index.php?id=luposdate-demo,
- support of whole SPARQL 1.0,
- browser-like query creation for absolute beginners (see Figure 1),
- visual editor of SPARQL queries (see Figure 5), which hides SPARQL syntax from users and supports the creation of more complicated queries,
- visual browsing and editing of RDF data,
- condensed data view for avoiding browsing the initial large datasets (see Figure 2), and
- extending and refining queries based on query results.

Table 1. Example RDF Data `Records.rdf`

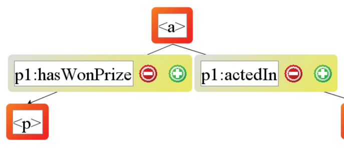
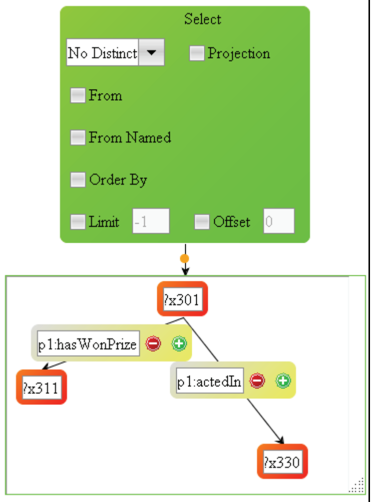
RDF data	Visual representation
<a> p1:hasWonPrize <p>. <a> p1:actedIn <f>.	

Table 2. Example SPARQL Query `DLCRecords.sparql`

Textual representation	Visual representation
PREFIX p1:<http://www.p/> *SELECT ** *WHERE {* *?x301 p1:hasWonPrize ?x311.* *?x301 p1:actedIn ?x330.* *}*	 Select No Distinct Projection From From Named Order By Limit -1 Offset 0 ?x301 p1:hasWonPrize ?x311 p1:actedIn ?x330

2. RDF VISUAL EDITOR

The core element of RDF data is the RDF triple *s p o*, where *s* is called the subject, *p* the predicate and *o* the object of the RDF triple. A set of RDF triples builds a directed graph, called RDF graph. In an RDF graph, the subject and object are nodes and the predicate is a directed edge from the subject to the object. Table 1 contains textual and visual representations (actually a screenshot of our RDF visual editor) of two RDF triples. As well as for browsing data, the RDF visual graphs can be used to update data easily. For example, the button + is used to add a triple, and – to delete a triple easily.

3. SPARQL VISUAL EDITOR

The natural way to visualize the triple patterns of SPARQL should be analogous to the way for RDF triples. Table 2 contains the textual and visual representations (a snapshot of our SPARQL visual editor) of a SPARQL query. Using check- and combo boxes in the visualization, users can immediately see the features of SPARQL without the need of learning the complex SPARQL syntax. Our SPARQL visual editor allows the modification of queries in a similar way as for RDF triples: using + to add a triple pattern or a new modifier; using – for removing. Furthermore, users' modification is parsed and checked immediately after each input, and error information is prompted to users. Consequently, our editor ensures users to construct syntactically correct queries.

4. BROWSER-LIKE QUERY CREATION

The browser-like query creation (see Figure 1) starts with a query for all data or with a query transformed from a visual or textual query. The result of the current query is presented in form of a result table. The result table contains buttons to further modify the query. For example, the "Rename" button allows renaming the column name of the result table, i.e., renaming the corresponding variable name in the query. The button "Sort" supports to sort the result according to a certain column, the button "Exclude" to hide the column and the button "Refine" to refine the query based on the values of this column. We will explain the refinement of queries in later sections. The "Filter" button supports to filter the result table according to values of a column and excludes the other rows in the result table. The browser-like query creation allows also to go back to previously created queries (button "<") and then to return to the later created queries (button ">").

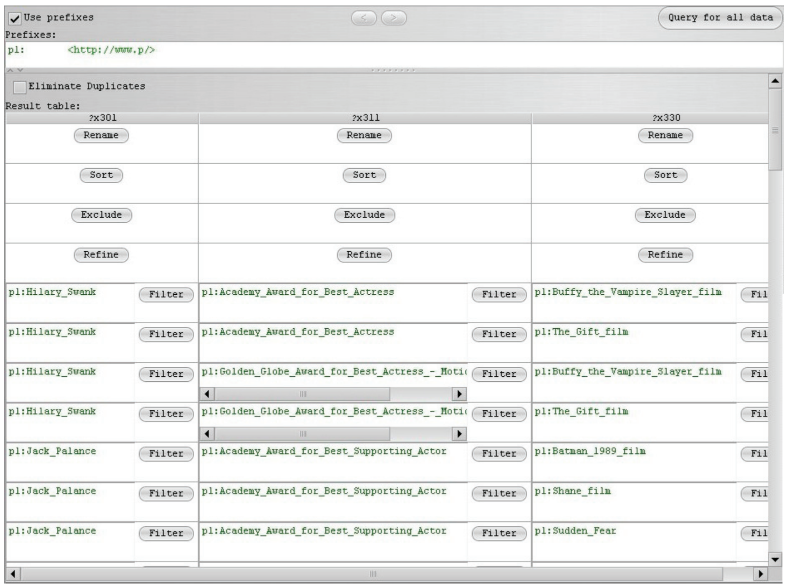
The advantage of the browser-like query creation is that it is very easy to use, features are always visible and no knowledge of the SPARQL syntax is necessary. Furthermore, users often like to work on the concrete data (rather than on an abstract query) seeing immediately the effect of the query modification regarding a correct query result as hint for a correct query. The disadvantage is that not all features of SPARQL can be provided by an easy graphical user interface like it is the case during visual query editing. For example, the union of two (sub-) queries cannot be created with the browser-like query creation approach. However, such a query can be taken over from the visual query editor and the query can be further modified with the browser-like query creation approach.

5. CONDENSED DATA VIEW

A condensed data view provides an efficient way to see the data structure and to get an overview of the original large datasets. Using the condensed view, one can easily determine the exact queries for the concrete purposes. A condensed view is generated by integrating the nodes of the RDF graph with certain common features into one node. We use a stepwise approach to condensing data. The main steps contain:

Step 1: integrating all *leaves* with the same subject into one node (see Figure 4 a)). A leaf is an object with only one incoming and without outgoing edges. We condense first all leaves in order to condense the information of a subject into one node, which is otherwise only displayed in a space-consuming way, and to preserve the information of long paths in the data.

Step 2: integrating all objects with the same subject and predicate into one node (see Figure 4 b)). The structure of the data related to objects with the same predicate are often the same or similar, such that this condensing strategy often yields good results.



Prefixes:

pl: <http://www.p/>

Result table:

?x301	?x311	?x330
pl:Hilary_Swank	pl:Academy_Award_for_Best_Actress	pl:Buffy_the_Vampire_Slayer_film
pl:Hilary_Swank	pl:Academy_Award_for_Best_Actress	pl:The_Gift_film
pl:Hilary_Swank	pl:Golden_Globe_Award_for_Best_Actress_-_Moti	pl:Buffy_the_Vampire_Slayer_film
pl:Hilary_Swank	pl:Golden_Globe_Award_for_Best_Actress_-_Moti	pl:The_Gift_film
pl:Jack_Palance	pl:Academy_Award_for_Best_Supporting_Actor	pl:Batman_1989_film
pl:Jack_Palance	pl:Academy_Award_for_Best_Supporting_Actor	pl:Shane_film
pl:Jack_Palance	pl:Academy_Award_for_Best_Supporting_Actor	pl:Sudden_Fear

Figure 1: The browser-like query creation

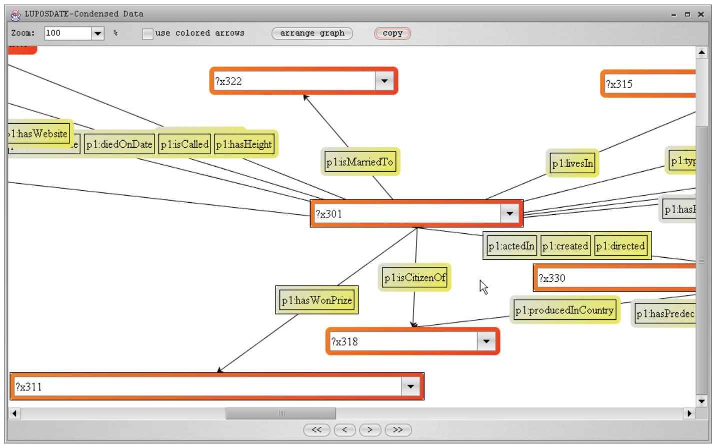
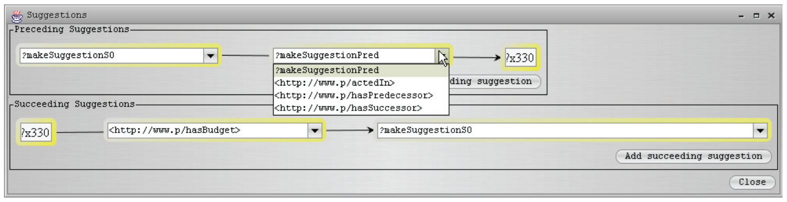


Figure 2: Condensed Data View of Yago data



Preceding Suggestions: ?makeSuggestionSO — ?makeSuggestionPred → ?x330

?makeSuggestionPred
<http://www.p/actedIn>
<http://www.p/hasPredecessor>
<http://www.p/hasSuccessor>

Succeeding Suggestions: ?x330 — <http://www.p/hasBudget> → ?makeSuggestionSO

Figure 3: Suggestion dialog

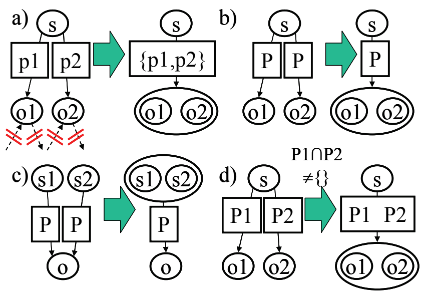
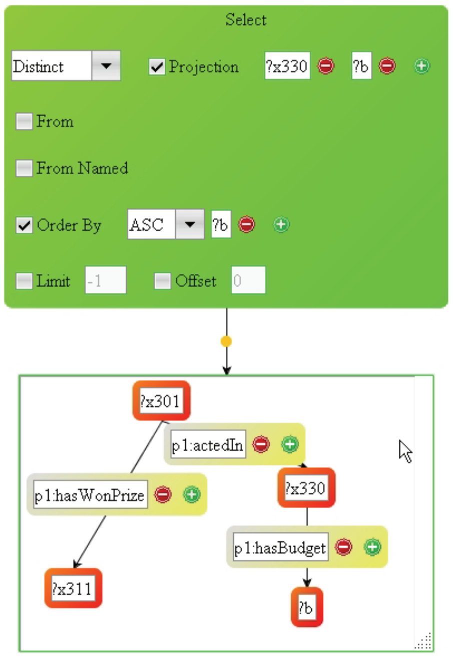


Figure 4: Condensing steps



Distinct ✓ Projection ?x330 ?b

From

From Named

✓ Order By ASC ?b

Limit -1 Offset 0

Figure 5: Final visual query in the SPARQL visual editor

Table 3. Suggestion-computing queries Q1 and Q2

Q1	Q2
PREFIX p1: <http://www.p/> *SELECT DISTINCT* *?x330 ?p ?so* *WHERE {* *?x301 p1:hasWonPrize ?x311.* *?x301 p1:actedIn ?x330.* ***?x330 ?p ?so.** }*	*PREFIX p1: <http://www.p/>* *SELECT DISTINCT* *?x330 ?p ?so* *WHERE {* *?x301 p1:hasWonPrize ?x311.* *?x301 p1:actedIn ?x330.* ***?so ?p ?x330.** }*

Step 3: integrating all subjects with the same predicates and objects into one node (see Figure 4 c)). Similar remarks as for Step 2 apply also to the quality of condensing for this step.

Step 4: integrating the objects with the same subject and at least one predicate in common into one node (see Figure 4 d)). This step condenses further allowing a sparse representation of the data.

Figure 4 describes the condensing steps and Figure 2 presents the condensed view of the Yago data [9]. All the four steps can be processed in reasonable time even for large datasets.

6. REFINING QUERIES

Our VQS efficiently supports refinement of queries by a *suggestion functionality* in our SPARQL visual editor. The refined queries can retrieve more exact results for efficient data analysis. In order to refine a query, the user selects the related node from the SPARQL graph in the SPARQL visual editor, e.g. the node for the variable *?x330* in Table 2, or presses the button “Refine” in the browser-like query creation frame (see Figure 1). Afterwards we create two suggestion-computing queries (Q1 and Q2 in Table 3), and evaluate them on the RDF data.

From the result of Q1, we can generate the *succeeding triple patterns* (i.e., those with *?x330* as subject) related to the node *?x330*. Let us assume *?x330=<TheFilmII>*, *?p=<http://www.p/hasSuccessor>* and ?*so=<TheFilmIII>* to be one result of Q1. Based on this result, we can recommend the following succeeding triple patterns:

- *?x330 <http://www.p/hasSuccessor> <TheFilmIII>*,
- *?x330 ?p <TheFilmIII>*, and
- *?x330 <http://www.p/hasSuccessor> ?so.*

Likewise, from the result of Q2, we can obtain the related *preceding triple patterns* (i.e., those with *?x330* as object). Figure 3 presents the dialog displaying the suggested triple patterns. Our system will automatically insert the suggested triple patterns selected by users into the query.

7. QUERY FORMULATION DEMO

Assume that there is a kind of users, who is neither familiar with the structure of the data nor with the SPARQL query language. We want to demonstrate by a simple example how such kind of users can easily create the queries with our VQS.

In order to formulate queries for e.g., Yago data, a user firstly browses the condensed view of the Yago data. In order to obtain the condensed view, the user selects the Yago data and then clicks "*Condense data*" from the main window. A condensed view window is popped, which contains the condensed data views from different condensing steps. Figure 2 presents a condensed data view of the Yago data [9].

The user is interested in the actors, which have won a prize, and in the films, in which the actors acted. Thus, the user selects the corresponding nodes in the condensed view and copies them by clicking *Copy* in the condensed view window. Afterwards, the user opens the query editor from the main window, chooses a SELECT query, and pastes the nodes into the query editor by clicking *Edit - Paste.* The generated visual query is described in Table 2 and its query result is displayed in the browser-like query creation frame (see Figure 1).

After investigating the query result, the user wants to have more information about the films, and thus the user should refine the query. In order to refine queries easily, the user uses our suggestion functionality in either the browser-like query creation approach (button "Refine" in the column for variable *?x330*) or in the SPARQL visual editor. In the SPARQL visual editor, the user needs to call *Edit – Make suggestions*. After that, the user clicks the node for the variable *?x330* (which selects films), and a suggestion window for query refinement is popped (see Figure 3).

From the suggestion dialog, the user chooses a succeeding triple pattern, *?x330 p1:hasBudget ?b.* The triple pattern is automatically integrated into the edited query in the visual editor as well as in the browser-like query creation frame after clicking the corresponding button *Add succeeding suggestion*.

Furthermore, the user also wants to see the querying result sorted according the budgets. By only looking at the visual editor, the user can easily see that the feature *Order By* is the one wanted. In the browser-like query creation frame it is even more obvious to click on the "Sort" button in the column for the variable *?b*. Figure 5 presents the final visual query.

8. RELATED WORK

We classify the existing *VQSs* for the Semantic Web according to the support of suggestions for extending and refining queries. The VQSs, which provide suggestions for extending and refining queries, include vSPARQL/NITELIGHT [8], GLOO [2], EROS [10], SEWASIE [1] and iSPARQL/OpenLink [6]. Other VQSs do not support suggestions, like RDF-GL/SPARQLinG [4], MashQL [5] and [3]. All the contributions for extending and refining a query are based on ontologies. In comparison, our VQS LUPOSDATE-VEdit extends and refines a query based on concrete data. Furthermore, we introduce the browser-like query creation.

9. SUMMARY AND CONCLUSIONS

In this paper, we present our solution to formulate SPARQL queries for analyzing the semantic data from e.g. the Social Semantic Web. Such data is often large-scale and is collected from a large number of different sources. By supporting a condensed data view and visual query editing as well as browser-like query creation, our VQS allows the users, who are neither familiar with the SPARQL language nor the data structure, to construct queries easily. Our system also provides a precise approach to query refinement based on the query result, and thus generating exacter queries.

10. REFERENCES

[1] T. Catarci, P. Dongilli, T. Di Mascio, E. Franconi, G. Santucci, S. Tessaris, An ontology based visual tool for query formulation support, *ECAI*, Valencia, 2004.

[2] A. Fadhil, V. Haarslev, GLOO: a graphical query language for OWL ontologies, OWL: Experience and Directions, pp. 215-260, 2006.

[3] A. Harth, S. Kruk, S. Decker, Graphical representation of RDF queries, *World Wide Web*, 2006.

[4] F. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak, RDF-GL: A SPARQL-Based Graphical Query Language for RDF, in Emergent Web Intelligence: Advanced Information Retrieval. Springer, 2010.

[5] M. Jarrar, M. D. Dikaiakos, A Data Mashup Language for the Data Web, *LDOW* at *WWW*, Madrid, Spain, 2009.

[6] OpenLink iSPARQL, http://demo.openlinksw.com/isparql.

[7] Prud'hommeaux E., Seaborne A. SPARQL Query Language for RDF, *W3C Recommendation*, 2008.

[8] A. Russell, P. R. Smart, NITELIGHT: A Graphical Editor for SPARQL Queries, *ISWC*, Karlsruhe, Germany, 2008.

[9] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.

[10] R. Vdovjak, P. Barna, G. Houben, EROS: Explorer for RDFS-based Ontologies, in Proceedings of Intelligent User Interfaces, Miami, Florida, USA, 2003.