OptiqueVQS: Ontology-based Visual Querying

Ahmet Soylu[1,2], Evgeny Kharlamov[3], Dmitriy Zheleznyakov[3],
Ernesto Jimenez-Ruiz[3], Martin Giese[1], and Ian Horrocks[3]

[1] Department of Informatics, University of Oslo, Norway
{ahmets, martingi}@ifi.uio.no
[2] Faculty of Informatics and Media Technology, Gjøvik University College, Norway
ahmet.soylu@hig.no
[3] Department of Computer Science, University of Oxford, United Kingdom
{name.surname}@cs.ox.ac.uk

Abstract. Visual methods for query formulation undertake the challenge of making querying independent of users' technical skills and the knowledge of the underlying textual query language and the structure of data. In this paper, we demonstrate an ontology-based visual query system, namely OptiqueVQS, which we have been developing for end users within a large industrial project.

Keywords: Visual Query Formulation, Ontology, Usability, SPARQL.

1 Introduction

Query interfaces play an essential role by enabling end users to express their ad hoc information needs. In this respect, visual query systems (VQSs) primarily undertake the challenge of making querying independent of users' technical skills and the knowledge of the underlying textual query language and the structure of data. To this end, we have been developing an ontology-based visual query system for end users, namely OptiqueVQS [1], within a large industrial project called Optique [2]. OptiqueVQS does not use a formal notation and syntax for query representation, but still conforms to the underlying formalism. It employs a formal approach projecting the underlying ontology into a graph for navigation, which constitutes the backbone of the query formulation process.

In this paper, we first demonstrate OptiqueVQS from an end-user perspective, and then present the ontology to graph projection approach.

2 OptiqueVQS

OptiqueVQS is meant for end users who have no or very limited technical skills and knowledge, such as on programming, databases, query languages, and have low/no tolerance, intention, nor time to use and learn formal textual query languages. It is not our concern to reflect the underlying formality (i.e., query language and ontology) per se; however, user behaviour is constrained so as to enforce the generation of valid queries. Secondly, we are not interested in providing full expressivity in order to reach a usability-expressivity balance.

2.1 User Interface

The OptiqueVQS interface is designed as a widget-based user-interface mashup (UI mashup). Apart from flexibility and extensibility, such a modular approach provides us with the ability to combine multiple representations, interaction, and query formulation paradigms, and distribute functionality appropriately.

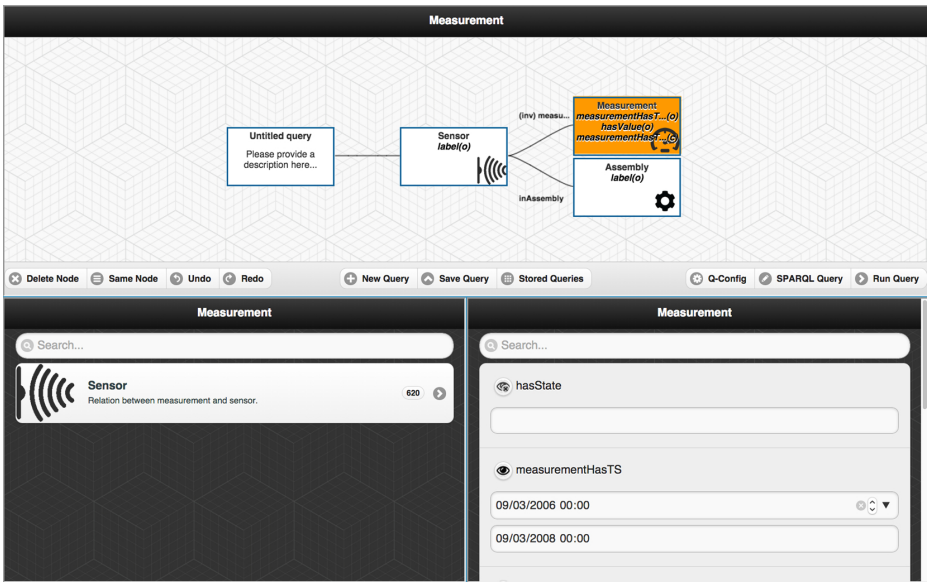

Fig. 1. An example query in visual mode is depicted.

Initially, three widgets appear in OptiqueVQS as depicted in Figure 1. The first widget (W1 – see the bottom-left part of Figure 1) is menu-based and allows users to navigate concepts by pursuing relationships between them. The second widget (W2 – see the bottom-right part of Figure 1) is form-based and presents the attributes of a selected concept for selection and projection operations. W1 and W2 provide view by focusing user to the active concept and provide means for gradual and on-demand exploration and construction. The third widget (W3 – see the top part of Figure 1) is diagram-based and provides an overview of the constructed query and functionality for manipulation.

Typically, a user first selects a kernel concept, i.e., the starting concept, from W1, which initially lists all domain concepts. The selected concept appears on the graph (i.e., W3) as a variable-node and becomes the pivot/active/focus node (i.e., the node coloured in orange or highlighted). W2 displays its attributes in the form of text fields, range sliders, etc. The user can select attributes to be included in the result list (i.e., using the "eye" button) and/or impose constraints on them through form elements in W2. Currently, the attributes selected for output appear on the corresponding variable-node with a letter "o", while constrained attributes appear with letter "c". The user can further refine the type of variable-node from

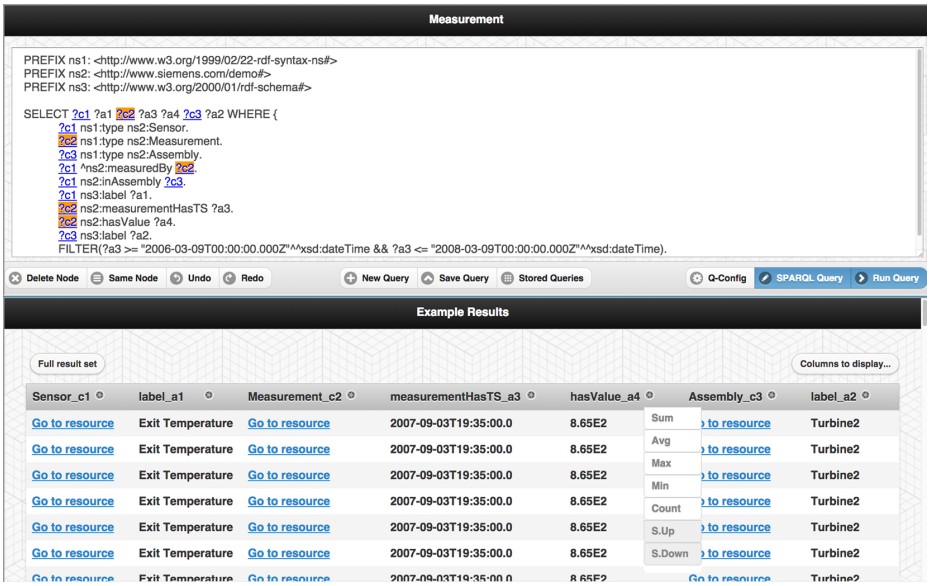

Fig. 2. An example query in textual mode and result view are depicted.

W2, by selecting appropriate subclasses, which are treated as a special attribute (named "Type") and presented as a multi-selection combo-box form element. Note that once there is a pivot node, W1 does not purely list concepts anymore but a set of (sub)paths. Each item/path in W1 represents a combination of a possible relationship with its range concept pertaining to the pivot (i.e., indeed a path of length one). The user can select any available item from the list; this results in a new path with a new variable-node of type specified by the selected item, a join between the pivot and the new variable-node over the specified relationship, and a move in focus to the new variable-node (i.e., pivoting). The user has to follow the same steps to involve new concepts in the query and can always jump to a specific part of the query by clicking on the corresponding variable-node in W3. The arcs that connect variable-nodes do not have any direction, but it is implicitly left to right. In W3, a tree-shaped query representation is employed to avoid a graph representation for simplicity.

The user can delete nodes, access the query catalogue, save/load queries, and undo/redo actions by using the buttons at the bottom part of W3. The user can also switch to editable textual SPARQL mode by clicking on "SPARQL Query" button at the bottom-right part of the W3 as depicted in Figure 2. The textual mode enables collaboration between end users and technology experienced users.

Finally, we recently extended OptiqueVQS with two new widgets, which provide an evidence on how a widget-based architecture allows us to hide complex functionality behind layers and combine different paradigms. The first widget is tabular result widget (W4 – see Figure 2). It provides an example result list from the current query and also means for aggregation and sequencing operations. The second widget is a map widget (W5 – see Figure 3). It allows end users to

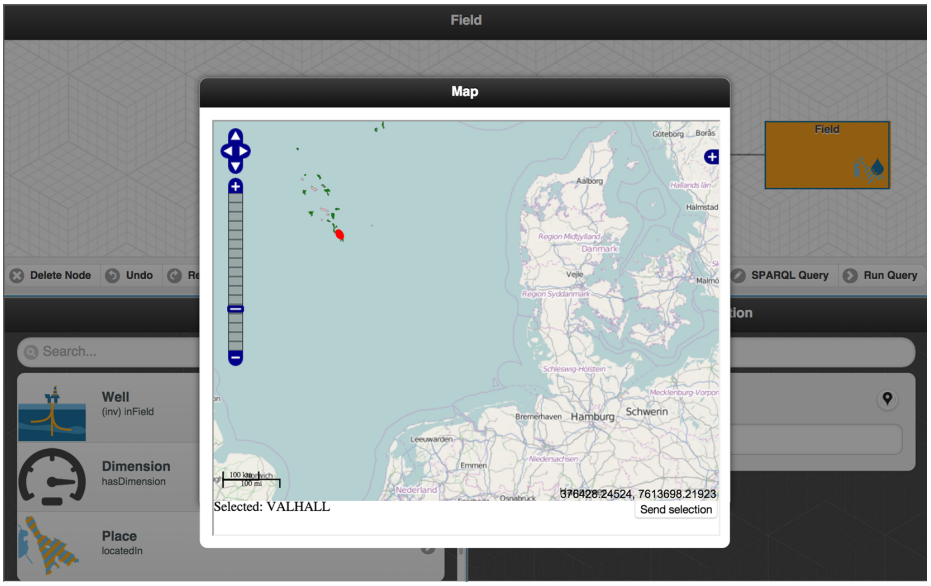

Fig. 3. An example query with the map widget is depicted.

constrain geospatial attributes by selecting an input value from the map. For this purpose, a button with a pin icon is placed next to every appropriate attribute.

2.2 Navigation Graph

Intuitively, OptiqueVQS allows users to construct tree-shaped conjunctive queries where each path is of the form: $Person(x), livesIn(x, y), City(y), ...$ Each such path essentially 'connects' classes like *Person* and *City* via properties like *livesIn*. At each query construction step OptiqueVQS suggests the user classes and properties that are semantically relevant to the already constructed partial query. We determine this relevance by exploiting the input OWL 2 ontology: we project the input ontology onto a graph structure that is called *navigation graph* [3] and use this graph at query construction time. More precisely, for each class in the partial query OptiqueVQS suggests only those properties and classes which are reachable in the navigation graph in one step. Note that OWL 2 ontologies are essentially sets of first-order logic axioms and thus there is no immediate relationship between them and a graph. This makes projection of OWL 2 ontologies onto a navigation graph a non-trivial task.

In the remaining part of this section we will formally introduce navigation graph, define when a query is meaningful with respect to it, and finally we define the grammar of queries that users can construct with the help of OptiqueVQS.

The nodes of a navigation graph are unary predicates and constants, and edges are labelled with possible relations between such elements, that is, binary predicates or a special symbol type. The key property of a navigation graph is that every X-labelled edge (v, w) is justified by a rule or fact entailed by $\mathcal{O} \cup D$

which "semantically relates" v to w via X. We distinguish three kinds of semantic relations: *(i) existential*, where X is a binary predicate and (each element of) v must be X-related to (an element of) w in the models of $\mathcal{O} \cup D$; *(ii) universal*, where (each instance of) v is X-related only to (instances of) w in the models of $\mathcal{O} \cup D$; and *(iii) typing*, where $X = \mathsf{type}$, and (the constant) v is entailed to be an instance of (the unary predicate) w. Formally:

Definition 1. *Let $\mathcal{O}$ be an OWL 2 ontology and D a knowledge graph. A* navigation graph *for $\mathcal{O}$ and D is a directed labelled multigraph G having as nodes unary predicates or constants from $\mathcal{O}$ and D and s.t. each edge is labelled with a binary predicate from $\mathcal{O}$ or* type*. Each edge e is justified by a fact or rule α_e s.t. $\mathcal{O} \cup \mathcal{C} \models \alpha_e$ and α_e is of the form given next, where c, d are constants, A, B unary predicates, and R a binary predicate:*

(i) if e is $c \xrightarrow{R} d$, then α_e is of the form $R(c, d)$ or $\forall y.[R(c, y) \rightarrow y \approx d]$;

(ii) if e is $c \xrightarrow{R} A$, then α_e is a rule of the form $\top(c) \rightarrow \exists y.[R(c, y) \wedge A(y)]$ or $\forall y.[R(c, y) \rightarrow A(y)]$;

(iii) if e is $A \xrightarrow{R} B$, then α_e is a rule of the form $\forall x.[A(x) \rightarrow \exists y.[R(x, y) \wedge B(y)]]$ or $\forall x, y.[A(x) \wedge R(x, y) \rightarrow B(y)]$;

(iv) if e is $A \xrightarrow{R} c$, then α_e is a rule of the form $\forall x.[A(x) \rightarrow R(x, c)]$ or $\top(c) \rightarrow \exists y.[R(y, c) \wedge A(y)]$ or $\forall x, y.[A(x) \wedge R(x, y) \rightarrow y \approx c]$;

(v) if e is $c \xrightarrow{\mathsf{type}} A$, then $\alpha_e = A(c)$.

The first (resp., second) option for each α_e in *(i)-(iii)* encodes the existential (resp., universal) R-relation between nodes in e; the first and second (resp., third) options for each α_e in *(iv)* encode the existential (resp., universal) R-relation between nodes in e; and *(v)* encodes typing. A graph may not contain all justifiable edges, but rather those that are deemed relevant to the given application.

To realise the idea of ontology and data guided navigation, we require that interfaces *conform to* the navigation graph. We assume that all the following definitions are parametrised with a fixed ontology $\mathcal{O}$ and a knowledge graph D.

Definition 2. *Let Q be a conjunctive query. The* graph of *Q is the smallest multi-labelled directed graph G_Q with a node for each term in Q and a directed edge (x, y) for each atom $R(x, y)$ occurring in Q, where R is different from $\approx$. We say that Q is* tree-shaped *if G_Q is a tree. Moreover, a variable node x is labelled with a unary predicate A if the atom $A(x)$ occurs in Q, and an edge (t_1, t_2) is labelled with a binary predicate R if the atom $R(t_1, t_2)$ occurs in O.*

Finally, we are ready to define the notion of conformation.

Definition 3. *Let Q be a conjunctive query and G a navigation graph. We say that Q* conforms to *G if for each edge (t_1, t_2) in the graph G_Q of Q the following holds:*

- *If t_1 and t_2 are variables, then for each label B of t_2 there is a label A of t_1 and a label R of (t_1, t_2) such that $A \xrightarrow{R} B$ is an edge in G.*

- *If t_1 is a variable and t_2 is a constant, then there is a label A of t_1 and a label R of (t_1, t_2) such that $A \xrightarrow{R} t_1$ is an edge in G.*
- *If t_1 is a constant and t_2 is a variable, then for each label B of t_2 there is a label R of (t_1, t_2) such that $t_1 \xrightarrow{R} t_2$ is an edge in G.*
- *If t_1 and t_2 are constants, then a label R of (t_1, t_2) such that $t_1 \xrightarrow{R} t_2$ is an edge in G.*

OptiqueVQS allows to construct conjunctive tree-shaped queries. The generation is done via reasoning over the navigation graph which contain edges of types (iii)-(v) (see Definition 1).

Now we describe the class of queries that can be generated using OptiqueVQS and show that they conform to the navigation graph underlying the system. First, observe that the OptiqueVQS queries follow the following grammar:

$$\begin{aligned} \texttt{query} &::= A(x)(\wedge\ \texttt{constr}(x))^*(\wedge\ \texttt{expr}(x))^* \\ \texttt{expr}(x) &::= \texttt{sug}(x,y)(\wedge\ \texttt{constr}(x))^*(\wedge\ \texttt{expr}(y))^* \\ \texttt{constr}(x) &::= \exists y R(x,y) \mid R(x,y) \mid R(x,c) \\ \texttt{sug}(x,y) &::= Q(x,y) \wedge A(y) \end{aligned}$$

where A is an atomic class, R is an atomic data property, Q is an object property, and c is a data value. The expression of the form $A(\wedge\ B)^*$ designates that B-expressions can appear in the formula 0, 1, and so on, times. An OptiqueVQS `query` is constructed using suggestions `sug` and constraints `constr`, that are combined in expressions `expr`. Such queries are conjunctive and tree-shaped. All the variables that occur in classes and object properties are output variables and some variables occurring in data properties can also be output variables.

3 Conclusion

OptiqueVQS enables non-experienced users to formulate comparatively complex queries at a conceptual level. The future work includes implementation of more features without compromising the usability, such as optionals.

Acknowledgements. This research is funded by "Optique" (EC FP7 318338), as well as the EPSRC projects Score!, DBOnto, and MaSI^3.

References

1. Soylu, A., et al.: Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users. Universal Access in the Information Society (in press)
2. Giese, M., et al.: Optique: Zooming in on Big Data. IEEE Computer Magazine **48**(3) (2015)
3. Arenas, M., et al.: Faceted Search over Ontology-Enhanced RDF Data. In: CIKM'14. (2014)