Optique™

Project Nº:	**FP7-318338**
Project Acronym:	**Optique**
Project Title:	**Scalable End-user Access to Big Data**
Instrument:	**Integrated Project**
Scheme:	**Information & Communication Technologies**

Deliverable D3.2
Techniques for Supporting Query Formulation

Due date of deliverable:	(T0+24)
Actual submission date:	November 2, 2014

Start date of the project: **1st November 2012**	Duration: **48 months**
Lead contractor for this deliverable:	**UiO**
Dissemination level:	**PU** – Public

Final version

Executive Summary:
Techniques for Supporting Query Formulation

This document summarises deliverable the Year 2 activities on visual query formulation (WP3) in the project Optique (FP7-318338), an Integrated Project supported by the 7th Framework Programme of the EC. Full information on this project, including the contents of this deliverable, is available online at `http://www.optique-project.eu/`.

In WP3 we address query formulation over ontologies. Query formulation interfaces that we aim at should enable end users to construct structured queries without prior knowledge of formal query languages. In this deliverable we report on the progress in WP3 during the second year of the project. In the second year, we continued to equip our query formulation interface with new functionalities that are aimed to improve user's experience. We discussed the technique that allows to suggest to users *relevant* query construction elements for further query construction based on the already built part of the query (Chapter 5). In Chapter 2 we discuss the interface *adaptivity* technique that aims at *ranking* query construction elements by using query history. Then we report on the support of formulation of geo-spatial queries (Chapter 3) and temporal and streaming queries (Chapter 4). All these functionalities require a special back-end support that we report in Chapter 6. Finally, we performed user evaluation of our system (Chapter 7).

List of Authors

Ahmet Soylu (UiO)
Dmitriy Zheleznyakov (UOXF)
Evgeny Kharlamov (UOXF)
Martin Giese (UiO)
Ernesto Jiménez-Ruiz (UOXF)
Martin G. Skjæveland (UiO)
Konstantina Bereta (UoA)
Ian Horrocks (UOXF)
Arild Waaler (UiO)

Contents

Chapter 1

Introduction

The purpose of this document is to describe Year 2 advances on *Techniques for supporting query formulation* corresponding to the Task T3.2 of WP3.

The goal of task T3.2 is to investigate and develop techniques to support query formulation and to design a suitable tool that, in particular, supports query by navigation (QbN) and direct query editing by relying on the domain model captured by the ontology.

Summary of Task Results. During the first two years of the project, we conducted an extensive study of related work and developed a QbN tool, OptiqueVQS, with an embedded SPARQL editor that allows to construct conjunctive tree-shaped queries with a limited form of disjunction between data values by navigation through the domain ontology. OptiqueVQS allows to construct one-shot, temporal, and streaming queries and addresses a number of use-case requirements. In particular, following the Statoil requirements a geospatial component of OptiqueVQS was developed; it allows to perform map-based value selections and a geospatial export of query answers. To improve query formulation experience for end-users, we developed techniques for ranking suggestions that OptiqueVQS provides to users during interactive query formulation process. In order to guarantee efficient query processing, the Optique platform relies on OWL 2 QL ontologies, at the same time, in order to provide rich and intuitive query formulation interfaces to end-users, a variety of information should be encoded in the ontologies much of which is not expressible in OWL 2 QL. To bridge this gap, we separate ontologies into logical and visualisation parts where the latter one is encoded using annotations and does not affect logical reasoning. We are developing the back end of OptiqueVQS in such a way that it provides a generic treatment of annotations and, in particular, allows to encode numerical and categorical data needed for ranges and drop-boxes, as well as selection of time intervals, and information needed for ranking. The back end of OptiqueVQS relies on a graph representation of ontologies, while ontological axioms are logical formulae without an obvious correspondence to graphs. To bridge the gap between logical axioms constituting ontologies and graphs needed for the back end, we proposed the semantic graph technique, that allow to "project" ontologies into graphs. Finally, we designed and conducted evaluation of OptiqueVQS with students and end-users from Statoil and Siemens. We published, submitted, or prepare for submission our techniques, implementations, and evaluations in international workshops and conferences.

List of Achievement in Year 2.

- The following functionalities of OptiqueVQS were extended:
  - `Query management`: users can save/load queries from the platform.
  - `Reversible actions`: users can undo/redo their actions over their working query.
  - `Subclass refinement`: users can refine a selected node to one or more of its subclasses, which are treated as multi-select field in the form-based widget.

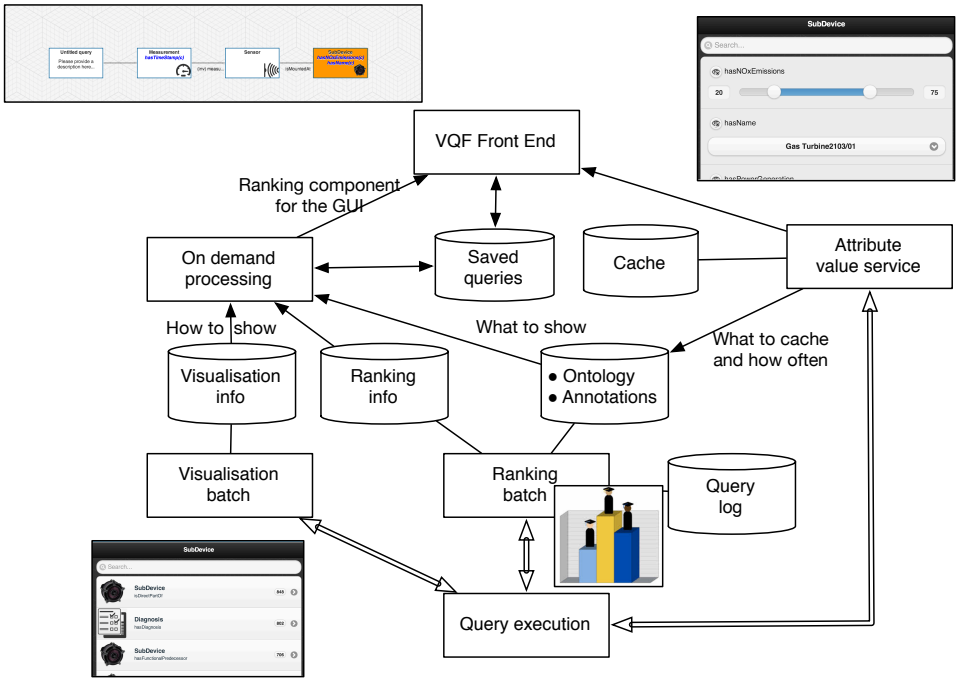


Figure 1.1: Year 2 Architecture of OptiqueVQS

  - `Widget binding to form elements`: system allows binding input widgets to form fields, for instance a map widget is attached to representative attributes of concepts that have geographical presence, so that a user can select an instance from the map widget rather than typing its name.
  - `Parameterised initialisation`: system now can be initiated with a set of predefined parameters such as, ontology to use, a stored query id to load, repository etc.
  - `Experiment mode`: system now can be operated in an experiment mode, in which the use of tool is controlled and session data is collected for experimental analysis. For instance, users' attempts are stored along with time to complete each attempt and a simple interface is made available for pre-loading tasks for an experiment.

- A ranking model for query suggestions was proposed and partially implemented.
- An ontology projection technique was proposed and properties of projections were studied.
- A preliminary query formulation support for temporal and streaming queries was added.
- An initial implementation of a geospatial component of OptiqueVQS was conducted.
- The architecture of OptiqueVQS was redesigned and updated in order to support new developments, see the new architecture in Figure 1.1.
- Evaluations with students and end-users in Statoil and Siemens was conducted.

Refined Query Interface Backend Architecture To provide an optimal user experience in the Visual Query Interface, many pieces of information are required. Exactly what is shown to the user, and in which order, may depend on:

- The ontology (available classes, etc.)
- The data (ranges of attribute values, etc.)

- Previous queries, a.k.a. the query log (e.g. for ranking, see Chapter 2
- Manual intervention – in case the automatic behaviour of the interface is not optimal.

For instance, consider a "temperature" attribute on some class. The ontology might constrain this to be a real number, which might make a range slider suitable. But what are the minimum and maximum numbers? If the ontology has no information about this, the data source can be queried for the extreme values. On the other hand, if the attribute in question is a water temperature, it might be most natural to manually instruct the system to use a range of 0 to 100.

Note that finding minimum and maximum numbers from the data source might be a very good approach for many cases – but an extremely expensive one for large amounts of data. So certainly, the range of attribute values should not have to be determined from the data every time the attribute in question appears in the interface. Some form of caching is needed; but then it has to be possible to configure the system's caching regime.

We have decided to develop an architecture where configuration information steering the behaviour of the visual query interface are added to the ontology in the form of OWL annotations, see Chapter 6.1. Ultimately, for any attribute, it will be possible to specify

- which of a variety of widgets to use (e.g., sliders, text fields, popups, etc.)
- parameters like possible values, range min/max values
- instructions to read such values from the data, including a caching schedule

Fig. 1.1 shows the back-end architecture that provides all required information to the front-end. The components, from the bottom up, are as follows:

- "Query execution" is the Optique query execution component that connects to the data sources.
- "Visualisation batch" periodically runs queries against the data in order to cache information about how attributes should be displayed (e.g. text field or popup depending on number of different values)
- "Query log" stores previously executed queries
- "Ranking batch" periodically analyses the ontology and the query log and runs queries against the data in order to cache information about the ranking of relations and attributes in the VQS. See Sect.2 for initial work on this.
- "Visualisation info" is a cache for information on visualisation configuration determined for the data
- "Ranking info" is a cache for information on ranking configuration determined for the data, ontology, and query log
- The ontology with its VQS annotations serves as single configuration point for the VQS backend
- "On demand processing" is the component the front-end contacts every time the focus changes, constraints are added, etc., to determine which relations and attributes to show, and in which order.
- "Saved queries" is the repository of queries saved from the front end, including the one currently authored.
- "Attribute value service" is contacted by the front end to determine attribute values in popup menus, sliders, for autocompletion, etc. Depending on the annotations in the ontology, these are extracted from the ontology, or from the data, with caching.
- "Cache" is the cache of possible attribute values used by the Attribute value service.

Ongoing Work on Query Driven Ontology Construction. We are currently working on query driven ontology construction module of the Optique platform. In particular, we work on techniques for learning ontologies from query catalogs and plan to test them on Siemens and Statoil query catalogs, which we have in the form of texts in English. We envision that application of such techniques would extract simple ontological structures form queries and then these structures can be incorporated in the ontologies already present in an Optique installation via alignment.

Compliance with T3.2 Task Description and Relationship with Other Tasks and Work-packages. Results of both Optique years in T3.2 match the goal of this task. The techniques and software components developed within T3.2 are tightly connected to other tasks and work-packages. In particular, there is a tight integration of OptiqueVQS with the query rewriting component of WP6. There is also a tight relationship with WP4: the ontology alignment module of T4.1 allows to incorporate visualisation ontology into the Optique platform; there is also a shared infrastructure with WP4: both work-packages rely on the a shared ontology reasoner Hermit.

Structure of the Deliverable. In the following chapters of the deliverable we will focus on several directions that we progressed on within Year 2. Chapter 2 discusses our development of ranking functions. In Chapter 3 we present how we address the geospatial dimension in query formulation and in answer exporting processes. Chapter 4 presents how OptiqueVQS supports temporal and streaming query formulation. Chapter 5 explains how to project ontologies into graphs. In Chapter 6 we explain how we support ontology annotations in OptiqueVQS. Finally, Chapter 7 reports on user evaluations that took place in Year 2.

Chapter 2

Extending OptiqueVQS with Ranking

One of the main problems that OptiqueVQS and typically any other VQS face is *scalability* against large ontologies (cf. [18]). A VQS has to provide its users with the elements of ontology (e.g., *concepts* and *properties*) continuously, so that users can select relevant ontology elements and iteratively construct their queries (see more details about relevant elements in Chapter 5). However, even with considerably small ontologies, the number of concepts and properties to choose from increases drastically due to the propagative effect of ontological reasoning (cf. [14]). In turn, the high number of ontology elements overloads the user interface and hinders *usability*.

We approach the aforementioned problem with *adaptivity* (cf. [5]) by exploiting a *query history* to *rank* and *suggest* ontology elements with respect to an incomplete query that a user has constructed so far (i.e., *context-aware*). The approach is specifically devised for *SPARQL* [15], takes semantics into account with reasoning support, and uses SPARQL, as a programming language, for the implementation.

The results presented in this chapter were published in [27] (see also Appendix B).

2.1 Adaptive Query Formulation

Currently, the widgets W2 and W3 present all the available concept-object property pairs and data properties to users respectively. This widgets can be found in Figure 2.1 in the left-bottom and right-bottom parts, respectively. However, the lists grow quickly due to *ontology size*, *number of relationships between concepts*, *subproperties*, *inverse properties*, *inheritance of restrictions* etc. As the lists grow, the time required for a user to find elements of interest increases; therefore ranking ontology elements with respect to previously executed queries and suggesting highly ranked elements first as possible query continuations have potential to increase the efficiency of the users. The nature of OptiqueVQS requires suggestions to be done for the pivot (i.e., cursor point) rather than for any part of a query.

In what follows, we first present a running example and then describe our ranking method for context-aware suggestions. The running example is built on one of the use cases, namely the Statoil use case.

2.1.1 Running Example

A partial simplified ontology for Statoil exploration department is depicted in Figure 2.2. In Figure 2.3, an example query log with three queries is assumed for the sake of brevity. The first query, *Q1*, is the one that is depicted in Figure 2.1 and asks for the names of wellbores with a drilling facility and a drilling company. The second query, *Q2*, asks for the content of all shallow wellbores that belong to wells and have drilling companies of type operator. The final query, *Q3*, asks for the content of all exploration wellbores that have fixed drilling facilities and drilling companies.

In Figure 2.3, *PQ* refers to an example partial query. The query in its incomplete form asks for all exploration wellbores with a drilling company; the cursor point is the variable of type exploration wellbore. At this point of query formulation session, the widgets W2 and W3 need to suggest the most relevant continuations by comparing the partial query with the queries in the query log.

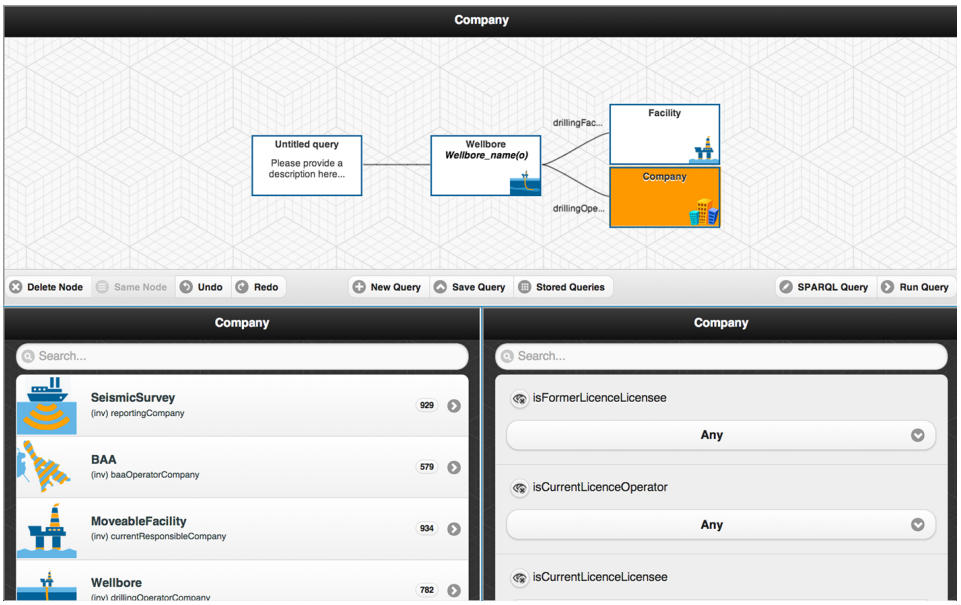


Figure 2.1: OptiqueVQS – an example query is depicted

2.1.2 Basic Notions

We assume the following pairwise disjoint alphabets: a set of *URIs* U, a set of *literals* L, and a set of *blank nodes* B. We say that a labelled directed graph is in *RDF* if *(i)* a node of the graph belongs to $U \cup L \cup B$ and *(ii)* an edge $\langle s, p, o \rangle$ of the graph belongs to $(U \cup B) \times U \times (U \cup L \cup B)$; here s is usually referred to as *subject*, p as predicate, and o as object. It is well known that an OWL 2 ontology can be represented as an RDF graph [7].

The standard language to query RDF graphs is SPARQL [15]. A SPARQL query can be seen as a *graph pattern*, which is a set of *graph patterns* $\langle s, p, o \rangle$ belonging to $(U \cup B \cup Var) \times (U \times Var) \times (U \cup L \cup B \cup Var)$, where Var is an infinite set of variables. SPARQL is based on matching graph patterns against the input RDF graph. With OptiqueVQS, however, user cannot formulate arbitrary queries, but queries that are constituted by graph patterns from $Var \times U \times (U \cup Var \cup L)$.

Example 2.1.1. *The query depicted in Figure 2.1 can be formally written as*

```
SELECT ?c1 ?a1 ?c2 ?c3 WHERE{
  ?c1 ns1:type ns2:Wellbore.
  ?c2 ns1:type ns2:Facility.
  ?c3 ns1:type ns2:Company.
  ?c1 ns2:drillingFacility ?c2.
  ?c1 ns2:drillingOperation ?c3.
  ?c1 ns2:name ?a1.
},
```

where `ns1:` *and* `ns2:` *are corresponding name spaces (i.e., predefined prefixes that, together with the rest of the name of the node, constitutes a unique URI).*

2.1.3 Ranking Method

A query log QL is basically a set of SPARQL queries: $QL = \{Q_1, Q_2, ..., Q_n\}$. We define a function p that takes a query Q as input and returns its graph pattern P. We define S as a set of suggestions

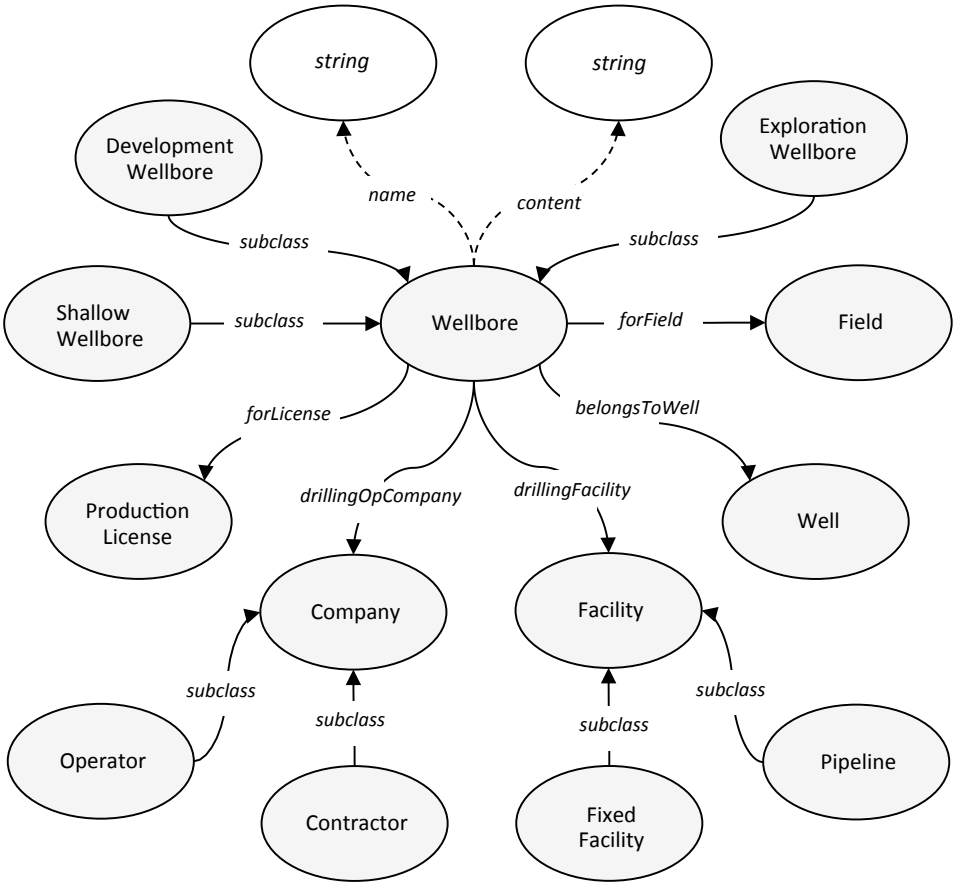


Figure 2.2: A partial simplified ontology for the Statoil use case

$\{T_1, T_2, ..., T_m\}$. Each suggestion in S is a triple set T_i, which either contains two triples for W2 in the form of $\{\langle x, o, y\rangle \in Var \times U \times Var, \langle y, \texttt{rdf:type}, w\rangle \in Var \times U \times U\}$ or one triple for W3 in the form of $\{\langle x, d, y\rangle \in Var \times U \times (Var \cup L)\}$, where x corresponds to the cursor variable in a partial user query Q_a. Note that subclass suggestion is not included in the ranking since it is always suggested by default.

The ranking score, at this point, basically corresponds to the *conditional probability* for each suggestion T_i in S, given a partial query Q_a and a query log QL, that is $Pr(T_i \mid p(Q_a))$. Conditional probability and probability functions are defined in the followings.

Within a query log QL, the probability of a graph pattern P is defined as the fraction of graph patterns in QL that are *supergraphs* [26] of P, as shown in the following equality:

$$Pr(P) = \frac{|\{Q_i \in QL | P \subseteq p(Q_i)\}|}{|QL|}.$$

The conditional probability of a triple set T given a graph pattern P is defined as the quotient of the probability of the union of T and P, and the probability of P as shown in the following equality:

$$Pr(T \mid P) = \frac{Pr(T \cap P)}{Pr(P)}.$$

Now two important questions come into play. First, how do we find supergraphs in the query log, given a partial user query? Second, how do we extract possible extensions, i.e., suggestions, for the partial query from found supergraphs? As far as the first problem is concerned, it boils down to a *graph matching* problem.

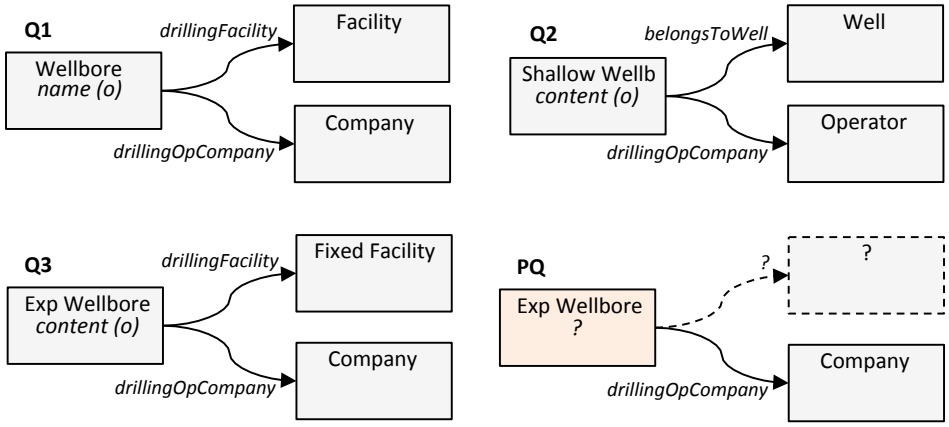


Figure 2.3: A query log with three queries and an example partial user query

We consider a graph pattern P_1 to be subgraph of another graph pattern P_2, if all the triple patterns of P_1 are covered by P_2, independent of variable names, ordering of triple patterns, and the values of constraints. Dividino and Groner [13] review different approaches for checking graph similarity, where our interest falls into *content-based* approaches. We propose a method that relies on SPARQL itself and provides us with an exhaustive solution, as it allows us to exploit semantic knowledge while matching queries.

The method starts with the instantiation of graph patterns of queries in the query log by replacing variable names and constraints on data type properties with blank nodes; blank node names are marked with a query identifier for preventing any overlap and identification purposes. Then, the resulted RDF graphs are stored in a common dedicated triple store; the instantiation of the query log depicted in Figure 2.3 is given in Figure 2.4. By applying the partial query over this triple store, one can retrieve all the queries that are the supergraphs of the partial query.

As far as the second question is concerned, i.e., finding possible extensions, the partial query is extended with a triple pattern from the cursor point to retrieve all extensions occurred in the matching supergraphs. The output of partial query is modified to retrieve the identifiers of matching queries, properties, and the types of variables for the returned extension. An example is given in Figure 2.5 for the partial query depicted in Figure 2.3 and the triple store depicted in Figure 2.4. The rest of the method involves calculation of conditional probabilities for the suggestions, as exemplified in Figure 2.5.

If one inspects the results in Figure 2.5 closely, they will realise that reasoning is involved. This is because in the query log, only Q3 is an exact match for the partial query. However, thanks to reasoning support, Q1 is also matched, since exploration wellbore is a subclass of wellbore. Likewise, this guarantees a match for any query that has a *semantic similarity* [16] to the partial query, involving subclasses, subproperties, inverses etc. Yet, it is possible to query the triple store without any reasoning, if one wants to eliminate such matches, hence avoiding any semantic distance.

The final stage involves ordering and dividing S into two sets, S_1 for W2 and S_2 for W3, with respect to ranking score and type of each suggestion (i.e., concept-relationship pair vs. data type property). Then, suggestions in each set are paginated into $\frac{|S_i|}{j}$ pages, where i is the set identifier and j is the *window size* for a page (i.e., the required number of suggestions for a page).

2.2 Related Work

There are a number of visual query formulation tools available in the literature (e.g., [8, 9, 17, 2]); however, to the best of our knowledge none of them supports adaptive visual query formulation. Existing approaches for adaptive query formulation are largely developed for *context-sensitive* textual query formulation.

Khoussainova et al. [20] provide a system, named *SnipSuggest*, for context-aware composition of textual

Query log: SPARQL form

```
Q1 SELECT DISTINCT ?c1 ?a1 ?c2 ?c3
WHERE {
  ?c1 ns1:type ns2:Wellbore.
  ?c2 ns1:type ns2:Facility.
  ?c3 ns1:type ns2:Company.
  ?c1 ns2:drillingFacility ?c2.
  ?c1 ns2:drillingOpCompany ?c3.
  ?c1 ns2:name ?a1.
}

Q2 SELECT DISTINCT ?c1 ?a1 ?c2 ?c3
WHERE {
  ?c1 ns1:type ns2:ShallowWellbore.
  ?c2 ns1:type ns2:Operator.
  ?c3 ns1:type ns2:Well.
  ?c1 ns2:drillingOpCompany ?c2.
  ?c1 ns2:belongsToWell ?c3.
  ?c1 ns2:wellboreContent ?a2.
}

Q3 SELECT DISTINCT ?c1 ?a1 ?c2 ?c3
WHERE {
  ?c1 ns1:type ns2:ExpWellbore.
  ?c2 ns1:type ns2:Company.
  ?c3 ns1:type ns2:FixedFacility.
  ?c1 ns2:drillingOpCompany ?c2.
  ?c1 ns2:drillingFacility ?c3.
  ?c1 ns2:wellboreContent ?a1.
}
```

Query log: triple form

```
_:q1c1 ns1:type ns2:Wellbore.
_:q1c2 ns1:type ns2:Facility.
_:q1c3 ns1:type ns2:Company.
_:q1c1 ns2:drillingFacility _:q1c2.
_:q1c1 ns2:drillingOpCompany _:q1c3.
_:q1c1 ns2:name _:q1a1.

_:q2c1 ns1:type ns2:ShallowWellbore.
_:q2c2 ns1:type ns2:Operator.
_:q2c3 ns1:type ns2:Well.
_:q2c1 ns2:drillingOpCompany _:q2c2.
_:q2c1 ns2:belongsToWell _:q2c3 .
_:q2c1 ns2:wellboreContent _:q2a1.

_:q3c1 ns1:type ns2:ExpWellbore.
_:q3c2 ns1:type ns2:Company.
_:q3c3 ns1:type ns2:FixedFacility.
_:q3c1 ns2:drillingOpCompany _:q3c2.
_:q3c1 ns2:drillingFacility _:q3c3.
_:q3c1 ns2:wellboreContent _:q3a1.
```

Figure 2.4: The instantiation of query graph patterns

SQL queries with respect to a given query log. The authors translate each SQL query in the query log into a set of features (e.g., a table name appearing in the *FROM clause*). Similarly, the partial query of the user is also translated into a set of features. Possible features for extension are identified by matching the feature sets of the partial query and the feature sets of queries in the query log and are ranked by calculating conditional probabilities. The approach generates suggestions for extending any part of the partial query rather than a single cursor point. Authors also propose a set of supportive algorithms and techniques for, such as feature set matching (i.e., what if the partial query does not appear in the query log), the selection of suggestions (i.e., accuracy vs. diversity), and query log elimination (i.e., to reduce the size). The elaborate approach provided by SnipSuggest system is relevant to our contribution in many aspects. However, a fundamental difference is in feature comparison; while the features of SnipSuggest system are a set of syntactic elements and the feature comparison is string based, for OptiqueVQS feature sets (i.e., correspond to the triple sets of graph patterns) have a semantic nature and compared semantically. The semantic aspects not only concern how the matching is done, but also the calculation of rankings, which we discuss in the following section.

As far as approaches for SPARQL are concerned, Campinas et al. [6] propose an approach for assisting textual SPARQL query formulation, however in a different context. Their approach assumes that an ontology describing the data set is unknown. Therefore, the authors propose a model that summarises the underlying data graph and extracts ontology elements to suggest. The approach extends a given partial user query from the cursor point, similar to our approach, and then evaluates it over the data graph summary to retrieve possible extensions. However, the approach does not realise any ranking of suggestions based on the previously executed queries and does not take semantic similarities between queries into account, possibly due to lack of rich domain knowledge (e.g., lack of subclass, inverse property axioms).

Kramer et al. [21] present a tool, named *SPACE*, to support autocompletion of textual SPARQL queries.

Partial user query

```
SELECT DISTINCT ?c1 ?c2
WHERE {
  ?c1 ns1:type ns2:ExpWellbore.
  ?c2 ns1:type ns2:Company.
  ?c1 ns2:drillingOpCompany ?c2.
}
```

Modified partial user query

```
SELECT DISTINCT ?c3 ?prop ?type
WHERE {
  ?c1 ns1:type ns2:ExpWellbore.
  ?c2 ns1:type ns2:Company.
  ?c1 ns2:drillingOpCompany ?c2.
  ?c1 ?prop ?c3.
  OPTIONAL { ?c3 rdf:type ?type }
}
```

Matches

?c3		?prop	?type	Pr(T|P)	Widget
_:q1c2	T_1	ns2:drillingFacility	ns2:Facility	0.16	W2
_:q1c3	T_2	ns2:drillingOpCompany	ns2:Company	0.33	W2
_:q1a1	T_3	ns2:name		0.16	W3
_:q3c2	T_2	ns2:drillingOpCompany	ns2:Company		
_:q3c3	T_4	ns2:drillingFacility	ns2:FixedFacility	0.16	W2
_:q3a1	T_5	ns2:wellboreContent		0.16	W3

Figure 2.5: Modified partial user query and possible query extensions

For this purpose, it takes a SPARQL query log as an input and then builds an *index structure* for the computation of query suggestions. The index structure has a root node at level 0, representing a set of queries, while each vertex at level 1 represents a SPARQL query. The vertices from level $n-2$ to index level 1 represent graph patterns recursively. Finally the vertices at the highest level $(n-1)$ represent IRIs, blank nodes, literals, variables, and binary operators such as *AND*, *UNION*, and *FILTER*. The suggestion process is done by subgraph matching for the partial user query in the index graph in a bottom up manner. However, the authors describe neither the subgraph matching process nor the details of ranking calculation. Finally, the index structure could grow quickly as it is built on recursive decomposition of graph patterns.

Chapter 3

Support of Geo-spatial Query Formulation

In order to make a system for data access truly adapted to the needs of any given domain, special attention needs to be given to the types of data relevant for that domain: they might require specialized storage (more than just relational databases), specialized reasoning and query processing, and of course also specialized mechanisms for query formulation, that correspond to the most natural ways of interacting with such data.

In the petroleum exploration domain, much of the data to be accessed is tied to specific geospatial extents. For instance, information is attached to wellbores, which are located within an oil field, a license area, some seismic survey, etc. Specifying entities in terms of their geospatial location is common practice in the tools and work flows at the Statoil exploration department (more details can be found in the deliverable D9.2). It was therefore decided to make a particular effort to include geospatial querying in the Optique platform, although this was not foreseen in (the original version of) the DoW.

It seems obvious that the most natural way of specifying geospatial constraints in a data access task is via a map – by selecting objects, or drawing outlines of regions. But a closer scrutiny shows that this is not necessarily the case. Depending on the information need, users might want to construct different kinds of queries:

1. selecting features (e.g. wellbores) that are related to (lie within, overlap, etc.) a **specific feature** or set of features (e.g. some fields or a license area) that are explicitly represented in the data set.

   Example: All wellbores within the Ekofisk field area.

2. selecting features that are related to (lie within, overlap, etc.) a **free form region** that does not have to coincide with the geometry of some entity in the data set.

   Example: All wellbores between 61°and 62°N and between 2°and 3°E.

3. selecting features purely based on their geospatial relationships without any reference to some fixed feature or region.

   Example: All wellbores within 1km of some pipeline.

To deal with variant 1, it is natural to allow selecting the relevant feature (e.g. the Ekofisk field in the example) on the map, and to combine this selection with the OptiqueVQS, in order to allow users to formulate more complex information needs.

Similarly, variant 2 can be approached by allowing the user to define the region in question by using the selection tools of some mapping software, and again interfacing with OptiqueVQS to allow for added relational complexity.

It is important to point out however, that variant 3 is not as obviously formulated using a map: the relationship of spatial containment or overlap is not something that is easily specified by indicating a particular point or region on a concrete map.

Moreover, we questioned users of map-based tools at both Statoil and technology provider Schlumberger SIS to find out to which extent these different variants of geospatial querying are used in practice. It turned

out that for organizational reasons, almost any information need of these users will be restricted to specific regions. Variant 1 is the most common pattern. Variant 2 is also in demand, since it gives users some flexibility, for instance in focusing on a slightly larger area than that defined by a specific block, field, etc. Queries of type 3, without any further spatial restriction seem to be needed rarely, if ever.

We therefore decided to create a specialized geospatial (i.e. map) widget for OptiqueVQS to handle variants 1 and 2.

To specify relationships like containment and overlap when required, it is entirely possible to treat these generically in OptiqueVQS, like any other binary relation between features.

An additional aspect is the processing ability that the different variants require from the backend. For variant 1, it is sufficient that the relevant spatial relations between features are available as relational data. For instance the example query "All wellbores within the Ekofisk field" can be solved by selecting the Ekofisk field on a map, and selecting the "contains" relation between fields and wellbores, which is found in the data set. For 2, the selected region will not in general correspond to any given feature in the data set. Therefore any query concerning containment in or overlap with a manually selected region will require some actual geospatial database operations.

As reported in deliverable D6.2, a geospatial extension of the ontop query transformation module has been developed by partner UoA in Y2. However, 1) this development is not yet integrated into the Optique platform, 2) it relies on a data storage layer that supports the required geospatial operations through a geospatial extension of SQL. Neither the Statoil data store, nor the storage we used for the NPD data set currently support these operations.

This is why our efforts in Year 1 have been restricted to variant 1, which allows the selection of features on a map, but does not require actual geospatial operations from the query transformation and execution components. Our development is described in Sect. 3.1.

More important for the Optique vision than geospatial query formulation, is the ability to interface with existing tools used by domain experts, to allow analysis of data, once the data access (i.e. query processing) is done. For this, we have developed functionality to *export the results* of a query in a standard format that can be read by Geographical Information Systems, to display them as a map layer, see Sect. 3.2.

In Year 3, we will attempt to tighten the integration between geospatial tools and Optique: Instead of selecting features (and later regions) on a map widget of our own construction, it should be possible to select them in an existing mapping tool used in the domain (like ArcGIS or Petrel), and transfer directly from those tools to OptiqueVQS, essentially making the query formulation appear as a plug-in to the existing tools, rather than a new and separate system. Conversely, the output from queries, that is now written into file that can be imported by other tools, should become more directly available to GIS software.

3.1 Support for Map-based Selection of Features

Support for simple geo-spatial query formulation by selecting features that are explicitly represented in the data set are added to the OptiqueVQS with the *map widget*. The map widget is implemented as a JavaScript client application that uses the OpenLayers JavaScript library[1] to read and render geographical data represented as GeoJSON[2] on maps. The GeoJSON files are constructed from shape files[3] that are available from the NPD FactPages website.[4] The shape files contain geo-spatial data related to most of the entities on the Norwegian Continental Shelf (NCS) that have a geographical location, e.g., wellbores, fields, pipelines and seismic surveys.

The map widget is enabled in the OptiqueVQS by annotating those concepts in the ontology to which geographical data exist via the map widget, see Chapter 6. In the OptiqueVQS, this is indicated by the map marker icon appearing on typically the name attribute in the form-based widget of the OptiqueVQS,

[1] URL: `http://openlayers.org`
[2] URL: `http://geojson.org/`
[3] URL: `https://en.wikipedia.org/wiki/Shapefile`
[4] URL: `http://factpages.npd.no/factpages/`

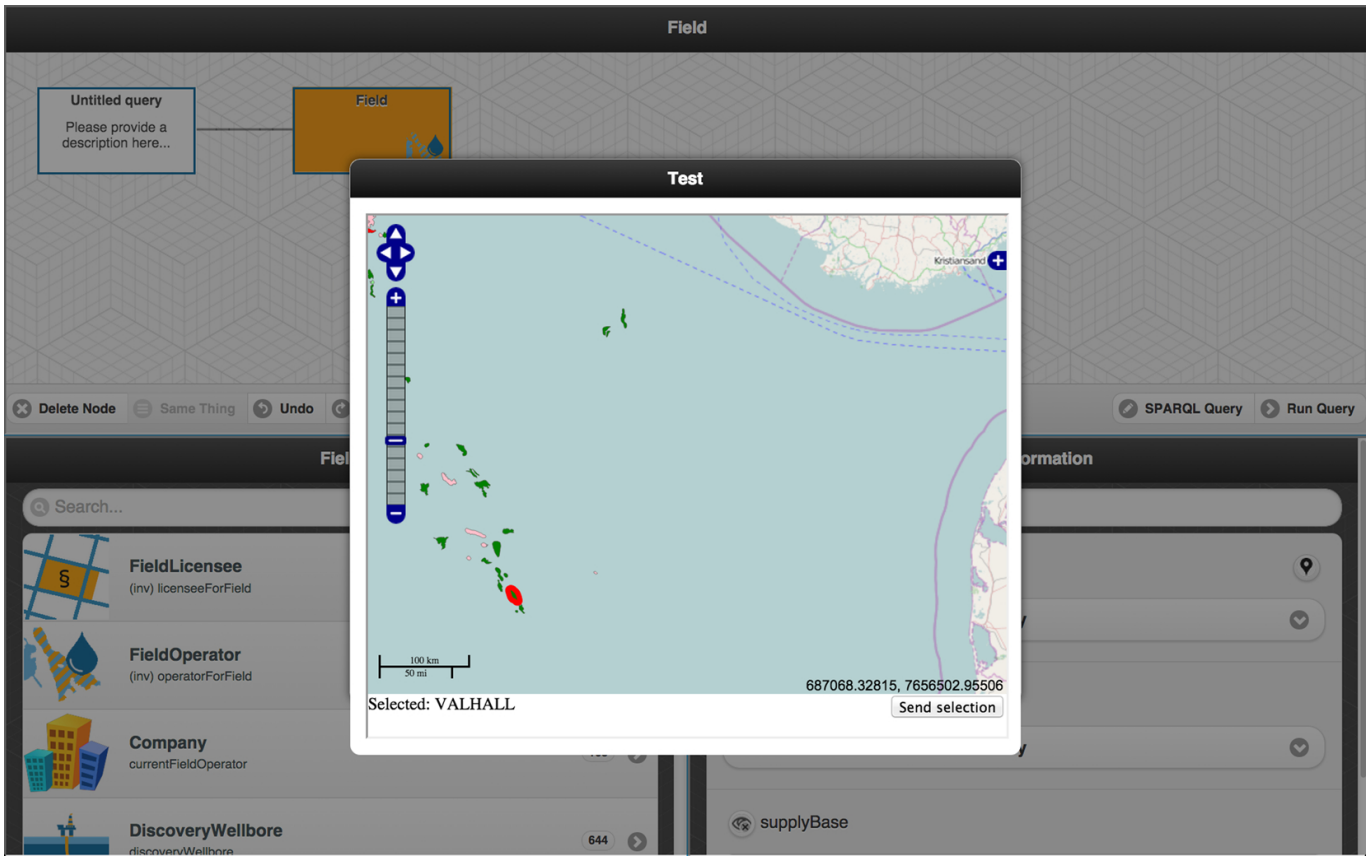


Figure 3.1: The map widget of the OptiqueVQS showing (an excerpt) of all fields on the Norwegian Continental Shelf; the Valhall field is selected by the user.

see Figure 3.1. Clicking the marker opens the map widget in a new window. The window displays the geo-spatial data related to the currently selected concept in the OptiqueVQS on a map. All other geo-spatial data available to the map widget may also be added to the map as additional map layers in order to make the map easier to navigate; for instance, in the case of NCS it is very common to include blocks and outlines of field areas in maps. The user may select a feature belonging to the map layer associated with the current selected concept in the OptiqueVQS, upon which the value of the attribute to which the marker icon appeared is displayed in the map widget; in Figure 3.1 the Valhall field is selected. When the selection attribute value is returned back to the form-based widget, it is displayed there as a value restriction.

The current map widget is only a "proof of concept" implementation, motivated by the prominent role geo-spatial query formulation has in existing end-user tools in the petroleum exploration domain. Notable shortcomings with the current implementation is that the geo-spatial data in the GeoJSON files is not retrieved directly from the ontology or the data source. Instead, it is currently prepared up front, and externally to the Optique system. Special care must be taken to ensure that the geo-spatial data is synchronised with the data at the data sources. Also, since the current map widget reads designated files for geo-spatial data, it must keep a mapping that controls which concepts in the ontology correspond to which geo-spatial data, and is therefore currently not able to generically handle arbitrary ontologies and data sets that represent and contain geo-spatial data.

In the remainder of the project, we intend to define the content of the map layers shown in the map widget or third party mapping tools within the Optique platform, using OBDA technology to access the data.

3.2 Export of Geo-spatial Query Results

Being able to project query results using well-established GIS applications is useful for the Statoil engineers. This requirement that was inferred from the first user evaluation workshop in Statoil, was addressed in the second year of the project. In order to offer a solution in this respect, we had to address the following issues:

- *Client-based or Server-based approach.* Our first approach was client-based; Since ArcGIS 10.0 is the GIS application which is used most by the Statoil engineers, we developed a customized tool inside ArcGIS in Python that retrieves results in CSV format and projects them on a map. However, this customized solution is not optimal, as for different versions of ArcGIS the tool should be adjusted accordingly, and of course such a solution would offer compliance with ArcGIS, which is the most popular but only one of the many GIS applications that exist nowadays. So eventually followed a server-based approach in order to enable the optique platform to export query results in a format which could be compliant with more GIS applications, i.e., an ESRI-compliant format. The idea is that the query results that contain geometries could then be exported as a geospatial file format that could be opened using *any* ESRI-compliant GIS application.
- *File format for geospatial data.* Another issue we had to tackle was to decide which ESRI-compliant file format would be supported by the platform, as standard SPARQL endpoints do not include this kind of formats. We decided to support the KML format. Keyhole Markup Language (KML) is an XML notation for expressing geographic annotation and visualization of geometries. It was introduced by Google, and in 2008 it became a standard of the Open Geospatial Consortium (OGC). Since then, it has been supported by all ESRI-compliant GIS applications, such as Google Earth, ArcGIS, QGIS, etc. In KML, the geometries, along with *auxiliary* information can be represented. This is the same approach that was followed for the web interface of the spatiotemporal RDF store Strabon[22, 4] (i.e, the Strabon endpoint), which is available open source[5]. The KMLWriter that was developed and integrated into the Optique platform is based on the respective module of the Strabon endpoint.

Figure 3.2 depicts a SPARQL query that retrieves the locations of wellbores that are contained in the NPD dataset. By selecting the option "KML" in the "Results format" drop-down menu, the user can download the results of the query as a KML file. We demonstrate this new feature of the Optique platform, by using three well-known applications for displaying and managing geospatial data:

- *ArcGIS.* ArcGIS[6] is one of the most popular GIS used mainly for creating maps. It is also extensively used by the Statoil engineers. The version that we use is an instance that runs natively on a Statoil server.
- *Google Earth.* Google Earth[7] is another widely used application used for managing geospatial data and displaying them on the map. It comes with three different licenses (we use the free version).
- *Sextant.* Sextant[3] is an open source[8], web-based browser and visualizer that can be used to produce thematic maps by layering geospatial information which exists in a number of data sources ranging from standard SPARQL endpoints, GeoSPARQL endpoints, or well-adopted geospatial file formats, such as KML and GeoJSON. We use Sextant as a representative of the family of the "linked geospatial data" browsers. By this way, geospatial data coming from the Optique platform could be combined with linked geospatial data coming from other sources (e.g., other SPARQL or GeoSPARQL[12] endpoints) .

Figure 3.3(a) shows how the exported file is opened using an instance of ArcGIS 10.0 (the ArcMap component) that runs natively in a Statoil server. Note that this version of ArcGIS, requires the use of the

[5] http://www.strabon.di.uoa.gr/
[6] https://www.arcgis.com/home/
[7] https://www.google.com/earth/
[8] http://sextant.di.uoa.gr/

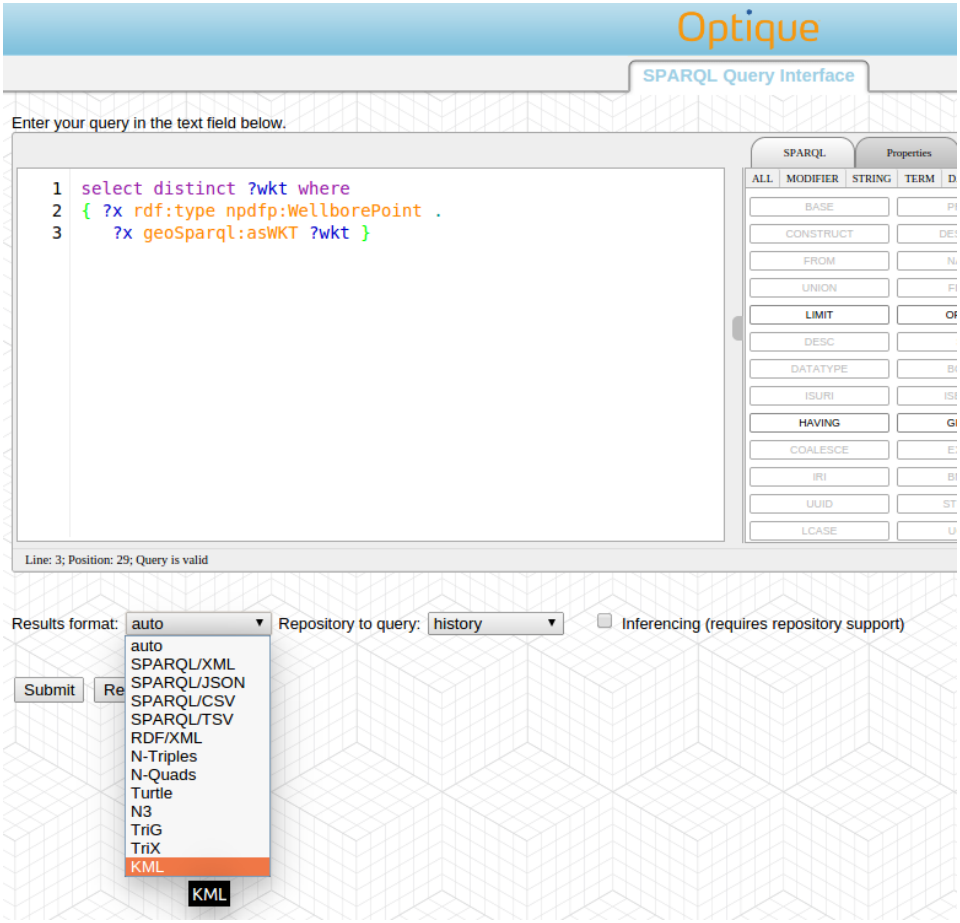


Figure 3.2: The results can be exported as a KML file.

built-in tool "KML to Layer" in order to project a KML file on a map. In newer versions of ArcGIS, this tool is no longer needed and KML files can be opened directly using the "Add Data" option. Figure 3.3(b) shows the same file opened with Google Earth and Figure 3.3(c) shows the file imported to an instance of Sextant.

(a) (b) (c)

Figure 3.3: The KML file opened with ArcGIS 10.0 (a), Google Earth (b), and Sextant (c) .

Chapter 4

Support of Temporal and Streaming Query Formulation

According to the requirements of the Siemens use case with sensor data, in WP5 a query language named STARQL, is developed (see D5.1 and D5.2) in order to handle the stream and temporal data.

As indicated in D5.2, instead of the reified approach for ontology-level stream processing, STARQL offers the means to attach to facts (rather than to reified objects such as measurements) times at which they hold. In our first and current attempt to bridge OptiqueVQS and STARQL, we tried to partially address this non-reified approach, basically by binding a stream widget to individual time-stamped data properties (with a relevant icon next to form element representing the corresponding data property – similar to map widget), so as to allow users to apply relevant operators, such as window and pulse. In Fig. 4.1, an example is depicted, which asks for an update of wellbore bottom heats for the last 1 hour in every 60 seconds. This example assumes that wellbore bottom heat property is timestamped and annotated as such.

As far as temporal querying is concerned, STARQL offers a unified mechanism to accommodate both temporal and stream queries. Currently, however, OptiqueVQS does not fully support all STARQL features. So in year 3 we are going to investigate the trade-off between the expressivity of STARQL and usability of its particular features for our use-cases. Then we will study a unified and homogeneous solution to implementation of the relevant fragment of STARQL in OptiqueVQS.

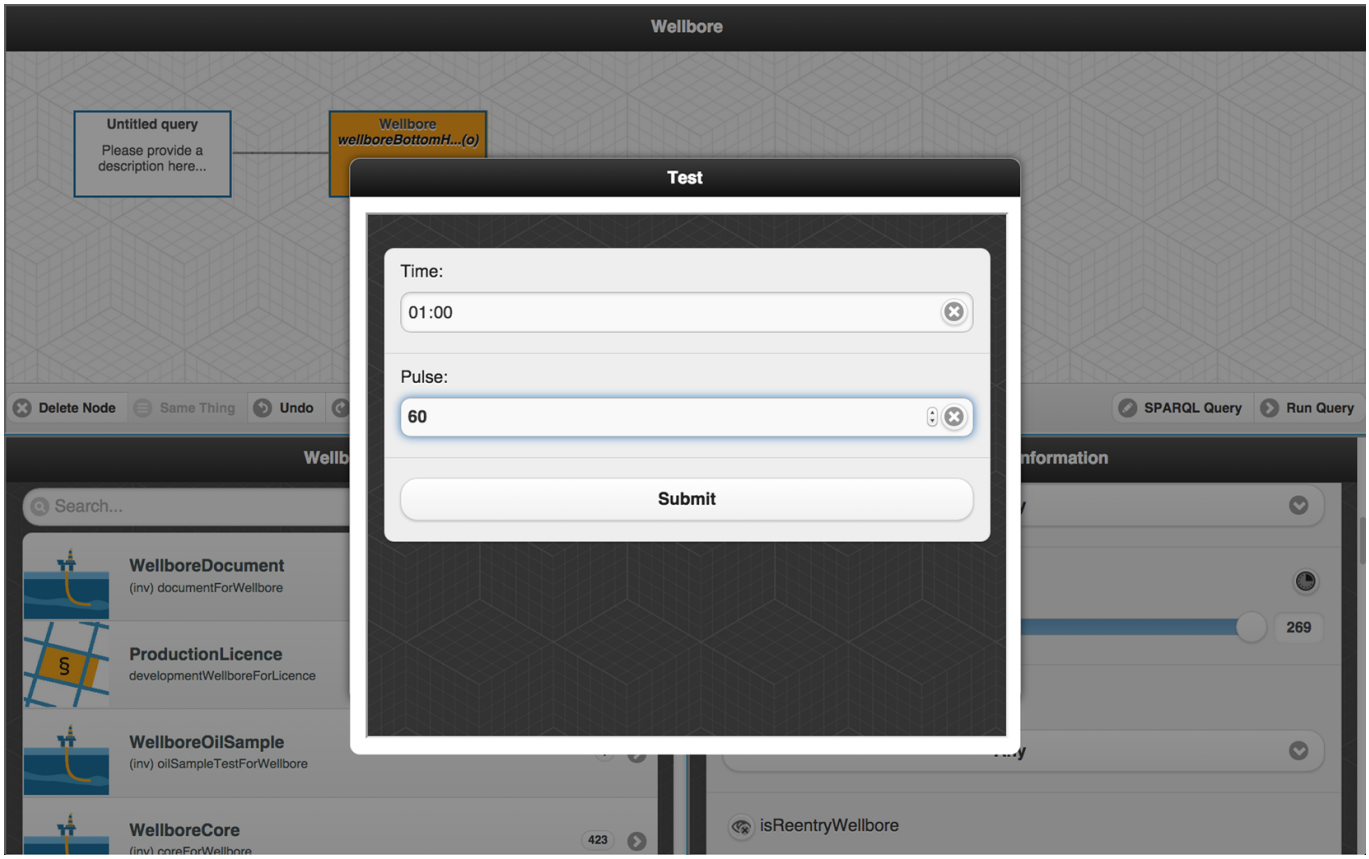


Figure 4.1: Stream widget in action

Chapter 5

Semantic Graph for Query Formulation

An ontology models the reality in "object-oriented" terms: the domain of discourse consists of *objects* that can form *classes*, are connected to each other via *roles*, and can have *properties*. Information needs coming from end-users are often formulated in these terms.

Example 5.0.1. *Consider the following information need coming form an end-user:*

> *Return me names of the companies that are licensed to work on the wellbores in Area A.*

Here we have that the user is interested in finding those objects from the class "Companies" that are connected via the property "Licensed" to the objects of the class "Wellbores in Area A". The last class, in turn, is also composite: it consists of objects of the class "Wellbores" that have the property "Located in Area A".

That is, based on this assumption, our conjecture is that a visual query interface that supports query formulation in these terms should help the user to *(i)* find and build these (probably composite) classes and *(ii)* connect them via roles. Thus, there is a need to develop techniques that would help such a system in these tasks. In what follows we present such technique which we call *semantic graph*. Some results presented in this chapter were published in [29, 1] (see Appendices A and D).

We start with basic definitions.

5.1 Basic Definitions

We use standard notions from first-order logic. We assume pairwise disjoint infinite sets of *constants*, *unary predicates* and *binary predicates*. A *signature* is a subset of constants, unary and binary predicates. We treat equality $\approx$ as an ordinary binary predicate and assume that any set of formulae contains the axioms of equality for its signature. We treat $\top$ as a special unary predicate, which is used to represent a tautology.

A *fact* is a ground atom and a *dataset* is a finite set of facts. A *rule* is a sentence of the form $\forall \mathbf{x} \forall \mathbf{z} [\varphi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})]$, where $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{y}$ are pairwise disjoint tuples of variables, the *body* $\varphi(\mathbf{x}, \mathbf{z})$ is a conjunction of atoms with variables in $\mathbf{x} \cup \mathbf{z}$, and the *head* $\exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})$ is an existentially quantified non-empty conjunction of atoms $\psi(\mathbf{x}, \mathbf{y})$ with variables in $\mathbf{x} \cup \mathbf{y}$. Universal quantifiers are often omitted for brevity. The restriction of $\psi(\mathbf{x}, \mathbf{y})$ being non-empty ensures satisfiability of any set of rules and facts, which makes query results meaningful.

OWL 2 defines three *profiles*: weaker languages with favourable computational properties [23]. Each profile ontology can be normalised as rules and facts using the correspondence of OWL 2 and first-order logic and a variant of the structural transformation.[1] Thus, we see an *ontology* as a finite set of rules. The results presented in this chapter are applicable to all three profiles, although we are interested only in OWL 2 QL. An OWL 2 QL ontology can be represented as a set of rules of the form as in Table 5.1.

[1] Note that the profiles provide the special concept $\bot$, which is immaterial to query answering over satisfiable profile ontologies.

(1) $R(x,y) \rightarrow S(x,y)$,	(2) $A(x) \rightarrow \exists y.[R(x,y) \wedge B(y)]$,
(3) $A(x) \rightarrow B(x)$,	(4) $R(x,y) \rightarrow A(x)$,
(5) $R(x,y) \rightarrow A(y)$,	(6) $R(x,y) \rightarrow S(y,x)$,
(7)$R(x,y) \wedge B(y) \rightarrow A(x)$	

Table 5.1: Rules corresponding to OWL 2 QL profile

5.2 Semantic Graph

We capture the sets of objects and relations between them that are relevant to an ontology $\mathcal{O}$ (and possibly data $\mathcal{D}$) in what we call a *semantic graph*. The graph can be seen as a concise representation of $\mathcal{O}$, and our interface generation and update algorithms are parameterised by such graph rather than by $\mathcal{O}$ itself.

The nodes of a semantic graph are possible "minimal" elements for building composite classes (unary predicates and constants), and edges are labelled with possible relations between such elements (binary predicates and a special symbol type). The key property of a semantic graph is that every X-labelled edge (v, w) is justified by a rule or fact entailed by $\mathcal{O} \cup \mathcal{D}$ 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 $\mathcal{O} \cup \mathcal{D}$; *(ii) universal*, where (each instance of) v is X-related only to (instances of) w in the models of $\mathcal{O} \cup \mathcal{D}$; and *(iii) typing*, where $X = \mathsf{type}$, and (the constant) v is entailed to be an instance of (the unary predicate) w. Formally:

Definition 5.2.1. *A* semantic graph *for $\mathcal{O}$ and $\mathcal{D}$ is a directed labelled multigraph G having as nodes unary predicates or constants from $\mathcal{O}$ and $\mathcal{D}$ 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. $\mathcal{O} \cup \mathcal{D} \models \alpha_e$ 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

$$R(c,d) \quad or \quad R(c,y) \rightarrow y \approx d;$$

(ii) if e is $c \xrightarrow{R} A$, then α_e is a rule of the form

$$\top(c) \rightarrow \exists y.[R(c,y) \wedge A(y)] \quad or \quad R(c,y) \rightarrow A(y);$$

(iii) if e is $A \xrightarrow{R} c$, then α_e is a rule of either of the form

$$A(x) \rightarrow R(x,c) \quad or \quad A(x) \wedge R(x,y) \rightarrow y \approx c;$$

(iv) if e is $A \xrightarrow{R} B$, then α_e is a rule of the form

$$A(x) \rightarrow \exists y.[R(x,y) \wedge B(y)] \quad or \quad A(x) \wedge R(x,y) \rightarrow B(y);$$

(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)-(iv)* encodes the existential (resp., universal) R-relation between nodes in e, whereas *(v)* encodes typing. A graph may not contain all justifiable edges, but rather those that are deemed relevant to the given application.

Example 5.2.1. *Recall our example ontology from Figure 2.2. A semantic graph may contain nodes for* Wellbore, Company*, as well as for* Statoil *which is an instance of the class* Company*. Example edges are: (i) a* drillingOpCompany*-edge linking* Wellbore *to* Company*, which is justified by the axiom* $\mathsf{Wellbore}(x) \wedge \mathsf{drillingOpCompany}(x,y) \rightarrow \mathsf{Company}$*; or (ii) a* type*-edge from* Statoil *to* Company*, which is justified by the fact* Company(Statoil)*.*

It follows from the following proposition that semantic graph computation can be efficiently implemented. In practice, the graph can be precomputed when first loading data and ontology, stored in RDF, and accessed using SPARQL queries. In this way, reasoning tasks associated to search are performed offline.

Proposition 5.2.2. *Checking whether a directed labelled multigraph is a semantic graph for $\mathcal{O}$ and $\mathcal{D}$ is feasible in polynomial time if $\mathcal{O}$ is in OWL 2 QL.*

Proof. It suffices to show that checking whether an edge in the graph is justified is feasible in polynomial time. We show that checking entailment for each different type of rule or fact α is feasible in polynomial time for OWL 2 QL.

Cases $\alpha_e = R(c,d)$ and $\alpha_e = A(c)$. As is known, fact entailment is tractable for OWL 2 QL.

Case α_e is a rule of the form $\gamma_1 \dots \gamma_n \rightarrow \eta$, where η is a single atom without existentially quantified variables. Consider a substitution $\sigma = \{x \mapsto e, y \mapsto f\}$ with e and f fresh constants not occurring in $\mathcal{O}$. Then, $\mathcal{O} \models \alpha_e$ iff $\mathcal{O} \cup \{\sigma(\gamma_i)\}_{i=1}^{n} \models \sigma(\eta)$. Tractability of checking $\mathcal{O} \models \alpha_e$ then follows immediately from tractability of fact entailment in OWL 2 QL.

Case $\alpha_e = A(x) \rightarrow \exists y.[R(x,y) \wedge B(y)]$. Tractability of checking $\mathcal{O} \models \alpha_e$ follows from tractability of subsumption checking for OWL 2 QL.

Case $\alpha_e = \top(c) \rightarrow \exists y.[R(c,y) \wedge A(y)]$. We have that $\mathcal{O} \models \alpha_e$ iff $\mathcal{O} \cup \{\top(c)\} \models \exists y.[R(c,y) \wedge A(y)]$. We have that $\mathcal{O} \cup \{\top(c)\} \models \exists y.[R(c,y) \wedge A(y)]$ iff c is an instance of the concept $\exists R.A$ w.r.t. $\mathcal{O}$, a tractable problem for OWL 2 QL. □

Note that this result can be extended to other OWL 2 profiles [1]; this, however, is out of the scope of the deliverable. Also note that the data in OBDA setting is stored in (relational) databases; thus, we might need to materialise some relevant fragment of it (e.g., such information as names of companies) that can be crucial for search, in order to compute the semantic graph.

To realise the idea of ontology and data guided navigation, we require that interfaces *conform to* the semantic graph in the sense that the presence of every element on the interface is supported by a graph edge. In this way, we ensure that interfaces mimic the structure of (and implicit information in) the ontology and data and that the interface does not contain irrelevant (combinations of) elements.

5.2.1 Query Conformation to Semantic Graph

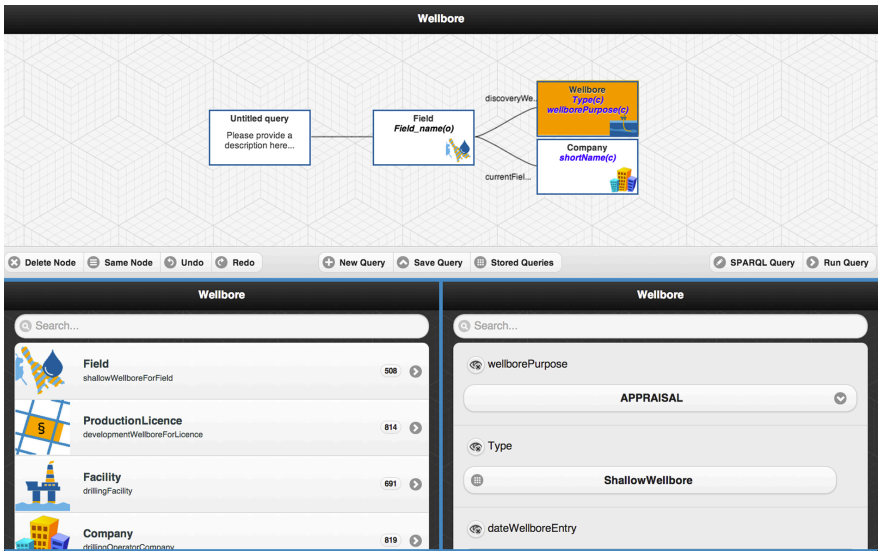
Our goal is to help a user to construct such queries that would be "justified" by the semantic graph. Since our interface currently supports tree-shaped queries, we focus on this class of queries. We assume that all the definitions in this section are parametrised with a fixed ontology $\mathcal{O}$ and dataset $\mathcal{D}$.

Definition 5.2.3. *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 5.2.4. *Let Q be a conjunctive query and G_S a semantic graph. We say that Q* conforms to *G_S 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_S.*
- *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_S.*
- *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_S.*
- *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_S.*




```
SELECT DISTINCT ?c1 ?a1 ?c2 ?c3 WHERE{
  ?c1 ns1:type ns2:Field.
  ?c2 ns1:type ns2:Company.
  ?c3 ns1:type ns2:Wellbore.
  ?c3 ns1:type ns2:ShallowWellbore.
  ?c1 ns2:currentFieldOperator ?c2.
  ?c1 ns2:discoveryWellbore ?c3.
  ?c1 ns2:name ?a1.
  ?c2 ns2:name "Statoil Petroleum AS".
  ?c3 ns2:wellborePurpose "APPRAISAL".
}
```

Figure 5.1: Interface of OptiqueVQS

5.3 OptiqueVQS

OptiqueVQS has three widgets (see Figure 5.1, left): W1 (the top part of the interface) employs the graph metaphor, gives an overview of the constructed query, and allows further manipulation of it; W2 (the left-bottom part of the interface) employs the menu-based representation paradigm to visualise suggestions that users can use to extend the query; and W3 (the right-bottom part of the interface) employs the form-based representation paradigm to visualise possible constraints (projection and selection) that users can set on different parts of the queries.

Query construction process in OptiqueVQS works as follows [25].

- The user starts with selecting in W2 a "starting" suggestion, i.e., a class, from the list of available ones and the selected suggestion appears in W1 and becomes "active".

- Then, the user can extend the query either by selecting in W2 one of the offered suggestions, i.e., a class reachable from the active suggestion via some object property, or by setting constraints, i.e., by restricting in W3 the data properties of the objects belonging to the class of the active suggestion. W1 displays all selected suggestions and organise them in a tree. The user can change the active suggestion by clicking on the ones in W1, or by adding a new one through W2. For each active suggestion OptiqueVQS automatically generates relevant further suggestions in W2 and constraints in W3.

The generation is done via reasoning over the semantic graph, which contain edges that are justified by rules of the form $A(x) \rightarrow \exists y.\big[R(x,y) \wedge B(y)\big]$ entailed from the ontology, and of the form $R(a,\ell)$ extracted from the data, where R is a data property (see Definition 5.2.1). To this end we exploit the HermiT reasoner [24].

Note that users have partial control on output variables, can delete fragments of constructed queries, access query catalogue, save/load queries, and undo/redo actions.

5.3.1 Queries of OptiqueVQS.

In this part we describe the class of queries that can be generated using OptiqueVQS and show that they conform to the semantic graph underlying the system.

First, observe that the OptiqueVQS queries follow the following grammar:

$$\begin{aligned}
\texttt{query} &::= A(x)(\wedge\texttt{constr}(x))^*(\wedge\texttt{expr}(x))^*, \text{ where } A \text{ is an atomic class,}\\
\texttt{expr}(x) &::= \texttt{sug}(x,y)(\wedge\texttt{constr}(x))^*(\wedge\texttt{expr}(y))^*,\\
\texttt{constr}(x) &::= \exists y\, R(x,y) \mid R(x,y) \mid R(x,c), \text{ where } R \text{ is a data property,}\\
\texttt{sug}(x,y) &::= Q(x,y) \wedge A(y), \text{ where } A \text{ and } Q \text{ are atomic class and object property,}
\end{aligned}$$

where variables y in different expressions $\texttt{expr}(x)$ of a structure `str` are different. An OptiqueVQS `query` is constructed using suggestions `sug` and constraints `constr`, that are combined in expressions `expr`. Such queries are clearly 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. An example of such a query can be found in Figure 5.1, right.

When users interacts with OptiqueVQS,

- They start with a "starting" class, as described above. Clearly, this initial query conforms to any semantic graph, including the one, underlying the system.
- Then, the system suggest the list of $\texttt{sug}(y,z)$ via W2 and of $\texttt{constr}(x)$ via W3 such that choosing any of them would leave the updated query conforming to the underlying semantic graph. In other words, all these choices are justified by the graph.

Chapter 6

Backend Support

This chapter describes the support provided by the backend in order to feed and populate the OptiqueVQS interface with the semantic information stored in the ontology.

6.1 Annotation Support

The Optique Visual Query Formulation Interface (OptiqueVQS) is driven by the information available in the (bootstrapped) ontology. We observed in our user study about the OptiqueVQS [28], that a purely axiom driven query interface suffers from important practical limitations, e.g., it does not allow users to set specific data values in queries, e.g., company/operator names. To address this issue we have enriched the ontology with annotations[1] which we precompute, i.e., materialise, from the DBs underlying a given OBDA deployment instance by 'executing' relevant mappings. E.g., we precomputed values that are frequently used, rarely changed, and from relatively small domains; this includes names of companies and oilfield, geolocations, temporal information, ranges of numerical values, e.g., min/max possible depth of wellbores. The OptiqueVQS has also been extended with a module to automatically customize the query interface by displaying data values as pre-populated dropdown lists and range sliders (see Figure 6.1).

In addition multiple property domains and ranges (i.e. dijunction of classes as domain and ranges) is not permitted in OWL 2 QL. Thus an OWL 2 QL approximator as the implemented in Optique [10, 11] will approximate or remove these axioms. However, although this information may not have a crucial impact in the rewriting process, it does have an important role in the OptiqueVQS, as for the list of values and numerical ranges in an OWL data property range. Hence, in order to be able to keep this non OWL 2 QL information, we have added annotations to the ontology about the multiple domains and ranges.

The annotation have been defined using the following annotation schema based on OWL 2 annotations axioms:[2]

- `http://eu.optique.ontology/annotations#geoLocation`: this annotation property is used to annotate class with geo-location information such as *fields* or *wellbores* (see Section 3.1 for details).
- `http://eu.optique.ontology/annotations#temporal`: this annotation property is used to annotate classes with temporal information such as *events* or *measurements*.
- `http://eu.optique.ontology/annotations#data_values`: this annotation property annotates data properties with specific data range values such as *company names* or *field names*.
- `http://eu.optique.ontology/annotations#range_class`: this annotation property annotates object properties with class ranges. For example, the property *hasOperator* is annotated with *Operator* as range class.

[1] Note that, list of values and numerical ranges in an OWL data property range fall outside OWL 2 QL, and thus it should be encoded as non logical axioms

[2] `http://www.w3.org/TR/owl2-new-features/#Extended_Annotations`

- `http://eu.optique.ontology/annotations#domain_class`: this annotation property annotates object or data properties with domains. For example, the property *name* is annotated with *Operator* and *Field* as domain classes. Additionally, this annotation property will also annotate annotation axioms using *range_class* or *data_values*. That is, the annotation axiom for *data_values* representing the list of company names is annotated with *Company* as domain class.

- `http://eu.optique.ontology/annotations#min_values`: this annotation property points out the minimum value in a range of numerical values for a data property.

- `http://eu.optique.ontology/annotations#max_values`: this annotation property points out the maximum value in a range of numerical values for a data property.

- `http://eu.optique.ontology/annotations#hidden`: this annotation will indicate if a data property should be considered for visualization in OptiqueVQS.

6.2 Ontology access

We have implemented a REST interface within the platform so that a selected ontology (stored in the platform's triple store) can feed OptiqueVQS.

Currently the interface exposes publicly, via the platform REST endpoint, the following methods:

- `getAvailableOntologies()`: gets the list of identifiers of the available ontologies in the triple store.

- `loadOntology(String ontologyURI)`: it loads an ontology given its URI.

- `getCoreConcepts()`: gets the core concepts of the active (i.e. loaded) ontology to be listed in the QF component.

- `getConceptFacets(String conceptURI)`: given a concept URI/Id, retrieves its associated facets.

- `getNeighbourConcepts(String conceptURI)`: given an concept URI/Id retrieves the associated concept neighbours.

The QF component will access the above methods by means of REST calls[3] as follows: `http://<SERVER>:<PORT>/REST/JSON/restInterface/?method=method_name&params=[''param1_value'']&id=<CALL_ID>`

Each REST call returns the ontology-related information serialised as JSON objects, which will populate OptiqueVQS. For example, the REST call: `http://fact-pages.fluidops.net/REST/JSON/getQFOntologyAccess/?method=getCoreConcepts&id=1` will return a set of JSON objects corresponding to the core concepts in the ontology.

The JSON objects for concepts include the following information:

- **id:** identifier or URI of the concept (e.g. `http://eu.optique.ontology/statoil_sota_ontology_demo.owl#Production_Licence`)

- **ns:** namespace of the URI (e.g. `http://eu.optique.ontology/statoil_sota_ontology_demo.owl`)

- **name:** name for the concept (e.g. `Production_Licence`)

- **label:** label of the concept for visualization purposes (e.g. "Production Licence"). Usually given in the ontology as a *rdfs:label* annotation.

[3]Note that, in order to automate your request, the QF will be required to pass the username and password (HTML basic authentication) as POST parameters. In a standard setting, these will be admin/iwb, but users can be created and modified.

- **description:** information related to the intended meaning of the concept (usually given in the ontology as a *rdfs:comment* annotation).
- **property:** when retrieving the neighbours for a given concept X this field will contain the JSON object of the object property connecting the concept Y from X.
- **alt_labels:** set of alternative labels (i.e. synonyms) of the concept (usually given in the ontology as *rdfs:label* annotations).

The JSON objects for facets (or data properties) include the following information:

- **id:** identifier or URI of the property (e.g. `http://eu.optique.ontology/statoil_sota_ontology_demo.owl#has_recoverable_oil`)
- **ns:** namespace of the URI (e.g. `http://eu.optique.ontology/statoil_sota_ontology_demo.owl`)
- **name:** name of the data property (e.g. `has_recoverable_oil`)
- **label:** label of the property for visualization purposes (e.g. "Recoverable Oil"). Usually given in the ontology as a *rdfs:label* annotation.
- **description:** information related to the intended meaning of the data property (e.g. "the property allows values from 0 to 100"). Usually given in the ontology as a *rdfs:comment* annotation.
- **type:** type of the data property (e.g. integer, string, boolean).
- **option:** JSON object with the set of allowed values if any (e.g. "minExclusive": "0" or "maxExclusive": "100").
- **alt_labels:** set of alternative labels (i.e. synonyms) of the data property (usually given in the ontology as *rdfs:label* annotations).

The JSON objects for object properties include the following information:

- **id:** identifier or URI of the property (e.g. `http://eu.optique.ontology/statoil_sota_ontology_demo.owl#has_production_licence`)
- **ns:** namespace of the URI (e.g. `http://eu.optique.ontology/statoil_sota_ontology_demo.owl`)
- **name:** name for the object property (e.g. `has_production_licence`)
- **label:** label of the property for visualization purposes (e.g. "Production License"). Usually given in the ontology as a *rdfs:label* annotation.
- **description:** information related to the intended meaning of the object property (usually given in the ontology as a *rdfs:comment* annotation).
- **alt_labels:** set of alternative labels (i.e. synonyms) of the object property (usually given in the ontology as *rdfs:label* annotations).

The semantic information in the ontology, provided to the QF component as JSON objects, will guide the visualization of the elements in the QF interface (QFI). For example: the facet "Recoverable Oil" allows values from 0 to 100 and hence it is represented in the QFI with a slider control (see bottom right side of Figure 6.1); while the neighbours of the concept `Field` are listed in the bottom left side of the QFI.

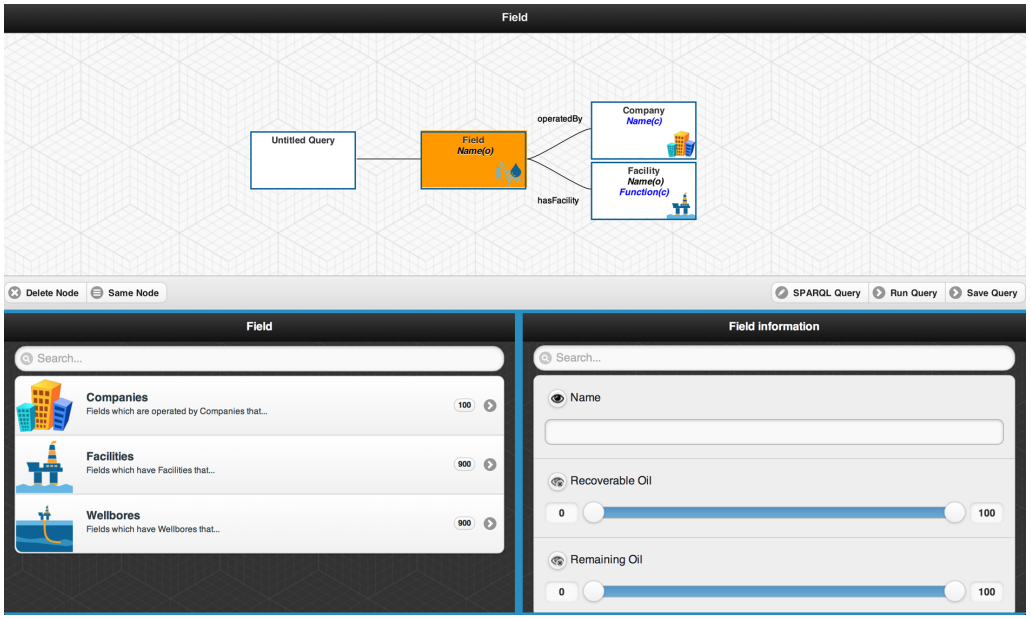


Figure 6.1: Visualization of ontology information in OptiqueVQS.

Chapter 7

User Evaluation and System Demonstration

OptiqueVQS has been evaluated in two settings. The first one has been conducted with casual users on an example movie ontology and the second one at Statoil with NPD ontology. In the following subsections, we report both studies. Also, a demo paper was submitted to ICDE 2015 [19].

7.1 Evaluation with casual users

The experiment was designed as a *think-aloud* study, since the goal of the experiment is not purely *summative*, but to a large extent *formative*. The experiment is built on a "movie ontology". A visual excerpt from the ontology is given in Figure 7.1; note that inverse properties are omitted in the figure for the sake of brevity. In total, the ontology includes 6 concepts, 16 relationships (including inverse properties), and 17 attributes, which already allows us to design complex queries. We avoided having a larger ontology in order to omit the effect of ontology size on the query formulation in this phase.

A total of 15 participants took part in the experiment; the profiles of participants are summarised in Table 7.1. We selected our participants particularly among non-technical people, since they are the primary target of OptiqueVQS. A five minutes introduction of the topic and tool had been delivered to the participants along with an example, before they were asked to fill in a profile survey. The survey asks users about their *age*, *occupation* and *level of education*, and asks them to rate their *technical skills*, such as on programming and query languages, and their *familiarity with similar tools* on a *Likert scale* (i.e., 1 for "not familiar at all", 5 for "very familiar"). Participants were then asked to formulate a set of information needs into queries with OptiqueVQS, given at most three attempts for each. Each participant performed the experiment in a dedicated session, while being watched by an observer. Participants were instructed to think aloud, including any difficulties they encounter (e.g., frustration and confusion), while performing the given tasks.

There were 6 tasks, representing the information needs used in the experiment – see Table 7.2. Each information need maps to a query at different level of complexity with respect to its topology and length, in an increasing order of complexity (all conjunctive): *short linear (T1)*, *long linear (T2)*, *short with branching (T3)*, *long with branching (T4)*, *short with branching and type III cycle (T5)*, and *long with branching and type III cycle (T6)*. Here type III cycle refers to repetition of concepts, that is the query includes at least one instance where a concept appears twice, while long queries are the ones whose maximum tree depth is at least 3.

Once users were done with their tasks, they were asked to fill an exit survey asking about their experiences with the tool. The survey asks users to rate whether the questions were easy to do with the tool (*S1*), the tool was easy to learn (*S2*), was easy to use (*S3*), gave a good feeling of control and awareness (*S4*), was aesthetically pleasing (*S5*), was overall satisfactory (*S6*), and was enjoyable to use (*S7*) on a Likert scale (i.e., 1 for "strongly disagree" and 5 for "strongly agree"). Users were also asked to comment on what they did like and dislike about the tool and to provide any feedback, which they deemed important.

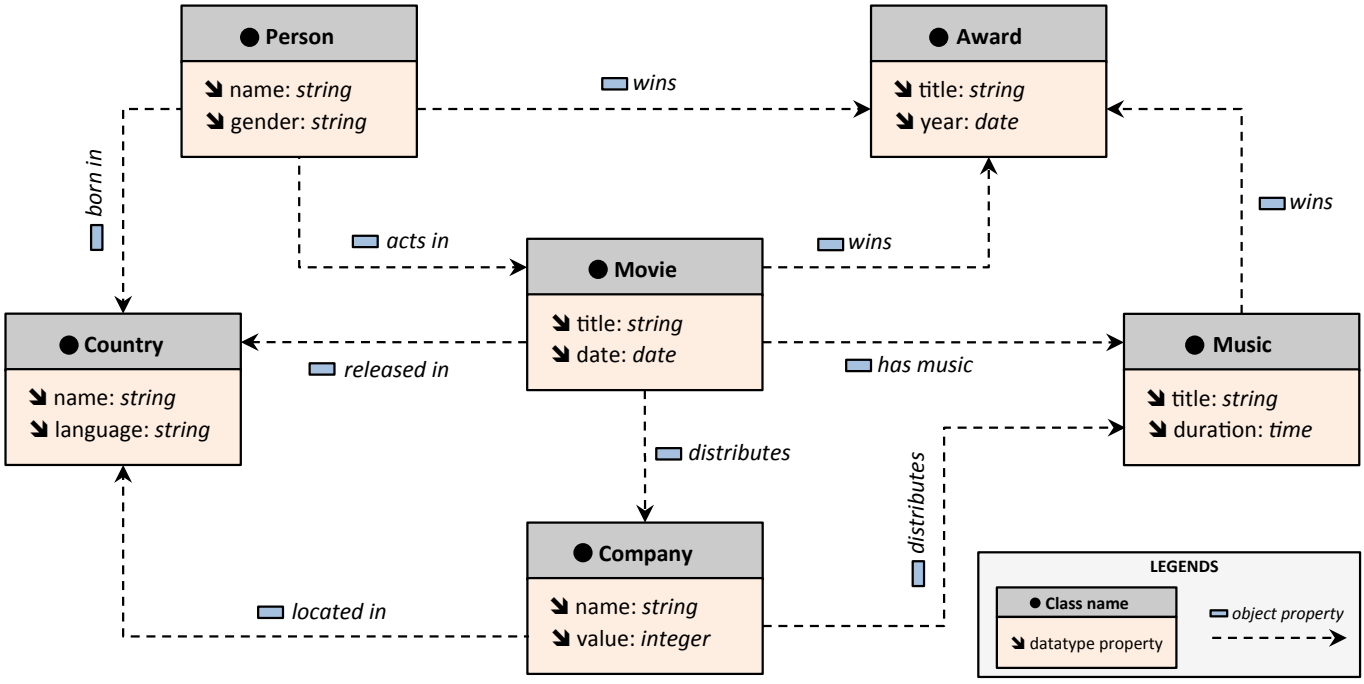


Figure 7.1: A visual excerpt from the movie ontology used in the experiment.

Table 7.1: Profile information of the participants.

#	**Age**	**Occupation**	**Education**	**Technical skills**	**Similar tools**
P1	32	Chemist	PhD	2	3
P2	26	Math teacher	Bachelor	1	1
P3	43	Law student	Master	1	1
P4	21	Political science student	Bachelor	1	2
P5	22	Criminology student	Bachelor	1	3
P6	31	Hydrology student	Master	2	4
P7	26	Complex systems student	Master	2	3
P8	23	Psychology student	Bachelor	1	3
P9	24	Finance student	Bachelor	2	3
P10	21	Law student	Bachelor	2	2
P11	21	Law student	Bachelor	1	1
P12	21	Biology student	Bachelor	1	1
P13	23	Natural sciences student	Bachelor	1	1
P14	24	History student	Bachelor	1	3
P15	22	Biology student	Bachelor	1	1
Avg.	25	-	-	1.3	2.1

Table 7.2: Information needs used in the experiment.

#	**Query type**	**Information need**
T1	Short linear	Find the *names* of all the *companies* that *distribute* a *movie titled* "Titanic".
T2	Long linear	Find the *names* of all the *people* who *acted in* a *movie released between* 1970 and 1980 and *distributed by* a *company located in* Germany.
T3	Short with branching	Find the *titles* of all the *musics* that *won* an *award titled* "Best of Movie Musics" and are *played in* a *movie titled* "The Red Warrior".
T4	Long with branching	Find the *titles* of all the *movies* that are *distributed by* a *company owned by* a *person born in* USA and *have* a *music* that *won award between* 1980 and 1990.
T5	Short with branching and type III cycle	Find the *names* of all the *companies* that *distribute* a *movie titled* "Titanic" and *distribute* a *music played in* a *movie released in* 1980.
T6	Long with branching and type III cycle	Find the *titles* of all the *musics distributed by* a *company located in* the UK and *played in* a *movie* that *has* an actor named "George" who was *born in a country* in the African continent and *won* an *award* in 1990.

7.1.1 Results

The results of the experiment are presented in Table 7.3. Total of 90 tasks were completed by the participants with a 80 percent first-attempt completion rate (i.e., percentage of correctly formulated queries in the first attempt). On average a task took 74 seconds to complete on 1.2 attempts; the first task and fourth task took the shortest and the longest times to complete, on average 34 seconds and 93 seconds respectively. The third task had the highest average in the number of attempts with 1.5, while the first and the sixth tasks had the lowest average in the number of attempts with 1 and 1.1 respectively.

According to the results and our observations, participants solved the first task (i.e., short linear) quite easily. However, when it came to the third task (short with branching), half of the participants failed in their first attempt. This is particularly due to fact that they were mostly not expecting a branching after two linear queries and did not pay attention to the text of the information need. Yet, as soon as they realised the case, they did quickly recover and manipulated their queries accordingly. The average number of attempts then decreased for the subsequent tasks (i.e., all with branching) as users became more aware. Fourth task (i.e., long with branching) took the longest time on average, since after the third task participants paid more attention to clearly understanding the information need. Participants solved the fifth task (i.e., short with branching and type III cycle) comparatively quickly; this was due to the short length of the query and due to the fact that participants did not have any confusion, when a concept appeared twice in the query (only one participant had this confusion and raised it). Finally, participants solved the last task (i.e., long with branching and type III cycle) quite smoothly and with confidence, although it was the longest and the most complex one (i.e., with two branches and one type III cycle). A snapshot from the final query is given in Figure 7.2.

The feedback provided by the participants through the exit survey is presented in Table 7.4 and Table 7.5. Participants overall rated the tool good with 4 out of 5 on average. The first statement (cf. S1 – the questions were easy to do with the tool) had the lowest rank with 3.7; according to our observations, this was mostly due to the texts of the information needs, rather than the tool. The texts describing the information needs (cf. Table 7.2) include a number of relative pronouns along with a passive sentence structure, which make them hard to understand at a first glance and to keep in the short-term memory. Although, this structure was intentionally selected in order to avoid a step-by-step question form, for subsequent evaluations, a different

Table 7.3: The results of the experiment (c for complete, t for time in seconds, and a for attempt count).

	T1			T2			T3			T4			T5			T6			Avg		
#	c	t	a	c	t	a	c	t	a	c	t	a	c	t	a	c	t	a	c	t	a
P1	1	50	1	1	70	1	1	94	2	1	55	1	1	53	1	1	68	1	1	65	1.2
P2	1	48	1	1	83	1	1	80	1	1	113	2	1	60	1	1	70	1	1	76	1.2
P3	1	81	1	1	87	1	1	80	2	1	180	2	1	141	2	1	145	1	1	119	1.5
P4	1	18	1	1	44	1	1	41	1	1	124	2	1	73	1	1	90	1	1	65	1.2
P5	1	32	1	1	85	1	1	62	1	1	74	1	1	82	1	1	85	1	1	70	1.0
P6	1	16	1	1	136	2	1	125	2	1	86	1	1	108	1	1	100	1	1	95	1.3
P7	1	27	1	1	105	2	1	102	2	1	126	2	1	122	2	1	135	1	1	103	1.7
P8	1	75	1	1	47	1	1	78	2	1	54	1	1	48	1	1	71	1	1	62	1.2
P9	1	23	1	1	59	1	1	54	1	1	82	1	1	45	1	1	81	1	1	57	1.0
P10	1	14	1	1	54	1	1	41	1	1	73	1	1	47	1	1	80	1	1	52	1.0
P11	1	17	1	1	42	1	1	65	1	1	53	1	1	105	2	1	60	1	1	57	1.2
P12	1	29	1	1	72	1	1	84	2	1	103	1	1	56	1	1	83	1	1	71	1.2
P13	1	38	1	1	54	1	1	44	1	1	75	1	1	46	1	1	80	1	1	56	1.0
P14	1	28	1	1	96	1	1	65	1	1	58	1	1	54	1	1	60	1	1	60	1.0
P15	1	19	1	1	125	2	1	112	2	1	144	1	1	50	1	1	168	2	1	103	1.5
Avg	1	34	1	1	77	1.2	1	75	1.5	1	93	1.3	1	72	1.2	1	91	1.1	1	74	1.2

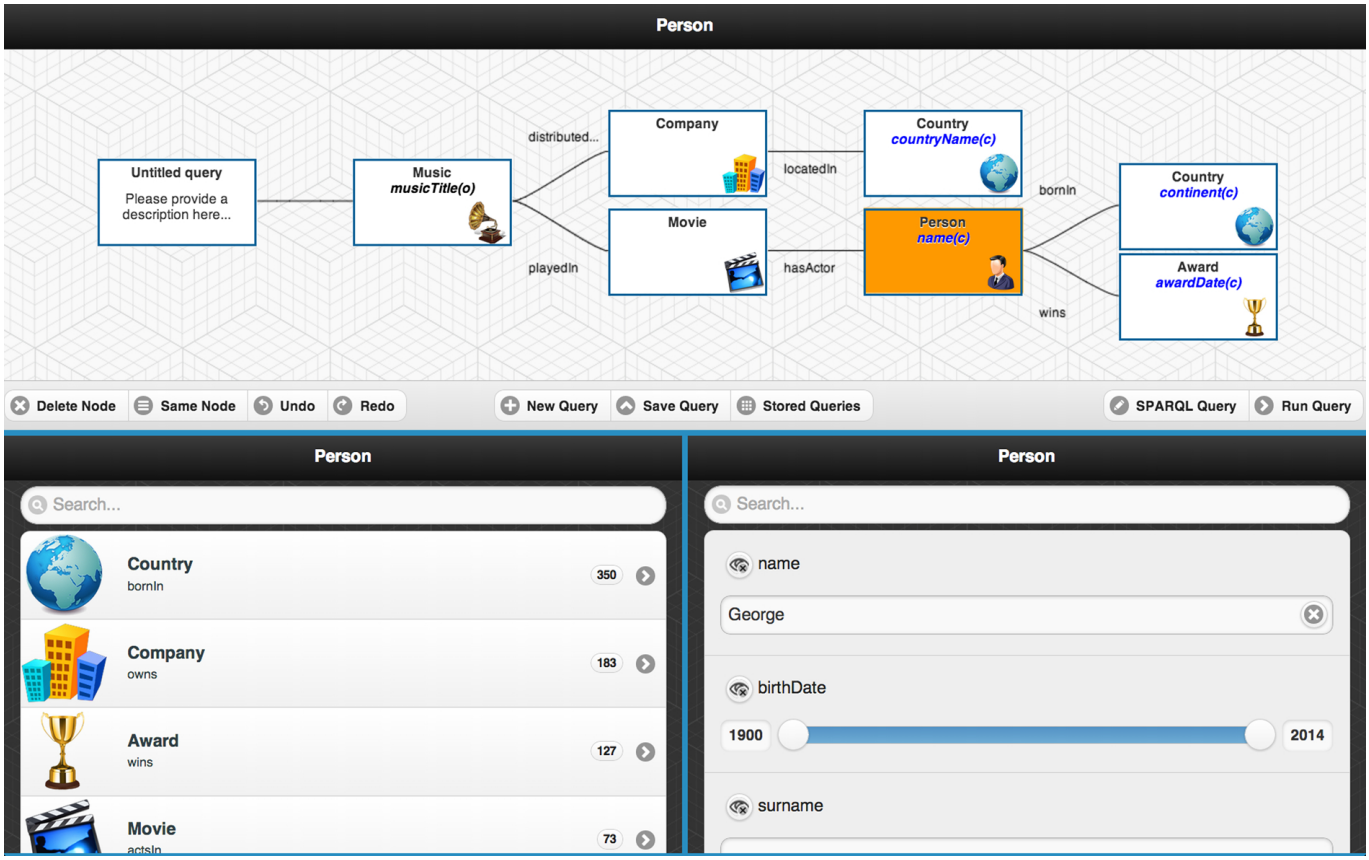


Figure 7.2: An excerpt from a query formulated by the participants during the experiment.

Table 7.4: The results of the exit survey.

#	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**Avg.**
P1	5	5	5	5	4	4	5	4.7
P2	4	4	5	5	5	4	5	4.6
P3	4	4	4	4	4	4	4	4.0
P4	3	4	2	3	4	4	4	3.4
P5	4	4	4	4	4	4	5	4.1
P6	4	5	4	5	5	4	4	4.4
P7	3	3	4	4	4	4	4	3.7
P8	5	5	5	5	4	5	5	4.9
P9	3	4	4	4	4	4	4	3.9
P10	3	3	4	4	3	4	4	3.6
P11	3	4	4	4	4	4	4	3.9
P12	4	4	4	4	4	4	4	4.0
P13	4	3	4	4	4	4	4	3.9
P14	3	3	4	3	4	4	4	3.6
P15	4	4	3	3	4	4	4	3.7
Avg	3.7	3.9	4.0	4.1	4.1	4.1	4.3	4.0

form could be considered. As listed in Table 7.5, participants mainly found the tool orderly. Participants liked the way that queries were visualised, i.e., a diagrammatic overview that users can interact with. They also appreciated the fact that the tool allows them to formulate detailed information needs easily and in an organised way. The introduction given to the users were only around five minutes with an example query, therefore participants were mostly expected to learn on the way, since one of our goals was to have a tool with a low learning curve and effort. This case is reflected and confirmed by the comments of participants.

Observing the participants in action allowed us to acquire some specific insights about the tool. One major issue was that while formulating the fourth task, participants initially looked for a "birth place" field in W2, since the information need was specifying a person born in USA. It took only a while for participants to realise that this information is only accessible through a relationship rather than an attribute. A participant first considered the branches as "OR" rather than "AND" and asked whether it was possible to construct "OR" branches. Two participants realised that indeed they do not have to follow the logical order given in the descriptions of information needs (i.e., to join the concepts in the given order), but the alternatives exist. One of these participants solved one of the tasks successfully with an alternative order. Finally, from a general perspective, users did not have any major difficulties in using and learning the tool and were quick in realising the given tasks. Participants largely stated that their experience with the tool was comparable to the games in terms of the joy they had, raised their interest on the tool, and asked further questions after the experiment, mostly concerning the context that the tool is going to be used.

7.2 Results of the Statoil End-User Workshop

The experiment was designed as a *think-aloud* study similar to the first experiment. The experiment is built on a bootstrapped NPD ontology. In total, the ontology includes 253 concepts, 208 relationships (including inverse properties), and 233 attributes.

A total of 3 participants took part in the experiment; the profiles of participants are summarised in Table 7.6. A five minutes introduction of the topic and tool had been delivered to the participants along with an example, before they were asked to fill in a profile survey. The survey asks users about their *age*, *occupation* and *level of education*, and asks them to rate their *technical skills*, such as on programming and query languages, and their *familiarity with similar tools* on a *Likert scale* (i.e., 1 for "not familiar at all",

Table 7.5: The feedback given by the participants.

#	**Like**	**Dislike**
P1	"Visual, easy to use, fast and easy to correct mistakes."	"...can be visually improved."
P2	"Easy to jump on [diagram] and suggestions [of the W1] were relevant"	-
P3	"Easy and organised. Good for an organised and focused search."	"Nothing"
P4	"I like that it can make search process go faster and make it more specific"	"It could get complicated as you have to link and sometimes go back to previous boxes."
P5	"It was OK to find what the tasks asked for without having to look too long for the right variables."	-
P6	"The schematic diagram"	"It has fixed options."
P7	"Good overview"	-
P8	"The way you connect the nodes, the way it was easy to incorporate a lot of information in the right way, and it was easy to be organised."	"Maybe seems a bit simple at the first glance, but then it was good!"
P9	"Nice visualisation [for diagram]"	"Many steps"
P10	"Easy to use"	-
P11	"It was quite simple."	"It felt I did not have much time [to learn]."
P12	"The organisation in images and scheme"	-
P13	"The scheme on top is pretty helpful to see where you are actually getting to what you are looking for"	"It took me some time to get used to it, but then I think it works!"
P14	"You could really go in to details and ask many things about same person/company etc."	"It was a bit tricky to learn, but I think that it is possible to get a hang on it if you use it for a while"
P15	"Easy access for specific information regarding the search options: movies, music etc."	"Some difficulties [for] managing the correct search option"

Table 7.6: Profile information of the participants.

Question	**P1**	**P2**	**P3**
"Age"	39	40	49
"What is your occupation?"	Geologist	Biostrat	Senior IT Advisor
"What is your level of education?"	Master	Master	Master
"I have technical skills (i.e., computer) such as programming and query languages (e.g., SQL, Java, PHP, SPARQL etc.)"	Neutral (3)	Disagree (2)	Strongly Agree (5)
"I am familiar with tools similar to OptiqueVQS"	Neutral (3)	Strongly Disagree (1)	Agree (4)

5 for "very familiar"). Participants were then asked to formulate a set of information needs into queries with OptiqueVQS, given at most three attempts for each. Each participant performed the experiment in a dedicated session, while being watched by an observer. Participants were instructed to think aloud, including any difficulties they encounter (e.g., frustration and confusion), while performing the given tasks.

There were 9 tasks, representing the information needs used in the experiment – see Table 7.7. The description of task characteristics follows the first experiment.

Once users were done with their tasks, they were asked to fill an exit survey asking about their experiences with the tool. The survey was built on the System Usability Scale (SUS) questionnaire[1] and users were also asked to comment on what they did like and dislike about the tool and to provide any feedback, which they deemed important.

7.2.1 Results

The results of the experiment are presented in Table 7.8. Total of 27 tasks were completed by the participants with 84 percent competition rate and 65 percent first-attempt completion rate. First user has only one incorrect, and second user has no incorrect. The third user has one missing record (task 2), therefore this has been omitted from the results. The third question was asking for fields operated by Statoil. Instead of formulating a Field - Company pair, the third user formulated a Field - FieldOperator pair. This confusion between FieldOperator and Company led him to incorrectly solve the task 5 as well.

In increasing order the task completion times are task 1 (53 s and 1 attempt on average) , task 2 (90 s 1 attempt on average), task 5 (113 s and 1.3 attempt on average), task 3 (142 s and 1.6 attempts on average), task 4 (276 s and 1.3 attempts on average), task 8 (280 s and 1.6 on average), task 9 (391 s and 1 attempt on average), and task 7 (495 s and 2.3 on average). The results suggest that queries with branching and type III cycles take longer time compared to others. Task 7 not only takes the longest time but also highest average attempts. This is particularly due to conceptual mismatch between users understanding of domain and the ontology, which forced users to iterate several times to understand.

The feedback provided by the participants through the exit survey is presented in Table 7.9 and Table 7.10. The usability scores given by participants is considerably low despite high completion rates. According to our observations this is particularly due to: the quality of ontology (i.e., ontology was bootstrapped with little manual fine tuning), the largeness of ontology, incomplete mappings (i.e., for some cases users found alternative means to formulate a given task for which there was no mapping support), insensible information needs (i.e., some of the query tasks did not make sense for the users), and finally lack of training (which has been intentional to observe learnability of the tool).

Although, the results of usability survey indicates low grades, high completion rates suggest that even

[1] http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Table 7.7: Information needs used in the experiment.

#	**Query type**	**Information need**
T1	Single concept	List all fields.
T2	Single concept and property	What is the water depth of the Snorre A platform (facility)?
T3	Short linear	List all fields operated by Statoil Petroleum AS company.
T4	Short with branching	List all exploration wellbores with the field they belong to and the geochronologic era(s) with which they are recorded.
T5	Short linear with type III cycle	List the fields that are currently operated by the company that operates the Alta field.
T6	Long linear	List the companies that are licensees in production licenses that own fields with a recoverable oil equivalent over more than 300 in the field reserve.
T7	Short with branching	List all production licenses that have a field with a wellbore completed between 1970 and 1980 and recoverable oil equivalent greater than 100 in the company reserve.
T8	Long linear	List the blocks that contain wellbores that are drilled by a company that is a field operator.
T9	Short with branching and type III cycle	List all producing fields operated by Statoil Petroleum AS that has a wellbore containing gas and a wellbore containing oil.

Table 7.8: The results of the experiment.

Participant	**Task**	**Correct**	**Attempt**	**Time**
1	1	1	1	87.9
1	2	1	1	91.6
1	3	1	1	121
1	4	1	1	286.6
1	5	1	1	143
1	6	1	1	281.1
1	7	0	3	507
1	8	1	1	162
1	9	1	1	525
2	1	1	1	45.1
2	2	1	1	89.2
2	3	1	1	109.2
2	4	1	2	437
2	5	1	2	102.6
2	6	1	1	490
2	7	1	2	521.6
2	8	1	3	454
2	9	1	1	311.5
3	1	1	1	26.7
3	2	*	*	*
3	3	0	3	197
3	4	1	1	105.4
3	5	0	1	94.9
3	6	0	2	266
3	7	1	2	456.5
3	8	1	1	224.7
3	9	1	1	339.4

Table 7.9: The results of the exit survey.

Question	**P1**	**P2**	**P3**
"I think that I would like to use this system frequently."	Neutral (3)	Agree (4)	Agree (4)
"I found the system unnecessarily complex."	Agree (4)	Agree (4)	Neutral (3)
"I thought the system was easy to use.	Disagree (2)	Disagree (2)	Neutral (3)
"I think that I would need the support of a technical person to be able to use this system."	Agree (4)	Neutral (3)	Agree (4)
"I found the various functions in this system were well integrated."	Neutral (3)	Neutral (3)	Agree (4)
"I thought there was too much inconsistency in this system."	Neutral (3)	Agree (4)	Disagree (2)
"I would imagine that most people would learn to use this system very quickly."	Disagree (2)	Disagree (2)	Agree (4)
"I found the system very cumbersome to use."	Neutral (3)	Agree (4)	Neutral (3)
"I felt very confident using the system."	Disagree (2)	Neutral (3)	Disagree (2)
"I needed to learn a lot of things before I could get going with this system."	Agree (4)	Neutral (3)	Neutral (3)

Table 7.10: The feedback given by the participants.

"What did you like about the tool?"	**Person**
"Ability to explore new relationships"	P1
"Flexibility"	P1
"Graphic frontend"	P2
"Idea of searching different content"	P2
"Easy graphical interface"	P3
"Responsive"	P3
"understandable fucntionality between windows"	P3
"What didn't you like about the tool?"	**Person**
"Similar concept names"	P1
"Mappings are not so easy to understand"	P1
"The meaning of branches"	P1
"Delete node"	P2
"Lack of drag & drop"	P2
"I probably missed some understanding of how to express some of the relations and constraints"	P3
"I needed some training."	P3

with no training and bootstrapped ontology, OptiqueVQS enables its users to formulate queries due to its simplicity and high learnability.

Bibliography

[1] Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Šarūnas Marciuška, and Dmitriy Zheleznyakov. Faceted search over ontology-enhanced rdf data. In *Proc. ACM International Conference on Information and Knowledge Management (CIKM)*, 2014.

[2] Guntis Barzdins, Edgars Liepins, Marta Veilande, and Martins Zviedris. Ontology Enabled Graphical Database Query Tool for End-Users. In *Proceedings of the 8th International Baltic Conference on Databases and Information Systems (DB&IS 2008)*, volume 187 of *Frontiers in Artificial Intelligence and Applications*, pages 105–116. IOS Press, 2009.

[3] Konstantina Bereta, Charalampos Nikolaou, Manos Karpathiotakis, Kostis Kyzirakos, and Manolis Koubarakis. Sextant: Visualizing time-evolving linked geospatial data. In *The 12th International Semantic Web Conference (ISWC2013)*, 2013.

[4] Konstantina Bereta, Panayiotis Smeros, and Manolis Koubarakis. Representation and querying of valid time of triples in linked geospatial data. In Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *The Semantic Web: Semantics and Big Data*, volume 7882 of *Lecture Notes in Computer Science*, pages 259–274. Springer Berlin Heidelberg, 2013.

[5] Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors. *The Adaptive Web: Methods and Strategies of Web Personalization*, volume 4321 of *LNCS*. Springer, 2007.

[6] Stephane Campinas, Thomas E. Perry, Diego Ceccarelli, Renaud Delbru, and Giovanni Tummarello. Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation. In *Proceedings of the 23rd International Workshop on Database and Expert Systems Applications (DEXA 2012)*, pages 261–266. IEEE, 2012.

[7] Jeremy Carroll, Ivan Herman, and Peter F. Patel-Schneider. OWL 2 Web Ontology Language RDF-Based Semantics. W3C Recommendation, W3C, 2012.

[8] Tiziana Catarci, Maria F. Costabile, Stefano Levialdi, and Carlo Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.

[9] Tiziana Catarci, Paolo Dongilli, Tania Di Mascio, Enrico Franconi, Giuseppe Santucci, and Sergio Tessaris. An ontology based visual tool for query formulation support. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI 2004)*, volume 110 of *Frontiers in Artificial Intelligence and Applications*, pages 308–312. IOS Press, 2004.

[10] M. Console, V. Santarelli, and D. Savo. Efficient Approximation in DL-Lite of OWL 2 Ontologies. In *DL*, 2013.

[11] Marco Console, Jose Mora, Riccardo Rosati, Domenico Fabio Savo, and Valerio Santarelli. Effective computation of maximal sound approximations of description logic ontologies. In *Proceedings of ISWC 2014*, 2014.

[12] Open Geospatial Consortium. OGC GeoSPARQL – A geographic query language for RDF data. OGC Candidate Implementation Standard, 02 2012.

[13] Renata Dividino and Gerd Groner. Which of the following SPARQL Queries are Similar? Why? In *Proceedings of the 1st International Workshop on Linked Data for Information Extraction (LD4IE 2013)*, volume 1057 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[14] Bernardo Cuenca Grau, Martin Giese, Ian Horrocks, Thomas Hubauer, Ernesto Jimenez-Ruiz, Evgeny Kharlamov, Michael Schmidt, Ahmet Soylu, and Dmitriy Zheleznyakov. Towards Query Formulation and Query-Driven Ontology Extensions in OBDA Systems. In *Proceedings of the 10th OWL: Experiences and Directions Workshop (OWLED 2013)*, volume 1080 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[15] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, W3C, March 2013.

[16] Hai Huang, Chengfei Liu, and Xiaofang Zhou. Computing Relaxed Answers on RDF Databases. In *Proceedings of the 9th International Conference on Web Information Systems Engineering (WISE 2008)*, volume 5175 of *LNCS*, pages 163–175. Springer, 2008.

[17] E. Kapetanios, D. Baer, and P. Groenewoud. Simplifying syntactic and semantic parsing of NL-based queries in advanced application domains. *Data & Knowledge Engineering*, 55(1):38–58, 2005.

[18] Akrivi Katifori, Constantin Halatsis, George Lepouras, Costas Vassilakis, and Eugenia Giannopoulou. Ontology visualization methods - A survey. *ACM Computing Surveys*, 39(4):10:1–10:43, 2007.

[19] Evgeny Kharlamov, Martin Giese, Peter Haase, Ernesto Jiménez-Ruiz, Christoph Pinkel, Martin G. Skjæveland, Ahmet Soylu, Johannes Trame, Dmitriy Zheleznyakov, Carsten Binnig, Eldar Bjørge, Ian Horrocks, and Arild Waaler. Towards Ontology Based Data Access for Statoil. Submitted to ICDE 2015 Demo Track.

[20] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. SnipSuggest: Context-aware Autocompletion for SQL. *Proceedings of the VLDB Endowment*, 4(1):22–33, 2010.

[21] Kasjen Kramer, Renata Dividino, and Gerd Groner. SPACE: SPARQL Index for Efficient Autocompletion. In *Proceedings of the ISWC 2013 Posters & Demonstrations Track (ISWC-PD 2013)*, volume 1035 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[22] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A semantic geospatial dbms. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, JosianeXavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *The Semantic Web – ISWC 2012*, volume 7649 of *Lecture Notes in Computer Science*, pages 295–311. Springer Berlin Heidelberg, 2012.

[23] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles. *W3C Recommendation*, 2009.

[24] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

[25] The Optique Project. OptiqueVQS. `https://www.youtube.com/watch?v=ks5tcPZVHp0`.

[26] Santanu Saha Ray. *Graph Theory with Algorithms and its Applications*, chapter Subgraphs, Paths, and Connected Graphs. Springer India, 2013.

[27] Ahmet Soylu, Martin Giese, Ernesto Jimenez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, and Ian Horrocks. Towards exploiting query history for adaptive ontology-based visual query formulation. In *Proceedings of the 8th Metadata and Semantics Research Conference (MTSR 2014)*, CCIS, Karlsruhe, Germany, 2014. Springer.

[28] Ahmet Soylu, Martin Giese, Ernesto Jimenez-Ruiz, Guillermo Vega-Gorgojo, and Ian Horrocks. Experiencing optiquevqs: A multi-paradigm and ontology-based visual query system for end users. In *Under Review*, 2014.

[29] Ahmet Soylu, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ernesto Jiménez-Ruiz, Martin Giese, and Ian Horrocks. Optiquevqs: Visual query formulation for OBDA. In Meghyn Bienvenu, Magdalena Ortiz, Riccardo Rosati, and Mantas Simkus, editors, *Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014.*, volume 1193 of *CEUR Workshop Proceedings*, pages 725–728. CEUR-WS.org, 2014.

Appendix A

OptiqueVQS: General

This appendix reports the papers:

- Ahmet Soylu, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ernesto Jiménez-Ruiz, Martin Giese, Ian Horrocks. **OptiqueVQS: Visual Query Formulation for OBDA**. Extended abstract. DL 2014.
- Ahmet Soylu, Martin Giese, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, and Ian Horrocks. **Why Not Simply Google?**. NordiCHI 2014.

OptiqueVQS: Visual Query Formulation for OBDA (Abstract)

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

[1] University of Oslo; [2] University of Oxford

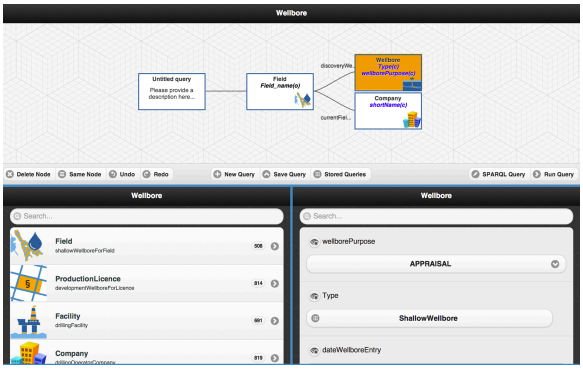
Motivation *Ontology Based Data Access* (OBDA) [16] is a recently proposed prominent approach that aims at providing domain experts with a *direct* access to available enterprise data sources without IT-experts being involved. OBDA is an alternative to centralised approaches, where an IT-expert translates the requirements of domain experts into Extract-Transform-Load (ETL) processes to first integrate the data and then to apply predefined analytical reporting tools. Currently, centralised approaches are commonly used in enterprises; they, however, can become too heavy-weight and inflexible in some cases [12], that can be addressed the OBDA approach.

The key idea behind OBDA is to use *ontologies* to mediate between users and data. Ontologies describe the domain of interest on a higher level of abstraction in terms that are clear for domain experts, and introduce modeling concepts such as inheritance and relationships between classes of objects, thus allowing to describe the intended meaning of the ontological vocabulary. Ontologies have become a common and successful mechanism to describe application domains in, e.g., biology, medicine, the (Semantic) Web [13]. This success is partially due to a number of available formal languages for describing ontologies, including RDF(S) [7] and OWL 2 [5] standardised by W3C.

In OBDA, users formulate their information needs as queries using terms defined in the ontology, and ontological queries are translated into SQL and executed over the data automatically, without an IT-expert's intervention. To this end a set of *mappings* is maintained that describe the relationship between the ontological vocabulary and the elements of the schema of the underlying data.

The standard query language for ontologies is SPARQL [8]. Writing queries using SPARQL, however, is not easy for domain experts and thus intuitive *visual query formulation* support is required for OBDA systems. Existing OBDA systems, e.g., [1, 2, 9, 10, 18–21] typically offer limited or no visual query formulation support. Our goal is to provide a solution for visual query formulation over ontologies that is specifically tailored for OBDA systems. The solution should rely on solid theory, be efficient, support interactive data exploration, and should follow the best Human-Computer Interaction practices to guarantee good usability. In the following we give a short overview of our ideas that were partially implemented in our OptiqueVQS system [22].

OptiqueVQS We first describe functionality of OptiqueVQS' components and then give their formal description. OptiqueVQS is a system for visual query formulation support that allows the user to construct a query over an ontology step by step where at each step the system provides the user with relevant information to continue the query construction. OptiqueVQS has a widget-based architecture and exploits multiple representation and interaction paradigms, see Fig. 1 for a screenshot where a sample query




```
SELECT DISTINCT ?c1 ?a1 ?c2 ?c3 WHERE{
  ?c1 ns1:type ns2:Field.
  ?c2 ns1:type ns2:Company.
  ?c3 ns1:type ns2:Wellbore.
  ?c3 ns1:type ns2:ShallowWellbore.
  ?c1 ns2:currentFieldOperator ?c2.
  ?c1 ns2:discoveryWellbore ?c3.
  ?c1 ns2:name ?a1.
  ?c2 ns2:name "Statoil Petroleum AS".
  ?c3 ns2:wellborePurpose "APPRAISAL".
}
```

Fig. 1. Interface of OptiqueVQS

over an ontology for the Oil and Gas domain is composed[1] together with its SPARQL counterpart. The query asks for oil fields, wellbores operated on these fields, and companies currently exploiting the fields. OptiqueVQS has three widgets: W1 employs the graph metaphor, gives an overview of the constructed query, and allows further manipulation of it, W2 employs the menu-based representation paradigm to visualise suggestions that users can use to extend the query, W3 employs the form-based representation paradigm to visualise possible constraints (projection and selection) that users can set on different parts of the queries.

Query construction process in OptiqueVQS works as follows [3]. The user starts with selecting in W2 a 'starting' suggestion, i.e., a class, from the list of available ones and the selected suggestion appears in W1 and becomes 'active'. Then, the user can extend the query either by selecting in W2 one of the offered suggestions, i.e., a class reachable from the active suggestion via some object property, or by setting constraints, i.e., by restricting in W3 the data properties of the objects belonging to the class of the active suggestion. W1 displays all selected suggestions and organise them in a tree. The user can change the active suggestion by clicking on the ones in W1, or by adding a new one through W2. For each active suggestion OptiqueVQS automatically generates relevant further suggestions in W2 and constraints in W3. The generation is done via reasoning (e.g., extraction of classification, inferred domain and ranges) over the ontology underlying the system and to this end we exploit the HermiT reasoner [17]. Moreover, users have partial control on output variables, can delete fragments of constructed queries, access query catalogue, save/load queries, and undo/redo actions.

Queries of OptiqueVQS. The queries follow the following grammar:

$$\begin{aligned}
\texttt{query} &::= A(x)(\wedge \texttt{constr}(x))^*(\wedge \texttt{expr}(x))^*, \text{ where } A \text{ is an atomic class,}\\
\texttt{expr}(x) &::= \texttt{sug}(x,y)(\wedge \texttt{constr}(x))^*(\wedge \texttt{expr}(y))^*,\\
\texttt{constr}(x) &::= \exists y\, R(x,y) \mid R(x,y) \mid R(x,c), \text{ where } R \text{ is an atomic data property,}\\
\texttt{sug}(x,y) &::= Q(x,y) \wedge A(y), \text{ where } A \text{ and } Q \text{ are atomic class and object property,}
\end{aligned}$$

where variables y in different expressions $\texttt{expr}(x)$ of a structure $\texttt{str}$ are different. An OptiqueVQS $\texttt{query}$ is constructed using suggestions $\texttt{sug}$ and constraints $\texttt{constr}$,

[1] This ontology was designed for Statoil [4] as a part of the Optique project [15].

that are combined in expressions $\texttt{expr}$. Such queries are conjunctive and tree shaped: the graph corresponding to the query where nodes are variables and edges are properties is a tree. 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.

When users interacts with OptiqueVQS, then for every $\texttt{sug}(x, y)$ that an $\texttt{expr}(x)$ starts with, that is, for every active suggestion, the system offers a list of relevant $\texttt{constr}(x)$ via W3 and relevant $\texttt{sug}(y, z)$ via W2 that can be used to construct further $\texttt{expr}(y)$. We explore several notions of relevance, including *local* where offered constraints and suggestions depend on $\texttt{sug}(x, y)$ only, and *global* where they depend on the entire query. We currently investigate complexity of suggestion generation for different ontologies and notions of relevance.

Treatment of Data Properties. An important feature of OptiqueVQS is a special treatment of data properties in W3: it automatically generates different end-user oriented representations of data values, including sliders restricting possible ranges of numerical values, such as age, depths, etc., and drop boxes with precomputed lists for categorical data, such as names of companies, geographical locations, etc. Throughout empirical evaluations we determined that this treatment of data properties is of high importance for end-users. To support different intuitive representations for data properties, we encode relevant information in the ontology underlying the system and generate the representations on the fly.

Query Construction vs Rewriting Ontology. We use OptiqueVQS for query formulation in the Optique OBDA system, and thus we convert queries constructed via OptiqueVQS in SPARQL and then they are processed by the Optique query processing component [20] that rewrites them with the system's ontology and unfolds it with mappings in SQL. We use OWL 2 QL ontologies for query rewriting, while the query construction is based on much richer OWL 2 ontologies that, in particular, make use of nominals. There are both theoretical and practical reasons for having two ontologies: conjunctive query rewriting for OBDA is well studied for OWL 2 QL ontologies [6], while for effective and efficient query support of conjunctive query construction the expressive power of OWL 2 QL ontologies is not sufficient. The query construction ontology that we use in the system extends the query rewriting ontology.

Feedback in Query Construction. To improve query construction experience and to allow data exploration OptiqueVQS provides users with feedback at each step of query construction: the users can see answers relevant to the constructed query. Since computation of answers in OBDA systems is expensive, we investigate several possibilities for the feedback: it can be, for example, a set of sample query answers, or a summary of query answers, or some statistics on query answers. We currently investigate complexity of different types of feedback. Moreover, we investigate influence of different types of feedback on the usability of the system.

To Sum Up. We developed OptiqueVQS in cooperation with Statoil and did preliminary user evaluation with Statoil geologists that gave us encouraging results. We also presented the system at several venues [11, 14, 22, 23]. Currently we investigate theoretical properties of our techniques. We also work on improvements of the system in several directions, e.g., we develop ranking functions for suggestions and constraints.

References

1. http://virtuoso.openlinksw.com/
2. http://www.revelytix.com/content/spyder
3. OptiqueVQS. https://www.youtube.com/watch?v=ks5tcPZVHp0
4. Statoil. http://www.statoil.com/en/Pages/default.aspx
5. W3C: OWL 2 Web Ontology Language. http://www.w3.org/TR/owl2-overview/
6. W3C: OWL 2 Web Ontology Language Profiles. http://www.w3.org/TR/owl-profiles/
7. W3C: Resource Description Framework (RDF). http://www.w3.org/RDF/
8. W3C: SPARQL 1.1 Query Language. www.w3.org/TR/sparql11-query/
9. Bizer, C., Seaborne, A.: D2RQ—Treating non-RDF Databases as Virtual RDF Graphs. In: ISWC (2004)
10. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO System for Ontology-Based Data Access. Semantic Web 2(1), 43–53 (2011)
11. Cuenca Grau, B., Giese, M., Horrocks, I., Hubauer, T., Jimenez-Ruiz, E., Kharlamov, E., Schmidt, M., Soylu, A., Zheleznyakov, D.: Towards Query Formulation, Query-Driven Ontology Extensions in OBDA Systems. In: OWLED (2013)
12. Doan, A., Halevy, A.Y., Ives, Z.G.: Principles of Data Integration. Morgan Kaufmann (2012)
13. Horrocks, I.: What are Ontologies Good for? In: Evolution of Semantic Systems, pp. 175–188. Springer (2013), download/2013/Horr13a.pdf
14. Kharlamov, E., et al.: Optique 1.0: Semantic Access to Big Data: The Case of Norwegian Petroleum Directorate's FactPages. In: ISWC (Posters & Demos) (2013), https://www.youtube.com/watch?v=PToyue4BFXA
15. Kharlamov, E., Jiménez-Ruiz, E., Zheleznyakov, D., Bilidas, D., Giese, M., Haase, P., Horrocks, I., Kllapi, H., Koubarakis, M., Özçep, Ö.L., Rodriguez-Muro, M., Rosati, R., Schmidt, M., Schlatte, R., Soylu, A., Waaler, A.: Optique: Towards OBDA Systems for Industry. In: ESWC (SE). pp. 125–140 (2013)
16. Kogalovsky, M.R.: Ontology-Based Data Access Systems. Programming and Computer Software 38(4), 167–182 (2012)
17. Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research 36, 165–228 (2009)
18. Munir, K., Odeh, M., McClatchey, R.: Ontology-Driven Relational Query Formulation Using the Semantic and Assertional Capabilities of OWL-DL. Knowledge-Based Systems 35, 144–159 (2012)
19. Priyatna, F., Corcho, O., Sequeda, J.: Formalisation and Experiences of R2RML-Based SPARQL to SQL Query Translation Using Morph. In: WWW (2014)
20. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-Based Data Access: Ontop of Databases. In: ISWC. pp. 558–573 (2013)
21. Sequeda, J.F., Miranker, D.P.: Ultrawrap: SPARQL Execution on Relational Data. Journal of Web Semantics 22(0), 19 – 39 (2013), http://www.sciencedirect.com/science/article/pii/S1570826813000383
22. Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I.: OptiqueVQS: Towards an Ontology-Based Visual Query System for Big Data. In: MEDES (2013)
23. Soylu, A., Skjæveland, M., Giese, M., Horrocks, I., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D.: A Preliminary Approach on Ontology-Based Visual Query Formulation for Big Data. In: MTSR. pp. 201–212 (2013)

Why Not Simply Google?

Ahmet Soylu
University of Oslo
Ole Johan Dahls hus
Gaustadallen 23B,
Oslo, N-0373 Norway
ahmets@ifi.uio.no

Evgeny Kharlamov
University of Oxford
Wolfson Building
Parks Road
Oxford, OX1 3QD UK
evgeny.kharlamov@cs.ox.ac.uk

Martin Giese
University of Oslo
Ole Johan Dahls hus
Gaustadallen 23B,
Oslo, N-0373 Norway
martingi@ifi.uio.no

Dmitriy Zheleznyakov
University of Oxford
Wolfson Building
Parks Road
Oxford, OX1 3QD UK
dmitriy.zheleznyakov@cs.ox.ac.uk

Ernesto Jimenez-Ruiz
University of Oxford
Wolfson Building
Parks Road
Oxford, OX1 3QD UK
ernesto@cs.ox.ac.uk

Ian Horrocks
University of Oxford
Wolfson Building
Parks Road
Oxford, OX1 3QD UK
ian.horrocks@cs.ox.ac.uk

NordiCHI'14, Oct 26-30 2014, Helsinki, Finland
ACM 978-1-4503-2542-4/14/10.
http://dx.doi.org/10.1145/2639189.2670270

Abstract
We demonstrate an ontology-based visual query system, namely OptiqueVQS, for end users without any technical background to formulate rather complex information needs into formal queries over databases. It is built on multiple and coordinated representation and interaction paradigms and a flexible widget-based architecture.

Author Keywords
Visual query formulation, ontologies, end-user data access

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces; H.5.4 [Information Interfaces and Presentation]: Hypertext/Hypermedia; D.1.7 [Programming Techniques]: Visual Programming

General Terms
Design, Human Factors

Introduction
Domain experts in organisations have rather complex *information needs* that go beyond the limits of well-known search approaches, such as keyword search. Usually, an army of *IT experts* mediates between domain experts and databases in an inherently time-consuming fashion, since *end users* often lack necessary technical *skills* and

knowledge and have low tolerance on formal textual query languages (e.g., SQL). Hence, engaging end users directly with data could free up substantial expert time that could be redeployed so as to contribute to *value creation* [6].

Visual query formulation (cf. [3]), as an *end-user development* practice (cf. [7]), is promising to remediate end-user data access problem. It is built on the *direct manipulation* idea [9], in which end users *recognise* and *interact* with the visual representations of domain elements, rather than *recalling* domain and syntax elements and *programatically* combining them. *Ontologies* are suggested to be more natural than *logical models* (e.g., database schemas) for end users (cf. [10]), since ontologies are problem domain artefacts, while models are solution domain artefacts (cf. [8]). Ontologies also help us to seamlessly *federate* distributed data sources and to extract implicit information from data with *reasoning*.

In this demo, we present an ontology-based visual query system, *OptiqueVQS* [12], for end-user database querying.

Related Work

One could categorise well-known approaches for querying structured data into *formal textual languages*, *keyword search*, *natural language interfaces*, *visual query languages*, and *visual query systems*. Formal textual languages are inaccessible to end users, since they demand a sound technical background. Keyword search (e.g., [1]) and natural language interfaces (e.g., [5]) remain insufficient for querying databases, due to their low *accuracy* and *completeness*. Visual query languages (e.g., [11]) rely on visual formalisms and are comparable to formal textual languages, while visual query systems are built on a system of interactions and have potential to offer a good balance between *expressiveness* and *usability*.

Ontology-based visual query formulation tools are either exclusively meant for visual query formulation (e.g., [4, 16]) or for semantic data browsing on the Web, which are inherently meant for less sophisticated information needs (e.g., [2]). The latter are usually built on *faceted search* (cf. [15]) and/or *query by navigation* (cf. [14]) and are well embraced by end users.

Existing tools suffer from one or more of the followings: *(i)* singular *representation* and *interaction paradigms*, *(ii)* inadequate support for *view* and/or *overview*, *(iii)* poor balance between *formulation* and *exploration*, and *(iv) non-modular* architectures.

OptiqueVQS

OptiqueVQS allows users to precisely describe complex information needs that demand joining and constraining information from multiple objects. In Figure 1, we ask for all the "Fields", operated by a "Company" named "Statoil", and the completion dates of its "Wellbores".

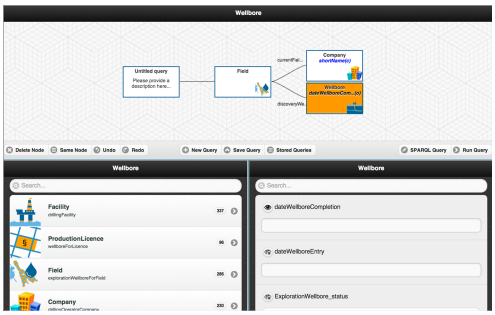


Figure 1: OptiqueVQS with an example query.

OptiqueVQS employs *OWL 2*[1] and *SPARQL*[2], as ontology and query languages respectively. It is designed as a *widget-based user-interface mashup* (i.e., UI mashups) (cf. [13]), which aggregates a set of applications in a common graphical space, in the form of *widgets*, and orchestrates them for common goals. This choice ensures modularity and, in turn, *flexibility* and *extensibility*.

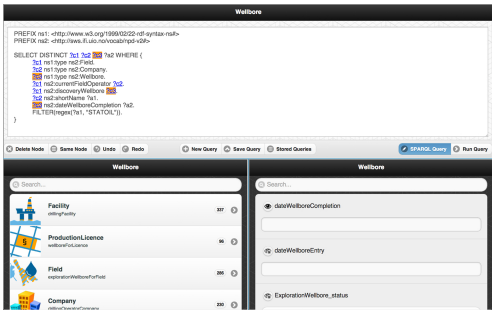


Figure 2: OptiqueVQS in textual mode.

OptiqueVQS (see a demo[3]) currently has three widgets, each employing a different representation and interaction paradigm, cf. Figure 1. A *multi-paradigm* approach offers suitability to a broad range of tasks and users. The first (*W1* – see the bottom-left part) allows users to join concepts through navigating relationships in between. The second (*W2* – see the bottom-right part) presents the attributes of a selected concept for selection and projection. The third (*W3* – see the top part) provides an overview of the constructed query and affordances for manipulation. Events, generated by each widget as a user interacts, are harvested to orchestrate the widgets.

Users can formulate *linear* and *tree-shaped conjunctive* queries, delete nodes, access query catalogue, undo/redo actions, and switch to SPARQL mode and interact with the system in the textual form – see Figure 2.

Holistically, W3 is meant to provide an overview of the active query task, while W1 and W2 are meant to keep the focus (i.e., view) on the active concept. Furthermore, each widget employs the human readable labels of ontology elements rather than their identifiers.

Conclusion and Future Work

Our initial user studies suggest high usability and the future work includes higher expressivity and visual scalability against large ontologies. OptiqueVQS is a part of a large project tackling a range of related questions, such as automated elicitation of ontologies (cf. [6]).

Acknowledgements

Funded by EC FP7 project "Optique" – Grant no. 318338.

References

[1] Bobed, C., Esteban, G., and Mena, E. Enabling keyword search on Linked Data repositories: An ontology-based approach. *International Journal of Knowledge-Based and Intelligent Engineering Systems 17*, 1 (2013), 67–77.

[2] Brunetti, J. M., Garcia, R., and Auer, S. From overview to facets and pivoting for interactive exploration of semantic web data. *International Journal on Semantic Web and Information Systems 9*, 1 (2013), 1–20.

[3] Catarci, T., Costabile, M. F., Levialdi, S., and Batini, C. Visual query systems for databases: A survey.

[1] http://www.w3.org/TR/owl2-overview/
[2] http://www.w3.org/TR/rdf-sparql-query/
[3] http://youtu.be/ks5tcPZVHp0

Journal of Visual Languages and Computing 8, 2 (1997), 215–260.

[4] Catarci, T., Dongilli, P., Di Mascio, T., Franconi, E., Santucci, G., and Tessaris, S. An ontology based visual tool for query formulation support. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI 2004)*, vol. 110 of *Frontiers in Artificial Intelligence and Applications*, IOS Press (2004), 308–312.

[5] Damljanovic, D., Agatonovic, M., Cunningham, H., and Bontcheva, K. Improving habitability of natural language interfaces for querying ontologies with feedback and clarification dialogues. *Web Semantics: Science, Services and Agents on the World Wide Web 19* (2013), 1–21.

[6] Giese, M., Calvanese, D., Horrocks, I., Ioannidis, Y., Klappi, H., Koubarakis, M., Lenzerini, M., Moller, R., Ozcep, O., Rodriguez Muro, M., Rosati, R., Schlatte, R., Soylu, A., and Waaler, A. Scalable End-user Access to Big Data. In *Big Data Computing*, A. Rajendra, Ed. Chapman and Hall/CRC, 2013.

[7] Lieberman, H., Paternó, F., Klann, M., and Wulf, V. End-User Development: An Emerging Paradigm. In *End-User Development*, H. Lieberman, F. Paternó, and V. Wulf, Eds., vol. 9 of *Human-Computer Interaction Series*. Springer, Netherlands, 2006, 1–8.

[8] Ruiz, F., and Hilera, J. R. Using Ontologies in Software Engineering and Technology. In *Ontologies for Software Engineering and Software Technology*, C. Calero, F. Ruiz, and M. Piattini, Eds. Springer-Verlag, 2006, 49–102.

[9] Shneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. *Computer 16*, 8 (1983), 57–69.

[10] Siau, K. L., Chan, H. C., and Wei, K. K. Effects of query complexity and learning on novice user query performance with conceptual and logical database interfaces. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans 34*, 2 (2004), 276–281.

[11] Smart, P. R., Russell, A., Braines, D., Kalfoglou, Y., Bao, J., and Shadbolt, N. A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer. In *Proceedings of the 16th International Conference on Knowledge Engineering: Practice and Patterns (EKAW 2008)*, vol. 5268 of *LNCS*, Springer (2008), 275–291.

[12] Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., and Horrocks, I. OptiqueVQS – Towards an Ontology-based Visual Query System for Big Data. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems (MEDES 2013)*, ACM (2013), 119–126.

[13] Soylu, A., Moedritscher, F., Wild, F., De Causmaecker, P., and Desmet, P. Mashups by orchestration and widget-based personal environments: Key challenges, solution strategies, and an application. *Program: Electronic Library and Information Systems 46*, 4 (2012), 383–428.

[14] Ter Hofstede, A. H. M., Proper, H. A., and Van Der Weide, T. P. Query formulation as an information retrieval problem. *Computer Journal 39*, 4 (1996), 255–274.

[15] Tunkelang, D., and Marchionini, G. *Faceted Search*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan and Claypool Publishers, 2009.

[16] Zviedris, M., and Barzdins, G. ViziQuer: a tool to explore and query SPARQL endpoints. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011)*, vol. 6644 of *LNCS*, Springer (2011), 441–445.

Appendix B

OptiqueVQS:
Extending OptiqueVQS with Ranking

This appendix reports the paper:

– Ahmet Soylu, Martin Giese, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, and Ian Horrocks. **Towards Exploiting Query History for Adaptive Ontology-based Visual Query Formulation**.

MTSR 2014.

Towards Exploiting Query History for Adaptive Ontology-based Visual Query Formulation

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

[1] Department of Informatics, University of Oslo, Norway
{ahmets, martingi}@ifi.uio.no
[2] Department of Computer Science, University of Oxford, United Kingdom
{name.surname}@cs.ox.ac.uk

Abstract. Grounded on real industrial use cases, we recently proposed an ontology-based visual query system for SPARQL, named OptiqueVQS. Ontology-based visual query systems employ ontologies and visual representations to depict the domain of interest and queries, and are promising to enable end users without any technical background to access data on their own. However, even with considerably small ontologies, the number of ontology elements to choose from increases drastically, and hence hinders usability. Therefore, in this paper, we propose a method using the log of past queries for ranking and suggesting query extensions as a user types a query, and identify emerging issues to be addressed.

Keywords: Visual Query Formulation, Ontology-based Data Access, SPARQL, Ranking, Recommendation.

1 Introduction

In data-intensive organisations, *domain experts* usually meet their *information needs* either by operating a set of predefined *queries* embedded into applications or by involving *IT experts* to translate their information needs into queries. This is because domain experts often lack necessary *technical knowledge* and *skills* pertaining to query languages and databases. This man-in-the-middle approach for extracting data introduces a bottleneck in *data access* and consequently delays in *value creation* processes (cf. [1]).

Visual query formulation (cf. [2]) is a longstanding research endeavour and, though oriented towards a wide spectrum of users, a particularly prominent approach to mitigate the data access problem of users without any technical skills (i.e., *end users* – cf. [3]). This is due to fact that visual query formulation tools rely on *recognition*, rather than *recall*, and *direct manipulation* of objects, rather than a *command language syntax*, by using *visual representations* to depict the domain of interest and queries. In this context, we have recently introduced an ontology-based *visual query system* (*VQS*) for end users, named *OptiqueVQS* [4,5]. It is built on a scalable data access platform for *Big Data* developed within an EU

project called *Optique*[1] [1]. *Ontologies* provide reasoning support and a domain representation closer to end users' understanding, compared to earlier approaches built on low-level domain models (e.g., relational schemas) (cf. [6,2]). Besides, Optique employs an *ontology-based data access* (*OBDA*) (cf. [7,8]) technology that extends the platform's data access capabilities to traditional *relational data sources*, which store a significant amount of the world's enterprise data today.

One of the main problems that OptiqueVQS and typically any other VQS face is *scalability* against large ontologies (cf. [9]). A VQS has to provide its users with the elements of ontology (e.g., *concepts* and *properties*) continuously, so that users can select relevant ontology elements and iteratively construct their queries. However, even with considerably small ontologies, the number of concepts and properties to choose from increases drastically due to the propagative effect of ontological reasoning (cf. [10]). In turn, the high number of ontology elements overloads the user interface and hinders *usability*.

We approach the aforementioned problem with *adaptivity* (cf. [11]) by exploiting a *query history* to *rank* and *suggest* ontology elements with respect to an incomplete query that a user has constructed so far (i.e., *context-aware*). The approach is specifically devised for *SPARQL* [12], takes semantics into account with reasoning support, and uses SPARQL, as a programming language, for the implementation. In the rest of paper, we first describe OptiqueVQS, present our ranking proposal, and then discuss the related work. Finally, we provide a discussion on the proposal and emerging issues and conclude the paper.

2 OptiqueVQS

OptiqueVQS is built on multiple and coordinated *representation* and *interaction paradigms* (cf. [2]) and enables end users to formulate comparatively complex queries. OptiqueVQS has a *widget-based* architecture, which underpins its multi-paradigm approach and provides extensibility and flexibility. In the followings, we describe the interface and the formal aspects of SPARQL generation.

2.1 Interface

OptiqueVQS currently has three widgets, see Fig. 1: *W1* (see the top part of Fig. 1) employs a *diagram-based* representation paradigm, gives an overview of the constructed query, and allows further manipulation of it; *W2* (see the bottom-left part of Fig. 1) employs a *menu-based* representation paradigm along with *query by navigation* interaction style (cf. [13]) to let users join concepts via relationships connecting them; *W3* (see the bottom-right part of Fig. 1) is *form-based* and presents the attributes of a selected concept for *selection* and *projection* operations. W3 also has a *faceted search* flavour (cf. [14]), as it uses several natural interaction mechanisms, such as range sliders.

Query construction process in OptiqueVQS works as follows [4,5] – a demo is available[2]. The user begins with selecting a starting concept in W2, i.e., a

[1] `http://www.optique-project.eu/`
[2] `http://youtu.be/ks5tcPZVHp0`

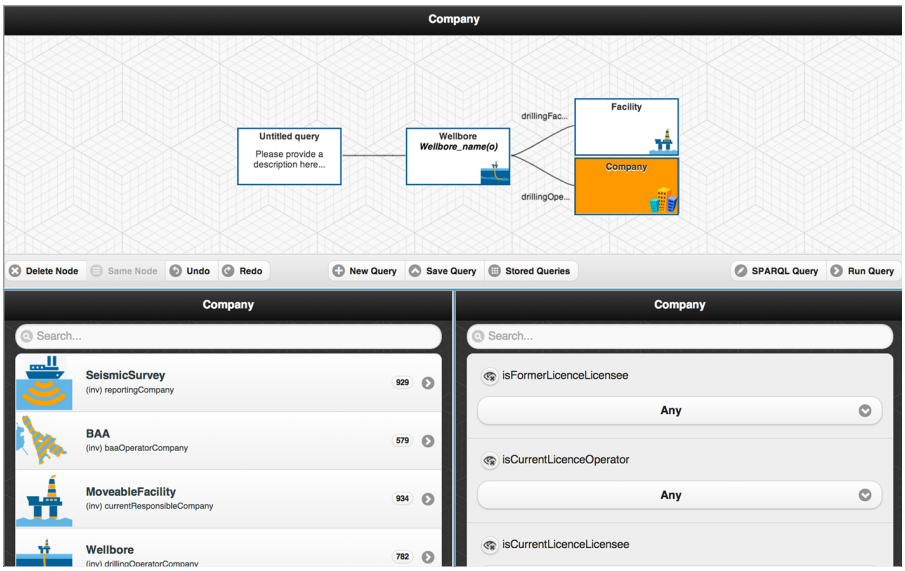


Fig. 1. OptiqueVQS – an example query is depicted.

kernel concept, the selected concept appears in W1 as a typed *variable-node*, and becomes *active* (aka *focus*, *pivot* etc.). Then, the user can extend the query either by selecting one of the offered concept-property pairs in W2, i.e., concepts reachable from the pivot via some *object property*, or by setting constraints on *data type properties* or selecting output variables in W3, i.e., by restricting the data properties of the objects belonging to the pivot. W3 also handles *subclass* selection, as it presents direct subclasses of the pivot concept as a multi-select form element. The user can change the pivot by clicking on any variable-node in W1 and continue extending the query by selecting a concept-property pair in W2. OptiqueVQS automatically extends the list of concept-property pairs and data properties in W2 and W3 via the *HermiT reasoner* [15] (e.g., a concept inherits all the properties of its parent concept). The user can delete nodes, access query catalogue, save/load queries, undo/redo actions, or continue query construction in the textual SPARQL mode.

2.2 Formal description

OptiqueVQS currently supports *linear* and *tree-shaped conjunctive* queries. The OBDA framework behind OptiqueVQS supports *OWL 2 QL* [16] and a conjunctive fragment of *SPARQL 1.1* [12]. OWL 2 QL is a *profile* of *OWL 2* and in this profile query answering can be implemented by rewriting queries into a standard relational query language [17].

The way the ontology controls the behaviour of OptiqueVQS should be seen from two perspectives: from a *knowledge representation* (*KR*) perspective, Optique exploits the graph-based organisation of ontological elements and data for representing the domain and query structures (cf. query by navigation); from a *logic* perspective, it uses ontological *axioms* to constrain the behaviour of the interface and to extend the available knowledge. On a purely structural level, OptiqueVQS could be controlled directly by a graph G that captures the concepts and the properties of an ontology O. An OWL ontology can be viewed as a *labeled directed* RDF graph $G = (N, E)$, where N is a finite set of labeled *nodes* and E is a finite set of labeled *edges* (cf. [17]). We consider pairwise disjoint alphabets U, a set of *URIs*, L, a set of *terminal literals*, and B, a set of *blank nodes*. An edge is a *triple* written in the form of $\langle s, p, o\rangle \in (U \cup B) \times U \times (U \cup L \cup B)$. The nodes of the graph mainly represent concepts and edges represent properties. A SPARQL query is formally represented by a tuple defined as $Q = (A, V, D, P, M, R)$. A is the set of *prefix declarations*, V is the *output form*, D is the *RDF graph* being queried, P is a *graph pattern*, M are *query modifiers*, which allow to modify the results by applying *projection*, *order*, *limit*, and *offset* options. SPARQL is based on matching graph patterns against RDF graphs. P is composed of a set of *triple patterns* and describes a *subgraph* of D. The main difference between a triple pattern and RDF triple comes from the fact that the former may have each of *subject*, *predicate* and *object* as a variable. However, once we substitute variables in triple patterns with constants or blank nodes, we reach an RDF graph $P'(N', E')$ that could be considered as a subgraph of the actual RDF data graph.

Every query generated by OptiqueVQS has a graph pattern represented by a set of triple patterns, where each triple pattern is a tuple $t \in Var \times U \times (U \cup Var \cup L)$ and Var is an infinite set of variables. The state of an edited query is composed of a partial graph pattern and a *cursor position* (cf. pivot). The cursor position is either blank (i.e., empty query) or points to a variable in the query. If the query is empty, the selection of a concept v from W2 results in a new tuple $\langle x, \texttt{rdf:type}, v\rangle \in Var \times U \times U$ in P, where x is a fresh variable. If the cursor points to a variable x, of type v, then each selection of a object property o with target class w from W1 (corresponding to an edge $\langle v, o, w\rangle \in G$) adds the following two triple patterns to P: $\langle x, o, y\rangle \in Var \times U \times Var$ and $\langle y, \texttt{rdf:type}, w\rangle \in Var \times U \times U$, where y is a fresh variable. Every selection and projection operation realised over a data property d in W3, while cursor is on a variable x, adds a new tuple $\langle x, d, y\rangle \in Var \times U \times (Var \cup L)$ to P. Finally, the selection of a subclass v for a typed variable x in W3 results in a new triple in P: $\langle x, \texttt{rdf:type}, v\rangle \in Var \times U \times U$.

3 Adaptive Query Formulation

Currently, the widgets W2 and W3 (see Fig. 1) present all the available concept-object property pairs and data properties to users respectively. However, the lists grow quickly due to *ontology size*, *number of relationships between concepts*,

subproperties, *inverse properties*, *inheritance of restrictions* etc. As the lists grow, the time required for a user to find elements of interest increases; therefore ranking ontology elements with respect to previously executed queries and suggesting highly ranked elements first as possible query continuations have potential to increase the efficiency of the users. The nature of OptiqueVQS requires suggestions to be done for the pivot (i.e., cursor point) rather than for any part of a query.

In what follows, we first present a running example and then describe our ranking method for context-aware suggestions. The running example is built on one of the industrial Optique use cases, namely the *Statoil* use case. Statoil[3] is a large international energy company focused on upstream oil and gas operations. The company reports that value creation processes could be improved considerably, if domain experts are to be able to access data on their own.

3.1 Running Example

The exploration department of Statoil has to find new hydrocarbon reserves in a cost effective way and ultimately the only way to prove the presence of a reserve is to drill an exploration well, which may consist of one or several well paths, i.e., wellbores. But since drilling is very expensive, it is important to maximise the chances of success. To do this, all available data from previous and ongoing exploration and production projects to extrapolate a model of the geology of a field, which then allows to anticipate the presence of hydrocarbon reserves.

A partial simplified ontology for Statoil exploration department is depicted in Fig. 2. The ontology currently contains 344 concepts, 148 object properties, 237 data properties, and 8190 axioms and it is yet to grow. In Fig. 3, an example query log with three queries is assumed for the sake of brevity. The first query, *Q1*, is the one that is depicted in Fig. 1 and asks for the names of wellbores with a drilling facility and a drilling company. The second query, *Q2*, asks for the content of all shallow wellbores that belongs to a well and has a drilling company of type operator. The final query, *Q3*, asks for the content of all exploration wellbores that has a fixed drilling facility and a drilling company.

In Fig. 3, *PQ* refers to an example partial query. The query in its incomplete form asks for all exploration wellbores with a drilling company; the cursor point is the variable of type exploration wellbore. At this point of query formulation session, the widgets W2 and W3 need to suggest the most relevant continuations, by comparing the partial query with the queries in the query log.

3.2 Ranking Method

A query log QL is basically a set of SPARQL queries: $QL = \{Q_1, Q_2, ..., Q_n\}$. We define a function p that takes a query Q as an input and returns its graph pattern P. We define S as a set suggestions $\{T_1, T_2, ..., T_m\}$. Each suggestion in S is a triple set T_i, which either contains two triples for W2 in the form of $\{\langle x, o, y\rangle \in Var \times U \times Var,\ \langle y, \texttt{rdf:type}, w\rangle \in Var \times U \times U\}$ or one triple for

[3] `http://www.statoil.com/`

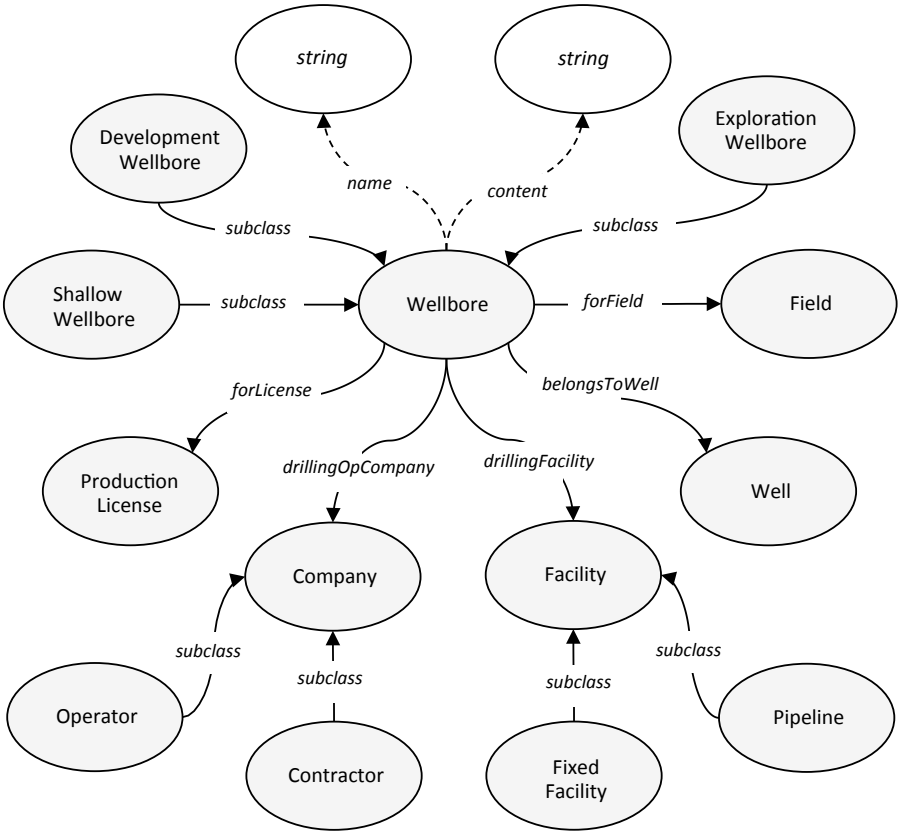


Fig. 2. A partial simplified ontology for the Statoil use case.

W3 in the form of $\{\langle x, d, y\rangle \in Var \times U \times (Var \cup L)\}$, where x corresponds to the cursor variable in a partial user query Q_a. Note that subclass suggestion is not included in the ranking, since it is always suggested by default.

The ranking score, at this point, basically corresponds to the *conditional probability* for each suggestion T_i in S, given a partial query Q_a and a query log QL, that is $Pr(T_i \mid p(Q_a))$. Conditional probability and probability functions are defined in the followings.

Within a query log QL, the probability of a graph pattern P is defined as the fraction of graph patterns in QL that are *supergraphs* [18] of P, as shown in Eq. 1.

$$Pr(P) = \frac{|\{Q_i \in QL | P \subseteq p(Q_i)\}|}{|QL|} \tag{1}$$

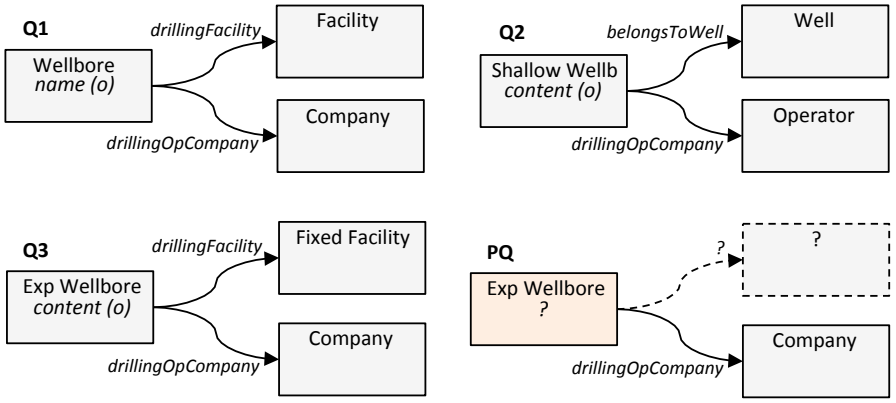


Fig. 3. A query log with three queries and an example partial user query.

The conditional probability of a triple set T given a graph pattern P is defined as the quotient of the probability of the union of T and P, and the probability of P as shown in Eq. 2.

$$Pr(T \mid P) = \frac{Pr(T \cap P)}{Pr(P)} \tag{2}$$

Now two important questions come into play. First, how do we find supergraphs in the query log, given a partial user query? Second, how do we extract possible extensions, i.e., suggestions, for the partial query from found supergraphs? As far as the first problem is concerned, it boils down to a *graph matching* problem. We consider a graph pattern P_1 to be subgraph of another graph pattern P_2, if all the triple patterns of P_1 are covered by P_2, independent of variable names, ordering of triple patterns, and the values of constraints. Dividino and Groner [19] review different approaches for checking graph similarity, where our interest falls into *content-based* approaches. We propose a method that relies on SPARQL itself and provides us with an exhaustive solution, as it allows us to exploit semantic knowledge while matching queries.

The method starts with the instantiation of graph patterns of queries in the query log by replacing variable names and constraints on data type properties with blank nodes; blank node names are marked with a query identifier for preventing any overlap and identification purposes. Then, the resulted RDF graphs are stored in a common dedicated triple store; the instantiation of the query log depicted in Fig. 3 is given in Fig. 4. By applying the partial query over this triple store, one can retrieve all the queries that are the supergraphs of the partial query.

As far as the second question is concerned, i.e., finding possible extensions, the partial query is extended with a triple pattern from the cursor point to retrieve all extensions occurred in the matching supergraphs. The output of partial query

Query log: SPARQL form

```
Q1 SELECT DISTINCT ?c1 ?a1 ?c2 ?c3
WHERE {
  ?c1 ns1:type ns2:Wellbore.
  ?c2 ns1:type ns2:Facility.
  ?c3 ns1:type ns2:Company.
  ?c1 ns2:drillingFacility ?c2.
  ?c1 ns2:drillingOpCompany ?c3.
  ?c1 ns2:name ?a1.
}

Q2 SELECT DISTINCT ?c1 ?a1 ?c2 ?c3
WHERE {
  ?c1 ns1:type ns2:ShallowWellbore.
  ?c2 ns1:type ns2:Operator.
  ?c3 ns1:type ns2:Well.
  ?c1 ns2:drillingOpCompany ?c2.
  ?c1 ns2:belongsToWell ?c3.
  ?c1 ns2:wellboreContent ?a2.
}

Q3 SELECT DISTINCT ?c1 ?a1 ?c2 ?c3
WHERE {
  ?c1 ns1:type ns2:ExpWellbore.
  ?c2 ns1:type ns2:Company.
  ?c3 ns1:type ns2:FixedFacility.
  ?c1 ns2:drillingOpCompany ?c2.
  ?c1 ns2:drillingFacility ?c3.
  ?c1 ns2:wellboreContent ?a1.
}
```

Query log: triple form

```
_:q1c1 ns1:type ns2:Wellbore.
_:q1c2 ns1:type ns2:Facility.
_:q1c3 ns1:type ns2:Company.
_:q1c1 ns2:drillingFacility _:q1c2.
_:q1c1 ns2:drillingOpCompany _:q1c3.
_:q1c1 ns2:name _:q1a1.

_:q2c1 ns1:type ns2:ShallowWellbore.
_:q2c2 ns1:type ns2:Operator.
_:q2c3 ns1:type ns2:Well.
_:q2c1 ns2:drillingOpCompany _:q2c2.
_:q2c1 ns2:belongsToWell _:q2c3 .
_:q2c1 ns2:wellboreContent _:q2a1.

_:q3c1 ns1:type ns2:ExpWellbore.
_:q3c2 ns1:type ns2:Company.
_:q3c3 ns1:type ns2:FixedFacility.
_:q3c1 ns2:drillingOpCompany _:q3c2.
_:q3c1 ns2:drillingFacility _:q3c3.
_:q3c1 ns2:wellboreContent _:q3a1.
```

Fig. 4. The instantiation of query graph patterns.

is modified to retrieve the identifiers of matching queries, properties, and the types of variables for the returned extension. An example is given in Fig. 5 for the partial query depicted in Fig. 3 and the triple store depicted in Fig. 4. The rest of the method involves calculation of conditional probabilities for the suggestions, as exemplified in Fig. 5.

If one inspects the results in Fig. 5 closely, she will realise that reasoning is involved. This is because in the query log, only Q3 is an exact match for the partial query. However, thanks to reasoning support, Q1 is also matched, since exploration wellbore is a subclass of wellbore. Likewise, this guarantees a match for any query that has a *semantic similarity* [20] to the partial query, involving subclasses, subproperties, inverses etc. Later in the paper, this is to be discussed further, as there is a *semantic distance* involved between the partial query and the matched query. Yet, it is possible to query the triple store without any reasoning, if one wants to eliminate such matches, hence avoiding any semantic distance.

Partial user query

```
SELECT DISTINCT ?c1 ?c2
WHERE {
  ?c1 ns1:type ns2:ExpWellbore.
  ?c2 ns1:type ns2:Company.
  ?c1 ns2:drillingOpCompany ?c2.
}
```

Modified partial user query

```
SELECT DISTINCT ?c3 ?prop ?type
WHERE {
  ?c1 ns1:type ns2:ExpWellbore.
  ?c2 ns1:type ns2:Company.
  ?c1 ns2:drillingOpCompany ?c2.
  ?c1 ?prop ?c3.
  OPTIONAL { ?c3 rdf:type ?type }
}
```

Matches

?c3		?prop	?type	Pr(T|P)	Widget
_:q1c2	T_1	ns2:drillingFacility	ns2:Facility	0.16	W2
_:q1c3	T_2	ns2:drillingOpCompany	ns2:Company	0.33	W2
_:q1a1	T_3	ns2:name		0.16	W3
_:q3c2	T_2	ns2:drillingOpCompany	ns2:Company		
_:q3c3	T_4	ns2:drillingFacility	ns2:FixedFacility	0.16	W2
_:q3a1	T_5	ns2:wellboreContent		0.16	W3

Fig. 5. Modified partial user query and possible query extensions.

The final stage involves ordering and dividing S into two sets, S_1 for W2 and S_2 for W3, with respect to ranking score and type of each suggestion (i.e., concept-relationship pair vs. data type property). Then, suggestions in each set are paginated into $\frac{|S_i|}{j}$ pages, where i is the set identifier and j is the *window size* for a page (i.e., the required number of suggestions for a page).

4 Related Work

There are a number of visual query formulation tools available in the literature (e.g., [2,21,22,23]); however, to the best of authors knowledge none of them supports adaptive visual query formulation. Existing approaches for adaptive query formulation are largely developed for *context-sensitive* textual query formulation.

Khoussainova et al. [24] provide a system, named *SnipSuggest*, for context-aware composition of textual SQL queries with respect to a given query log. The authors translate each SQL query in the query log into a set of features (e.g., a table name appearing in the *FROM clause*). Similarly, the partial query of the user is also translated into a set of features. Possible features for extension are identified by matching the feature sets of the partial query and the feature sets of queries in the query log and are ranked by calculating conditional probabilities. The approach generates suggestions for extending any part of the partial query rather

than a single cursor point. Authors also propose a set of supportive algorithms and techniques for, such as feature set matching (i.e., what if the partial query does not appear in the query log), the selection of suggestions (i.e., accuracy vs. diversity), and query log elimination (i.e., to reduce the size). The elaborate approach provided by SnipSuggest system is relevant to our contribution in many aspects. However, a fundamental difference is in feature comparison; while the features of SnipSuggest system are a set of syntactic elements and the feature comparison is string based, for OptiqueVQS feature sets (i.e., correspond to the triple sets of graph patterns) have a semantic nature and compared semantically. The semantic aspects not only concern how the matching is done, but also the calculation of rankings, which we discuss in the following section.

As far as approaches for SPARQL are concerned, Campinas et al. [25] propose an approach for assisting textual SPARQL query formulation, however in a different context. The approach assumes that an ontology describing the data set is unknown. Therefore, the authors propose a model that summarises the underlying data graph and extracts ontology elements to suggest. The approach extends a given partial user query from the cursor point, similar to our approach, and then evaluates it over the data graph summary to retrieve possible extensions. However, the approach does not realise any ranking of suggestions based on the previously executed queries and does not take semantic similarities between queries into account, possibly due to lack of rich domain knowledge (e.g., lack of subclass, inverse property axioms).

Kramer et al. [26] present a tool, named *SPACE*, to support autocompletion of textual SPARQL queries. For this purpose, it takes a SPARQL query log as an input and then builds an *index structure* for the computation of query suggestions. The index structure has a root node at level 0, representing a set of queries, while each vertex at level 1 represents a SPARQL query. The vertices from level $n-2$ to index level 1 represent graph patterns recursively. Finally the vertices at the highest level $(n-1)$ represent IRIs, blank nodes, literals, variables, and binary operators such as *AND*, *UNION*, and *FILTER*. The suggestion process is done by subgraph matching for the partial user query in the index graph in a bottom up manner. However, the authors describe neither the subgraph matching process nor the details of ranking calculation. Finally, the index structure could grow quickly as it is built on recursive decomposition of graph patterns.

5 Discussion

The fact that there exist SPARQL engines capable of handling large triple sets effectively [27,28] is a positive evidence for the execution performance of our approach, since our proposal relies on SPARQL querying for matching partial user queries against a query log. One should also note that the size of a triple store for a query log is only expected to be in the order of thousands triples, if maintained – e.g., pruned, clustered etc.

As far as the *precision* of suggestions is concerned, approaches that take the partial query into account are reported to be better than *popularity-based*

approaches purely built on the number of occurrences of terms in the query log [24]. Note that, initially, when no kernel concept is selected, our approach behaves like a popularity-based approach, as it extends an empty query. Below, we discuss a set of issues that need to be addressed:

Semantic distance: In Fig. 3, the match between the first query and partial query is due to their semantic similarity and is not exact (e.g., exploration wellbore is a subclass of wellbore); and in Fig. 5, the drillingFacility - Facility and drillingFacility - FixedFacility suggestions are semantically similar. Therefore one could incorporate the semantic distance involved as a cofactor into the ranking function, so that semantically distant queries contribute less to the ranking. Huang et al. [20] suggest a similarity measure, which can readily incorporated to our proposal. It uses the depth of compared concepts and properties and their least common ancestors from the root of hierarchy to compute similarity between concepts and properties and combine them to compute similarity between triple patterns, hence queries.

No match: A problematic situation arises when no match is found for the partial query in the query log (cf. [24]). A possible solution could be pruning the partial query until a match is found. At each step of a pruning process, a leaf node, which is not the cursor point, could be randomly selected and deleted (or with respect to some heuristics), so that partial query graph pattern does not get disconnected and the cursor point is preserved.

Cold start: The proposal cannot draw any suggestions, when the query log contains no or insufficient number of queries. Mostly likely sources to use for addressing this problem are the ontology and data set. A statistical inspection of ontology, e.g., concept centrality with respect to the number of incoming and outgoing relationships, and the data set, e.g., the number of times each concept and property appears in the dataset, could reveal useful information to overcome the cold start problem.

Collective, group, or individual: The ranking and suggestions could be applied on an individual basis for each user, i.e., only over the portion of query log that belongs to the subject user, on group basis, i.e., only over the portion of query log that belongs to the users of same type, and on a collective basis, i.e., over the whole query log for every user (cf. [11]). The decision possibly should consider whether users are homogeneous or there exist different user groups, each using a part of the ontology heavily – e.g., geologist and chemists. In the former case, a group or even user specific approach is more feasible, as each user group/user focuses on a specific part of the ontology.

6 Conclusion and Future Work

Ontology-based end-user visual query formulation is promising for enhancing value creation processes; yet existing approaches are not scalable against large ontologies. Although there are some attempts for assisted textual query formulation in the literature; they are either not elaborate enough to be readily used in our case or do not take previously executed queries into account. In this paper, we proposed

a method for ranking and suggesting SPARQL query extensions, which relies on the partial user query, the queries in the query history, and their semantic similarity. We also identified notable issues to be addressed in order to reach an elaborate solution.

The future work involves comparative evaluation of the proposed method and its variants (e.g., with/without semantic similarity) in terms of precision. End-user studies are also planned to measure the *perceived usefulness*, i.e., whether in practice users find ranking approach useful or not.

Acknowledgements. This research is funded by the FP7 of the European Commission under Grant Agreement 318338, "Optique".

References

1. Giese, M., Calvanese, D., Horrocks, I., Ioannidis, Y., Klappi, H., Koubarakis, M., Lenzerini, M., Moller, R., Ozcep, O., Rodriguez Muro, M., Rosati, R., Schlatte, R., Soylu, A., Waaler, A.: Scalable End-user Access to Big Data. In Rajendra, A., ed.: Big Data Computing. Chapman and Hall/CRC (2013)
2. Catarci, T., Costabile, M.F., Levialdi, S., Batini, C.: Visual query systems for databases: A survey. Journal of Visual Languages and Computing **8**(2) (1997) 215–260
3. Lieberman, H., Paternó, F., Klann, M., Wulf, V.: End-User Development: An Emerging Paradigm. In Lieberman, H., Paternó, F., Wulf, V., eds.: End-User Development. Volume 9 of Human-Computer Interaction Series. Springer, Netherlands (2006) 1–8
4. Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I.: OptiqueVQS – Towards an Ontology-based Visual Query System for Big Data. In: Proceedings of the International Conference on Management of Emergent Digital EcoSystems (MEDES 2013), ACM (2013) 119–126
5. Soylu, A., Skjæveland, M., Giese, M., Horrocks, I., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D.: A Preliminary Approach on Ontology-based Visual Query Formulation for Big Data. In: Proceedings of the 7th International Conference on Metadata and Semantic Research (MTSR 2013). Volume 390 of CCIS., Springer (2013) 201–212
6. Siau, K.L., Chan, H.C., Wei, K.K.: Effects of query complexity and learning on novice user query performance with conceptual and logical database interfaces. IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans **34**(2) (2004) 276–281
7. Spanos, D.E., Stavrou, P., Mitrou, N.: Bringing relational databases into the Semantic Web: A survey. Semantic Web **3**(2) (2012) 169–209
8. Kogalovsky, M.R.: Ontology-Based Data Access Systems. Programming and Computer Software **38**(4) (2012) 167–182
9. Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., Giannopoulou, E.: Ontology visualization methods - A survey. ACM Computing Surveys **39**(4) (2007) 10:1–10:43
10. Grau, B.C., Giese, M., Horrocks, I., Hubauer, T., Jimenez-Ruiz, E., Kharlamov, E., Schmidt, M., Soylu, A., Zheleznyakov, D.: Towards Query Formulation and Query-Driven Ontology Extensions in OBDA Systems. In: Proceedings of the 10th OWL: Experiences and Directions Workshop (OWLED 2013). Volume 1080 of CEUR Workshop Proceedings., CEUR-WS.org (2013)

11. Brusilovsky, P., Kobsa, A., Nejdl, W., eds.: The Adaptive Web: Methods and Strategies of Web Personalization. Volume 4321 of LNCS. Springer (2007)
12. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C Recommendation, W3C (March 2013)
13. Ter Hofstede, A.H.M., Proper, H.A., Van Der Weide, T.P.: Query formulation as an information retrieval problem. Computer Journal **39**(4) (1996) 255–274
14. Tunkelang, D., Marchionini, G.: Faceted Search. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan and Claypool Publishers (2009)
15. Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research **36**(1) (2009) 165–228
16. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language Profiles. W3C Recommendation, W3C (October 2009)
17. Grau, B.C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: OWL 2: The Next Step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web **6**(4) (2008) 309–322
18. Ray, S.S.: Subgraphs, Paths, and Connected Graphs. In: Graph Theory with Algorithms and its Applications. Springer India (2013)
19. Dividino, R., Groner, G.: Which of the following SPARQL Queries are Similar? Why? In: Proceedings of the 1st International Workshop on Linked Data for Information Extraction (LD4IE 2013). Volume 1057 of CEUR Workshop Proceedings., CEUR-WS.org (2013)
20. Huang, H., Liu, C., Zhou, X.: Computing Relaxed Answers on RDF Databases. In: Proceedings of the 9th International Conference on Web Information Systems Engineering (WISE 2008). Volume 5175 of LNCS., Springer (2008) 163–175
21. Catarci, T., Dongilli, P., Di Mascio, T., Franconi, E., Santucci, G., Tessaris, S.: An ontology based visual tool for query formulation support. In: Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI 2004). Volume 110 of Frontiers in Artificial Intelligence and Applications., IOS Press (2004) 308–312
22. Kapetanios, E., Baer, D., Groenewoud, P.: Simplifying syntactic and semantic parsing of NL-based queries in advanced application domains. Data & Knowledge Engineering **55**(1) (2005) 38–58
23. Barzdins, G., Liepins, E., Veilande, M., Zviedris, M.: Ontology Enabled Graphical Database Query Tool for End-Users. In: Proceedings of the 8th International Baltic Conference on Databases and Information Systems (DB&IS 2008). Volume 187 of Frontiers in Artificial Intelligence and Applications., IOS Press (2009) 105–116
24. Khoussainova, N., Kwon, Y., Balazinska, M., Suciu, D.: SnipSuggest: Context-aware Autocompletion for SQL. Proceedings of the VLDB Endowment **4**(1) (2010) 22–33
25. Campinas, S., Perry, T.E., Ceccarelli, D., Delbru, R., Tummarello, G.: Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation. In: Proceedings of the 23rd International Workshop on Database and Expert Systems Applications (DEXA 2012), IEEE (2012) 261–266
26. Kramer, K., Dividino, R., Groner, G.: SPACE: SPARQL Index for Efficient Autocompletion. In: Proceedings of the ISWC 2013 Posters & Demonstrations Track (ISWC-PD 2013). Volume 1035 of CEUR Workshop Proceedings., CEUR-WS.org (2013)
27. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^2Bench: A SPARQL Performance Benchmark. In: Proceedings of the IEEE International Conference on Data Engineering (ICDE 2009), IEEE Computer Society (2009) 222–233
28. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. International Journal on Semantic Web and Information Systems **5**(2) (2009) 1–24

Appendix C

OptiqueVQS: Demonstration

This appendix reports the paper:

– Evgeny Kharlamov, Martin Giese, Peter Haase, Ernesto Jiménez-Ruiz, Christoph Pinkel, Martin G. Skjæveland, Ahmet Soylu, Johannes Trame, Dmitriy Zheleznyakov, Carsten Binnig, Eldar Bjørge, Ian Horrocks, Arild Waaler. Ian Horrocks. **Towards Ontology Based Data Access for Statoil**.
ICDE 2015.

Towards Ontology Based Data Access for Statoil

E. Kharlamov[1] M. Giese[2] P. Haase[3] E. Jiménez-Ruiz[1] C. Pinkel[3] M. G. Skjæveland[2]
A. Soylu[2] J. Trame[3] D. Zheleznyakov[1] C. Binnig[4] E. Bjørge[5] I. Horrocks[1] A. Waaler[2]
[1] University of Oxford; [2] University of Oslo; [3] fluid Operations AG; [4] DHBW Mannheim; [5] Statoil ASA

***Abstract*—Ontology Based Data Access (OBDA) is a prominent approach to provide end users with high-level access to data via an ontology that is 'connected' to the data via mappings. State-of-the-art OBDA systems, however, suffer from limitations restricting their applicability in industry. Existing solutions focus on separate critical components of OBDA systems, while, to the best of our knowledge, there is no end-to-end OBDA solution allowing to deploy an OBDA system in an enterprise from scratch, effectively maintain and use it. In particular, development of necessary prerequisites to deploy an OBDA system, i.e., ontologies and mappings, as well as end user oriented query interfaces, are poorly addressed. The Optique platform provides an integrated end-to-end OBDA solution that addresses a number of practical challenges including the ones above. During the demonstration the attendees can try the platform with preconfigured scenarios from the petroleum industry and music domain, and try its end-to-end functionality: from deployment to query answering.**

I. Introduction

The growth of available information sources in enterprises requires new efficient methods for data access by domain experts whose ability to understand and analyse data is at the core of making business decisions. Currently in Statoil[1] and other data intensive companies there is a domination of centralised approaches, where an IT-expert translates information requests of domain experts into Extract-Transform-Load (ETL) processes to first integrate the data and then to apply predefined analytical reporting tools [8]. At the same time, in many scenarios an interactive data exploration, where domain experts want to access and analyse available data sources *directly*, without involving IT-experts, is of a high importance and centralised approaches are too heavy-weight and inflexible to do the job [7, 14]. In the Optique project [14] we aim at developing a direct data access solution that on the one hand would provide such access to Statoil's Exploration and Production Data Store (EPDS) and on the other hand would be generic enough to be applied in similar industries.

Challenges in providing domain experts with the direct data access include *(i)* the *complexity* of schemata that could contain hundreds and thousands of tables, e.g., EPDS has 3,000 tables with about 37,000 columns, and *(ii)* the *conceptual mismatch* between the language and structures that the domain experts use to describe the data, and the way the data is described and structured by database schema languages. Indeed, schemata are often integrated from autonomously evolving systems (yielding schema complexity), that have been adapted over years to the purpose of the applications they underly—EPDS was created 15 years ago–and not to the purposes of being intuitive for domain experts (yielding the conceptual mismatch). Further important practical challenges in providing direct data access are *(iii) formal query languages*, e.g., to access EPDS domain experts should be proficient in SQL, and *(iv) data incompleteness*. Regarding the latter challenge, considerable amount of Statoil exploration data are results of measurements taken during exploration activities by different teams and they are often incomplete and fragmented. Thus, a SQL encoding of a given information need over such data is inevitably complex, it may involve multiple data sources and tables referring to conceptually the same data stored in different places, e.g., queries over EPDS often contain thousands of words and have 50–200 joins.

Ontology Based Data Access (OBDA) [17] is a prominent, so-called *virtual*, approach to direct data access for end users, i.e., it provides an integration and access layer on top of databases while the data stays in the original stores. In OBDA users and data are mediated by an *ontology*, a semantically rich conceptual model,[2] and users formulate their information needs as queries over the ontology. These queries are enriched via *logical reasoning* over the ontology, translated into SQL over the underlying databases with the help of *mappings* (declarative specifications describing the relationship between the ontological vocabulary, e.g., 'wellbore' or 'oilfield', and the elements of the database schema) and finally executed over these databases automatically, without IT-experts intervention.

OBDA naturally addresses three out of four challenges above. Indeed, in contrast to a DB schema, an ontology describes a domain of interest rather than a structure of a DB and do it on a high level of abstraction in terms that are clear for domain experts, thus, avoiding the complexity of DB schemata required of Challenge (i), and the conceptual mismatch of Challenge (ii). Challenge (iv) can be addressed with the help of both mappings and ontologies. Indeed, an ontology is written in a logic based formal language e.g., W3C Web Ontology Language (OWL 2) as a set of OWL 2 axioms, and admits logical reasoning that allows to enrich ontological queries and address the incompleteness;[3] while mappings can relate one ontological term, e.g., 'wellbore', to many different parts of a DB.

State-of-the art OBDA systems, despite success stories in various scenarios, e.g. [5], have a number of important limitations. To the best of our knowledge there is no *end-to-end* OBDA solution providing both IT-experts and end users with the necessary tools to deploy an OBDA system in an enterprise from scratch, effectively maintain and use it. The existing solutions, e.g., [4, 5, 18, 19] focus on separate critical components of OBDA systems, and do not offer sufficient support for end users oriented query formulation discussed in Challenge (iii), as well as deployment and maintenance of OBDA systems, which is in practice expensive.

[1]The Norwegian oil and gas company, http://www.statoil.com/.

[2]Ontologies are common in many areas, e.g., medicine, Semantic Web [11].

[3]Many efficient off-the-shelf reasoning tools are available, e.g., [9, 19].

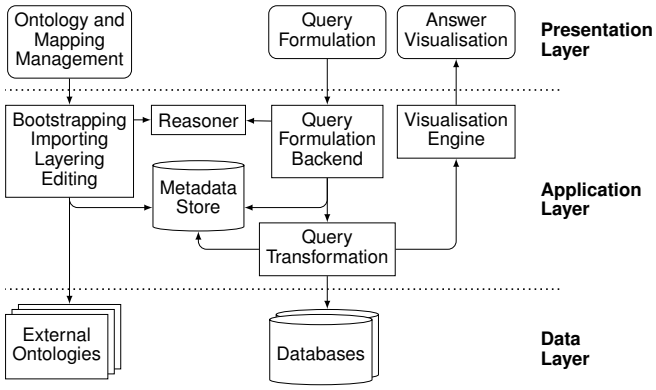


Fig. 1: General architecture of the Optique system

In the Optique project we have been developing an end-to-end OBDA system that satisfies Statoil requirements and addresses a number of practical challenges, including Challenges (i)-(iv) above. Our solution integrates in a unified platform a number of existing and novel components and allows one to deploy, maintain, and use the Optique platform in enterprises. Among the novel components, there is a deployment and maintenance module allowing to extract ontologies and mappings from relational databases in a semi-automatic fashion, integrate preexisting ontologies in an existing OBDA deployment instance, and edit mappings. We also developed a component for query formulation support that relies on novel techniques of projecting ontologies on graphs as well as components to visualise and browse query answers. We evaluated the platform with Statoil over EPDS and with other real world databases with encouraging results [22].

In this demonstration we show the platform's end-to-end functionality with the focus on its novel components. We demonstrate the platform on two preconfigured scenarios and allow the attendees to deploy it over either of the two databases underlying the scenarios, improve the deployment using the mapping editor, query the resulting deployment, see and browse query answers on maps, in tables, etc. Our first demo scenario is inspired by Statoil data:[4] we demonstrate the system on the NPD FactPages database [20], a publicly available data about petroleum activities on the Norwegian Continental Shelf that overlaps with EPDS. Since understanding NPD FactPages requires basic knowledge about the petroleum domain, we also demonstrate the platform over a large open music encyclopedia MusicBrainz [1]. The demo video illustrating the functionalities of our system and the demo system is available in [2].[5]

II. Overview of the Optique Platform

The general three-layer architecture of the Optique platform is illustrated in Figure 1. To deploy the platform over a relational DB, one can use its tools to extract ontologies and mappings from the DB, incorporate external ontologies, edit and author mappings of the resulting OBDA instance. After the system is deployed, the underlying DB can be queried using our visual query formulation tool that allows to compose queries by navigation over the system's ontology. Visually formulated queries are translated into SPARQL and sent to the query transformer for processing: query enrichment using the ontology and further unfolding with the mappings into

[4]Due to privacy we cannot demo our solution on Statoil's corporate data.
[5]A very preliminary version of the Optique platform was presented in [13].

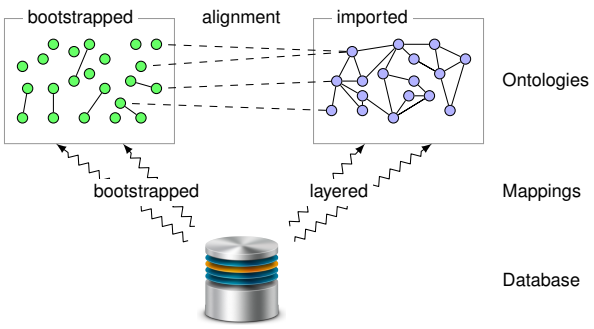


Fig. 2: Semi-automatic deployment approach

SQL queries. We rely on the Ontop [19] query transformer, which is an integral component of the Optique platform. SQL queries are executed over the data sources underlying the system by the DBMSs of the sources. We offer a number of templates and widgets such as tables, timelines, maps, charts, etc., depending on the data modalities, to visualise and browse resulting query answers (see two screenshots of platform's answer visualisation in the bottom of Figure 3). The integration of the Optique platform is based on the Information Workbench [10], a generic and extensible platform for semantic data management, providing the platform with many base components, including interfaces and APIs as well as a triple store for managing ontologies, mappings, query logs, (excerpts of) query answers, DB metadata, etc.

III. Deployment and Maintenance

The Optique platform provides semi-automatic support for deployment, which is schematically depicted in Figure 2. The platform supports different deployment scenarios. For example one can start with *bootstrapping*, i.e., a semi-automatic extraction of an ontology and mappings from the database. Then, one can *import* a pre-existing ontology and 'connect' it to the bootstrapped one via alignment with our LogMap ontology alignment system [12], which we have been developing during the last four years. This scenario can be applied, e.g., when the database schema or some of its fragments have a good correspondence with the domain of interest, or the available pre-existing ontology has a limited coverage of the domain of interest. Another possible scenario is to *layer* a pre-existing ontology directly over the database, i.e., to 'connect' it to the database schema with semi-automatically generated mappings. This scenario can address the case when there are several good ontologies available and they can serve as entry points to data for users with potentially different needs. The Optique platform supports ontologies expressible in the OWL 2 QL profile of OWL 2 ontology language, which was specifically designed for efficient data access. Imported or layered, OWL 2 ontologies that cannot be captured in OWL 2 QL are automatically approximated in OWL 2 QL using the technique of [6]. For mapping maintenance the platform offers a novel *mapping editor*. We now discuss Optique's modules in detail.

Mapping Bootstrapping Module automatically extracts so called *direct mappings* by relying on and extending the W3C specification, i.e., it extracts an ontological vocabulary from a relational schema, and it extracts mappings relating this vocabulary to the schema via SQL queries. The vocabulary consists of one class for each table that is not many-to-many, one property for each attribute and many-to-many table, and a special property for each foreign key (FK). The

bootstrapped mappings are similar to view definitions, but serve a different purpose and technically more involved. In particular, mappings address a so-called *impedance mismatch* problem: ontologies are object oriented and objects are identified by URIs, while relational DBs contain tuples of values. We implemented several strategies to generate URIs for tuples that rely on heuristics as well as DB constraints, e.g., primary and foreign keys guarantee coherent URI generation for tuples from different tables. Figure 3 contains a screenshot of one of our bootstrapping wizards.

Ontology Bootstrapping Module enriches the bootstrapped vocabulary with OWL 2 axioms extracted from databases and implements a number of novel ontology bootstrapping techniques that are both schema (i.e., transforming explicit and implicit database constraints into ontological axioms) and data driven. For example, we turn FKs into OWL 2 axioms of domain and range restrictions on properties. Here we rely on FKs that are explicitly in schemas and candidate FKs that we derive by checking containment between attribute values in different tables. Computation of implicit FKs is motivated by our observation that in EPDS some FKs are not specified. We proposed several techniques to induce OWL 2 class and property hierarchies, as well as disjointness axioms over bootstrapped vocabularies. For example, by checking that all attribute names of a table T_1 occur in the attributes of a table T_2, we create a candidate OWL 2 class inclusion axiom saying that the class $C[T_1]$ corresponding to T_1 is a subclass of $C[T_2]$. We verify these candidate axioms by looking at the data instances and return a ranked list of axioms. By looking at the common set of attributes A between similar tables T_1 and T_2 (we have several notions of similarity), we induce a candidate class $C[A]$ corresponding to A and axioms that $C[T_1]$ and $C[T_2]$ are subclasses of $C[A]$; then, the user should assign a meaningful name to $C[A]$. By looking at tables T_1 and T_2 with similar structure while non-overlapping tuples we induce candidate disjointness axioms between $C[T_1]$ and $C[T_2]$. Each primary key that is not null or each unique attribute is represented as a functional property axiom. We developed a number of other techniques that we do not present here due to space limit. After the bootstrapper computes the set of all candidate axioms, it checks the set for logical consistency using the ontology reasoner HermiT [9], repairs the ontology if it is inconsistent, and presents the remaining candidate axioms to the users for verification, i.e., the user can edit, accept, or discard candidate axioms.

Ontology Importing Module allows to incorporate an existing ontology O_1 in the system by aligning it with the ontology O_2 already used by the system, e.g., with the bootstrapped one. Alignment introduces subclass and equivalence axioms between O_1's and O_2's classes and properties. Based on our experience with bootstrapping and importing for EPDS, we extended LogMap so that it guarantees that the resulting aligned ontology does not violate the so-called conservativity principle wrt the vocabulary of O_2, i.e., it does not add (potentially) undesired inclusions among O_2 classes [21].

Ontology Layering Module offers *layering* of an input ontology over an input DB schema resulting in a set of direct mappings between the ontology and the schema, by relying on the IncMap system [16] that we developed for the Optique platform. The module represents the ontology and schema as graphs 'preserving' their structure, computes ranked correspondences between elements of the graphs using lexical and structural similarities as in the Similarity Flooding algorithm of Melnik et al. converts the correspondences into direct mappings between the ontology and schema, and finally offer the mappings to the user for verification. Different to bootstrapping and importing, layering can map user-specified fragments of DB schemata to user-specified fragments of ontologies.

R2RML Mapping Editor of the Optique platform is tailored towards W3C R2RML mappings for which direct mappings is a special case, and was evaluated with encouraging results [15]. It provides an intuitive mapping visualisation, semi-automatic suggestions of mapping corrections, and step-by-step wizards for writing complex (non direct) mappings.

IV. Query Formulation

Visual query formulation component of the platform, OptiqueVQS [23], allows to compose conjunctive tree-shaped queries by navigation over ontologies, has a widget-based architecture, and exploits multiple representation and interaction paradigms for query composition. Ontologies are object oriented and data conforming to an ontology is a set of statements of the form 'wellbore(uri123)', 'locatedIn(uri123,uri456)', and 'oilfield(uri456)' that are instantiations of classes and properties with (URIs representing) objects. These data can be seen as a *data-graph* where nodes correspond to objects and labeled with classes, while edges correspond to properties and labeled with property names. Data-graphs are often enriched with extra nodes and edges to encode class and property hierarchies, thus, they can partially include information from ontological axioms. Furthermore, it is common to design query formulation interfaces over an ontology by visualising (relevant fragments of) its data-graph and the query formulation process boils down to navigation through the data-graph. For OBDA, where the ontological data is virtual and the user has access only to the ontological axioms, data-graph driven query interfaces are not appropriate. Thus, we developed novel techniques to 'project' axioms rather than data in a graph structure as an *axiom-graph*: an OWL 2 axiom is projected into a set of nodes and edges relating them, where nodes correspond to classes and edges to properties [3]. Projecting axioms to graphs is not a trivial task since axioms are first-order logic formulae and do not have an immediate correspondence to graphs. In OptiqueVQS query construction is iterative, i.e., users construct queries step-by-step, and it boils down to navigation over an axiom-graph. At the moment the system supports axiom-graphs encoding those types of axioms which can be bootstrapped by the deployment module, including class hierarchies, and domain and range restrictions. E.g., consider two axioms saying that the classes 'wellbore' and 'oilfield' are respectively a domain and range of a property 'locatedIn', then the corresponding axiom-graph contains two nodes labeled respectively with 'wellbore' and 'oilfield', and one edge connecting these nodes labeled with 'locatedIn'. In Figure 3 there is a screenshot of OptiqueVQS, where in the upper part there is a query constructed by the user and in the lower-left part there is a fragment of axiom-graph relevant to the constructed query. Important feature of OptiqueVQS is that it does not require to store the axiom-graph: during each query construction session we compute (using logical reasoning with

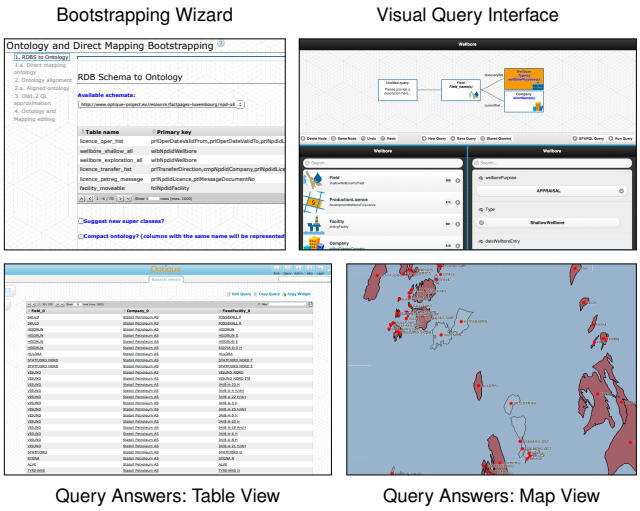


Fig. 3: Screenshots of the Optique platform

HermiT) relevant fragments of axiom-graph on-the-fly and present to the user. As we observed in our user studies [22], a purely axiom driven query interface suffers from important practical limitations, e.g., it does not allow users to set specific data values in queries, e.g., company names. To address this issue we enrich axiom-graphs with data annotations which we precompute, i.e., materialise, from the DBs underlying a given OBDA deployment instance by 'executing' relevant mappings. E.g., for EPDS we precomputed values that are frequently used, rarely changed, and from relatively small domains; this includes names of companies and oilfield, geolocations, ranges of numerical values, e.g., min/max possible depth of wellbores. Data values are visualised using sliders, drop boxes, etc., see the lower-right part of the interface in Figure 3.

V. Demonstration Scenario

We will demonstrate the Optique platform end-to-end, i.e., from deployment to query answering over two databases. Moreover, for these databases we prepared OBDA deployments with fine tuned ontologies and mappings and the attendees of the demo will be able to formulate queries over these deployments, load queries from query catalogs, execute queries and browse query answers on maps and in tables. We next describe the demonstration scenarios in detail.

Demonstration on NPD FactPages One deployment of the Optique platform is made over the NPD FactPages [20], an important public dataset heavily used in the oil and gas industry. This DB has 70 tables, 276 different attributes, 96 foreign keys, and about 50 MB of mostly aggregated data, e.g., seismic surveys. The choice of this demo database was motivated by its importance for the oil and gas industry and our work with Statoil within the Optique project. This deployment was tested by Statoil engineers who gave us positive feedback. Usage of this deployment requires a basic knowledge of geophysics.

Demonstration on MusicBrainz Database The other deployment of the Optique platform is made over the MusicBrainz database [1], which is an open music encyclopaedia that contains music information about roughly 830,000 artists, 1.2 million releases, and 13.2 million recordings. This domain does not require any special knowledge, so it is easy for anyone to use. This deployment was tested by students and the results were encouraging, e.g., students were able to accomplish query formulation tasks using OptiqueVQS.

End-to-End Demonstration Besides querying the two pre-configured deployments, the demo attendees will be able to deploy the Optique platform over both NPD FactPages and MusicBrainz databases and then query their own deployments. Specifically, one will be able to bootstrap an ontology and mappings from these databases, either *(i)* in a 'simple' mode, suitable for inexperienced users, with bootstrapping performed in an automatic regime using default parameters, or *(ii)* in a step-by-step mode that allows a user to tune the deployment parameters, e.g., to discover implicit database constraints and propagate them to the bootstrapped ontology. Then, one will be able to import pre-existing ontologies from our ontology catalogue. This can be performed in two ways: *(i)* either after the bootstrapping, in which case the imported and bootstrapped ontologies will be aligned, or *(ii)* using our ontology layering component, thus, skipping the bootstrapping step. Moreover, the attendees of the demo will be able to manually edit mappings with our mapping editor. Finally, they will be able to query the resulting deployments and browse query answers.

VI. References

[1] URL: http://musicbrainz.org/statistics.
[2] URL: http://fact-pages.fluidops.net/resource/demoICDE.
[3] M. Arenas et al. Faceted Search over Ontology-Enhanced RDF Data. In: *CIKM*. 2014.
[4] C. Bizer and A. Seaborne. D2RQ—treating non-RDF databases as virtual RDF graphs. In: *ISWC*. 2004.
[5] D. Calvanese et al. The MASTRO System for Ontology-Based Data Access. In: *Semantic Web* 2.1 (2011).
[6] M. Console et al. Efficient Approximation in DL-Lite of OWL 2 Ontologies. In: *DL*. 2013.
[7] J. Crompton. *Keynote talk at the W3C Workshop on Sem. Web in Oil & Gas Industry*. http://www.w3.org /2008/12/ogws-slides/Crompton.pdf. 2008.
[8] A. Doan et al. *Principles of Data Integration*. Morg. Kauf.'12.
[9] Glimm et al. Optimising Ontology Classification. In: *ISWC'10*.
[10] P. Haase et al. The Information Workbench as a Self-Service Platform for Linked Data Applications. In: *WWW*. 2012.
[11] I. Horrocks. What are ontologies good for? In: *Evolution of Semantic Systems*. Springer, 2013.
[12] E. Jimenez-Ruiz et al. Large-Scale Interactive Ontology Matching: Algorithms and Implementation. In: *ECAI'12*.
[13] E. Kharlamov et al. Optique 1.0: Semantic Access to Big Data: The Case of Norwegian Petroleum Directorate's FactPages. In: *ISWC (Posters & Demos)*. 2013.
[14] E. Kharlamov et al. Optique: Towards OBDA Systems for Industry. In: *ESWC (SE)*. 2013.
[15] C. Pinkel et al. How to Best Find a Partner? An Evaluation of Editing Approaches to Construct R2RML Mappings. In: *ESWC*. 2014.
[16] C. Pinkel et al. IncMap: Pay as You Go Matching of Relational Schemata to OWL Ontologies. In: *OM*. 2013.
[17] A. Poggi et al. Linking Data to Ontologies. In: *J. Data Semantics* 10 (2008).
[18] F. Priyatna et al. Formalisation and Experiences of R2RML-based SPARQL to SQL query translation using Morph. In: *WWW*. 2014.
[19] M. Rodriguez-Muro et al. Onto-logy-Based Data Access: Ontop of Databases. In: *ISWC*. 2013.
[20] M. G. Skjæveland et al. Publishing the NPD FactPages as Semantic Web Data. In: *ISWC*. 2013.
[21] A. Solimando et al. Detecting and Correcting Conservativity Principle Violations in Onto-to-Onto Mappings. In: *ISWC'14*.
[22] A. Soylu et al. Experiencing OptiqueVQS: A Multi-paradigm and Ontology-based Visual Query System for End Users. In: *Under Review*.
[23] A. Soylu et al. OptiqueVQS: Towards an Ontology-Based Visual Query System for Big Data. In: *MEDES*. 2013.

Appendix D

Faceted Search

This appendix reports the papers:

- Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, and Dmitriy Zheleznyakov. **Faceted Search over Ontology-Enhanced RDF Data**. CIKM 2014.
- Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, and Dmitriy Zheleznyakov. **Enabling Faceted Search over OWL 2 with SemFacet**. OWLED 2014
- Bernardo Cuenca Grau, Evgeny Kharlamov, Dmitriy Zheleznyakov, Marcelo Arenas, and Sarunas Marciuska. **On Faceted Search over Knowledge Bases**. Extended abstract. DL 2014
- Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, Dmitriy Zheleznyakov, and Yujiao Zhou. **Faceted Search over OWL 2 Life Science Datasets and Ontologies with SemFacet**. SWAT4LS 2014 Demo
- Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, Dmitriy Zheleznyakov, Marcelo Arenas, and Ernesto Jiménez-Ruiz. **SemFacet: Semantic Faceted Search over Yago**. WWW 2014 Demo
- Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, and Dmitriy Zheleznyakov. **Towards Semantic Faceted Search**. WWW 2014 Poster

Faceted Search over Ontology-Enhanced RDF Data

Marcelo Arenas[†]
PUC Chile

Bernardo Cuenca Grau[‡]
University of Oxford

Evgeny Kharlamov[‡]
University of Oxford

Sarunas Marciuska[‡]
University of Oxford

Dmitriy Zheleznyakov[‡]
University of Oxford

ABSTRACT

An increasing number of applications rely on RDF, OWL 2, and SPARQL for storing and querying data. SPARQL, however, is not targeted towards end-users, and suitable query interfaces are needed. Faceted search is a prominent approach for end-user data access, and several RDF-based faceted search systems have been developed. There is, however, a lack of rigorous theoretical underpinning for faceted search in the context of RDF and OWL 2. In this paper, we provide such solid foundations. We formalise faceted interfaces for this context, identify a fragment of first-order logic capturing the underlying queries, and study the complexity of answering such queries for RDF and OWL 2 profiles. We then study interface generation and update, and devise efficiently implementable algorithms. Finally, we have implemented and tested our faceted search algorithms for scalability, with encouraging results.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

Keywords

Faceted search; Ontology; OWL 2; RDF; SPARQL; Algorithms

1. INTRODUCTION

The Resource Description Framework (RDF) is the W3C recommendation graph data model for representing information about Web resources, and SPARQL is the standard language for querying RDF. In the last ten years, we have witnessed a constant growth in the amount of available RDF data, and an increasing number of applications are relying on RDF and SPARQL for storing, publishing, and querying data. The functionality of many such applications is enhanced by an OWL 2 ontology: a set of first-order sentences that are used to provide background knowledge about the application domain and enrich query answers with information not explicitly given in the RDF data.

Although the growing popularity of RDF, OWL 2, and SPARQL has been accompanied by the development of better and better query answering engines, writing SPARQL queries is not well-suited for the majority of users. Thus, an important challenge is the development of simple yet powerful query interfaces that capture well-defined fragments of SPARQL.

Faceted search is a prominent approach for querying document[1] collections where users can narrow down the search results by progressively applying filters, called *facets* [1]. A facet typically consists of a property (e.g., 'gender' or 'occupation' when querying documents about people) and a set of possible string values (e.g., 'female' or 'research'), and documents in the collection are annotated with property-value pairs. During faceted search users iteratively select facet values and the documents annotated according to the selection are returned as the search result.

Several authors have proposed faceted search for querying document collections annotated with RDF, and a number of systems have been developed, e.g. [2–10]. The theoretical underpinnings of faceted search in the context of semantic technologies, however, have received less attention [11–13]. In particular, the following key questions have not been satisfactorily addressed (see related work section).

(Q1) What fragments of SPARQL can be naturally captured using faceted search as a query paradigm?
(Q2) What is the complexity of answering such queries?
(Q3) What does it mean to generate and interactively update an interface according to a given RDF graph?

Questions 1 and 2 correspond to the study of expressive power and complexity of query languages in the context of faceted search. These are central topics in data management and addressing them is a key requirement to develop information systems that can provide correctness, robustness, scalability, and extensibility guarantees. Furthermore, update (Question 3) is a key task in information systems where query formulation is fundamentally interactive. Our first goal is to answer these questions, thus providing rigorous and solid foundations for faceted search over RDF data.

Furthermore, existing works have focused mostly on RDF, thus essentially disregarding the role of OWL 2 ontologies. We see this as an important limitation. Ontological axioms can be used to enrich query answers with implicit information, thus enhancing the search for relevant documents. Moreover, they provide schema-level structure, which can be exploited to improve faceted interfaces. Finally, RDF-based faceted search systems are data-centric, and hence cannot be exploited to browse large ontologies or to formulate meaningful queries at the schema level. Our second aim is to address this limitation and provide a framework for faceted search that is also applicable to the wider setting of OWL 2.

[1]We use 'document' to refer to any resource referenced by a URI.

Research supported by the Royal Society, the EPSRC projects Score! and MaSI[3], and the EU FP7 project "Optique" (n. 318338).
[†]Email: marenas@ing.puc.cl
[‡]Email: first.middle.lastname@cs.ox.ac.uk

CIKM'14, November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2662027.

$(1)\ A(x) \wedge R(x, y_1) \wedge B(y_1) \wedge R(x, y_2) \wedge B(y_2) \rightarrow y_1 \approx y_2$,
$(2)\ R(x, y) \rightarrow S(x, y)$, $(3)\ A(x) \rightarrow \exists y.[R(x, y) \wedge B(y)]$,
$(4)\ A(x) \rightarrow x \approx a$, $(5)\ R(x, y) \wedge S(y, z) \rightarrow T(x, z)$,
$(6)\ A(x) \rightarrow B(x)$, $(7)\ A(x) \wedge B(x) \rightarrow C(x)$,
$(8)\ R(x, y) \rightarrow A(x)$, $(9)\ A(x) \wedge R(x, y) \rightarrow B(y)$,
$(10)\ A(x) \rightarrow R(x, a)$, $(11)\ R(x, a) \rightarrow B(x)$,
$(12)\ R(x, y) \rightarrow A(y)$, $(13)\ R(x, y) \rightarrow S(y, x)$,
$(14) R(x, y) \wedge B(y) \rightarrow A(x)$

Table 1: Rules corresponding to OWL 2 profiles

In Section 3 we formalise faceted interfaces that are tailored towards RDF and OWL 2 and which capture the key functionality implemented in existing faceted search systems. Our interfaces capture both the combination of facets displayed during search, and the facet values selected by users. In this way, an interface encodes both a query, whose answers constitute the current search results, and the facet values available for further selection. Analogously to existing work on RDF-based faceted search and in contrast to traditional faceted search, our notion of interface allows users to 'navigate' across interconnected collections of documents and establish filters to each of them. Furthermore, it abstracts from considerations specific to GUI design (e.g., facet and value ranking), while at the same time reflecting the core functionality of existing systems.

In Section 4 we study the expressivity and complexity of *faceted queries*: queries encoded by faceted interfaces. To this end, we identify a fragment of first-order logic that is sufficient to capture such queries, and study the complexity of query answering in the presence of OWL 2 ontologies. Since OWL 2 reasoning can be computationally expensive and hence significantly affect systems' performance and robustness, we focus on ontologies in the OWL 2 profiles [14]: language fragments with favorable computational properties. For each of these profiles we establish tight complexity bounds and propose practical query answering algorithms.

In Section 5 we study interface generation and update. Existing techniques for RDF are based on exploration of the underlying RDF graph. In this way, by generating facets according to the RDF graph, systems can guide users in the formulation of 'meaningful' queries. We lift this approach by proposing a graph-based representation of OWL 2 ontologies and their logical entailments for the purpose of faceted navigation. Then, we characterise what it means for an interface to conform to an ontology, in the sense that every facet and facet value in the interface is justified by an edge in the graph (and hence by an entailment of the ontology). Finally, we propose generic interface generation and update algorithms that rely on the information in the graph, and show tractability of these tasks for ontologies in the OWL 2 profiles.

In Section 6 we present a faceted search platform that provides functionality for generating and updating interfaces based on our algorithms in Section 5. Our platform relies on an external triple store with OWL 2 reasoning capabilities, and it is compatible with faceted search GUIs, as well as with text search engines for retrieving documents from keywords. We have tested the scalability of our platform using different triple stores, with encouraging results. As proof of concept, we have integrated our platform in a faceted search system that bundles the triple store JRDFox, the search engine Lucene, and our own faceted search GUI.

2. PRELIMINARIES

We use standard notions from first-order logic. We assume pairwise disjoint infinite sets of *constants* $\mathbf{C}$, *unary predicates* $\mathbf{UP}$, and *binary predicates* $\mathbf{BP}$. A *signature* is a subset of $\mathbf{C} \cup \mathbf{UP} \cup \mathbf{BP}$. We treat equality $\approx$ as an ordinary predicate in $\mathbf{BP}$, and assume that any set of formulae contains the axioms of equality for its signature. We treat $\top$ as a special symbol in $\mathbf{UP}$, which is used to represent a tautology. W.l.o.g. we assume all formulae to be rectified; that is, no variable appears free and quantified in a first-order formula φ, and every variable is quantified at most once in φ. The set of free variables of a formula φ is denoted as $\mathsf{fvar}(\varphi)$.

A *fact* is a ground atom and a *dataset* is a finite set of facts. A *rule* is a sentence of the form $\forall \mathbf{x} \forall \mathbf{z}\, [\varphi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})]$, where $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{y}$ are pairwise disjoint tuples of variables, the *body* $\varphi(\mathbf{x}, \mathbf{z})$ is a conjunction of atoms with variables in $\mathbf{x} \cup \mathbf{z}$, and the *head* $\exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})$ is an existentially quantified non-empty conjunction of atoms $\psi(\mathbf{x}, \mathbf{y})$ with variables in $\mathbf{x} \cup \mathbf{y}$. Universal quantifiers are omitted. The restriction of $\psi(\mathbf{x}, \mathbf{y})$ being non-empty ensures satisfiability of any set of rules and facts, which makes query results meaningful. A rule is *Datalog* if its head has at most one atom and all variables are universally quantified.

OWL 2 defines three *profiles*: weaker languages with favourable computational properties [14]. Each profile ontology can be normalised as rules and facts using the correspondence of OWL 2 and first-order logic and a variant of the structural transformation.[2] Thus, we see an *ontology* as a finite set of rules and facts. Table 1 specifies the rules allowed in these profiles. An ontology with only sentences from Table 1 is *(i)* RL if it has no rules of Type (3); *(ii)* EL if it does not contain rules (1), (9), and (13); and *(iii)* QL if it does not contain rules (1), (4), (5), (7), (9)-(11), and (14).

Let V be a signature, $\mathsf{at}(V)$ the set of equality-free and constant-free atoms over V, and $\mathsf{eq}(V)$ the set of atoms $x \approx c$ with x a variable and c a constant from V. A *positive existential query* (PEQ) $Q(\mathbf{x})$ is a formula with free variables $\mathbf{x}$, constructed using $\wedge$, $\vee$ and $\exists$ from atoms in $\mathsf{at}(V) \cup \mathsf{eq}(V)$. A PEQ Q is *monadic* if $\mathsf{fvar}(Q)$ is a singleton, and it is a *conjunctive query* (CQ) if it is $\vee$-free.

We consider two different semantics for query answering. Under the *classical semantics*, a tuple $\mathbf{t}$ of constants is an *answer* to $Q(\mathbf{x})$ w.r.t. an ontology $\mathcal{O}$ if $\mathcal{O} \models Q(\mathbf{t})$. Under the *active domain semantics*, $\mathbf{t}$ is an answer to Q w.r.t. $\mathcal{O}$ if there is a tuple $\mathbf{t}'$ of constants from $\mathcal{O}$ such that $\mathcal{O} \models \varphi(\mathbf{t}, \mathbf{t}')$, where $\varphi(\mathbf{x}, \mathbf{y})$ is the formula obtained from Q by removing all quantifiers. The evaluation problem under classical (resp. active domain) semantics is to decide, given a tuple of constants $\mathbf{t}$, a PEQ Q and an ontology $\mathcal{O}$ in a language $\mathcal{L}$, whether $\mathbf{t}$ is an answer to Q w.r.t. $\mathcal{O}$ under the given semantics. The classical semantics is the default in first-order logic, whereas active domain is the default semantics of the SPARQL entailment regimes [15]. The latter can be seen as an approximation of the former (an active domain answer is also an answer under classical semantics, but not vice versa). The difference between both semantics manifests itself only in the presence of existentially quantified rules and queries; thus, both semantics coincide if either the input ontology is Datalog, or if all variables in the input query are free.

3. FACETED INTERFACES

In this section, we formalise a notion of a *faceted interface*, which provides a rigorous foundation for faceted search over RDF graphs enhanced with OWL 2 ontologies. To motivate our definitions, we will use an example based on an excerpt of DBpedia. Our goal is to find US presidents who graduated from Harvard or Georgetown and have a child who graduated from Stanford.

EXAMPLE 1. *The document URIs d_{tr} and d_{bc} for Theodore Roosevelt and Bill Clinton are annotated with the category 'president'. Roosevelt's son Kermit d_{kr} and Clinton's daughter Chelsea d_{cc} are categorised as 'person'. The document URIs for Georgetown d_g, Harvard d_h, and Stanford d_s are categorised under 'university', and the USA d_{us} and UK d_{uk} as 'country'. These annotations are given in RDF and correspond to the following facts:*

[2]Note that the profiles provide the special concept $\bot$, which is immaterial to query answering over satisfiable profile ontologies.

$\mathsf{President}(d_{tr})$ $\mathsf{President}(d_{bc})$ $\mathsf{Person}(d_{kr})$
$\mathsf{Person}(d_{cc})$ $\mathsf{Country}(d_{us})$ $\mathsf{Country}(d_{uk})$
$\mathsf{Univ}(d_h)$ $\mathsf{Univ}(d_g)$ $\mathsf{Univ}(d_s)$

Specific information about documents is represented by literals. For example, Theodore Roosevelt's date of birth is encoded as $\mathsf{dateOfBirth}(d_{tr}, \textit{1858-10-27})$*. Most importantly, documents are also annotated with other documents; such annotations are represented in RDF and correspond to the following facts:*

$\mathsf{citiz}(d_{tr}, d_{us})$ $\mathsf{citiz}(d_{bc}, d_{us})$ $\mathsf{child}(d_{tr}, d_{kr})$ $\mathsf{child}(d_{bc}, d_{cc})$
$\mathsf{grad}(d_{tr}, d_h)$ $\mathsf{grad}(d_{bc}, d_g)$ $\mathsf{grad}(d_{kr}, d_h)$ $\mathsf{grad}(d_{cc}, d_s)$

Finally, DBpedia can be extended with ontological rules, which are exploited to describe the meaning of the predicates and constants in the vocabulary. Consider for example the rules given next:

$$\mathsf{President}(x) \wedge \mathsf{citiz}(x, d_{us}) \rightarrow \mathsf{USpres}(x), \quad (1)$$
$$\mathsf{USpres}(x) \rightarrow \mathsf{President}(x) \wedge \mathsf{citiz}(x, d_{us}), \quad (2)$$
$$\mathsf{grad}(x, y) \rightarrow \mathsf{Person}(x) \wedge \mathsf{Univ}(y), \quad (3)$$
$$\mathsf{Person}(x) \rightarrow \exists y.\big(\mathsf{citiz}(x, y) \wedge \mathsf{Country}(y)\big). \quad (4)$$

Rules (1) *and* (2) *define US presidents as those with US nationality. Rule* (3) *specifies the domain and range of* grad*. Finally,* (4) *mandates that each person has a (maybe unspecified) nationality.*

Analogously to traditional faceted search, we represent *facets* as pairs of a predicate (or facet name) and a set of values. In the context of RDF, however, documents can be used to annotate other documents, and thus annotations form a graph, rather than a tree. Thus, facet values can be either document URIs or literals. Examples of facet names are the relations 'grad' and 'dateOfBirth', and example values are documents such as 'd_s' and literals such as '1858-10-27'. Selection of multiple values within a facet can be interpreted conjunctively or disjunctively, and hence we distinguish between conjunctive and disjunctive facets. We also distinguish a special facet type, whose values are categories (i.e., unary predicates) rather than documents or literals. Finally, a special value any denotes the set of all values compatible with the facet name.

DEFINITION 2. *Let* type *and* any *be symbols not occurring in* $\mathbf{C} \cup \mathbf{UP} \cup \mathbf{BP}$*. A* facet *is a pair* $(X, \circ\Gamma)$*, with* $\circ \in \{\wedge, \vee\}$*,* Γ *a non-empty set, and either (i)* $X = \mathsf{type}$ *and* $\Gamma \subseteq \mathbf{UP}$*, or (ii)* $X \in \mathbf{BP}$*,* $\mathsf{any} \in \Gamma$ *and either* $\Gamma \subseteq \mathbf{C} \cup \{\mathsf{any}\}$ *or* $\Gamma \subseteq \mathbf{UP} \cup \{\mathsf{any}\}$*. A facet of the form* $(X, \wedge\Gamma)$ *is* conjunctive*, and a facet of the form* $(X, \vee\Gamma)$ *is* disjunctive*. In a facet* $F = (X, \circ\Gamma)$*,* X *is the* facet name*, denoted by* $F|_1$*, and* Γ *contains the* facet values *and it is denoted by* $F|_2$*.*

EXAMPLE 3. *The following facets are relevant to our example.*

$F_1 = (\mathsf{type}, \vee\{\mathsf{USpres}, \mathsf{Country}\})$,
$F_2 = (\mathsf{child}, \vee\{\mathsf{any}, d_{kr}, d_{cc}\})$, $F_3 = (\mathsf{grad}, \vee\{\mathsf{any}, d_h, d_s, d_g\})$,
$F_4 = (\mathsf{citiz}, \wedge\{\mathsf{any}, d_{us}, d_{uk}\})$, $F_5 = (\mathsf{citiz}, \vee\{\mathsf{any}, d_{us}, d_{uk}\})$.

The disjunctive facet F_1 *can be exploited to select the categories to which the relevant documents belong. Facet* F_2 *can be used to narrow down search results to those individuals with children; furthermore, the value* any *can be used to state that we are not looking for any specific child. The intuition behind* F_3*,* F_4*, and* F_5 *is similar; note, however, that facet* F_4 *is conjunctive.*

3.1 The Notion of Faceted Interface

We next move on to the definition of a faceted interface, which encodes both a query (whose answers determine the search results) and the choices of facet values available for further refinement.

Figure 1: A visualisation of the example interface

DEFINITION 4. *A* basic faceted interface *(BFI) is a pair* (F, Σ)*, with* F *a facet and* $\Sigma \subseteq F|_2$ *the set of* selected values*. The set of* faceted interfaces *(or interfaces, for short) is given by the following grammar, where* I_0 *and* $I_1 = (F, \Sigma)$ *are BFIs and* $F|_1 \in \mathbf{BP}$*:*

$$I ::= \mathit{path} \mid (\mathit{path} \wedge \mathit{path}) \mid (\mathit{path} \vee \mathit{path}),$$
$$\mathit{path} ::= I_0 \mid (I_1/I).$$

A BFI encodes user choices for a specific facet, e.g., the BFI $(F_1, \{\mathsf{USpres}\})$ selects the documents categorised as US presidents. BFIs are put together in *paths*: sequences of nested facets that capture navigation between sets of documents. Documents are annotated with other documents by means of binary relations (e.g., child connects parents to their children); thus, nesting (I_1/I) requires the BFI I_1 to have a binary relation as facet name. With nesting we can capture queries such as 'people with a child who graduated from Stanford' by using the interface $(F_2, \{\mathsf{any}\})/(F_3, \{d_s\})$ which first selects people having (any) children and then those children with a Stanford degree. Finally, two types of branching can be applied: $(\mathit{path}_1 \wedge \mathit{path}_2)$ indicates that search results must satisfy the conditions specified by both path_1 and path_2, while $(\mathit{path}_1 \vee \mathit{path}_2)$ indicates that they must satisfy those in path_1 or path_2.

EXAMPLE 5. *Consider the following interface* I_{ex}*, which is visualised in our system as in Figure 1.*

$$\big((F_1, \{\mathsf{USpres}\}) \wedge (F_3, \{d_h, d_g\})\big) \wedge \big((F_2, \{\mathsf{any}\})/(F_3, \{d_s\})\big).$$

The interface consists of three paths connected by $\wedge$*-branching. The first path selects US presidents. The second path selects graduates of Harvard or Georgetown. The third path selects individuals with a child who is a Stanford graduate. Since paths are combined conjunctively their constraints apply simultaneously. Thus, we obtain the US presidents who graduated from either Harvard or Georgetown and who have a child who graduated from Stanford.*

The query encoded by the selected values in an interface is formally specified in terms of first-order logic as given next.

DEFINITION 6. *Let* I *be an interface, and let each* x_w *with* $w \in \{0, 1, \ldots, 9, \cdot\}^*$ *be a variable. The query of* I *is the formula* $Q[I] = [\![I, x_\varepsilon, x_0]\!]$*, with one free variable* x_ε*, defined as in Table 2.*

Our semantics assigns to each interface a PEQ with one free variable. For each facet F we have $[\![(F, \emptyset), v, x_w]\!] = \top(v)$, indicating that no restriction is imposed by F if no value is selected. BFIs with a type-facet are interpreted as the conjunction (disjunction) of unary atoms over the same variable. BFIs having as facet name a binary predicate result in either an atom whose second argument is existentially quantified (if any is selected), or in a conjunction (disjunction) of binary atoms having a variable as second argument that must be equal to a constant or belong to a unary predicate. Branching $(\mathit{path}_1 \circ \mathit{path}_2)$ with $\circ \in \{\wedge, \vee\}$ is interpreted by constructing the conjunction (disjunction) of the queries for each path_i; furthermore, if for some path_i we have that $[\![\mathit{path}_i, v, x_w]\!] = \top(v)$, indicating that no value from the facets occurring in path_i is selected,

Basic Faceted Interfaces: $[\![(F,\Sigma),v,x_w]\!] =$	
$\top(v)$	if $\Sigma = \emptyset$
$\exists x_w\, F\vert_1(v,x_w)$	if $\mathsf{any} \in \Sigma$
$\mathop{\circ}_{C\in\Sigma} C(v)$	if $F\vert_1 = \mathsf{type}$ and $\Sigma \neq \emptyset$
$\mathop{\circ}_{t_i\in\Sigma} \exists x_{w\cdot i}\, F\vert_1(v,x_{w\cdot i}) \wedge x_{w\cdot i} \approx t_i$	if $F\vert_1 \neq \mathsf{type}$, $\mathsf{any} \notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C}$
$\mathop{\circ}_{C_i\in\Sigma} \exists x_{w\cdot i}\, F\vert_1(v,x_{w\cdot i}) \wedge C_i(x_{w\cdot i})$	if $F\vert_1 \neq \mathsf{type}$, $\mathsf{any} \notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP}$
Nesting: $[\![((F,\Sigma)/I),v,x_w]\!] =$	
$\top(v)$	if $\Sigma = \emptyset$
$\exists x_w\, F\vert_1(v,x_w) \wedge [\![I,x_w,x_{w\cdot 0}]\!]$	if $\mathsf{any} \in \Sigma$
$\mathop{\circ}_{t_i\in\Sigma} \exists x_{w\cdot i}\, F\vert_1(v,x_{w\cdot i}) \wedge x_{w\cdot i} \approx t_i \wedge [\![I,x_{w\cdot i},x_{w\cdot i\cdot 0}]\!]$	if $\mathsf{any} \notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C}$
$\mathop{\circ}_{C_i\in\Sigma} \exists x_{w\cdot i}\, F\vert_1(v,x_{w\cdot i}) \wedge C_i(x_{w\cdot i}) \wedge [\![I,x_{w\cdot i},x_{w\cdot i\cdot 0}]\!]$	if $\mathsf{any} \notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP}$
Branching: $[\![(\mathit{path}_1 \circ \mathit{path}_2),v,x_w]\!] =$	
$([\![\mathit{path}_1,v,x_{w\cdot 0}]\!] \circ [\![\mathit{path}_2,v,x_{w\cdot 1}]\!])$	if $[\![\mathit{path}_1,v,x_{w\cdot 0}]\!] \neq \top(v)$ $[\![\mathit{path}_2,v,x_{w\cdot 1}]\!] \neq \top(v)$
$[\![\mathit{path}_1,v,x_{w\cdot 0}]\!]$	if $[\![\mathit{path}_1,v,x_{w\cdot 0}]\!] \neq \top(v)$ $[\![\mathit{path}_2,v,x_{w\cdot 1}]\!] = \top(v)$
$[\![\mathit{path}_2,v,x_{w\cdot 1}]\!]$	if $[\![\mathit{path}_1,v,x_{w\cdot 0}]\!] = \top(v)$ $[\![\mathit{path}_2,v,x_{w\cdot 1}]\!] \neq \top(v)$
$\top(v)$	if $[\![\mathit{path}_1,v,x_{w\cdot 0}]\!] = \top(v)$ $[\![\mathit{path}_2,v,x_{w\cdot 1}]\!] = \top(v)$

Table 2: Semantics of faceted interfaces

then path_i is ignored. Finally, nesting involves a "shift" of variable from the parent BFI to the nested subexpression.

EXAMPLE 7. *Interface I_{ex} encodes the following query:*

$$Q_{\mathsf{ex}}(x) = \mathsf{USpres}(x) \wedge \big(\exists y_1\,(\mathsf{grad}(x,y_1) \wedge y_1 \approx d_h) \vee \exists y_2\,(\mathsf{grad}(x,y_2) \wedge y_2 \approx d_g)\big) \wedge \exists z\,\big(\mathsf{child}(x,z) \wedge \exists w(\mathsf{grad}(z,w) \wedge w \approx d_s)\big).$$

If we consider only facts, the answer is empty (e.g., no document is categorised as 'US president'). If we also consider the ontology rules, however, we obtain d_{bc} (i.e., Bill Clinton) as an answer.

Our notion of interface motivates the class of *faceted queries*, i.e., PEQs that can be captured by some faceted interface.

DEFINITION 8. *A first-order formula φ is a* faceted query *if there exists a faceted interface I such that φ and $Q[I]$ are identical modulo renaming of variables.*

Note that our notion of interface abstracts from several considerations that are critical to GUI design. For instance, our notion is insensitive to the order of BFIs composed by $\wedge$- or $\vee$- branching, as well as to the order of facet values (which are carefully ranked in practice). Furthermore, we model type-facet values as 'flat', whereas in applications categories are organised hierarchically. Although these issues are important from a front-end perspective, they are immaterial to our technical results.

3.2 Faceted Interfaces with Refocussing

The interface in Example 5 finds presidents (such as Bill Clinton) who graduated from either Harvard or Georgetown and have children who graduated from Stanford. If we want to know who these children are (i.e., see Chelsea Clinton as an answer), we must provide *refocussing* (or *pivoting*) functionality [9, 10]. We next extend faceted interfaces to allow for such functionality.

DEFINITION 9. *Let* focus *be a symbol not occurring in $\mathbf{C} \cup \mathbf{UP} \cup \mathbf{BP}$. An* extended basic faceted interface *(EBFI) is either a BFI or a pair $(F, \Sigma \cup \{\mathsf{focus}\})$, where (F,Σ) is a BFI and $F\vert_1 \in \mathbf{BP}$. Moreover, the set of* extended faceted interfaces *(EFIs) is defined by the same grammar given in Definition 5, but where I_0 is a BFI and $I_1 = (F,\Delta)$ is an EBFI with $F\vert_1 \in \mathbf{BP}$. Finally, each EFI I must have at most one occurrence of the symbol* focus*.*

Extended Basic Faceted Interfaces: $[\![(F,\Sigma\cup\{\mathsf{focus}\}),v,x_w]\!] =$	
$F\vert_1(v,x_w)$	if $\Sigma = \emptyset$
$[\![(F,\{\mathsf{focus}\}),v,x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C} \cup \{\mathsf{any}\}$
$[\![\big((F,\{\mathsf{focus}\})/((\mathsf{type},\vee F\vert_2),\Sigma)\big),v,x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP} \cup \{\mathsf{any}\}$
Nesting: $[\![((F,\Sigma\cup\{\mathsf{focus}\})/I),v,x_w]\!] =$	
$F\vert_1(v,x_w) \wedge [\![I,x_w,x_{w\cdot 0}]\!]$	if $\Sigma = \emptyset$
$[\![((F,\{\mathsf{focus}\})/I),v,x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C} \cup \{\mathsf{any}\}$
$[\![\big((F,\{\mathsf{focus}\})/\big(((\mathsf{type},\vee F\vert_2),\Sigma)\wedge I\big)\big),v,x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP} \cup \{\mathsf{any}\}$

Table 3: Semantics of extended faceted interfaces

The special value focus is used to change the free variable of the query Q, which determines the kinds of objects returned as answers. Thus, refocussing is used over a facet that generates new variables in the query, which by Table 2 requires that $F\vert_1 \in \mathbf{BP}$.

The query encoded by an extended interface can be specified in terms of first-order logic as given next.

DEFINITION 10. *Let I be an EFI and $[\![I,x_\varepsilon,x_0]\!]$ be a formula defined by the extension of Table 2 with the rules in Table 3. Then the* query *of I is the formula $Q[I]$ defined as follows:*

$$Q[I] \;=\; \begin{cases} [\![I,x_\varepsilon,x_0]\!] & \text{if } \mathsf{focus} \text{ does not occur in } I, \\ \exists x_\varepsilon\,[\![I,x_\varepsilon,x_0]\!] & \text{otherwise.} \end{cases}$$

Finally, a formula φ is an extended faceted query *if there is an EFI I s.t. φ and $Q[I]$ are identical modulo renaming of variables.*

For example, consider the following EFI I, which is focused on the children of the US presidents:

$$\big((F_1,\{\mathsf{USpres}\}) \wedge (F_3,\{d_h,d_g\})\big) \wedge \big((F_2,\{\mathsf{focus}\})/(F_3,\{d_s\})\big).$$

Then, $Q[I]$ is obtained from $Q_{\mathsf{ex}}(x)$ in Example 7 by first dropping the existential quantifier $\exists z$ from $Q_{\mathsf{ex}}(x)$, and then adding the existential quantifier $\exists x$ to the resulting query, thus obtaining $Q'_{\mathsf{ex}}(z)$:

$$\exists x\,\big(\mathsf{USpres}(x) \wedge \big(\exists y_1\,(\mathsf{grad}(x,y_1) \wedge y_1 \approx d_h) \vee \exists y_2\,(\mathsf{grad}(x,y_2) \wedge y_2 \approx d_g)\big) \wedge \big(\mathsf{child}(x,z) \wedge \exists w(\mathsf{grad}(z,w) \wedge w \approx d_s)\big)\big).$$

The answer to $Q_{\mathsf{ex}}(z)$ is precisely d_{cc} (Chelsea Clinton).

4. ANSWERING FACETED QUERIES

Each time a user selects a facet value to refine the search results, a faceted search system must compute the answers to a query. Thus, query evaluation is a key reasoning problem for the development of efficient and robust faceted search systems.

As discussed in Section 3, faceted queries are monadic positive existential queries resulting from the selection of facet values in an interface. By standard results for relational databases, PEQ evaluation is an NP-hard problem, even if we restrict ourselves to CQs and ontologies consisting of just a dataset.

Our main result is that, in contrast to PEQs (and even CQs), faceted query evaluation over datasets is tractable; furthermore, the problem remains tractable in most cases if we consider ontologies belonging to the OWL 2 profiles. Our tractability results concern

Algorithm 1: ANSWER-FQ: Faceted Queries over Datasets

INPUT : $\mathcal{D}$ a dataset; Q a faceted query
OUTPUT: Answers to Q w.r.t. $\mathcal{D}$

$S :=$ Set of disjunctive subformulas of Q
$\preceq :=$ partial order on S s.t. $\varphi \preceq \varphi'$ iff φ is a subformula of φ'
for each $\varphi = (\varphi_1 \vee \varphi_2) \in S$ *listed in ascending* $\preceq$*-order* **do**
  for each $1 \leq i \leq 2$ **do**
    $\varphi'_i :=$ REWRITE(φ_i)
    $Ans_i :=$ ANSWER-TREE-CQ$(\varphi'_i, \mathcal{D})$
  $\mathcal{D} := \mathcal{D} \cup \{C_{\varphi_1 \vee \varphi_2}(d) \mid d \in Ans_1 \cup Ans_2\}$
$Q' :=$ REWRITE(Q)
Ans $:=$ ANSWER-TREE-CQ$(Q', \mathcal{D})$
return *Ans*

Function REWRITE

INPUT : φ a faceted query
OUTPUT: A conjunctive query

case φ *an atom* **return** φ
case $\varphi = \exists z.\varphi'$ **return** $\exists z.$REWRITE(φ')
case $\varphi = \varphi_1 \wedge \varphi_2$ **return** REWRITE$(\varphi_1) \wedge$ REWRITE(φ_2)
case $\varphi = \varphi_1 \vee \varphi_2$ **return** $C_{\varphi_1 \vee \varphi_2}(y)$ with $y = \mathsf{fvar}(\varphi_i)$

combined complexity, which takes into account the size of the entire input (i.e., ontological rules, RDF data and queries).

4.1 Faceted Query Answering Over Datasets

The rationale behind our tractability result is that PEQs originating from faceted interfaces are of a rather restricted shape, which is determined by Table 2 in Section 3. A closer look at the table reveals that variables in a faceted query can be arranged in a tree with root x_ε and where each variable $x_{w.i}$ is a child of x_w.

DEFINITION 11. *Let $Q(x)$ be a monadic PEQ.* The graph *of Q is the smallest directed graph G_Q with a node for each variable in Q and a directed edge (y, y') for each atom $R(y, y')$ occurring in Q where R is different from $\approx$. Moreover, Q is* tree-shaped *if (i) G_Q is a (possibly empty) directed tree rooted at x; (ii) for each edge (y, y') there is at most one binary atom in Q of the form $R(y, y')$.*

Note that query $Q_{\mathsf{ex}}(x)$ in Example 7 is tree-shaped. The second observation in Table 2 is that disjunction in a faceted query originates from either a disjunctive facet or from $\vee$-branching between paths. In either case, disjunctive subqueries are monadic tree-shaped PEQs. These observations are summarised as follows:

PROPOSITION 12. *Every faceted query Q is a monadic tree-shaped PEQ with the following property: if $\varphi = (\varphi_1 \vee \varphi_2)$ is a subformula of Q, then $\mathsf{fvar}(\varphi_1) = \mathsf{fvar}(\varphi_2) = \{x\}$ for some x.*

We next show how the restricted shape of faceted queries can be exploited to make query answering more efficient. We start by providing a polynomial algorithm for answering faceted queries over datasets. The key observation is that the disjunctive subqueries $\varphi = \varphi_1 \vee \varphi_2$ in the input query Q can be evaluated w.r.t. the input dataset in a 'bottom-up' fashion . To answer one such φ, we solve φ_1 and φ_2 independently and 'store' the answers as facts in the dataset using a fresh unary predicate C_φ uniquely associated to φ.

EXAMPLE 13. *Query Q_{ex} can be answered over the dataset in our running example as follows. First, solve the subquery φ asking for graduates from either Harvard or Georgetown; each disjunct is a tree-shaped CQ, and we obtain B. Clinton, T. Roosevelt and K. Roosevelt as answers. Then, extend the dataset with facts $C_\varphi(d_{bc})$, $C_\varphi(d_{tr})$ and $C_\varphi(d_{kr})$ over a fresh predicate C_φ. Finally, rewrite Q_{ex} by replacing $\varphi(x)$ with $C_\varphi(x)$ and answer the rewritten query over the extended dataset. We obtain the empty set of answers since no document is explicitly categorised as US president.*

Algorithm 2: ANSWER-FQ-ACTIVE

INPUT : $\mathcal{O}$ an ontology; Q a faceted query
OUTPUT : Active domain answers to Q w.r.t. $\mathcal{O}$

$\mathcal{D} :=$ COMPUTE-ENTAILED-FACTS$(\mathcal{O})$
Ans $:=$ ANSWER-FQ$(Q, \mathcal{D})$
return *Ans*

Algorithm 1 implements these ideas. The algorithm relies on a specialised algorithm ANSWER-TREE-CQ to answer (monadic) tree-shaped CQs, which is used as a 'black box'. The following theorem establishes correctness of our algorithm.

THEOREM 14. *Algorithm 1 computes all answers to Q w.r.t. $\mathcal{D}$.*

Thus, faceted queries can be evaluated in polynomial time with an oracle for the evaluation of tree-shaped CQs. By a classic result, acyclic CQs (and hence also tree-shaped CQs as in Def. 11) can be answered in polynomial time [16]. Thus, tractability tree-shaped CQ evaluation transfers to the evaluation of faceted queries.

COROLLARY 15. *Faceted query evaluation over datasets is feasible in polynomial time.*

In what follows we study query answering over ontologies (and not just datasets) under both active domain and classical semantics.

4.2 Active Domain Semantics

In practice, queries over ontology-enhanced RDF data are typically represented in SPARQL and executed using off-the-shelf reasoning engines with SPARQL support. The specification of SPARQL under entailment regimes [15] is based on active domain semantics, which requires existentially quantified variables in the query Q to map to actual constants in the input ontology $\mathcal{O}$. In this case, we can answer queries using Algorithm 2, which first computes the dataset $\mathcal{D}$ of all facts entailed by $\mathcal{O}$ and then answers Q w.r.t. the dataset $\mathcal{D}$. The correctness of Algorithm 2 follows directly from Theorem 14 and the following proposition.

PROPOSITION 16. *Let Q be a PEQ, let $\mathcal{O}$ be an ontology, and let $\mathcal{D}$ be the set of all facts α such that $\mathcal{O} \models \alpha$. Then, the active domain answers to Q w.r.t. $\mathcal{O}$ and w.r.t. $\mathcal{D}$ coincide.*

Fact entailment is tractable for all the OWL 2 profiles; thus, by committing to the active domain semantics of SPARQL we can achieve tractability without emasculating the ontology language.

THEOREM 17. *Active domain evaluation of faceted queries is in* PTIME *w.r.t. all normative OWL 2 profiles. Furthermore, it is* PTIME*-complete w.r.t. the EL and RL profiles.*

4.3 Classical Semantics

Classical and active domain semantics coincide if we restrict ourselves to Datalog ontologies. Thus, Algorithm 2 can also be used for query answering under classical semantics if the input ontology is Datalog. An immediate consequence is that our results in Theorem 17 transfer to OWL 2 RL ontologies under classical semantics.

In contrast to RL, the EL and QL profiles can capture existentially quantified knowledge and hence active domain and classical semantics may diverge for queries with existentially quantified variables. To deal with EL ontologies, we exploit techniques developed for the *combined approach* to query answering [17, 18]. These techniques are currently applicable to *guarded* EL ontologies, i.e., EL ontologies without axioms of Type (5). The idea is to rewrite rules of Type (3) in Table 1 into Datalog by Skolemising existentially quantified variables into fresh constants.

DEFINITION 18. *Let $\mathcal{O}$ be in EL. The ontology $\Xi(\mathcal{O})$ is obtained from $\mathcal{O}$ by replacing each rule $A(x) \rightarrow \exists y.[R(x,y) \wedge B(y)]$ with $A(x) \rightarrow P(x, c_{R,B})$, $P(x,y) \rightarrow R(x,y)$, $P(x,y) \rightarrow B(y)$, where P is a fresh predicate and $c_{R,B}$ is a globally fresh constant uniquely associated with R and B.*

Although this transformation strengthens the ontology, it preserves the entailment of facts [17]. The following theorem establishes that the evaluation of faceted queries is also preserved.

THEOREM 19. *Let Q be a faceted query, $\mathcal{O}$ a guarded EL ontology, and c a constant in $\mathcal{O}$. Then, $\mathcal{O} \models Q(c)$ iff $\Xi(\mathcal{O}) \models Q(c)$.*

Thus, we can answer faceted queries over an EL ontology $\mathcal{O}$ by applying Algorithm 2 to $\Xi(\mathcal{O})$. Since Ξ is a linear transformation and $\Xi(\mathcal{O})$ is an RL ontology, we can conclude tractability of faceted query evaluation for EL (a result consistent with existing results for acyclic CQs in EL [19]). In contrast, the evaluation of acyclic CQs is already NP-hard for OWL 2 QL [20] and we can show that faceted query evaluation is NP-complete for OWL 2 QL. The following theorem summarises our results.

THEOREM 20. *Faceted query evaluation under classical semantics is (i)* PTIME*-complete for RL and guarded EL ontologies; and (ii)* NP*-complete for QL ontologies.*

4.4 Extended Faceted Queries

We conclude by arguing that the refocussing functionality does not increase complexity of query evaluation. PEQs obtained from EFIs satisfy Proposition 12, with the only difference that the corresponding query graph is no longer rooted in the answer variable. Algorithm 1 can be extended to prove that Corollary 15 also holds for extended faceted queries. From this, and using the same techniques as in the proofs of Theorems 17 and 20, we obtain that:

PROPOSITION 21. *Extended faceted query evaluation under classical semantics is (i)* PTIME*-complete for RL and guarded EL; and (ii)* NP*-complete for QL. Moreover, active domain evaluation of extended faceted queries is in* PTIME *w.r.t. all normative OWL 2 profiles, and it is* PTIME*-complete for RL and EL.*

5. INTERFACE GENERATION & UPDATE

Faceted navigation is an interactive process. Starting with an initial interface generated from a keyword search, users 'tick' or 'untick' facet values and the system reacts by updating both search results (query answers) and facets available for further navigation.

EXAMPLE 22. *Consider the interactive construction of our example interface I_{ex}. Navigation starts with the following interface with no selected value, which may have been generated as a response to a keyword search (facets F_i are given in Example 3):*

$$I_0 = (F_1, \emptyset) \wedge (F_3, \emptyset) \wedge (F_2, \emptyset) \wedge (F_5, \emptyset).$$

We may then select the category USpres *in F_1, which narrows down the search to US presidents. In response, the system may construct the following new interface I_1:*

$$I_1 = (F_1, \{\mathsf{USpres}\}) \wedge (F_3, \emptyset) \wedge (F_2, \emptyset).$$

Interface I_1 incorporates the required filter on US presidents. Furthermore, it no longer includes facet F_5 since US presidents have only US nationality and hence any filter over this facet becomes redundant. Next, we select Harvard and Georgetown in facet F_3, which narrows down the search to US presidents with either a Harvard or Georgetown degree and yields the following interface:

$$I_2 = (F_1, \{\mathsf{USpres}\}) \wedge (F_3, \{d_h, d_g\}) \wedge (F_2, \emptyset).$$

Next, we select any *in facet F_2 to look for presidents with children. In response, the system constructs the following interface:*

$$I_3 = (F_1, \{\mathsf{USpres}\}) \wedge (F_3, \{d_h, d_g\}) \wedge \big((F_2, \{\mathsf{any}\})/(F_3, \emptyset)\big).$$

Interface I_3 provides a nested BFI $(F_3, \emptyset)$, which allows us to select the university that children of US presidents attended. We pick Stanford, and the system finally constructs I_{ex}.

We next propose interface generation and update algorithms that are 'guided' by the (explicit and implicit) information in $\mathcal{O}$. Our algorithms are based on the same principle: each element of the initial interface (resp. each change in response to an action) must be 'justified' by an entailment in $\mathcal{O}$. In this way, by exploring the ontology, we guide users in the formulation of meaningful queries.

There is an inherent degree of non-determinism in faceted navigation: if a user selects a facet value, it is unclear whether the next facet generated by the system should be conjunctive or disjunctive, and whether it should be incorporated in the interface by means of conjunctive or disjunctive branching. In applications, however, different values in a facet are typically interpreted disjunctively, whereas constraints imposed by different facets are interpreted conjunctively. Thus, to resolve such ambiguities and devise fully deterministic algorithms, we focus on a restricted class of interfaces where conjunctive facets and disjunctive branching are disallowed.

DEFINITION 23. *A faceted interface I is* simple *if all facets occurring in I are disjunctive, and it does not contain sub-interfaces of the form $(path_1 \vee path_2)$.*

5.1 The Ontology Facet Graph

We capture the facets that are relevant to an ontology $\mathcal{O}$ in what we call a *facet graph*. The graph can be seen as a concise representation of $\mathcal{O}$, and our interface generation and update algorithms are parameterised by such graph rather than by $\mathcal{O}$ itself.

The nodes of a facet graph are possible facet values (unary predicates and constants), and edges are labelled with possible facet names (binary predicates and type). The key property of a facet graph is that every X-labelled edge (v, w) is justified by a rule or fact entailed by $\mathcal{O}$ which 'semantically relates' v to w via X. We distinguish three kinds of semantic relations: *existential*, where X is a binary predicate and (each instance of) v must be X-related to (an instance of) w in the models of $\mathcal{O}$; *universal*, where (each instance of) v is X-related only to (instances of) w in the models of $\mathcal{O}$; and *typing* where $X = \mathsf{type}$, and (the constant) v is entailed to be an instance of (the unary predicate) w.

DEFINITION 24. *A* facet graph *for $\mathcal{O}$ is a directed labelled multigraph G having as nodes unary predicates or constants from $\mathcal{O}$ 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. $\mathcal{O} \models \alpha_e$ 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

$$R(c,d) \quad \textit{or} \quad R(c,y) \rightarrow y \approx d;$$

(ii) if e is $c \xrightarrow{R} A$, then α_e is a rule of the form

$$\top(c) \rightarrow \exists y.[R(c,y) \wedge A(y)] \quad \textit{or} \quad R(c,y) \rightarrow A(y);$$

(iii) if e is $A \xrightarrow{R} c$, then α_e is a rule of either of the form

$$A(x) \rightarrow R(x,c) \quad \textit{or} \quad A(x) \wedge R(x,y) \rightarrow y \approx c;$$

(iv) if e is $A \xrightarrow{R} B$, then α_e is a rule of the form

$$A(x) \rightarrow \exists y.[R(x,y) \wedge B(y)] \quad \textit{or} \quad A(x) \wedge R(x,y) \rightarrow B(y);$$

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

Moreover, $\mathsf{range}_G(R)$ *denotes the set of nodes in G with an incoming R-labelled edge.*

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

EXAMPLE 25. *Recall our ontology in Example 1. A facet graph may contain nodes for d_{bc} (Bill Clinton) and d_{cc} (Chelsea Clinton), as well as for predicates such as* USpres *and* Univ. *Example edges are: (i) a* child*-edge linking d_{bc} to d_{cc}, which is justified by the fact* $\mathsf{child}(d_{bc}, d_{cc})$*; (ii) a* citiz*-edge from* Person *to* Country *justified by Rule* (4)*; and (iii) a* grad*-edge from d_{cc} to* Univ *since d_{cc} graduated from Stanford and hence the ontology entails* $\mathsf{Person}(d_{cc}) \rightarrow \exists y.(\mathsf{grad}(d_{cc}, y) \wedge \mathsf{Univ}(y))$.

It follows from the following proposition that facet graph computation can be efficiently implemented. In practice, the graph can be precomputed when first loading data and ontology, stored in RDF, and accessed using SPARQL queries. In this way, reasoning tasks associated to faceted search are performed offline.

PROPOSITION 26. *Checking whether a directed labelled multigraph is a facet graph for $\mathcal{O}$ is feasible in polynomial time if $\mathcal{O}$ is in any of the OWL 2 profiles.*

To realise the idea of ontology-guided faceted navigation, we require that interfaces *conform* to the facet graph, in the sense that the presence of every facet and value in the interface is supported by a graph edge. In this way, we ensure that interfaces mimic the structure of (and implicit information in) the ontology and the interface does not contain irrelevant (combinations of) facets. Since a given facet or value can occur in many different places in an interface, we need a mechanism for unambiguously referring to each element in the interface. To this end, we introduce an alternative representation of interfaces in the form of a tree. This representation will also be instrumental to our notions of update in Section 5.3.

DEFINITION 27. *The node-labelled tree* $\mathsf{tree}(I) = (N, E, \lambda)$ *of a simple EFI I is recursively defined as follows.*

(i) If I is an EBFI, then $N = \{\varepsilon\}$, $E = \emptyset$, and $\lambda(\varepsilon) = I$.

(ii) If $I = (I_0 \wedge I_1)$ where $\mathsf{tree}(I_i) = (N_i, E_i, \lambda_i)$, then

$$\begin{aligned} N &= \{\varepsilon\} \cup \{0w \mid w \in N_0\} \cup \{1w \mid w \in N_1\}, \\ E &= \{(\varepsilon, 0), (\varepsilon, 1)\} \cup \{(iu_1, iu_2) \mid (u_1, u_2) \in E_i\}. \end{aligned}$$

Furthermore, $\lambda(w) = \varepsilon$ if $w = \varepsilon$, and $\lambda(w) = \lambda_i(u)$ if w of the form iu with $i \in \{0, 1\}$.

(iii) If $I = (I_0 / I_1)$, where $\mathsf{tree}(I_1) = (N_1, E_1, \lambda_1)$, then

$$\begin{aligned} N &= \{\varepsilon\} \cup \{0w \mid w \in N_1\}, \\ E &= \{(\varepsilon, 0)\} \cup \{(0u_1, 0u_2) \mid (u_1, u_2) \in E_1\}. \end{aligned}$$

Furthermore, $\lambda(\varepsilon) = I_0$, and for each $w \in N \setminus \{\varepsilon\}$ it holds that $\lambda(w) = \lambda_1(u)$ where $w = 0u$.

A position *in I is a pair (w, v) where w is a node in* $\mathsf{tree}(I)$ *with label an EBFI (F, Σ) and $v \in F|_2 \cup \{\mathsf{focus}\}$.*

We can now define conformance of an interface to a facet graph.

DEFINITION 28. *Let G be a facet graph for $\mathcal{O}$ and I a simple EFI. Let (w_1, v_1) and (w_2, v_2) be distinct positions in I, where $\lambda(w_i)$ in* $\mathsf{tree}(I)$ *is (F_i, Σ_i) and $F_i|_1 = X_i$ for $i = 1, 2$. Position (w_2, v_2) is* justified *by (w_1, v_1) in G if w_1 is the least ancestor of w_2 in* $\mathsf{tree}(I)$ *with $\lambda(w_1) \neq \varepsilon$ and one of the following holds: (i) there is an X_2-labelled edge from v_1 to v_2; or (ii) $v_1 =$* any *and there is an X_2-labelled edge from some $u \in \mathsf{range}_G(X_1)$ to v_2; or (iii) $v_2 =$* any *and v_1 has an outgoing X_2-edge; or (iv) $v_1 = v_2 =$* any *and u has an outgoing X_2-edge for some $u \in \mathsf{range}_G(X_1)$.*

Interface I conforms to *G if for each position (w, v) in I, one of the following holds: (i) there is no ancestor w' of w in* $\mathsf{tree}(I)$ *with $\lambda(w) \neq \varepsilon$; or (ii) there is a position (w', v') in I s.t. $\lambda(w')$ is (F', Σ'), $v' \in \Sigma'$ and (w, v) is justified by (w', v') in G.*

Algorithm 3: CREATEINTERFACE

INPUT : A facet graph $G = (V, E)$ for $\mathcal{O}$, a set S of nodes in G
OUTPUT : A simple faceted interface

1 $\Upsilon = \{w \mid v \xrightarrow{\mathsf{type}} w \in E \text{ and } v \in S\}$
2 $I = ((\mathsf{type}, \vee\Upsilon), \emptyset)$
3 **for each** $R \in \mathbf{BP}$ **do**
4 $\quad \Gamma, \Upsilon' := \emptyset$
5 $\quad$ **for each** $v \in S$ *and* $v \xrightarrow{R} w \in E$ **do**
6 $\quad\quad$ **if** w *is a constant* **then** $\Gamma := \Gamma \cup \{w\}$
7 $\quad\quad$ **else** $\Upsilon' := \Upsilon' \cup \{w\}$
8 $\quad$ **if** $\Gamma \neq \emptyset$ **then** $I := I \wedge ((R, \vee(\Gamma \cup \{\mathsf{any}\})), \emptyset)$
9 $\quad$ **if** $\Upsilon' \neq \emptyset$ **then** $I := I \wedge ((R, \vee(\Upsilon' \cup \{\mathsf{any}\})), \emptyset)$
10 **return** I

Intuitively, (w_2, v_2) is justified by (w_1, v_1) if there is an edge from v_1 to v_2 labelled with the facet name X_2 of F_2. This indicates that there is an entailment in $\mathcal{O}$ that justifies the appearance of v_2 given v_1 and X_2. Our definition, however, must also consider that v_1 can be any, which indicates that any value reachable by using the facet name X_1 of facet F_1 can be used to justify v_2. Analogously, v_2 can also be any, in which case it is enough to use v_1 to justify any value reachable by using the facet name X_2.

5.2 Interface Generation

Algorithm 3 shows how a fresh interface can be generated from a starting set mS of nodes in a facet graph G. The algorithm starts by grouping all unary predicates categorising the constants in S in a BFI (Lines 1-2). Then, for each binary predicate R and each $v \in S$, the algorithm collects the nodes w with an incoming R-edge from v and groups them in sets Γ and Υ' depending on whether they are constants or unary predicates (Lines 3-7). All constants in Γ (resp. predicates in Υ') are put together in a BFI with facet name R, which is coupled to the interface using $\wedge$-branching (Lines 8-9).

Algorithm 3 can be directly exploited to generate an initial interface from a set of keywords. A faceted search backend would first compute an initial set D of documents relevant to the keywords (e.g., using a text search engine), and then generate an initial interface by calling Algorithm 3 with input D and a facet graph for $\mathcal{O}$. The resulting interface I has no selected facet values or nested facets, which reflects that I constitutes the starting point to navigation. Furthermore, I is conformant to the input graph G.

PROPOSITION 29. *On input G and S, Algorithm 3 outputs a simple interface that conforms to G.*

5.3 Interface Update

The initial interface where no facet value has been yet selected marks the start of the navigation process. User actions on an interface can be seen as elementary 'ticking' and 'unticking' operations on facet values that result in another interface. We define these actions by exploiting the tree representation of interfaces (c.f. Definition 27). We start with the ticking operation.

DEFINITION 30. *The action* TICK *is applicable to a simple EFI I, a position (w, v) in I, and a facet graph G for $\mathcal{O}$ under the following preconditions:* (i) *v is not selected in $\lambda(w)$ and* (ii) *if an ancestor w' of w in* $\mathsf{tree}(I)$ *is labelled with an EBFI (F', Σ'), then $\Sigma' \neq \emptyset$. The result is the interface computed by Algorithm 4.*

Algorithm 4: TICK

INPUT : I, (w, v), and G as in Def. 30, with $\lambda(w) = (F, \Sigma)$
OUTPUT : A simple EFI

1 **if** $v = \mathsf{focus}$ **then**
2 $\quad I_{out} :=$ remove all occurrences of focus in I, and then replace Σ in $\lambda(w)$ with $\Sigma \cup \{\mathsf{focus}\}$
3 **else**
4 $\quad I_1 :=$ replace Σ in I with $\Sigma \cup \{v\}$
5 $\quad$ **if** $v \in \mathbf{C} \cup \mathbf{UP}$ **then** $I_2 :=$ CREATEINTERFACE$(G, \{v\})$
6 $\quad$ **else** $I_2 :=$ CREATEINTERFACE$(G, \mathsf{range}_G(F|_1))$
7 $\quad$ **if** w *is a leaf in* $\mathsf{tree}(I_1)$ **then**
8 $\quad\quad I_{out} :=$ replace $\lambda(w)$ in I_1 with $(\lambda(w)/I_2)$
9 $\quad$ **else** $I_{out} :=$ replace $\lambda(w0)$ in I_1 with $(\lambda(w0) \wedge I_2)$
10 **return** I_{out}

Algorithm 5: UNTICK

INPUT : I, (w, v) and G as in Def. 32, with $\lambda(w) = (F, \Sigma)$
OUTPUT : A simple EFI

1 **if** $v = \mathsf{focus}$ **then** $I_{out} :=$ replace Σ in I with $\Sigma \setminus \{\mathsf{focus}\}$
2 **else**
3 $\quad S := \{(w', v') \mid (w', v')$ is uniquely justified by (w, v) in G, $\lambda(w') = (F', \Sigma')$ and $v' \in \Sigma\}$
4 $\quad$ **for each** $(w', v') \in S$ **do** $I :=$ UNTICK$(I, (w', v'), G)$
5 $\quad I_{out} :=$ replace Σ in I with $\Sigma \setminus \{v\}$
6 $\lambda_{out} :=$ labelling function of $\mathsf{tree}(I_{out})$
7 **for each** *node* w' *in* $\mathsf{tree}(I_{out})$ **do**
8 $\quad (F', \Sigma') := \lambda_{out}(w')$
9 $\quad$ **if** $\lambda_{out}(w'') = (F'', \emptyset)$ *for some ancestor* w'' *of* w' *in* $\mathsf{tree}(I_{out})$ **then** $I_{out} :=$ replace Σ' in I_{out} with $\emptyset$
10 **return** I_{out}

Algorithm TICK starts by checking whether the value v is focus, in which case it adds v to Σ and removes all other occurrences of focus in I (Lines 1-2). Otherwise, it generates a fresh EFI I_1 from I by adding v to Σ (Line 4), and constructs a new EFI I_2 that collects all the values adjacent to v in G (Line 5). Notice that if $v = \mathsf{any}$, then the value v itself is not considered; instead, v is replaced by the values in G with an incoming $F|_1$-labelled edge. Finally, Algorithm TICK includes in I_1 the navigation alternatives encoded in I_2 by considering two cases. If w is a leaf in $\mathsf{tree}(I_1)$, then we incorporate I_2 via nesting by replacing $\lambda(w)$ in I_1 with $(\lambda(w)/I_2)$ (Line 7); otherwise, w has a nested child $w0$ in $\mathsf{tree}(I_1)$, in which case the navigation alternatives encoded in I_2 are included in $w0$ by replacing $\lambda(w0)$ in I_1 with $(\lambda(w0) \wedge I_2)$.

PROPOSITION 31. *Assume that I, (w, v) and G are as in Definition 30. If I conforms to G, then* TICK$(I, (w, v), G)$ *is a simple EFI that also conforms to G.*

We next define the unticking operation. Intuitively, when unticking a value v in a given position of an interface all values that were justified by v (and only by v) should also be unticked. In particular, we say that (w_2, v_2) is *uniquely justified* by (w_1, v_1) in G if (w_2, v_2) is justified by (w_1, v_1) in G and (w_2, v_2) is not justified in G by any pair other than (w_1, v_1).

DEFINITION 32. *The action* UNTICK *is applicable to a simple EFI I, a position (w, v) in I and a facet graph G for an ontology $\mathcal{O}$, if $v \in \Sigma$ with (F, Σ) the label of w in $\mathsf{tree}(I)$. The result is the interface computed by Algorithm 5 .*

Algorithm UNTICK considers two cases depending on what kind of value v is unticked. If v is focus, then the value is simply unselected (Line 1). Otherwise, not only Σ must be replaced in I with $\Sigma \setminus \{v\}$, but also all the positions in I that are uniquely justified by (w, v) have to be unticked (Lines 2-5). Unticking propagates recursively along the tree of I since positions deeper down the tree could ultimately be affected. Finally, the algorithm makes sure that no selected value remains 'disconnected' with the rest (Lines 7-9).

PROPOSITION 33. *Assume that I, (w, v) and G are as in Definition 32. If I conforms to G, then* UNTICK$(I, (w, v), G)$ *is a simple EFI that also conforms to G.*

5.4 Minimising Interfaces

An important issue in the design of faceted interfaces is to avoid the overload of users with redundant facets or facet values. Intuitively, an (unselected) facet value v is redundant if selecting v either leads to a 'dead end' (i.e., an empty set of answers) or it does not have an effect on query answers. Then, a faceted interface is minimal if none of its component BFIs contains redundant values.

DEFINITION 34. *Let I be a simple EFI and G a facet graph for $\mathcal{O}$. Then I is minimal w.r.t. G if for each position (w, v) in I s.t.* TICK *is applicable to I, (w, v) and G, the following holds: (i) $Q[\text{TICK}(I, (w, v), G)]$ has non empty answers w.r.t. $\mathcal{O}$; and (ii) the answers to $Q[\text{TICK}(I, (w, v), G)]$ w.r.t. $\mathcal{O}$ are different from the answers to $Q[I]$ w.r.t. $\mathcal{O}$.*

Note that the transition from interface I_0 to I_1 in Example 22 involves a minimisation step. The BFI in I_0 involving F_5 is pruned since ticking a value will either not affect the search results (if any or d_{us} is ticked) or yield an empty set of answers (if d_{uk} is ticked).

To avoid overwhelming users with irrelevant information, our system minimises the output of Alg. 4 before showing it to the user. Our system 'runs' each possible expansion of a EFI in the background by calling the reasoning engine, and prunes all facet values that do not change query answers, or make them empty.

6. IMPLEMENTATION AND TESTING

We have developed a faceted search platform providing the following main functionality: *(i)* computation of facet graphs from an ontology; *(ii)* interface generation from facet graphs; and *(iii)* interface update in response to user actions. Our platform relies on an external triple store for querying and OWL reasoning.

In our implementation, facet graphs are represented in RDF: each R-labelled edge from v to w is stored as a triple (v, R, w). A graph G can be either loaded from an existing RDF document, or constructed from (the facts entailed by) the ontology $\mathcal{O}$ by adding additional edges. The kinds of relevant additional edges are described by means of customisable rules, and the edges themselves are computed by materialisation of such rules. Furthermore, our platform implements Algorithm 3 for generating interfaces, Algorithms 4 and 5 for interface update, and strategies for interface minimisation. Our algorithms can operate both under the assumption that the facet graph G is explicitly materialised, or it is defined virtually using rules and then generated 'on the fly' as needed.

We have implemented a proof-of-concept system, called SemFacet, that bundles our platform with JRDFox[3] as triple store, Lucene for keyword search, and an HTML 5 GUI [21]. The system's architecture is given in Figure 2(c). Our system is available online.[4]

6.1 Performance Metrics

Performance of our platform critically depends on the following parameters of the underlying triple store, which can be estimated empirically by benchmarking the triple store over the dataset of interest: (i) t[*run query*]: time to execute an atomic query; and (ii) t[*look up*]: time to iterate over query results.

Interface generation (Algorithm 3) requires computing all triples (v, w, u) in the facet graph G for each v in the input nodes S, and then iterating over the results to compose the interface. Thus, to estimate the cost of interface generation (t_{CI}), we can use Alg. 6

[3] www.cs.ox.ac.uk/isg/tools/RDFox/

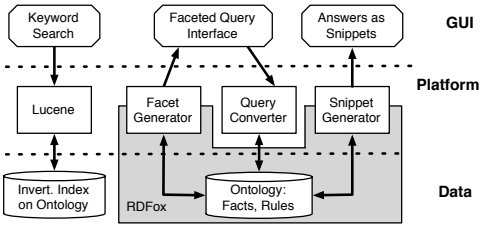
[4] http://www.cs.ox.ac.uk/isg/tools/SemFacet/

#(answers)	JRDFox	Stardog	Sesame
100	0.000	0.010	0.011
1,000	0.000	0.064	0.060
10,000	0.002	0.521	0.294
100,000	0.021	2.934	0.566
1,000,000	0.206	4.475	2.513
10,000,000	2.056	n/a	n/a

(a) Average runtime in seconds for lookup in a set of query answers

#(queries)	JRDFox	Stardog	Sesame
1	0.000	0.007	0.012
10	0.000	0.188	0.233
100	0.004	2.414	0.630
1,000	0.059	5.666	3.683
10,000	0.498	15.025	26.126
100,000	4.799	n/a	n/a

(b) Average runtime in seconds for processing a set of queries



(c) Architecture of our faceted search system

Figure 2: Experimental results for JRDFox, Stardog, and Sesame

Algorithm 6: CREATEINTERFACEIMPLEMENTED

INPUT : G: facet graph; S: set of nodes in G
OUTPUT: A simple faceted interface

1 $I :=$ Empty interface
2 **for each** $v \in S$ **do**
3 | Triples$_v$:= SELECT ?y,?z FROM G WHERE (v, ?y, ?z)
4 | **for each** $t \in$ *Triples*$_v$ **do** $I :=$ COMPOSEINTERFACE(t, I)

instead of Alg. 3. We assume constant time for the call to COMPOSEINTERFACE. The cost can then be estimated as follows:

$$t_{CI} = (|S| \times t[\textit{run query}]) + (\#[\textit{answers}] \times t[\textit{look up}]). \quad (5)$$

In this expression, $\#[\textit{answers}]$ is the union of all sets Triples$_v$ for each $v \in S$. In the worst-case, $\#[\textit{answers}]$ is $|G|$, whereas in the best-case it corresponds to $|S|$. For improved efficiency, our platform implements a variation of Algorithm 6 where facets are computed lazily: facet names are computed first, and values are computed 'on demand' when users click on a facet. For this, we modify the query in Line 3 such that $?y$ is the only answer variable. Then, $\#[\textit{answers}]$ is estimated as follows, where the number of facet names corresponds to the number of different edge labels in G, and the number of facet values to the number of nodes:

$$\#[\textit{answers}]_{\textit{naive}} = O(\#[\textit{facet names}]) \times O(\#[\textit{facet values}]),$$
$$\#[\textit{answers}]_{\textit{lazy}} = O(\#[\textit{facet names}]).$$

The cost t_{CI} in (5) can also be used to estimate the cost of interface updates. The Algorithm for ticking (Sec. 5.3) can be seen as a variant of Alg. 6 with S the set of values relevant to the tick. In the case of unticking, the worst-case cost is estimated as $k \times t_{CI}$, with k the number of selected values in the interface. Indeed, k measures the worst-case number of recursive calls to UNTICK (Alg. 5), whereas t_{CI} estimates the cost of a single recursive call.

6.2 Performance Estimations

To estimate the parameters $t[\textit{run query}]$ and $t[\textit{look up}]$, thus also estimating the cost t_{CI} of interface generation, we have conducted experiments over a fragment of DBpedia enriched with OWL 2 RL rules and we have used JRDFox, Stardog (`http://stardog.com/`) and Sesame (`http://www.openrdf.org/`). All experiments were conducted on a MacBook Pro laptop with OS X 10.8.5, 2.4 GHz Intel Core i5 processor, and 8GB 1333 MHz DDR3 memory. Since triple stores such as JRDFox operate in main memory, and we wanted to test our algorithms on stock hardware, we considered a fragment that covers 20% of DBpedia (3.5 million triples) and which can be loaded using 8GB of RAM. Each experiment was executed 100 times it total, and we measured average and median running time for each experiment. Since results never differ in more than 5% for a single experiment, we report only average times.

Results are summarised in Figures 2(a) and 2(b). Figure 2(a) estimates $\#[\textit{answers}] \times t[\textit{look up}]$ by measuring time required to iterate over an answer set of a given size. In turn, Figure 2(b) estimates $|S| \times t[\textit{run query}]$ by computing the times required for the triple store to answer a given number of atomic queries. We can make the following observations: (i) The time needed to iterate over query results is small in comparison to query execution times; for example, to execute $10,000$ queries, JRDFox requires $0.498s$, whereas to iterate over $10,000$ answers it requires $0.002s$. This should be taken into account when optimising interface generation. (ii) In some triple stores (i.e., Stardog and Sesame), iteration and query answering times do not grow linearly, and they have to be determined empirically. In contrast, JRDFox shows linear behaviour.

We first discuss query execution times. To generate the initial interface, the size of S is determined by the number of relevant results returned by the search engine from keywords. If the ranking algorithm of the search engine produces high quality results, one can establish a cap on S. As shown in Figure 2(b), obtaining a reasonable cap is important since query execution is expensive. For example with a cap of $1,000$ results in S, JRDFox would execute the queries necessary for interface generation almost instantaneously.

Concerning iteration times over query results, JRDFox could perform this task in $0.2s$ for 1 million results and $2s$ for 10 million. We were not even able to conduct experiments with 10 million answers over Stardog and Sesame since loading the data in our machine consumed all RAM and system behavior became unstable. The facet graph for the whole of DBpedia contains 24 million facet values and $1,843$ facet names [4]. JRDFox would require $5s$ in the worst-case to iterate through that many values using the exhaustive algorithm. When computing interfaces lazily, all triple stores would complete the required iteration over facet names instantaneously.

7. RELATED WORK

The design of visual interfaces for querying ontologies has received significant attention in recent years. Existing systems typically support query formulation by exploiting either a form of controlled natural language (e.g., Quelo [22]), or different graphical representations for queries (e.g., SEWASIE [23], iSPARQL [24], OntoVQL [25], Wonder [26], or the OptiqueVQS [27], or other approaches, including interactive exploration of [28].

Faceted search over RDF has also attracted a great deal of attention. Developed systems include mSpace [5], /facet [7], Piggy Bank [8], Tabulator [2], gFacet [6], Humboldt [9], Parallax [10], Longwell [29], faceted DBpedia [4], X-ENS [3], Broccoli [30], and others [31–33]. The functionalities provided by these systems include navigation through different sets of documents, refocussing, and interface minimisation via elimination of dead-ends.

These works are primarily systems-oriented and their main focus is on improving user experience, development of ranking functions and value grouping heuristics [11, 13], and backend optimisation via indexing schemes [4, 34]. Our framework was inspired by the capabilities of existing systems, and covers their main functionalities. Since our aim was to study the fundamental properties of query languages and update tasks, our framework abstracts from (and is compatible with) usability, ranking, and indexing considerations.

The expressivity of the query languages supported by existing systems is discussed in the literature mostly verbally, which makes it difficult to determine the underpinning SPARQL fragment. Most systems seem to support some form of conjunctive queries (e.g.,

see [11, 13]), and disjunction is present only in a limited form [3, 4]. The approach of [12] allows for conjunction, disjunction, and other operators, e.g., negation, thus, they cover a wider fragment of SPARQL than we do. At the same time, [12] is orthogonal to other faceted search approaches for RDF, including ours: their facet values are possible *queries* rather than (set of) documents, and a selection of a facet value corresponds to a syntactic query transformation rather than to setting a filter on a set of documents. Expressiveness of this approach is determined by the expressiveness of queries that are allowed to be used as facet values.

When query languages have not been formalised, the complexity of query answering was not addressed. The common assumption is that user selections in an interface are compiled in SPARQL [12] or Prolog [7], and executed by a query evaluation engine over the underlying RDF data. Complexity considerations are, however, critical when RDF data is enhanced with OWL 2 reasoning. This setting was not addressed by existing systems, where ontological axioms are limited to class and property hierarchies [7, 11], and reasoning plays little or no role. Interface generation and update mechanisms are mostly informally described. A common approach is to generate and update interfaces from the RDF data graphs. Since we generate interfaces from facet graphs that subsume RDF datasets, we see our approach as a generalisation of existing work. Finally, scalability of faceted search systems over large RDF datasets is an important concern [4, 34]. Since facet graphs can be much larger than the underlying RDF datasets, scalability becomes even more critical in our setting. Our experiments, however, suggest that our approach is feasible in practice.

The works closest to ours are [11–13]. The query language in [13] is formalised using CQs, whereas the language in [11] (also conjunctive) is introduced via set operations. These works, however, do not study the complexity of query answering, and ontological reasoning is also not considered. We can also find notions of facet trees and graphs in the literature [7, 11, 13, 29, 35]. These represent combinations of (possibly nested) facets displayed in a GUI as a tree or a graph, and they depend on both search results and front-end considerations. We see our notions of interface and facet graph as GUI-independent generalisation of existing notions since our graphs are derived from ontologies and independently from search results. Finally, the 'navigation graph' of [12] defines navigation links at the syntactic level as query transformations, rather than semantic relations between sets and objects, as in our case.

8. CONCLUSION AND FUTURE WORK

In this paper, we have established theoretical foundations for faceted search in the context of RDF and OWL 2. Our results suggest many problems for future work, such as exploring extensions of our update algorithms beyond simple interfaces. Concerning system design, substantial work is needed to improve GUI design, especially with respect to refocussing. We are also planning to benchmark our platform on real-world applications.

9. REFERENCES

[1] D. Tunkelang. *Faceted Search*. Morgan & Claypool Publishers, 2009.

[2] T. Berners-Lee, J. Hollenbach, K. Lu, J. Presbrey, E. Prudhommeaux, and M. M. C. Schraefel. Tabulator Redux: Browsing and Writing Linked Data. In: *LDOW*. 2008.

[3] P. Fafalios and Y. Tzitzikas. X-ENS: Semantic Enrichment of Web Search Results at Real-Time. In: *SIGIR*. 2013.

[4] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, et al. Faceted Wikipedia Search. In: *BIS*. 2010.

[5] m.c. schraefel, D. A. Smith, A. Owens, A. Russell, C. Harris, and M. L. Wilson. The Evolving mSpace Platform: Leveraging the Semantic Web on the Trail of the Memex. In: *Hypertext*. 2005.

[6] P. Heim, J. Ziegler, and S. Lohmann. gFacet: A Browser for the Web of Data. In: *IMC-SSW*. 2008.

[7] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. In: *ISWC*. 2006.

[8] D. Huynh, S. Mazzocchi, and D. R. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In: *J. Web Sem.* 5.1 (2007).

[9] G. Kobilarov and I. Dickinson. Humboldt: Exploring Linked Data. In: *LDOW*. 2008.

[10] D. F. Huynh and D. R. Karger. Parallax and Companion: Set-based Browsing for the Data Web. 2013.

[11] E. Oren, R. Delbru, and S. Decker. Extending Faceted Navigation for RDF Data. In: *ISWC*. 2006.

[12] S. Ferré and A. Hermann. Semantic Search: Reconciling Expressive Querying and Exploratory Search. In: *ISWC*. 2011.

[13] A. Wagner, G. Ladwig, and T. Tran. Browsing-oriented Semantic Faceted Search. In: *DEXA*. 2011.

[14] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles. In: *W3C Recommendation* (2009).

[15] *W3C: SPARQL 1.1 Entailment Regimes*. `www.w3.org/TR/sparql11-entailment/`.

[16] M. Yannakakis. Algorithms for Acyclic Database Schemes. In: *VLDB*. 1981.

[17] G. Stefanoni, B. Motik, and I. Horrocks. Introducing Nominals to the Combined Query Answering Approaches for EL. In: *AAAI*. 2013.

[18] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The Combined Approach to Ontology-Based Data Access. In: *IJCAI*. 2011.

[19] M. Bienvenu, M. Ortiz, M. Simkus, and G. Xiao. Tractable Queries for Lightweight Description Logics. In: *IJCAI*. 2013.

[20] S. Kikot, R. Kontchakov, and M. Zakharyaschev. On (In)Tractability of OBDA with OWL 2 QL. In: *DL*. 2011.

[21] M. Arenas, B. Cuenca Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, and E. Jiménez-Ruiz. SemFacet: Semantic Faceted Search over Yago. In: *WWW*. 2014.

[22] E. Franconi, P. Guagliardo, M. Trevisan, and S. Tessaris. Quelo: an Ontology-Driven Query Interface. In: *DL*. 2011.

[23] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The SEWASIE Network of Mediator Agents for Semantic Search. In: *J. UCS* 13.12 (2007).

[24] *iSPARQL QBE*. `http://dbpedia.org/isparql/`.

[25] A. Fadhil and V. Haarslev. OntoVQL: A Graphical Query Language for OWL Ontologies. In: *DL*. 2007.

[26] D. Calvanese, C. M. Keet, W. Nutt, M. Rodriguez-Muro, and G. Stefanoni. Web-based Graphical Querying of Databases Through an Ontology: the Wonder System. In: *SAC*. 2010.

[27] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks. OptiqueVQS: Towards an Ontology-based Visual Query System for Big Data. In: *MEDES*. 2013.

[28] N. Manolis and Y. Tzitzikas. Interactive Exploration of Fuzzy RDF Knowledge Bases. In: *ESWC (1)*. 2011.

[29] C. Veres, K. Johansen, and A. L. Opdahl. Browsing and Visualizing Semantically Enriched Information Resources. In: *CISIS*. 2010.

[30] H. Bast, F. Bäurle, B. Buchhold, and E. Haußmann. Easy Access to the Freebase Dataset. In: *WWW*. 2014.

[31] O. Suominen, K. Viljanen, and E. Hyvönen. User-Centric Faceted Search for Semantic Portals. In: *ESWC*. 2007.

[32] P. Haase, D. M. Herzig, M. A. Musen, and T. Tran. Semantic Wiki Search. In: *ESWC*. 2009.

[33] S. Buschbeck, A. Jameson, R. Troncy, H. Khrouf, O. Suominen, and A. Spirescu. A Demonstrator for Parallel Faceted Browsing. In: *EKAW*. 2012.

[34] H. Bast and B. Buchhold. An Index for Efficient Semantic Full-Text Search. In: *CIKM*. 2013.

[35] P. Heim, T. Ertl, and J. Ziegler. Facet Graphs: Complex Semantic Querying Made Easy. In: *ESWC*. 2010.

Enabling Faceted Search over OWL 2 with SemFacet*

Marcelo Arenas[1], Bernardo Cuenca Grau[2], Evgeny Kharlamov[2], Šarūnas Marciuška[2], and Dmitriy Zheleznyakov[2]

[1] Pontificia Universidad Católica de Chile; [2] University of Oxford

Abstract. Lots of applications nowadays rely on RDF, OWL 2, and SPARQL 1.1 for storing, publishing and querying data. However, SPARQL 1.1 is not targeted towards end-users, and suitable query interfaces are needed. Faceted search is a prominent approach to facilitate end-users query formulation which is widely used in information systems. This approach was recently adapted to the context of RDF, however, the proposed solutions lack rigorous theoretical underpinning and essentially ignore OWL 2 axioms. We develop a clean theoretical framework for faceted search that accounts for both RDF data and OWL 2 axioms. We implemented and tested some of our solutions in the SemFacet platform.

1 Introduction

In the last decade we have witnessed a constant increase in the number of applications that store and publish data in RDF, in the of use OWL 2 ontologies for providing background knowledge about the application domain and enriching query answers with information not explicitly given in the data, and in the use of SPARQL 1.1 for querying this data. It was acknowledged by many that writing SPARQL 1.1 queries requires special training and is not well-suited for the majority of end users. Thus, an important challenge is the development of simple yet powerful query-formulation interfaces that capture well-defined fragments of SPARQL 1.1.

This challenge was acknowledged by the community, and a number of approaches to facilitate query formulation over RDF and OWL 2 have been proposed in the last decade. Proposed solutions rely on different query formulation paradigms and include controlled natural language, e.g., [4–8], diagram based approaches, e.g., [9–15], faceted search interfaces,e.g., [16–18], and others. In this work we focus on faceted search due to its importance in Web bases information systems and intuitiveness to end users.

Faceted search is a prominent approach for querying document[1] collections where users can narrow down the search results by progressively applying filters, called *facets* [19]. A facet typically consists of a property (e.g., 'gender' or 'occupation' when querying documents about people) and a set of possible string values (e.g., 'female' or 'research'), and documents in the collection are annotated with property-value pairs. During faceted search users iteratively select facet values, and the documents annotated according to the selection are returned as the search result. Several authors have proposed faceted search for querying document collections annotated with RDF, and a number of RDF-based faceted search systems have been developed, e.g. [16–18, 20–26]. Although there has been intensive efforts in system development, the theoretical underpinnings received less attention [27–29].

* Work supported by the Royal Society, the EPSRC projects Score!, Exoda, and MaSI[3], and the FP7 project OPTIQUE [1–3] under the grant agreement 318338.

[1] We use the term 'document' to refer to any resource that can be referenced using a URI.

In particular, it is unclear what fragments of SPARQL 1.1 can be naturally captured using faceted search as a query paradigm, and what is the complexity of answering such queries. Moreover, faceted search interfaces are typically generated and updated based only on RDF data graphs. We see this as an important limitation as OWL 2 axioms are essentially ignored by the existing solutions, thus overlooking that ontological axioms can be used to enrich query answers with implicit information. Besides, these axioms provide schema-level structure that can be exploited to improve faceted interfaces. Finally, purely RDF-based faceted search systems are data-centric, and hence cannot be exploited to browse large ontologies or to pose meaningful queries at the schema level.

We address these limitations of existing solutions by formalising faceted interfaces that are tailored towards RDF and OWL 2, and which capture the key functionality implemented in existing faceted search systems. Our interfaces capture both the combination of facets displayed during search, and the facet values selected by users. In this way, an interface encodes both a query, whose answers constitute the current search results, and the facet values available for further selection. Analogously to existing work on RDF-based faceted search, and in contrast to traditional faceted search, our notion of interface allows users to 'navigate' across interconnected collections of documents and establish filters to each of them. Furthermore, it abstracts from considerations specific to GUI design (e.g., facet and value ranking), while at the same time reflecting the core functionality of existing systems. For faceted queries, i.e., queries that can be encoded by faceted interfaces, we study the expressivity and complexity of evaluation over RDF data enhanced with OWL 2 axioms. Finally, we study interface generation and update. Existing techniques for RDF are based on exploration of the underlying RDF graph: by generating facets according to the RDF graph, systems can guide users in the formulation of 'meaningful' queries. We lift this approach by proposing a special graph-based representation of OWL 2 ontologies and their logical entailments for the purpose of faceted navigation. Further details on our techniques can be found in [30].

To put our ideas in practice we developed a faceted search platform, called SemFacet (available for download and installation, and as a Web service [31, 32]), for generating and updating faceted interfaces. SemFacet relies on an external triple store with OWL 2 reasoning capabilities, and it is compatible with faceted search GUIs and text search engines for retrieving documents from keywords. For the demonstration purpose we bundled SemFacet with the triple store JRDFox, the search engine Lucene, and our own faceted search GUI. We have tested SemFacet over synthetic ontologies as well as Yago [33] extended with DBpedia (`dbpedia.org/`) abstracts with encouraging results.

2 Preliminaries

We use standard notions from first-order logic. We assume pairwise disjoint infinite sets of constants, unary predicates, and binary predicates. A *fact* is a ground atom and a *dataset* is a finite set of facts. A *rule* is a sentence of the form $\forall \mathbf{x} \forall \mathbf{z} \, [\varphi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \, \psi(\mathbf{x}, \mathbf{y})]$, where $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{y}$ are pairwise disjoint tuples of variables, $\varphi(\mathbf{x}, \mathbf{z})$ is a conjunction of atoms with variables in $\mathbf{x} \cup \mathbf{z}$, and $\exists \mathbf{y} \, \psi(\mathbf{x}, \mathbf{y})$ is an existentially quantified non-empty conjunction of atoms $\psi(\mathbf{x}, \mathbf{y})$ with variables in $\mathbf{x} \cup \mathbf{y}$. Universal quantifiers are omitted. The restriction of $\psi(\mathbf{x}, \mathbf{y})$ being non-empty ensures satisfiability of any set of rules and facts, which makes query results meaningful. A rule is *Datalog* if $\exists \mathbf{y} \, \psi(\mathbf{x}, \mathbf{y})$ has at most one atom and all variables are universally quantified. OWL 2 defines three *profiles*: RL, EL, and QL, weaker languages with favourable computational properties [34]. Every profile ontology can be normalised as a set of rules and

facts using the correspondence of OWL 2 axioms with first-order logic and a variant of the structural transformation (e.g., see [35]). Thus, we define an *ontology* $\mathcal{O}$ as a finite set of rules and facts. We refer to [30, 35] for the details of rules corresponding to the OWL 2 profiles. A *positive existential query* (PEQ) is a formula that may have free variables which is constructed using $\wedge$, $\vee$ and $\exists$. A PEQ s *monadic* if it has one free variable, and it is a *conjunctive query* (CQ) if it is $\vee$-free.

We consider two different semantics for query answering. Under the *classical semantics*, given a query $Q(\mathbf{x}) = \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$, we have that a tuple $\mathbf{t}$ of constants is an *answer* to $Q(\mathbf{x})$ w.r.t. an ontology $\mathcal{O}$ if $\mathcal{O} \models \exists \mathbf{y} \varphi(\mathbf{t}, \mathbf{y})$. Under the *active domain semantics*, $\mathbf{t}$ is an answer to Q w.r.t. $\mathcal{O}$ if there is a tuple $\mathbf{t}'$ of constants from $\mathcal{O}$ such that $\mathcal{O} \models \varphi(\mathbf{t}, \mathbf{t}')$. The evaluation problem under the classical (resp. active domain) semantics is to decide, given a tuple of constants $\mathbf{t}$, a PEQ Q and an ontology $\mathcal{O}$ in a language $\mathcal{L}$, whether $\mathbf{t}$ is an answer to Q w.r.t. $\mathcal{O}$ under the classical (resp. active domain) semantics. The classical semantics is the default in first-order logic, whereas active domain is the default semantics of the SPARQL 1.1 entailment regimes [36]. The difference between both semantics manifests itself only in the presence of existentially quantified rules and queries; thus, both semantics coincide if either the input ontology is Datalog, or if all variables in the input query are free.

3 Faceted Interfaces

Our notion of *faceted interface* comes with a clean first-order logic semantics, and provides a rigorous foundation for faceted search over RDF graphs enhanced with OWL 2 ontologies. We start with an example based on an excerpt of from Yago [33]. Our goal is to find presidents who graduated from St. Petersburg or Georgetown and have a child who graduated from some university.

Example 1. The document d_{vp} and d_{bc} for Vladimir Putin and Bill Clinton are annotated with the category 'president', and d_{vp} is also annotated with 'Russian'. Putin's daughter Yekaterina d_{yp} and Clinton's daughter Chelsea d_{cc} are categorised as 'person'. The documents for St. Petersburg State Uni. d_{sp}, Stanford Uni. d_s, and Georgetown Uni. d_g are categorised under 'university'. These annotations are given in RDF and correspond to the following facts:

$$\begin{array}{llll} \mathsf{President}(d_{vp}), & \mathsf{President}(d_{bc}), & \mathsf{Russian}(d_{vp}), & \mathsf{Person}(d_{yp}), \\ \mathsf{Person}(d_{cc}), & \mathsf{Univ}(d_{sp}), & \mathsf{Univ}(d_s), & \mathsf{Univ}(d_g). \end{array}$$

In RDF specific information about documents is represented using literals, e.g., Vladimir Putin's date of birth is encoded as $\mathsf{dateOfBirth}(d_{vp}, 1952\text{-}10\text{-}07)$. Most importantly, documents are also annotated with other documents:

$$\mathsf{child}(d_{vp}, d_{yp}),\ \mathsf{child}(d_{bc}, d_{cc}),\ \mathsf{grad}(d_{vp}, d_{sp}),\ \mathsf{grad}(d_{bc}, d_g),\ \mathsf{grad}(d_{cc}, d_s).$$

Moreover, Yago can be extended with ontological rules. Consider for example the following rule that says that at least one child of each Russian president graduate from some university, which is a reasonable assumption:

$$\mathsf{Russian}(x) \wedge \mathsf{President}(x) \wedge \mathsf{child}(x, y) \rightarrow \mathsf{child}(x, z) \wedge \mathsf{grad}(z, w) \wedge \mathsf{Univ}(w). \quad (1)$$

Analogously to traditional faceted search, we represent a *facet* as a pair of a predicate (or facet name) and a set of values. In the context of RDF, however, documents (and not just strings) can be used to annotate other documents, and thus annotations form a graph, rather than a tree. Consequently, facet values can be either document URIs or literals. Examples of facet names are the relations 'grad' and 'dateOfBirth', and example

values are documents such as 'd_s' (Stanford) and literals such as '1952-10-07'. Selection of multiple values within a facet can be interpreted conjunctively or disjunctively, and hence we distinguish between conjunctive and disjunctive facets. Furthermore, we distinguish a special facet type, whose values are categories (i.e., unary predicates) rather than specific documents or literals. Finally, a special value any denotes the set of all values compatible with the facet name.

Definition 1. *Let* type *and* any *be special symbols. A* facet *is a pair* $(X, \circ\Gamma)$*, with* $\circ \in \{\wedge, \vee\}$*,* Γ *a non-empty set, and either*

(i) $X =$ type *and* Γ *is a set of unary predicates, or*

(ii) X *is a binary predicate and* Γ *is a set that consists of* any *and either some constants or some unary predicates.*

A facet of the form $(X, \wedge\Gamma)$ *is* conjunctive*, and a facet of the form* $(X, \vee\Gamma)$ *is* disjunctive*. In a facet* $F = (X, \circ\Gamma)$*,* X *is the* facet name*, denoted by* $F|_1$*, and* Γ *contains the* facet values *and it is denoted by* $F|_2$*.*

Example 2. Consider the following facets: $F_1 = (\mathsf{type}, \vee\{\mathsf{President}, \mathsf{Univ}\})$, $F_2 = (\mathsf{child}, \vee\{\mathsf{any}, d_{yp}, d_{cc}\})$ and $F_3 = (\mathsf{grad}, \vee\{\mathsf{any}, d_{sp}, d_g, d_s\})$. The disjunctive facet F_1 can be exploited to select the categories to which the relevant documents belong (president or university). Facet F_2 can be used to narrow down search results to those individuals with children d_{yp} or d_{cc}; furthermore, the value any can be used to state that we are not looking for any specific child. F_3 allows to select graduation places. □

3.1 The Notion of Faceted Interface

Our faceted interfaces encode both a query (whose answers determine the search results) and the choices of facet values available for further refinement.

Definition 2. *A* basic faceted interface *(BFI) is a pair* (F, Σ)*, with* F *a facet and* $\Sigma \subseteq F|_2$ *the set of* selected values*. The set of* faceted interfaces *(or interfaces, for short) is given by the following grammar, where* I_0 *and* $I_1 = (F, \Sigma)$ *are BFIs and* $F|_1 \in \mathbf{BP}$*:*

$$I ::= \ \mathit{path} \mid (\mathit{path} \wedge \mathit{path}) \mid (\mathit{path} \vee \mathit{path}), \qquad \mathit{path} ::= \ I_0 \mid (I_1/I).$$

A BFI encodes user choices for a specific facet, e.g., the BFI $(F_1, \{\mathsf{President}\})$ selects the documents categorised as presidents. BFIs are put together in *paths*: sequences of nested facets that capture navigation between sets of documents. Documents are annotated with other documents by means of binary relations (e.g., child connects parents to their children); thus, nesting (I_1/I) requires the BFI I_1 to have a binary relation as facet name. With nesting we can capture queries such as 'people with a child who graduated from Stanford' by using the interface $(F_2, \{\mathsf{any}\})/(F_3, \{d_s\})$ which first selects people having (any) children and then those children with a Stanford degree. Finally, two types of branching can be applied: $(\mathit{path}_1 \wedge \mathit{path}_2)$ indicates that search results must satisfy the conditions specified by both path_1 and path_2, while $(\mathit{path}_1 \vee \mathit{path}_2)$ indicates that they must satisfy those in path_1 or path_2.

Example 3. Consider the interface I_{ex}:

$$\big((F_1, \{\mathsf{President}\}) \wedge (F_3, \{d_{sp}, d_g\})\big) \wedge \big((F_2, \{\mathsf{any}\})/(F_3, \{\mathsf{any}\})\big).$$

It could be visualises it in our system SemFacet as in Figure 1, left. The interface consists of three paths connected by $\wedge$-branching. In the first path, we select documents

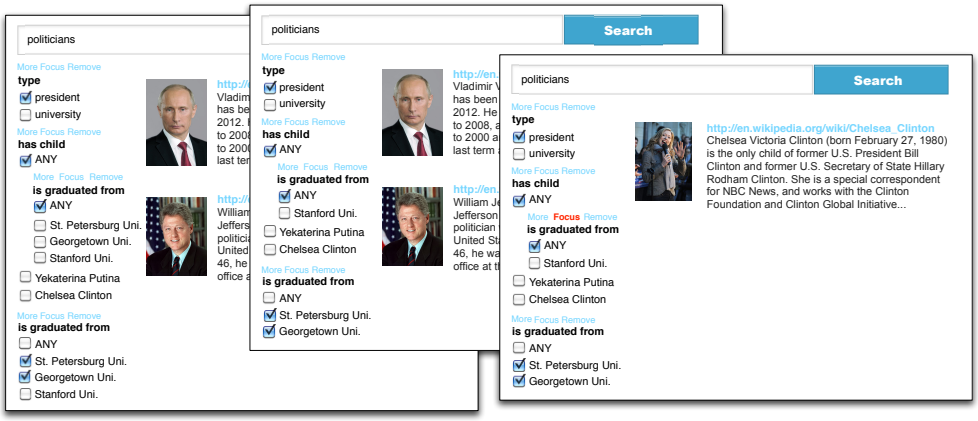


Fig. 1. Example interfaces implemented in SemFacet platform. Left: without minimisation; Center: after minimisation is applied; Right: minimised and refocussed.

categorised as 'president'. In the second path, we select graduates of St. Petersburg or Georgetown. In the third path, we first select individuals with a child who have some degree. Since paths are combined conjunctively, the constraints encoded in them apply simultaneously. Thus, we obtain the presidents who graduated from either St. Petersburg or Georgetown and who have a child graduated from some Uni. □

We formally specify the *semantics* of the queries encoded by the selected values in an interface in terms of first-order logic, see [30] for details. Intuitively, Our semantics assigns to each interface a PEQ with one free variable, which retrieves a set of documents. For each facet F we have that no restriction is imposed by F if no facet value is selected. BFIs with a type-facet are interpreted as the conjunction (disjunction) of unary atoms over the same variable. BFIs having as facet name a binary predicate result in either an atom whose second argument is existentially quantified (if any is selected), or in a conjunction (disjunction) of binary atoms having a variable as second argument that must be equal to a constant or belong to a unary predicate. Branching $(path_1 \circ path_2)$ with $\circ \in \{\wedge, \vee\}$ is interpreted in the natural way by constructing the conjunction (disjunction) of the queries for each $path_i$; furthermore, if for some $path_i$ we have that no value from the facets occurring in $path_i$ is selected, then $path_i$ is ignored. Finally, nesting involves a "shift" of variable from the parent BFI to the nested subexpression. A first order formula φ is a *faceted query* if there exists a faceted interface I such that φ is are identical modulo renaming of variables to the semantics of I

Example 4. Our example interface $I_{\mathtt{ex}}$ encodes the positive existential query $Q_{\mathtt{ex}}(x)$:

$$\mathsf{President}(x) \wedge \left(\exists y_1 \left(\mathsf{grad}(x, y_1) \wedge y_1 \approx d_{sp}\right) \vee \exists y_2 \left(\mathsf{grad}(x, y_2) \wedge y_2 \approx d_g\right)\right) \\ \wedge \exists z \left(\mathsf{child}(x, z) \wedge \mathsf{grad}(z, w)\right).$$

If we consider only facts, the answer is Bill, since we do not know whether Yekaterina graduated from some university. If we also consider Rule (1), then both Vladimir and Bill are answers, since it says that one of the Vladimir's children has a degree. □

Note that our notion of interface abstracts from several considerations that are critical to GUI design. For instance, our notion is insensitive to the order of BFIs composed by $\wedge$- or $\vee$- branching, as well as to the order of facet values (which are carefully

ranked in practice). Furthermore, we model type-facet values as 'flat', whereas in applications categories are organised hierarchically. Although these issues are important from a front-end perspective, they are immaterial to our technical results.

Minimisation of Faceted Interfaces. An important issue in the design of faceted interfaces is to avoid the overload of users with redundant facets or facet values. Intuitively, an (unselected) facet value v is redundant if selecting v either leads to a 'dead end' (i.e., an empty set of answers) or it does not have an effect on query answers. Then, a faceted interface is minimal if none of its component BFIs contains redundant values. In [30] we provide interface minimisation algorithms. In Figure 1, center, we present a minimised version of the interface in Figure 1, left.

Faceted Interfaces with Refocussing. The interface in Example 3 finds two presidents. If we want to know who their children who guaranteed that these presidents returned by the query, (i.e., see Chelsea Clinton as an answer), we must provide *refocussing* (or *pivoting*) functionality [25, 26] that allows to change the output variable of faceted queries. In [30] we provide and extension of faceted interfaces to support refocussing. In Figure 1, right, we have a screenshot of SemFacet where the refocusing is set to the children and Chelsea is returned by the system. Note that Yekaterina is not in the answer set since we do not know whether she is the child of Vladimir with a degree.

3.2 Answering Faceted Queries

Each time a user selects a facet value to refine the search results, a faceted search system must compute the answers to a query. Thus, query evaluation is a key reasoning problem affecting efficiency of such systems. As discussed above, faceted queries are monadic PEQs resulting from the selection of facet values in an interface. By standard results for relational databases, PEQ evaluation is an NP-hard problem, even for CQs over datasets. Our main result is that, in contrast to PEQs (and even CQs), faceted query evaluation over datasets is tractable, feasible in polynomial time; furthermore, the problem remains tractable in most cases if we consider ontologies belonging to the OWL 2 profiles. Our tractability results concern *combined complexity*, which takes into account the size of the entire input: ontological rules, RDF data and queries.

Active Domain Semantics. In practice, queries over ontology-enhanced RDF data are typically represented in SPARQL 1.1 and executed using off-the-shelf reasoning engines with SPARQL 1.1 support. The specification of SPARQL 1.1 under entailment regimes [36] is based on active domain semantics, which requires existentially quantified variables in the query Q to map to actual constants in the input ontology $\mathcal{O}$. In this case, we can answer queries by first computing the dataset $\mathcal{D}$ of all facts entailed by $\mathcal{O}$ and then answers Q w.r.t. the dataset $\mathcal{D}$. Fact entailment is tractable for all the OWL 2 profiles; thus, by committing to the active domain semantics of SPARQL 1.1 we can achieve tractability without emasculating the ontology language.

Theorem 1. *Active domain evaluation of faceted queries is in* PTIME *w.r.t. all normative OWL 2 profiles. Furthermore, it is* PTIME*-complete w.r.t. the EL and RL profiles.*

Classical Semantics. Classical and active domain semantics coincide if we restrict ourselves to Datalog ontologies. Thus, the above result holds for query answering under classical semantics if the input ontology is Datalog. An immediate consequence is that our results in Theorem 1 transfer to OWL 2 RL ontologies under classical semantics.

In contrast to RL, the EL and QL profiles can capture existentially quantified knowledge and hence active domain and classical semantics may diverge for queries with existentially quantified variables. To deal with EL, we exploit techniques developed for the *combined approach* to query answering [37, 38]. These techniques are currently applicable to so-called *guarded* EL ontologies. The idea is to rewrite (EL) rules into Datalog by Skolemising existential variables. Although this transformation strengthens the ontology, it preserves the entailment of facts [37]. We showed [30] that the evaluation of faceted queries is also preserved. Thus, we conclude tractability of faceted query evaluation for EL. In contrast, we can show that faceted query evaluation is NP-complete for QL under classical semantics. The following theorem summarises our results.

Theorem 2. *Faceted query evaluation under classical semantics is (i)* PTIME*-complete for RL and guarded EL ontologies; and (ii)* NP*-complete for QL ontologies.*

In [30] we showed that theorems above hold for faceted queries with refocussing.

4 Interface Generation and Update

Faceted navigation is an interactive process. Starting with an initial interface generated from a keyword search, users 'tick' or 'untick' facet values and the system reacts by updating both search results (query answers) and facets available for further navigation. We propose to generate and update interfaces by 'traversing' the (explicit and implicit) information in $\mathcal{O}$. Our approach is based on a general principle: each element of the initial interface (resp. each change in an interface as a response of an action) must be 'justified' by a suitable entailment in $\mathcal{O}$. In this way, by exploring the ontology, we can guide users in the formulation of meaningful queries.

There is an inherent degree of non-determinism in faceted navigation: if a user selects a facet value, it is unclear whether the next facet generated by the system should be conjunctive or disjunctive, and whether it should be incorporated in the interface by means of conjunctive or disjunctive branching. In applications, however, different values in a facet are typically interpreted disjunctively, whereas constraints imposed by different facets are interpreted conjunctively. In [30] we resolve this ambiguities and devise fully deterministic algorithms for a restricted class of interfaces where conjunctive facets and disjunctive branching are disallowed. Example of such an interface is in Example 3 and in the screenshots in Figure 1.

The Ontology Facet Graph. We capture facets relevant to an ontology $\mathcal{O}$ in what we call a *facet graph*. The graph can be seen as a concise representation of $\mathcal{O}$, and our interface generation and update algorithms are parameterised by such graph rather than $\mathcal{O}$. The nodes of a facet graph are possible facet values (unary predicates and constants), and edges are labelled with possible facet names (binary predicates and type). The key property of a facet graph is that every X-labelled edge (v, w) is justified by a rule or fact entailed by $\mathcal{O}$ which 'semantically relates' v to w via X. We distinguish three kinds of semantic relations: *existential*, where X is a binary predicate and (each instance of) v must be X-related to (an instance of) w in the models of $\mathcal{O}$; *universal*, where (each instance of) v is X-related only to (instances of) w in the models of $\mathcal{O}$; and *typing* where $X = \mathsf{type}$, and (the constant) v is entailed to be an instance of (the unary predicate) w.

Definition 3. *A* facet graph *for $\mathcal{O}$ is a directed labelled multigraph G having as nodes unary predicates or constants from $\mathcal{O}$ and s.t. each edge is labelled with a binary predicate from $\mathcal{O}$ or* type*, where the latter labels only edges from a constant to a unary*

predicate. Each edge e is justified by a fact or rule α_e s.t. $\mathcal{O} \models \alpha_e$ 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 $R(c,y) \rightarrow y \approx d$;

(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 $R(c,y) \rightarrow A(y)$;

(iii) if e is $A \xrightarrow{R} c$, then α_e is a rule of either of the form $A(x) \rightarrow R(x,c)$ or $A(x) \wedge R(x,y) \rightarrow y \approx c$;

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

(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)-(iv)* encodes the existential (resp. universal) R-relation between nodes in e, whereas *(v)* encodes typing. A graph may not contain all justifiable edges, but rather those that needed for a given application.

Example 5. A facet graph for the ontology in Example 1 may contain nodes for d_{bc} and d_{cc}, as well as for predicates such as President and Univ. Example edges are: a child-edge linking d_{bc} to d_{cc}, which is justified by the fact $\mathsf{child}(d_{bc}, d_{cc})$; and a grad-edge from d_{vp} to Univ due to Rule (1) in Example 1 and the facts about d_{vp}. □

It follows from the following proposition that facet graph computation can be efficiently implemented. In practice, the graph can be precomputed when first loading data and ontology, stored in RDF, and accessed using SPARQL 1.1 queries. In this way, reasoning tasks associated to faceted search are performed offline.

Proposition 1. *Checking whether a directed labelled multigraph is a facet graph for $\mathcal{O}$ is feasible in polynomial time if $\mathcal{O}$ is in any of the OWL 2 profiles.*

To realise the idea of ontology-guided faceted navigation, we require that any faceted interface over an ontology $\mathcal{O}$ *conforms* to the facet graph of $\mathcal{O}$, in the sense that the presence of every facet and value in the interface is supported by edges of the graph. In this way, we can ensure that interfaces mimic the structure of (and implicit information in) the ontology and, hence, that the interface does not contain irrelevant (combinations of) facets. Since a given facet or facet value can occur in many different places in an interface, we need a mechanism that allows us to refer to the elements of an interface in an unambiguous way. We refer to [30] for a formal definition of conformance. In [30] we also provide algorithms that by relying on facet graphs allow to create faceted interfaces (conformant to the graph), e.g., from sets of keywords, and to update such interfaces as a reaction on users actions, i.e., 'ticking' and 'unticking' operations on facet values.

5 SemFacet Platform

We have developed a faceted search platform SemFacet providing the following main functionality: *(i)* computation of facet graphs from an ontology; *(ii)* interface generation from facet graphs; and *(iii)* interface update in response to user actions. Our platform relies on an external triple store for querying and OWL reasoning.

We bundled the platform in a prototypical system which powers faceted search over a fragment of Yago RDF data [33] enriched with DBPedia abstracts and OWL 2 RL Our system is available for downloading and installation, as well as a Web service [31]. In the remainder of this section, we will describe the system's workflow and architecture and present the results of SemFacet evaluation over both Yago and synthetic data.

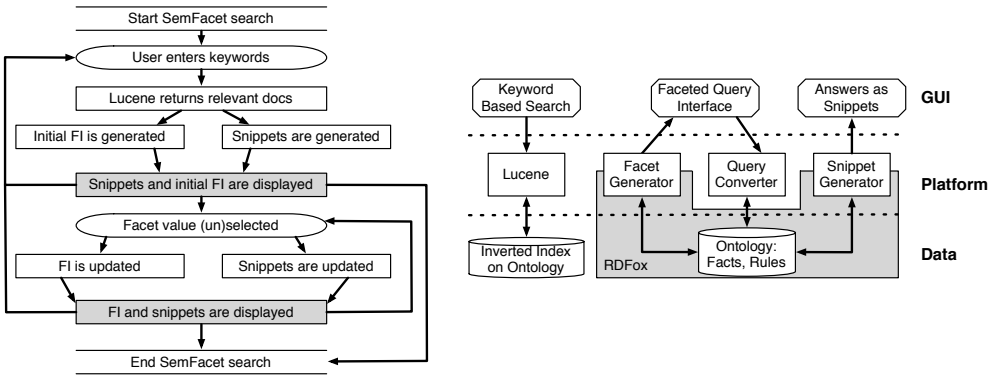


Fig. 2. Left: workflow of SemFacet, Right: architecture of SemFacet

5.1 System Architecture and Implementation

If Figure 2, left, we present the workflow diagram for SemFacet. The steps relevant to users' activity are depicted in ovals, and those relevant to SemFacet's activity are depicted as boxes. Grey boxes represent front-end activity and the remaining ones represent back-end tasks. Users start their interaction with SemFacet by entering a set of keywords, and the system returns as initial answers those documents that are relevant to the keywords; e.g., in Yago we considered as relevant those document URIs whose abstract contains at least one of the keywords. SemFacet displays the answers as snippets, combining the relevant URIs with the corresponding images and abstracts. The initial faceted interface is generated from the relevant URIs. Users can then perform faceted navigation by applying actions on the interface. SemFacet responds to each action by first updating the interface and then recomputing the query answers. Observe that in Figure 1 we nest facets to visualise the / operator and users can perform refocusing by clicking on the button 'focus' attached to facets.

The architecture of SemFacet is in Figure 2, right, where the components are arranged in three layers. SemFacet relies on RDFox [39][2] for storing RDF triples, and computing answers for SPARQL 1.1 queries. SemFacet converts faceted interfaces in SPARQL 1.1 queries and executes them using RDFox. Note that RDFox supports only conjunctive queries; thus, to answer faceted queries, we extended its query module with a support of `UNION`. To support keyword search, we load DBpedia abstracts in Lucene (`lucene.apache.org/`). Note that the implementation of SemFacet allows to substitute both Lucene an RDFox with any other software that provide the same functionality.

5.2 Experiments

We evaluated only the back-end of SemFacet (c.f. Figure 2), left. We excluded front-end activities since they highly depend on the client machine and network connection. In our experiments we evaluated the runtime performance to compute answer URIs and their associated snippet, as well as to update the interface. Our system presents ten snippets at time, thus, snippet generation time is negligible. Since we use Lucene as a black box, Lucene-related experiments are out of the scope of this paper.

Experimental Setup To evaluate the runtime performance of SemFacet, we used a MacBook Pro laptop with OS X 10.8.5, 2.4 GHz Intel Core i5 processor, and 8GB 1333

[2] RDFox is a parallel in-memory RDF triple store; it implements reasoning for OWL 2 RL; it materialises all implicit data via forward chaining and evaluates CQs over the materialisation.

# of queries	Avg time
10	0.0001 s
100	0.0181 s
1000	0.1449 s
10,000	1.1676 s
100,000	30.8467 s

(a) RDFox query time performance

# of facets	Avg time
1	0.0015 s
10	0.0122 s
100	0.0537 s
1000	0.3724 s
10,000	3.2964 s

(b) Interface construction run time

# of values	Avg time
1	0.005 s
10	0.0263 s
100	0.1054 s
1000	0.4573 s
10,000	3.3762 s

(c) Interface construction run time

# of values	Avg time
10	0.0027 s
100	0.0317 s
1000	0.1869 s
10,000	1.3101 s
100,000	22.6759 s

(d) Facet value hiding time

# of answers	# of facets	total # of values	Avg time
10	10	90	0.4918 s
100	15	727	0.5581 s
1000	21	6538	0.8616 s
10,000	28	51317	3.5472 s

(e) SemFacet time on the real data: Yago and DBpedia

Table 1. Experiment results for SemFacet

MHz DDR3 memory. The laptop runs the Apache Tomcat 7.0, with 4 GB of allocated memory. Each experiment presented in the following section was executed over 10 different runs to avoid cashing of objects. Each run was repeated 10 times in a FOR-loop to obtain a bigger sample size. Therefore, each experiment, having different parameters, was executed 100 times it total. We report average and median of running time performance results for each experiment.

Generation of Initial Interface SemFacet computes the initial interface by *(i)* gathering relevant facet names and values, and *(ii)* combining these in an interface.To perform Step (i), SemFacet sends to RDFox one atomic query for each relevant URI. Thus, to evaluate Step (i), we sent from 10 to 100,000 atomic queries to RDFox and measured response time. For this, we generated atomic queries and synthetic data containing exactly one answer per query Note that due to the RDFox indexing, evaluation time of atomic queries is constant regardless data size. The results are presented in Table 1a. Times are promising for up to 1,000 atomic queries (0.14 s), which corresponds to a keyword search that returns 1000 URIs; for $10,000$ queries, average runtime increases to $1.16s$ and for $100,000$ queries to $30.8s$. Thus, one may need to restrict the number of URIs returned by Lucene to guarantee reasonable response time for interface construction; in our online version of SemFacet, the output of Lucene is restricted to 10,000 URIs.

To evaluate Step (ii), we conducted two sets of experiments. In the first experiment we fixed the number of facets to 10 and increased the number of values in each facet from 1 to 10,000. In the second, we fixed the number of values per facet to 10 and increased the number of facets from 1 to 10,000. Results are presented in Tables 1b and 1c. Generation times for up to 1000 facets having 10 values each is promising (0.37 s). Generation times for 10,000 facets having 10 values each is increased to 3.3s on average. Moreover, the interface construction time is similar for n facets with m values each or for m facets with n values each. Note that the time in Tables 1b and 1c grows linearly.

Faceted Interface Update The experiments required to evaluate update of faceted interfaces are the same as those described for the generation of the initial interface [30]. Hence, we focus here on interface minimisation. We have evaluated our minimisation algorithm by generating faceted interfaces of growing sizes from 10 to 100,000 total facet values. Since RDFox performs reasoning by first materialising implicit data (a step performed only once) and then answering queries over the materialisation, we did not use an ontology in our experiments, and relied on synthetic RDF data. Results are

presented in Table 1d: the time to minimise 1000 values is promising (0.18s), and grows linearly upto 100,000 values (22.6s).

Experiments with Real Data Previous experiments were conducted to evaluate particular components of SemFacet in isolation. To evaluate the overall performance, we tested the system's running time on a fragment of Yago data enriched with DBpedia (`dbpedia.org/`) abstracts and OWL 2 RL axioms (around 5 million RDF triples in total). To mimic a real world scenario, we selected keywords that return from 10 to 10,000 answers. We measured the number of facets, the total number of values, and the response time of the system. The results of the experiments are presented in Table 1d: the time to obtain 1000 answers and to generate the corresponding interface is promising (0.86 s), and grows to 3.5 s for 10,000 answers. About one third of this time is spent by Lucene, one third by RDFox, and one third by the interface construction component of SemFacet.

6 Conclusion and Future Work

We presented a solid theoretical foundations for faceted search in the context of RDF and OWL. We have analysed the expressive power and complexity of queries stemming from faceted interfaces, and developed practical faceted navigation algorithms. Our results suggest many interesting problems for future work. In particular, we will explore extensions of our navigation algorithms beyond simple interfaces, as well as the design of more scalable interface minimisation algorithms. Concerning system design, substantial work is needed to improve GUI design, in particular with respect to advanced features such as refocusing. Finally, we are planning to conduct further experiments with knowledge bases other than Yago and perform user studies.

7 References

[1] E. Kharlamov, M. Giese, E. Jiménez-Ruiz, M. G. Skjæveland, A. Soylu, et al. Optique 1.0: Semantic Access to Big Data: The Case of Norwegian Petroleum Directorate's FactPages. In: *ISWC (Posters & Demos)*. 2013.

[2] E. Kharlamov, E. Jiménez-Ruiz, D. Zheleznyakov, D. Bilidas, M. Giese, et al. Optique: Towards OBDA Systems for Industry. In: *ESWC (Satellite Events)*. 2013.

[3] E. Kharlamov, N. Solomakhina, O. Ozcep, D. Zheleznyakov, T. Hubauer, et al. How Semantic Technologies can Enhance Data Access at Siemens Energy. In: *ISWC*. 2014.

[4] E. Kaufmann and A. Bernstein. Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases. In: *J. Web Sem.* 8.4 (2010).

[5] A. Bernstein, E. Kaufmann, A. Göhring, and C. Kiefer. Querying Ontologies: A Controlled English Interface for End-Users. In: *ISWC*. 2005.

[6] E. Franconi, P. Guagliardo, M. Trevisan, and S. Tessaris. Quelo: an Ontology-Driven Query Interface. In: *DL*. 2011.

[7] U. Waltinger, D. Tecuci, M. Olteanu, V. Mocanu, and S. Sullivan. Natural Language Access to Enterprise Data. In: *AI Magazine* 35.1 (2014).

[8] *Protege*. `http://protege.stanford.edu`.

[9] A. Soylu, M. G. Skjæveland, M. Giese, I. Horrocks, E. Jiménez-Ruiz, et al. A Preliminary Approach on Ontology-Based Visual Query Formulation for Big Data. In: *MTSR*. 2013.

[10] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks. OptiqueVQS: towards an ontology-based visual query system for big data. In: *MEDES*. 2013.

[11] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The SEWASIE Network of Mediator Agents for Semantic Search. In: 13.12 (Dec. 1, 2007). `http://www.jucs.org/jucs_13_12/the_sewasie_network_of`.

[12] A. Fadhil and V. Haarslev. OntoVQL: A Graphical Query Language for OWL Ontologies. In: *DL*. 2007.
[13] *iSPARQL QBE*. http://dbpedia.org/isparql/.
[14] D. Calvanese, C. M. Keet, W. Nutt, M. Rodriguez-Muro, and G. Stefanoni. Web-based graphical querying of databases through an ontology: the Wonder system. In: *SAC*. 2010.
[15] A. Russell and P. R. Smart. NITELIGHT: A Graphical Editor for SPARQL Queries. In: *ISWC (Posters & Demos)*. 2008.
[16] T. Berners-Lee, J. Hollenbach, K. Lu, J. Presbrey, E. Prudhommeaux, and M. M. C. Schraefel. Tabulator Redux: Browsing and Writing Linked Data. In: *LDOW*. 2008.
[17] P. Fafalios and Y. Tzitzikas. X-ENS: Semantic Enrichment of Web Search Results at Real-Time. In: *SIGIR*. 2013.
[18] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, et al. Faceted Wikipedia Search. In: *BIS*. 2010.
[19] D. Tunkelang. *Faceted Search*. Morgan & Claypool Publishers, 2009.
[20] E. Hyvönen, S. Saarela, and K. Viljanen. Ontogator: Combining View- and Ontology-Based Search with Semantic Browsing. In: *XML Finland*. 2003.
[21] m.c. schraefel, D. A. Smith, A. Owens, A. Russell, C. Harris, and M. L. Wilson. The Evolving mSpace Platform: Leveraging the Semantic Web on the Trail of the Memex. In: *Hypertext*. 2005.
[22] P. Heim, J. Ziegler, and S. Lohmann. gFacet: A Browser for the Web of Data. In: *IMC-SSW*. Vol. 417. 2008.
[23] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. In: *ISWC*. 2006.
[24] D. Huynh, S. Mazzocchi, and D. R. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In: *J. Web Sem.* 5.1 (2007).
[25] G. Kobilarov and I. Dickinson. Humboldt: Exploring Linked Data. In: *LDOW*. 2008.
[26] D. F. Huynh and D. R. Karger. Parallax and Companion: Set-based Browsing for the Data Web. 2013.
[27] E. Oren, R. Delbru, and S. Decker. Extending Faceted Navigation for RDF Data. In: *ISWC*. 2006.
[28] S. Ferré and A. Hermann. Semantic Search: Reconciling Expressive Querying and Exploratory Search. In: *ISWC*. 2011.
[29] A. Wagner, G. Ladwig, and T. Tran. Browsing-oriented Semantic Faceted Search. In: *DEXA*. 2011.
[30] M. Arenas, B. C. Grau, E. Kharlamov, S. Marciuska, and D. Zheleznyakov. Faceted Search over Ontology-Enhanced RDF Data. In: *CIKM*. 2014.
[31] *SemFacet*. http://www.cs.ox.ac.uk/isg/tools/SemFacet/.
[32] B. C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, M. Arenas, and E. Jimenez-Ruiz. SemFacet: Semantic Faceted Search over Yago. In: *WWW Demo*. 2014.
[33] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In: *WWW*. 2007.
[34] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles. In: *W3C Recommendation* (2009).
[35] B. Motik, R. Shearer, and I. Horrocks. Hypertableau Reasoning for Description Logics. In: *J. Artif. Int. Res.* (2009).
[36] *W3C: SPARQL 1.1 Entailment Regimes*. www.w3.org/TR/sparql11-entailment/.
[37] G. Stefanoni, B. Motik, and I. Horrocks. Introducing Nominals to the Combined Query Answering Approaches for EL. In: *AAAI*. 2013.
[38] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The Combined Approach to Ontology-Based Data Access. In: *IJCAI*. 2011.
[39] *RDFox*. www.cs.ox.ac.uk/isg/tools/RDFox/.

On Faceted Search over Knowledge Bases (Abstract)

Bernardo Cuenca Grau[1], Evgeny Kharlamov[1], Dmitriy Zheleznyakov[1], Marcelo Arenas[2], and Šarūnas Marciuška[1]

[1] University of Oxford; [2] Pontificia Universidad Católica de Chile

Motivation An increasing number of applications are relying on RDF [2] and SPARQL [3] for storing, publishing, and querying semistructured data. The functionality of many such applications is enhanced with OWL 2 ontologies [1], which are used to provide a conceptual layer on top of data and enrich query answers with implicit information.

Although the growing popularity of RDF and OWL 2 and the fact that SPARQL has been accompanied by the development of better and better query answering engines, it is widely recognised that writing SPARQL queries requires specialised skills and training, and hence is not well-suited for the majority of users. Thus, an important challenge is the development of simple yet powerful query interfaces, which can capture well-defined and efficiently implementable fragments of SPARQL.

Faceted search is a prominent approach for accessing document collections that allows users to narrow down search results by incrementally applying filters, called *facets*, on the annotations associated to documents [12]. Faceted search has become a mainstream commercial technology, and it is ubiquitous in e-commerce websites. For example, hotel booking websites such as *Booking.com* allow users to refine search results by selecting suitable values in facets such as 'Price', 'Star Rating', or 'Facilities'.

Thus, adapting faceted search to the context of RDF and OWL seems to be a promising direction for the development of simple yet powerful query interfaces for SPARQL. However, this adaptation poses new research challenges. In contrast to data models underlying faceted search applications, RDF is a graph data model that allows documents to be annotated with other documents, which could in turn have other annotations attached. Furthermore, the domain knowledge represented in the application ontology should influence both the results of the search and the interaction with the user during the search process.

Faceted search has been proposed as a suitable paradigm for querying document collections annotated with RDF, and several RDF-based faceted search systems have been developed [4, 7–9]. Existing approaches are, however, rather systems-oriented, and there is a lack of rigorous theoretical underpinnings. Our goal is to provide such solid foundations. We have formalised faceted interfaces tailored towards graph-based data models, and identified a fragment of first-order logic capturing the underlying queries. We have studied the complexity of answering such queries for RDF and ontologies expressed in the OWL 2 profiles. Moreover, we have devised practical and generic algorithms for recomputing faceted interfaces in response to user actions. Finally, we have implemented and tested our faceted search algorithms in a prototype system [6], with encouraging results. In this extended abstract, we provide a short overview of the main ideas underlying our approach.

Technical Approach To illustrate our definitions and make our discussion concrete, we consider an excerpt of Yago [11] about US presidents, which is depicted in Figure 1. In this figure, d_{th} and d_{bi} stand for T. Roosevelt and B. Clinton, which are annotated as US presidents with 'USpres'. Moreover, Theodore's son Kermit d_{ke} and Bill's daughter Chelsea d_{ch} are categorised as 'person'. Finally, Stanford d_s, Harvard d_h, and Georgetown d_g are annotated with 'university', and the USA d_{usa} with 'country', which, in turn, is annotated with its 'founding date' 1783-09-03. Our goal in this example is to find US presidents who graduated from either Harvard or Georgetown and have a child who graduated from Stanford.

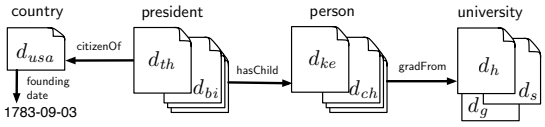


Fig. 1. Annotated entities

We model facets as pairs consisting of a predicate (or facet name) and a set of values (typically documents represented by URIs or literals). Examples of facet names are the binary relations 'gradFrom' and 'dateOfBirth', and examples of values for these facets are specific documents such as 'd_s' (Stanford) and literals such as '1858-10-27'. Selection of multiple values within a facet can be interpreted either conjunctively or disjunctively, and hence we distinguish between conjunctive and disjunctive facets. Furthermore, we distinguish a special facet type, whose values are categories (i.e., unary predicates) rather than specific documents or literals. Finally, we also allow for a special value any which denotes the set of all values compatible with the facet predicate. We assume that $\mathbf{C}$, $\mathbf{UP}$ and $\mathbf{BP}$ are pairwise disjoint infinite sets of *constants*, *unary predicates* and *binary predicates*.

Definition 1. *Let* type *and* any *be symbols that do not occur in* $\mathbf{C} \cup \mathbf{UP} \cup \mathbf{BP}$. *A* facet *is a pair of the form* $(X, \circ\Gamma)$*, where* $\circ \in \{\wedge, \vee\}$*,* Γ *is a non-empty set, and either (i)* $X = \mathsf{type}$ *and* $\Gamma \subseteq \mathbf{UP}$*, or (ii)* $X \in \mathbf{BP}$ *and* $\Gamma \subseteq \mathbf{C} \cup \{\mathsf{any}\}$*. A facet is called* conjunctive *if* $\circ \in \{\wedge\}$*, and it is called* disjunctive *otherwise. Finally,* X *is called the* facet name *and denoted by* $F|_1$ *and* Γ *contains the* facet values *and is denoted by* $F|_2$.

The following facets could be of use when searching over these documents.

$$F_1 = (\mathsf{type}, \vee\{\mathsf{USpres}, \mathsf{president}, \mathsf{person}, \mathsf{country}, \mathsf{university}\}),$$
$$F_2 = (\mathsf{hasChild}, \vee\{\mathsf{any}, d_{ke}, d_{ch}\}), \quad F_3 = (\mathsf{gradFrom}, \vee\{\mathsf{any}, d_h, d_s, d_g\}),$$
$$F_4 = (\mathsf{citizenOf}, \wedge\{d_{usa}, d_{uk}\}), \quad F_5 = (\mathsf{citizenOf}, \vee\{d_{usa}, d_{uk}\}).$$

F_1 can be exploited to restrict types of entities, F_2 to narrow down search results to entities with children, where 'any' can be used to say that we are not looking for a specific child, F_3 to select an alma mater, and F_4 with F_5 to select a citizenship.

Faceted Interfaces. A faceted interface represents an arrangement of facets that can be displayed for users, and captures the choices of facet values made by them. Thus, it encodes both a query, whose answers constitute the current search results, and the possible choices of facet values available to users for further refinement. In contrast to

traditional faceted search, our notion of interface allows the user to 'navigate' across interconnected sets of documents and establish independent filters to each of them.

Definition 2. *A* basic faceted interface *(BFI) is a pair* (F, Σ)*, with* F *a facet and* $\Sigma \subseteq F|_2$ *a set of* selected values*. The set of* faceted interfaces *(FIs) is defined by the following grammar, where* I_0 *and* $I_1 = (F, \Sigma)$ *are BFIs with* $F|_1 \in \mathbf{BP}$*:*

$$I ::= \ \mathit{path} \mid (\mathit{path} \wedge \mathit{path}) \mid (\mathit{path} \vee \mathit{path}); \qquad \mathit{path} ::= \ I_0 \mid (I_1/I).$$

A BFI encodes user choices for a particular facet, which introduce specific constraints to search results as well as the values available for further selection. BFIs are put together in paths (*path*), where each path is a sequence of facets that have to be applied according to the sequence order and intuitively correspond to navigation between different sets of documents. Two types of branching can be applied in interfaces: $(\mathit{path}_1 \wedge \mathit{path}_2)$ indicates that each of the search results has to satisfy the sequence of conditions specified by both path_1 and path_2, while $(\mathit{path}_1 \vee \mathit{path}_2)$ indicates that each of these results must satisfy the conditions specified by path_1 or path_2. Note that our notion of interface abstracts from considerations that are specific to GUI design (such as ranking and arrangement of facets), while at the same time reflecting the core functionality available in both traditional and RDF-based faceted search systems.

The following faceted interface I_{ex} has three paths connected by $\wedge$-branching and encodes the query 'return US presidents who graduated from Harvard or Georgetown and who have a child who graduated from Stanford'. For instance, the first path in I_{ex} selects entities categorised as USpres.

$$\big((F_1, \{\mathsf{USpres}\}) \wedge (F_3, \{d_h, d_g\})\big) \wedge \big((F_2, \{\mathsf{any}\})/(F_3, \{d_s\})\big).$$

Faceted Queries. Queries encoded in faceted interfaces can be captured by formulae in the positive existential fragment of first-order logic with one free variable, where in every disjunction $\varphi_1 \vee \varphi_2$, the subformulae φ_1 and φ_2 share at most one variable. Moreover, these queries involve predicates of arity at most two and are tree shaped. The output variable of a faceted query is the root variable in the query graph. We also considered extensions of faceted interfaces that allow us to choose different output variables (a functionality typically referred to as *refocusing* [5]). The query encoded by I_{ex} returns two presidents: Roosevelt and Clinton. We investigated the query evaluation problem for faceted queries for ontologies expressed in the OWL 2 profiles [10]. We showed that the problem is PTIME-complete for RL and EL ontologies, NP-complete for QL, and PTIME-complete for the core fragment of QL. We also investigated the complexity under the active domain semantics, which is commonly used for SPARQL query evaluation, and showed that it is PTIME-complete for all the profiles.

Faceted Navigation with Faceted Queries. Users interact with the a faceted search system, i.e., do faceted navigation, by selecting or unselecting a facet value, and the system responses by updating the interface and answers. We formally captured this interaction and proposed a practical algorithm that updates the interface in response to users' actions, while taking into account both the data and ontology. Our algorithm is based on the principle that each individual change in the interface made by the system in response to an action must be 'justified' by a suitable entailment from the ontology and data. Besides, we provide an extension of the algorithm that minimises the amount of irrelevant facet values that are available to users for further navigation.

References

1. W3C: OWL 2 Web Ontology Language. http://www.w3.org/TR/owl2-overview/
2. W3C: Resource Description Framework (RDF). http://www.w3.org/RDF/
3. W3C: SPARQL 1.1 Query Language. www.w3.org/TR/sparql11-query/
4. Buschbeck, S., Jameson, A., Troncy, R., Khrouf, H., Suominen, O., Spirescu, A.: A Demonstrator for Parallel Faceted Browsing. In: EKAW (2012)
5. Clarkson, E., Navathe, S.B., Foley, J.D.: Generalized Formal Models for Faceted User Interfaces. In: JCDL. pp. 125–134 (2009)
6. Grau, B.C., Kharlamov, E., Marciuska, S., Zheleznyakov, D., Arenas, M., Jimenez-Ruiz, E.: SemFacet: Semantic Faceted Search over Yago. In: WWW Demo (2014)
7. Huynh, D.F.: The Nested Faceted Browser. people.csail.mit.edu/dfhuynh/projects/nfb/ (2013)
8. Huynh, D.F., Karger, D.R.: Parallax and Companion: Set-based Browsing for the Data Web. www.davidhuynh.net (2013)
9. Kobilarov, G., Dickinson, I.: Humboldt: Exploring Linked Data. In: LDOW (2008)
10. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language Profiles. W3C Recommendation (2009)
11. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge. In: WWW. pp. 697–706 (2007)
12. Tunkelang, D.: Faceted Search. Synthesis Lectures on Information Concepts, Retrieval, and Services, Morgan & Claypool Publishers (2009)

Faceted Search over OWL 2 Life Science Datasets and Ontologies with SemFacet*

Bernardo Cuenca Grau, Evgeny Kharlamov, Šarūnas Marciuška,
Dmitriy Zheleznyakov, and Yujiao Zhou

Department of Computer Science, University of Oxford
first.middle.lastname@cs.ox.ac.uk

1 Introduction

In the last decade numerous RDF datasets and OWL ontologies in the life sciences domain have become available [1–3]. Accessing the required information, however, remains a challenging task for end users and often requires proficiency in SPARQL. In order to make data and ontological knowledge more human accessible numerous query formulation, data exploration, and browsing tools have been developed. Many such interfaces have beed tailored for specific life science datasets [3, 4]. More generic systems typically rely on controlled natural language, [5, 6] diagrammatic query constructors [7, 8], or exploratory search [9].

Faceted search is the de facto query paradigm in e-commerce applications [10]. A facet typically consists of a property (e.g., 'gender' or 'occupation' when querying documents about people) and a set of possible string values (e.g., 'female' or 'research'), and documents in the collection are annotated with property-value pairs. During faceted search, users iteratively select facet values and the documents annotated according to the selection are returned as the search result.

Several authors have proposed faceted search for querying RDF, and a number of systems have been developed [11–15]. Existing systems, however, have been designed for plain RDF data, and do not take into account ontological axioms other than subsumption statements between atomic classes and properties [16, 17], with reasoning playing little or no role. In stark constrast to other domains, life sciences applications tend to require a great deal of the expressive power available in OWL 2; in particular, data often involves complex class or property assertions (e.g., see FlyBase [3]) and ontologies largely consist of complex axioms which encapsulate highly valuable information for faceted search. As a result, existing faceted search systems are not well-suited for typical life sciences applications.

In [18] we developed a faceted search approach for RDF data enhanced with OWL 2 ontologies. Our solution is based on a solid theoretical framework and it addresses many of the limitations of existing techniques. To put our ideas into practice we developed SemFacet [18, 19]: a faceted search system that relies on state-of-the-art triple stores and OWL 2 reasoners to generate and update faceted query interfaces, as well as for computing search results. For demonstration purposes our platform integrates

* Work supported by the Royal Society, the EPSRC projects Score!, Exoda, and MaSI^3, and the FP7 project OPTIQUE under the grant agreement 318338.

JRDFox [20] and Sesame [21] as RDF triple stores, as well as PAGOdA [22] and HermiT [23] as fully-fledged OWL 2 reasoners. Our system is fully generic and can be used to query arbitrary data and ontologies. In this demonstration we will show how SemFacet can be used to access several datasets and ontologies from the domain of life sciences and illustrate the main advantages of our approach over existing techniques designed for plain RDF.

2 The SemFacet System

SemFacet [24] combines keyword search and faceted navigation to query arbitrary ontology-enhanced RDF datasets. Our system offers the following main functionality.

- *Keyword search*. Seach in SemFacet typically starts with a set of keywords, which are matched against the annotations in the ontology and data.
- *Faceted interface generation and update*. SemFacet implements dedicated infrastructure for automatically generating a faceted interface from the result of a keyword search as well as for updating an interface in response to users' actions. A distinguishing aspect of our algorithms for interface generation and update is that they are 'guided' by both explicit and implicit information in the ontology and data (see [18] for details).
- *Query answering*. User selections of facet values in an interface are compiled into SPARQL queries, which are then evaluated against the ontology and data using a reasoner. Our system allows for both disjunctive facets (i.e., those where multiple value selections are interpreted disjunctively) and conjunctive facets. Thus, the SPARQL graph patters relevant to our approach can be captured by the AND-UNION fragment of SPARQL 1.1. The current version of SemFacet integrates the following reasoners: Sesame [21] (a widely used system for RDF(S) reasoning), JRDFox [20] (a parallel in-memory RDF triple store supporting sound and complete reasoning for OWL 2 RL), HermiT [23] (a standard fully-fledged OWL 2 reasoner), and PAGOdA [22] (a pay-as-you-go reasoner for OWL 2 that combines JRDFox and HermiT for increased efficiency).
- *Refocusing*. SemFacet provides functionality for changing the focus of the search from one type of object to another. For instance, if the system is displaying as search results neurons that develop from cells, where "develops from" is a facet name and "cell" is a facet value, we can refocus the search and display as search results the particular cells that are related to the selected neurons.
- *Customisation*. Our system is generic and highly customisable for different datasets and applications. Users can upload arbitrary ontologies and datasets, select the reasoner to be exploited for faceted navigation and query answering, customise the kinds of annotations relevant for keyword search, select which facets should be interpreted disjunctively or conjunctively as well as which facets should be excluded from the search process, or select what properties are relevant for image thumbnails and snippets (if any).

On the left-hand-side of Figure 1 we can see a screenshot of SemFacet with a search over the Adult Brain Anatomy dataset [1]. The *navigation map* in the interface enables

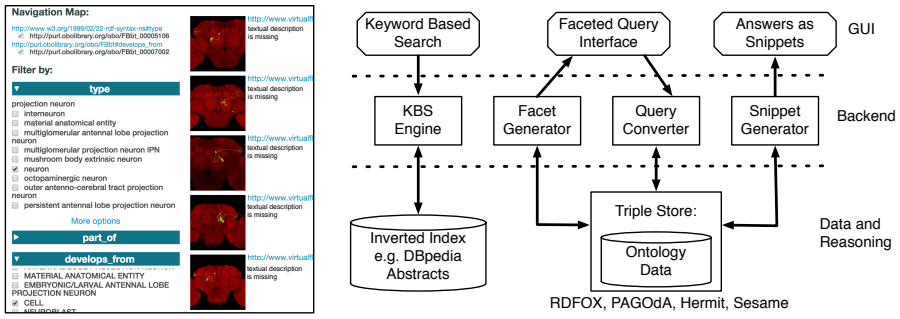


Fig. 1. Left: screenshot of SemFacet over FlyBase OWL 2 data, Right: architecture of SemFacet

refocusing, the *filter by* section displays the relevant facet names and values, and search results (i.e., query answers) are displayed on the rightmost part of the interface. The general architecture of SemFacet including its main software components is summarised on the right-hand-side of Figure 1.

3 Demonstration Scenarios

During the demonstration we will show how to explore and query OWL 2 life science datasets and ontologies with SemFacet. To this end, we will preconfigure the system for several test cases, including fragments of FlyBase [3], SNOMED CT [2], as well as a selection of Bio2RDF [1] datasets. In all cases the input for the search will be a dataset and an ontology. We will demonstrate the following variants of our algorithms for interface generation and update.

- *Data driven*, where only the data is exploited for interface generation and update. This configuration simulates existing approaches to faceted search over RDF.
- *Ontology driven*, where only the axioms in the ontology are considered. In this configuration, facet names and values in an interface reflect semantic relationships between entities in the input ontology.
- *Both data and ontology driven*, where both the data and ontology are exploited in interface generation and update. This is the default configuation of SemFacet, and the aim here is to show how reasoning and ontologies can improve data driven faceted interfaces and allow for enhanced data exploration.

Besides querying preconfigured scenarios, the demo attendees will be able to try SemFacet end-to-end. This would require to load a data set and ontology, to customise the system parameters, and to query the uploaded ontology and data with the selected parameters. For the end-to-end test of SemFacet the demo attendees will be able to use datasets and ontologies either from the preconfigured scenarios or the ones they provide (of reasonable size), e.g., by downloading them from the Web.

4 References

[1] F. Belleau, M. Nolin, N. Tourigny, et al. Bio2RDF: Towards a mashup to build bioinformatics knowledge systems. In: *Journal of Biomedical Informatics* 41.5 (2008).

[2] *SNOMED CT*. www.ihtsdo.org/snomed-ct.
[3] *FLyBase*. http://flybase.org/.
[4] N. Milyaev, D. Osumi-Sutherland, S. Reeve, et al. The Virtual Fly Brain browser and query interface. In: *Bioinformatics* 28.3 (2012).
[5] E. Franconi, P. Guagliardo, M. Trevisan, and S. Tessaris. Quelo: an Ontology-Driven Query Interface. In: *DL*. 2011.
[6] A. Bernstein, E. Kaufmann, A. Göhring, and C. Kiefer. Querying Ontologies: A Controlled English Interface for End-Users. In: *ISWC*. 2005.
[7] D. Calvanese, C. M. Keet, W. Nutt, et al. Web-based graphical querying of databases through an ontology: the Wonder system. In: *SAC*. 2010.
[8] A. Soylu, M. G. Skjæveland, M. Giese, et al. A Preliminary Approach on Ontology-Based Visual Query Formulation for Big Data. In: *MTSR*. 2013.
[9] S. Ferré and A. Hermann. Semantic Search: Reconciling Expressive Querying and Exploratory Search. In: *ISWC*. 2011.
[10] D. Tunkelang. *Faceted Search*. Morgan & Claypool Publishers, 2009.
[11] P. Fafalios and Y. Tzitzikas. X-ENS: Semantic Enrichment of Web Search Results at Real-Time. In: *SIGIR*. 2013.
[12] R. Hahn, C. Bizer, C. Sahnwaldt, et al. Faceted Wikipedia Search. In: *BIS*. 2010.
[13] D. F. Huynh and D. R. Karger. Parallax and Companion: Set-based Browsing for the Data Web. 2013.
[14] P. Heim, J. Ziegler, and S. Lohmann. gFacet: A Browser for the Web of Data. In: *IMC-SSW*. 2008.
[15] G. Kobilarov and I. Dickinson. Humboldt: Exploring Linked Data. In: *LDOW*. 2008.
[16] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. In: *ISWC*. 2006.
[17] E. Oren, R. Delbru, and S. Decker. Extending Faceted Navigation for RDF Data. In: *ISWC*. 2006.
[18] M. Arenas, B. C. Grau, E. Kharlamov, et al. Faceted Search over Ontology-Enhanced RDF Data. In: *CIKM*. 2014.
[19] M. Arenas, B. C. Grau, E. Kharlamov, et al. SemFacet: semantic faceted search over yago. In: *WWW, Companion Volume*. 2014.
[20] *RDFox*. www.cs.ox.ac.uk/isg/tools/RDFox/.
[21] *Sesame*. http://www.openrdf.org/.
[22] Y. Zhou, Y. Nenov, B. C. Grau, and I. Horrocks. Complete Query Answering over Horn Ontologies Using a Triple Store. In: *ISWC*. 2013.
[23] B. Glimm, I. Horrocks, B. Motik, et al. HermiT: An OWL 2 Reasoner. In: *Journal of Automated Reasoning* 53.3 (2014).
[24] *SemFacet*. http://www.cs.ox.ac.uk/isg/tools/SemFacet/.

SemFacet: Semantic Faceted Search over Yago

Bernardo Cuenca Grau
Evgeny Kharlamov
Dmitriy Zheleznyakov
Ernesto Jimenez-Ruiz
University of Oxford
fname.mname.lname@cs.ox.ac.uk

Šarūnas Marciuška
Faculty of Computer Science
Free Univ. of Bozen-Bolzano
marciuska@inf.unibz.it

Marcelo Arenas
Dept. of Computer Science
PUC Chile
marenas@ing.puc.cl

1. INTRODUCTION

Faceted search is a technique for accessing document collections that combines text search and *faceted navigation* applied to the documents' metadata. With faceted navigation, users can narrow down search results by incrementally applying multiple filters, called *facets* [18]. Although faceted search has become a mainstream commercial technology, traditional models impose rather severe constraints in the way faceted metadata is represented, and queries are formulated.

In particular, conventional faceted search models assume that documents are not "linked" to each other. E.g., in travel websites like *TripAdvisor.com* one can perform independent faceted search for hotels and then for restaurants, but there are no facets that link specific hotel and restaurant documents. As a result, values of facets for different kinds of documents cannot be joined in a single query; for example, we cannot rely on *TripAdvisor*'s query interface to formulate a query that returns hotels that have restaurants serving local specialties: even if we can query for both hotels or restaurants independently, when we "switch view" from hotels to restaurants, the constraints imposed on hotels are lost. Data sparsity is also an important issue in faceted search applications; in particular, document annotations are intrinsically incomplete, which leads to both expected missing answers and facets. For example, in *TripAdvisor* one can find 5-star hotels that do not feature a restaurant; however, no hotel can be rated as 5-star without a restaurant on site; as a result, when users select the "restaurant" box in the "amenities" facet, some relevant hotels are filtered out. There are also issues concerning the meaning of queries: in a faceted search front-end, users are presented with facets and are allowed to tick multiple choices for each facet. Choices within a single facet are understood as either a logical OR or AND. Ambiguity is resolved in the back-end when queries are translated into operations over inverted indices. This process is ad-hoc and application dependent, and it is not grounded on a formal query model that can be independently studied.

In this paper we demonstrate SemFacet, a proof of concept prototype implementing faceted search enhanced with Semantic Web technologies. RDF provides the required flexibility to semantically link documents in arbitrary ways, thus overcoming the limitations of conventional metadata models for faceted search; furthermore, RDF is powerful enough to capture existing metadata models for faceted search, and existing metadata can be easily transformed into RDF triples. OWL 2 can be used to provide rich domain knowledge on top of faceted metadata, which can not only be exploited to capture complex dependencies between facets in a declarative and semantically unambiguous way, but also to provide a schema-level solution to the data sparsity problem. Finally, faceted queries can be captured by SPARQL 1.1, which provides not only well-understood semantics and computational properties, but also powerful application-independent infrastructure for query processing.

Our system SemFacet allows Web developers to automatically generate faceted query interfaces over any application, provided that its metadata and domain knowledge are given in RDF and OWL 2, respectively. For the initial keyword based search SemFacet exploits Lucene [1]. Then, SemFacet implements algorithms for automatically generating facet names and their values from RDF and OWL documents, as well as for determining which facets are relevant at any point during faceted navigation. Since RDF triples can be exploited to link different types of documents, SemFacet also provides functionality to refocus the search from one type of document to another (e.g. from hotels to restaurants and vice-versa), without losing the constraints imposed thus far by means of facets. User choices in facet values are automatically translated into SPARQL 1.1 queries, which are then executed using the OWL 2 RL triple store RDFox [2].

Although our platform is generic, for this demo we have used the Yago [21] knowledge base extended with DBpedia [3] abstracts. Further details on semantic faceted search are in our technical report [19]; SemFacet [4] and demo video [17] are available online.

Related Work. RDF has been proposed by many as a promising metadata model for faceted search [5, 20, 6, 7, 18] and several faceted search systems based on RDF have been developed [7, 20, 8]. Prominent examples are Parallax [7] and Humboldt [20]. Parallax provides faceted navigation on top of Freebase [9] and it supports a limited form of refocusing (it is possible to switch view between different sets of entities while keeping faceted constraints; however, when switching back to entities visited before, some constraints are lost); it also does not support branching in queries. Humboldt [20] provides similar functionality to Parallax. Other RDF-based systems provide more limited functionality and in particular they do not support refocusing; these include mSpace [10], /facet [11], Piggy Bank [12], FacetedDBLP [13], a faceted DBpedia browser [14], BrowseRDF [5], Ontogrator [15], Tabulator [6], parallel faceted browser [8], and many others. Query model underlying all these system is unclear; furthermore, their focus is on RDF while OWL 2 reasoning plays little or no role.

2. SEMANTIC FACETED SEARCH

In this section, we introduce the basics of semantic faceted search. First, we describe a metadata model that provides the required flexibility for documents to annotate other documents. This metadata model is described in an abstract way, and it is not particularly tied to specific semantic technologies; however, it can be trivially realised using RDF. Then, we describe a query language that provides the necessary features to enable faceted navigation across different types of documents. This query language is also independent from exising query languages for semantic technologies; however, it has a clean embedding into SPARQL 1.1 (see [4] for details).

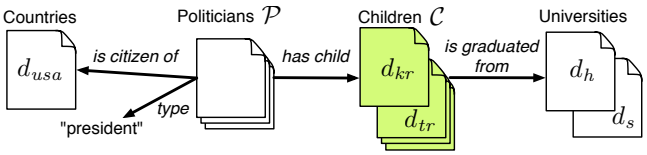


Figure 1: Example of semantic annotation

Semantic Faceted Annotations. We next formalise an extension of conventional faceted data models where documents are annotated not only with facet values (typically strings), but also with other documents, which in turn can have their own annotations.

In the remainder of this section, we assume that V_N, V_L, V_D are pairwise disjoint sets of (facet) names, labels, and documents, respectively, and that $V = V_N \cup V_L \cup V_D$. We will refer to elements of V_N as *facet-names* and of $V_L \cup V_D$ as *facet-values*.

DEFINITION 1. *The* semantic annotation *of a document $d \in V_D$ is a subset of $V_N \times (V_L \cup V_D)$. Moreover, d is said to be* n-annotated with x, *where $n \in V_N$ and $x \in V_L \cup V_D$, if (n, x) belongs to the semantic annotation of d.*

Note that in terms of this definition, the annotation of a document with conventional facets is simply a subset of $V_N \times V_L$; that is, documents cannot be used to annotate other documents.

EXAMPLE 2. *Assume that V_D consists of (i) documents $\mathcal{P}$ about politicians; (ii) documents $\mathcal{C}$ about children of politicians, including a document d_{trj} about Theodore Roosevelt Jr. and d_{kr} about Kermit Roosevelt; (iii) a set of (documents about) countries, including d_{usa} about USA; and (iv) a set of (documents about) universities, including d_h for Harvard University and d_s for Stanford University. Moreover, assume that V_L includes the string* president. *As illustrated in Figure 1, the documents of $\mathcal{P}$ are* type*-annotated with values from V_L (one of which is* president*),* has_child*-annotated with document from $\mathcal{C}$, and* is_citizen_of*-annotated with documents about countries. Finally, the documents of $\mathcal{C}$ are* is_graduated_from*-annotated with university documents.*

Faceted Queries. A conventional faceted query can be seen as a filter over a set of documents; it specifies what facet names should be used to annotate a target document and what faceted values should be used for each such name. The answer for such a query is the subset of the given documents satisfying the filter. In the semantic faceted search approach, we go further: a semantic faceted query can relate sets $S_1, \ldots, S_n$ of documents with annotations by describing relations between these sets, and setting separate filters over each one of them. Analogously to conventional faceted queries, the effect of these filters is to select subsets $S'_1 \subseteq S_1, \ldots, S'_n \subseteq S_n$. In contrast to a conventional faceted query, we can now "navigate" between different types of documents and select those documents $S'_i \subseteq S_i$ to be returned as output of the query.

In what follows, we formalise the notion of faceted query, starting with the notion of basic faceted query.

DEFINITION 3. *A* basic conjunctive faceted query (BCFQ) *over V is a pair $F = (X, \wedge\Gamma)$, where $X \in V_N$ and $\Gamma \subseteq V_L \cup V_D$. A* basic disjunctive facet query (BDFQ) *over V is a pair $F = (X, \vee\Gamma)$, where X and Γ satisfy the same conditions as in BCFQs. A* basic faceted query (BFQ) *over V is either a BCFQ or BDFQ over V.*

Intuitively, a document satisfies a BCFQ $(X, \wedge\Gamma)$ if it is X-annotated with *every* value in Γ, and it satisfies a BDFQ $(X, \vee\Gamma)$, if it is X-annotated with *some* value in Γ. We do not give the semantics formally (see [4] for details), but rather illustrate BFQs on an example.

EXAMPLE 4. *The following are BCFQs about politicians:*

$$Q^1_{\mathcal{P}} = (\mathsf{type}, \wedge\{\mathsf{president}\}), \quad Q^3_{\mathcal{P}} = (\mathsf{has\_child}, \wedge\{\}),$$
$$Q^2_{\mathcal{P}} = (\mathsf{is\_citizen\_of}, \wedge\{d_{usa}\}).$$

The query $Q^1_{\mathcal{P}}$ asks for (all the documents about) presidents, $Q^2_{\mathcal{P}}$ for politicians with American citizenship, $Q^3_{\mathcal{P}}$ for politicians with children. The empty condition $\wedge\{\}$ is used in $Q^2_{\mathcal{P}}$ since we impose no restrictions on the children. Moreover, the BDFQ $Q_{\mathcal{C}} = (\mathsf{is\_graduated\_from}, \vee\{d_h, d_s\})$, over the children of politicians in $\mathcal{C}$ asks for documents about children of politicians who graduated from either Harvard, i.e., d_h or Stanford,i.e., d_s.

We now define how to combine BFQs in complex queries. Faceted search is typically initiated by a keyword search that returns the initial set of documents over which a user can set filters. To model keyword-based search, we introduce a special set KW disjoint from V, and assume that every element $S \in$ KW stands for a set of documents retrieved by a specific keyword-based search.

DEFINITION 5. *A* query expression *over V is defined by the following grammar:*

$$\begin{array}{rlrl} \mathit{start} & ::= S \mid \star(S), & \mathit{expr} & ::= \mathit{start} \mid \mathit{start}/\mathit{rest}, \\ \mathit{step} & ::= F \mid \star(F), & \mathit{path} & ::= \mathit{step} \mid \mathit{step}/\mathit{rest}, \end{array}$$
$$\mathit{rest} ::= \mathit{path} \mid (\mathit{path} \wedge \mathit{path}) \mid (\mathit{path} \vee \mathit{path}),$$

where $S \in$ KW and F is a BFQ over V. A (semantic) faceted query (FQ) over V *is a query expression over V where the symbol $\star$ occurs exactly once.*

The starting point (*start*) of a query expression *expr* captures the content of the initial search used in typical interfaces (prior to faceted navigation), which is given by a keyword search. In each step (*step*) of the remainder of the query expression (*rest*), a BFQ is applied to narrow down the search results. These steps are put together in paths (*path*), where each path is a sequence of BFQs that have to be applied according to the sequence order. Two different types of branching can be applied in a query expression: $(\mathit{path}_1 \wedge \mathit{path}_2)$ indicates that each of the search results has to satisfy the sequence of conditions specified by both path_1 and path_2, while $(\mathit{path}_1 \vee \mathit{path}_2)$ indicates that each of these results has to satisfy the sequence of conditions specified by path_1 or path_2. Finally, the symbol $\star$ in a query expression indicates which documents must be returned. In a faceted query this symbol is mentioned exactly once, as the result of such query is one set of documents. We illustrate the syntax and semantics of FQs in the following example.

EXAMPLE 6. *Consider the following information request about the scenario introduced in Examples 2 and 4:*

> *Find children of US presidents who graduated from Harvard University or Stanford University.*[1]

It corresponds to the following FQ:

$$S_{\mathit{politician}}/\Big(Q^1_{\mathcal{P}} \wedge Q^2_{\mathcal{P}} \wedge (\star(Q^3_{\mathcal{P}})/Q_{\mathcal{C}})\Big), \tag{1}$$

Intuitively, the query first retrieves all documents of $S_{\mathit{politician}} \in$ KW, i.e., all documents returned by a keyword search with politicians. Then these results are filtered with the queries $Q^1_{\mathcal{P}}$ and $Q^2_{\mathcal{P}}$, obtaining the set of documents about American presidents. Finally, this set is filtered by using the query $\star(Q^3_{\mathcal{P}})/Q_{\mathcal{C}}$, returning the set of documents about children of American presidents who graduated from Harvard or Stanford. Notice that $Q_{\mathcal{C}}$ could be replaced by $(Q^s_{\mathcal{C}} \vee Q^h_{\mathcal{C}})$, where $Q^u_{\mathcal{C}} = (\mathsf{is\_graduated\_from}, \wedge\{d_u\})$ for $u \in \{h, s\}$.

[1] A similar query was used to illustrate the Parallax system in [7].

Figure 2: Semantic Faceted Search Interface

Semantic Faceted Search using RDF and SPARQL. Semantic annotations can be realised using different technologies. Our demo implementation exploits RDF, which we see as the most natural choice: URIs can be seen as documents of V_D, object and data properties as facet-names of V_N, literals as labels of V_L. Moreover, annotations are encoded in RDF as triples: each triple $(s,\ p,\ o)$ corresponds to a p-annotation of a document s with a facet-value o. As we will see in Sections 3 and 4, RDF gives more than that: by combining RDF with RDFS and OWL 2, one can use reasoning to address natural limitations of current faceted search.

We now discuss how to embed our faceted query language in SPARQL, the standard language for querying RDF [16]. In [19] we provided semantics to our faceted query language by translation into first-order logic. It follows from this translation that any FQ corresponds to a filter-free SPARQL query Q that: *(i)* is positive (neither `OPTIONAL` nor `NOT EXISTS` nor `MINUS` occurs in Q); *(ii)* is tree-shaped (the dependency graph of Q's variables, where $?x$ and $?y$ are connected if there is a triple $?x\ Z\ ?y$ in Q, is tree-shaped); and *(iii)* satisfies the condition that different triples in Q cannot share more than one variable. Thus, FQs over annotated documents represented as RDF can be translated in SPARQL and evaluated using state of the art SPARQL engines. Moreover, as we discuss in Sections 3 and 4, the restricted shape of the generated SPARQL queries allows to do reasoning about queries w.r.t. ontologies efficiently.

EXAMPLE 7. *A SPARQL query, where* wiki *is a name-space for Wikipedia, corresponding to Equation* (1) *is:*

```
SELECT ?z
 WHERE {?x rdf:type "president"@en ;
        ?x is_citizen_of wiki:United_States ;
        ?x has_child ?z ;
        {?z is_graduated_from wiki:Harvard_Uni }; UNION
        {?z is_graduated_from wiki:Stanford_Uni } ; }.
```

3. FRONT-END

We next discuss how to provide query formulation support to end-users, and describe our implementation of a query formulation interface in the SemFacet system.

3.1 Semantic Faceted Search Interface

In a conventional faceted search interface, users are presented with facet-values organised in groups according to facet-names; these groups are typically referred to as simply *facets*, and we will follow this terminology. Users can exploit facets to refine the search by selecting values, and value choices are converted into queries over the underlying documents. Answers to queries are typically displayed in the form of snippets combining images, textual descriptions, links, etc.

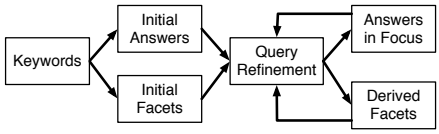


Figure 3: workflow of semantic faceted search

Facets in SemFacet. Our implementation relies on conventional facets to support BFQs; complex queries are formed by *nesting* facets in a hierarchical fashion (see Figure 2 for a screenshot of SemFacet's interface).

EXAMPLE 8. *Fig. 2 shows how the query in Eq.* (1) *can be composed using the facets computed by* SemFacet*. In the screenshot there are 4 facets with facet-names: (i)* rdftype *where "*president*" is selected, (ii)* has_child *where "*ANY*" is selected (it is a special facet-value meaning "the value is not important" and corresponding to the empty condition $\wedge\{\}$ in Ex. 4), (iii)* is_graduated_from *where both "*Harvard_Uni*" and "*Stanford_Uni*" are selected, and (iv)* is_citizen_of *where "*United_States*" is selected.*

Focusing in SemFacet. Our interface supports the $\star(\,)$ operator, which can be used to select the kinds of output documents in faceted queries. To this end, we allow users to *focus* on facets: if the focus is set on a facet, then SemFacet outputs documents corresponding the facet-values chosen by the user in this facet.

EXAMPLE 9. *In Figure 2 the focus of the query is set on a facet with the name* has_child*; thus, the system's output consists only of documents in $\mathcal{C}$; more precisely, the system outputs all children of US presidents who graduated from Harvard or Stanford. On the right hand side of Figure 2, we can see the answers to the query, namely Theodore and Kermit Roosevelt and Allan Hoover. Note that the interface also has* more *and* remove *buttons, where the former one allows to expand the list of possible facet-values, and the latter one allows to hide a facet when it is not needed.*

Query Language of SemFacet. Our interface currently supports only a fragment of the query language specified in Defs 3 and 5. First, SemFacet only supports BDFQs as basic queries, i.e., different values within the same facet are interpreted disjunctively. Second, our interface does not support $\vee$ in query expressions (i.e., values in different facets are always interpreted conjunctively).

EXAMPLE 10. *Fig. 2 captures the FQ in Eq.* (1)*. Indeed, Harvard and Stanford are values within the facet* is_graduated_from*, which corresponds to the BDFQ $Q_{\mathcal{C}}$; in contrast, the choices of "*president*" in* rdftype *and* United_States *in* is_citizen_of *correspond to the conjunction $Q^1_{\mathcal{P}} \wedge Q^2_{\mathcal{P}}$.*

Workflow of SemFacet. The process of constructing a semantic FQ in SemFacet is summarised in Fig. 3. The first step is to provide a set of keywords (e.g., in Fig. 2 the keyword "politicians" was used), which leads to a set of initial answers and initial facets. Query refinement is then an iterative process, where users can either choose available facet-values, or refocus the query to a different facet. In response the system updates the query answers as well as the facets available to continue query refinement.

3.2 Improving the Interface using Axioms

Deriving New Annotations. Data sparsity is an important challenge for faceted search interfaces. Document annotations are intrinsically incomplete, which leads to both missing expected answers and facets. This problem is addressed in SemFacet by exploiting OWL 2 axioms to enrich RDF data with implicit triples.

EXAMPLE 11. *Assume that RDF data contains triples stating that Kermit Roosevelt is a Harvard alumni, but it does not say that he graduated from Harvard. Thus, if one queries the data for Harvard graduates, Kermit Roosevelt would not be returned as an answer. The relationship between Harvard graduates and alumni can be modelled at the schema level using the following OWL 2 axiom:*

$$\mathit{SubClassOf}(\mathsf{alumHarvard}\ \ \mathit{ObjectHasValue}(\mathsf{is\_graduated\_from}\ \ \mathsf{Harvard\_Uni})).$$

Together with the triple (Kermit_Roosevelt rdftype alumHarvard)*, this axiom entails the triple*

(Kermit_Roosevelt is_graduated_from Harvard_Uni).

Another benefit of axioms is that they can naturally model *hierarchical* facets (e.g., documents about hotels can be annotated with facet-values *B&B*, or *hostel*, which are both kinds of *accommodation*). In traditional faceted search applications each document needs to be annotated with all values in a path from the root of the hierarchy to the specific value of interest (e.g., when annotating a hotel document with *B&B*, one would also need to annotate with *accommodation*). Modelling such ISA relationships between facet-values as OWL 2 axioms has the advantage that only the most specific values are required in annotations since the remaining ones can be derived using a reasoner.

Showing Relevant Information. Another important challenge for faceted search interfaces is to avoid "dead ends" (i.e., facet-value selections that lead to queries with empty answer). In conventional faceted search applications, the detection of such dead ends is data driven, in the sense that the interface does not display facet-values for which no document exists. Axioms provide an alternative, declarative, way to detect dead ends during faceted search. E.g., in the presence of the axiom

$$\mathit{DisjointClasses}(\mathsf{USPresident}\ \mathsf{FrenchPresident})$$

once the facet-value USPresident is selected, the interface should not display the value FrenchPresident. Axioms are particularly important when annotations are inconsistent (this can happen with automatically generated annotations). SemFacet uses both axiom- and data-driven techniques to avoid dead ends during faceted search.

4. BACK-END

SemFacet is available as a Web service [4] and runs on a machine with 1vCPU, 4Gb of memory, and 20Gb of disk space. SemFacet is implemented on top of a fragment of Yago [21] and DBpedia [3] abstracts; it contains around 15 million RDF triples extended with OWL 2 axioms. A general architecture of SemFacet is in Figure 4. SemFacet's back-end relies on RDFox [2] for storing RDF triples, performing reasoning, and answering queries. RDFox is a massively parallel in-memory RDF triple store; it implements reasoning for the OWL 2 RL profile. RDFox supports only a conjunctive fragment of SPARQL 1.1, thus, to answer faceted queries, we extended its query module with a support of `UNION`.

In short, the back-end's workflow is the following. First, DBpedia abstracts are loaded to Lucene and RDFox; Yago is loaded in RDFox only. Every initial user's input via keyword based search is executed over Lucene that returns initial relevant document IDs, and a *view of IDs* is created. Based on the view of IDs the initial facets and answer snippets compound of abstracts, thumbnails, and links to Wikipedia are generated. Then, the user performs query refinement and refocusing as described in Section 3 in the workflow of SemFacet. Faceted queries Q are executed by RDFox and the projections of results on the Q's first variable are intersected with the view of IDs. We rely on Lucene's ranking function to display

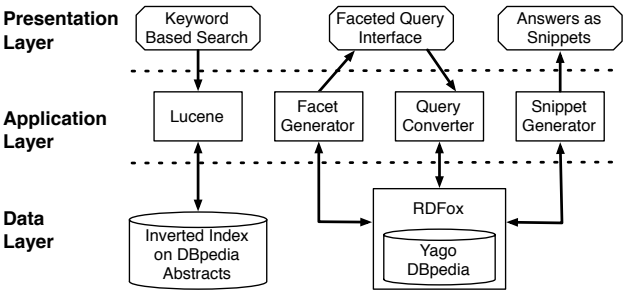


Figure 4: General architecture of SemFacet

answers under the default query focus; if the user refocuses the query, then we display answers in the order they are returned by RDFox. Note that SemFacet is implemented in such a way that both Lucene and RDFox can be substituted with any other software that provide the same functionality.

5. DEMO SCENARIO

During the demo we will show how one can find relevant information available in Wikipedia using semantic faceted search. The search will be performed over Yago using SemFacet system. The users will see several search scenarios that are hard to accomplish using conventional search engines, and we will show how they can be handle using SemFacet. Moreover, the users will be invited to try to express their own information needs over SemFacet.

6. CONCLUSION

We demonstrated semantic faceted search over Yago. Our approach is flexible and versatile: the same backend implementation can be used to power faceted search over any application based on RDF and OWL 2. Our system is still an early prototype. Further work includes development of ranking functions for both facets and answers, extension of the interface to support wider fragments of our query language, experiments with larger data sets, and work on improving systems scalability and concurrency of users' access.

7. REFERENCES

[1] `http://lucene.apache.org/`.
[2] `http://www.cs.ox.ac.uk/isg/tools/RDFox/`.
[3] `http://dbpedia.org/`.
[4] SemFacet. `http://tinyurl.com/SemFacet`.
[5] `https://launchpad.net/browserdf`.
[6] `http://www.w3.org/2005/ajar/tab`.
[7] `http://sparallax.deri.ie/`.
[8] `http://eventmap-ui.appspot.com/`.
[9] `http://www.freebase.com/`.
[10] `http://mspace.fm/`.
[11] `http://slashfacet.semanticweb.org/`.
[12] `http://simile.mit.edu/wiki/Piggy_Bank`.
[13] `http://dblp.l3s.de/`.
[14] `http://dbpedia.org/FacetedSearch`.
[15] `http://www.ontogrator.org/`.
[16] `http://www.w3.org/TR/rdf-sparql-query/`.
[17] Demo Video. `http://tinyurl.com/www14demo`.
[18] D.Tunkelang. *Faceted Search*. Morgan & Claypool Pubs.'09.
[19] M. Arenas et al. Semantic faceted search: Foundations, algorithms, implementation. `www.tinyurl.com/qazsvle`.
[20] G. Kobilarov and I. Dickinson. Humboldt: Exploring linked data. In *LDOW'08*.
[21] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *WWW 2007*.

Towards Semantic Faceted Search

Marcelo Arenas[†]
Dept. of Computer Science
PUC Chile

Bernardo Cuenca Grau[‡]
Dept. of Computer Science
University of Oxford

Evgeny Kharlamov[‡]
Dept. of Computer Science
University of Oxford

Šarūnas Marciuška[‡]
Dept. of Computer Science
University of Oxford

Dmitriy Zheleznyakov[‡]
Dept. of Computer Science
University of Oxford

ABSTRACT

In this paper we present limitations of conventional faceted search in the way data, facets, and queries are modelled. We discuss how these limitations can be addressed with Semantic Web technologies such as RDF, OWL 2, and SPARQL 1.1. We also present a system, SemFacet, that is a proof-of-concept prototype of our approach implemented on top of Yago knowledge base, powered by the OWL 2 RL triple store RDFox, and the full text search engine Lucene.

1. MOTIVATION AND PROPOSAL

Faceted search is a technique for accessing document collections that combines text search and *faceted navigation* applied to the documents' metadata. With faceted navigation, users can narrow down search results by incrementally applying multiple filters called *facets* [6]. During the last decade, faceted search has become a mainstream commercial technology, and it is ubiquitous in e-commerce websites and online libraries. Despite the numerous success stories, however, traditional faceted search models impose severe constraints in the way *(i)* faceted metadata is represented, *(ii)* facets are defined, and *(iii)* queries are formulated [4, 14]. Pushing the boundaries of faceted search beyond the current state-of-the-art requires addressing several challenges, which we discuss next. To make the discussion concrete, suppose we are looking in a travel website such as *TripAdvisor* for accommodation in Seoul to attend the WWW 2014 conference. We look for a 4-star or 5-star hotel with a Korean or Japanese vegetarian restaurant.

Limitations of the data model. Classical faceted search models assume that documents are not "linked" to each other. We can start our search in *TripAdvisor* by filling in an initial form to obtain all available hotel documents in Seoul during the conference dates. The search can then be further refined by using the facets "hotel class" and "amenities" to select 4-star or 5-star hotels with restaurants. To complete our query, we need additional constraints about restaurant documents; however, the relevant facets are associated to restaurants, and not to hotels. Thus, we switch to the interface for restaurants, where we can use the available facets to select Japanese or Korean vegetarian-friendly restaurants in Seoul. Although the hotel-specific and the restaurant-specific "views" have in common the information provided in the initial search form (i.e., city and dates), there is no link between hotel and restaurant documents and hence the constraints we imposed to restaurants are not transferred to the hotel view. Thus, although many hotels featured in *TripAdvisor* satisfy our query, narrowing down the search to only those hotels requires significant manual browsing effort.

Limitations of the facet model. In their most basic form, facets consist of a heading and a set of values; e.g., hotel star ratings in *TripAdvisor* are modelled as a facet having one value for each 1-star to 5-star rating. Many applications, however, also define facets that are *hierarchical*. For example, accommodation in *TripAdvisor* is divided into hotels, B&B, and rentals; hotels into luxury, business etc. Hierarchical facets provide *background domain knowledge* which can be exploited to improve faceted search; however, they are still rather limited. Although a hierarchical facet establishes dependencies between its values, the underlying semantic relationship (e.g., "is-a", "part-of") is undefined. There are also issues concerning dependencies between facets, which cannot be represented in such a simple model. E.g, the type of hotel and the star ratings are correlated (e.g., motels cannot be 5-star); these dependencies are typically implemented ad-hoc, which negatively impacts systems' maintainability, performance, and reliability.

Limitations of the query model. The limitations above affect queries that users can pose. In particular, facet values for different kinds of documents cannot be joined in a single query. Thus, in *TripAdvisor* our example query cannot be formulated: even if we can query for both hotels or restaurants independently, when we "switch view" from hotels to restaurants, the constraints imposed on hotels are lost. Similar limitations were observed for faceted search over interlinked documents [3, 14], webpages [12], databases [7], dataspaces [17], and knowledge bases [3, 12]. Orthogonally, there are issues with the meaning of queries, which affect the way they are processed in the backend and their results are interpreted by users. In a faceted search front-end, users are presented with facets and allowed to make a multiple choice within each facet. Typically, choices in one facet are understood as logical OR, and constraints for different facets are combined with logical AND. Thus, if a user chooses "2-star" and "3-star", they are looking for hotels with two *or* three stars. Multiple choice in a single facet could also be interpreted conjunctively, e.g., when the users choses "WiFi" and "parking" facilities. Ambiguity is resolved in the backend when queries are translated into operations over inverted indicies. This process is application dependent, and it is not grounded on a formal query model that can be independently studied.

Semantic Faceted Search. RDF has been proposed by many authors as a promising technology to overcome some of the lim-

[†]Email: marenas@ing.puc.cl
[‡]Email: firstname.middlename.lastname@cs.ox.ac.uk

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2577011.

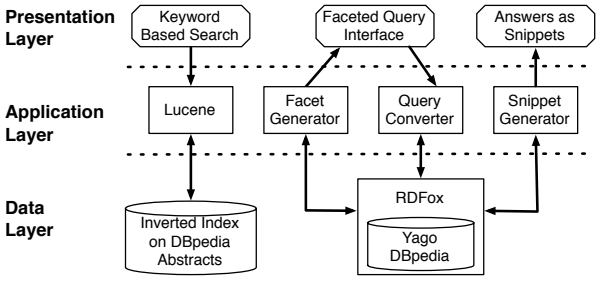


Figure 1: Architecture of SemFacet

itations of faceted search systems [2, 3, 5, 9, 10, 11, 12, 13, 15, 16, 18]. Although several RDF-based faceted search systems have been developed, there is a lack of rigorous understanding of the underlying data and query models. We aim at providing solid foundations to *semantic faceted search*—the extension of the faceted search paradigm with Semantic Web technologies. RDF was designed as a language for the representation of loosely-structured metadata, and it provides the required flexibility to semantically link different documents in arbitrary ways. OWL 2 can be used to provide rich domain knowledge on top of faceted metadata: OWL 2 axioms can capture hierarchical facets, and complex dependencies between facets in a declarative and semantically unambiguous way (e.g., business hotels cannot be 2-star, every 5-star hotels has a restaurant etc.). Finally, faceted queries can be captured by SPARQL 1.1, which provides well-understood semantics, computational properties, and powerful for query processing. Also, Semantic Web technologies provide important additional benefits. First of all, semantic facets and faceted query interfaces can be automatically generated from RDF and OWL 2 ontologies. Then, OWL 2 axioms can be used to specify which facets and values to display at each step of query refinement, thus providing valuable guidance to users. These techniques are orthogonal and complementary to the facet ranking mechanisms.

OWL 2 can also be used to simplify the annotation of documents with faceted metadata and deal with sparse and incomplete annotations. E.g., annotating data items with hierarchical facets is cumbersome since data must contain a value for each level of the hierarchy; in contrast, by representing hierarchies in OWL 2, we only need annotations for the most specific relevant values since the remaining ones can be automatically derived. Finally, semantic facets give a mechanism to query semantically related data sources, and hence are a natural query paradigm for ontology-enhanced linked data. We refer the reader to [1, 8] for more details on our approach.

2. THE SEMFACET SYSTEM

Our approach is general and can be used to provide faceted search over any RDF and OWL 2 ontology. To illustrate its potential in practice and assess the feasibility of our techniques, we implemented a prototypical faceted search system, called SemFacet (see [1, 8] for details), on top of (a fragment of) Yago [19] ontology and DBpedia containing around 15 million triples altogether.

A general architecture of SemFacet is in Figure 1. The backend relies on Lucene for keyword based search and RDFox, a massively parallel in-memory RDF triple store, for storing RDF triples, performing reasoning, and answering queries. SemFacet is implemented in such a way that both Lucene and RDFox can be substituted with any other software that provide the same functionality.

The front-end of SemFacet, by relying on nesting of conventional facets, allows users to formulate tree shaped SPARQL queries over RDF and OWL 2. The process of constructing queries is (see [8] for details): the first step is to provide a set of keywords, which leads to a set of initial answers and initial facets. Query refinement is then an iterative process, where users can either choose available facet values, or refocus the query to a different facet. In response the system updates the query answers as well as the facets available (they are automatically generated from the underlying RDF and OWL 2 ontology) to continue query refinement.

SemFacet also exploits OWL 2 axioms to enrich RDF data with implicit triples. This helps in addressing *sparsity* of annotations and modelling of *hierarchical* facets. Moreover, OWL 2 axioms help in avoiding "dead ends" (i.e., facet value selections that lead to queries with the empty answer). In conventional faceted search applications, the detection of such dead ends is data driven, in the sense that the interface does not display facet values for which no document exists. Axioms provide an alternative, declarative, way to detect dead ends during faceted search, e.g., by exploiting axioms expressing disjointness between classes of objects.

SemFacet is available as a Web service [1] and runs on a machine with 1vCPU, 4Gb of memory, and 20Gb of disk space. Although we have not formally evaluated our system, preliminary experiments show typical response time comparable with well known conventional faceted search systems.

3. REFERENCES

[1] SemFacet: Semantic Faceted Search Project. `http://www.cs.ox.ac.uk/isg/projects/SemFacet/`.

[2] T. Berners-Lee and et al. Tabulator redux: Browsing and writing linked data. In *LDOW*, 2008.

[3] S. Buschbeck and et al. A demonstrator for parallel faceted browsing. In *EKAW'12*, 2012.

[4] E. Clarkson, S. B. Navathe, and J. D. Foley. Generalized formal models for faceted user interfaces. In *JCDL'09*.

[5] J. Diederich, W.-Tilo Balke, and U. Thaden. Demonstrating the semantic growbag: automatically creating topic facets for FacetedDBLP. In *JCDL*, page 505, 2007.

[6] D.Tunkelang. *Faceted Search*. Morgan & Claypool Pubs.'09.

[7] G. H. L. Fletcher and et al. Towards a theory of search queries. *ACM Trans. Database Syst.*, 35(4):28, 2010.

[8] B.Cuenca Grau, E.Kharlamov, Š.Marciuška, D.Zheleznya−kov, M.Arenas, and E.Jimenez-Ruiz. Semfacet: Semantic faceted search over yago. In *WWW (Companion Volume)'13*.

[9] R. Hahn and et al. Faceted wikipedia search. In *BIS*, 2010.

[10] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A browser for heterogeneous semantic web repositories. In *ISWC'09*.

[11] D. Huynh and et al. Piggy bank: Experience the semantic web inside your web browser. *J. Web Sem.*, 2007.

[12] David F. Huynh and David R. Karger. Parallax and companion: Set-based browsing for the data web. www.davidhuynh.net.

[13] E. Hyvönen, S. Saarela, and K. Viljanen. Ontogator: Combining view- and ontology-based search with semantic browsing. In *XML Finland*, 2003.

[14] A. Jameson. How can we support multifocal exploration of semantic data? www.imash.leeds.ac.uk/event/keynote.html.

[15] G. Kobilarov and I. Dickinson. Humboldt: Exploring linked data. In *LDOW'08*.

[16] E. Oren, R. Delbru, and S. Decker. Extending faceted navigation for rdf data. In *ISWC*, 2006.

[17] Kenneth A. Ross and Angel Janevski. Querying faceted databases. In *SWDB*, pages 199–218, 2004.

[18] M. C. Schraefel and et al. The evolving mSpace platform: leveraging the Semantic Web on the trail of the memex. In *Hypertext*, 2005.

[19] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *WWW 2007*.