Ontology Based Access to Exploration Data at Statoil

E. Kharlamov[1,⋆] D. Hovland[2] E. Jiménez-Ruiz[1] D. Lanti[3] H. Lie[4] C. Pinkel[5]
M. Rezk[3] M. G. Skjæveland[2] E. Thorstensen[2] G. Xiao[3] D. Zheleznyakov[1] I. Horrocks[1]

[1] University of Oxford; [2] University of Oslo; [3] Free University of Bozen-Bolzano;
[4] Statoil ASA; [5] fluid Operations AG

Abstract. Ontology Based Data Access (OBDA) is a prominent approach to query databases which uses an ontology to expose data in a conceptually clear manner by abstracting away from the technical schema-level details of the underlying data. The ontology is 'connected' to the data via mappings that allow to automatically translate queries posed over the ontology into data-level queries that can be executed by the underlying database management system. Despite a lot of attention from the research community, there are still few instances of real world industrial use of OBDA systems. In this work we present data access challenges in the data-intensive petroleum company Statoil and our experience in addressing these challenges with OBDA technology. In particular, we have developed a deployment module to create ontologies and mappings from relational databases in a semi-automatic fashion, and a query processing module to perform and optimise the process of translating ontological queries into data queries and their execution. Our modules have been successfully deployed and evaluated for an OBDA solution in Statoil.

1 Introduction

The competitiveness of modern enterprises heavily depends on their ability to make use of business critical data in an efficient and timely manner. Providing this ability in data intensive enterprises is not a trivial task as the growing size and complexity of information sources makes data access and exploitation increasingly challenging. Indeed, data is often scattered across heterogeneous and autonomously evolving systems or has been adapted over the years to the needs of the applications they serve, making it difficult to extract data in a useful format for the business of the organisation.

Statoil is a large and data intensive enterprise where the workflow of many units heavily depends on timely access to data. For example, the job of exploration geologists is to analyse existing relevant data in order to find exploitable accumulations of oil or gas in given areas. This process is typically done in two separate steps: first by gathering data from multiple sources, then by analysing the data using specialised analytical tools. As is often the case in large enterprises, naming conventions for schema elements, constraints, and the structure of database schemata are very complex and documentation may be limited or nonexistent. As a result, the data gathering task is often the most time-consuming part of the decision making process.

Ontology Based Data Access (OBDA) [21] is a prominent approach to data access in which an *ontology* is used to mediate between data users and data sources. The

⋆ Corresponding author: evgeny.kharlamov@cs.ox.ac.uk

ontology provides 'a single point of semantic data access', and allows queries to be formulated in terms of a user-oriented conceptual model that abstracts away complex implementation-level details typically encountered in database schemata. Domain experts are thus able to express information needs in their own terms without any prior knowledge about the way the data is organised at the source, and to receive answers in the same intelligible form. The ontology is connected to the data via a set of *mappings*: declarative specifications that relate ontological terms with queries over the underlying data. OBDA systems automatically translate ontological queries, i.e., SPARQL, into database queries, i.e., SQL, and delegate execution of SQL queries to the database systems hosting the data. OBDA is a natural fit to address the Statoil data access challenges described above: if complex databases are presented to users via an ontology, then they can formulate queries in terms of classes and properties in an object-centric fashion, e.g., asking for all *wellbores penetrating* a *rock layer* of a specific *geological age*. Moreover, OBDA is a so-called *virtual* approach, providing an access layer on top of databases while leaving the data in its original stores. Thus, OBDA has the potential to improve data access with a minimal change to existing data management infrastructure.

OBDA has recently attracted a lot of attention and a number of systems have been developed, e.g., [5, 22]. However, to the best of our knowledge, the following two problems have attracted only limited attention:

(i) How to *create* ontologies and mappings for a deployment of an OBDA system?
(ii) How to ensure that OBDA query processing is *efficient* in practice?

At the same time, these problems have high practical importance for OBDA systems in general and in particular for deploying an OBDA system in Statoil. First, deployment of an OBDA system comes with a high modelling cost due to the complexity of the domain and of the database schemata. Second, unoptimised OBDA query processing may become impractical when the ontology and/or database are large [23].

In this paper we present our experience in the development of OBDA deployment and query optimisation modules, and their use in providing an OBDA solution for Statoil. In particular, our solution *(i)* applies a novel set of semi-automatic techniques to bootstrap new ontologies and mappings from relational databases and to integrate existing ones, and *(ii)* uses novel optimisation techniques to improve query processing for producing compact and efficient SQL queries.

Our goal is to improve the efficiency of the data gathering routine for Statoil geologists by focusing on their access to two prominent data sources: their *Exploration and Production Data Store (EPDS)*, and the *NPD FactPages*. EPDS is Statoil's corporate data store for exploration and production data and their own interpretations of this data, and is thus an important and heavily used database. EPDS was created about 15 years ago and it currently has about 3,000 tables with about 37,000 columns, and contains about 700 GB of data. The NPD FactPages (NPD FP) is a publicly available dataset that is published and maintained by the Norwegian authorities. It contains reference data for many aspects of the Norwegian petroleum industry, and is often used as a data source by geologists in combination with EPDS. The NPD FP is converted into a relational database with about 70 tables, 250 columns and 50 MB of data [26]. As an example of the difficulty of the data gathering task at Statoil: due to the complexity of EPDS, writing queries over it is a significant effort that requires proficiency with all the variety of its underlying schema

components, and it is common for SQL queries that gather data needed by geologists to contain thousands of terms and have 50–200 joins.

Our OBDA solution has been successfully deployed at Statoil over EPDS and NPD FP.[1] In particular, as a measure of success, we used a catalogue of queries collected from Statoil geologists which cover a wide range of typical information needs, and that are hard to formulate over EPDS and NPD FP; our deployment covers this catalogue. Finally, we demonstrate empirically that our optimisation techniques guarantee good execution times for the catalogue queries.

The paper is organised as follows: in Section 2 we present Statoil's data access challenges, in Section 3 we give an overview of OBDA, in Section 4 we give technical background of our deployment and query processing solutions, in Sections 5 and 6 we share our experience on OBDA deployment and query execution at Statoil, and in Sections 7 and 8 we summarise the lessons we learned, propose future work, and conclude.

2 Data Access Challenges at Statoil

Following common practices of large enterprises, geologists at Statoil analyse data in two steps: they first gather relevant data from EPDS and other data sources, and then apply analytical reporting tools on top of the gathered data [9]. The first step is typically done via a variety of query interfaces and data extraction tools, such as geographical information system (GIS) tools and specialised data manipulation tools, that we shall collectively refer to as *access points*. The flexibility of the access points is limited and in general users can control them only by inserting values for certain query parameters. When information needs cannot be satisfied with any of the available access points, geologists, possibly with the help of IT staff, try to combine answers obtained from several access points. In some cases, geologists have to contact IT staff to provide a new access point. Developing new access points is a very time consuming process; in Statoil it commonly takes from several days to weeks to produce an access point that completely answers the required information need. It has been estimated that in the oil and gas industry 30–70% of the time geologists spend on analytical tasks used for gathering data [8]; this estimate is confirmed by Statoil.

To get a better understanding of why data gathering takes so long, consider a common procedure of creating an access point. Each such point is typically based on a materialised special purpose database view. The process of making such view over EPDS consists of the three ETL steps: *(i)* extracting, *(ii)* transforming, and *(iii)* loading data. For Step (i) Statoil IT staff locate relevant data in EPDS, other data sources or existing access points, and produce SQL code for its extraction. This is done using specialised data extraction tools, since directly accessing complex database schemata like EPDS' is prone to error, time consuming, and not always feasible due to its complexity and limited documentation. During Step (ii), using specialised tools, IT staff define how to preprocess the data extracted at Step (i). This adds another layer of data processing, this time over the relevant data, to perform projections, filtering, joins, schema renamings and complex computations such as geographical coordinate conversions. In Step (iii), IT staff populate the access point's view with the data: both data extraction and manipulation code is executed which materialises the view. In sum, building an ETL process that

[1] We also have a preliminary deployment of the solution at Siemens [16].

establishes an access point for a complex information need consists of a myriad of data access and processing steps, many of which require deep knowledge of the data that is being processed and how it is represented.

There are around 900 geologists and geophysicists at Statoil and accessing data is their daily routine. Since they often need new access points, hundreds of highly skilled employees often have to wait several days for accessing data before they can conclude analytical tasks. Reducing this time from days to hours would bring a significant saving and, more importantly, would improve the effectiveness of Statoil's exploration department, which is key to their overall competitiveness and profitability.

3 Ontology Based Data Access

An OBDA instance $\mathcal{S} = (\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$ is a quadruple where $\mathcal{D}$ is an RDB, $\mathcal{V}$ is an ontological vocabulary, i.e., a set of classes and properties, $\mathcal{O}$ is an ontology over $\mathcal{V}$, i.e., a set of axioms expressed in a fragment of first-order logic, and $\mathcal{M}$ is a set of mappings between $\mathcal{D}$ and $\mathcal{V}$, i.e., assertions of the form: $P(f(x), f(y)) \leftarrow \mathtt{SQL}(x, y)$ or $C(f(x)) \leftarrow \mathtt{SQL}(x)$ where C and P are class and property names from $\mathcal{V}$, $\mathtt{SQL}(x)$ and $\mathtt{SQL}(x, y)$ are SQL queries over $\mathcal{D}$ with one and two output variables, and f is a function 'casting' values returned by SQL into URIs and values (e.g., strings, dates).

In order to answer an ontological query Q expressed in terms of $\mathcal{V}$ over $(\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$, one can execute over $\mathcal{D}$ the SQL queries occurring in $\mathcal{M}$, then use the computed answers to 'populate' the extensions of the corresponding classes and properties occurring in $\mathcal{M}$, thus creating a set of ontological facts $\mathcal{A}$,[2] and finally evaluate Q over $\mathcal{A} \cup \mathcal{O}$. Since $\mathcal{A} \cup \mathcal{O}$ is a logical theory, query answering over it corresponds to logical reasoning and is defined in terms of *certain answers*. Intuitively, a tuple $\boldsymbol{t}$ is a certain answer to Q over $\mathcal{A} \cup \mathcal{O}$ if $\mathsf{Q}(\boldsymbol{t})$ holds in every first-order model of $\mathcal{O} \cup \mathcal{A}$ [6]. Computation of $\mathcal{A}$ and certain answers, however, can be very expensive, with worst-case complexity depending on both ontology and query language expressiveness. In particular, it was shown [6] that the computation is tractable in data complexity (i.e., in the size of $\mathcal{D}$) if ontological queries Q are conjunctive (CQs) and ontologies $\mathcal{O}$ are expressed in OWL 2 QL. Computation of certain answers in this setting can be accomplished using the two stage approach of *(i) rewriting* and *(ii) unfolding* as depicted in Figure 1. Rewriting of ontological queries essentially corresponds to compilation of relevant ontological information into Q; it is similar to the resolution procedure in Prolog, and can be accomplished with the *perfect reformulation algorithm* [6], that takes as input a conjunctive query Q and an OWL 2 QL ontology $\mathcal{O}$ and returns a union of conjunctive queries RQ. Computation of certain answers for RQ over $\mathcal{A}$ returns the same answers as for Q over $\mathcal{A} \cup \mathcal{O}$. Unfolding inputs RQ and $\mathcal{M}$ and translates RQ into an SQL query $\mathtt{UQ}$ by essentially substituting occurrences of classes and properties in RQ with the SQL queries that they correspond to in $\mathcal{M}$. Evaluation of $\mathtt{UQ}$ over D effectively returns the certain answer computed by RQ over $\mathcal{A}$ and thus by Q over $\mathcal{A} \cup \mathcal{O}$.

4 Technical Background of Our OBDA System

Our OBDA deployment in Statoil relies on two self-developed systems: BOOTOX [13] for creating ontologies and mappings, and *Ontop* [24] for performing query optimisation

[2] This process is called *materialisation* of the ontological facts from data via mappings.

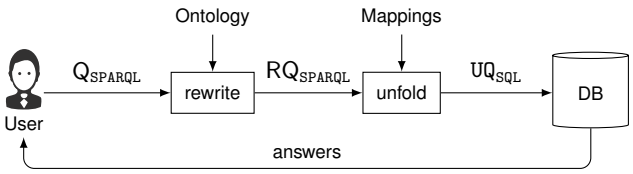


Fig. 1. Query processing in OBDA

and SPARQL query processing. During the deployment and use of BOOTOX and *Ontop* in Statoil we experienced many practical challenges, which required substantial improvements to both systems.

4.1 Deployment

We support three deployment scenarios:

(i) *Bootstrapping* is used for semi-automatic extraction of an ontology and mappings from an RDB. It takes as an input a dataset $\mathcal{D}$ and returns a vocabulary $\mathcal{V}$, a set of mappings $\mathcal{M}$ relating $\mathcal{D}$ to $\mathcal{V}$, and an ontology $\mathcal{O}$ over $\mathcal{V}$.

(ii) *Merging* is applied to incorporate pre-existing ontologies in existing OBDA instances, e.g., by aligning it with the bootstrapped ontology. It takes as input an OBDA instance $(\mathcal{O}_1, \mathcal{M}, \mathcal{D})$ and an ontology $\mathcal{O}_2$, and returns $(\mathcal{O}, \mathcal{M}, \mathcal{D})$ where $\mathcal{O}$ is a 'merger' of $\mathcal{O}_1$ and $\mathcal{O}_2$.

(iii) *Layering* is used to to 'connect' pre-existing ontologies directly to an RDB with semi-automatically generated mappings. It takes an ontology $\mathcal{O}$ and a dataset $\mathcal{D}$ as input and returns a set of mappings $\mathcal{M}$ such that $(\mathcal{O}, \mathcal{M}, \mathcal{D})$ is an OBDA instance.

The mappings and ontologies we obtain in all the three scenarios require further inspection by ontology engineers and domain experts to detect the most promising classes, properties, axioms, and mappings, and to verify their quality. To the best of our knowledge existing bootstrapping systems provide limited or no support for the three deployment scenarios above, see, e.g., [25, 32] for an overview of such systems.

Bootstrapping. The goal of bootstrapping is to find patterns in $\mathcal{D}$, i.e., SQL queries $\texttt{SQL}(x)$ and $\texttt{SQL}(x, y)$ that correspond to meaningful classes and properties. We bootstrap three types of mappings depending on what relational algebra operators can appear in their SQL-parts: *(i) projection*, *(ii) selection*, and *(iii) full mappings* that allow any relational algebra operators.

A special kind of projection mappings, that are recommended by W3C as the standard way to export relational data in RDF, are *direct mappings*. They mirror RDB schemata by essentially creating one class for each table and one property for each column. Ontological vocabulary extracted by direct mappings can be enriched with axioms by propagating RDB constraints, e.g., a foreign key relating two tables could be propagated into a subclass assertion between the two corresponding classes. BOOTOX supports extraction of direct mappings and propagation of constraints; moreover, it outperforms existing bootstrapping systems that were available for benchmarking [19]. BOOTOX can also discover implicit constraints in databases, e.g., minimal primary keys in tables, candidate foreign keys by checking containment between (samples from) projections of tables [14]. While working with EPDS we found that the discovery of implicit constraints was practically important since prominent tables are materialised views without specified primary or foreign keys, and axioms constructed from such constraints are exploited in query optimisation. Note that the bootstrapped ontology can be quite close to the

source schema and we see this as a natural outcome: bootstrapping is the first step in OBDA system deployment, and the resulting assets are by no means perfect, but provide a starting point for post-processing and extension.

Selection mappings are bootstrapped in order to create class and property hierarchies, e.g., we take a class C bootstrapped with direct mappings and verified by users, and in the corresponding tables we learn attributes whose values give good clustering of tuples; then, for each cluster we create a subclass of C. To name detected subclasses, we categorise attribute values that determine clusters by matching them to DBPedia. We also apply these techniques to other types of mappings.

In order to bootstrap full mappings, we analyse schema dependencies between tables and use statistical methods on data values. In particular, we learn chains of tables that are 'well joinable', that is, connected via foreign keys or with good overlap on some sets of attributes, and convert them into candidate classes and properties. For instance, for each chain we detect the 'leading' table T and relate the chain to a class by projecting it on T's primary key; then, we combine names of the tables in the chain and attributes on which they were joined to suggest a name for this class.

Alignment. A way to extend a bootstrapped ontology is to align it with an existing high quality domain ontology. For this purpose we extended an existing ontology alignment system LogMap [12] that aligns two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ by deriving equivalence and subclass(property) assertions between the terms from $\mathcal{O}_1$'s and $\mathcal{O}_2$'s vocabularies using the lexical characteristics of the terms and the structure of the ontologies. Our extension [27] is a highly scalable solution that ensures that after $\mathcal{O}_1$ and $\mathcal{O}_2$ are aligned the resulting ontology $\mathcal{O}$ is a *conservative extension* of $\mathcal{O}_1$ and $\mathcal{O}_2$ w.r.t. atomic subsumptions, i.e., $\mathcal{O}$ does not entail any sub-class(property) assertion over $\mathcal{O}_1$'s and $\mathcal{O}_2$'s vocabulary which is not already entailed by $\mathcal{O}_1$ or $\mathcal{O}_2$. When experimenting with EPDS, we noticed that these logical guarantees are often a necessity since an alignment $\mathcal{O}$ of a bootstrapped $\mathcal{O}_1$ with an imported domain ontology $\mathcal{O}_2$ that does not preserve consistency or conservativity gives the following side-effects: query answering over $\mathcal{O}$ produces answers that are unexpected by domain experts, and that would not be obtained if one queries $\mathcal{O}_1$ alone, or $\mathcal{O}$ entails axioms over $\mathcal{O}_2$'s terms that are unexpected and counter-intuitive for domain experts.

Layering. Our layering procedure uses a novel technique for the semi-automatic discovery of direct mappings between $\mathcal{O}$ and $\mathcal{D}$ by performing string matching of $\mathcal{O}$ and $\mathcal{D}$'s schemata enhanced with their structural comparison [20].

4.2 Query Processing Optimisation

Our query processing system relies on the two stage process with rewriting and unfolding as described above. It was observed [23] that a naive implementation of this approach performs poorly in practice; thus, we developed and implemented a number of techniques to optimise both stages of query processing, which we present in detail below. We empirically tested the efficiency of our optimisation techniques over EPDS and will present results on query execution in Section 6. We also evaluated our techniques in a controlled environment and our tests show that thanks to these optimisation techniques our query processing solution can dramatically outperform existing OBDA solutions [18]. Furthermore, we observed that even after we optimise rewriting and unfolding, the SQL

queries UQ we produce often return answers with duplicates; below we will discuss why this is an issue and how we addressed it.

Optimisation of Rewriting. We address two challenges:

(i) *redundancy* in RQ: fragments of RQ may be subsumed by each other and thus evaluation of UQ over RDBs will require redundant computation;

(ii) *inefficiency* of rewriting: computation of RQ is in the worst case exponential in the size of Q and $\mathcal{O}$, and thus its online computation is often slow for large Q and $\mathcal{O}$.

The main source of redundancy in RQ is that classes (properties) can participate in multiple mappings either directly, or indirectly via their multiple sub-classes (sub-properties).[3] To avoid this, we minimise both the mappings and the UCQ RQ using query containment. To address the inefficiency, we proposed two novel techniques. Both techniques can be applied in isolation or combined. Our first technique is to improve computation of class hierarchies entailed by the ontology, which the rewriting heavily relies on, by applying graph reachability techniques to a DAG-encoding of dependencies among classes. The second one is to move part of online reasoning offline: for all atomic queries we perform expensive rewriting up front and compile the results of this computation into the existing mappings, and use these enriched mappings when user queries Q are unfolded, see [17] for details.

Optimisation of Unfolding. We address three challenges with query unfolding:

(i) *redundant unions* due to redundancies in the bootstrapped ontology or mappings;

(ii) *redundant joins*, that come from the fact that on the ontological level the data is modelled as a graph, i.e., as a ternary relation, while on the data level in RDBs it is modelled with n-ary relations, and thus an unfolding of RQ into SQL over an n-ary table naturally introduces n-1 self-JOIN operations;

(iii) *inefficient joins* come from the so-called *impedance mismatch*, i.e., on the ontological level objects are represented with object ids URIs while in RDBs with tuples; thus, joins in RQ are unfolded into SQL joins over string concatenation that prevents RDBs from the use of existing indices.

To address these issues, we developed *structural* and *semantic* optimisation techniques. For structural optimisations we push joins inside the unions and special functions (such as URI construction) as high as possible in the query tree; we also detect and remove inefficient joins between sub-queries. Semantic optimisations remove redundant unions, joins, detect unsatisfiable or trivially satisfiable conditions, etc.

Optimisation of distinct answers. Removing duplicates from query answers raises an interesting problem for OBDA systems. On the one hand, OBDA theory assumes set semantics for computation of query answers, that is, each answer occurs only once in the answer set. On the other hand answering queries over relational databases RDF databases is typically implemented under bag semantics, that is, any answer can occur multiple times in the answer set. In particular, evaluation of SQL queries produced by our query processing system returns answers with duplicates. From our experience with the OBDA deployment at Statoil, these duplicates bring a number of challenges: duplicate answers appear as noise to most end-users, visualisation systems are significantly slowed down when flooded with repeated answers, and the large number of such answers negatively

[3] This issue is partially tackled at the bootstrapping stage [13].

affects network capacity and database connectivity. Using the `distinct` modifier in SPARQL queries to remove redundant answers is unfolded into the SQL `distinct`, which, in our experiments, was extremely detrimental to performance and led to a number of queries to time out. In order to overcome this problem, we propose to defer the removal of duplicate tuples to the OBDA system rather than the underlying database engine. In particular, we filter out redundant answers using predefined Java hash functions. This simple solution outperforms SQL `distinct` by orders of magnitude in most cases, and opens the door for further optimisation (see Section 6).

5 OBDA Deployment at Statoil

In this section we present our experience in deploying an OBDA instance over EPDS. We start with the requirements, then discuss how we developed the ontology and mappings for EPDS, and conclude with a quality assessment of the deployment.

5.1 Requirements

Our OBDA solution should enable efficient formulation of information needs from Statoil geologists. We gathered these needs via interviews with geologists and the IT experts who support them, which gave us a catalogue of 60 representative information requests in English. The following are examples of gathered information needs:

1. In my area of interest, e.g., the Gullfaks field, return wellbores penetrating a specific chronostratigraphic unit, and information about the lithostratigraphy and the hydrocarbon content in the wellbore interval penetrating this unit.
2. Show all the core samples overlapping with the Brent formation.
3. Show all permeability measurements in Brent.

The requests in the catalogue can be seen as 'patterns' of information needs, and each such request represents one topic that geologists are typically interested in. We verified with domain experts that the catalogue provides a good coverage for the information needs of Statoil geologists, and from this we derived the first natural minimum requirement for the ontology and mappings:

Requirement 1: *The ontology should enable formulation of queries corresponding to the catalogue's requests and mappings should enable answering these queries.*

To fulfil Requirement 1, the ontology must contain all the terms occurring in the catalogue. For example, Information need 1 contains the terms *wellbores*, *penetrating*, *chronostratigraphic unit*, *lithostratigraphy*, *hydrocarbon content*, and *wellbore interval*. All in all the catalogue contains 140 relevant domain terms. As we verified with Statoil geologists, the terms occurring in the catalogue are important, but, as expected, do not provide a sufficient domain coverage; that is, geologists need many more domain specific terms for expressing their information needs. Via interviews with geologists, we determined six domains that should be reflected in the ontology: geospacial, geometrical, enterprise, production, seismic and oil related facilities, which gave the following requirement:

Requirement 2: *The ontology should cover a wide range of geological domain terms including the ones from the catalogue and the six relevant domains.*

Table 1. Ontology metrics.

Ontology	Axioms	Classes	Object prop.	Datatype prop.
EPDS, bootstrapped	73,809	3,069	3,266	34,336
EPDS, relevant excerpt	16,592	627	864	7,634
EPDS, manually built	469	97	39	34
NPD FactPages	8,208	344	148	237
– ISC	6,644	18	4	4
– GeoSPARQL	191	78	43	9
– BFO	278	39	0	0

5.2 Development of Ontologies and Mappings

The ontology developed for Statoil consists of a part which is bootstrapped from EPDS and a second part which is the NPD FactPages ontology developed in an earlier phase of the project [26]. Running the bootstrapper over EPDS extracted an ontology comprising 3,069 classes, 37,602 properties, and 73,809 axioms from explicit and 155,152 axioms from implicit constraints. Many of the bootstrapped classes and properties have names that are related to the specific structure of EPDS and thus are meaningless to geologists, e.g., `RCA_SO_DS` or `EXP_UNIQUENESS_RULE_ATT 00022_X`, while others have a clear geological semantics, e.g., `RESERVOIR` and `WELL_ELEVATION`. In order to mitigate the meaningless terms, we did a semi-automatic quality assessment and manual improvement of the bootstrapped ontology, and extracted a fragment with 627 classes and 8,498 properties that are relevant to the domain, and 16,592 axioms that provide meaningful semantic relationships between these classes and properties. See Table 1 for the ontology metrics.

The NPD FP ontology helps us meet Requirement 2 by defining vocabulary about exploration and development facilities, oil company organisation, and seismic activities, and furthermore it contains geology and geometry terminology by importing *International Chronostratigraphic Chart* developed by the *International Stratigraphy Community* (ISC) and *GeoSPARQL*, which is used for representing basic geometric concepts and, in particular, information about topology. We partially aligned the ontology to the bootstrapped ontology and partially layered it to EPDS.

Most of the axioms in the ontologies, including all bootstrapped axioms, fall in the OWL 2 QL profile, which is required for OBDA to guarantee correctness query processing. Examples of OWL 2 QL axioms are $\mathsf{NaturalGasLiquid} \sqsubseteq \mathsf{Petroleum}$ (natural gas liquids are a sort of petroleum), $\mathsf{Wellbore} \sqsubseteq \exists \mathsf{hasLicense}$ (each wellbore has a license associated to it), and $\mathsf{Company} \sqsubseteq \exists \mathsf{hasName}$ (each company has a name). The ontology also contains 16 non-OWL 2 QL axioms, 13 of which we approximated in OWL 2 QL using the techniques of [7], and the other three involved universal restrictions, e.g., $\mathsf{WlbCoreSet} \sqsubseteq \forall \mathsf{member}.\mathsf{WlbCore}$, which were dropped.

In order to complement the bootstrapped mappings, it was necessary to manually create complex mappings to cover 24 classes and 34 properties. These are classes and properties from the catalogue that were either not discovered by the bootstrapper or the bootstrapped mappings did not sufficiently reflect their relation to EPDS. On average the size of the SQL query in each mapping is 9 words (including keywords), and they are all conjunctive queries. The queries involve up to 6 tables, with an average of 3. The number of output variables SQL queries of mappings ranges from 2 to 4, with the average of 3. In order to develop the mappings we analysed predefined queries used by different information extraction tools over EPDS. In particular, we analysed the results of the predefined queries over EPDS and compared them with the expected results, by interviewing Statoil domain experts. This allowed us to fragment the predefined queries

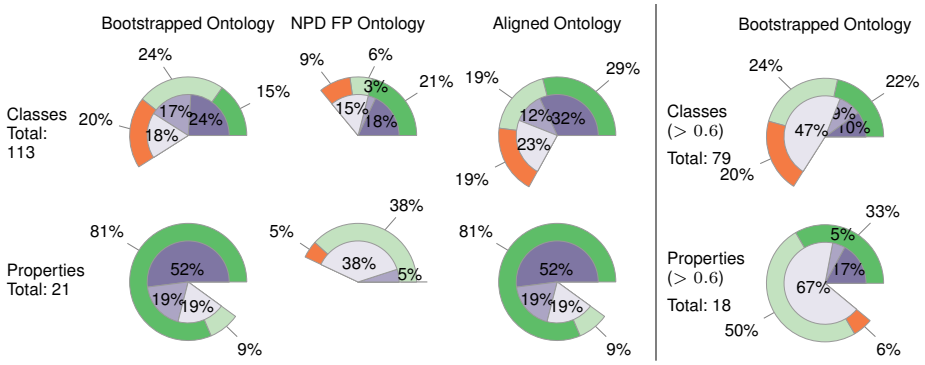


Fig. 2. Inner pie charts show coverage by lexical confidence using I-SUB [33]: ■ in $[0.9, 1.0]$, ■ in $[0.8, 0.9)$, □ in $[0.6, 0.8)$. Outer pie charts represent the quality of the terms with a coverage above 0.6: ■ *true positive*, ■ *semi-true positive*, ■ *false positive*. Note that semi-true positives are not clear-cut cases where the ontology term has a broader or narrower meaning with respect to the query term. Left part displays the coverage of terms from the query catalogue with terms from ontologies; Right part shows the overlap between terms from bootstrapped and imported ontologies that (with confidence > 0.6) occur in the query catalogue

and extract small excerpts useful for OBDA mappings. The relatively small size of the SQL parts of mappings was dictated by two practical reasons: to make maintenance and documenting of the mappings easier, and to ensure efficiency of query processing.

5.3 Assessing the Quality of Our OBDA Deployment

We assess the quality of our deployment by first checking how good the ontology covers the query catalogue, and then how good our mappings cover the terms in the catalogue.

To evaluate the coverage of the query catalogue by the ontology, we aligned them by first *(i)* a syntactic match of their terms and then *(ii)* a structural comparison of neighbourhoods around terms that have a syntactic match. The alignment outputs a set of pairs of matched terms together with a score showing how well they match. For this purpose we extended the ontology alignment system LogMap [27], that can perform both syntactic and structural matching of ontologies, so that it also can perform the required alignment of ontologies and query catalogues. The main challenge was to define the notion of a structural neighbourhood of a query term in a set of conjunctive queries. We introduced the following notion: given a set of queries S and a term t, its neighbourhood in S is the set of all terms t' occurring in some $Q \in S$ that contains t, together with the surrounding sequence of terms that is common in all such Q. We did the coverage assessment separately for the ontology bootstrapped from EPDS, the NPD FP ontology, and the alignment of these two. The results of the matching are in Figure 2, in the inner circles. The six pie charts on the left describe the coverage of the query terms by ontologies: the upper three show the coverage of classes by, respectively (left-to-right) bootstrapped, NPD FP, and aligned ontologies, the lower three show the coverage of properties. Finally, together with three domain experts we performed a manual assessment of each match for both classes and properties. The results of our assessments are also in Figure 2 in the outer circles. For example, manual assessment of coverage of classes by the bootstrapped ontology gave 15% of true positives (they are correct for domain experts), then, 24% of semi-true positives, and 20% are false-positives (the matches are wrong for domain experts). In the case of properties, most bootstrapped ones that were matched to the query catalogue terms, i.e., 81%, are true positive.

To evaluate the coverage of the query catalogue by the mappings, we used a different approach. If a query term is present in the ontology it can be used in query composition. At the same time there might be no mapping relating this term to EPDS, and all queries that use this term will always return the empty answer set. Thus, we checked how well the query terms reflected in the aligned ontology are covered with mappings. Clearly, by the construction, all the query terms that are reflected in the bootstrapped ontology are related to EPDS via bootstrapped direct mappings. While, if a query term is reflected in the NPD FP ontology, then it does not have a mapping to EPDS unless it got aligned to a term in the bootstrapped ontology. Indeed, alignment sets a subset or equivalence relationship between terms, and thus a mapping for one aligned term may be reused by another term. We verified how many query terms reflected in the NPD FP ontology are also present in the bootstrapped ontology, and in Figure 2 the right two pie charts depict the outcome. In this experiment we verified the confidence of the alignment and did manual assessment with domain experts. From these experiments we conclude that for about 50% of query classes (and about 80% of properties) from the NPD FP ontology (that are aligned to the bootstrapped classes) we can reuse direct mappings of the bootstrapped ontology.

6 Query Answering over OBDA Deployment in Statoil

In this section we present query evaluation experiments with our OBDA solution over EPDS and NPD FP. We start with the requirements.

6.1 Requirements

The goal of our OBDA deployment is to make gathering of data from EPDS more efficient; this leads us to the next requirement:

Requirement 3: Queries from the Statoil catalogue expressed over the ontology should be much simpler than the data queries in corresponding access points. Execution time of these queries over our OBDA deployment should be similar to the data extraction time of the corresponding access points.

We start with analysing the structure of queries from the Statoil catalogue. Most of them, 73%, are either linear or three-shaped conjunctive queries, the others contain aggregate functions and negation. No query in the catalogue has a cycle. Expressing them over the ontology requires from 3 to 13 ontological terms, and the longest query contains 23 terms. In Figure 3, we illustrate the query from the catalogue that correspond to Information needs 3 from Section 5 expressed over the ontology. These queries are quite simple and contain 8 and 13 terms only. At the same time, the Statoil access points corresponding to these two queries are based on two complex SQL queries Q_{IN2} and Q_{IN3}, where Q_{IN2} involves 7 tables and 24 statements in the WHERE clause with 7 technical statements of the form 'T.A is not null' that ensure correctness of query execution, and 10 joins; Q_{IN3} involves 14 tables and 38 statements in the WHERE clause with 9 technical statements and 18 joins. Due to the space limit, we do not show here Q_{IN2} and Q_{IN3}. This provides clear evidence that the catalogue queries over the ontology are much simpler than the corresponding access point queries. Moreover, expressing catalogue queries over the ontology is relatively easy and could be done quite quickly by IT experts—we did it in one day. We also believe that even geologists could formulate these queries relatively quickly with appropriate tool support.

$$
\begin{aligned}
Q_{\text{IN3}}(x_5, x_6, y_1) :- \; & \mathsf{Core}(x_1), \mathsf{extractedFrom}(x_1, x_2), \mathsf{WellboreInterval}(x_2), \\
& \mathsf{hasCoreSample}(x_1, x_5), \mathsf{CoreSample}(x_5), \\
& \mathsf{hasPermeabilityMeasurement}(x_5, x_6), \\
& \mathsf{PermeabilityMeasurementResult}(x_6), \mathsf{valueInStandardUnit}(x_6, y_1), \\
& \mathsf{overlapsWellboreInterval}(x_2, x_3), \mathsf{WellboreInterval}(x_3), \\
& \mathsf{hasUnit}(x_3, x_4), \mathsf{StratigraphicUnit}(x_4), \mathsf{name}(x_4, \text{“}BRENT\text{”}).
\end{aligned}
$$

Fig. 3. Information need 3 from Section 5 expressed over the ontology vocabulary

6.2 Running Queries over EPDS

We conducted three sets of experiments with the OBDA deployment using the Statoil query catalogue, and ontology and mappings relevant for the queries in the catalogue. The queries were run over the production server[4] of EPDS; the results are presented in the top two plots of Figure 4. In the first set of experiments, that we refer to as *noDist*, we executed queries as they appear in the catalogue, while in the other two sets we executed these queries with `distinct`, where in *dbDist* experiments `distinct` is processed by the database engine, while in *obdaDist* by the OBDA engine, as discussed in Section 4.2. Each query in the experiments was executed four times with a 20-minute timeout.

In the *noDist* experiment, 17 out of 60 queries timed out. Out of the successful 43, for 2 queries the time is less than 1s, for 3 queries it is between 2m and 6m, while for most queries it is between 3s and 2m; the average time is 36.5s and the median is 12.5s. These numbers are impressive, as the SQL queries produced by the OBDA system and sent to EPDS are large: they have on average 51k characters and the largest query is 275k characters. Note that execution time here does not include the time for rewriting and unfolding of SPARQL queries, but only the actual execution of the resulting SQL queries; the maximum unfolding time is 187ms which does not add a significant time overhead to the reported query execution time. In order to understand how good the times reported in Figure 4 are for Statoil geologists, we gathered statistics on execution time for queries behind the access points available in Statoil. For the ones that correspond to the Statoil catalogue, execution time over EPDS is on average several seconds, which is comparable to the numbers in our experiments. Moreover, in general for the simplest access points the execution takes from seconds to minutes, and for the ones corresponding to complex analytical tasks it can take overnight; thus, the response time for even the slowest of our queries should not be too slow for Statoil geologists.

We also analysed why some of our queries require longer execution times than others, and why some queries failed, i.e., timed out. The main reason for slow queries is redundant self-joins in the SQL queries generated after the SPARQL queries were rewritten and unfolded. The problem comes from the fact that for many EPDS tables used in mappings, primary keys are not declared; thus our optimisation techniques described in Section 4.2, that remove self-joins, become inapplicable. Another common pattern in the SQL queries that is responsible for slowing down execution time is redundancy of subqueries introduced by mappings, which is not detected by our query optimisation techniques and redundancy of query answers. The latter factor is important since the answer sets for 30 out of 43 queries that did not time out contained duplicates; among them, the mean ratio of redundant answers is 51.6% while the maximum is 99.8% or 83k redundant answers.

In the *dbDist obdaDist* experiments we eliminated the redundant answers at a reasonable cost: execution times of *dbDist* and *obdaDist* are similar to *noDist*. Moreover,

[4] Oracle 10g, HP ProLiant server with 24 Intel Xeon CPUs (X5650 @ 2.67 GHz), 283 GB RAM

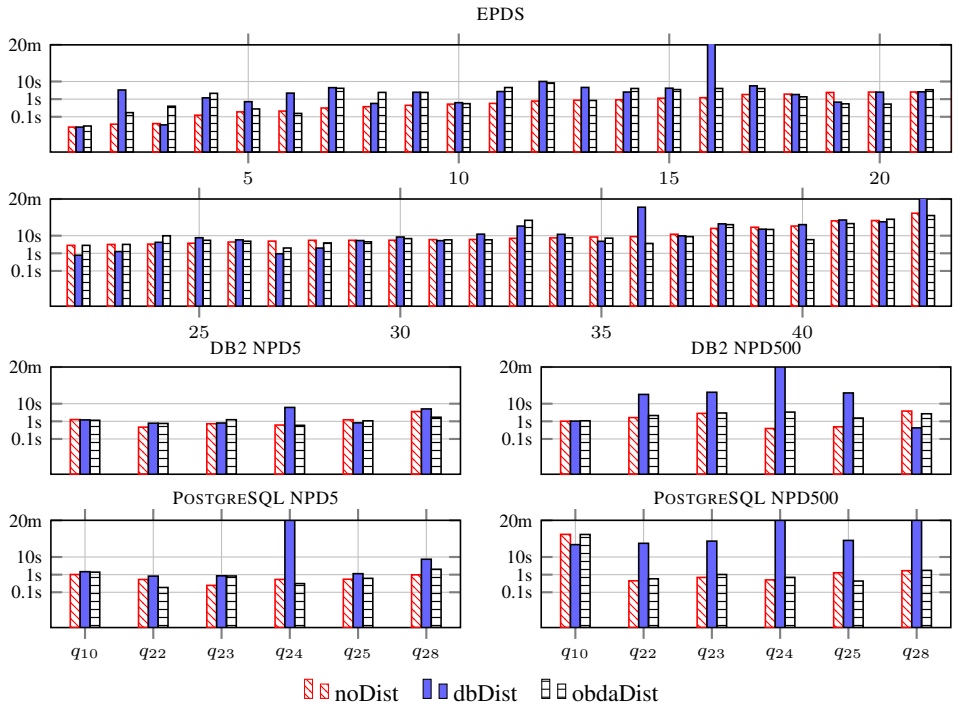


Fig. 4. Query execution times for 43 Statoil catalogue queries over EPDS, sorted after noDist; and 6 selected NPD benchmark queries over NPD5 and NPD500. Logarithmic scale

obdaDist outperforms *dbDist* in 67% cases, and in 2 cases *dbDist* execution gave a time out, while *obdaDist* never did so, which shows the benefits of our approach over *dbDist*.

6.3 Running Queries in Controlled Environment

We also conducted experiments in a controlled environment, which is important since EPDS is a production server, and hence the load on the server varies constantly and affects the results of experiments. For controlled experiments we used our own server and the NPD benchmark [18], a benchmark for OBDA systems. Compared to EPDS, our controlled setting contains smaller datasets: from 240MB to 25GB with +1.3B triples. We ran the experiments in an HP Proliant server with 24 Intel Xeon CPUs (144 cores@3.47GHz), 106GB of RAM and five 1TB 15K RPM HD using Ubuntu 12.04 64-bit edition. The tests were performed using DB2 and PostgreSQL as underlying database engines.

In the lower four plots in Figure 4 we present *noDist*, *dbDist*, and *obdaDist* experiments with 6 NPD queries whose performance was affected by the use of the `distinct`. Each experiment was conducted over two datasets: NPD5 and NPD500 corresponding to the scaling of NPD 5 and 500 times, and the results are presented separately for PostgreSQL and DB2. Note that there are 30 queries in the last version of the NPD benchmark (v1.7), while we report experiments with only the ones where the use of `distinct` had an impact (positive or negative) on performance.

Our experiments show that execution of the 6 queries without `distinct` is in most cases under 1s. At the same time if one uses `distinct` in a naive way, by delegating it to the database engines, it is extremely detrimental to performance, leading even more queries to time out (see *dbDist* experiments). Nevertheless, with our treatment of `distinct` (see *obdaDist*), in most cases the queries perform orders of magnitude

better than with the naive implementation; as in the EPDS case, the optimisation seems to be more effective when the ratio of redundant answers is high, e.g., the highest ratio of redundant answers is 90% for query q_{24}, and the mean ratio for the queries is around 50%.

Importantly, compared to queries without `distinct`, the overhead of removing redundancy with our techniques is small. In some cases the execution time in *obdaDist* was even better than in *noDist* because the number of the returned results shipped to the end user is significantly reduced. However, there is one interesting exceptional case where the first query in *dbDist* over PostgreSQL (and the last query over DB2) performs better than *noDist* and *obdaDist*. This phenomenon still requires further investigation.

7 Lessons Learned and Future Work

During the course of the project with Statoil we had a unique opportunity to deploy OBDA technology in a real industrial setting, to understand the limitations of the current technology, to address these limitations, and to get new ideas for further research. Our OBDA solution for Statoil has improved efficiency of data gathering for geologists by allowing them to *(i)* efficiently express information needs as ontological queries and *(ii)* efficiently execute these queries over the OBDA deployment.

Regarding Item (i), our experiments show that, although we achieved our objectives, there is still room for improvement. Indeed, our OBDA deployment addresses Requirements 1 and 2 from Section 5: it has enough ontological terms to express queries in the query catalogue, and they all are 'connected' to EPDS via mappings (thus addressing Req. 1); and the ontology underlying the deployment has a wide range of terms coming from several respected sources (thus addressing Req. 2). Importantly, most of the ontology was developed using our semi-automatic deployment module and only minor manual work was required to extend the ontology in order to cover the query catalogue. Of course, the resulting ontology by no means gives an exhaustive coverage of the oil and gas domain, but we see it as a good starting point for developing a more sophisticated ontology, and as a suitable point of entry for OBDA to Statoil's EPDS database. Moreover, from the research point of view, we plan to develop ontology bootstrapping techniques that can be more targeted to specific scenarios, e.g., that could work not only with database schemata and data, but also with log files of queries, precomputed views and other assets specific for a given domain or scenario. We also plan to compare performance and quality of our bootstrapper with existing analogous systems. Finally, in order to improve usability of our OBDA deployment and to facilitate construction of ontological query to end users with limited system experience we work on intuitive query interfaces [1–4, 28–31].

Regarding Item (ii), the execution time of most queries from the Statoil catalogue was impressive and comparable to the performance of Statoil's existing access points, thus addressing Requirement 3 from Section 6. Ensuring that the remaining queries do not time out requires further research and we currently look into this, e.g., we work on optimisation techniques to eliminate redundant self-joins. Besides, we plan to conduct experiments with Statoil queries using other available OBDA query processing engines and compare results with the outcome of experiments reported in this paper. Finally, our treatment of `distinct` led to a significant improvement in performance in comparison to standard processing of `distinct` in OBDA systems. In our opinion, not only `distinct`, but also other query operators can benefit if the query planning is at least partially delegated from

SQL to OBDA processing engines, where the planner exploits statistics of concepts and properties and the structure of the SPARQL query; this, however, requires further research.

For future work we plan to integrate our OBDA deployment in the business processes of Statoil engineers and IT personnel. Moreover, we are working on a better integration of the deployment with existing Statoil analytical and data visualisation tools, and have already integrated our solution with a geospatial data visualisation tool. Another challenge that we plan to address in the remaining stages of the project is extension of the deployment from EPDS and NPD FP to other Statoil data sources. This will require development of a distributed query processing infrastructure, extension of the Statoil query catalogue, and experiments with our deployment and query execution solutions. An important request that we got while evaluating the OBDA deployment with Statoil engineers is to allow for users' interaction with the system: engineers would like to send their feedback to the OBDA system, e.g., by saying that some ontological terms are missing, or that some answers are wrong, and to get explanations from the system, e.g., to get provenance to query answers that include the name of the database where the answers came from as well as the exact tables that were used in the mappings. Enabling such user interaction is also an area of future work. We also work on developing access control mechanisms for our OBDA deployment that can ensure that users can access only the data they are allowed to see [10, 11].

8 Conclusion

In this paper we presented a description of a data analyses routine at Statoil and challenges with access to business critical data required for these analyses. These challenges delay the analytical tasks significantly, thus addressing them is of a high importance for Statoil. Additionally, the challenges are representative for large data intensive industries and a good solution would be beneficial not only for Statoil, but for a wide range of enterprises. We believe that OBDA technology is a promising way to address the challenges while to the best of our knowledge existing off-the-shelf OBDA solutions can not be directly applied to do the job. Thus, we developed an OBDA solution that is capable of dealing with the challenges and is equipped with a deployment module BOOTOX for semi-automatic creation of ontologies and mappings, and a query processing module *Ontop* that ensures efficient OBDA query processing. We deployed our solution at Statoil and evaluated it using three requirements that are based on interviews of Statoil engineers, analyses of their business processes, and in particular on a catalogue of typical information needs of Statoil geologists. Results of the evaluation suggest that our OBDA deployment in Statoil was successful, while there is still a number of both technical and research challenging that should be addressed to improve our solution.

Acknowledgements. This work was partially funded by the EU project Optique [14, 15] (FP7-ICT-318338) and the EPSRC projects MaSI^3, Score!, and DBOnto.

9 References

[1] M. Arenas et al. Enabling Faceted Search over OWL 2 with SemFacet. In: *OWLED*. 2014.
[2] M. Arenas et al. Faceted Search over Ontology-Enhanced RDF Data. In: *CIKM*. 2014.
[3] M. Arenas et al. SemFacet: semantic faceted search over Yago. In: *WWW, Companion Volume*. 2014.
[4] M. Arenas et al. Towards semantic faceted search. In: *WWW, Companion Volume*. 2014.

[5] D. Calvanese et al. The MASTRO System for Ontology-Based Data Access. In: *Semantic Web* 2.1 (2011).
[6] D. Calvanese et al. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. In: *JAR* 39.3 (2007).
[7] M. Console et al. Efficient Approximation in DL-Lite of OWL 2 Ontologies. In: *DL*. 2013.
[8] J. Crompton. *Keynote talk at the W3C Workshop on Sem. Web in Oil & Gas Industry*. 2008.
[9] A. Doan et al. *Principles of Data Integration*. Kaufmann, 2012.
[10] B. C. Grau et al. Controlled Query Evaluation for Datalog and OWL 2 Profile Ontologies. In: *IJCAI*. 2015.
[11] B. C. Grau et al. Controlled Query Evaluation over OWL 2 RL Ontologies. In: *ISWC*. 2013.
[12] E. Jimenez-Ruiz et al. Large-Scale Interactive Ontology Matching: Algorithms and Implementation. In: *ECAI*. 2012.
[13] E. Jiménez-Ruiz et al. BootOX: Practical Mapping of RDBs to OWL 2. In: *ISWC*. 2015.
[14] E. Kharlamov et al. Optique 1.0: Semantic Access to Big Data: The Case of Norwegian Petroleum Directorate FactPages. In: *ISWC Posters & Demos*. 2013.
[15] E. Kharlamov et al. Optique: Towards OBDA Systems for Industry. In: *ESWC (Selected Papers)*. 2013.
[16] E. Kharlamov et al. How Semantic Technologies Can Enhance Data Access at Siemens Energy. In: *ISWC*. 2014.
[17] R. Kontchakov et al. Answering SPARQL Queries over Databases under OWL 2 QL Entailment Regime. In: *ISWC*. 2014.
[18] D. Lanti et al. The NPD Benchmark: Reality Check for OBDA Systems. In: *EDBT*. 2015.
[19] C. Pinkel et al. RODI: A Benchmark for Automatic Mapping Generation in Relational-to-Ontology Data Integration. In: *ESWC*. 2015.
[20] C. Pinkel et al. IncMap: Pay as You Go Matching of Relational Schemata to OWL Ontologies. In: *OM*. 2013.
[21] A. Poggi et al. Linking Data to Ontologies. In: *J. Data Sem.* 10 (2008).
[22] F. Priyatna et al. Formalisation and Experiences of R2RML-based SPARQL to SQL query translation using Morph. In: *WWW*. 2014.
[23] M. Rodriguez-Muro and D. Calvanese. High Performance Query Answering over DL-Lite Ontologies. In: *KR*. 2012.
[24] M. Rodriguez-Muro et al. Ontology-Based Data Access: Ontop of Databases. In: *ISWC*. 2013.
[25] J. Sequeda et al. Survey of directly mapping SQL databases to the Semantic Web. In: *Knowledge Eng. Review* 26.4 (2011).
[26] M. G. Skjæveland et al. Publishing the Norwegian Petroleum Directorate's FactPages as Semantic Web Data. In: *ISWC*. 2013.
[27] A. Solimando et al. Detecting and Correcting Conservativity Principle Violations in Ontology-to-Ontology Mappings. In: *ISWC*. 2014.
[28] A. Soylu et al. OptiqueVQS: towards an ontology-based visual query system for big data. In: *MEDES*. 2013.
[29] A. Soylu et al. Towards Exploiting Query History for Adaptive Ontology-Based Visual Query Formulation. In: *MTSR*. 2014.
[30] A. Soylu et al. Why not simply Google? In: *NordiCHI*. 2014.
[31] A. Soylu et al. A Preliminary Approach on Ontology-Based Visual Query Formulation for Big Data. In: *MTSR*. 2013.
[32] D.-E. Spanos et al. Bringing relational databases into the Semantic Web: A survey. In: *Semantic Web* 3.2 (2012).
[33] G. Stoilos et al. A String Metric for Ontology Alignment. In: *ISWC*. 2005.