Web-based Graphical Querying of Databases through an Ontology: the WONDER System

Diego Calvanese, C. Maria Keet, Werner Nutt, Mariano Rodríguez-Muro, Giorgio Stefanoni
KRDB Research Centre
Free University of Bozen-Bolzano
Bolzano, Italy
{calvanese, keet, nutt, rodriguez}@inf.unibz.it, giorgio.stefanoni@unibz.it

ABSTRACT

Biological scientists have made large amounts of data available on the Web, which can be accessed by canned or precomputed queries presented via web forms. To satisfy further information needs, users currently have to have a good understanding of SQL and how the data is stored in the database. While accessing information at the ontological layer seems more appropriate, this poses two challenges: (1) to query data in databases and triple stores through an ontology with little performance overhead, and (2) to provide an intuitive web-based access to users that are not IT experts. To address these issues, we draw upon the theory and technology developed for Ontology-Based Data Access for *DL-Lite*. With an OWL ontology and the DIG-QUONTO reasoner as building blocks, we have developed an application that allows for graphical ontology browsing, query formulation, and answer retrieval via a Web browser. We have evaluated our system for Web-ONtology baseD Extraction of Relational data (WONDER) with an existing large genomics database about horizontal gene transfer and found that it meets both the scalability and the usability requirements.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: database applications; H.5.2 [**Information Systems**]: User Interfaces; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

Keywords

Ontology-Based Data Access, Graphical Query Interface, Semantic Web, Bioinformatics

1. INTRODUCTION

Experimentation with Semantic Web technologies, such as the development and use of ontologies, is most notable in the subject domain of health care and life sciences, which is exemplified by the W3C Health Care and Life Science Interest Group dedicated to this topic [22]. Full adoption by not only these front-runners but also the large amount of end users, however, is a hurdle to be overcome. Among the issues that have to be addressed are user-friendliness and scalability of the tools. In particular the latter issue is a challenge, given that scientists and medical practitioners require semantic access to large amounts of data that in size exceeds by far the customary toy examples. Moreover, there are few approaches that marry the linking of an ontology at the semantic layer with data in databases or triple stores and the ability to query the data by availing of the knowledge represented in the ontology. These approaches either load the data from the database into the ABox as instances in the ontology, such as DataMaster [14], or link the ontology to its instances that are stored in secondary storage, which is pursued by [18, 21, 1] in the framework of Ontology-Based Data Access (OBDA). OBDA experimentation has focussed on methodology and basic queries [13] and data integration [2] to demonstrate the proof of concept. However, these examples neither did address requirements that the system implementing OBDA should be end-user usable nor that, given a Semantic *Web* setting, it should be usable on the web through a web interface instead of a separate application. Many biological databases do have a web interface, however, and the result of a transformation from legacy technology to Semantic Web technology ought to be at least as good as the original (in casu, the online services these databases offer), else there is no technological reason to migrate. In addition, now that some domain experts have mastered SQL, with the new Semantic Web technologies they would have to go through a new learning process to master the SPARQL query language, which can be a prohibiting step in adoption of new technologies. In fact, *how*, i.e., with which technology and languages, the required and desired features are implemented should not be of the domain expert's chores, instead, the focus of his concern should be on *what* aspects of the subject domain are in the system and what he can do with them.

To address these issues, we build upon the theory, technology, and implementation developed for OBDA [9, 1, 18, 20] and extend it with the proverbial 'last mile' to make it user-usable. We add a new Semantic Web technology-enabled web application to the existing OBDA infrastructure to implement graphical ontology browsing, query formulation, and execution in an Internet browser; i.e, we realise this without the need for a stand-alone application at the client side. Moreover, the rigorous formal characterisation of the graphical query language and its coupling with an OWL ontology on the one hand, and conjunctive queries (in SPARQL syntax), epistemic queries [8], and the DIG-QUONTO [18] reasoner on the other, ensures correct formulation and evaluation of the queries. The resulting Web-ONtology baseD Extraction of Relational data (WONDER) system then not only meets the scalability requirement, but also the usability requirement. To put this to the test, we have used it on an existing widely used genomics database that stores 4GB of data about horizontal gene transfer among prokaryotes [12].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'10 March 22-26, 2010, Sierre, Switzerland.
Copyright 2010 ACM 978-1-60558-638-0/10/03 ...$10.00.

The remainder of the paper is structured as follows. We first introduce the motivating and running example about horizontal gene transfer (Section 2). The OBDA approach and its realisation are described in Section 3. The novel addition to the OBDA infrastructure, i.e., the web-based graphical query feature for ontologies and its implementation in the WONDER system, is described in Section 4. We evaluate the WONDER system in Section 5 and conclude in Section 6.

2. RUNNING EXAMPLE: HORIZONTAL GENE TRANSFER

An important aspect of evolutionary microbiology is horizontal gene transfer (HGT), where DNA from one bacterium is transmitted to another bacterium; that is, in addition to the common strict vertical descent of genetic traits from one generation to another, bacteria can swap genes among each other. This is a significant force in bacterial innovation, with key examples such as the transfer of antibiotic resistance among pathogenic bacteria and degradation of xenobiotics by bacteria to clean up polluted environments. Key questions scientists seek to answer are to find out which genes have been horizontally transferred, what do they do, and where do they come from. Few databases and tools, such as the HGT-DB database and $\delta\rho$-web tool [12, 16], have been developed that contribute to addressing these questions, and in some cases functional assessments of predicted acquired genes have been made, such as the overrepresentation of virulence factors in HGT processes [15]. However, more comprehensive and integrated tools are required to find answers concerning the origin, composition, quantity, and significance of gene flow within microbial communities. Thus, given the huge amounts of genomic data available nowadays, sophisticated querying is an imperative to answer such scientific questions that can have an impact on the lives of millions of people.

For this case study, we will enhance and simplify access to the HGT-DB [12], which is a MySQL database with web-based front end [1] that was kindly made available for testing purposes by its PI Santiago Garcia-Vallvé. In a prior activity, the database has been reverse engineered into an ORM conceptual data model and improved upon through interactions with the domain experts. The main classes in the diagram are about organisms and related information, such as name, position in the species taxonomy, number of chromosomes, and statistics of the genome (e.g., GC content), and genes with related information, such as KEGG and COG code, name, position on the genome, various statistics, and if it is predicted to be horizontally transferred; details about the conceptual data model will be elaborated on in a separate paper. This conceptual data model has been transformed from its first order logic representation in ORM into a *DL-Lite*$_\mathcal{A}$ representation (see Section 3) to make it 'OBDA-ready' for the current tools[2].

The present HGT-DB web interface offers canned queries and the possibility of retrieving text files of pre-computed queries, but interesting queries for *in silico* biology, such as the following ones, cannot be posed.

A. *For organism Bordetella_pertussis, retrieve the genes that are predicted to be acquired by hgt.* This query can be trivially extended for *B. pertussis* (the causative agent of whooping cough), or any other prokaryote in the database, to also retrieve attributes such as the KEGG and COG codes;
B. *Retrieve all the organisms that have gene products involved in KEGG pathway ko03010.* To make the query more specific, one can change *all the organisms* with, say, *for Helicobacter pylori* (the causative agent of stomach ulcers) *all the genes;*
C. *For organisms of the Bacillus spp., retrieve the hgt prediction of the gene dnaA;*
D. *For the Firmicutes, retrieve the organisms and their genes that have a GC3 contents higher than 80.*

Instead of browsing through the database to find the needle in the haystack, we will not burn the haystack, but add a semantic layer over the database to enhance the legacy system and simplify querying for the needle by using the novel, domain expert friendly, graphical query language that enables the users to pose queries at the ontology layer without having to bother learning to write SPARQL and SQL queries.

3. ONTOLOGY BASED DATA ACCESS

To realise the easy-to-use semantic layer for the HGT-DB (or any other relational database), we build upon the OBDA theory and infrastructure. In OBDA, the objective is to provide access to one or more *data sources* through a mediating ontology. The data in the data sources are associated to the entities of the ontology (i.e., classes, datatypes, object properties, and data properties) by means of mappings with formally defined semantics (see Fig. 1). An OWL reasoner capable of working with the OBDA architecture is called an *OBDA enabled reasoner*. We elaborate now on the components of the specific OBDA framework used with the WONDER system. We first introduce the formal OBDA framework in Section 3.1, and then describe the software tools that realize the framework and that allow us to design and deploy the HGT scenario in Section 3.2.

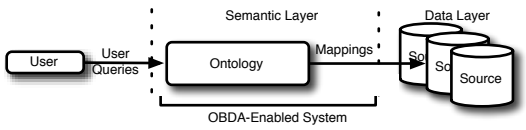


Figure 1: OBDA architecture

3.1 Query Answering over *DL-Lite*$_\mathcal{A}$ Ontologies with Mappings

The OWL DL fragment of the Web Ontology Language (OWL) is based on Description Logics (DLs) [5], which are logics specifically designed for structured representation of knowledge. We now describe the formal syntax and semantics of *DL-Lite*$_\mathcal{A}$, a DL of the *DL-Lite* family [9, 7] particularly well-suited for OBDA. The *DL-Lite* family is at the basis of OWL 2 QL, one of the three profiles of OWL 2, which has been designed specifically for efficient access to large amounts of data[3].

Syntax of *DL-Lite*$_\mathcal{A}$ Ontologies. In DLs, the domain of interest is represented by means of classes and properties[4], which denote unary and binary predicates, respectively. Following conceptual data models and OWL, in *DL-Lite*$_\mathcal{A}$ we distinguish between (abstract) objects and (data) values. A *class expression* denotes a set of objects, while a *datatype*[5], denotes a set of values. Similarly, an *object property* denotes a binary relation between objects, and

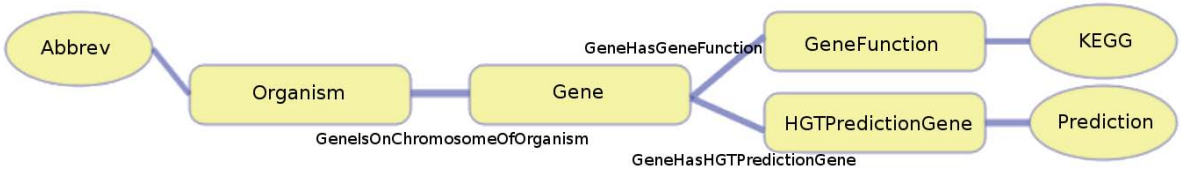
[1] http://genomes.urv.cat/HGT-DB/

[2] The OWL file and a DL notation is available as supplementary material at http://obda.inf.unibz.it/obdahgtdb/obdahgtdb.html; see also Section 3.1 and Fig. 2.

[3] OWL 2 is the new version of the Web Ontology Language OWL, which is currently in the process of being standardized by the W3C, see http://www.w3.org/2007/OWL/.

[4] Classes and properties are traditionally called *concepts* and *roles* in DLs, but here we follow the OWL terminology.

[5] In OWL, most datatypes are taken from the set of XML Schema



$\delta(Abbrev) \sqsubseteq Organism$	An abbreviation is for an organism
$\rho(Abbrev) \sqsubseteq$ `xsd:string`	An abbreviation is of type string
$Organism \sqsubseteq \delta(Abbrev)$	Each organism has an abbreviation
(funct $Abbrev$)	Each individual has a single abbreviation
$\exists GeneIsOnChromosomeOfOrganism \sqsubseteq Gene$	Domain of object property
$\exists GeneIsOnChromosomeOfOrganism^- \sqsubseteq Organism$	Range of object property
$Gene \sqsubseteq \exists GeneIsOnChromosomeOfOrganism$	Each gene belongs to some organism
$Organism \sqsubseteq \exists GeneIsOnChromosomeOfOrganism^-$	Each organism has some gene
(funct $GeneIsOnChrOfOrganism$)	Each gene belongs to at most one organism
...	

Figure 2: Section of the HGT application ontology.

a *data property* denotes a binary relation between objects and values. We assume to have a set $\{T_1, \ldots, T_n\}$ of pairwise disjoint and unbounded datatypes, each denoting a set $val(T_i)$ of values (e.g., integers, strings, etc.). $\top_d$ denotes the set of all values. Class expressions, denoted C, and object property expressions, denoted R, are formed according to the following syntax, where A denotes a class, P an object property, and U a data property:

$$C \longrightarrow A \mid \exists R \mid \delta(U), \qquad R \longrightarrow P \mid P^-.$$

Here, $\exists R$ denotes an unqualified existential class expression, $\delta(U)$ denotes the domain of U, and P^- denotes the inverse of P.

A *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, is constituted by a *TBox* $\mathcal{T}$ representing intensional knowledge, and an *ABox* $\mathcal{A}$, representing extensional knowledge. The TBox is constituted by a set of *axioms* of the form

$$\begin{array}{cccc} C_1 \sqsubseteq C_2, & \rho(U) \sqsubseteq T_i, & R_1 \sqsubseteq R_2, & U_1 \sqsubseteq U_2, \\ (\mathsf{disj}\ C_1\ C_2), & & (\mathsf{disj}\ R_1\ R_2), & (\mathsf{disj}\ U_1\ U_2), \\ & & (\mathsf{funct}\ R), & (\mathsf{funct}\ U). \end{array}$$

The axioms in the first row denote *inclusions* ($\rho(U)$ denotes the range of U), those in the second row *disjointness* (distinct datatypes are implicitly disjoint), and those in the third row *functionality*. The ABox is constituted by a set of *assertions* of the form $A(a)$, $P(a, a')$, and $U(a, \ell)$, where a, a' are individuals (denoting objects) and ℓ is a literal (denoting a value). To ensure that *DL-Lite*$_\mathcal{A}$ enjoys the nice computational properties of the *DL-Lite* family [9], we have to restrict the form of the TBox by requiring that object and data properties occurring in functionality assertions cannot be specialized (i.e., appear in the right hand side of an inclusion axiom). We refer to [17] for a justification.

As an example, a portion of the HGT application ontology, which overall has 31 classes, 32 object properties, 61 data properties, and 108 subclass axioms, is shown in Fig. 2, together with some of the corresponding *DL-Lite*$_\mathcal{A}$ axioms and their intuitive meaning.

Semantics of *DL-Lite*$_\mathcal{A}$ Ontologies. The semantics of *DL-Lite*$_\mathcal{A}$ is, as usual in DLs, based on first-order *interpretations* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a nonempty interpretation domain, partitioned into two disjoint sets, $\Delta^\mathcal{I}_O$ of objects, and $\Delta^\mathcal{I}_V$ of values, the latter containing for each datatype T_i a set of values $val(T_i)$. The interpretation function $\cdot^\mathcal{I}$ maps each individual a to $a^\mathcal{I} \in \Delta^\mathcal{I}_O$, each class A to $A^\mathcal{I} \subseteq \Delta^\mathcal{I}_O$, each object property P to $P^\mathcal{I} \subseteq \Delta^\mathcal{I}_O \times \Delta^\mathcal{I}_O$, and each data property U to $U^\mathcal{I} \subseteq \Delta^\mathcal{I}_O \times \Delta^\mathcal{I}_V$. Each literal ℓ is interpreted as the value $\ell^\mathcal{I} = val(\ell)$, each datatype T_i as the set of values $T_i^\mathcal{I} = val(T_i)$, and $\top_d^\mathcal{I} = \Delta^\mathcal{I}_V$. Then, the semantics of expressions is determined as follows:

$$\begin{array}{rcl} (\exists R)^\mathcal{I} & = & \{o \mid \exists o' .\, (o, o') \in R^\mathcal{I}\}, \\ (P^-)^\mathcal{I} & = & \{(o, o') \mid (o', o) \in P^\mathcal{I}\}, \\ (\delta(U))^\mathcal{I} & = & \{o \mid \exists v .\, (o, v) \in U^\mathcal{I}\}, \\ (\rho(U))^\mathcal{I} & = & \{v \mid \exists o .\, (o, v) \in U^\mathcal{I}\}. \end{array}$$

We adopt the *unique name assumption*, i.e., for every interpretation $\mathcal{I}$ and distinct individuals or values c_1, c_2, we have that $c_1^\mathcal{I} \neq c_2^\mathcal{I}$.

We say that $\mathcal{I}$ *satisfies* $\alpha_1 \sqsubseteq \alpha_2$ if $\alpha_1^\mathcal{I} \subseteq \alpha_2^\mathcal{I}$, it satisfies $(\mathsf{disj}\ \alpha_1\ \alpha_2)$ if $\alpha_1^\mathcal{I} \cap \alpha_2^\mathcal{I} = \emptyset$, and it satisfies $(\mathsf{funct}\ S)$ if $S^\mathcal{I}$ is a function (i.e., if $(o, z_1) \in S^\mathcal{I}$ and $(o, z_2) \in S^\mathcal{I}$, then $z_1 = z_2$). Also, $\mathcal{I}$ satisfies $A(a)$ if $a^\mathcal{I} \in A^\mathcal{I}$, it satisfies $P(a, a')$ if $(a^\mathcal{I}, a'^\mathcal{I}) \in P^\mathcal{I}$, and it satisfies $U(a, \ell)$ if $(a^\mathcal{I}, val(\ell)) \in U^\mathcal{I}$.

Query Answering. The main inference service that we consider here is query answering over an ontology. A *conjunctive query* (CQ) q over an ontology $\mathcal{O}$ is an expression of the form $q(\vec{x}) \leftarrow \exists \vec{y} .\, conj(\vec{x}, \vec{y})$, where $conj(\vec{x}, \vec{y})$ is a conjunction of atoms of the form $D(z)$, $S(z, z')$, $z = z'$, where D denotes a class or a datatype and S an object or data property occurring in $\mathcal{O}$, and z, z' are individuals or literals in $\mathcal{O}$ or variables in $\vec{x}$ or $\vec{y}$. Given an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, $q^\mathcal{I}$ is the set of tuples of $\Delta^\mathcal{I}$ that, when assigned to the variables $\vec{x}$, make the formula $\exists \vec{y} .\, conj(\vec{x}, \vec{y})$ true in $\mathcal{I}$. Then, the set $cert(q, \mathcal{O})$ of *certain answers to q over $\mathcal{O}$* is the set of tuples $\vec{a}$ of individuals or literals appearing in $\mathcal{O}$ such that $\vec{a}^\mathcal{I} \in q^\mathcal{I}$, for every model $\mathcal{I}$ of $\mathcal{O}$. In the following we will also make use of unions of CQs (UCQs), which are disjunctions of CQs. *Query answering* is the problem of computing $cert(q, \mathcal{O})$, given a (U)CQ q and a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}$.

The certain answers to an (U)CQ q over a *DL-Lite*$_\mathcal{A}$ ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ can be computed by first *rewriting* q using $\mathcal{T}$ into a new UCQ q', and then *evaluating* q' over $\mathcal{A}$, considered simply as a database [9, 17]. To overcome the limitations of CQs, and express e.g., order conditions on literals, we resort to *EQL-Lite* queries, which are obtained by embedding into an arbitrary SQL query one or more (U)CQs (over a *DL-Lite*$_\mathcal{A}$ ontology) to which we have applied an epistemic operator. We refer to [8] for details, and just note that also *EQL-Lite* queries can be rewritten into queries that can be evaluated over the ABox to produce the certain answers.

Datatypes, version 1.1, and the RDF specification, see `http://www.w3.org/2007/OWL/`.

Ontologies with Mappings to a Database. A key aspect in OBDA is that the database $\mathcal{D}$ accessed through (the TBox of) the ontology might have been developed independently of the ontology itself. Moreover, while the ontology models the domain of interest in terms of (abstract) objects, the database stores data values belonging to concrete data types. To bridge the gap between the data level and the ontology level, we resort to mappings, which intuitively specify how the objects populating the classes and properties of the ontology are constructed from the data values stored in the database. Specifically, we follow the approach introduced in [17], and make use of a set $\mathcal{M}$ of *mapping assertions* of the form $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{y}, \vec{t})$, where $\Phi(\vec{x})$ is an arbitrary SQL query over the database with $\vec{x}$ as output variables. Instead, $\Psi(\vec{y}, \vec{t})$ is a CQ without existentially quantified variables over the TBox $\mathcal{T}$ of the ontology, whose atoms make use of the variables $\vec{y}$ (a subset of $\vec{x}$) and of so-called *variable terms* $\vec{t}$. Variable terms are built by applying *function symbols* to variables or literals, and are used to construct the identifiers of individuals from the data values extracted by Φ from $\mathcal{D}$. An example of mapping assertions for the HGT TBox is shown in Fig. 3, where we have used the function symbols **gene**, **organism**, and **function**, to create instances of the classes $Gene$, $Organism$, and $GeneFunction$, respectively.

`SELECT id, abbrev`
`FROM organism`
`JOIN genes`
`ON abbrev = idorganism`
	$\rightsquigarrow$	$OrganismHasGene($
		gene(id),
		organism$(abbrev))$

`SELECT id, kegg` `FROM genes`	$\rightsquigarrow$	$GeneHasGeneFunction($ **gene**(id), **function**$(id))$ $KEGG($**function**$(id), kegg)$

Figure 3: Extract of the mapping from the HGT-DB database to the *DL-Lite*$_\mathcal{A}$ application ontology.

Intuitively, a *DL-Lite*$_\mathcal{A}$ *ontology with mappings* $\langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ has the same semantics as the ontology $\langle \mathcal{T}, A_{\mathcal{D},\mathcal{M}} \rangle$, where $A_{\mathcal{D},\mathcal{M}}$ is a (virtual) ABox generated by "applying" the mappings in $\mathcal{M}$ to the data in $\mathcal{D}$. Answering a (U)CQ or an *EQL-Lite* query q over a *DL-Lite*$_\mathcal{A}$ ontology with mappings $\langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ can be carried out by first rewriting (the UCQs embedded in) q, and then "unfolding" the resulting query w.r.t. $\mathcal{M}$ using logic-programming techniques. This results in a single SQL query expressed over $\mathcal{D}$ that can be evaluated by a standard commercial relational database engine. We refer to [17] for the formal details.

3.2 Ontology-Based Data Access software

We present now two tools that implement the theory described in the previous section and that are used in the development and deployment of the WONDER project, namely the DIG-QUONTO server and the OBDA Plugin for Protege 3.3.1.

DIG-QUONTO [18] is the DIG 1.1 Interface [6] implementation for the QUONTO reasoner. DIG-QUONTO is able to deal with *ontologies with mappings* (as described in the previous section) in which $\mathcal{D}$ is a relational database accessible through JDBC connectors. The reasoning and mapping techniques for *DL-Lite*$_\mathcal{A}$ implemented in DIG-QUONTO are such that the ABox $\mathcal{A}$ characterized by $\langle \mathcal{D}, \mathcal{M} \rangle$ is never materialized while computing the answers to reasoning task involving $\mathcal{A}$. Instead, using the *RDBMS-ontology mapping* module, DIG-QUONTO is able to rewrite the original queries into SQL queries that are executed by the RDBMS. This feature is specially important in the HGT scenario, as materializing the HGT ABox would imply loading several gigabytes of data in main memory, turning the scenario unfeasible. DIG-QUONTO implements all reasoning services developed for *DL-Lite*$_\mathcal{A}$, which were outlined in the previous section. In order to allow access to functionality available in DIG-QUONTO that is not considered in DIG 1.1, DIG-QUONTO implements the following extensions: (*i*) In order to expose its OBDA functionality, DIG-QUONTO implements the *OBDA Extensions to DIG 1.1* [10, 20], which have as main objective to augment DIG 1.1 with the concepts of *Data Source* and *Mapping*; (*ii*) In order to expose its UCQ answering service, DIG-QUONTO implements the DIG 1.2 specification [19], an extension to DIG 1.1 that provides the ability to pose UCQs to DIG reasoners; (*iii*) In order to expose the functionality not covered by any of the extensions mentioned above, e.g., epistemic query answering, ontology consistency checking, etc., DIG-QUONTO uses ad-hoc extensions to the DIG protocol; a reference to these extensions can be found on the QUONTO website[6].

The *OBDA Plugin* for Protégé 3.3.1[7] [10, 21] is a plugin for the well known ontology editor that provides facilities to develop *ontologies with mappings*. Using the OBDA plugin, we are able to associate a JDBC data source to an existing OWL ontology and to relate SQL queries over the data source to the entities (classes and properties) of the ontology. Moreover, using the plugin we are able to synchronize and query the created domain formalisation for OBDA with any DIG reasoner implementing the OBDA extensions to DIG, e.g., DIG-QUONTO. A key feature, used in the development of the HGT formalisation for OBDA, is the ability of the plugin to pose UCQs and epistemic queries to reasoners supporting those services. UCQs are expressed in a restricted SPARQL syntax, whereas epistemic queries are expressed in EQL syntax. The queries are transformed into DIG requests that are sent to the reasoner and whose response can be visualized with the plugin.

4. WEB-BASED GRAPHICAL QUERYING

Suppose a biologist wants to retrieve all genes of the organism *Neisseria meningitidis* for which horizontal gene transfer is predicted. He can express this request as a CQ over the ontology, which can be written in SPARQL syntax in the OBDA plugin as:

```
SELECT $gene
WHERE {$gene :GeneHasOrganism $org.
       $org :OrganismHasOrganismInfo $info.
       $info :OrganismName 'Neisseria meningitidis'.
       $gene :GeneHasHGTPredictionGene $pred.
       $pred :Prediction 'HGT'}
```

Suppose now that the biologist would like to generalise his request, asking for organisms that either have a name including "`Neisseria`" or have a `GC3stats` value $\geq$ `80`. Since this query contains a disjunction, it goes beyond the expressivity of CQs and since there is also a comparison, it goes beyond the expressivity of UCQs. It can be captured, however, by the following query in EQL, which combines SPARQL and SQL syntax:

```
SELECT stbl.gene
FROM sparqltable
     (SELECT $gene $orgName $gcVal $predVal
      WHERE {$gene :GeneHasOrganism $org.
             $org  :OrganismHasOrganismInfo $info.
             $info :OrganismName $orgName.
             $gene :GeneHasHGTPredictionGene $pred.
             $pred :Prediction $predVal.
             $gene :GeneHasGCstatsGene $gcstats.
             $gcstats :GC3 $gcVal}) stbl
WHERE stbl.orgName LIKE '%Neisseria%' AND
      (stbl.predVal = 'hgt' OR stbl.gcVal > '80')
```

While a diagram like the one in Fig. 2, capturing the classes and properties of the *DL-Lite*$_\mathcal{A}$ ontology, conveys a simple and easy-to-

[6] `http://www.dis.uniroma1.it/~quonto/`
[7] `http://obda.inf.unibz.it/protege-plugin/`

grasp view of the data, the query examples show that a user nevertheless needs to be versatile with technically involved query languages in order to express his information request. With the WONDER system we aim at bridging the gap between the conceptual model and the information demand of users, by offering not only a graphical view of the application ontology but also a graphical query language.

The way in which the WONDER system lets users create graphical queries resembles the one proposed for SQL query formulation in the QBD approach for databases [3] and the stand-alone application OntoVQL for querying OWL-DL ontologies [11], but WONDER uses Semantic Web technologies for interoperability and is web-based, respectively. GRQL [4] and NITELIGHT [23] provide a visual, web-based system, where GRQL allows progressive exploration of an RDF/S class and its properties based on a tree-view and hiding RQL, and the latter allows writing SPARQL queries over an RDF-ontology, but WONDER also allows one to graphically browse the lean OWL-ontology it uses, including the properties, and supports more complex queries. We first describe the theory and then the implementation of the WONDER system.

4.1 WONDER's Formal Foundation for Ontology Browsing and Querying

Accessing information by means of an ontology comprises three activities:

1. browsing the ontology, to understand the structure of the information;
2. formulating a query, to express an information request; and
3. retrieving data that answer the query according to a high-level semantics.

So far, our work focused on support for the first two activities because this is where users encounter a bottleneck when they want to get more out of available data sources. The WONDER Web interface consists of a separate component, called "pane", for each of these activities.

Ontology Browsing. The ontology pane shows the ontology as a graph, which represents the elements of an ontology together with some of the axioms that hold between them. In the graph, there are two kinds of nodes, *class nodes* (visualized as roundtangles), and *attribute nodes* (shown as ovals), and three kinds of edges, *is-a links,* connecting class nodes and shown as thin lines, *role links,* also connecting class nodes, but shown as thick lines, and *attribute links,* connecting class and attribute nodes and shown as thick lines. Role links represent object properties, while attribute links and attribute nodes represent data properties. A role link can connect a node with itself, which is not possible for is-a and attribute links. Nodes, role links, and attribute links are labeled with names of classes, object properties, and data properties, respectively. Table 1 gives an overview of the graphical elements occurring in the ontology pane and their semantics. Note that this representation mentions only the name of a data property, not its range. Similarly, other constraints expressible in *DL-Lite*$_\mathcal{A}$ are not visualized (e.g., functionality or mandatory participation of a class in a property), since a user need not be aware of them to formulate a query.

Since an ontology will often not fit into a single pane, ontologies can be divided into several *pages,* which are shown individually. Conceptually, the pages are glued together by the classes that occur in more than one page. The reader is referred to the online supplementary material to see the full interface.

Query Formulation. The query formulation tool provides visual means to specify CQs extended with comparisons over an ontology. Technically, a WONDER query consists of a *query graph* and a *constraint expression*. The query graph is edited in the *query pane*, and there is a special constraint editor that can be started from the query pane.

The query graph resembles an ontology graph, except for four notable differences: (*i*) there are no is-a links, (*ii*) a class node can be labeled with more than one class name, (*iii*) to each node, a distinct variable is associated, which becomes visible if the mouse hovers over the node, and (*iv*) some of the nodes are highlighted. The semantics of the two kinds of graphs, in spite of their apparent similarity, differs substantially, in that the first represents a set of axioms while the latter a set of atoms and projection variables. In Table 1 we show the possible nodes of a query graph and the links between them together with the atoms they stand for. The variables x, y stand for the variables attached to the nodes appearing in each row. The semantics of a query graph (ignoring for the time being the highlighted nodes) is the conjunction $conj(\vec{z})$ of the atoms corresponding to the elements of the graph. We exploit the apparent similarity between ontology and query graphs to present the task of query construction as one of copying elements from the ontology pane into the query pane and joining them together. The user repeatedly selects several nodes and links in the ontology pane and copies them into the query pane. When selecting a role or attribute link, the two adjacent nodes are selected, too. In the query pane, one can select several class nodes and join them. As a result, these nodes are merged into one, which inherits all the class labels and the links of the selected nodes.

With the Constraint Editor one can impose *elementary constraints* on the query variables, defined in terms of the built-in operators "$=$", "$\leq$", etc. as well as by the SQL string matching operator `LIKE`. They can be combined with boolean connectives in the Constraint Manager, which results in a constraint expression $cons(\vec{w})$, where the vector $\vec{w} \subseteq \vec{z}$ comprises the constrained variables.

Note that constraints involving inequalities and string matching are crucial for the needs of our application, but are not supported by the semantics of CQs over *DL-Lite*$_\mathcal{A}$, which allow only for equality constraints. For this reason, given a conjunction of atoms $conj(\vec{z})$ and a constraint expression $cons(\vec{w})$, we define their semantics as that of an *EQL-Lite* query, which imposes constraints on top of the certain answers retrieved by a *DL-Lite*$_\mathcal{A}$ CQ. Let $\vec{x}$ be a vector comprising the variables corresponding to the highlighted nodes in the query pane, which determine the output of the entire *EQL-Lite* query, and let $\vec{y}$ be a vector comprising the variables in $\vec{x}$ and in $\vec{w}$. These are the distinguished variables of the CQ evaluated under *DL-Lite*$_\mathcal{A}$-semantics. The *EQL-Lite* query, where $conj(\vec{z})$ is written in SPARQL notation and in $cons(\texttt{stbl.}\vec{w})$ all variables are prefixed by `stbl`, is as follows:

$$\begin{array}{l}\texttt{SELECT stbl.}\vec{x}\ \texttt{FROM sparqltable}\\ \quad\texttt{(SELECT}\ \vec{y}\ \texttt{WHERE \{}conj(\vec{z})\texttt{\}) stbl}\\ \texttt{WHERE}\ cons(\texttt{stbl.}\vec{w})\end{array} \tag{1}$$

Due to the semantics of *EQL-Lite*, the results of the query above are obtained by (*i*) computing the certain answers for the CQ $q(\vec{y}) \leftarrow conj(\vec{z})$, (*ii*) filtering the resulting tuples according to the constraint $cons(\vec{w})$, and (*iii*) projecting onto $\vec{x}$. When editing the query graph and the constraints, the corresponding *EQL-Lite* query (1) is shown in a special window in the menu on the right-hand side.

The formulation of a query with WONDER still requires some effort, which should not be lost when creating a new query. Therefore, any query can be stored under a name and description. A user also can load a stored query into the query pane, modify it, and store it under another name. The possibility to load stored queries provides a functionality similar to the one of canned queries.

Element name	Graphical Representation	Semantics of ontology elements	Semantics of building blocks of query graphs
Class node	C	C	$C(x)$
	C, D		$C(x), D(x)$
Is-a link	C — D	$C \sqsubseteq D$	
Attribute node and link	C — A	$\delta(A) \sqsubseteq C$ $\rho(A) \sqsubseteq \top_d$	$C(x),\ A(x,y)$
Role link	C — P — D	$\exists P \sqsubseteq C$ $\exists P^- \sqsubseteq D$	$C(x),\ R(x,y),\ D(y)$

Table 1: Name and graphical representation of ontology elements and of building blocks of query graphs in WONDER.

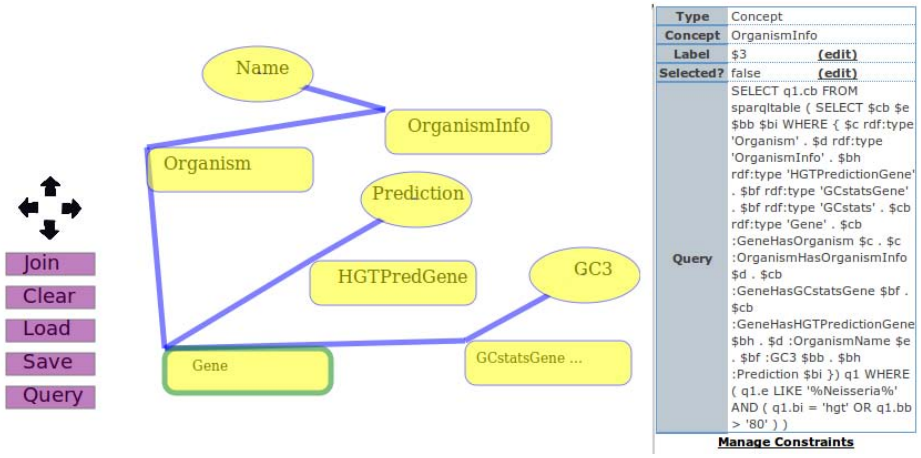


Figure 4: Query to retrieve the genes of *Neisseria* spp. that have a GC3 content > 80 or are predicted to be horizontally transferred. The textual version of the graphically constructed query (on the right) is generated automatically by the WONDER system.

Query Execution. When the query in the query pane has been executed, the results can be retrieved either in *browse mode* or in *batch mode*. In browse mode, the answer tuples can be viewed on web pages, each containing a predefined number of tuples, which is useful when the number of answers is low. In batch mode, the entire set of answer tuples can be downloaded as a CSV file.

4.2 WONDER Implementation

In this section, we first introduce the high-level architecture of the WONDER system and then outline the implementation of the query environment.

Architecture. The WONDER system extends the OBDA architecture presented in Fig. 1 by using as client a web application that provides a visual query environment exploiting the services provided by DIG-QUONTO (see Fig. 5). The web application is divided into two main parts: the server-side and the client-side, which communicate with each other using AJAX. This technology allows to asynchronously transfer data from the server to the client. WONDER uses an additional database for storing meta-information, such as registered users and their saved queries.

The WONDER server is a semantic-aware application written using the Java EE framework, which is responsible for providing interaction with DIG-QUONTO and access to the metadata. It is semantic-aware in the sense that it holds the ontology (OWL file) and the set of semantic mappings (OBDA file) required by OBDA. These files are used at query execution time for accessing the query answering service provided by DIG-QUONTO. The communication between the server and DIG-QUONTO is made over HTTP by using the DIG protocol. The query answering service returns the result set computed from the execution of the translated query over the relational database. The results are stored in a CSV file that logged users can download. During this process, the first hundred tuples are sent back to the client, which is in charge of presenting them. The advantage of this feature is to reduce latency of transferring data over the Internet.

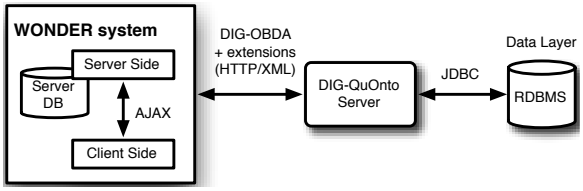


Figure 5: The architecture of the WONDER system.

Query Environment. The client-side is responsible for providing the graphical representation of the ontology and a visual query environment for creating the queries.

There are different technologies that allow for the creation of complex diagrams, together with the support for user interaction; the most interesting being Macromedia Flash, HTML Canvas, and Scalable Vector Graphics (SVG). We based our application on SVG, an XML-based language for defining vector graphics. The reasons for this choice are that SVG is a W3C-recommendation, it allows the injection of JavaScript code for handling user interaction, and, moreover, images can be modified at runtime by using the standard DOM interface.

One of the biggest issues faced during the development of this query environment is the decision on how to extract the graphical representation of the ontology and convert it into an SVG image. We inspected two possible solutions: dynamically generating the diagram from the ontology and relying on an external tool that helps the user in drawing such a diagram. On the one hand, the former solution is the most attractive one since it does not require, ideally, any user involvement. On the other hand, it is not always possible to dynamically create a schema without overlapping figures and, generally, user involvement is needed to improve the diagram anyway. Therefore, the WONDER system relies on an external tool, which produces an XML file containing information on how to visualise the ontology. For this we have defined an XML Schema language called OWLX, which provides graphical properties (e.g., colour, positioning) for each element that has to be represented. The advantage of this solution is that the SVG can then be dynamically generated by applying an XSLT transformation to the OWLX file. This graphical information could have been stored inside the OWL file describing the ontology rather than in a separate file, but then the XSLT transformation would have been more complicated and the OWL file would have contained information outside the domain of interest, which was deemed undesirable.

As soon as the ontology is created and opened, the SVG diagram is inserted in the web page and sent to the browser. At this point, users can interact with the schema and select elements that are interesting for their queries and move them to the Query Pane. The Query Pane is an SVG-based environment, which aims at giving users the possibility to visually refine their queries. All the operations defined in the query language have been implemented by means of JavaScript functions. Moreover, the queries created through this interface are stored, together with layout information, in a datastructure, which is used for creating a textual representation of the designed query at query execution time. Finally, this textual query is sent to the server-side for execution and the results are shown in an HTML table in the results pane.

5. EVALUATION

In this section we compare the WONDER system with the original HGT-DB, and reflect on the usage of the chosen Semantic Web technologies.

Web-based OBDA-enhanced Access to Biological Databases. Although both the HGT-DB and the WONDER system's interfaces are web-based, the latter makes use of SVG, a W3C recommendation, as a component for *graphical* browsing and querying, whereas the former uses the rigid HTML+scripts like other online biological database. In addition, the WONDER icons are not mere figures but rely on the formal foundation (see Sections 3.1 and 4.1) and thereby have an unambiguous meaning thanks to the correspondence with the underlying *DL-Lite*$_\mathcal{A}$ application ontology stored as OWL file, and with the SPARQL and *EQL-Lite* queries, which are automatically generated and thus syntactically correct.

For the domain experts, the principal advantages of the differences in the used technologies are that the WONDER system gives the freedom to construct any query deemed necessary, yet also offers the feature of template queries, whereas the 'standard' way of accessing biological databases offers only canned and precomputed queries. Furthermore, when domain experts create a new type of query, they do not have to contact the database administrator first, but simply can do it through the graphical query interface, thereby having greater control over the data source. For instance, when we suggested several queries and asked our domain experts for sample queries irrespective of the current interface, all queries, including those mentioned in Section 2, were beyond the current HGT-DB's web interface possibilities. Formulating such complex queries was greatly facilitated by having communicated the conceptual model that provides a succinct and clear overview of *what* kind of data is stored in the database, instead of bothering domain experts with *how* it is stored and how to get data out of the database. Concerning the types of queries that can be posed through the graphical interface in WONDER, the main additional ones, compared to the HGT-DB web interface, involve flexible *projections of multiple attributes together with selections of values*, and *modifications of the* `WHERE` *clause* through the constraint manager (e.g., changing `AND` for `OR` and nesting of constraints). Without these options, the domain expert is left to browse the full table and manually examine the data, which is a laborious task and error prone.

Assessment of Semantic Web Technologies. It is well-known that scaling up Semantic Web technologies to large ontologies and/or large amounts of instances is not trivial. The original HGT-DB was about 4GB, with many data items for about roughly 500 organisms (a tuple each), which have together 1.7 million genes (also a tuple each) and, depending on the query, for which the join has to be computed. Performance results for the WONDER system are as follows (the HGT-DB is hosted on a Windows XP PC with an Intel Core Duo 1.66 Ghz proc., 2 Gb of RAM and running Oracle 10g):

Query (see Section 2)	*A*	*B*	*C*	*D*
Tuples retrieved	226	392	13	3119
WONDER *interface (ms)*	2826	48436	60767	50562

The corresponding SPARQL queries and a performance comparison of them with a bare DIG-client are included in the supplementary material online (see footnote 3). Notable is that the additional semantic usability layer of WONDER has an insignificant effect on overall performance.

In addition to secondary storage for the instances in the ABox, another Semantic Web performance improver is built into the current realisation of OBDA: the use of an ontology language of relatively low expressivity (recollect Section 3.1). It is indeed the case that not all constraints of the original first order logic theory (ORM conceptual data model; see Section 2) can be represented in the *DL-Lite*$_\mathcal{A}$ version used in the WONDER system, but fancy constraints, such as value restrictions, are enforced in the database already anyway (hence, not representing them in the TBox will not lead to a state that is inconsistent with the semantics of the domain). Yet at the same time, *DL-Lite*$_\mathcal{A}$ is expressive enough regarding existential quantification and functionality of object and data properties to offer some reasoning services compared to a mere SPARQL query over a bare taxonomy in an OWLized .obo file[8], which is relevant in particular for data integration under incomplete information. Data integration is a major drive for bio-ontologists [22, 24], which the WONDER infrastructure thus can cope with; this will be demonstrated at a later date for the HGT-DB scenario and various extensions (e.g., the HEG-DB, species taxonomy).

[8] `http://bioontology.org/wiki/index.php/OboInOwl:Main_Page`

An indirect beneficial effect of using Semantic Web technologies was the improvement of the database itself. Aside from creating views to make more transparent mappings and adding indexes to the tables to achieve better query answering times, we also transferred the data from MySQL to Oracle, which had a major beneficial impact on the performance of the joins of the full organism and gene tables. Although this database engineering fine-tuning could have been done when the HGT-DB was developed, it was not deemed essential. However, when scaling up Semantic Web technologies to deal with large amounts of data, considering performance optimizations is a significant parameter for success. Overall, we now not only have similar or better response times with the semantically enhanced HGT-DB in the WONDER system compared to the plain HGT-DB web-interface, but also offer good performance with queries that were hitherto not possible. Taken together, the HGT-DB enhanced with the WONDER system empowers the domain experts to carry out more sophisticated *in silico* experiments more quickly.

6. CONCLUSIONS

We have introduced the Web-ONtology baseD Extraction of Relational data (WONDER) system, which builds upon the theory, technology, and implementation developed for Ontology-Based Data Access. This extension implements graphical ontology browsing, query formulation, and query execution in a Web browser. The graphical browsing and query language enjoys a rigorous formal characterisation and uses a coupling with an OWL file on the one hand and CQs (in SPARQL syntax) and *EQL-Lite* queries for the DIG-QUONTO reasoner on the other hand, thereby ensuring correct formulation and evaluation of the queries. This WONDER system not only meets the scalability requirement, but also the usability requirement to allow domain experts to query through a web browser the database without the need to learn SPARQL or *EQL-Lite*. We have evaluated this infrastructure with the horizontal gene transfer genomics database HGT-DB. We are currently planning feature extensions, such as query completion and more sophisticated ontology browsing, and use case extensions in the direction of database integration.

Acknowledgements. We thank our domain experts Santiago Garcia-Vallvé and Mark van Passel.

7. REFERENCES

[1] A. Acciarri et al. QuOnto: Querying Ontologies. In *Proc. of AAAI 2005*, pages 1670–1671, 2005.

[2] A. Amoroso, G. Esposito, D. Lembo, P. Urbano, and R. Vertucci. Ontology-based data integration with MASTRO-I for configuration and data management at SELEX Sistemi Integrati. In *Proc. of SEBD 2008*, pages 81–92, 2008.

[3] M. Angelaccio, T. Catarci, and G. Santucci. QBD*: A graphical query language with recursion. *IEEE Trans. on Software Engineering*, 16(10):1150–1163, 1990.

[4] N. Athanasis, V. Christophides, and D. Kotzinos. Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL). In *Proc. of ISWC*, pages 486–501, 2004.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[6] S. Bechhofer, R. Möller, and P. Crowther. The DIG description logic interface. In *Proc. of DL 2003*, volume 81 of *CEUR,* `ceur-ws.org`, pages 196–203, 2003.

[7] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of KR 2006*, pages 260–270, 2006.

[8] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of IJCAI 2007*, 2007.

[9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[10] D. Calvanese and M. Rodríguez. Towards an open framework for Ontology Based Data Access with Protégé and DIG 1.1. In *Proc. of OWLED 2008*, 2008.

[11] A. Fadhil and V. Haarslev. OntoVQL: a graphical query language for OWL ontologies. In *Proc. of DL'07*, 2007. Bressanone, Italy.

[12] S. Garcia-Vallvé, E. Guzman, M. Montero, and A. Romeu. HGT-DB: a database of putative horizontally transferred genes in prokaryotic complete genomes. *Nucleic Acids Research*, 31(1):187–189, 2003.

[13] C. M. Keet, R. Alberts, A. Gerber, and G. Chimamiwa. Enhancing web portals with Ontology-Based Data Access: the case study of South Africa's Accessibility Portal for people with disabilities. In *Proc. of OWLED 2008*, 2008.

[14] C. Nyulas, M. OConnor, and S. Tu. DataMaster – a plug-in for importing schemas and data from relational databases into Protégé. In *Proc. of Protégé 2007*, 2007. Stanford Medical Informatics.

[15] M. W. v. Passel et al. Phylogenetic validation of horizontal gene transfer? *Nature Genetics*, 36(10):1028, 2004.

[16] M. W. v. Passel et al. Deltarho-web, an online tool to assess composition similarity of individual nucleic acid sequences. *Bioinformatics*, 21(13):3053–3055, 2005.

[17] A. Poggi et al. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.

[18] A. Poggi, M. Rodriguez, and M. Ruzzi. Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé. In K. Clark and P. F. Patel-Schneider, editors, *Proc. of OWLED 2008 DC*, 2008.

[19] Racer Systems GmbH & Co. KG. Release notes for racerpro 1.9.2 beta. Website, Last access, July 2008. `http://www.sts.tu-harburg.de/~r.f.moeller/racer/Racer-1-9-2-beta-Release-Notes/release-notes-1-9-2se8.html`.

[20] M. Rodríguez-Muro and D. Calvanese. An OBDA extension to the DIG 1.1 Interface. Website, July 2008. `http://obda.inf.unibz.it/dig-11-obda/`.

[21] M. Rodríguez-Muro, L. Lubyte, and D. Calvanese. Realizing Ontology Based Data Access: A plugin for Protégé. In *Proc. of the ICDE Workshop IIMAS 2008*. IEEE CS Press, 2008.

[22] A. Ruttenberg et al. Advancing translational research with the Semantic Web. *BMC Bioinformatics*, 8(Suppl 3):S2, 2007.

[23] P. R. Smart et al. A visual approach to semantic query design using a web-based graphical query designer. In *Proc. of EKAW 2008*, 2008.

[24] B. Smith et al. The OBO Foundry: Coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25(11):1251–1255, 2007.