Towards an open framework for Ontology Based Data Access with Protégé and DIG 1.1

Mariano Rodriguez-Muro, Diego Calvanese

Faculty of Computer Science, Free University of Bozen-Bolzano
rodriguez, calvanese @inf.unibz.it

Abstract. Ontology Based Data Access (OBDA) is concerned with the problem of mediating access to data sources through an ontology describing the information represented therein. As OBDA reaches into the mainstream of Semantic Web technology, a need for standardization is emerging. On the one hand, early availability of common methodologies to access and interact with systems that offer OBDA functionality would prevent the multitude of API's and protocols that commonly appear when a new breed of systems becomes available and which complicates systems interoperability. On the other hand, a collection of software tools that provide functionality that is required by most OBDA systems could help minimizing duplication and redundancy across implementation efforts for these systems. This article presents the initial components of an Open Framework for OBDA that targets the previous issues; the first issue is addressed with an OBDA extension to the DIG 1.1 Interface; the second one with a plugin for the ontology editor Protégé that provides core OBDA functionality. Both components are open, extensible, and designed to accommodate the different techniques that are used in OBDA today.

1 Ontology Based Data Access, the common features

In OBDA, the aim is to use an ontology describing a domain of interest to mediate the access to typically large data sources storing information about that domain. It is a field that has recently caught the attention of researchers working in different areas, such as databases, description logics, OWL, and the semantic web. The added value of OBDA, w.r.t. accessing a data source directly, is, on the one hand, that the ontology provides a semantic account of the information stored in the data source. On the other hand, the answer to user queries may be enriched by exploiting the constraints expressed by the ontology, thus overcoming incompleteness that may be present in the data.

As the technology pushes itself into the mainstream of Ontology and Semantic Web technology, a need for standardization is emerging, motivated by the plethora of options that can characterize OBDA system components. The prototypical structure of an OBDA system comprises: (*i*) a semantic layer expressed in the form of an ontology in some ontology language, (*ii*) one or more possibly heterogeneous data sources, (*iii*) a set of *mappings* expressing semantic relationships between data in the sources and entities of the semantic layer, and (*iv*) a client-system interaction layer through which clients can manipulate or query the system. Each of the these components can be realized in different ways, depending on the objective of the OBDA system. For instance, *ontologies* can be expressed in different languages (e.g., OWL, rules, datalog programs, etc), *mappings* can be given with different syntaxes (e.g., graphical, complex query based mappings) and semantics assumptions (e.g., sound/complete sources), and

data sources can be of different nature (e.g., databases, documents, repositories) or have different access features (e.g., online vs. offline, always vs. intermittently available).

When building OBDA systems, implementors commit to a certain characterization of the OBDA setting. We claim that a considerable amount of implementation efforts could be reduced if a proper framework for OBDA systems is established. More importantly, having a common framework shared by many implementations of OBDA systems, would greatly ease the job of porting an already built solutions to alternative OBDA settings. The work presented here is our attempt towards the realization of such a framework. We targeted two key aspects of OBDA systems which we believe are the ones that can profit the most from standardization, that is, the client-system interaction layer and software support for the configuration and manipulation of the rest of the components of OBDA systems, i.e., data sources and mappings editors tools.

In order to maximize the chance of reaching the goals of the OBDA framework and based on observations regarding the previously identified OBDA architecture and the previous developments of semantic web technology, we formulated the following requirements for the components of the framework: (*i*) To be based on already available and used standards; (*ii*) Separation of the general OBDA related functionality from the details related to specific characterizations of the OBDA setting; (*iii*) Ability to handle any OBDA characterization, including the ability to handle different syntaxes and semantics for each of the components of an OBDA system; (*iv*) Clear and accessible means to incorporate new characterizations of the OBDA setting into the framework.

2 OBDA extensions to DIG 1.1

The first component of the framework is an OBDA System interaction layer in the form of an extension to the DIG 1.1 DL reasoner interface [2]. The extension's goal is to provide a channel through which OBDA clients/servers can access/provide OBDA specific functionality. We built this interaction layer on top of the DIG 1.1 interface due to one main reason, this protocol is currently implemented in many DL/Semantic Web related project, including all major DL reasoners. We put special focus on providing a straightforward extensibility mechanism, which is compatible with the current DIG 1.1 interface. The extension presented here is implemented in the DIG server for the latest version of the QuOnto [1] system and in the OBDA plugin for Protégé (see Section 3). Because of space limitations, we do not fully describe the extension here. Instead, we point to the extensions publication website [5] and we proceed with a short description of the core components of the OBDA extension.

Data Sources A data source represents an external entity, identified by a URI, that stores data relevant to the domain of the ontology. It is expected that the OBDA Systems will access these sources to retrieve the data that populate the instance level of the ontology. The extension makes no assumptions regarding the offline/online accessing of these sources. Associated to each data source there is a set of connection parameters that provide the reasoner with the required information to access the source. Each parameter is identified by a URI and has associated a value.

Different types of data sources have different access methods, assumptions or options, therefore require different access parameters. To cope with this situation, the OBDA extension provides a mechanism based on URIs to define which parameters are associated to which data sources. Associated to each data source there is a datasource-type-URI. Similarly, for each datasource-type-URI there is a set of parameter-URIs,

which represent the parameters that are relevant to access the sources of that datasource-type-URI. Implementors of OBDA Systems that support a specific datasource-type-URI can assume that clients will define all the values for each of these parameters. For details on how to define these URIs, see section *modules*.

Mappings In the OBDA extension to DIG 1.1, a mapping specifies the relationship between the entities of the ontology and the data in a source. Based on [3], the OBDA extension for DIG 1.1 defines a mapping as a statement of relationship between a query over the source and a query over the ontology. We note that here, the term *query* is used in the most general sense of the word, as it stands for *any* kind of computation. This way of modeling mappings can account for any type of mapping used in previous approaches to map schemes. Thus, we believe that it is well suited for the OBDA framework.

On the other hand, this modeling of mappings carries a considerable amount of variability that comes in the form of different semantics for the mappings and different languages (syntax and semantics) for the source and ontology queries. To cope with these issues, the OBDA extension to DIG 1.1 does not fix any of these parameters. Instead, we fix a) the general structure of the messages that will describe mappings, b) the operations that will be available over these mappings and c) a mechanism for implementors to fix the specific semantics and syntax that they support. For more details on how to define specific types of mappings, see section *modules*.

Modules The creation of modules is the mechanism that allows for the definition of specific types of data sources and mappings. In a module implementors make publicly available the following information: (*i*) the new `datasource-type-URI` that identifies the new data source type, (*ii*) the URIs of the parameters that are associated to the new `datasource-type-URI` (*iii*) enough details about the data source type and its parameters to understand their intended use, (*iv*) the semantics of the mappings used with the new data source type, (*v*) the relation of valid ontology and source queries for the new data source type as well as their intended semantics, and last (*vi*) a XML Schema (XSD) file that extends the base OBDA-DIG extension XSD file with the definition of new XML elements that allow for the expression of the newly introduced *source* and *ontology* queries.

A module for RDBMS sources. In addition to the description of the OBDA extensions to DIG, we provide a module that gives support for RDBMS data sources. The module defines the query over the source as an SQL query, with the traditional semantics. The query over the ontology is in the form of a `<retrieve>` element from the DIG 1.2 proposal in [4]. The full description of this module can be found in [5].

3 An open OBDA component editor, the OBDA Plug-in for Protégé

The *OBDA plugin* is an extension for Protégé that enables the popular ontology editor to function as an editor for the components of an OBDA system. Specifically, the OBDA plugin provides the following functionalities: (*i*) the ability to define a/multiple data source(s); (*ii*) the ability to create/delete/duplicate source-ontology mappings; (*iii*) the ability to pose *data centric* queries to reasoners; (*iv*) facilities for inspecting/querying the data sources directly, and (*v*) permanent storage support for the OBDA information designed with the plugin. All this is done through a set of tabs and controls introduced to Protégé's GUI by the plugin plus modification of some core mechanisms of Protégé.

Moreover, the OBDA plugin extends Protégé's DIG 1.1 reasoner synchronization procedure to include the new, OBDA specific, information entered with the plugin.

In order to do this, the OBDA plugin hooks into Protégé's synchronization stage and modifies it to include the new information about data sources and mappings entered by the user. The translation of this information into DIG messages is done using the OBDA extension to DIG presented in the previous section. The procedure is transparent to the user, who can keep using Protégé's facilities in the usual way.

The plugin provides a new *Query Tab*, which is used to pose *data centric* queries to the reasoners. Responses are then presented to the user for visualization and manipulation (i.e., saving, exporting, importing). At the moment, the plugin provides the ability to pose Union of Conjunctive Queries (UCQs) expressed in SPARQL syntax, which get translated into DIG 1.2 UCQ queries [4] and sent to the reasoner. The *Query Tab* is extensible, allowing implementors to incorporate new types of queries. In order to do so, implementors simply need to access the plugin's Java API and provide DIG translators and editors for the new types of queries

At the moment, the plugin offers facilities to edit data sources and mappings as defined in the *RDBMS module* presented earlier. Nevertheless, the plugin is not fixed to this kind of data sources and mappings; implementors can extend the plugin's editor tabs by overriding key classes of the plugin source code with their own editors and DIG translators. Due to space restrictions we do not fully describe the plugin, nor provide screenshots. Instead we point to the OBDA plugin's website [6], where documentation, binaries, and manuals can be found and downloaded. The source code of the plugin will be released with an open source license shortly on that website.

4 Conclusion and future steps

We are currently working towards extending the framework's components to the next generation of tools and protocols present in the stage of semantic web technologies. Based on the implementation experiences for the OBDA extension to DIG 1.1, we are updating our OBDA extension to the new OWLLink Interface (previously DIG 2.0) and at the same time aligning the specification to the upcoming OWL 2.0. We are also in the process of developing an OBDA-API for the OWL-API, in order to provide a solution to the issues that rise when using DIG 1.1 and large amounts of data. At the same time we are working towards Protégé 4.0 and Neon Toolkit versions of the OBDA Plugin.

References

1. A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. Quonto: Querying ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 1670–1671, 2005.
2. S. Bechhofer. The DIG Description Logic interface: DIG/1.1. Technical report, University of Manchester, 2003. http://dl-web.man.ac.uk/dig/2003/02/interface.pdf.
3. M. Lenzerini. Data integration: A theoretical perspective. In L. Popa, editor, *PODS*, pages 233–246. ACM, 2002.
4. 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`.
5. M. Rodríguez-Muro and D. Calvanese. An OBDA extension to the DIG 1.1 Interface. Website, July 2008. `http://www.inf.unibz.it/~rodriguez/OBDA/dig-11-obda/`.
6. M. Rodríguez-Muro and D. Calvanese. The OBDA plugin for Protégé. Website, July 2008. `http://www.inf.unibz.it/~rodriguez/obda/protege-plugin/`.