Providing End-user Oriented OBDA over Existing Relational Databases

Aleksandar Jeremić, Milan Čeliković, Vladimir Dimitrieski, Slavica Kordić, Ivan Luković
Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
{jeremicaleksandar, milancel, dimitrieski, slavica, ivan}@uns.ac.rs

***Abstract*—Ontology-based data access (OBDA) is one possible approach to data management. In OBDA, the data is stored in external sources, such as the databases or files. The end-user view of the data is realized though ontologies, and queried as such. In the case of relational data sources, ontologies abstract away the technical details relating to the relational data model, and provide end-user with a more intuitive, object-oriented view of the data. The data layer of such systems may use many heterogenous data storage mechanisms. Relational databases are the most common, but other data storage mechanisms such as NoSQL databases, structured and semi-structured files may be used as well. The view layer of OBDA system is usually implemented though OWL ontologies, queried via SPARQL language. OWL ontologies in OBDA systems only serve to provide the conceptual view of the data stored in the database. Therefore the ontology only contains concept definitions, and not the data itself. This separation of the data (in the database) and its descriptions (in the ontology) imposes some challenges in the development of OBDA systems. For example, mappings need to be created that will connect data definitions in the data sources to the elements of the ontology. Also, using these mappings to accurately and efficiently answer end-user queries is an intricate task in itself. In this work, we present an automated system for bootstrapping a generic OBDA architecture, based on the *ontop* tool. Our system is able to generate the ontology that will serve as a conceptual view of the provided data sources, and map the data in the data sources to the elements of the generated ontology. An open-source tool *ontop* is used to subsequently provide a SPARQL endpoint for quering the data over the generated ontology. We put emphasis on generating an ontology that will describe the underlying data in a best possible way, by using database constraints to refine concept descriptions. Also, we provide a web-based front-end application, to aid in understanding and verifying the generated ontology by the end-user.**

I. Introduction

Relational databases are commonly used in modern information systems as a way to manage large amounts of data. Strong theoretical foundations and the maturity of the technologies make these systems reliable, and, despite new trends emerging, it is unlikely they will soon be replaced by something else. Alternative approaches such as NoSQL databases and data lakes find use in some specific scenarios, although, for the majority of the use cases, relational databases are still the primary choice for data storage.

Instead of replacing existing data management systems, different technologies that provide additional capabilities on top of the existing systems should be considered. One such approach is OBDA, a data management method focused on providing views of the data though ontologies. Usually, OBDA is implemented over an existing relational database, but it can be implemented over other types of data storages, too. In this work, we specifically focus on the variant of OBDA systems deployed over an existing relational database.

One OBDA system, therefore, provides a view of the data stored in the database through ontology, and allows for exploration of the data by querying the ontology. This approach to data management is beneficial due to the structure of the data model provided by ontologies. Real life entities are represented through classes, and relationships between them are described using object properties. Characteristics of objects and relationships are expressed through data properties. Ontologies also provide an intuitive mechanism for describing hierarchies that exist between real-life entities, by means of subclass relationships. Overall, the described way of structuring data is simpler and more intuitive to a human mind, compared to the flat structure of tables connected by foreign keys, which is characteristic for the traditional relational systems. For example, using ontologies as views over the existing relational data can be a good way to help IT-inexperienced users familiarize themselves with the data and its structure.

Usually, these views are implemented as OWL ontologies, and SPARQL language is user to query the data. This naturally introduces many challenges in the process of deployment of one OBDA system. For example, a mechanism to map database structures to elements of the ontology is necessary. Also, SPARQL queries expressed over the ontology need to be rewritten into a proper SQL form that shall be evaluated over the underlying database.

A large portion of the work relating to the OBDA deployment can be automated. For example, the task of creating the ontology can be automated, i.e., the ontology can be automatically generated based on the data definitions present in the database. This process is called *bootstrapping*. Usually, the ontology generated this way only serves as a starting point in development of a more expressive ontology that would describe the underlying data better. The issues of automatic generation of an expressive ontology, as well as the automated deployment of OBDA systems over an existing database, are the main issues this paper is concerned with.

With the goal of easing the development of one OBDA system, we propose one possible architecture of a system that deploys OBDA over an existing database, and we also provide the tools necessary to accomplish this task. These tools are provided as a part of one OBDA-deploying system, and these tools handle all the tasks necessary to bootstrap one OBDA system. The results presented in this work should be of use to organizations aiming to

implement OBDA into their business processes. An important part of our solution is the web-based client application, which should make it easy for IT-inexperienced users to operate the system, making OBDA accessible to a wider audience. The developers of OBDA tools might also find our results beneficial to their work, helping them create better software in the future, and improve on the existing tools.

II. Related Work

A general purpose architecture for OBDA systems consisting of a data source (or multiple data sources), ontology with mappings, and a query engine has been proposed in the earliest of works relating to the field of OBDA. The approach we advocate for is more akin to what has been presented in [1]. The system proposed in [1] takes in data definitions present in the database as an input, generates the appropriate ontology and mappings, and subsequently provides querying services over the generated ontology. Aforementioned paper does not deal with some more advanced methods applicable to ontology generation, such as using database constrains (other than primary and foreign key constraints), to more appropriately map relations to classes. Similar approach has been proposed in [2], with the same shortcomings. Solutions to the ontology generation problem also exists in the form of various *Protégé* plugins, although the plugin nature of these tools makes them difficult to implement into an automated workflow that we are suggesting in this work. The *ontop* tool offers ontology generation as a way to bootstrap the development of OBDA systems, and, being available through a command line interface, it is fit to easily incorporate into the automated workflow. In our work, *ontop* will be used to provide a SPARQL endpoint of the OBDA system. Still, *ontop*'s ontology generation capabilities are very basic and prove unsatisfactory for creation of a ready-to-use OBDA system. Another solution exists in a form of *Optique Platform*[3], although the proprietary nature of the project makes it unfit for further advancements coming from the userbase. Of significance is also the work presented in [4], but the solution in question focuses mostly on enriching the generated ontology with external ontology resources. End-user applications that would ease the use of the system also weren't considered in most of the aforementioned works.

The system we propose should address the shortcomings of the aforementioned works. We aim to provide more advanced ontology generation by relying on database constrains, as well as the end-user oriented application that should aid users (both engineers and domain specialists) in using the system. Our system shall also remain open-source, allowing for further improvements coming from the community.

III. Proposed Architecture

The architecture of the system we propose does not differ too much from the architectures similar systems for automated OBDA deployment. This architecture is shown in Fig. 1.

As shown in Fig. 1, the data is stored in an existing database. This database also stores the data definitions and the database constraints in the data dictionary, and these are fed to the *generator* tool. The *generator* tool is

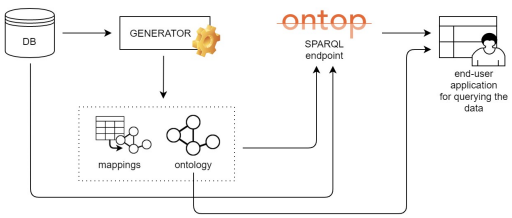

Figure 1. System architecture

responsible for generating the ontology based on the data definitions and constraints specifications. The mappings between the database structures and ontology elements are also created by this tool. The *generator* tool has been developed specifically for the system presented in this work.

Once the ontology and the mappings have been generated, they are forwarded to the *ontop* tool to bootstrap the OBDA system over the data sources. *Ontop* tool is used to provide SPARQL endpoint that will allow for querying the data within the database, with reliance on the generated ontology and the mappings. Finally, the entire system is used by the end-users through our web-based client application for querying the data. This application provides a view of the ontology via a graph and allows for visual query formulation, allowing for easy querying of the data.

IV. Solution

In this section, we will briefly explain the technologies used in our system, according to the architecture presented in the previous section. We shall also give special attention to the two key components of our system – the *generator* tool and the web-based client application.

A. OWL

OWL[5] is a family of ontology description languages, proposed by World Wide Web Consortium in 2004. It was originally designed to fulfill the requirements of the Semantic Web for an ontology language with formally defined semantics, to allow for machine processing of information available on the web. Nowadays, however, OWL is used in many different problem domains. It is the default language of many ontology modeling tools and semantic reasoners.

There exist various different OWL syntaxes. Of interest for this work is the default RDF/XML syntax, which describes ontologies in the form of RDF graphs contained within RDF documents. In the system presented in this work, these RDF documents containing ontology descriptions are shared among system components.

B. R2RML

R2RML[6] is a declarative language for expressing mappings between relational databases and RDF documents. Every mapping is also an RDF graph, specified in Turtle RDF syntax, or some variant of it. Key components of every mapping are the source (specification of the database structures to map to RDF elements) and the target (specification of RDF elements to map to).

In the system presented in this work, we use mappings specified in *Quest* syntax, characteristic for the *ontop* tool.

In the following text we refer to files containing these mappings as *OBDA files*, due to the file extension associated with them.

C. SPARKQL

SPARQL[7] is a language for querying RDF graphs. The queries are expressed as collections of patterns describing RDF triples. The set of all RDF triples of an RDF graph that satisfy the query pattern compose the query result.

As we store ontologies in the form of RDF documents, it is possible to use SPARQL to express queries over these ontologies.

D. ontop

Ontop[8] is an open-source OBDA tool for providing ontology-based views over relational databases. Provided an OWL ontology, a mapping file and a connection to a database, *ontop* is able to process incoming SPARQL queries and efficiently translate them to appropriate SQL form that can be then evaluated over the data sources. *Ontop* also handles the tasks of fetching the query results and transforming them into the appropriate form based on the original SPARQL query.

Besides handling SPARQL query translation, *ontop* possesses various other functionalities that can aid in OBDA development, such as basic ontology bootstrapping and ABox materialization.

Ontop is available in the form of a command-line application, as well as a *Protégé* plugin. In the system presented in this work we use the *ontop* command-line tool to provide a SPARQL endpoint for querying the data.

E. Generator tool

The *generator* tool creates the ontology and the mappings based on the data structures and constraint definitions present in the database.

To ensure the ontology and mappings are properly generated, the table definitions present in the database must satisfy the following prerequisite. That is, the primary key of each table must either be simple, consisting of a single attribute, or be a composite key consisting of exactly 2 attributes which are both foreign keys referencing to other tables.

Our approach to generate the ontology and the mappings is similar to the approach presented in [9], with some modifications. Specifically, in order to generate the ontology and the mappings based on the database schema definitions, the following rules are applied:

1) Every table with a simple primary key (primary key composed of only one attribute) is mapped to a class definition, and the primary key attribute of the table is used for construction of identifiers for the objects of this class.
2) Every table with a composite primary key composed of 2 attributes, which are also foreign keys (i.e. references to other tables), is mapped to an object property connecting objects of classes representing the referenced tables.
3) Every attribute that is a foreign key but is not a part of the primary key of a table is mapped to a data property connecting the objects of the two classes that represent the referencing and the referenced tables.
4) Every attribute that is not a foreign key is mapped to a data property of an appropriate XML datatype.
5) Every *check* constraint that is an expression of the form *attr in* $(s_1, s_2, ..., s_n)$, where *attr* is the attribute name and $s_1, s_2, ..., s_n$ are constants of variable-length character type, each with length greater than a preset value, produces n class definitions that are subclasses of the class representing the table that the *check* constraint is defined on.

By applying these rules, in combination with the table definitions read from the database's data dictionary, we generate the elements of the ontology, one by one, specifying at the same time the mappings that relate to these ontology elements.

For example, let us observe the database schema consisting of the following 3 tables:

COMPUTER

column	datatype	is primary
id	integer	yes
type	varchar	no

EMPLOYEE

column	datatype	is primary
id	integer	yes
name	varchar	no
surname	varchar	no

REPAIRS

column	datatype	is primary
employee_id	integer	yes
computer_id	integer	yes

Also, we assume the following *check* constraint is defined on the *type* attribute of *COMPUTER* table:

```
check type in (`desktop`, `laptop`)
```

Applying the rule 1) on the tables *COMPUTER* and *EMPLOYEE* generates the class definitions *computer* and *employee*, as well as the following two R2RML mappings (given in *Quest* notation used by the *ontop* tool, with empty string being used as the prefix for the default ontology namespace):

```
select id from computer
        ⇝
:computer-{id} a :computer .

select id from employee
        ⇝
:employee-{id} a :employee .
```

The rule 2), applied on the *REPAIRS* table produces object property *repairs*, as well as the following R2RML mapping:

```
select employee_id, computer_id from repairs
                    ⇝
:employee-{employee_id} :repairs
                        :computer-{computer_id} .
```

The *repairs* object property describes the relationship between the *computer* and *employee* classes.

Applying the rule 3) on the *COMPUTER* and *EMPLOYEE* tables generates the data properties for table attributes (*name* and *type* attributes of *COMPUTER* table and *name* and *surname* attributes of *EMPLOYEE* table). We create the names of these data properties by prefixing the name of the attribute with the table name. Appropriate mappings are also generated. These mappings are very similar, therefore, here we only show one of them:

```
select id, name from employee
                  ⇝
:employee-{name} :employee_name
                        {name}^^xsd:string .
```

In the end, the rule 5) would be used, to generate *computer_laptop* and *computer_desktop* classes as subclasses of the *computer* class, based on the *check* constraint. Appropriate mappings for these subclasses are also defined:

```
select id from computer where type = `laptop`
                  ⇝
      :computer-{id} a :computer_laptop .

select id from computer where type = `desktop`
                  ⇝
      :computer-{id} a :computer_desktop .
```

Described ontology is shown in Fig. 2.

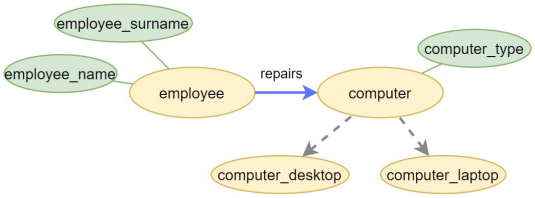

Figure 2. Example generated ontology

These rules ensure the generated ontology well describes the underlying database structures, while also utilizing different OWL and RDF mechanisms to properly model the data (e.g. using subclass relationships to model the hierarchies among entities).

The ontology and the mappings are generated at the same time, and serialized to appropriate OWL and OBDA files. Along with these, a simplified specification of the generated ontology (in the form of a JSON document) is also generated. This file is used by the end-user application, for displaying the graph-based view of the ontology to the user. We found it more efficient to generate this separate file with the ontology description, fit for the front-end application, instead of parsing the OWL file on the client side.

F. Client application

Once the ontology and mappings files have been generated, these are fed to the *ontop* tool, to provide a SPARQL endpoint for querying the data, and to the end-user application, to provide a graphical user interface for interaction with the endpoint.

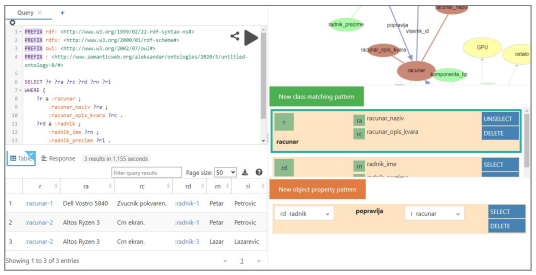

Figure 3. Web-based client application for querying the data

Fig. 3. shows a screenshot of the end-user application running inside of a web browser. The screen space is split into 4 parts:

a. top-left – text editor, for writing queries,
b. bottom-left – result display, for displaying the results of executed queries,
c. top-right – graph-based visual representation of the ontology,
d. bottom-right – utility for pattern-based query construction (aided by the graph-based view of the ontology).

Key elements of this application are the graph-based view of the ontology and the visual query builder utility.

Graph-based view of the ontology allows for panning of the entire view, as well as rearrangement of ontology elements in order to provide a readable display of the ontology that would help the user in formulating queries.

Query-building utility allows for specification of queries by defining *matching patterns* which compose the query. At the time of writing this paper, two types of patterns are supported – class matching patterns and object property matching patterns. Class matching patterns match objects of a specified class, along with selected data properties describing those objects. Object property matching patterns describe object properties that represent relationships between objects of selected classes. Values of the pattern parameters are set by the user, by selecting the ontology elements from the graph, and assigning them to the appropriate pattern parameters. The query is generated on the fly – every change of a pattern parameter is instantly followed by an appropriate update of the query text. Names of the variables that appear in the generated SPARQL query are derived from the names of classes, object properties and data properties that serve as values of pattern parameters.

For example, creating the following two class matching patterns:

Class	Data properties
computer	computer_type

Class	Data properties
employee	employee_name employee_surname

along with the following object property matching pattern:

Class1	ObjectProperty	Class2
employee	repairs	computer

results in generation of the following SPARQL query (namespace prefix declarations are omitted for clarity):

```
SELECT ?c ?ct ?e ?en ?es
WHERE {
      ?c a :computer ;
             :computer_type ?ct .
      ?e a :employee ;
             :employee_name ?en ;
             :employee_surname ?es .
      ?e :repairs ?c .
}
```

The above query selects the types of all computers and the names and the surnames of all employees that repair those computers.

V. CONCLUSION

In this work we proposed the architecture of an automatically deployed OBDA system, as well as a system for automated deployment of OBDA over existing databases. Our system uses a set of rules to generate ontology and mappings, with relation to the database schema, and provides end-user tools for operating the system. This system should allow organizations to implement OBDA into their work in a relatively easy way. The web-based client application should aid in understanding and using the system, and make it accessible to problem domain experts who might not possess the IT-experience necessary to operate classical relational systems. Pattern based query builder serves as an intuitive way to explore and analyse the data. For testing purposes, we also provide *docker* and *docker-compose* files that set up the entire system with a single `docker-compose up` command. The source code of the system (implemented over an example database for a computer repair service) can be found at *https://gitlab.com/jeremicaleksandar/obda_machine/*.

In the future, we plan to improve certain aspects of the system. Notably, the generator utility, which generates the ontology and mappings, could be extended to allow further refinement of the ontology with reliance on external resources (e.g. other existing ontologies). Also, akin to [10], machine learning algorithms could be applied to data to further refine the ontology. For example, clustering algorithms could be used to identify new classes and better describe the underlying data.

Another component that requires improvement is the visual query builder utility of the client application. At the moment, only patterns matching classes, data properties and object properties are supported. In the future, we plan to support additional query patterns such as optional patterns, group by clause, ask queries etc. This way, we would provide users with an intuitive way to formulate even more complex queries.

REFERENCES

[1] de Medeiros, Luciano Frontino, Freddy Priyatna, and Oscar Corcho. "MIRROR: Automatic R2RML mapping generation from relational databases." International Conference on Web Engineering. Springer, Cham, 2015.

[2] Jiménez-Ruiz, Ernesto, et al. "BootOX: Practical mapping of RDBs to OWL 2." International Semantic Web Conference. Springer, Cham, 2015.

[3] Haase, Peter, et al. "Optique system: towards ontology and mapping management in OBDA solutions." (2013).

[4] Postanogov, I. S. "Towards automating the creation of OBDA systems." Procedia Computer Science 150 (2019): 511-517.

[5] OWL Working Group andothers. OWL 2 Web Ontology Language Document Overview (Second Edition). 2012. url: https://www.w3.org/TR/ owl2-overview/.

[6] Souripriya Das, Seema Sundara and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. 2012. url: https://www.w3.org/TR/r2rml/.

[7] Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark and Elias Torres. SPARQL 1.1 Protocol. 2013. url: https://www.w3.org/TR/ sparql11-protocol/

[8] Calvanese, Diego, et al. "Ontop: Answering SPARQL queries over relational databases." Semantic Web 8.3 (2017): 471-487.

[9] Zhang, Lei, and Jing Li. "Automatic generation of ontology based on database." Journal of Computational Information Systems 7, no. 4 (2011): 1148-1154.

[10] Gajderowicz, Bart, Alireza Sadeghian, and Mikhail Soutchanski. "Ontology enhancement through inductive decision trees." In Uncertainty Reasoning for the Semantic Web II, pp. 262-281. Springer, Berlin, Heidelberg, 2010.