Towards a Framework for Ontology-Based Data Access: Materialization of OWL Ontologies From Relational Databases

Sergio Alejandro Gómez[1,2] and Pablo Rubén Fillottrani[1,2]

[1]Laboratorio de I+D en Ingeniería de Software y Sistemas de Información (LISSI)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
San Andrés 800 - Campus Palihue – Bahía Blanca, Buenos Aires, Argentina
Email: {sag,prf}@cs.uns.edu.ar
[2]Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC-PBA)

Abstract. The landscape of information systems applications is comprised of legacy components that many times rely on heterogeneous data sources, proprietary data formats, and a low-level knowledge representation. Only seasoned programmers who maintain those systems may interpret these data. Ontology-based Data Access is a novel approach for developing information systems where an ontology defines a high-level global schema of already existing data sources and provides a vocabulary for user queries. In this work, we report on the construction of a system that exports such data, represented as a legacy relational database, as an OWL ontology in accordance to the W3C Direct Mapping specification. We discuss several case studies that show how the proposed application works.

Keywords. Ontology-based data access, Ontology bootstrapping, Web Ontology Language, Relational databases.

1 Introduction

The landscape of information systems applications is comprised of legacy components that many times rely on heterogeneous data sources, often having proprietary data formats, that also are too complex, because of having a low-level, non-uniform knowledge representation whose interpretation is only understood by the database administrator or the seasoned programmers who maintain those systems. Another challenge in modern information systems is given by need of performing the integration of data scattered among many data sources, again in different formats. Therefore, both end-users and new programmers that have to query and maintain those systems face great difficulties when writing and testing new queries (a process that, in the case of having hundreds of tables, can take a time in the order of weeks).

In the last years, Semantic Web technologies in the form of data artifacts such as ontologies with their associated technologies such as ontology editors,

ontology reasoners and programming libraries have reached a maturity degree that is good enough for encouraging its use for solving the aforementioned challenges. Ontology-based Data Access (OBDA) is a novel approach for developing the new generation of information systems in whose paradigm an ontology defines a high-level global schema of already existing data sources and provides a vocabulary for expressing user queries [1]. The two approaches to OBDA include (i) virtualization (in which the datasources are accessed through query rewriting in SQL of a DL query) and (ii) materialization (in which an ontology is populated from the data sources possibly using SQL filters).

In this work, we present a prototypical application for the materialization of OWL ontologies from relational databases. The prototype has been coded in the Java programming language, and accesses an H2 database in order to export its tables in several OWL formats by using the OWL API. Our implementation follows the directives of the direct mapping specification [2] for generating assertional knowledge and the directives in [1] for expressing terminological knowledge. In Sect. 2, we review the main components of the problem of relational databases, ontology reasoning and OBDA. We present the theoretical framework upon which our implementation is based in Sect. 3, formalizing the rules for materialization and discussing implementation details. We review related work in Sect. 4 and finally conclude and suggest possible research avenues for future work in Sect. 5. This work extends state-of-the-art technologies by providing general definitions of how the ontologies have to be built. To the best of our knowledge, the direct mapping specification, although very clear, only provides examples with no formalization whatsoever. We think that this research could be useful for developers of modern information systems where multiple data sources might exist that need to be accessed in unified way with a simple query language (such us SPARQL) or simply using a DL reasoning engine (such us Hermit).

2 Background

Here, we briefly review the concepts of relational databases, ontologies, mappings between databases and ontologies, and query answering in the context of ontology-based data access. We base part of our presentation in [3].

Relational databases. In the relational data model, a database is composed of a schema and an instance, where the schema defines a set of tables with their attribute names and constraints such us primary and foreign keys. The instance populates the tables of the schema with tuples assigning values to the attributes of the tables. As it is customary, the schema of a table T with a primary key k and attributes $a_1, \ldots, a_n$ will be denoted as $T(\underline{k}, a_1, \ldots, a_n)$ and the instance of T is a set of p tuples denoted as $\{(k^1, a_1^1, \ldots, a_n^1), \ldots, (k^p, a_1^p, \ldots, a_n^p)\}$. When necessary, we will refer to the domains (i.e. the set of valid values for the attributes) of $k, a_1, \ldots, a_n$ as $D_k, D_1, \ldots, D_n$, resp.

Ontologies in Description Logics. An ontology is a machine-processable conceptualization of a portion of the world. *Description Logics* (DL) [4] are a well-known

family of knowledge representation formalisms used for describing ontologies. In this work, we will consider a very tight subset of DL, to which we will restrict our discussion, based on the notions of *concepts* (unary predicates or classes) and *roles* (binary predicates or properties). Concept descriptions are built from concept names $C, D, \ldots$ and roles names $p, q, \ldots$ using the constructors conjunction ($C \sqcap D$), disjunction ($C \sqcup D$) and negation ($\neg C$) and inverse role p^-. The empty concept is denoted by $\bot$ and the universal concept is denoted by $\top$. As it is usual, we will abbreviate $\exists p.\top$ by $\exists p$. A DL ontology $\Sigma = (T, A)$ consists of two finite and mutually disjoint sets: a *Tbox* T which introduces the *terminology* and an *Abox* A (assertional box) which contains facts about particular objects in the application domain. The Tbox contains inclusion axioms $C \sqsubseteq D$, where C and D are (possibly complex) concept descriptions, meaning that every individual of C is also a D. In this work, we will restrict ourselves to two forms of axioms, viz. $C \sqsubseteq \exists p$, meaning that every individual of type C is related through role p to some other individual, and $\exists p^- \sqsubseteq C$, meaning that individuals related by role p have to necessarily be related to an individual of a concept C. Objects in the Abox are referred to by a finite number of *individual names* and these names may be used in assertional statements $C(a)$, meaning the individual a is a member of concept C, and $p(a, b)$, meaning that the individual a is related to the individual b by role p.

Web Ontology Language. The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web [5]. OWL provide syntactic constructors to express classes, properties, individuals, and data values. OWL ontologies can be used along with information written in RDF, and OWL ontologies themselves are primarily exchanged as XML documents. In OWL, objects are represented with Uniform Resource Identifiers, which essentially are generalized web addresses (e.g. `http://foo.org/`) that do not necessarily have to be hosted in some web server. As OWL ontologies are serialized as XML (or N3/Turtle) documents they could be edited with simple text editors if desired. However to provide support for both knowledge engineers and programmers, there are ontology editors such as Protégé and Java libraries such as the OWL API [6].

Mappings. Mappings define how ontological terms are related to terms occurring in the relational schema and are essentially view definitions of the form $\mathit{Class}(f_o(x)) \leftarrow \mathsf{SQL}(x)$, that declare how to populate classes with objects; to populate properties with object-object, of the form $\mathit{objectProperty}(f_o(x), f_o(y)) \leftarrow \mathsf{SQL}(x, y)$, and object-value pairs, of the form $\mathit{dataProperty}(f_o(x), f_d(y)) \leftarrow \mathsf{SQL}(x, y)$, where $\mathsf{SQL}(\cdot)$ and $\mathsf{SQL}(\cdot, \cdot)$ are SQL queries with one and two output variables, resp., and $f_o(\cdot)$ and $f_v(\cdot)$ are functions that cast values returned by the SQL queries into objects (i.e. URIs and literals). Classes are populated with URIs $f_o(x)$ computed from the values x returned by $\mathsf{SQL}(x)$. Properties can relate two objects or assign a value to an object; in the former case, with pairs of objects $f_o(x)$ and $f_o(y)$ and in the latter case assigning the value $f_v(y)$ to the object $f_o(x)$ when the query $\mathsf{SQL}(x, y)$ is computed.

Query answering. Given a data access instance $(\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$, where $\mathcal{D}$ is a relational database, $\mathcal{V}$ is an ontological vocabulary, $\mathcal{O}$ is a set of ontological axioms over $\mathcal{V}$, and $\mathcal{M}$ is a set of mappings between $\mathcal{V}$ and $\mathcal{D}$. There are two approaches to answer a query Q over $\mathcal{V}$: (i) *materialization*: ontological facts are materialized (i.e. classes and properties participating in mappings are populated with individuals by evaluating SQL queries participating in mappings) and this gives a set of ontological facts $\mathcal{A}$ and then Q is evaluated against $\mathcal{O}$ and $\mathcal{A}$ with standard query-answering engines for ontologies, or (ii) *virtualization*: Q should be first rewritten into SQL using $\mathcal{O}$ and $\mathcal{M}$ and then SQL should be executed over $\mathcal{D}$. In this work, we will use the materialization approach.

3 Materialization of OWL Ontologies from Relational Databases

Materializing an OWL ontology from a relational database requires exporting the database contents as a text file in OWL format. For doing this, we need to export the schema information of each table as Tbox axioms and the instance data of the tables as Abox assertions. Here we formalize the process of exporting a single table having a single primary key, two tables participating in a one-to-many relationship, three tables in a many-to-many relationship and a table without a primary key. After that, we report implementation details.

3.1 Rules for Exporting Tables as Ontologies

We implemented the direct mapping specification from relational databases to OWL according the directions given by [1, 2]. Building an ontology from a database requires creating at least a class C_T for every table T, and for every attribute a of domain d in T we need two inclusion axioms $C_T \sqsubseteq \exists a$ and $\exists a^- \sqsubseteq d$. Primary key values k_i serve the purpose of establishing the membership of individuals to classes as Abox asssertions of the form $C_T(C\#k^j)$. For indicating that a^j is the value of attribute a, we will use a role expression of the form $C_T\#a(C_T\#k^j, C_T\#a^j)$. When it is clear from context, we might drop the prefix $C_T\#$ for simplifying our notation. A foreign key fk in table T_1 referencing a primary key field in table T_2 will also require to add two Tbox axioms $C_{T_1} \sqsubseteq \exists \mathsf{ref}\_fk$ and $\exists \mathsf{ref}\_fk^- \sqsubseteq C_{T_2}$ and an Abox assertion $\mathsf{ref}\_fk(k^j, fk^t)$ for expressing that the individual named k^j in C_{T_1} is related to the individual named fk^t in C_{T_2}. Besides, in any case, if we want to consider a subset of a table for its mapping into an ontology, we might define an SQL query that will act as an SQL filter.

We will present how to map a table with a single primary key, two tables participating in a one-to-many relationship, three tables participating in a many-to-many relationship, and a table non-having a primary key. For each case, we give a formal specification of how the mapping must be done and present a concrete example. In all cases, the base URI for the ontology will be `http://foo.org`.

Definition 1 (Mapping of a table with a single primary key). *Let T be a table with schema $T(\underline{k}, a_1, \ldots, a_n)$ and instance $\{(k^1, a_1^1, \ldots, a_n^1), \ldots, (k^m, a_1^m, \ldots, a_n^m)\}$. To map T into DL, we have to create a class T and for each attribute a_i of domain D_i we have to add two axioms: $T \sqsubseteq \exists a_i$, indicating that every T has an attribute a_i, and $\exists a_i^- \sqsubseteq D_i$, meaning that the type of a_i is D_i. The assertional box for T will contain $\{C(k^1), \ldots, C(k^m)\}$. For every attribute a_i of the schema and instance value a_i^j, produce a property $a_i(k^j, a_i^j)$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.*

Example 1. Let us consider a table for representing cars with the following schema: *Car(licensePlate, brand, model, weight, preowned, dateOfPurchase, numberOfDoors)*, populated with:

licensePlate	*brand*	*model*	*weight*	*preowned*	*dateOfPurchase*	*numberOfDoors*
ABC123	Chevrolet	Corsa	1000.1	FALSE	2010-10-01	4
CDE456	VW	Suran	1000.4	TRUE	2013-10-01	5

The OWL file generated by the system from the URI "http://foo.org" and SQL filter select * from Car where dateOfPurchase <= 2011-12-31. Then the name of the class would be *Car*. For example, for the attribute *brand* of type *string*, we will have two terminological axioms $Car \sqsubseteq \exists brand$ and $\exists brand^- \sqsubseteq String$. For saying that $ABC123$ is a car, we will have an assertion $Car(ABC123)$ and for establishing that the brand of $ABC123$ is Chevrolet, we will have an assertion $brand(ABC123, Chevrolet)$. Likewise, for saying that cars have an integer number of doors and that $ABC123$ has 4 doors, we will have $Car \sqsubseteq \exists numberOfDoors$, $\exists numberOfDoors^- \sqsubseteq Integer$ and $numberOfDoors(ABC123, 4)$. These DL assertions are expressed in OWL as depicted in Fig. 1.

```
<owl:Class rdf:about="http://foo.org#Car"/>

<owl:DatatypeProperty rdf:about="http://foo.org/Car#brand">
   <rdfs:domain rdf:resource="http://foo.org#Car"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="http://foo.org/Car#numberOfDoors">
   <rdfs:domain rdf:resource="http://foo.org#Car"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>
...
<owl:NamedIndividual rdf:about="http://foo.org/Car/licenseplate=ABC123">
   <rdf:type rdf:resource="http://foo.org#Car"/>
   <Car:brand rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Chevrolet</Car:brand>
   ...
   <Car:numberOfDoors rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">4
   </Car:numberOfDoors>
    ...
</owl:NamedIndividual>
```

Fig. 1. OWL code for part of the *Car* class from Ex. 1

We now present how to map two tables participating in a one-to-many relationship.

Definition 2 (Mapping of a one-to-many relationship). *Let* $A(\underline{k_1}, a_1, \ldots, a_n)$ *and* $B(\underline{k_2}, b_1, \ldots, b_m, k_1)$ *be two tables participating in a one-to-many relationship where* k_1 *is both the primary key in* A *and a foreign key in* B*. Tables* A *and* B *are translated in DL according to Def. 1. Besides, the two axioms are added:* $B \sqsubseteq \exists \mathsf{ref}\_k_1$ *and* $\exists \mathsf{ref}\_k_1^- \sqsubseteq A$*. And for every tuple* $(k_1^i, a_1^i, \ldots, a_n^i)$ *of* A *related to a tuple* $(k_2^j, b_1^j, \ldots, b_m^j, k_1^i)$ *in* B*, an assertion* $\mathsf{ref}\_k_1(k_2^j, k_1^i)$ *is added.*

Example 2 (Continues Ex. 1). Consider a one-to-many relation of table *Car* from Ex. 1 with a table $AutoPart(\underline{autoPartID}, name, carID)$, populated with:

autoPartID	*name*	*carID*
1	engine	ABC123
2	windshield	ABC123

The following assertions are added: $autoPartID(1)$, $name(1, engine)$, $carID(1, ABC123)$, $\mathsf{ref}\_carID(1, ABC123)$, $autoPartID(2)$, $name(2, windshield)$, $carID(2, ABC123)$, $\mathsf{ref}\_carID(2, ABC123)$. Notice that we actually abuse notation in $carID(1, ABC123)$ as the second argument is an string literal while in $\mathsf{ref}\_carID(1, ABC123)$ it stands for an object identifier. See Fig. 2 to see how the assertions for the first tuple are codified in OWL.

```
<owl:Class rdf:about="http://foo.org#AutoPart"/>
...
<owl:ObjectProperty rdf:about="http://foo.org/AutoPart#ref-carID"/>
...
<owl:NamedIndividual rdf:about="http://foo.org/AutoPart/autopartid=1">
   <rdf:type rdf:resource="http://foo.org#AutoPart"/>
   <AutoPart:ref-carID rdf:resource="http://foo.org/Car/licensePlate=ABC123"/>
   <AutoPart:autoPartID rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1
    </AutoPart:autoPartID>
   <AutoPart:carID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">ABC123
    </AutoPart:carID>
   <AutoPart:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">engine
    </AutoPart:name>
</owl:NamedIndividual>
...
```

Fig. 2. Part of the OWL code for *AutoPart* class from Ex. 2

We now address the issue of mapping two tables in a many-to-many relationship. As it is customary in the relational data model, the relationship is reified an intermediate table containing the primary keys of the tables being related. We motivate the concept of *composite property name* for expressing tables having primary keys comprised of more than one key field.

Definition 3 (Mapping of a binary many-to-many relationship). *Let* $A(\underline{k_1}, a_1, \ldots, a_n)$ *and* $B(\underline{k_2}, b_1, \ldots, b_m)$ *be two tables in a many-to-many relationship* R *reified as* $R(\underline{k_1}, \underline{k_2}, r_1, \ldots, r_l)$*. Tables* A *and* B *are expressed in DL according to Def. 1. The primary key of table* R *is expressed as a* composite property name $k_1 \circ k_2$*. The axiom* $R \sqsubseteq \exists(k_1 \circ k_2)$ *is added as well as two axioms for* k_1 *and* k_2*:* $\exists \mathsf{ref}\_k_1 \sqsubseteq A$ *and* $\exists \mathsf{ref}\_k_2 \sqsubseteq B$*.*

Example 3 (Continues Ex. 2). Consider another table *Person* for representing people and yet another one for representing what person drives what car of table *Car* from Ex. 1, with schemas $Person(\underline{identification}, name)$ and $Drives(\underline{personID}, \underline{carID}, lastTime)$, resp., and populated with:

Person

identification	*name*
10	John

Drives

personID	*carID*	*lastTime*
10	ABC123	2018-06-10

The following axioms have to be added to the Tbox for representing people: $Person \sqsubseteq \exists identification$, $Person \sqsubseteq \exists name$, $\exists identification^- \sqsubseteq String$, and $\exists name^- \sqsubseteq String$. For representing who drives what car, we have to add: $Drives \sqsubseteq \exists personID$, $Drives \sqsubseteq \exists carID$, $Drives \sqsubseteq \exists lastTime$, $\exists personID^- \sqsubseteq String$, $\exists carID^- \sqsubseteq String$, and $\exists lastTime^- \sqsubseteq Date$. Finally, for defining the composite property built up of the concatenation of the primary key fields of the *Drives* table, the following axioms are added: $Drives \sqsubseteq \exists(personID \circ carID)$. For modeling the instances of table *Person*, the following assertions are added to the ontology: $Person(10)$, $identification(10, 10)$, $name(10, john)$. And for the table *Drives*, whose only tuple has the composite primary key "$10 \circ ABC123$", the following assertions are included in the ontology: $Drives(10 \circ ABC123)$, $personID(10 \circ ABC123, 10)$, $carID(10 \circ ABC123, ABC123)$, $\mathsf{ref}\_personID(10 \circ ABC123, 10)$, $\mathsf{ref}\_carID(10 \circ ABC123, ABC123)$, and $lastTime(10 \circ ABC123$, 2018-06-10). Part of the OWL code for modeling this situation is presented in Fig. 3.

```
<owl:Class rdf:about="http://foo.org#Drives"/>
...
// Object Properties
<owl:ObjectProperty rdf:about="http://foo.org/Drives#ref-carID"/>
<owl:ObjectProperty rdf:about="http://foo.org/Drives#ref-personID"/>
...
<owl:NamedIndividual rdf:about="http://foo.org/carID=ABC123;personID=10">
  <rdf:type rdf:resource="http://foo.org#Drives"/>
  <Drives:ref-carID rdf:resource="http://foo.org/Car/licensePlate=ABC123"/>
  <Drives:ref-personID rdf:resource="http://foo.org/Person/identification=10"/>
  <Drives:carID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">ABC123</Drives:carID>
  <Drives:lastTime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
      2018-06-10T00:00:00</Drives:lastTime>
</owl:NamedIndividual>
```

Fig. 3. OWL code for part of the representation of table *Drives* from Ex. 3

In the case of tables non-having a primary key, the direct mapping specification requires to build blank nodes as the individual identifiers for each row of the table, we take a simpler approach by defining a *surrogate primary key* with auto-incrementing values.

Definition 4 (Mapping of tables non-having a key). *Let* $T(a_1, \ldots, a_n)$ *be a table, we will interpret it as if it was a table* $T(\underline{blankkey}, a_1, \ldots, a_n)$ *where*

blankkey is called a surrogate primary key *and an instance of* T $\{(a_1^1, \ldots, a_n^1), \ldots, (a_1^m, \ldots, a_n^m)\}$ *will be interpreted as* $\{(1, a_1^1, \ldots, a_n^1), \ldots, (m, a_1^m, \ldots, a_n^m)\}$.

Example 4. Consider a table *Measurement*(*time*, *temperature*) populated with:

time	*temperature*
2018-07-03T12:00:00	23.4
2018-07-03T13:00:00	25.6

Notice that in this case, we could have had *time* defined as the primary key of the table but it could have also be the case of having several measurements recorded by different instruments and that situation would have made that option infeasible. The Tbox is then composed by: $Measurement \sqsubseteq \exists blankkey$, $\exists blankkey^- \sqsubseteq Integer$, $Measurement \sqsubseteq \exists time$, $\exists time^- \sqsubseteq TimeStamp$, $Measurement \sqsubseteq \exists temperature$ and $\exists temperature^- \sqsubseteq Double$. From the table's contents, we obtain the following assertions: $Measurement(1)$, $time(1, 2018\text{-}07\text{-}03\ 12{:}00{:}00.0)$, $temperature(1, 23.4)$, $Measurement(2)$, $time(2, 2018\text{-}07\text{-}03\ 13{:}00{:}00.0)$ and $temperature(2, 25.6)$. Part of the OWL code for modeling this situation can be seen in Fig. 4.

```
<owl:NamedIndividual rdf:about="http://foo.org/Measurement/BLANKKEY=1">
  <rdf:type rdf:resource="http://foo.org#Measurement"/>
  <Measurement:temperature rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
     23.4</Measurement:temperature>
  <Measurement:time rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTimeStamp">
     2018-07-03 12:00:00.0</Measurement:time>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://foo.org/Measurement/BLANKKEY=2">
  <rdf:type rdf:resource="http://foo.org#Measurement"/>
  <Measurement:temperature rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
     25.6</Measurement:temperature>
  <Measurement:time rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTimeStamp">
     2018-07-03 13:00:00.0</Measurement:time>
</owl:NamedIndividual>
```

Fig. 4. Individuals of the *Measurement* class of Ex. 4

3.2 Implementation Details

We now discuss implementation details of the application that we developed for bootstrapping ontology contents from a relational database. The application in its current state can process relational databases in H2 format. The access to a certain database is performed via JDBC, where the system, after establishing a connection with user provided parameters such as name of the database, user ID and password, can automatically obtain the database schema. OWL code generation is done via the OWL API [6]. In this option, the user can specify a base URI, the name of the table to be mapped as an ontology, an SQL filter for selecting a subset of the table's records, and can see the schema information of the table (e.g. attributes, primary and foreing keys). The application allows to

generate the ontologies in two OWL syntaxes, viz. the XML syntax as shown in Ex. 1–4 and the Turtle (i.e. Terse RDF Triple Language) syntax [7]. There is another option for generating an ontology for the contents of the entire database (where the user has to specify the name of the ontology and a base URI). An executable JAR file and a sample database with the contents of Ex. 1–4 can be downloaded from `http://cs.uns.edu.ar/~sag/obda-test`.

4 Related Work

We now discuss related work. MASTRO [8] is an Ontology-Based Data Access (OBDA) management system. Ontologies in MASTRO are specified through languages belonging to the DL-Lite family of lightweight DLs. The ontology is connected to external relational data management or data federation systems through a mapping establishing a semantic relation between SQL queries issued over the underlying databases and elements of the ontology. To access data, users can specify SPARQL queries over the ontology and make use of the query answering services provided by Mastro. Mastro is developed in Java. Our implementation is also implemented in Java what expedites its interaction with OWL API. MASTRO represents ontologies in OWL Lite, a dialect of OWL that, despite its efficiency, imposes too much restrictions on its knowledge representation capabilities. We aim towards more rich and flexible dialects of OWL that will eventually allow us to model richer data features such as the cardinality of relations.

BOOTOX [3] is a system that aims at facilitating ontology and mapping development by their automatic extraction (i.e., bootstrapping) from relational databases. It allows to control the OWL 2 profile of the output ontologies, bootstrap complex and provenance mappings, which are beyond the W3C direct mapping specification, and also it allows to import pre-existing ontologies via alignment. Our implementation exports the ontologies in several formats of OWL (viz., Turtle and XML) though we have not yet committed to specific OWL profiles because we plan on using standard DL reasoners (e.g., Hermit or Pellet).

RODI [9] is a benchmark for the evaluation of the practical utility of ontology-based data integration systems and their application in practice. RODI includes test scenarios from the domains of scientific conferences, geographical data, and oil and gas exploration. Scenarios are constituted of databases, ontologies, and queries to test expected results. Systems that compute relational-to-ontology mappings can be evaluated using RODI by checking how well they can handle various features of relational schemas and ontologies, and how well the computed mappings work for query answering. We plan on using RODI for extending the evaluation of our implementation.

5 Conclusions and Future Work

We have presented an application in a prototypical status for performing OBDA by allowing the user to materialize the contents of an H2 relational database

as an OWL ontology. Our main contribution is a clear and precise description of the mappings to guarantee that the database data are well interpreted as ontological data. Our application is programmed in Java and uses JBDC to access the database and the OWL API for generating the ontology. We showed how our application is capable of handling the cases of exporting a single table with or without a key field, two tables in a one-many-relationship and three tables implementing a many-to-many relationship. As future work, we plan to perform an experimental evaluation of its performance on different databases. We also plan to add more features, e.g. mapping generation, SQL unfolding and supporting more database formats, this will allow to include more precise user-defined specifications of each mapping such as implicit information present in neither the database's schema nor the instance but in the application usage patterns of the database.

Acknowledgments. This research is funded by Secretaría General de Ciencia y Técnica, Universidad Nacional del Sur, Argentina and by Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC-PBA).

References

1. Kontchakov, R., Rodríguez-Muro, M., Zakharyaschev, M.: Ontology-Based Data Access with Databases: A Short Course. In: Reasoning Web: Semantic Technologies for Intelligent Data Access of the LNCS. Volume 8067. Springer (2013) 194–229
2. Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J.: A Direct Mapping of Relational Data to RDF. W3C Recommendation 27 September 2012 (2012)
3. Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I., Pinkel, C., Skjæveland, M.G., Thorstensen, E., Mora, J.: BootOX: Practical Mapping of RDBs to OWL 2. In: the 14th International Semantic Web Conference. (2015) 113–132
4. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook – Theory, Implementation and Applications. Cambridge University Press (2003)
5. Bao, J., Kendall, E.F., McGuinness, D.L., Patel-Schneider, P.F.: OWL 2 Web Ontology Language Quick Reference Guide (Second Edition) W3C Recommendation 11 December 2012 (2012)
6. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL Ontologies. Semantic Web **2**(1) (2011) 11–21
7. Becket, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1 Turtle. Terse RDF Triple Language. W3C Recommendation 25 february 2014 (2014)
8. Calvanese, D., Giacomo, G.D., Lembo, D., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web **2**(1) (2011) 43–53
9. Pinkel, C., Binnig, C., Jiménez-Ruiz, E., Kharlamov, E., May, W., Nikolov, A., Bastinos, A.S., Skjæveland, M.G., Solimando, A., Taheriyan, M., Heupel, C., Horrocks, I.: RODI: Benchmarking Relational-to-Ontology Mapping Generation Quality. Semantic Web (2016) 1–26