ENHANCING CONNECTION BETWEEN ONTOLOGIES AND DATABASES WITH OWL 2 CONCEPTS AND SPARQL

Ernestas Vysniauskas, Lina Nemuraite, Algirdas Sukys, Bronius Paradauskas

Kaunas University of Technology, Department of Information Systems, Studentu 50-308a, Kaunas, Lithuania, Ernestas.Vysniauskas@stud.ktu.lt, Lina.Nemuraite@ktu.lt, Sukys.Algirdas@gmail.com, BoniusParadauskas@ktu.lt

Abstract. The goal of the paper is to present the enhanced database schema for storing ontologies considering new features of OWL 2 and possibilities of querying these ontologies using SPARQL. The growing size of ontologies and the scope of their applications require the effective means for storing ontology data that relational databases already have approved. Many existing ontology reasoning tools are using relational databases for this purpose. However, in practice almost all of them are using the straightforward approach restricted to representing instances whereas the effectiveness of processing ontological data may be considerably improved by keeping information about ontology classes, object properties and more advanced concepts in database tables. Previously we have presented the method and tool for transforming OWL ontologies to relational database. Currently, we have extended our representation with novel concepts of OWL 2, the recent Recommendation of W3C. Also, we present a prototype of a tool for extracting ontologies from relational databases and thus allowing the step-wise processing of SPARQL queries where SPARQL is used for querying ontology structures in a main memory and SQL is used for querying instances in the database.

Keywords. Ontology, relational database, OWL 2, SPARQL, SQL, mapping, transformation.

1 Introduction

Ontology descriptions, in particularly Web Ontology Language OWL, become more and more widely used in the World Wide Web and other fields as common information systems, data integration and software engineering. Currently, many areas are becoming knowledge-based [12]. Ontologies allow creating of better information systems by empowering them with advanced possibilities for operation and interoperability with other systems by opening access to existing heterogeneous and distributed databases and other information resources. Since 2004, a practical experience with OWL 1 has shown that it lacks several constructs that are often necessary for modelling complex domains [11]. The new developments of OWL 1, initially informally undertaken by some group of its users, in 2009 were issued as OWL 2 – a new W3C Recommendation [10, 23], augmenting the previous OWL with advanced features: qualified cardinality restrictions; complex sub-property axioms between a property and a property chain; local reflexivity restrictions; disjoint, reflexive, irreflexive, symmetric, and anti-symmetric properties; negative property assertions; vocabulary sharing (punning) between individuals, classes, and properties; the richer set of datatypes and datatype restrictions etc.

In our previous work [29, 30] we have proposed the mapping between OWL ontology and relational database and a tool for transforming ontologies into databases. Such an approach is needed because ontology based systems are growing in scope and storages of ontology reasoners are becoming unsuitable. While there are other solutions and tools for keeping ontologies in databases, the most of them are storing only RDF data. Lee and Goodwin, who have proposed the database-centric approach to ontology management support, notice that such methodology is still in its infancy [20]. In our approach, some concepts, e.g. ontology classes and properties are mapped to relational tables, relations and attributes, other (constraints) are stored like metadata in special tables. Using both direct mapping and metadata, it is possible to obtain appropriate relational structures and do not lose the ontological data. In connection with new features of OWL 2 and its supporting tools as Protégé [18] and Pellet [27], we have revised the previous representation and supplemented it with new concepts\(^1\).

Our approach is well-suited for creating new databases from ontologies, however, in practice ontologies often are used for accessing already existing databases that usually are heterogeneous and distributed. Therefore, methodologies are even more needed for extracting ontologies from existing databases. It is a hard task to automatically obtain meaningful knowledge from such legacy systems without human intervention, so it is worth to beforehand ontology structures and to store them in databases for ontology management purposes. In cases when ontologies are being created from databases our approach fits as well, because the database schema obtained by our transformation is capable of the lossless\(^2\) representation of ontological

---

\(^1\) The research is pursued according the project proposal “Methodology and Technology Foundations for Semantically-Based Information System Design (SEMIS)"

\(^2\) We have in mind the lossless transformation of OWL ontologies formulated using sufficient subset of its concepts because criteria of completeness and performance require for some compromise. Complete representation of OWL in a database is even undesirable as inference in OWL FULL is undecidable [14].
information in databases and the lossless retrieval of this information from databases into ontology reasoning tools. We present a prototype of a tool for extracting ontologies from relational databases, satisfying our schema, and allowing the step-wise processing of SPARQL queries where SPARQL is used for querying ontology structures in a main memory and SQL is used for querying instances in the database.

The rest of the paper is organized as follows. Section 2 presents related works. Section 3 is devoted to mapping of OWL 2 concepts to RDB concepts. Section 4 presents our approach to querying OWL 2 ontologies stored in relational databases. Section 5 draws conclusions and outlines the future work.

2 Related Work

OWL is different from conceptual modelling languages as ER or UML class diagrams as it has richer capabilities to describe classes and to handle incomplete knowledge. OWL 2, the emerging new version of OWL, is more expressive and still allows for complete and decidable computing [11]. Significant improvement in ontology management and reasoning tools has been achieved due to enhancement and additional functionality provided in OWL 2. For example, the widely used Protégé and Pellet systems and graphical OWL notation were extended with additional constructs of OWL 2 [18, 27, 16]. Consequently, we are aiming for the extending and improving our previous OWL2RDB transformation in accordance with new possibilities of OWL 2.

The new features of OWL 2 aim at increasing the relational expressivity of OWL 1 by allowing propagation of constraints along properties: transitivity of properties, subproperty and property chain axioms [10, 11, 23]. Object property axioms now can define reflexivity and symmetry, and various property restrictions: all values from, some values from, restrictions on values. The set of built-in OWL datatypes was extended from strings and integers in OWL 1 to XML schema datatypes and various datatype restrictions. As the lack of keys in OWL 1 was recognized as an important limitation in expressive power keys were introduced into OWL 2. They may be defined on a list of object or data properties. Also, OWL 2 adds syntactic sugar to make some common patterns easier to write. Since these constructs are simply shorthands, they do not change the expressiveness, semantics, or complexity of the language.

There were three dialects defined in OWL 1: OWL DL, OWL Full and the syntactic subset OWL Lite. These dialects exposed their insufficiency for implementing tools working with OWL ontologies [10]. To resolve the issues of OWL tools, three OWL profiles were proposed having properties useful for different kinds of computation. OWL 2 EL captures the expressive power used by large-scale ontologies, having many classes and properties. OWL 2 QL captures the expressive power of simple ontologies like thesauri and ER/UML languages; it is well suited for working with very large number of individuals, and where it is needed to access data directly via relational queries. OWL 2 RL is designed to implementation using rule-based technologies. All of these profiles have certain restrictions. Although we are oriented towards application of ontologies in information systems and storing in databases, we do not intend to link with OWL 2 QL profile as it has serious limitations and is suitable to applications requiring only very simple ontologies.

We already have discussed existing approaches for representing ontologies in databases [1–3, 8, 19, 20, 22] in [30] and concluded in the rationale of creating bidirectional, lossless, model-based transformations between ontologies and database schemas. Metamodels of ontology language and database schema serve for this purpose. We are following the OWL 2 metamodel [23] for representing ontology. For a relational database, we use a part of Common Warehouse Metamodel (CWM) [4] that currently is under extension to more powerful CWM 2.x named as “Information Management Metamodel” (IMM) [15].

Also, we studied the approaches for inverse mapping – i.e. from relational databases to ontology [5, 6, 9, 13, 26] (the survey is given in [28]). We analyze aspects of RDB2OWL transformations for lossless OWL2RDB transformation as most of relational concepts may be mapped to ontology structures, but not every ontology concept may be directly mapped to a relational database.

Among all these methods we can distinguish two ultimate information-lossless cases: storing ontology and its instances in the same manner (one fact table) or storing ontology and its instances in different schemas in order to improve access to instances while retaining the capacity of reasoning over the ontology. The first transformation method does not lose information, but it uses advantages of relational databases just for saving many records and does not preserve the real relational structure. The schema is in a low normal form and the performance of using transformed information normally should be slow e.g. [20]. The similar method is highly powered in Oracle Semantic Storage as it is supported with the native functionality of the Oracle database [31], where functionality of triggers helps to reasoning in ontology (alike in business rule manipulation techniques e.g. [21]). The second approach is much more promising for ordinary database management systems (e.g. [1, 2]). However, existing methods of that kind do not cover the sufficient subset of ontology concepts.

Our OWL2RDB transformation combines direct mapping of ontology classes, properties and instances with representing axioms and restrictions in metadata tables. Herewith we consider the reverse transformation of ontology from a database for efficient reasoning that may be achieved by joint usage of ontology query language SPARQL [25] and relational database query language SQL [7]. Reasoners that use ontologies represented in
XML files usually extract ontology schema along with its instances into a main memory and perform all inference there [27]. Performing full reasoning in memory ensures the completeness of query results but it is unsuitable for large ontologies having many instances. In our case, only ontology structures are extracted into a memory and processed by the inference engine. Results of inference are used for accessing individuals by SQL queries obtained by converting fragments of SPARQL to SQL.

This process is optimized in PelletDB reasoner for Oracle DB 11g due to the powerfulness of Oracle [16]. If ontology is of the acceptable size, PelletDB loads both the schema and the instance data from Oracle DB, then computes and saves all inferences back to the Oracle Database, which can be queried without additional reasoning. When instance data are too large to fit into memory, PelletDb extracts only the schema, computes additional schema inferences, and saves these inferences in Oracle Database. Then it is possible to perform instance reasoning using the schema inferences. The combination of reasoning in-memory together with instance reasoning in database provides a viable means to achieving more complete inference and query results than either solution can offer alone. While there is no question about competing with Oracle, our approach scales for every size of applications and may be implemented in non-commercial database management systems.

3 OWL 2 Concepts and their Mapping to RDB Concepts

OWL 1 was mainly focused on constructs for expressing information about classes and individuals, and exhibited some weakness regarding expressiveness for properties. OWL 2 offers new constructs for describing properties, a richer set of datatypes and makes some common patterns easier to write. In this paper we analyse these OWL 2 concepts that in our opinion are the most useful for real world applications and can be transformed to relational database schemas. As basic mappings are similar to OWL 1 mappings presented in [29, 30], in this paper we are focusing on mappings of new constructs. As previously, we are combining mappings of OWL 2 concepts with RDB concepts and storing the problematic (in mapping sense) knowledge in metadata tables. For explaining the proposed mapping, we will use the extended excerpt of Wine Ontology as our example (Figure 1) where it is represented using UML OWL 2 profile [24] implemented in Protégé OWL2UML plug-in.

![Figure 1. Example of OWL 2 wine ontology](image)

3.1 OWL Classes and Class Axioms

In OWL 2, classes and property expressions are used to construct class expressions that represent sets of individuals by formally specifying conditions on the individuals’ properties; individuals satisfying these conditions are said to be instances of the respective class expressions. OWL 2 provides axioms that allow relationships to be established between class expressions (Figure 2).

When we are converting the OWL ontology description to relational database schema, we map each ontology class to a database table. As the name of an ontology class is unique in the whole ontology and instances have unique names, we create a primary key for each table by adding some suffix to the corresponding class name, e.g. “Id”, and the additional column by adding “Name” suffix to the class name for saving names of instances of the class. The fundamental taxonomic construct – the SubClassOf axiom, which allows to state that each instance of one class expression is also an instance of another class expression, is mapped to 1:0..1 relation...
in RDB. The subclass doesn’t need a column for saving names of its instances, because the instance of the subclass is also the instance of the super class and its name is already stated. These mappings for PortableLiquid, Wine, WineGrape, WineMaker and WineTaster classes of Wine Ontology example are presented in the upper part of Figure 3. We use our own UML profile for representing database schema where <<PK>>, <<FK>>, <<UK>> stereotypes mark primary keys, foreign keys and unique constraints; tags “id” mark names of foreign keys, and tags “uk” mark names of unique constraints.

![Figure 2. The OWL class descriptions diagram ([23])](image)

The EquivalentClasses axiom defines that several class expressions are equivalent to each other, i.e. they have the same instances. The DisjointClasses axiom states that several class expressions are pair wise disjoint. The DisjointUnion class expression allows to define a class as a disjoint union of several class expressions and thus to express covering constraints. In our example, disjoint classes are the WineMaker and the CertificationCompany; they comprise one disjoint class group and we could be able to define a superclass of these classes e.g. “Company” as the disjoint union class if there were more disjoint groups. For preserving such information, we suggest saving all classes of the ontology in OWLClasses table with two main columns classId, which is an auto increment identification number, and className, which saves the unique name of the class. This name is also the name of the corresponding table. Information about groups of disjoint and equivalent classes is saved in metatables OWLDisjointClasses and OWLEquivalentClasses. The groups of equivalent or disjoint classes also are represented in OWLEquivalentGroup and OWLDisjointGroup tables. Metatables for OWL 2 disjoint, equivalent and disjoint union classes are presented in the lower part of Figure 3.

### 3.2 OWL 2 Properties and Property Axioms

OWL 2 has two main categories of properties – object and data properties, and also annotation properties that may be useful for ontology documentation. Object properties relate individuals to other individuals. Data properties relate individuals to literals. We map the object property to the foreign key. Depending on the local cardinality of some class property and the object property is functional or not, one-to-many or many-to-many relation between tables of classes are created. In a case of many-to-many relation, an intermediate table must be created.

OWL 2 provides axioms for establishing relationships between object property expressions. The ObjectPropertyDomain and ObjectPropertyRange axioms can be used to restrict the first and the second individual, connected by an object property expression. The FunctionalObjectProperty and InverseFunctionalObjectProperty axioms define that each individual can have at most one outgoing or incoming connection of the specified object property expression respectively. The InverseObjectProperties axiom can be used to state that two object property expressions are the inverse of each other. The ReflexiveObjectProperty, IrreflexiveObjectProperty, SymmetricObjectProperty, AsymmetricObjectProperty, and TransitiveObjectProperty axioms define that an object property expression is reflexive, irreflexive, symmetric, asymmetric, or transitive. These axioms are represented in metatable “OWLObjectProperties” (Figure 3).

In OWL 2 there are two forms of object subproperties axioms. The basic form is SubObjectPropertyOf(OPE1, OPE2). This axiom states that the object property expression OPE1 is a subproperty of the object property expression OPE2 — that is, if an individual x is connected by OPE1 to an individual y, then x is also connected by OPE2 to y. E.g. in our example the class PortableLiquid has the object property HasMaker, and the class Wine has the object property HasWineMaker which is the subproperty of the property HasMaker. Information that one property is a subproperty of another property we save in the metatable OWLObjectProperties.

Another form of OWL 2 object subproperty is ObjectPropertyChain. The axiom SubObjectPropertyOf (ObjectPropertyChain(OPE1, ..., OPEn), OPE) states that, if an individual x is connected by a sequence of object property expressions OPE1, ..., OPEn with an individual y, then x is also connected with y by the object property expression OPE. E.g. we have the class Wine and the object property isTastedBy with the range class
WineTaster. The class WineTaster has the object property worksForCompany with the range class CertificationCompany. We can declare the axiom SubObjectPropertyOf(ObjectPropertyChain(a:isTastedBy a:worksForCompany) isVerifiedBy) that means if some wine is tasted by the taster who works for some certification company then this wine is verified by this company. ObjectPropertyChain axioms are represented in metatable OWLPropertyChain. (Figure 3). This table has links to the compound and component object properties and the sequence number of some component property in the property chain.

![Figure 3. Example of OWL wine ontology transformed into relational database](image)

At the last stage of transforming domain ontology into relational database, when whole schema is created, we must convert all assertions of classes and properties into records and fill the database. During this process object property chains can be used to gain some missing information about relations between objects. E.g. if we have both object property assertions isTastedBy and worksForCompany and the axiom SubObjectPropertyOf(ObjectPropertyChain(a:isTastedBy a:worksForCompany) isVerifiedBy) on some instance, we can create object property assertion and insert the appropriate value in the column isVerifiedBy of the table Wine automatically.

Ontology data properties relate individuals to literals. Functional data properties can be mapped to relational database columns of the tables corresponding to the domain classes of these properties. Because the OWL 2 was extended for representing ranges of data properties by the XML schema datatypes, we map XML schema datatypes to corresponding SQL datatypes. In a case of the data property is not functional or it has cardinality more than one, the data property is mapped to the additionally created table named by the data property name. This additional table has three columns – the auto increment identification number, the foreign key to the table of the corresponding domain class of this property and the value. The value column is SQL datatype corresponding to the XML schema datatype of the data property.
OWL 2 provides a new construct “HasKey” which allows keys to be defined for a given class. With this construct it is possible to give a list of object or data properties, which together identify resources of a given type. For example, if individuals of the class “Wine” are uniquely identified by data properties “wineName”, “wintageYear” and the object property “hasWineMaker”, then the OWL 2 axiom HasKey(wine:Wine :wineName :wintageYear :hasWineMaker) states that each named instance of the class “Wine” is uniquely identified by this set of properties – that is, if two named instances of the class coincide on values for each of key properties, then these two individuals are the same.

For converting the OWL ontology description to the RDB schema, we map the “HasKey” axiom on some properties for the certain class to the uniqueness constraint of columns of the corresponding table. Depending on “HasKey” properties count (one or many), we create the unique key on the single column, or the multi column (combination of columns) unique index of the table.

3.3 OWL Restrictions

In OWL 2 class expressions can be formed by placing restrictions on object property expressions. The ObjectSomeValuesFrom(OPE CE) class expression allows for existential quantification over an object property expression OPE, and it contains those individuals that are connected through an object property expression OPE to at least one instance of a class expression CE. The ObjectAllValuesFrom(OPE CE) class expression allows for universal quantification over an object property expression OPE, and it contains those individuals that are connected through an object property expression OPE only to instances of a class expression CE. The ObjectHasValue(OPE a) class expression contains those individuals that are connected by an object property expression OPE to a particular individual a. Finally, the ObjectHasSelf(OPE) class expression contains those individuals that are connected by an object property expression OPE to themselves.

When we are converting the OWL ontology description to the relational database schema we save this information in special metadata tables. ObjectAllValuesFrom, ObjectSomeValuesFrom and ObjectHasValue restrictions have their own metadata tables with column restrictedProperty which links to the table OWLObjectProperties. Metadata tables for ObjectAllValuesFrom and ObjectSomeValuesFrom restrictions also have column restrictionClass, which points to the table of the corresponding restriction source class (Figure 3). The ObjectHasValue restriction metadata table has the column “Value” for storing the value of the restricted resource of the corresponding property. Indication that object property has ObjectHasSelf restriction is saved in the column hasSelf of the OWLObjectProperties metatable.

Object property restrictions in OWL 2 can also be formed by placing restrictions on the cardinality of object property expressions ObjectMinCardinality, ObjectMaxCardinality, and ObjectExactCardinality that are saved in the metadata table OWLCardinality.

4 Querying OWL 2 Ontologies from Relational Databases

In this section we present the prototype for querying ontology, stored in a relational database according to the representation we have proposed. Usually, ontology reasoner (e.g. Pellet) reads ontology, including individuals, from a XML file (Figure 4). In our case, only ontology classes, their hierarchies, object and data properties, axioms and restrictions are extracted into a memory. Individuals are accessed by SQL queries obtained by converting fragments of SPARQL to SQL. The Ontology Database Integration component creates ontology model for the reasoner. This component analyses the database schema and metatables, builds the ontology model, rewrites SPARQL queries and executes SQL for obtaining results. The algorithm of transforming the database to ontology is based on the features of the previously described transformation from ontology into the database schema.

The following SPARQL query finds all potable liquids and their makers verified by the certain company:

```sparql
select ?type ?drink ?maker where
 ?drink wine:isVerifiedBy wine:French_Certification_Company}
```

The presented query has four conditional clauses. According to the 1st clause, Pellet OWL Reasoner finds all subclasses of the PotableLiquid class according to predicate ?type rdfs:subClassOf wine:PotableLiquid,
where FILTER ensures finding of proper subclasses of the PotableLiquid (Pellet OWL Reasoner treats every class as a subclass of itself). The first clause of the query (?type rdfs:subClassOf wine:PotableLiquid.FILTER(?type!= wine:PotableLiquid)) returns the single record Wine and assigns it to the variable ?type. SQL executes the remaining clauses. The second clause ?drink rdf:type ?type rewritten to SQL finds all individuals of the class Wine. The 3rd clause ?drink wine:hasMaker ?maker finds all makers of all previously found individuals and assigns them to the variable ?maker. The example of SQL query that finds the maker "Marieta" of the drink "MariettaPetiteSyrah" (WineId=3):

```
SELECT MakerName, MakerId FROM PotableLiquid, Maker, Wine
WHERE PotableLiquid.hasMaker = Maker.MakerId and
PotableLiquid.potableLiquidId = Wine.WineId and Wine.WineId = 3
```

The last, 4th clause ?drink wine:isVerifiedBy wine:French_Certification_Company rewritten into SQL filters selected individuals by checking which of them is certified by the certain certification company. The sample data and results of the query are presented in Figure 5.

### Figure 5. Results from querying relational data representing Wine ontology

#### Data in Relational Database

<table>
<thead>
<tr>
<th>PortableLiquid</th>
<th>Maker</th>
<th>WineMaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>potableLiquidId</td>
<td>potableLiquidName</td>
<td>hasMaker</td>
</tr>
<tr>
<td>1</td>
<td>FoxenCheninBlank</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>MariettaZinfandel</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>MariettaPetiteSyrah</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Orange Juice</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wine</th>
<th>CertificationCompany</th>
</tr>
</thead>
<tbody>
<tr>
<td>wineId</td>
<td>isTastedBy</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Query results

```
?type ?drink ?maker
Wine FoxenCheninBlank Foxen
Wine MariettaPetiteSyrah Marietta
```

### 5 Conclusions and Future Work

In this paper we presented the mapping for transforming ontologies described in OWL 2 to relational database schemas. These mappings extend our previous transformation, oriented to OWL 1, with new concepts and offer more possibilities for representing the rich knowledge about a problem domain. Our OWL2RDB transformation combines direct mapping of ontology classes, properties and instances to database schema with representing axioms and restrictions in metadata tables. Our transformation is capable of the lossless representation of the chosen subset of ontology concepts in a database and the lossless retrieval of ontology schema from the database into ontology reasoning tools.

Our approach is well-suited for creating new databases from ontologies and creating ontologies for already existing databases. We argue that it is worth to store ontology structures in databases for ontology management purposes. We have tried a prototype of a tool for extracting ontologies from relational databases, satisfying our schema, and allowing the step-wise processing of SPARQL queries where SPARQL is used for querying ontology structures in a main memory and SQL is used for querying instances in the database.

Currently, we are working on the extension of our transformation tool and are willing to provide the transformations of the subset of OWL 2 concepts sufficient for representation of ontologies appropriate for applications of information systems. The OWL 2 QL profile that is oriented towards efficient implementation of tools working with ontologies stored in databases is unsuitable for real needs of information systems. OWL 2 QL profile has strong restrictions and is capable of working only with very simple ontologies when the actual applications require capturing business rules and transforming them to software code, integrating data from distributed resources, effectively communicating on the World Wide Web etc.

### References


