

REVERSIBLE LOSSLESS TRANSFORMATION FROM OWL 2 ONTOLOGIES INTO RELATIONAL DATABASES

Ernestas Vyšniauskas, Lina Nemuraite, Rimantas Butleris, Bronius Paradauskas

*Kaunas University of Technology, Department of Information Systems
Studentų St. 50-308, LT-51368 Kaunas, Lithuania*

e-mail: vernest@email.lt, lina.nemuraite@ktu.lt, rimantas.butleris@ktu.lt, bronius.paradauskas@ktu.lt

crossref <http://dx.doi.org/10.5755/j01.itc.40.4.979>

Abstract. The goal of the paper is to define reversible, information preserving transformation from OWL 2 ontologies into relational databases using our proposed hybrid approach, when a part of ontology constructs is directly represented by relational database structures and another part having no direct correspondences in a relational database is stored in metadata tables. The desirable transformation is defined in QVT Relations language following additional requirements under which such transformation is reversible and does not lose semantic information when performing from ontology to database and backward.

Keywords: Ontology, relational database, OWL 2, RDB, reversible lossless transformation, QVT Relations language.

1. Introduction

Ontology comes from philosophy and defines fundamental concepts, relationships and constraints of the problem domain. Its technological embodiment, the Web Ontology Language OWL [1], now OWL 2 [2], is the indispensable means for representing human-readable and machine-processable semantics in the Semantic Web. The problem of storing large ontologies is not new and currently it attracts more and more attention. Native RDF repositories as OWLIM [3] and commercial database management systems as Oracle [4] are following the mentioned target, apart other different implementations and approaches for representing OWL ontologies using relational database constructs. Each of these approaches has its advantages and drawbacks; none of them is recognized as a standard. Recently, the W3C Consortium has announced “The Use Cases and Requirements for Mapping Relational Databases to RDF” [5]. Naturally, all these efforts and achievements should be united for having at least a common reference and understanding how different representations could interact for reaching the vision of “Linked data” [6].

In 2006, we have proposed a hybrid approach for representing OWL ontology in relational database when a part of ontology constructs is directly represented by relational database structures, and another part is stored in metadata tables [7]. The approach was extended for OWL 2 [8]. As the quality of mapping between ontologies and database belongs on capability to perform queries, we conducted an

experiment with 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 was used for querying ontology structures in a main memory and SQL was used for querying instances in the database [9]. The experiment has shown that 1) it is possible to restore the original ontology from a database created using the hybrid approach; 2) query performance times of our proposed method were better than using the memory based methods when ontology is stored using a native storage.

The limitation of our approach is the fact that ontologies undergo changes, and database structures representing ontology concepts are obliged for changing too. Metadata tables do not belong on these changes; it is an advantage of the hybrid approach. Also, the intuition tells that changes of ontology axioms represented in metadata tables are occurring much faster than changes in tables representing classes and properties of ontology. Nevertheless, the ensuring of lossless transformations in the sense of preserving information is a very important issue when ontology or database evolves.

For firming our approach, we present the specification of $OWL2_{TORDB}$ transformation in QVT Relation (QVT-R) language that is capable for defining bidirectional transformations. Information preserving transformations have additional requirements with regards to which we propose a reversible transformation that does not lose semantics during performing forward from ontology to database and backward.

The rest of the paper is organized as follows. In Section 2, the overview of related works is presented. Section 3 introduces OWL 2 and RDB metamodels and the overall $OWL2TORDB$ transformation. Section 4 analyses transformations of OWL 2 classes and class axioms; Section 5 – OWL 2 object properties, property axioms and restrictions. Section 6 summarizes the transformation of remaining OWL 2 constructs. Section 7 analyses the $OWL2TORDB$ transformation with regards to requirements of being a lossless transformation in the information preserving sense. Section 8 gives conclusions and highlights the future work.

2. Related Works

OWL 2, a new version of OWL, has occupied the positions of its predecessor straight after its occurrence in 2009 [2], [10]. OWL 2 offers new constructs for describing properties such as qualified cardinality restrictions; property chain axioms; 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 their restrictions etc. When extending our previous mapping, we used the same hybrid approach: direct mappings of OWL 2 class and property concepts with RDB concepts, and storing the problematic (in mapping sense) knowledge in metatables.

There are a lot of approaches for representing ontologies in databases [11]–[21] and inverse mappings from databases to ontology [22]–[26]. We can classify ontology storage models into three main cases [9]: storing ontology and its instances in the same manner (one or three tables); storing ontology concepts in database schema corresponding to full or partial ontology metamodel; and 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 storage model does not lose information, but it uses advantages of relational databases just for saving many records, so the performance of queries to ontology information normally should be slow e.g. [14]. As an exception, the similar method is highly powered in Oracle Semantic Storage as it is supported with the native functionality of the Oracle database and optimizing techniques [4]. The second approach does not lose information, but it is oriented at storing ontologies and does not consider their relation to existing databases and advantages of database management systems [11], [15], [18], [21]. The third approach unites capabilities of ontologies and database management systems (e.g. [16], [17]). However, existing methods of that kind are losing semantics as they do not cover the sufficient subset of ontology concepts. Our proposed method intends to fill this gap by showing that it is possible to cover all ontology constructs by storing them in

metadata tables, and to perform efficient querying by retaining ontology classes and properties as native RDB constructs.

There are a lot of approaches to defining transformations: the Z notation [27]; abstract syntax trees and attributes [28]; atomic and composite transformation models [29]; two-level transformations [30] etc. Every approach has its rationale, but for our work reversible, information preserving transformations are of great importance. Such transformations are vital for validation and evolution of ontology or database. First, the question about information preserving in $OWL2TORDB$ transformation arises as not every ontology construct may be directly mapped to a relational database. Secondly, ontologies and databases are evolving in time. Recurrent $OWL2TORDB$ and $RDBTOOWL2$ transformations should not damage existing data and ensure the coherent performance.

The simplest way to ensure reversible, lossless transformations is to implement transformations in both directions and to test them. We have partially done this with our first prototype [7] and its extension [9]; now we are trying to surely verify the $OWL2TORDB$ transformation by defining it in a suitable, explicit transformation language. As candidate languages, ATL [31], QVT [32], formal language for model transformation specification [33], a use of predicates and functions [34], and triple graph grammars [36] were examined. The ATL is a popular but not bidirectional language; [33] presents promising, but undeveloped specification; [34] approach requires defining a lot of predicates and functions and is too cumbersome for complex transformations. Also, except QVT and triple graph grammars, all these languages are not suitable for bidirectional transformations.

The most of formal methods related to bidirectional transformations are based on graph grammars. Ehrig et al. presented a formal proof of the sufficient requirement for reversible transformations [35]. This requirement is based on the “notion of source transformation which is the projection of a triple graph transformation to its source component. It is sufficient to show that a source structure can be constructed by source transformations only”. In this case, the forward transformation is “source consistent” and it “can be inverted, i.e. there is a backward transformation leading back to the same source structure as the original one”. It means, the projection of transformation on the source model should cover all source constructs. However, this requirement is sufficient to bijective transformations only whereas most of practical transformations are not bijective, and $OWL2TORDB$ is so too.

The problem of defining information preserving, bidirectional and not bijective model transformations was analysed by Stevens [36]. She points that currently there is no transformation language that could guarantee the lossless transformations per se, this has to be verified. Stevens acknowledges the QVT-R language as the most suitable language for

defining bidirectional transformations and formulates the requirements for ensuring the coherent (not bijective) transformations. In Section 3, we present these criteria in more detail as they have an impact on the way transformations $OWL2ToRDB$ are constructed.

We are considering the reversible lossless transformation as a limited bidirectional transformation that was defined by Hainaut [37]: a transformation $T1: M \times N \rightarrow N$ is reversible iff for all instances m of source models M exists direct transformation $T1$ into instances n of target models N and also there exists a reverse transformation $T2: M \times N \rightarrow M$ such that $\forall m \in M \Rightarrow T1(m, n) = n, n \in N; \forall m \in M \Rightarrow T2(m, T1(m, n)) = m, n \in N$.

Transformation $T1$ is reversible, but not vice versa. That means for any arbitrary instance $n \in N$ may not satisfy $T1(T2(m, n), n) = m$. If $T2$ is reversible as well, then $T1$ and $T2$ are called symmetrically reversible and comprise a bidirectional lossless transformation T . Our $OWL2ToRDB$ transformation is reversible and lossless, but not bidirectional and lossless one as the latter currently seems too big challenge because of different nature of ontologies and databases. Transforming arbitrary database into ontology is rather reverse engineering task, requiring human intervention [38]; it may be very complex taking into account procedural components such as triggers and procedures.

When defining transformations, we use the OWL 2 metamodel [2] for representing ontology. For a relational database, we use the Common Warehouse Metamodel (CWM) [39], modified by eliminating its procedural elements. Also, we have analysed the ontology definition metamodel [40], which defines the (partial) QVT_R1.0 transformation between OWL1.1 and UML2.1.2.

3. The overall transformation between OWL 2 and relational metamodels

For transforming OWL 2 ontologies into database schemas, we use the original OWL 2 metamodel [2] as a source. Here we present only excerpts of that metamodel for explaining presented transformations. Figure 1 presents the top structure of OWL 2 metamodel – ontology, its annotations and axioms.

Axiom is a main instrument to define OWL 2 semantic constructs. OWL 2 building blocks are entities (i.e. classes, object properties, annotation properties, data properties, named individuals, and datatypes) that comprise the vocabulary or signature of ontology (Figure 2). One can declare an entity by stating an axiom. Conversely, annotations have no semantics but serve as a powerful means for associating additional information with ontologies, entities, and axioms.

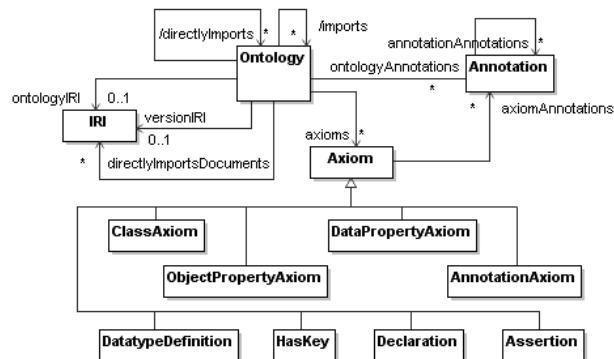


Figure 1. The OWL ontology structure and axioms

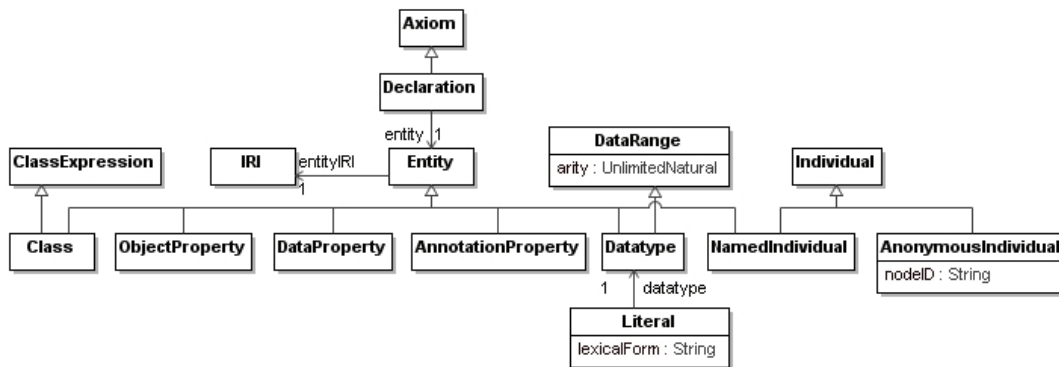


Figure 2. OWL 2 entities

As a target metamodel, we use a subset of CWM metamodel [39], obtained by eliminating procedural components of CWM and supplementing it with metatables (class `Metatable`) for preserving ontology elements having no corresponding constructs in the

relational model. Also, we introduced `SchemaName`, `TableName` and `ColumnName` (same as a `KeyName`) for database schema names regarding possibly different rules for creating them (e.g. names of schemas and tables start with capital letters etc) (Figure 3).

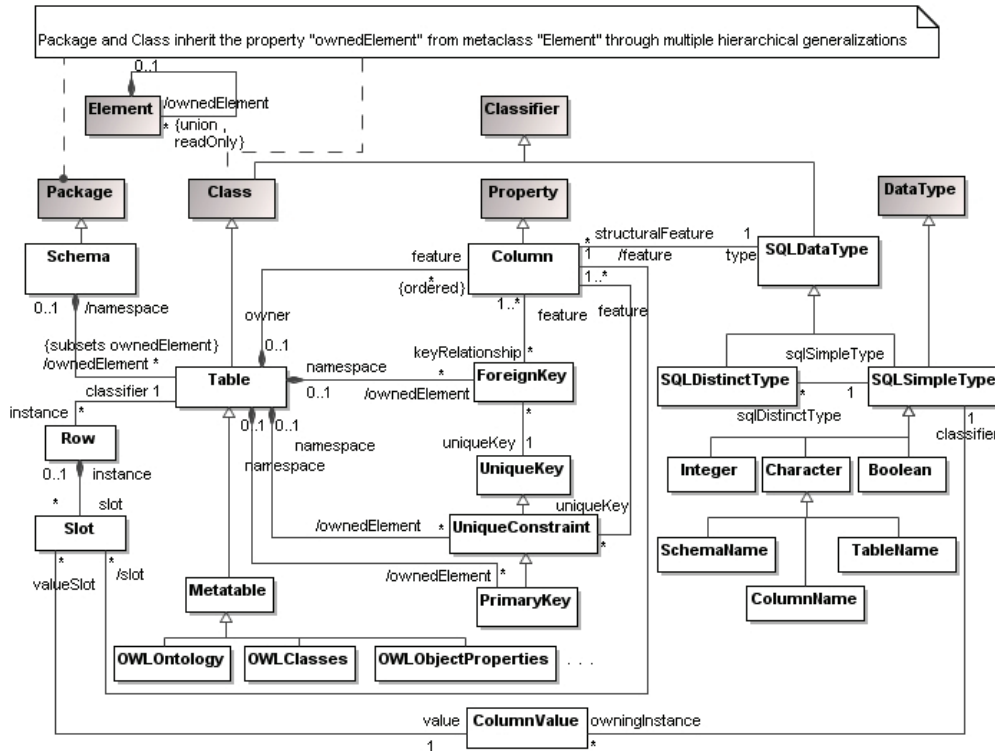


Figure 3. Simplified Relational metamodel (adopted from [39])

First we present the overall transformation owl2ToRDB written in QVT Relations language [32]. For brevity, we assume that model of ontology under transformation, owl, has no importing ontology, does not import other ontologies etc. Also, we assume that all names of entities are represented as fullIRI and OWL2 classes are inserted in stead of missing domains or ranges of properties where appropriate [2]. During this transformation, ontology is transformed into RDB schema, ontology IRI is transformed into RDB schema name, metatables are created, and a row is inserted into metatable OWLOntology for preserving ontology IRI and prefix (Figure 4).

```

transformation owl2ToRDB(owl:OWL2, rdb:RDBM)
{
  key Ontology{iri};
  key Entity{iri};
  key Schema{name};
  key Table{name, namespace};
  key Column{name, owner};
  key PrimaryKey{name, namespace};
  key ForeignKey{name, namespace, uniqueKey,
  uniqueKey.namespace};
  key OWLClasses{classId}|{className};...;
  top relation OntologyToSchema
  {
    checkonly domain owl ont:Ontology
    {ontologyIRI=iri:IRI};
    enforce domain rdb schem:Schema{
      name=sn:SchemaName,
      ownedElement->{classes:Metatable{
        name='OWLClasses',
        feature->{mc1:Column{
          name='classId', type=Integer},
          mc2:Column{name='className',
          type=Character},
          mc3:Column{name='superClass',
          type=Integer}},
        ownedElement->{pk:PrimaryKey{
          name='classId, feature->first()=mc1},
          uk:UniqueConstraint{
            name='ClassName',

```

```

      feature->first()=mc2},
      fk:ForeignKey{name='SuperClassId',
      feature->first()=mc3}},
      mont:Metatable{name='OWLOntology',
      feature->{mc1:Column{name='ontologyId',
      type=Integer},
      mc2:Column{name='prefixIRI',
      type=Character}},
      instance{x1:Row{
        slot->{s11:Slot{feature=mc1,
        value=genUniqId(mont)},
        s12:Slot{feature=mc2,
        value=iri.prefix().qNameToChar()}},
      disjointClasses:Metatable{
        name='OWLDisjointClasses',
        ...}}};
    when {ont.versionIRI->isEmpty();
    ont.imports->isEmpty();
    ont.directImports->isEmpty()};
    where {IRIToName(iri, sn)};
  } //OntologyToSchema

```

Here we omit the lengthy definition of the remaining metatables; some of them are presented in Figures 4, 8 and 12. Operation genUniqId(table :Table):Integer generates the unique identifier for a row of a table “table”. Operation prefix(iri:IRI):Qname separates prefix from iri, where QName is a QualifiedName from XML Namespaces.

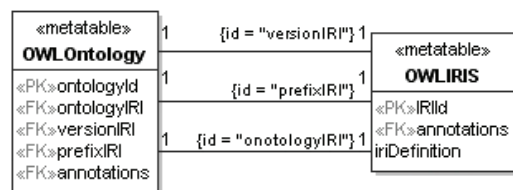


Figure 4. Metatables for keeping information about ontologies and their IRIs

A large subset of OWL 2 axioms is transformed into rows of metatables; e.g. `SubClassOf` is presented later in the paper.

```

top relation AxiomToMetaRow
{
  checkonly domain owl ax:Axiom{
  opposite(Ontology.axioms)=ont:Ontology};
  enforce domain rdb mrow:Row{
  namespace.namespace=schem:Schema}};
  when {OntologyToSchema(ont,schem)};
  where {ClassAxiomToMetaRow(ax,mrow);
  ObjectPropAxiomToMetaRow(ax,mrow);
  DataPropAxiomToMetaRow(ax,mrow)}
} //AxiomToMetaRow
    
```

The important requirement for `owl2ToRdb` transformation being bidirectional and lossless is a possibility to convert database schema names into IRIs. It means that we should preserve a prefix when converting OWL 2 into RDB, or define a prefix if we wish to start a transformations from a database having no ontology. The transformation `IRIToName` itself is a simple one; however, the problem may be in defining a prefix if we wish to transform a freely chosen database schema that was not obtained by transforming some existing ontology. In such a case, we leave choosing a prefix as a decision. When transformation starts from the left side, the transformation `IRIToName` can be specified as operation in OCL:

```

context owl2ToRDB:
  IRIToName(iri:IRI,sn:Character):Character
  post sn=let
    n=iri.prefix().qNameToString().size(),
    m=iri.iriToString().size() in
    sn=iri.iriToString().substring
      (lower=n+1,upper=m).stringToChar(),
    
```

where operations `qNameToString()`, `iriToString()`, `stringToChar()` (and others, as `qNameToChar()` used in relation `OntologyToSchema`) perform conversions of corresponding data types.

4. Transforming OWL 2 classes and class axioms

4.1. OWL 2 classes

In OWL 2, classes and property expressions are used to construct class expressions that represent sets of individuals by formally specifying conditions on the individual properties; individuals satisfying these conditions are instances of the corresponding class expressions. OWL 2 provides axioms that allow establishing relationships between class expressions (Figure 5).

When we are converting the OWL 2 ontology description to relational database schema, we map one ontology class to one database table.

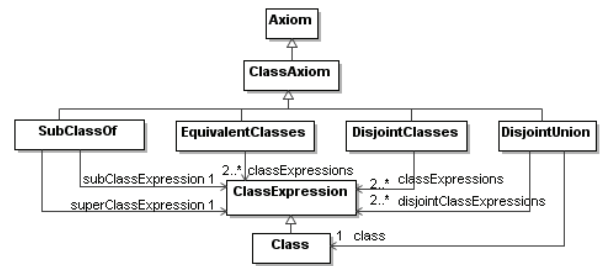


Figure 5. The OWL 2 metamodel for class axioms [2]

As the name of an ontology class is unique in the ontology, and instances of the ontology class have unique names, we can automatically create a primary key for corresponding table and name the primary key column by adding some suffix to the class name, e.g. “Id”. Also, we create the additional column by adding “Name” suffix to the class name for saving names of instances of the class. This mapping and the example are illustrated in Figure 6.

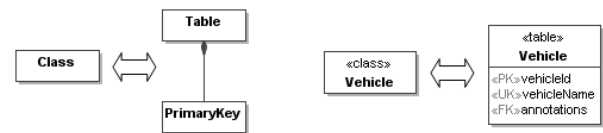


Figure 6. Illustration of transforming OWL 2 class into RDB table

4.2. OWL 2 class axioms

The fundamental taxonomic construct for classes is the `SubClassOf` axiom, which allows one to state that each instance of one class expression is also an instance of another class expression. It relates the specific class to the more generic class and enables to construct the hierarchy of classes. When transforming OWL 2 ontology representation into a relational database schema, we create one table for every class in ontology with one-to-one (1:0..1) relationships between tables representing classes and their subclasses defined by the axiom `subClassOf` (Figure 7). Here each value of the primary key of the supertable can appear as the foreign key (and also as the primary key) for at most one row of the subtable. The subclass does not need a column for saving names of its instances, because the instance of the subclass is also the instance of the superclass and its name is saved in the superclass table.

The `EquivalentClasses` axiom allows one to state that several class expressions are equivalent to each other, i.e. these classes have precisely the same instances. The `DisjointClasses` axiom states that several class expressions are pairwise disjoint – an individual that is a member of one class cannot simultaneously be an instance of the other class. The `DisjointUnion` axiom allows constructing a class as a disjoint union of several class expressions.

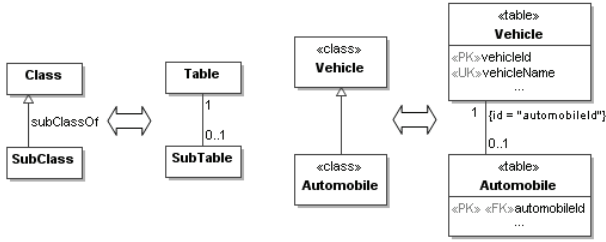


Figure 7. Illustration of transforming OWL SubClassOf axiom into RDB

We suggest saving this information in special metatables. All classes of the ontology are saved in OWLClasses metatable with two main columns classId, which is an automatically generated unique identification number, and className, which saves the unique name of the class (Figure 8).

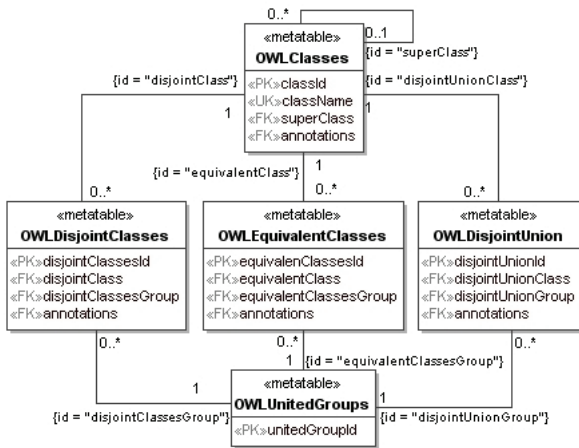


Figure 8. Metaschema of OWL 2 class axioms

Also, the metatable OWLClasses has the column that is a foreign key named superClass which saves information about hierarchy of OWL classes, and the column – foreign key named annotation for relating a class with a set of its annotations. Information about groups of disjoint, disjoint union and equivalent classes is saved in metatables OWLDisjointClasses, OWLDisjointUnion and OWLEquivalentClasses (Figure 8). These tables have automatically generated unique primary keys, foreign keys from OWLClasses table and identification numbers of groups that are represented by foreign keys from metatable OWLUnitedGroups. Equivalent, disjoint and disjoint union classes have the same group identification numbers that are created during the transformation process.

OWL 2 provides a new construct HasKey which allows keys to be defined for a given class. This construct gives a list of object or data properties, which together uniquely identify named instances of a given class. For example, if individuals of the class Automobile are uniquely identified by data properties modelTitle, produceYear and the object property isProducedBy, then the OWL 2 axiom HasKey (:Automobile :modelTitle :produceYear :isProducedBy) states that each named instance of

the class Automobile 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 HasKey axiom on some properties for the certain class to the uniqueness constraint on columns of the corresponding table, Depending on the number of HasKey properties (one or more), we create the unique key on the single column, or on a combination of columns of the table.

4.3. Defining class and class axiom transformations in QVT-R

Transformation ClassToTable transforms common features inherent for all OWL 2 classes. It generalizes transformation TopClassToTable that transforms classes having no superclass expressions. Features of classes having superclass expressions are added to the corresponding table by transformation SubClassRelationToFK. Also, the transformation ClassToTable creates a row r for each class c1 in the metatable OwlClasses.

```

top relation ClassToTable
{
  checkonly domain owl cl:Class{
    entityIRI=ciri:IRI{ },
    opposite(Axiom.entity) .
  }
  opposite(Ontology.axioms)=ont:Ontology;
  enforce domain rdb t:Table{
    name=tn:TableName,
    namespace=schem:Schema,
    feature->{mcl:Column{
      name=tn+'Id', type=Integer};
    }
  }
  enforce domain rdb mr:Row{
    classifier=mt:Metatable{
      name='OWLClasses',
      slot->{s1:Slot{feature=col1:Column{
        name='classId',
        value=genUniqId(mt)},
        s2:Slot{feature=col2:Column{
          name='className', value=tn}
        }
      }
    }
  }
  when {OntologyToSchema(ont, schem);};
  where {tn=IRIToName(ciri, tn);
    TopClassToTable(cl, t);};
}
//ClassToTable

```

Transformation of classes having no superclass expressions is further refined by the transformation

TopClassToTable:

```

relation TopClassToTable
{
  checkonly domain owl cl:Class{
    entityIRI=ciri:IRI{ },
    opposite(Axiom.entity) .
  }
  opposite(Ontology.axioms)=ont:Ontology;
  enforce domain rdb t:Table{
    name=tn:TableName,
    namespace=schem:Schema,
    feature->{tcl1:Column{
      name=tn+'Name', type=Character};
    }
    ownedElement->{pk:PrimaryKey{
      name=tn+'Id',
      feature->select(pkcl:Column|
        pkcl.name=tn+'Id'),
      uk:UniqueConstraint{name=tn+'Name',
        feature->select(tcl1:Column|
          tcl1.name=tn+'Name')}};
    }
  }
  enforce domain rdb msl:Slot{
    feature=col2:Column{
      name='superClass', value='NULL'},
    instance=mr:Row{
      classifier=mt:OWLClasses{
        namespace=schem,
        slot->exists(msl1|msl1.feature.name=
          'className' and msl1.value=tn)}};
  }
}

```

```

when {ClassToTable(c1,t);
    cl.opposite(SubClassOf.
        superClassExpression)→isEmpty();
} //TopClassToTable
    
```

Transformation of class `c1` having a superclass expression is refined by the transformation `SubClassOfAxiomToFK`, which creates a primary key `pk` for a table representing a subclass `scl`. This `pk` also is a foreign key from the table representing the corresponding superclass. Also, the name `stn` of the table, which represents superclass `scl`, is inserted into the row representing class `c1` in metatable `OWLClasses`.

```

top relation SubClassOfAxiomToFK
{ checkonly domain owl sub:SubClassOf{
    subclassExpression=c1:Class{
        entityIRI=c1n:IRI},
    superClassExpression=scl:Class{
        entityIRI=scln:IRI},
    opposite(Ontology.axioms)=ont:Ontology};
enforce domain rdb fk:ForeignKey{
    name=tn+'Id',
    namespace=t:Table{name=tn,
        namespace=schem:Schema},
    feature→first(=fcol:Column,
        uniqueKey=pk:PrimaryKey{
            namespace=st:Table{name=stn,
                namespace=schem},
            feature→first(=pcol:Column)});
enforce domain rdb ssl:Slot{
    feature=col2:Column{name='superClass',
        value=opClassId(stn)},
    instance=mr:Row{
        classifier=mt:OWLClasses{
            namespace=schem},
        slot→exists(msl1|msl1.feature.name=
            'className' and msl1.value=tn)}};
when {OntologyToSchema(ont,schem);
    ClassToTable(c1,t);
    ClassToTable(scl,st);
    t.feature→exists(clmn|
        clmn.name=tn+'Id' and fcol=clmn);
    st.feature→exists(sclmn|
        sclmn.name=stn+'Id' and pcol=sclmn);
where {IRIToName(c1n,tn);
    IRIToName(scln,stn);
} // SubClassOfAxiomToFK
    
```

For finding a row of a class `c1` in the metatable `OWLClasses`, the operation `opClassId(cn:Character):Integer` is defined that returns a value of the identifier of the row, which has the value of the column `className` equal to the `cn`, in the metatable `OWLClasses`:

```

context OWLClasses::opClassId(cn:Character):
    Integer
post result=self.instance→select(crow:Row|
    crow.slot→exists(csl:Slot|
        csl.feature.name='className' and
        csl.value=cn))→select(isl:Slot|
        isl.feature.name='classId').value}
    
```

5. Transforming OWL 2 Object Properties, Object Property Axioms and Restrictions

5.1. OWL 2 object properties

OWL provides axioms for characterizing and establishing relationships between object property expressions.

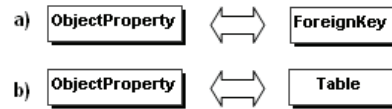


Figure 9. Illustration of transforming OWL 2 object properties into foreign keys or tables

The object property in OWL 2 ontology relates the individual to other individuals. Depending on the cardinality of the object property (or the object property is functional or not), or existing a class with the same IRI as the object property, we transform the object property expression into the foreign key corresponding to one-to-many (1:0..*) relationship, or into the table corresponding to many-to-many (0..*:0..*) relationship between classes (Figure 9).

Metamodel of part of `ObjectProperty` axioms is presented in Figure 10. The `ObjectPropertyDomain` and `ObjectPropertyRange` axioms can be used to restrict the first and the second individual, connected by an object property expression, to be instances of the domain and range class expressions.

The `FunctionalObjectProperty(OPE)` axiom allows one to state that an object property expression `OPE` is functional – that is, that for each individual x , there can be at most one distinct individual y such that x is connected by `OPE` to y (Figure 16). This axiom is a syntactic shortcut for the axiom `SubClassOf(owl:Thing ObjectMaxCardinality(1 OPE))`.

So, if the OWL 2 object property is functional, or `ObjectMaxCardinality` or `ObjectExactCardinality` of the object property is ≤ 1 , and the object property IRI does not match an IRI of any class in that ontology, then the foreign key corresponding to one-to-many (1:0..*) relationship between tables is created. The possibility for having a class and an object property or an individual with exactly the same IRI is a new feature of OWL 2 called “punning”. Punning allows, for example, defining properties for OWL 2 object properties by attributing these properties to a class having the same name as the object property under consideration.

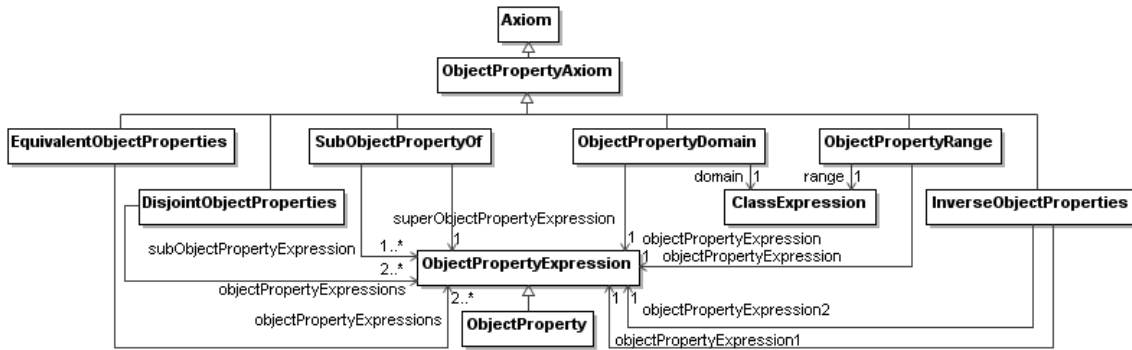


Figure 10. Fragment of the of OWL 2 metamodel representing part of object properties and object property axioms

If the object property is not functional, or its exact, min or max cardinality is > 1 , or it has no cardinality restrictions, or if the object property IRI matches the IRI of some class in that ontology, then the object property is transformed into the table corresponding to many-to-many ($0..*:0..*$) relation between classes. In a case of punning, a table is needed for ($1:0..*$) relation for preserving properties of a class with IRI matching the object property IRI.

5.2. Object property axioms

In OWL2 there are two forms of object subproperty 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 `Vehicle` has the object property `hasMaker`, and the class `Automobile` has the object property `isProducedBy`, which is the subproperty of the property `hasMaker` (Figure 11). Information that one property is a subproperty of another property we save in the metatable `OWLObjectProperties` (Figure 12).

Another form of OWL 2 object subproperty axiom 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 `Automobile` and the object property `isVerifiedBy` with the range class `Assurer`. The class `Assurer` has the object property `employedBy` with the range class `InsuranceCompany`. We can declare the axiom `SubObjectPropertyOf (ObjectPropertyChain (a:isVerifiedBy a:employedBy) isInsuredBy)` that means if some automobile is verified by the assurer employed by some insurance

company, then this automobile is insured by this company (Figure 11).

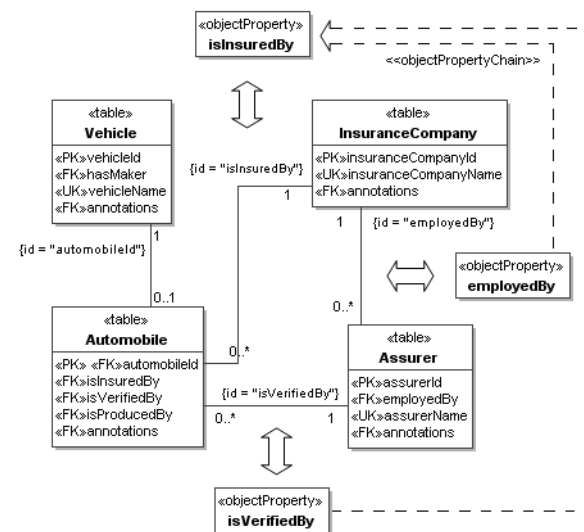


Figure 11. Illustration of transforming OWL 2 object property chains

Object property chains can be used to derive some additional information about relationships between objects. E.g. if we have both object property assertions `isVerifiedBy` and `employedBy` and the axiom `SubObjectPropertyOf (ObjectPropertyChain (a:isVerifiedBy a:employedBy) isInsuredBy)` on some instance, we can derive the object property assertion and automatically insert the appropriate value in the column `isInsuredBy` of the table `Automobile` during filling the database with instances.

`ObjectPropertyChain` axioms are represented in metatable `OWLObjectPropertyChains` (Figure 13). This table has links to the compound and component object properties, and the sequence number of each component property in the property chain.

ObjectMinCardinality, ObjectMaxCardinality, and ObjectExactCardinality contain those individuals that are connected by an object property expression to at least, at most, and exactly a given number of instances of a specified class expression, respectively. E.g. a minimum cardinality expression ObjectMinCardinality (n OPE CE) consists of a nonnegative integer n , an object property expression OPE, and a class expression CE, and contains all those individuals that are connected by OPE to at least n different individuals that are instances of CE. Similarly, the cardinality constraints ObjectMaxCardinality and ObjectExactCardinality are defined.

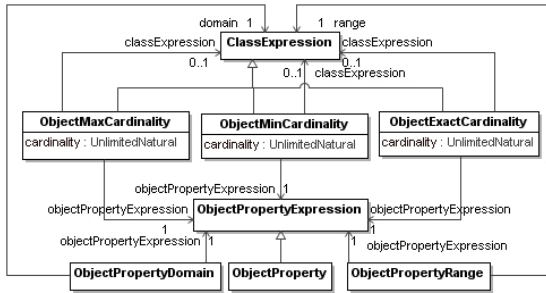


Figure 15. Metamodel of object property cardinality restrictions [2]

Cardinality restrictions of object properties are saved in the metatable OWLObjectCardinality. It has the column restrictedObjectProperty, which links to OWLObjectProperties table, and three additional nullable columns for each type of OWL cardinality restriction (Figure 14). E.g. if we have the object property with the cardinality restriction ObjectExactCardinality equal to 3, after transformation this metatable has value 3 in the field objectExactCardinality, and the other two columns objectMinCardinality and objectMaxCardinality have value “NULL”.

5.5. Object property axioms representing characteristics

The metamodel of object property axioms representing characteristics of object properties is presented in Figure 16. The InverseObjectProperty axiom can state that two object property expressions are the inverse of each other. The InverseFunctionalObjectProperty axiom states that an object property expression is inverse-functional. That is, for each individual y there can be at most one distinct individual x such that x is connected by OPE to y . The ReflexiveObjectProperty, IrreflexiveObjectProperty, SymmetricObjectProperty, AsymmetricObjectProperty, and TransitiveObjectProperty axioms allow one to state that an object property expression is reflexive, irreflexive, symmetric, asymmetric, or transitive.

OWL 2 object property axioms representing characteristics of object properties are transformed into values of columns of the rows, representing the

corresponding object properties in the metatable OWLObjectProperties (Figure 13 or Figure 14).

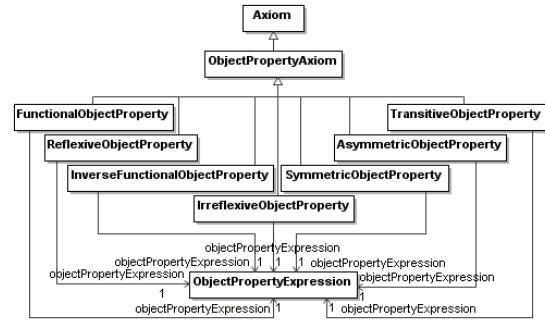


Figure 16. Metamodel of OWL 2 object property axioms defining characteristics

5.6. Defining transformations of object properties, object property axioms and restrictions in QVT-R

OWL 2 functional object properties (or, equivalently, object properties having maximum cardinality value less or equal to 1; or having exact cardinality equal to 1) are transformed into foreign keys and foreign key columns of a table representing a domain class, where foreign key is defined by the primary key of the table representing a range class. The exception arises when ontology has a class c with the same IRI as the object property under consideration, e.g. OPE. Then the result of transforming OPE is merged with a table representing class c (see transformation objectPropertyToTable).

```

top relation ObjectPropertyToFK
{
  checkonly domain owl op:ObjectProperty{
    entityIRI=opn:IRI,
    opposite(ObjectPropertyDomain.
      objectPropertyExpression).
    domain=dcl:Class{entityIRI=diri:IRI,
      opposite(Axiom.entity)}.
    opposite(Ontology.axiom)=ont:Ontology},
    opposite(ObjectPropertyRange.
      objectPropertyExpression).
    range=rcl:Class{entityIRI=riri:IRI,
      opposite(Axiom.entity)}.
    opposite(Ontology.axiom)=ont:Ontology});
  enforce domain rdb fk:ForeignKey{
    name=fkn:ColumnName,
    feature->first()=col:Column{name=fkn;
      type=ColumnName;owner=dt:Table},
    namespace=dt:Table{name=dt,
      namespace=schem:Schema},
    uniqueKey=pk:PrimaryKey{
      namespace=rt:Table{name=rtn;
        namespace=schem},
      feature->first()=rcol:Column{
        name=rtn+'Id',namespace=schem}
      type=ColumnName;
      owner=rt:Table{name=rtn,
        namespace=schem}}};
  enforce domain rdb mr:Row{
    classifier=mt:OWLObjectProperties,
    namespace=schem,
    slot->{s11:Slot{feature=col1:Column{
      name='objectPropertyId',
      type=Integer,value=genUniqId(mt)}},
      s12:Slot{feature=col2:Column{
        name='objectPropertyDomain',
        type=Integer,value=mt.opClassId(dtn)},
      s3:Slot{feature=col3:Column{
        name='objectPropertyRange',
        type=Integer,value=mt.opClassId(rtn)},
      s4:Slot{feature=col4:Column{

```

```

        name='objectPropertyName',
        type=Character,value=fkn}}};
    when (OntologyToSchema(ont,schem);
        ClassToTable(dcl,dt);
        ClassToTable(rcl,rt);
        IRIToName(diri,dtn);
        IRIToName(riri,rtn);
        op.opposite(ObjectPropertyAxiom.
            objectPropertyExpression)→
            exists(fp|fp.ocIsTypeOf
                (FunctionalObjectProperty))or
    let maxcar:ObjectMaxCardinality=
        op.opposite(ObjectMaxCardinality.
            objectPropertyExpression)
    in maxcar.classExpression=rcl
        and maxcar.cardinality<=1 or
    let exactcar:ObjectExactCardinality=
        op.opposite(ObjectExactCardinality.
            objectPropertyExpression)
    in exactcar.classExpression=rcl
        and exactcar.cardinality=1;
    not ont.axiom→exists(ax|
        ax.entity.ocIsTypeOf(Class) and
        ax.entity.entityIRI=op.entityIRI);
    where {IRIToName(opn,fkn)};
} //ObjectPropertyToFK
    
```

OWL 2 object property having maximum, minimum or exact cardinality value greater than 1, or having unrestricted cardinality, or having a class with the same IRI as the object property under consideration is transformed into a table. Such transformation has 2 cases: when a class with the same IRI exists and when not. The semantics of QVT-R allows specifying these cases in one transformation because of its “check and enforce” semantics: if class exists and it is already transformed, only new features are added during the transformation execution. It means, the transformation adds new columns to the existing table.

```

top relation ObjectPropertyToTable
{ checkonly domain owl op:ObjectProperty{
    entityIRI=opn:IRI,
    opposite(objectPropertyExpression).
    domain=dcl:Class{entityIRI=diri:IRI,
    opposite(dcl.entity)}.
    opposite(Ontology.axiom)=ont:Ontology},
    opposite(objectPropertyExpression).
    range=rcl:Class{entityIRI=riri:IRI,
    opposite(rcl.entity)}.
    opposite(Ontology.axiom)=ont:Ontology}};
enforce domain rdb tmn:Table{
    name=tmn:TableName,
    namespace=schem:Schema,
    feature→{cmn1:Column{
        name=tn+'Id',type=Integer},
    cmn2:Column{
        name=opn+'Name',type=Character},
    cmn3:Column{
        name=dn:ColumnName,type=Integer},
    cmn4:Column{
        name=rn:ColumnName,type=Integer}},
    ownedElement→{pmn:PrimaryKey{
        feature→first(=cmn1),
        fkd:ForeignKey{uniqueKey=
            pkd:PrimaryKey(namespace=dt),
            feature→first(=cmn3)},
        fkr:ForeignKey{uniqueKey=
            pkr:PrimaryKey(namespace=rt),
            feature→first(=cmn4)},
        uk:UniqueConstraint{name=tn+'Name',
            feature→first(=cmn2)}};
    enforce domain rdb mr:Row{
        classifier=mt:OWLObjectProperties,
        namespace=schem,
        slot→{sl1:Slot{feature=col1:Column{
            name='objectPropertyId',
            type=Integer,value=genUniqId(mt)}},
            sl2:Slot{feature=col2:Column{
            name='objectPropertyDomain',
            type=Integer,value=mt.opClassId(dtn)}},
    
```

```

        sl3:Slot{feature=col3:Column{
            name='objectPropertyRange',
            type=Integer,value=mt.opClassId(rtn)}},
            sl4:Slot{feature=col4:Column{
            name='objectPropertyName',
            type=Character,value=tmn}}}}};
    when (OntologyToSchema(ont,schem);
        ClassToTable(dcl,dt);
        ClassToTable(rcl,rt);
        IRIToName(diri,dtn);
        IRIToName(riri,rtn);
        not op.opposite(ObjectPropertyAxiom.
            objectPropertyExpression)→
            exists(fp|fp.ocIsTypeOf
                (FunctionalObjectProperty))or
    let maxcar:ObjectMaxCardinality=
        op.opposite(ObjectMaxCardinality.
            objectPropertyExpression)
    in maxcar.classExpression=rcl
        and maxcar.cardinality>1 or
        maxcar.ocIsUndefined() or
    let exactcar:ObjectExactCardinality=
        op.opposite(ObjectExactCardinality.
            objectPropertyExpression)
    in exactcar.classExpression=rcl
        and exactcar.cardinality>1 or
        exactcar.ocIsUndefined() or
    let mincar:ObjectMinCardinality=
        op.opposite(ObjectMinCardinality.
            objectPropertyExpression)
    in mincar.classExpression=rcl
        and mincar.cardinality>1 or
        mincar.ocIsUndefined()
    or ont.axiom→exists(ax|
        ax.entity.ocIsTypeOf(Class) and
        ax.entity.entityIRI=op.entityIRI);
    where {IRIToName(opn,tmn)};
} //ObjectPropertyToTable
    
```

Transformation of object property axioms in many cases results in adding column values to rows representing object property rows in metatables. E.g. functionalObjectProperty, symmetricObjectProperty axiom and others are represented by a column having the same name as the corresponding axiom in the metatable OWLObjectProperties (Figure 12). Equivalent and disjoint object property axioms are represented in separate tables. Each set of equivalent (or disjoint) properties comprise a property group that is represented in the metatable OWLUnitedGroups (Figure 12).

6. The coherence of OWL 2ToRDB transformation

As stated in [36], the coherent transformation is the one that is correct, hippocratic and undoable. The correctness of our transformation means that for every construct of OWL 2 ontology metamodel the direct and reverse transformation exist and they are related by QVT-R relations. The hippocraticness or “check then enforce” means that if transformation is not bijective, then it must look at both source and target models and do not damage them by rewriting already transformed elements. While according to Stevens [36] correctness and hippocraticness are clearly placed on QVT-R in its specification (we consider [32]) and are ensured by constructing relations, undoability also requires that any changes in source and target models could be undone. That means we should not define transformations that lead to irrevocable changes.

For example, changing ontology IRI into RDB schema name would become irrevocable if the operation `IRIToName(iri:IRI,sn:Character):Character` cannot be executed in the reverse order, i.e. a result of operation `NameToIRI(sn:Character,iri:IRI):IRI` is undefined (we dealt with this issue in Section 3). The same requirements were applied to all OWL2TORDB transformations including ones defined as operations in OCL.

Let consider the correctness of the transformation OWL2TORDB. As was mentioned in Section 0, correctness or “source consistency” means that all variety of source constructs is covered by the transformation. For meeting this requirement, first, definitions of source models should be complete though these definitions may be spread over multiple transformations; secondly, different conditions, concerning the different transformations of the same concept, may be specified in several “when” clauses.

For example, the transformation `ClassToTable`, presented in the paper, has two cases: `TopClassToTable`, when class `c` has no superclass, and `SubclassOfAxiomToFK`, when class `c` has a superclass. The transformation `ClassToTable` defines the common properties of classes and their corresponding tables (i.e. OWL2 class is transformed into RDB table, class IRI – into the identifier column of the table and a row in metatable `OWLClasses` with two slot values of columns representing class identifier and class name). `TopClassToTable` adds a column representing the class name, defines a primary key on that column, and adds a slot value “NULL” for a column `superClass` in the row representing that class. The transformation `SubclassOfAxiomToFK` adds properties to tables representing classes having superclasses; clauses “when” of these transformations involve both cases:

```
Context class inv:
self.opposite(SubClassOf.
  superClassExpression) → isEmpty() or
self.opposite(SubClassOf.
  superClassExpression) → notEmpty()
```

Similarly, transformations of object properties and data properties, having the class as their domain, add columns and foreign keys to the table, create rows and slot values for corresponding metatables, etc. Note that here we limit our considerations to a case when OWL2 class may have only one superclass as multiple inheritance is not permitted in well formed ontologies.

7. Conclusions

In this paper we presented the reversible and non-bijective transformations between OWL2 ontology and relational database applying hybrid mapping that combines direct representation of ontology classes, properties and instances in database tables with representing axioms and restrictions in metatables. Aiming at a lossless storing of ontologies in

databases and the lossless retrieval of them into ontology reasoning tools, we presented our transformation in QVT-R language and followed the conditions under which such a transformation is coherent.

To our knowledge, reversible and lossless transformation from OWL2 ontology into relational database was not defined for storing ontology and its instances in different schemas. Currently, we are working on the extension of our previous OWL2TORDB implementation in two directions: 1) to implement complete reversible OWL2TORDB transformations based on the presented QVT-R specification and 2) fulfilment of well-rounded experiments with various ontologies for comprehensively investigating and making further improvements in querying capabilities of the hybrid OWL2RDB approach.

References

- [1] **W3C, 2004.** OWL Web Ontology Language Overview. *W3C Recommendation 10 February 2004*. Available from: <http://www.w3.org/TR/owl-features/> [Accessed 10 Jan 2011].
- [2] **B. Motik, P.F. Patel-Schneider, B. Parsia.** OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. *W3C Proposed Recommendation 22 September 2009*. Available from: <http://www.w3.org/TR/2009/PR-owl2-syntax-20090922/>. [Accessed 10 Jan 2011].
- [3] **Ontotext, 2010.** Bringing the Semantic Web Closer to its Tipping Point. Available from: http://www.ontotext.com/owlim/OWLIM_FactForge_jul10.pdf [Accessed 10 Jan 2011].
- [4] **Z. Wu, G. Eadon, S. Das, E.I. Chong, V. Kolovski, M. Annamalai, J. Srinivasan.** Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. *Proceedings of IEEE 24th International Conference on Data Engineering*, 2008, 1239–1248.
- [5] **W3C, 2010.** Use Cases and Requirements for Mapping Relational Databases to RDF. *W3C Working Draft 8 June 2010*. Available from: <http://www.w3.org/TR/rdb2rdf-ucr/> [Accessed 10 Jan 2011].
- [6] **C. Bizer, T. Heath, T. Berners-Lee.** Linked Data – The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS), Special Issue on Linked Data*, 5(3), 2009, 1–22.
- [7] **E. Vyšniauskas, L. Nemuraitė.** Transforming Ontology Representation from OWL to Relational Database. *Information Technology and Control*, 35(3A), 2006, 333–343.
- [8] **E. Vyšniauskas, L. Nemuraitė, A. Šukys, B. Paradauskas.** Enhancing connection between ontologies and databases with OWL 2 concepts and SPARQL. *Information Technologies 2010: Proceedings of the 16th International Conference on Information and Software Technologies, IT 2010, Kaunas, Lithuania*, 2010, 350–357.
- [9] **E. Vyšniauskas, L. Nemuraitė, A. Šukys.** A hybrid approach for relating OWL 2 ontologies and relational databases. In *P. Forbrig, H. Gunther (Eds.): Perspectives in Business Informatics Research. Proceedings of the 9th international conference, BIR 2010, Rostock*,

- Germany, September 29 – October 1, 2010, Berlin-Heidelberg-New York, Springer, 2010, 86–101.
- [10] **C. Golbreich, E.K. Wallace, P.F. Patel-Schneider.** OWL 2 Web Ontology Language New Features and Rationale. *W3C Proposed Recommendation*, 2009, Available from: <http://www.w3.org/TR/2009/PR-owl2-new-features-20090922/> [Accessed 11 Jan 2011].
- [11] **J. Broekstra, A. Kampman, F. van Harmelen.** Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. In *The Semantic Web – ISWC 2002*, LNCS 2342, Springer Berlin Heidelberg, 2002, 54–68.
- [12] **A. Gali, C.X. Chen, K.T. Claypool, R. Uceda-Sosa.** From Ontology to Relational Databases. In *S. Wang, D. Yang, K. Tanaka, F. Grandi, S. Zhou, E.E. Mangina, T.W. Ling, I.-Y. Song, J. Guan, D.G. Yang, H.C. Mayr (Eds.): Conceptual Modeling for Advanced Application Domains, ER Workshops 2004*, LNCS 3289, Springer, Heidelberg, 2004, 278–289.
- [13] **S. Bechhofer, I. Horrocks, D. Turi.** The OWL Instance Store: System Description. In *R. Nieuwenhuis (Ed.): CADE 2005. LNCS (LNAI)*, 3632, Springer, Heidelberg, 2005, 177–181.
- [14] **J. Lee, R. Goodwin.** Ontology management for large-scale enterprise systems. *Electronic Commerce Research and Applications* 5(1), 2006, 2–15.
- [15] **J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, Y. Pan.** Minerva: A Scalable OWL Ontology Storage and Inference System. In *The Semantic Web – ASWC 2006*, LNCS 4185, 2006, 429–443.
- [16] **I. Astrova, N. Korda, A. Kalja.** Storing OWL Ontologies in SQL Relational Databases. *International Journal of Electrical, Computer and Systems Engineering*, 2007, 1(4), 242–247.
- [17] **C.D. Barranco, J.R. Campana, J.M. Medina, O. Pons.** On Storing Ontologies Including Fuzzy Datatypes in Relational Databases. *IEEE International Proceedings of Fuzzy Systems Conference 2007*, 2007, 1–6.
- [18] **J. Lu, L. Ma, L. Zhang, J.S. Brunner, C. Wang, Y. Pan, Y. Yu.** SOR: a practical system for ontology storage, reasoning and search. *Proceedings of the 33rd International Conference on Very Large Data Bases, Vienna, Austria*, 2007, 1402–1405.
- [19] **B. Motik, I. Horrocks, U. Sattler.** Bridging the Gap Between OWL and Relational Databases. In *WWW 2007, International World Wide Web Conference*, 2007, 807–816.
- [20] **N. Konstantinou, D.M. Spanos, M. Nikolas.** Ontology and database mapping: a survey of current implementations and future directions. *Journal of Web Engineering* 7(1), 2008, 001–024.
- [21] **A. Khalid, A.H. Shah, M.A. Qadir.** OntRel: An Ontology Indexer to store OWL-DL Ontologies and its Instances. *Proc. of 2009 International Conference of Soft Computing and Pattern Recognition*, 2009, 478–483.
- [22] **C.P. De Laborda, S. Conrad.** Relational OWL – A Data and Schema Representation Format Based on OWL. In *Proc. Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005)*. Newcastle Australia CRPIT 43, 2005, 89–96.
- [23] **C.P. De Laborda, S. Conrad.** Database to Semantic Web Mapping using RDF Query Languages. In *Conceptual Modeling – ER 2006, 25th International Conference on Conceptual Modeling, Tucson, Arizona*, LNCS 4215, Springer Verlag, 2006, 241–254.
- [24] **R. Ghawi, N. Cullot.** Database-to-Ontology Mapping Generation for Semantic Interoperability. In *VDBL’07 conference, VLDB Endowment ACM*, 2007, 1–8.
- [25] **G. Hillairet, F. Bertrand, J. Yves, J.Y. Lafaye.** MDE for publishing Data on the Semantic Web. In *Workshop on Transformation and Weaving Ontologies and Model Driven Engineering TWOMDE*, 395, 2008, 32–46.
- [26] **M. Seleng, M. Laclavík, Z. Balogh, Z. Hluchý.** RDB2Onto: Approach for creating semantic metadata from relational database data. In *INFORMATICS’ 2007: proceedings of the ninth international conference on informatic, Bratislava Slovak Society for Applied Cybernetics and Informatics*, 2007, 113–116.
- [27] **K. Czarnecki, S. Helsen.** Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3), 2006, 621–645.
- [28] **A. Armonas, L. Nemuraitė.** Using Attributes and Merging Algorithms for Transforming OCL Expressions to Code. *Information Technology and Control*, 2009, 38(4), 283–293.
- [29] **L. Ablonskis, L. Nemuraitė.** Discovery of Complex Model Implementation Patterns in Source Code. *Information Technology and Control, Kaunas, Technologia*, 2010, 39(4), 291–300.
- [30] **V. Štuikys, R. Damaševičius.** Design of Ontology-Based Generative Components Using Enriched Feature Diagrams and Meta-Programming. *Information Technology and Control*, 2008, 37(4), 301–310.
- [31] **Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.** ATL: a QVT-like transformation language. In *Proc. OOPSLA Companion*, 2006, 719–720.
- [32] **OMG**, 2011. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. *OMG Document Number: formal/2011-01-01*, 2011.
- [33] **D. Song, K. He, P. Liang, W. Liu.** A formal language for model transformation specification. Available from: <http://www.cs.rug.nl/search/uploads/Publications/song2005flm.pdf> [Accessed 10 Jan 2011].
- [34] **S.H. Tirmizi, J. Sequeda, D. Miranker.** Translating SQL Applications to the Semantic Web. In *Proceedings of the 19th international conference on Database and Expert Systems Applications*, LNCS 5181, 2008, 450–464.
- [35] **H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, G. Taentzer.** Information preserving bidirectional model transformations. In *Proceedings of Fundamental Approaches to Software Engineering (FASE 2007)*, LNCS 4422, 2007, Springer, Heidelberg, 72–86.
- [36] **P. Stevens.** Bidirectional model transformations in QVT: semantic issues and open questions, *Software and Systems Modeling*, 9, 2010, 7–20.
- [37] **J.L. Hainaut, C. Tonneau, M. Joris, M. Chadelon.** Transformation-based Database Reverse Engineering. In *Proceedings of the 12th International Conference on the Entity-Relationship Approach*, LNCS, 823, Springer-Verlag, 1993, 364–375.

- [38] **B. Paradauskas, A. Laurikaitis.** Extracting conceptual data specifications from legacy information systems. *Electronics and Electrical Engineering*, 2011, 1(107), 46–50.
- [39] **OMG, 2006.** Common Warehouse Metamodel Specification. *Object Management Group, OMG Document Number: pas/06-04-02*, 2006.
- [40] **OMG, 2009.** Ontology Definition Metamodel. Version 1.0. *OMG Document Number: formal/2009-05-01*, 2009.

Received March 2011.