

Mapping Process of Relational Schema to OWL Ontologies using XSLT

Abderrahim Marzouk

Département de Mathématiques et Informatique
Laboratoire IR2M, Université Hassan 1er, FST Settat, Maroc

ABSTRACT

One of the challenges of the semantic Web is to integrate the information already available on the standard Web, usually stored in relational databases. We propose in this paper a transformation process from existing relational database schema defined by a set of tables into OWL ontologies using XSLT. This process allows us to mapping some constraints on columns such as primary key, foreign key, unique, not null. Our approach start by translate the database schema into a XML document validate by a suitable XML schema.

Since our process is based on XSLT stylesheets, its transformation rules can be modified in a very flexible manner in order to consider different mapping strategies and requirements.

Keywords

Relational databases schema, XML, XSD, XSLT, Ontology, OWL.

1. INTRODUCTION

Ontology has been used widely in the knowledge engineering, artificial intelligence, information retrieval, heterogeneous information process, semantic web and other related fields. Ontology should be designed and constructed elaborately with the importance of ontology resources. It has become an important task to improve the efficiency of ontology construction. However, manual ontology construction is a complex and mistakable work and causes that the period of ontology building is long and the cost is high[1,4,6,9,10,11]. It is a meaningful and challenging task to build ontology from existing relational database to improve the efficiency of ontology construction and the information contains in the tables can be shared and linked with other information.

This paper proposes a set of rules XSLT transforming an existing relational database schema to OWL ontologies.

The remainder of this paper is organized as follows: Section 2 describes our mapping process. Section 3 presents mappings between relational and ontological concepts. The process for mapping the given relational schema into OWL is explained in Section 4. In Section 5, we give a mapping example. Section 6 concludes the paper and considers the future work.

2. DESCRIPTION OF MAPPING PROCESS

In this section we present some database concepts and we propose an XML schema which includes elements corresponds to tables, columns, data types, primary keys, foreign keys and other constraints.

Our mapping process is based on this XML schema. A relational schema is represented by a XML document

according this XML schema. The elements in this document will be transformed in OWL using XSLT.

2.1. Relational Schema

The relational schema is given by a finite set of table's schemas and set of integrity constraints. Tables are represented by their name, and columns

The primary keys gets marked on the table by underlining and foreign keys are marked by “*”.

Example.

Employee(employeeID:int, name:string not null,age:int)

Project(projectID:int, managerID*:int, dueDate:date, budget:int not null)

Participation(employeeID*:int, projectID*:int)

Fig 1: Relational schema of database projects

2.2. Classification of Tables

Each table in relational schema is classified into one of the three categories: entity, relation and composite tables [4, 5]. An entity table is a table which represents entity in ER diagram; a relation table is a table represents a binary relationship many-to-many between two entities in ER diagram without additional columns attached to the relationship; a composite table is a table represents a binary relationship many-to-many between two entities in ER diagram with additional columns attached to the relationship.

That is, an entity table can include the primary key of another table.

Example 1. A table Employee(employeeID, name) is an entity table.

Example 2. A table Project(projectID, managerID*) is an entity table.

Example 3. A table Participation(employeeID*, projectID*) is a relation table, because its primary key is composed of the primary keys of two other tables: Employee and Project. That table consists of the primary keys of the two tables

Example 4. A table Participation(employeeID*, projectID*,hours) is a composite table since it contains an additional column hours.

2.3. Relational Schema to XML Schema

Formally a relational schema can be represented as a file XML validated by the follows XML schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd
="http://www.w3.org/2001/XMLSchema">
<xsd:element name="relational-schema">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="table" minOccurs="0"
maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="column" minOccurs="0"
maxOccurs="unbounded">
<xsd:complexType>
<xsd:attribute name="name" type="xsd:string"/>
<xsd:attribute name="type" type="xsd:string"/>
<xsd:attribute name="isUnique" type="xsd:boolean"
use="optional"/>
<xsd:attribute name="isNullable" type="xsd:boolean"
use="optional"/>
</xsd:complexType>
</xsd:element >
<xsd:element name="primaryKey">
<xsd:complexType>
<xsd:attribute name="name" type="xsd:string"/>
<xsd:attribute name="type" type="xsd:string"/>
</xsd:complexType>
</xsd:element >
<xsd:element name="foreignKey" minOccurs="0"
maxOccurs="unbounded">
<xsd:complexType>
<xsd:attribute name="name" type="xsd:string"
use="optional"/>
<xsd:attribute name="foreignTable" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string"/>
<xsd:attribute name="type">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="entity"/>
<xsd:enumeration value="relation"/>
<xsd:enumeration value="composed"/>
</xsd:restriction>
</xsd:simpleType>
```

```
</xsd:attribute>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Fig 2: Schema.xsd

3. APPROACH FOR MAPPING RELATIONAL SCHEMAS TO OWL

This section gives steps for transforming relational databases schemas (tables, columns, constraints) to OWL ontologies (class, property, relation). There are three steps of our approach: (1) mapping tables, (2) mapping columns and (3) mapping constraints.

3.1. Mapping Tables

The first step of our approach is mapping tables. It's consists of the following rules:

Rule 1: Each entity table maps to an OWL class with the same name.

Rule 2: Each composite table maps to an OWL class with the same name.

Rule 3: Each relation table is not mapped to an OWL class, but it maps to a pair of object properties op1 and op2. Each of these two properties op1, op2 is inverse to the other.

Example 1.

Employee(employeeID,name)→<owl:Class
rdf:ID="Employee"/> and Project(projectID,
managerID*)→<owl:Class rdf:ID="Project"/>

Fig 3 : Mapping table by rule 1

The table Employee and Project are maps to an OWL class, because there are entity tables.

Example 2. Consider a relation table Participation (employeeID*,projectID*). This table represents a binary relationship between two entities in the ER diagram: Employee and Project. Since the table entirely consists of the primary keys of the two tables, it maps to a pair of object properties: Project-Employee and Employee-Project. One object property is the inverse of another, meaning that the relationship is bidirectional; i.e. a project involves an employee and an employee is involved in a project.

```
<owl:ObjectProperty rdf:ID="Project-Employee">
<rdfs:domain rdf:resource="#Project"/>
<rdfs:range rdf:resource="#Employee"/>
</owl:ObjectProperty>
and
<owl:ObjectProperty rdf:ID="Employee-Project">
<owl:inverseOf rdf:resource="#Project-Employee"/>
rdfs:domain rdf:resource="#Employee"/>
```

```
<rdfs:range rdf:resource="#Project"/>
</owl:ObjectProperty>
```

Fig 4 : Mapping table by rule 2

Example 3. Consider a composite table Participation (employeeID*, projectID*, hours). Therefore, it maps to an OWL class and a pair of object properties for each table participating in the relationship (The object properties are not shown in Fig. 5).

```
Participation(employeeID*, projectID*, hours) → <owl:Class
rdf:ID="Participation"/>
```

Fig 5: Mapping table by rule 3

3.2. Mapping Columns

The second step of our approach is mapping columns. For all tables, we map their columns that are not foreign keys to data type properties. The range of a data type property is the XML schema data type equivalent to the data type of its original column. Foreign keys are ignored for a while, as they represent relationships.

Example. Consider a column hours in table Participation in Figure 1. This column maps to a data type property hours.

```
<owl:DatatypeProperty rdf:ID="hours">
<rdfs:domain rdf:resource="#Participation"/>
<rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

Fig 6: Mapping columns

3.3. Mapping Constraints

The third step of our approach is mapping constraints. Constraints specify if a column in a table is unique or not null, or if the column is a primary key or a foreign key.

3.3.1. Mapping unique constraints

This constraint specifies that a column in a table is unique, meaning that no two data in the table have the same value for the column. Therefore, the constraint maps to a functional property.

Example.

```
Project(projectID, budget unique) → <owl:FunctionalProperty
rdf:ID="budget"/>
```

Fig 7: Mapping unique constraints

3.3.2. Mapping not null constraints

This constraint specifies that a column in a table is not null, meaning that all data in the table contains values for the column. Therefore, the constraint maps to a minimum OWL cardinality of 1.

Example.

```
Project(projectID, budget not null) →
<owl:Class rdf:ID="Project">
<rdfs:subClassOf>
```

```
<owl:Restriction>
<owl:onProperty rdf:resource="#budget"/>
<owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">
1 </owl:minCardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Fig 8: Mapping not null constraints

3.3.3. Mapping primary key constraints

Since a value of primary key can uniquely determines a single row of table, the primary key constraint maps to OWL InverseFunctional properties.

Example. Consider a primary key projectID in table Project (projectID, managerID*). This column maps to an OWL inverse functional property with same name.

```
<owl:InverseFunctionalProperty rdf:ID="projectID">
<rdfs:domain rdf:resource="#Project"/>
<rdfs:range rdf:resource="&xsd;integer"/>
</owl:InverseFunctionalProperty>
```

Fig 9: Mapping primary key constraints

3.3.4. Mapping foreign key constraints

A foreign key of a table T_1 establishes a relationship between T_1 and another table T_2 . Therefore, it's mapped to an object property named T_1 - T_2 with the OWL class corresponding to T_1 as domain class and the OWL class corresponding to T_2 as range class[2,3].

Example. Consider a foreign key managerID in table Project (projectID, managerID*). This column maps to an object type property Project-Employee.

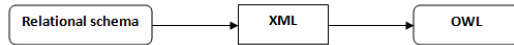
```
<owl:ObjectProperty rdf:ID="Project-Employee">
<rdfs:domain rdf:resource="#Project"/>
<rdfs:range rdf:resource="#Employee"/>
</owl:ObjectProperty>
```

Fig 10: Mapping foreign key constraints

4. MAPPING SCENARIO

In this section, we present a proposal for a mapping process from relational schema as XML document to OWL which raises the XML source documents to OWL ontology. The XML document contains a description of relational schema.

Knowing that XSLT (XML StyleSheet Language Transformation) is a W3C standard widely used, that our source document is in XML and our destination language is OWL. The transformation relational schema to OWL is implemented as a set of XSLT stylesheets that take as input an XML description of relational schema.

**Fig 11 : Mapping diagram**

The following file called *transformation.xsl* contains a set of rules XSLT transforming a XML description of existing relational schema to OWL ontologies.

```

<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
version="1.0">

<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">

<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<xsl:apply-templates select="//table"/>
<xsl:apply-templates select="//column"/>
<xsl:apply-templates select="//primaryKey"/>
<xsl:apply-templates select="//foreignKey"/>
</rdf:RDF>

</xsl:template>

<xsl:template match="//table">
<xsl:choose>
<xsl:when test="(./@type='entity' or ./@type='composed')">
<xsl:element name="owl:Class">
<xsl:attribute name="rdf:ID">
<xsl:value-of select="@name"/>
</xsl:attribute>
<xsl:if test="column/@isNullable='false'">
<rdfs:subClassOf>
<owl:Restriction>
<xsl:element name="owl:onProperty">
<xsl:attribute name="rdf:resource">
<xsl:value-of select="concat('#',./column/@name)"/>
</xsl:attribute>
</xsl:element>
<owl:minCardinality
rdf:datatype="nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction>
</rdfs:subClassOf>
</xsl:if>
</xsl:element>

```

```

</xsl:when>
<xsl:when test="./@type='relation'">
<xsl:element name="owl:ObjectProperty">
<xsl:attribute name="rdf:ID">
<xsl:value-of select="concat(foreignKey[1]/@foreignTable,'-','foreignKey[2]/@foreignTable)"/>
</xsl:attribute>
<xsl:element name="rdfs:domain">
<xsl:attribute name="rdf:resource">
<xsl:value-of
select="concat('#',foreignKey[1]/@foreignTable)"/>
</xsl:attribute>
</xsl:element>
<xsl:element name="rdfs:range">
<xsl:attribute name="rdf:resource">
<xsl:value-of
select="concat('#',foreignKey[2]/@foreignTable)"/>
</xsl:attribute>
</xsl:element>
</xsl:element>
<xsl:element name="owl:ObjectProperty">
<xsl:attribute name="rdf:ID">
<xsl:value-of select="concat(foreignKey[2]/@foreignTable,'-','foreignKey[1]/@foreignTable)"/>
</xsl:attribute>
<owl:inverseOf>
<xsl:attribute name="rdf:resource">
<xsl:value-of select="concat(foreignKey[1]/@foreignTable,'-','foreignKey[2]/@foreignTable)"/>
</xsl:attribute>
</owl:inverseOf>
<xsl:element name="rdfs:domain">
<xsl:attribute name="rdf:resource">
<xsl:value-of
select="concat('#',foreignKey[2]/@foreignTable)"/>
</xsl:attribute>
</xsl:element>
<xsl:element name="rdfs:range">
<xsl:attribute name="rdf:resource">
<xsl:value-of
select="concat('#',foreignKey[1]/@foreignTable)"/>
</xsl:attribute>
</xsl:element>
</xsl:element>
</xsl:when>

```

```

</xsl:choose>
</xsl:template>
<xsl:template match="//table[@type='entity' or
@type='composed']/column">
<xsl:element name="owl:DatatypeProperty">
<xsl:attribute name="rdf:ID">
<xsl:value-of select="@name"/>
</xsl:attribute>
<xsl:element name="rdfs:domain">
<xsl:attribute name="rdf:resource">
<xsl:value-of select="concat('#',../@name)"/>
</xsl:attribute>
</xsl:element>
<xsl:element name="rdfs:range">
<xsl:attribute name="rdf:resource">
<xsl:value-of select="concat('#',@type)"/>
</xsl:attribute>
</xsl:element>
</xsl:element>
</xsl:template>
<xsl:template match="//table/primaryKey">
<xsl:element name="owl:InverseFunctionalProperty">
<xsl:attribute name="rdf:ID">
<xsl:value-of select="@name"/>
</xsl:attribute>
<xsl:element name="rdfs:domain">
<xsl:attribute name="rdf:resource">
<xsl:value-of select="concat('#',../@name)"/>
</xsl:attribute>
</xsl:element>
<xsl:element name="rdfs:range">
<xsl:attribute name="rdf:resource">
<xsl:value-of select="concat('#',@type)"/>
</xsl:attribute>
</xsl:element>
</xsl:template>
<xsl:template match="//table[@type='entity' or
@type='composed']/foreignKey">
<xsl:element name="owl:ObjectProperty">
<xsl:attribute name="rdf:ID">
<xsl:value-of select="concat('../@name','-','@foreignTable)"/>
</xsl:attribute>

```

```

<xsl:element name="rdfs:domain">
<xsl:attribute name="rdf:resource">
<xsl:value-of select="concat('#',../@name)"/>
</xsl:attribute>
</xsl:element>
<xsl:element name="rdfs:range">
<xsl:attribute name="rdf:resource">
<xsl:value-of select="concat('#',@foreignTable)"/>
</xsl:attribute>
</xsl:element>
</xsl:element>
</xsl:template>
<xsl:template match="//table/column">
<xsl:if test="@isUnique='true'">
<xsl:element name="owl:FunctionalProperty">
<xsl:attribute name="rdf:ID">
<xsl:value-of select="@name"/>
</xsl:attribute>
</xsl:element>
</xsl:if>
</xsl:template>
</xsl:stylesheet>

```

Fig 12: transformation.xsl

5. A MAPPING EXAMPLE

We give in this section an example of mapping a relational schema into OWL. Consider the relational schema in figure 1. The XML description of this relational schema is stored in a following text file called *schema.xml*. This file is the input of our transformation process.

```

<relationnal-schema name="projects">
<table name="Employee" type="entity">
<column name="name" type="string" isNullable="false"/>
<primaryKey name="employeeID" type="int"/>
</table>
<table name="Project" type="entity">
<column name="dueDate" type="date"/>
<column name="budget" type="int" isNullable="false"/>
<primaryKey name="projectID" type="int"/>
<foreignKey type="int" foreignTable="Employee"/>
</table>
<table name="Participation" type="relation">
<foreignKey type="int" foreignTable="Employee"/>
<foreignKey type="int" foreignTable="Project"/>
</table>

```

</relational-schema>

Fig 13. schema.xml

We use the following php script called transformation.php to transform the file *schema.xml* to an OWL file with the XSLT StyleSheet.

```
<?php
$doc = new DOMDocument();
$doc->load("schema.xml");
$xml = new XSLTProcessor();
$xmlsl = new DOMDocument();
$xmlsl->load("transformation.xml");
$xml->importStyleSheet($xmlsl);
echo $xml->transformToXML($doc);
?>
```

Fig 14. Script transformation.php to perform XSL transformation

When we apply this previous XSL Stylesheet to the XML file, the following OWL output is obtained:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="&owl;" xmlns:rdf="&rdf;"
xmlns:rdfs="&rdfs;" xmlns:xsd="&xsd;">
  <owl:Class rdf:ID="Employee">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#name"/>
        <owl:minCardinality
rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Project">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#budget"/>
        <owl:minCardinality
rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="Employee-Project">
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:range rdf:resource="#Project"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Project-Employee">
```

```
<owl:inverseOf rdf:resource="Employee-Project"/>
  <rdfs:domain rdf:resource="#Project"/>
  <rdfs:range rdf:resource="#Employee"/>
</owl:ObjectProperty>
<owl:InverseFunctionalProperty rdf:ID="employeeID">
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:range rdf:resource="#int"/>
</owl:InverseFunctionalProperty>
<owl:InverseFunctionalProperty rdf:ID="projectId">
  <rdfs:domain rdf:resource="#Project"/>
  <rdfs:range rdf:resource="#int"/>
</owl:InverseFunctionalProperty>
<owl:ObjectProperty rdf:ID="Project-Employee">
  <rdfs:domain rdf:resource="#Project"/>
  <rdfs:range rdf:resource="#Employee"/>
</owl:ObjectProperty>
</rdf:RDF>
```

Fig 15. OWL output

6. CONCLUSION AND FUTURE WORK

This paper has presented a mapping process which makes a transformation from relational schemas to OWL ontology. This transformation preserves semantics of some constraints of the table's columns in order to obtain a usable ontology for needs in shared information. For the realization of this transformation we developed a suitable XML schema for relational schemas. Our process is implemented running the XSLT transformation over the XML instances document, produces an OWL output document containing the OWL classes and their properties. Our future work will transform the relational data into to ontology instances.

7. REFERENCES:

- [1] J. Fon g, F. Pang, C. Bloor "Converting Relational Database into XML Document". DEXA Workshop, pp 61-65. 2001
- [2] N. GHERABI, K. ADDAKIRI, M. BAHAI "Mapping relational database into OWL Structure with data semantic preservation". CoRR abs/1205.5922. 2012
- [3] M. Arnoux, T. Despeyroux " Multi-représentation d'une ontologie : OWL, bases de données, systèmes de types et d'objets". CoRR abs/1104.2982. 2011
- [4] L. Zhang et al "Automatic Generation of Ontology Based on Database". Journal of Computational Information Systems 7:4 1148-1154. 2011
- [5] IRINA ASTROVA, AHTO KALJA "Mapping of SQL Relational Schemata to OWL Ontologies" Proceedings of the 6th WSEAS International Conference on Applied Informatics and Communications, Elounda, Greece, August 18-20, 2006 (pp375-380)
- [6] Nasser Alalwan, Hussein Zedan, François Siewe "Generating OWL Ontology for Database Integration"

- 2009 Third International Conference on Advances in Semantic Processin.
- [7] Jamal Bakkas, Mohamed Bahaj, Abderrahim Marzouk Direct Migration Method of RDB to Ontology while Keeping Semantics International Journal of Computer Applications (0975 – 8887) Volume 65– No.3 March 2013
- [8] Pierre-Antoine Champin, Greet-Jan Hourben, and Philippe Thiran “Cross: An OWL Wrapper for Reasoning on Relational Databases “ C. Parent et al (Eds.): ER 2007, LNCS 4801, PP. 502-517, 2007. Springer-Verlag Berlin Heidelberg 2007
- [9] Juan F. Sequeda, Marcelo Arenas, Daniel P “ Miranker On Directly Relational Databases to RDF and OWL” www 2012, April 16-20, 2012, Lyon, France. ACM 978-1-4503-1229-5/12/04.
- [10] A. Bertails, and E Prud’hommeaux “Interpreting relational databases in the RDF domain” K-CAP, pages 129-136, 2011
- [11] Nasser Alalwan, Hussein Zedan, François Siewe “Generating OWL Ontology for Database Integration Third International” Conference on Advances in Semantic Processing 2009.