Journal of Theoretical and Applied Information Technology
10th October 2014. Vol. 68 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

UPGRADING THE SEMANTICS OF THE RELATIONAL MODEL FOR RICH OWL 2 ONTOLOGY LEARNING

[1]BOUCHRA EL IDRISSI, [2]SALAH BAINA, [3]KARIM BAINA

[1]PhD Student, ENSIAS, University Mohammed V Rabat, BP 713, Rabat Morocco
[2]Professor, ENSIAS, University Mohammed V Rabat, BP 713, Rabat Morocco
[3]Professor, ENSIAS, University Mohammed V Rabat, BP 713, Rabat Morocco
E-mail: [1]bouchra.idrissi@um5s.net.ma, [2]sbaina@ensias.ma, [3]baina@ensias.ma

ABSTRACT

This paper is interested in the ontology learning from relational databases (RDB) that exploits already approved semantics about a domain and translates them to application ontology. However, the relational model is recognized to be less expressive and incapable to support some conceptualizations. Without an explicit model of the domain semantics in the relational model, the automatic learning of ontology risks to infer incorrect semantics. In this paper, we give some proof case studies and we propose a model to upgrade the semantics of the relational model, before the ontology learning. The paper presents the constructs of the proposed model and it shows how they are translated to constructs of OWL 2 ontology.

Keywords: *Ontology Learning, Relational Database, Semantic Enrichment, OWL 2.*

1. INTRODUCTION

In the domain of semantic interoperability of enterprise applications (EA), one aspect of the role of ontology (formal, explicit specifications of shared conceptualizations [1]) is to provide a precise meaning of information exchanged among interoperated (EA). Unfortunately, ontology has been omitted from the development life cycle of information systems and many enterprise applications have no ontology that describes their business information. Moreover, the manual development of ontology is hard, time-consuming and error-prone.

Ontology learning field aims to overcome the ontology acquisition bottleneck by automatically or semi-automatically generating ontology from some input information sources of types structured, semi-structured or unstructured [2]. The present paper focuses on the ontology learning from RDBs. It is an attractive field for developing application ontologies, on one hand due to the pervasiveness of the relational model in industry, commercial and open-source applications and on the other hand because an RDB already incorporates an approved semantics of the domain.

Application ontology is an ontology that describes a domain application (the specific-domain knowledge modeled in the database (for more information about the classification of ontologies, please see Watch et al. [3] where four types of ontologies are defined: top-level, domain, task and application). For clarity, we borrowed the definition of the term '*domain application semantics'* from [4]: '*information about the application domain, which should be captured during the requirements specification phase of database design*'.

Several approaches have been proposed for ontology learning from RDB. Some approaches rely only on the RDB to construct ontology and the result is an ontology that mirrors the relational model. Other approaches proposed to enrich the resulting ontology by investigating additional resources like domain ontologies; lexical vocabularies and conceptual models of the RDB (see section 3 for more details). Our methodology for ontology learning from RDB presented in [5] is based on a primordial process of the semantic enrichment of the relational model. In fact, according to Lezoche et al. [6], '*the main prerequisite for achievement of interoperability of information systems is to maximize the amount of semantics which can be used and make it increasingly explicit* [7], *and consequently, to make the systems semantically interoperable'*.

This paper aims to present the semantics we add to the relational database model and to present the model that we propose for its remodeling with more explicit semantics. At present, the upgrade of

Journal of Theoretical and Applied Information Technology
10th October 2014. Vol. 68 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the model semantics is manual, but (semi-) automatic methods (such as data mining) will be exploited to discover '*candidate semantics*' and to enrich automatically the model. In this way, the database analyzers and/or domain experts are able to validate all the integrated semantics (tacit, enriched and embedded) in the proposed model, before the ontology learning.

The paper is structured as follows. We introduce and we motivate in the next section, the semantics by which we enrich the database model. These semantics are illustrated by examples. Section 3 gives an overview of related works. We focus on how existing approaches have addressed the requested semantics. Section 4 describes the proposed model and it outlines its constructs. Section 5 presents the implementation. Section 6 assesses the transformation of considered semantics into ontology constructs. A conclusion, in section 7, summarizes the whole paper and considers future work.

.

2. REQUESTED SEMANTICS

This section describes the semantics by which we upgraded the RDB model. We motivate each semantic and we give some examples for illustration.

2.1 The Meaning of Relationships

The relational model does not store the meaning of relationships between relations (tables). It only indicates that there is a link between them through the use of foreign-keys. Our objective is to allow the designation of each relationship (link between two relations) by a term that gives a meaning to it. While this seems to be unimportant for machine processing, it is, however, indispensable for the ontology readability and its validation. We give particular attention to three kinds of relationships: multiple relationships, recursive relationship and inclusion relationship.

Multiple Relationships are different binary relationships defined between two separate relations. An example is the relationships between *m_product* and *m_product_bom* shown in Fig.1 (from the database of OpenBravo [8]). Two foreign-keys from *m_product_bom* reference the table *m_product*: one of them identifies the BOM (Bill of Material) product and the other designates a product component that is part of the BOM product.

According to [8], the Bill of Materials defines those products that are generated from other products. A Bill of Material (BOM) is one or more products or BOMs. The table *m_product_bom* defines the BOM product and its product parts with the associated quantities and information.

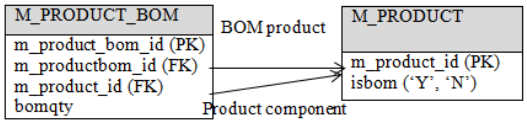


Figure 1: Example of Multiple Relationships.

So, in terms of relationships, there are two kinds: a *hierarchy* (a BOM is a product) and a *whole-part* relationship (a BOM is an aggregation of other products) (we use here the term of aggregation as defined in UML [9] instead of composition since the existence of product parts does not depend on the BOM).

Recursive relationship is a relationship where the source and the target are the same table. Fig. 2A presents an example of a recursive relationship (from the database of OpenERP [10]). This recursive relationship indicates a hierarchy of categories (*hasParent*). In other cases of recursive relationships, driving the sense of them may be less obvious. Example is the case of a table employee with two relationships, one indicating a *hasSpouse* relationship and the other a *hasSuperior* relationship (Fig. 2B).

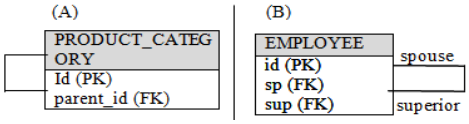
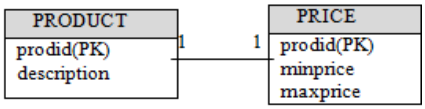


Figure 2: Examples of Recursive Relationships.

Inclusion relationships. According to Chiang et al. **[4]**, if two entity types (relations) *A* and *B* have, not only the same key *X*, but also the same set of data instances in their keys (there are two inclusion dependencies, *A.X*«*B.X* and *B.X*« *A.X*). Then, the user must specify the proper type of *inclusion* relationship between them, such as *A is-a B, A is-a-kind-of B, A is-part-of B, A has* B, etc (see example in Fig. 3). In operational databases, the primary key may be named arbitrarily or contains the name of the table. Moreover, the values could be generated automatically according to a sequence. All these factors complicate the detection of such relationship.



ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 3: An Example of Inclusion Relationship of Type 'Has': A product has a price (from [4]).

Generally, in the literature a one-to-one relationship is automatically transformed into *is-a* (hierarchy relation with the construct *subClassOf* of OWL [14]) (see [11, 12, 13]).

2.2 Domain Names of Relations and Attributes

The names of database tables and columns are not always significant. They can be abbreviated or use ambiguous terms without a clear meaning. *SAP [15]* uses abbreviated names for their database columns and tables. For instance, the table *VBPA* designates partners in sales order and all its columns are codified (Fig. 4).

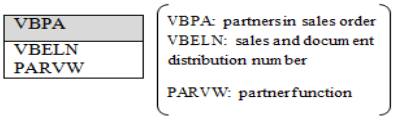


Figure 4: A Subset of the Table VBPA from SAP [15].

2.3 Concepts Aggregated in One Relation

We mean by an aggregated relation, a relation that embodies more than once concept. In the flag approach [16] for the representation of subtype hierarchies, one single relation represents the whole hierarchy. This approach uses a flag (called also discriminator) attribute to determine a specific subtype. While data mining techniques and the estimation of data redundancy may help in detecting the flag attributes, the discovery of aggregated concepts necessitates the implication of a database analyzer. An example is the table *c_bpartner* in Fig. 5, which groups customers, vendors and employees. The flag attributes are *isvendor*, *iscustomer* and *isemployee* (this example had significant names of flag attributes, but it is not generally the case). Another example is the table *c_order* that concerns purchase and sale orders.

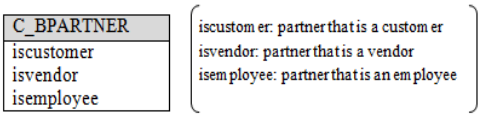


Figure 5: An Example of Aggregated Relations in the Table C_BPARTNER of OpenBravo.

In the ontology, the aggregated relations should be made explicit.

2.4 Enumerated Relation and its Domain Values

An enumerated relation is equivalent to enumeration class in UML [9]. In most cases, this type of relation has no foreign-key and store one business information, but it is difficult to generalize this rule. In existing approaches for ontology learning, relations that implement an enumeration are classified as *strong* (regular) relations. An example of relation that stores such information is the table *product_uom_categ* shown in Fig. 6. This table contains the list of UOM (Unit of Measure) categories. It contains also some audit information and the column *name* represents the UOM categories. This type of relation raises two challenges: How to detect them and how to find the domain values equivalent to those stored and codified.

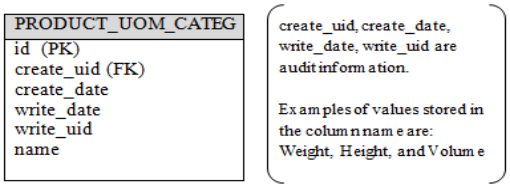


Figure 6: An Example of Relation of Type Enumeration (from OpenERP Database).

2.5 Domain Values of Attributes that Denote an Enumeration

The objective in this case concerns the association of explicit domain values to correspondent attribute values in the database. An example is the *producttype* column in the table *m_product* that identifies the type of the product from a predefined list: item, service, resource, expense and online. But, the values stored in the column are codified as (I, S, R, E, O) (see Fig.7). Other codified domain values can be found in *docStatus* column of the table c_*order* (of OpenERP). It designates the possible status of an order.

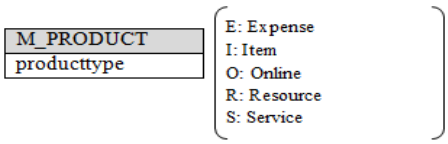


Figure 7: An Example of an Attribute with Codified Values and Their corresponding Meaning (from OpenBravo Database).

We can distinguish different roles for this attribute type. Either it is a *categorizing* attribute that designates different sub-concepts of the concept modeled through the relation (like the example giving in [17]); a *selective* attribute, where their distinct values present a collection of multiple choices (Fig.7); a *flag* attribute with codified values and where each one indicates an aggregated concept (Fig.5 where *iscustomer* has two possible

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

values 'y' for 'yes' and 'n' for 'no'). So the question is how to discover this distinction?

2.6 Composite Attribute

Fig. 8 shows two cases for relating a composite attribute (*address*) to the master relation (*employee*). On the right side (Fig. 8B), the simple attributes of *address* are implemented as attributes of the master relation, and on the left side (Fig. 8A) a relation is created to model the composite attribute. Fahad in [18] had discussed how to map a composite attribute from en ER model to OWL data

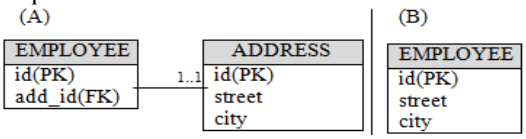


Figure 8 : Example of Composite Attribute as an Externalized Table (A) or Presented as Simple Attributes of a Table (B)

type properties. In the ER model the information about composite attributes is explicit, but from a database model and unless it is indicated by database analyzers, it is impossible to discover automatically such semantics.

2.7 Multi-Valued Attribute

A multi-valued attribute refers to an attribute that can have more than one value per instance. There is no standardizing way for modeling a multi-valued attribute in RDB. Fig.9 presents the three basic implementations: (A) a separate table is created for the multi-valued attribute (*hobbies)*; (B) values of a multi-valued attribute are combined in a single column and separated by a character (e.g., comma); (C) a row is created for each value of the multi-valued attribute and this attribute is part of the primary-key of the relation.

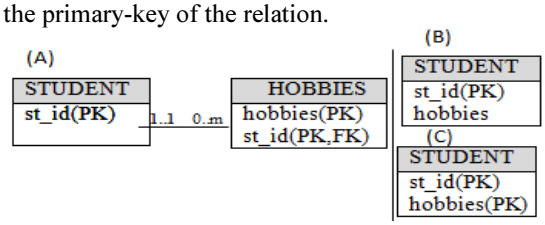


Figure 9 : RDB Basic Implementations of Multi-Valued Attribute.

Although (A) is the recommended implementation, this does not exclude the possibility of implementation of (B) and (C). Hwang et al. in [19] stated that the case (A) may result in some efficiencies and it is suitable only when the multi-valued attribute is an entity (when talking at the conceptual level (ER diagram), an entity is a relation in the RDB) and when it has many-to-many relationship with the other entity. Otherwise, alternative possibilities should be adopted.

3. RELATED WORKS

We have studied and compared in our previous works [20, 21], some approaches for ontology learning from RDB. Our objective in this section is to discuss how the existing approaches addressed the requested semantics presented in the previous section. Recent approaches are also considered in this study.

Few approaches have been interested in discovering implicit and hidden semantics of an application domain. In [22] some absent semantics from the input model expressed in SQL are transferred manually from the entity relationship model (ER model) to the produced ontology. Examples of these semantics are composite attributes and cardinalities. Astrova in [23, 24] proposed a reverse engineering process that used a database-driven HTML forms. The objective was to overcome the limits of relational schema like bad database design, meaningless names, de-normalization and so on. A recent work of Ramathilagam and Valarmathi [25] proposed an approach where a direct mapping of RDB components to ontology constructs is followed by the addition of some semantic rules explored from the ER model. Alalwan et al. [26] analyze both database instances and schema to drive some uncommon semantics: *fragmentation* of tables, multi-valued attribute (case (A) in 2.8), and hierarchy. Albarak and Sibley in [27] proposed a method to detect *sparse-columns (*columns that have sparse values*)* by computing the number of distinct values and checking that this number' distribution ratio is within user-specified thresholds. They used the distinct values stored in the *sparse-column* as a restriction on the range of its equivalent data property. Their approach is interesting; however, it does not allow distinguishing whether these distinct values designate the values of a *selective*, *categorizing* or *flag* attribute.

The use of OWL 1 [14] as the language of the targeted ontology has resulted in additional inconsistencies. An example is the use of *owl*:*InverseFunctionalProperty* in defining that a data property presents a primary-key of a relation (e.g., in [24]). According to [28], OWL 1 does not allow data properties (that relate individuals to literals) to be directly declared as inverse functional properties. *InverseFunctional* characteristic applies to object properties (that relate individuals to individuals).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

As we can see, many of our requested semantics presented in the previous section have not yet been addressed by existing approaches (e.g., semantics of relationships, domain values, aggregated concepts, and so on). The addressed semantics are obtained either manually or through the investigation of additional resources (like data instances, HTML forms, ER model). However, contrary to the knowledge of database analyzers, the automatic discovered semantics present '*candidate semantics*' that may be out-of-date or incorrect, by cause of many factors such as *dirty* data.

4. PROPOSED MODEL

4.1 The Choice of XML

The semantic enrichment is generally a process that upgrades the semantics of a model. For some authors, semantic enrichment is generally considered a synonym of annotating source data with formal descriptions of concepts in a domain ontology [29], [30]. The main aim of these annotations is to make explicit the meaning and the structure of the models to enable their understanding, their exchange and their possible transformation between collaborating actors (human or machine) [30]. For others, especially in the context of database schema enrichment, it is considered a process for remodeling database schema in a higher data model, in order to explicitly express semantics that is implicit or hidden [16, 31, 32].

The choice of XML schema was not arbitrary. The main reasons that have motivated us to opt for XML schema are its extensibility and the existence of open source libraries that map a RDB schema to an XML schema.

4.2 The Model Constructs

Table 1 describes the model constructs, in terms of classes and attributes. We noted that the classes: *Database*, *Table*, and *Reference* are part of the model proposed by *DdlUtils* [33]. The other classes represent our extension. Due to the constraint of space, the attributes that are defined by *DdlUtils* are not presented. The class *Foreign-Key* is also absent from the table because no additional attribute has been defined for it.

Fig. 10 presents the model as a schema XML in order to show the relations between its elements. While case studies in section 6 help in understanding the purpose of the model elements.

5. IMPLEMENTATION

Our approach relies on the input database and therefore requires an easy to use system that allows the exploration and the management of the database. For this objective, we have chosen *SQuirrel* [34] and we have extended it with plugins that fit our need.

SQuirreL SQL Client is a graphical tool built using Java and therefore it allows viewing the structure of any JDBC compliant database. The advantages of SQuirrel are multiples, including access to several databases installed locally or on remote machines, through a single graphical user interface; the exploration of the database objects and metadata; the edition of database data; and its functionality (DB-specific or general) can be extended through the use of plugins.

For the objective of the paper, we have implemented a new plugin, called *DbToXML*. It is built upon the Apache library *DdlUtils*. Through the *DbToXML* preferences tabular, the user may configure *DbToXML* to generate one XML file regrouping the translation of all the tables or to get one XML file per table. *DdlUtils* uses the Apache betwixt library [35] that maps beans to XML. At the core of *DdlUtils*, there are the classes defining the database schema. To get a description of these database objects, the system uses the *DatabaseMetaData* Java class.

6. THE TRANSFORMATION TO OWL 2

The objective of this section is not to exhibit the formal rules necessary for the transformation of the database schema to OWL 2 [36] ontology. It is out the scope of this paper, but we show the practicability of our contribution. For each case of the section 2, we showed the result of the XML transformation and how it is semantically enriched. Then, we outlined the expected result for its transformation to OWL 2 ontology. For more readability, we write the OWL 2 ontologies in a compact form using the functional syntax. For the reason of the lack of space, we do not present the ontologies resulting from direct mapping.

6.1 Multiple Relationship

Direct mapping: The result is two object properties having the same domain and range, and no rule for how to assign a significant name to them.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: A Description of the Proposed Model. We Indicated Only the New Attributes that we have added to DDLUtils Model. Classes With (+) are Already Existing in the DDLUtils Model. Elements with (*) indicated zero to many.

		Description	Attributes/Elements	Description/Possible Values
Classes	Database+	Represents the studied database	Domain name (dn)	The business name (the subject area) modeled by the database. The default value is the name of the database.
	Table+	Represents a table	Ignored	Indicates whether the table must be ignored during the learning process. Default value is false.
			Domain name (dn)	A business name of the concept that the table represents.
			DnType	Indicates the type of the table from TableType.
	Enumerated-table	Table that stores an enumeration	Table	A reference to the table that models the enumeration.
			Column	Indicates the column that stores the enumeration.
			Value*(key, dn)	Indicates the domain values of the enumeration. *Key* is the value stored and *dn* is the domain value. The default value of *dn* is *key*.
	Multi-valued-table	Table that stores an enumeration	Table	A reference to the table that models the multi-valued attribute.
			Column	Indicates the multi-valued column.
	Composite-attribute- table	Table that models composite attributes	Table	A reference to the table within the database that presents the composite attribute.
			TransToClass	Indicates whether the table is to be translated to an OWL class or to add the attributes as properties of the OWL class resulting from the master table.
	Column+	Represents columns	SQLType	Indicates the SQL type of the column.
			Ignored	Indicates whether the column must be ignored during the learning process. Default value is false.
			Domain name (dn)	A business name of the property that the column represents.
			DnType	Indicates the type of the table from ColumnType.
			HasValue	Indicates a specific value for the column.
			IsNullable	Indicates whether the column may accept null values.
	Composite-column	Represents a composite column	Name	A business name of the composite column.
			Column* (name)	The columns that must be aggregated in one concept.
	Categorizing-column	Represents a categorizing column	Column	A reference to the column that stores categorizing values.
			Value*(key, dn)	Domain values of the categorization. The default values are the distinct values stored. *Key* is the value stored and *dn* is the domain value. The default value of *dn* is *key*.
	Flag-column	A discriminator column	Column	A reference to the flag column from the table.
			Concept* (key, dn)	Indicates the list of aggregated concepts. Each concept is defined by a pair (key, dn), where *key* is the value of the flag column that identifies the concept *dn*.
	Multi-valued-column	Represents a multi-valued column	Column	A reference to the multi-valued column from the table.
			Separator	The character used to separate the values in the column.
	Selective-column	Represents a selective column	Column	A reference to the column that stores selective values.
			Selection*(key, dn)	Equivalent domain values of selective values stored in the column. *Key* is the value stored and *dn* is the domain value. The default value of *dn* is *key*.
	Reference	Represents a unidirectional relationship from the local relation to the target relation	RelType	Indicates the type of the relationship from RelationshipType.
			RelName	Designates the relationship with a significant term.
	TableType	Enumeration (the types of tables)	NA (Not Applied)	regular, enumerated, multi-valued, composite-attribute
	ColumnType	Enumeration (the types of columns)	NA	regular, multi-valued, flag, categorizing, selective, enumerated
	RelationshipType	Indicates the types of relationships	NA	has, is-a, part-of, fragment-of, whole-part

Journal of Theoretical and Applied Information Technology
10th October 2014. Vol. 68 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

XML file with enrichment: The information that presents the enrichment is in bold. It consists of indicating that the column *isbom* is a *flag* column that identifies the BOM products from the other ones. The element *relType* in the element *reference* signals the type of the relation between the *BOM* product and *product*.

```
<table name="m_product" dn="product">
  <column name="isbom" dnType="flag"/>
  <flag-column name="isbom">
    < concept key="Y" dn="BOMProduct"/>
  </flag-column ></table>
<table name="m_product_bom" dn="BOMProduct"
    ignore="false" dnType="regular" >
  <column name="m_product_bom_id" ignore="false"
    dnType="regular"/>
  <column name="m_product_id" ignore="false"
    dnType="regular"/>
  <column name="m_productbom_id" ignore="false"
    dnType="regular"/>
<foreign-key foreignTableName="m_product"
  name="mproduct_mproductbom">
<reference local-column="m_product_id" foreign-
  column="m_product_id" relType="is-a"/>
</foreign-key>
<foreign-key foreignTableName="m_product"
  name="mproduct_bomproduct">
<reference local-column="m_productbom_id" foreign-
  column="m_product_id" relType="whole-part"/>
</foreign-key>
</table>
```

Translation to OWL:

There are no built-in modeling constructs in OWL for modeling part-whole relationships [37]. To preserve the semantics of the case study, we have chosen to model the *BOMProduct*, the *BOMProductPart* and the relation between them.

The *is-a* relation type identified between *m_product_bom* and *m_product*, in addition to the *flag* column *isbom* that indicates that instances of this table having a value of *isbom* equal to *'Y'* are BOM products. All these statements guide us to define the class *BOMProduct* (the *dn* of the table *m_product_bom*) as follows:

```
Declaration(Class(:BOMProduct))
Declaration(Class(:Product))
Declaration(DataProperty(:isbom))
DataPropertyDomain(:isbom :Product)
DataPropertyRange(:isbom xsd:string)
SubClassOf(:BOMProduct DataHasValue(:isbom
 "Y"^^xsd:string))
```

The *BOMProductPart* identifies for each *BOM* product the list of its products parts. The *BOMProductParts* has as properties the business information stored through the table *m_product_bom.* For instance, the property *bomqty* that designates the quantity by which a product participates in a *BOM*.

```
Declaration(Class(:BOMProductPart))
Declaration(DataProperty(:bomqty))
DataPropertyDomain(:bomqty :BOMProductPart)
DataPropertyRange(:bomqty xsd:positiveInteger)
SubClassOf(:BOMProductPart :Product)
```

Now we define the relations between classes: *Product*, *BOMProduct* and *BOMProductParts.* Two object properties are to be defined:

```
Declaration(ObjectProperty(:part-of-BOM Product))
ObjectPropertyDomain(:part-of-BOMProduct
:BOMProductPart)
ObjectPropertyRange(:part-of-BOMProduct :BOMProduct)
Declaration(ObjectProperty(:has-part-product))
ObjectPropertyDomain(:has-part-product :BOMProduct)
ObjectPropertyRange(:has-part-product :BOMProductPart)
InverseObjectProperties(:part-of-BOMProduct :has-part-
product)
```

As we can see our result is so far semantically more rich and consistent than the result of the direct mapping.

6.2 Recursive Relationship

The result of the direct mapping is similar to the previous case, where two object properties are created but in this case the domain and the range refer both to the same concept (*employee).*

XML file with enrichment: the objective is to nominate each relationship with a significant term.

```
<table name="employee" >
  <column name="id" …>
  <foreign-key foreignTableName="employee" …>
  <reference local-column="sp" foreign-
    column="id" relType="has" relName="hasSpouse" />
  </foreign-key>
<foreign-key foreignTableName="employee" …>
 <reference local-column="sup" foreign-
  column="id" relType="has" relName="hasSuperior" />
</foreign-key>
</table>
```

Translation to OWL: two object properties are created, but each one is designated with the *relName* given in the semantic enrichment phase.

```
Declaration(Class(:Employee))
Declaration(ObjectProperty(:hasSpouse))
ObjectPropertyDomain(:hasSpouse :Employee)
ObjectPropertyRange(:hasSpouse : Employee)
Declaration(ObjectProperty(:hasSuperior))
ObjectPropertyDomain(:hasSuperior :Employee)
ObjectPropertyRange(:hasSuperior :Employee)
```

6.3 Tables and Columns with Meaningless Names

This example is obvious and we can see it in the previous examples through the *dn* attribute used in order to define the classes and the properties with meaningful names.

6.4 Aggregated Concepts

Direct mapping: in existing approaches, there is no distinction of *flag* attributes that allow designating several concepts stored in one relation. These attributes are directly mapped to data type properties.

Journal of Theoretical and Applied Information Technology
10th October 2014. Vol. 68 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

XML file with enrichment: it consists of making explicit the concepts aggregated in the relation. The specification of *columnType* is optional. However, it allows detecting the *flag* columns for which the user has omitted to add the *flag-column* element.

```
1 <table name="c_bpartner " dn="Partner ">
2  <column name="iscustomer"  dnType="flag"/>
3  <column name="isvendor"  dnType="flag "/>
4 <flag-column column=" iscustomer ">
5      <concept  key="Y" dn="Customer"/>
6 </flag-column>
7 <flag-column column=" isvendor ">
8      <concept  key="Y" dn="Vendor"/>
9 </flag-column>
   …
10 </table>
```

Translation to OWL: we have to create the properties that represent the flag attributes. The concepts created are sub-classes of the class that represent the relation, with the restriction on the value of the flag attributes.

```
1 Declaration(Class(:Partner))
2 Declaration(Class(:Customer))
3 Declaration(DataProperty(:iscustomer))
4 DataPropertyDomain(:iscustomer : Partner)
5 DataPropertyRange(:iscustomer xsd:string)
6 SubClassOf(:Customer DataHasValue(:iscustomer
 "Y"^^xsd:string))
7 The same for Vendor
```

6.5 Enumerated Relations

Direct mapping: we neglect the audit information that is not business information. We focus on the relation and the attribute that stores the enumerated values. The direct mapping does not distinguish this kind of relation from the others and the result is a class that models the relation and data properties for the attributes.

XML file with enrichment: it consists of specifying the type of the table by adding the attribute *dnType* and defining the enumerated attribute and domain values through the element *enumerated-table*.

```
1 <table name=" product_uom_categ "
2   dnType="enumerated" dn="ProductUOMCategory"/>
3  <column name="name"  … />
…
4 </table>
5 <enumerated-table  table=" product_uom_categ " >
6   <value  key="V"  dn="Volume"/>
7   < value  key="H"  dn="Height "/>
8   < value  key="W"  dn=" Weight "/>
9 </enumerated-table>
```

Translation to OWL: for more reusability, we proposed to map this case to a new restricted data type through *DatatypeDefinition* feature of OWL.

```
1Declaration(Datatype (:ProductUOMCategory))
2 DatatypeDefinition(:ProductUOMCategory
      3     DataOneOf("Height" "Volume" "Weight"))
```

6.6 Enumerated Attributes

Direct mapping: in [27] the authors proposed the use of *oneOf* feature of *OWL*.

XML file with enrichment: it consists of making explicit the *selective* nature of the attribute and the domain values of its stored and codified ones.

```
1  <table name="m_product " dn="Product ">
2  <column name="producttype " dnType="selective"/>
3  <selective-column  column="producttype " />
4   <selection key= "E "   dn= "Expense " >
5   <selection key= "I "   dn= "Item" >
6   <selection key= "O "   dn= "Online" >
7  <selective-column />
8 </table>
```

Translation to OWL: we propose to use the *DataOneOf* construct. We define a new type in order to use it when necessary, without duplicating the declaration of OneOf or DataOneOf for other classes.

```
1 Declaration(Datatype (:ProductType))
2 DatatypeDefinition(:ProductType
3 DataOneOf("Expense" "Item" "Online" ))
4 Declaration(Class(:Product))
5 Declaration(DataProperty(:producttype))
6 DataPropertyDomain(:producttype : Product)
7     DataPropertyRange(:producttype  :ProductType)
```

6.7 Composite Attributes

Direct mapping: to our knowledge, no approach had proposed a way to drive such semantics form the RDB. In [18], the authors proposed to map (from the ER model) the composite attribute (*address*) to a *datatype* property and then to map its simple component attributes to sub-properties of the created *datatype* property.

XML file with enrichment: we distinguished the two cases (A) and (B). For (A), the enrichment involves the assigning of the type *composite-attribute* to the table *Address.* We qualify the relation with the table *employee* to be of type *has.* For the case (B), the enrichment includes the definition of an element of type *composite-column* with the set of columns (from the table) that forms this element.

```
   <!--Case (A) -->
1 <table name="employee" >
2   <column name="id"  primaryKey="yes" >
3   <foreign-key foreignTableName="address "   …>
4   <reference local-column="add_id" foreign-
5    column="id" relType="has" />
6   </foreign-key>
7 </table>
8<table name="address " dn="Address"
9     dnType="composite-attribute">
10  <column name="street"  … />
11  <column name="city" />
12 </table>
   <!—End Case (A) -->
   <!--Case (B) -->
13 <table name="employee" >
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
14  <column name="id"  primaryKey="yes" >
15  <column name="street"  … />
16  <column name="city" />
17  <composite-column  name="Address"/>
18      <column name="street"/>
19      <column name="city"/>
20  <composite-column/>
21 </table>
   <!—End Case (B) -->
```

Translation to OWL: for both cases (A) and (B), we propose to map the composite attribute to a class. In this way, the class of composite attribute can be reused when necessary.

```
1  Declaration(Class(:Employee) )
2  Declaration(Class(:Address))
3  Declaration(DataProperty(:hasCity))
4 Declaration(DataProperty(:hasStreet))
5  Declaration(ObjectProperty(:hasAddress))
6  ObjectPropertyDomain(:hasAddress  :Employee)
7  ObjectPropertyRange(:hasAddress :Address)
8  DataPropertyDomain(:hasStreet  :Address)
9  DataPropertyRange(:hasStreet  xsd:string)
10  DataPropertyDomain(:hasCity  :Address)
11 DataPropertyRange(:hasCity  xsd:string)
```

7 CONCLUSION

The role of ontology is to model a domain in an unambiguous way. In this context, it is evident that the meaning of concepts and the relations between them should be clear and well designed. However, driving ontology directly from an RDB is far from capturing accurate semantics of the domain application. In fact, the relational model is less expressive and does not support some conceptualizations.

Our approach proposed the enrichment of the RDB model before processing on ontology learning. It has many advantages, including: (i) it takes into consideration some uncommonly studied semantics; (ii) it gives the possibility to preserve the additional semantics when the database schema is changed. Indeed, reported changed XML file of the database and the old one containing additional semantics can be merged.

However, an intensive manual work is required in our approach at the enrichment phase. So, we have to think about how to provide a tool for collaboration, and how to automatically add some 'candidate semantics' (from the proposed meta-data of the XML schema) in the XML file. Moreover, as future work, we are working on extending our model to treat complex check and unique constraints. Some experimentation will be published in order to measure the benefit of our approach compared to existing ones.

REFERENCES:

[1] T. Gruber et al., "A translation approach to portable ontology specifications," Knowledge acquisition, vol. 5, no. 2, pp. 199–220, 1993.

[2] A. Gómez-Pérez, D. Manzano-Macho et al., "A survey of ontology learning methods and techniques," OntoWeb Deliverable D, vol. 1, p. 5, 2003.

[3] H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner, "Ontology-based integration of information-a survey of existing approaches," in IJCAI-01 workshop: ontologies and information sharing, vol. 2001. Citeseer, 2001, pp. 108–117.

[4] R. H. Chiang, T. M. Barron, and V. C. Storey, "Reverse engineering of relational databases: Extraction of an eer model from a relational database," Data & Knowledge Engineering, vol. 12, no. 2, pp. 107–142, 1994.

[5] B. El Idrissi, S. Baïna, and K. Baïna, "A methodology to prepare real-world and large databases to ontology learning," in Enterprise Interoperability VI. Springer, 2014, pp. 175–185.

[6] M. Lezoche, H. Panetto, A. Aubry et al., "Conceptualisation approach for cooperative information systems interoperability," in 13th International Conference on Enterprise Information Systems, ICEIS 2011, 2011, pp. 101–110.

[7] L. Obrst, "Ontologies for semantically interoperable systems," in Proceedings of the twelfth international conference on Information and knowledge management. ACM, 2003, pp. 366–369.

[8] Openbravo. http://wiki.openbravo.com/wiki.

[9] UML. http://www.uml.org/.

[10] OpenERP. https://www.odoo.com.

[11] A. Jaleel, S. Islam, A. Rehmat, A. Farooq, and A. Shafiq, "Ontology construction from relational database," International Journal of Multidisciplinary Sciences and Engineering, vol. 2, no. 8, November 2011.

[12] M. R. C. Louhdi, H. Behja, and S. O. El Alaoui, "Transformation rules for building owl ontologies from relational databases," Computer Science, 2013.

[13] N. Cullot, R. Ghawi, and K. Yétongnon, "Db2owl: A tool for automatic database-to-ontology mapping," in Proceedings of the 15th Italian Symposium on Advanced Database Systems (SEBD 2007), Torre Canne di Fasano (BR), Italy, 2007, pp. 491–494.

[14] W3C. (2004, February) Owl web ontology language. http://www.w3.org/TR/owl-features/.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[15] SAP. http://www.sap.com/.
[16] U. Hohenstein and V. Plesser, "Semantic enrichment: A first step to provide database interoperability," in Workshop Föderierte Datenbanken, Magdeburg. Citeseer, 1996, pp. 3–17.
[17] F. Cerbah, "Learning highly structured semantic repositories from relational databases," The Semantic Web: Research and Applications, pp.777–781, 2008.
[18] M. Fahad, "Er2owl: Generating owl ontology from er diagram," Intelligent Information Processing IV, pp. 28–37, 2008.
[19] M. I. Hwang, J. D. Becker, and J. W. Lin, "Representing multivalued attributes in database design," IACIS 2001.
[20] B. El Idrissi, S. Baïna, and K. Baïna, "Automatic generation of ontology from data models: A practical evaluation of existing approaches," in RCIS, 2013 IEEE, pp. 1–12.
[21] B. El Idrissi, S. Baïna, and K. Baïna, "Ontology learning from data models: Advance and the requirement for a database preparation and enrichment process," in ISKO-Maghreb, 2013 IEEE, pp. 1–8.
[22] H. Santoso, S. Haw, and Z. Abdul-Mehdi, "Ontology extraction from relational database: Concept hierarchy as background knowledge," Knowledge-Based Systems, vol. 24, no. 3, pp. 457–464, 2011.
[23] I. Astrova, "Reverse engineering of relational databases to ontologies," in The Semantic Web: Research and Applications, ser. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, vol. 3053, pp. 327–341.
[24] I. Astrova, N. Korda, and A. Kalja, "Rule-based transformation of sql relational databases to owl ontologies," in Proceedings of the 2nd MTSR. Citeseer, 2007.
[25] C. Ramathilagam and M. L. Valarmathi, "A framework for owl dl based ontology construction from relational database using mapping and semantic rules," International Journal of Computer Applications, vol. 76, no. 17, pp. 31–37, August 2013, published by Foundation of Computer Science, New York, USA.
[26] N. Alalwan, H. Zedan, and F. Siewe, "Generating owl ontology for database integration," in Advances in Semantic Processing, 2009. SEMAPRO'09, IEEE, 2009, pp. 22–31.
[27] K. Albarrak and E. Sibley, "An extensible framework for generating ontology models from data models, int'l transactions on sys," Science and Apps.(ITSSA), vol. 6, no. 2/3, pp. 97–112, 2010.
[28] T. Halpin, "Ontological modeling: Part 13," Business Rules Journal, vol. 14, no. 3, March 2013.
[29] C. Diamantini and N. Boudjlida, "About semantic enrichment of strategic data models as part of enterprise models," in Business Process Management Workshops. Springer, 2006, pp. 348–359.
[30] N. Boudjlida, H. Panetto, S. Baïna, C. Diamantini, J. Krogstie, Y. Lin, J. Sarraipa, N. Zouggar, A. Hahn, M. Delgado, M.-A. Abian, and M.-J. Nunez, "DTG4.2: Experimental Semantic Enrichment of Enterprise Models for Interoperability and its Practical Impact," May 2007, 118 pages http://www.interop-vlab.eu Deliverable of the INTEROP Network of Excellence Européen.
[31] M. Castellanos, F. Saltor, and M. Garcia-Solaco, "A canonical model for the interoperability among object-oriented and relational databases." in IWDOM, 1992, pp. 309–314.
[32] M. A. Maatuk, A. Ali, and N. Rossiter, "Semantic enrichment: The first phase of relational database migration," in Innovations and Advances in Computer Sciences and Engineering. Springer, 2010, pp. 373–378.
[33] DdlUtils. https://db.apache.org/ddlutils/.
[34] SQuirrel. http://squirrel-sql.sourceforge.net/.
[35] Commons-betwixt. http://commons.apache.org
[36] W3C. (2012, December) Owl 2 web ontology language document overview (second edition). http://www.w3.org/TR/owl2-overview/.
[37] ——. (2005, Aug) Simple part-whole relations in owl ontologies. http://www.w3.org/.

Journal of Theoretical and Applied Information Technology
10th October 2014. Vol. 68 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

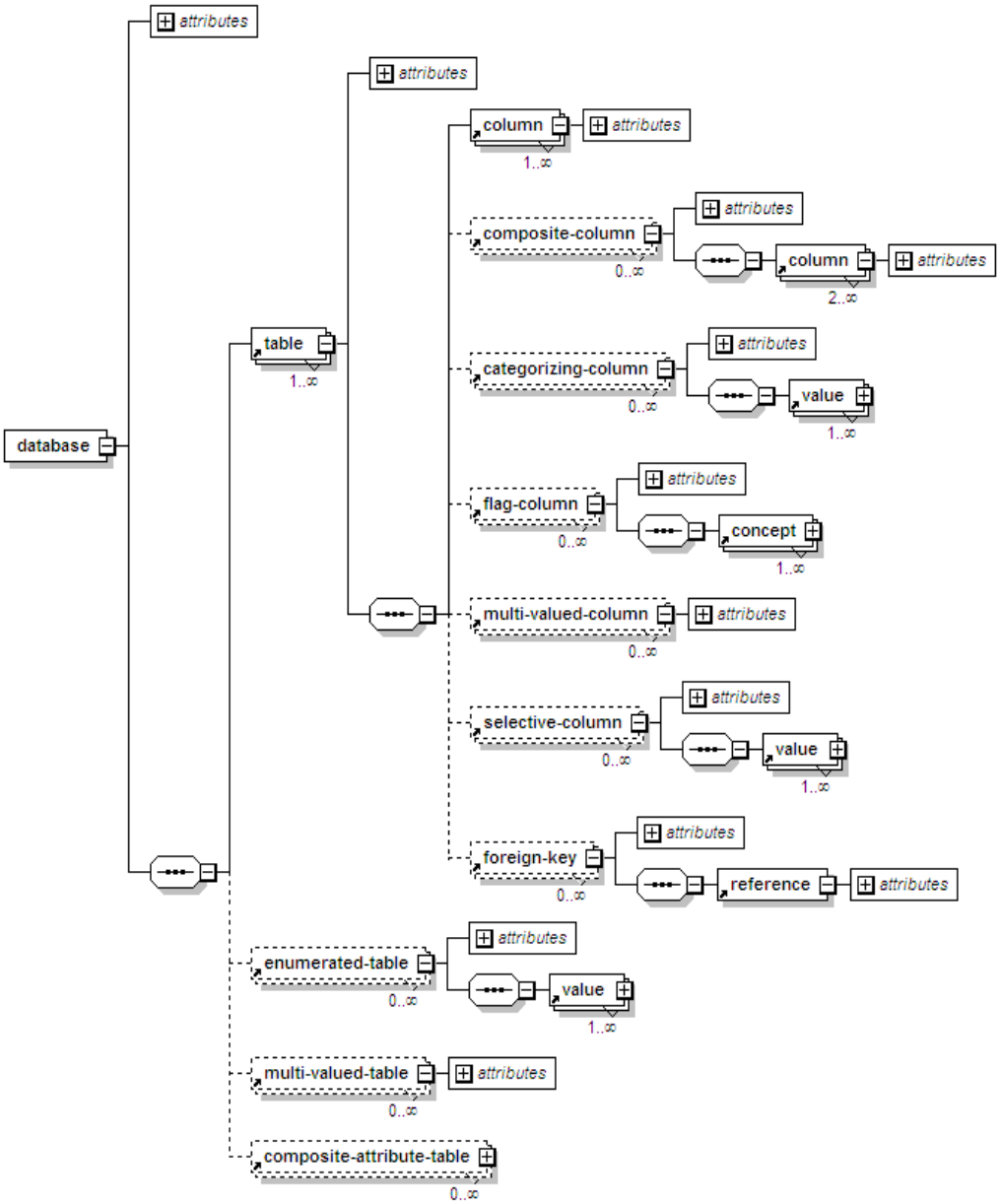


Figure 10: XML Schema of the Proposed Model.