Automatic Conversion Method of Class Diagrams to Ontologies Maintaining Their Semantic Features

Mohamed Bahaj, Jamal Bakkas

Abstract— In this paper, we propose a new conversion's method from UML class diagram to ontology in order to serve the Semantic Web. The ontology which results from the conversion is expressed in OWL / XML. This method allows us to preserve semantic of some feature's UML diagram such as inheritance, encapsulation, types of associations (composition, aggregation, or simple association), constraints of integrity, class identifier...etc.

Keywords— UML, ontology, mapping, OWL.

I. INTRODUCTION

Several works of databases to semantic web migration exist which began with conversion of database schema to XML / XML Schema standards [4], and result in implementation of XMI standard for models conversion and exchange [2]. However this standard has some problems of implementation, which has prompted research in other axes, especially with the advent of the knowledge representation language OWL [1], [2]. Thus, there are formalisms of mapping from databases to ontology like RO2 [11] and D2RQ [Bizer 2003], and conversion tools like RDB2Onto [9], DataMaster [10]...

In this paper we propose an automatic method to convert a UML class diagram to an ontology using OWL / XML language with keeping the features meaning of the diagram.

If our proposed method allows representing the basics concepts such as inheritance, identifier of class… the major contribution is solutions we have conceived to preserve the semantic of types of associations and also that of the notion of attributes encapsulation after conversion.

The remainder of this paper is organized as follows: Section 2 describes the steps involved in the conversion process. Section 3 describes the proposed solution to convert UML class and its attributes while keeping the notion of inheritance between classes and the attributes encapsulation. Section 4 describes the conversion of associations while retaining their type and cardinalities. Section 5 presents the algorithm of conversion. Section 6 describes the implementation with a case study. The last section is devoted to the conclusion and perspectives.

II. DESCRIPTION OF METHOD

Our approach provides an algorithm for mapping in a schema level; it takes as input a class diagram that undergoes conversion to generate ontology as a set of concepts with data type properties. Those concepts are semantically related to each other by object properties and hierarchical relationships while keeping the semantic of converted class diagram

Thus, to preserve the notion of inheritance, we exploited the hierarchy of concepts provided by ontologies to represent inheritance of classes in UML. And to maintain the notion of encapsulation, our approach proposes a determined structure of data type properties, which reflect the visibility levels of UML converted attributes. Another feature of our proposition concerns the proposed structure of the object properties for keeping the meaning of the association types of UML (i.e. composition, aggregation).

Inheritance, encapsulation, composition, aggregation, class identifier ... etc, are UML concepts that carry a meaning and their conversion, while maintaining this meaning, is possible because the ontology allows to represent meaning.

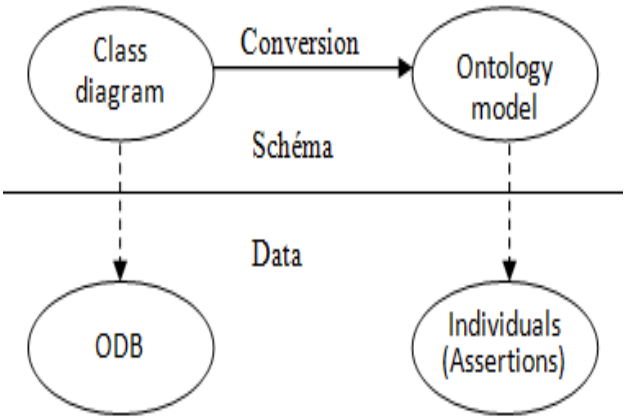

Fig. 1: conversion at the Schema level

In this procedure a class diagram is considered as a set of classes. Each class is characterized by a name, a list of its attributes, a list of relationships (roles) linking this class with other classes, as well as the name of its parent class. If a class has no superclass, it is considered subclass of the superclass Object. An attribute can be the identifier of its class or a simple attribute, each of them has a type that is one of predefined types of UML. A relationship represents a role; it is characterized by its name, type (aggregation, composition, or simple association), the target class, and target cardinalities.

Formally a class diagram can be represented as follows:

Manuscript received November 28, 2013.

Jamal BAKKAS, Department of Mathematics and computer science, University Hassan I, FSTS, Settat, Morocco.

Mohamed BAHAJ, Department of Mathematics and computer science, University Hassan I, FSTS, Settat, Morocco.

classDiagram= { C /
C={ (**classeName**, **classeParent** , **attributesList** , **relationShipsList**) /
classeName ∈ classesNamesList
classeParent ∈ classesNamesList ∪ {"Object"}
attributesList={(attributeName, ,is_id, attVisibility, nbOcuurences, attType) /
attType ∈ AttributesTypesList
nbOcuurences > 0
}
relationShipsList = { (relationName , relationType , classeTarget , cardTarget) /
relationType ∈ {"AGGREGATION", "COMPOSITION","ASSOCIATION"}
classeTarget ∈ classesNamesList ∪ {"Object"}
cardTarget ∈ {"0..1" , "1..1" , "1..*" , "*" }
}
}
}

Fig. 2: detailed description of the class diagram

With:

- *classesNamesList* : names of classes that compose the class diagram,
- *AttributesTypesList* : list of predefined data types of UML,
- *is_id* : Boolean which set to true if the attribute is an identifier otherwise it is set to false
- *nbOcuurences* : If the attribute is an array of primitive type then nbOcuurences> 1, otherwise nbOcuurences= 1.

III. DESCRIPTION OF THE MAPPING ALGORITHM

A. Classes

Before starting the conversion, we create a class OWL called *Object* to represent the superclass *Object* of UML. Recall that all UML diagram classes inherit default from the *Object* class. And then each class of diagram is converted into an OWL concept with the same name.

B. The Inheritance between Classes

Inheritance is one of the fundamental notions in UML whose preservation after the mapping is of a major utility. Thus we propose to use the hierarchy of concepts provided by the ontology to represent this notion.

Each inheritance relationship between two classes in UML is translated to a relationship hierarchy (is_a) between two concepts in ontologies. Thus, any concept of the resultant ontology has a relationship of hierarchy, either with other concept or with the super concept *Object* that will be the root of the ontology.

C. Attributes

An attribute of class with an UML primitive type is mapped to property data type defined using the *DatatypeProperty* class; however, data type properties in OWL can have multiple values for a given instance, which is not the case for an UML attribute (atomicity). In order to remedy this problem; we declare the property as a functional property with *FunctionalProperty* class as follows:

```
<owl:FunctionalProperty rdf:about="#name" />
```

Recall that for a functional property, there cannot be two distinct values y1 and y2 such that the pairs (x, y1) and (x, y2) are both instances of this property.

To represent array type attributes, we use non-functional data type properties, but with a cardinality restriction limiting the maximum number of occurrences to the size of the converted array using the OWL class: *maxCardinality* as follows:

```
<owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
        nbOccurrences
</ owl: maxCardinality>
```

D. Encapsulation

The solution that we propose to keep the semantics of encapsulation is among the strengths of our method. Indeed, we propose to create a super-property "Attribute" with *DatatypeProperty* class, and then create three sub-properties of the property "*Attribute*", These properties represent the three visibility levels of UML attributes: *private* (-), *protected* (#), and *public* (+). They are created using the *rdfs:subPropertyOf* class as follows:

```
<owl:DatatypeProperty rdf:ID="private">
        <rdfs:subPropertyOf rdf:resource="#Attribute"/>
</ owl: DatatypeProperty>
```

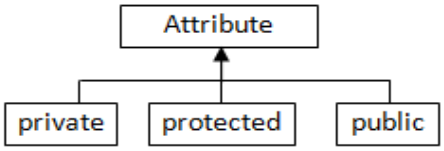

Fig. 3: The Hierarchy of Data Type Properties

A Thus, we obtain the previous structure [Fig. 3]. Subsequently, all attributes are represented by data type properties that inherit directly and necessarily of one of the three properties: *private*, *protected* or *public*, as the following example shows:

```
<owl:DatatypeProperty rdf:ID="matricule">
   <rdfs:domain rdf:resource="#Professor"/>
   <rdfs:range rdf:resource="&xsd;integer"/>
   <rdfs:subPropertyOf rdf:resource="#private"/>
</owl:DatatypeProperty>
```

We chose to name the super-property "*Attribute*", because all the final properties which are sub-properties indirect of "*Attribute*" are conversions of UML attributes.

E. The Identifier Of The Class

The identifier of the class is considered as a simple attribute; therefore, it is converted into a data type property as described above. And because the identifier must have a unique value for any given individual, then the property should be mentioned as inverse functional using OWL class *owl:InverseFunctionalProperty* as follows:

```
<owl:InverseFunctionalProperty rdf:about="#matricule"/>
```

IV. ASSOCIATIONS CONVERSION

An association is seen as two roles, these roles can be mapped intuitively by two object properties one is the inverse of the other using the ObjectProperty class:

```
<owl:ObjectProperty
rdf:ID="teaches">
        <rdfs:domain
```

```
rdf:resource="#professor"/>
    <rdfs:range rdf:resource="#course"/>
    <owl:inverseOf rdf:resource="#TaughtBy"/>
</owl:ObjectProperty>
```

A. Associations Type

Among the difficulties that arise at this level, the representation of these types of relationships (aggregation, composition, or simple association). To remedy this problem, several solutions have been proposed, among them, adding annotation properties to the object property that represents the relationship. This possibility is offered by OWL DL. However, these properties do not have semantic value; they are not used by reasoners in the process of reasoning, which leads to a semantic imperfection of converted diagrams.

Our proposal consists to structure the object properties representing the relationships in a hierarchical form whose root is the super-property "*ASSOCIATION*" which has two sub-properties "*AGGREGATION*" and "*COMPOSITION*" [Fig. 4]. Then each relationship of class diagram is represented by an object property that inherits necessarily one of the three previous properties.

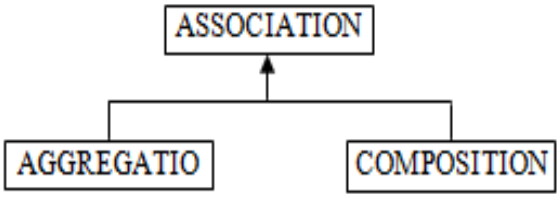

Fig. 4: Structure of Relationship Types

We opted for this hierarchy, because it means that all relationships are associations. Those who inherit directly of the property "*ASSOCIATION*" or those who inherit them via the two others

B. Cardinalities Conversion

OWL Classes: cardinality, maxCardinality, minCardinality, allow applying cardinality restrictions and value restrictions to property linking two concepts of ontology. We use these classes to represent UML cardinalities by applying cardinality restrictions to object properties representing the relations of diagram. If the association is an integrity constraint (cardinality 1..1), we propose to use a functional property to the linker of the cardinality restriction to value 1. Below a table of various restrictions to apply to object properties based on the UML cardinalities:

Table I: Cardinality Restrictions Applied To Object Properties

Card	Restrictions
0..1	<owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger"> 1 </owl:maxCardinality>
1..1	<rdf:type rdf:resource="&owl;FunctionalProperty" />
1..*	<owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger"> 1 </owl:minCardinality>
*	no restriction

V. THE CONVERSION ALGORITHM

The algorithm of conversion is as follow:

```
Create Class "Object"
Create ObjectProperties "ASSOCIATION"
Create ObjectProperties "COMPOSITION",
"AGGREGATION" sub-propertyof "ASSOCIATION"
Create DataTypeProperty "Attribute"
Create DataTypeProperty "private","protected","public"
sub-propertyof "Attribute"
FOR EACH C of classDiagram do
Create Class "C.classeName" sub-ClassOf
"C.classeParent"
  For each A of C.attributList do
      Create DataTypeProperty "A.attributeName "
      sub-PropertyOf "A.attVisibility"
      with Domain "C.ClasseName"
      with Range "getType(A.attributType)"
      if   A.is_id  then
           "A" is InverseFunctionalProperty
      else
        if  nbOcuurences=1  then
              "A" is FunctionnalProperty
        else  maxCardinality= nbOcuurences
        endif
      endif
  End For
  For each  R  of C.relationShipsList  do
    Create ObjectProperty "R" sub-PropertyOf
    "R.relationType"
    with Domain "C.ClasseName"
    with Range "R.classTarget" InverseOf
 "getRelationName(C.className,R.classTarget)"
    RestrictionCardinalities(C.R.cardTarget)
  End For
END FOR
```

Fig. 5: Algorithm of Conversion

With

- *getType (UMLDataType):* a function that takes as parameter the UML primitive data type and returns the corresponding XSD type.
- *getRelationName (C1, C2):* a function that returns the name of the relationship (role) that connects C1 with C2
- *RestrictionCardinalities(UMLCardinality):* a function that takes as parameter UML cardinality and returns the cardinality restrictions corresponding [Table I]

A. The OWL sublanguage used:

All previous conversions can be expressed using only OWL Lite except attribute conversions. To convert these, we are forced to switch to OWL Full to apply Functional Property and Inverse Functional Property, which are reserved to the object properties, to data type property. Because in OWL, Full DatatypeProperty class is a subclass of ObjectProperty class, unlike in OWL DL and OWL Lite where the two classes are disjoint.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

VI. CASE STUDY

Consider the following case study: A university is an establishment, which is composed of a set of departments to whom attached a set of teachers. This example thus illustrates three types of relationships that are: inheritance, aggregation and composition, and attributes with three levels of visibility.

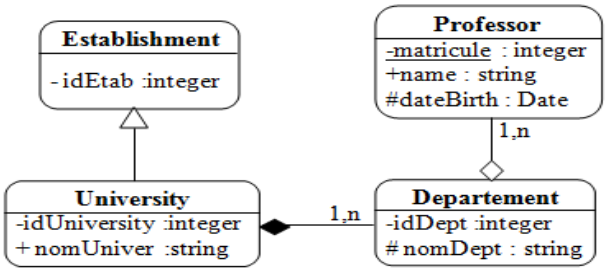

Fig. 6: Example of a Class Diagram

Detailed description of the class diagram [Fig. 7], as we described [in Fig. 2], is stored in a text file. This file is the input of our system, which parses it using the previous algorithm [Fig. 5] and generates an output file containing the resultant ontology of the conversion.

```
classDiagram={
(Professor,Object,{(matricule,private,
1,integer,true),(name,public,1,string,
false),(dateBirth,protected,1,date,
false)},{(isAttached,AGGREGATION,
Departement,1)}),
(Departement,Object,{(idDept,private,1
,integer,true),(nomDept,protected,1,
string,false)},{(groups,AGGREGATION,
Professor,1..*),(compose,COMPOSITION,
University,1)}),
(University,Establishement,{(
idUniversity,private,1,integer,true),
(nomUniver,public,1,string,false)},
{(isComposedOf,COMPOSITION,Departement
,1..*)}),
(Establishement,Object,{(idEtab,
private,1,integer)},{(,,,)})
}
```

Fig. 7: Detailed Description of the Class Diagram

To test the semantic consistency of our resultant ontology, we loaded it under Protegé, and using the OntoGraf plugin we obtained the hereafter form [Fig. 8]. Then we created individuals for different classes with assertions of their properties without applying any reasoner [Fig. 9(a)].

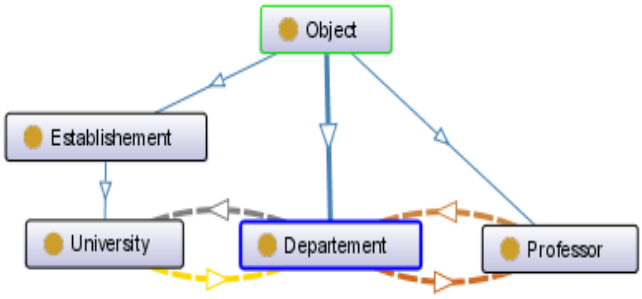

Fig. 8: Ontograf Diagram of the Resultant Ontology

Subsequently, we applied a reasoner. The version used of Protégé integrates both reasonneurs Fact + + and Hermit. If we take, for example, an individual *P1*, of *Professor* type, and an individual *mathematics* of *Department* type, connected to each other by the object property *IsAttached*, the reasoner automatically infers that a *IsAttached* is an association and that this association is of type: aggregation, as shown below [Fig. 9(b)]. The same for the attributes encapsulation.

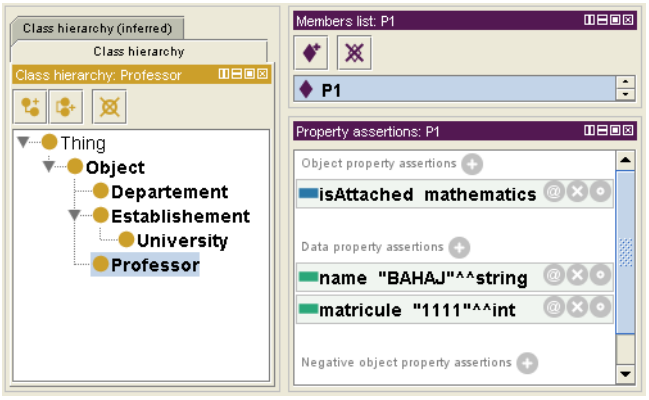

Fig.9 (A): Before the Launch The Reasoner

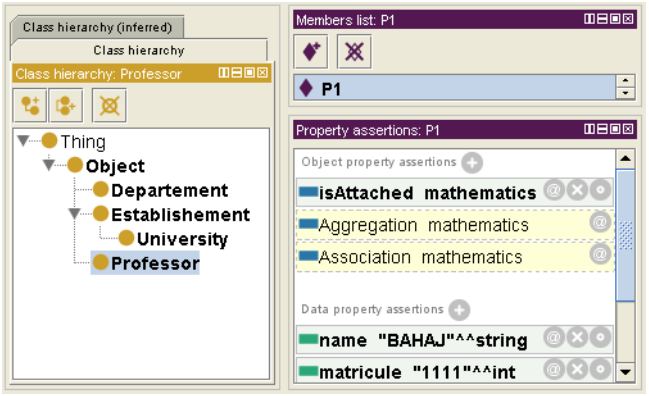

Fig. 9(B): After the Launch the Reasoner

VII. CONCLUSION AND PERSPECTIVES

Our proposal differs from antecedent proposals by preservation of the semantics of some specific characteristics of the class diagram, namely inheritance, attributes encapsulation, the types of associations (composition and aggregation), and cardinalities.

In this paper we propose a conversion at the schema level between a UML class diagram and the model part of ontology (TBOX). Our future work will focus on the level "data" [Fig. 10] to convert an object database to the instance's part of ontology (ABOX), which contains assertions of different elements of schema level. After this conversion, we will discuss the querying of resultant ontology in analogy to that of an object database.

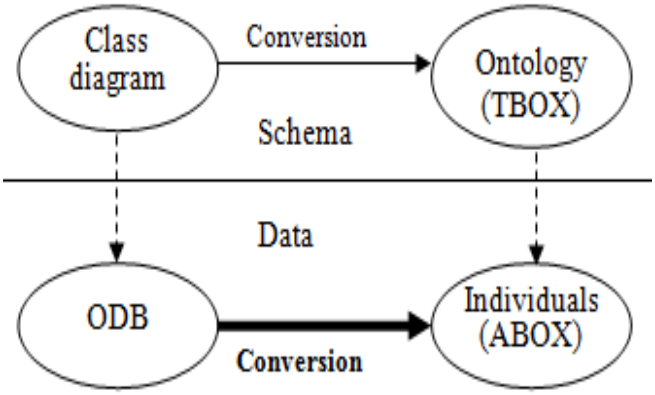

Fig. 10: Conversion At The "Data" Level

REFERENCES

1. G. Antoniou, F. van Harmelen "Web Ontology Language: OWL". pages 76-92 Springer-verlag .2003
2. D. L. McGuinness, F. van Harmelen, http://www.w3.org
3. M. R. Jensen, T. H. Møller Torben, B. Pedersen "Converting XML Data to UML Diagrams For Conceptual Data Integration". Data & Knowledge Eng., vol. 44, no. 3, pp. 323-346, 2003
4. J. Fong, F. Pang, C. Bloor "Converting Relational Database into XML Document". DEXA Workshop, pp 61-65. 2001
5. N. GHERABI, K. ADDAKIRI, M. BAHAJ "Mapping relational database into OWL Structure with data semantic preservation". CoRR abs/1205.5922. 2012
6. J. Seidenberg, A. Rector "Web Ontology Segmentation: Analysis, Classification and Use". IW3C 2006. ACM, 2006
7. M. Arnoux, T. Despeyroux " Multi-représentation d'une ontologie : OWL, bases de données, systèmes de types et d'objets". CoRR abs/1104.2982. 2011
8. D. Gasevic, D. Djuric, V. Devedzic, V. Damjanovi "Converting UML to OWL ontologies". In Proceedings of the 13 th International World Wide Web Conference, NY, USA, pp. 488-489. 2004
9. M. Šeleng, M. Laclavík, Z. Balogh, L. 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, 113–116. 2007
10. C. Nyulas, M. O'Connor, S. Tu "DataMaster – a Plug-in for Importing Schemas and Data from Relational Databases into Protégé" In Proceedings of 10 th International Protégé Conference, Budapest, Hungary, 2007
11. J. Barrasa, Ó. Corcho, A. Gómez-Pérez "R2O, an Extensible and Semantically Based Database to ontology Mapping Language". In Proceedings of the 2nd Workshop on Semantic Web and DatabasesSWDB2004Springer, p. 1069-1070, 2004

AUTHORS PFOFILE

Jamal Bakkas, was born in 1979, in Marrakech, Morocco. He is PhD student in the Department of Mathematics and computer science, FSTS, University Hassan I, Settat, Morocco. His area of interest includes web ontologies and semantic web.

Mohamed Bahaj, was born in 1964, in ouezzane, Morocco. He got his PhD in Applied Mathematics, from University of Pau, France, in 1993. He is now working as a Professor at the Department of Mathematics & Computer Sciences, University of Hassan 1er, Faculty of Sciences & Technology Settat, Morocco. His research interests include pattern recognition, Semantic web & Ontology in MAS, Controls of mobiles agents.