# Transformation of UML Models towards OWL Ontologies

Aissam BELGHIAT
Department of Computer Science
University of Md Boudiaf
Msila, 28000, Algeria
Belghiatissam@gmail.com

Mustapha BOURAHLA
Department of Computer Science
University of Md Boudiaf
Msila, 28000, Algeria
mbourahla@hotmail.com

***Abstract*—This article presents a contribution to the work taken in the bridging software development and ontology technologies. This contribution makes it possible to generate automatically an ontology OWL (Web Ontology Language) starting from a model UML and to profit from the convergence between UML and OWL. Our approach is carried out in the Eclipse platform, we use a style sheet XSL to make the transformation of a model UML describes in Topcased in an OWL ontolgy represented in RDF/XML form. The transformation is based on transformation rules which make it possible to connect the concepts of UML and OWL.**

***Keywords- UML; Ontologie; OWL; Topcased; XSLT.***

## I. INTRODUCTION

UML is most commonly used in the requirements specification and design of object oriented software in the middle tier of enterprise applications. It is inspired from the Merise method and it became a standard. In the other side, ontologies became an essential means for the semantic Web and which are described formally using description logics which are implemented in a standard language called OWL. The development of ontologies with OWL is a very difficult task.

In this work we propose a set of transformation rules of models UML (we choose class diagrams) into ontologies which will be described in the OWL language. These rules will be implemented in a software to automate this transformation. Thus, we benefit from the UML language in order to have models on ontologies to make preliminary analyzes and implementations OWL to test consistence of ontologies.

## II. UML

UML (Unified Modeling Language) is a language to visualize, specify, build and document all the aspects and artifacts of a software system [6].

As its name indicates it, UML is the result of the fusion of a whole of other languages of modeling (OMT, Booch, and OOSE). It is quickly became a standard that is impossible to avoid it.

In practice, UML relates to any system, in the broad sense (nonrestrictive with the only software systems), built by using the approach oriented object (OO).

UML 2 comprises thirteen (13) types of diagrams representing many distinct visions to represent particular concepts of the information system, these diagrams are: Activity diagram, Class diagram, Communication diagram, Component diagram, Composite structure diagram, Deployment diagram, Interaction diagram, Object diagram, Package diagram, Sequence diagram, State machine diagram, Timing diagram, Use case diagram. Figure 1 represents a classification of the types of UML diagrams.

The class diagram is considered as the most important of object oriented modeling, it is the only obligatory one in such modeling, it shows the internal structure of a system and makes it possible to provide an abstract representation of its objects which will interact together to realize the use cases [1].

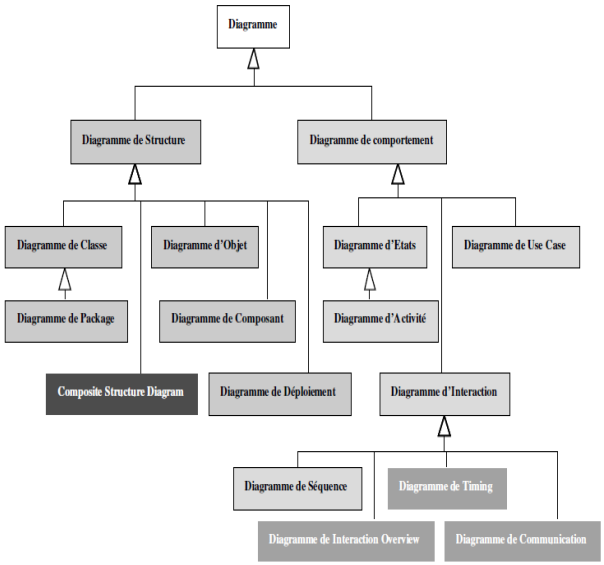



Figure 1. Types of UML diagrammes [3]

## III. OWL

OWL (Ontology Web Language), was recommended by the W3C in 2004, and its version 2 in 2009, it is designed for

the applications needing to treat the contents of information instead of simply presenting them to the human ones. Language OWL allows an interpretation of the Web contents by the machines higher than that offered by the languages XML, RDF and RDF schema (RDF-S), by providing an additional vocabulary with a formal semantics.

Language OWL1 offers three sub-languages with increasing expression directed for specific developers communities and users: OWL Lite, OWL DL, and OWL Full [8].

OWL Lite Language answers for needs of hierarchy of classification and functionalities of simple constraints of cardinality 0 or 1.

OWL DL Language concerns the users who wish a maximum expressivity coupled with the completed of calculation (that means that all the inferences will be ensured to be taken into account) and the decidability of the reasoning system (i.e. all calculations will be finished in a finished time interval). This language includes all structures OWL with certain restrictions, like the separation of the types: a class cannot also be an individual or a property. It is named DL because it corresponds to descriptive logic.

Language OWL Full is intended to the people wishing a maximum expressivity. It has the advantage of the full compatibility with RDF/RDFS, but the disadvantage of having an high level of capacity of description, and not to be able to guarantee the completed and the decidability of calculations related to ontology.

OWL2 Language defines three new profiles (sub-languages) conceived to meet various applicative needs: OWL2 EL, OWL2 QL, OWL2 RL [11].

OWL2 Language EL is particularly useful in the applications using of ontologies which contain a very great number of properties and/or classes. This profile reflects the capacity of expression used by good number of these ontologies and is a subset of OWL2. With OWL2 EL the reasoning can be carried out in polynomial time which depends on the size of ontology.

OWL2 QL Language is directed for applications which use very great volumes of data, therefore of authorities. The conjunctive requests are the principal form of requests implemented in the management systems of databases. Therefore, OWL2 QL must make it possible to solve these requests in polynomial time because these applications from their use are the request object many. The expressive power of this profile inevitably is very limited, but it includes/understands the majority of the main features of the conceptual models, such as the UML class diagrams and of the ER diagrams (entities associations).

OWL2 RL Language is intended for applications which require great capacities of reasoning without to sacrifice too much expressive power. As considering previously, it was inspired by OWL DL and DLP.

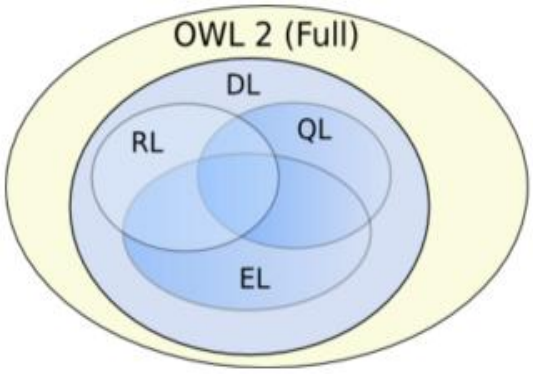



Figure 2. Venn Diagram of OWL2 sub-langages (profils) [11].

## IV. CONVERGENCE BETWEEN UML AND OWL

UML and OWL comprise some components which are similar in several regards, like: classes, associations, properties, packages, types, generalization and instances [5]. Whereas UML allows the modeling of the dynamic behavior of a system, OWL does not allow, on the other hand OWL is capable to inferred generalization and specialization between classes as well as individuals of a class based on the constraints imposed on the properties in the definition of the classes although UML does not allow that.

## V. SIMILAR WORK

### A. *UML2OWL* [7]

ODM (Ontology Metamodel Definition) offers a set of metamodels and mappings for bridging the metamodeling world and the ontologies.

This work presents an implementation of ODM by using ATL language, it supports the UML 2.0 meta-model and the OWL meta-model which are defined in the ODM, figure 3 illustrates the complete scenario of the transformation.

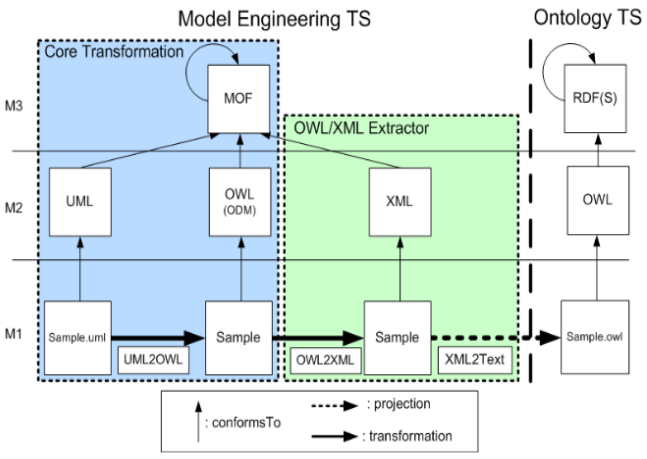



Figure 3. The complet scenario of the transformation [SID 03].

However the fact that the result is in OWL Full poses some problems, and does not reflect really what is modelled in UML.

### B. *OWLfromUML* [4]

This work consists to use a stylesheet *"OWLfromUML.xsl"* applied to a file XMI, this last is exported from an UML modeling tool (Rational Rose, Poseidon, ArguUML…). An OWL DL ontology represented in RDF/XML results from this transformation.

This work provides a simple, flexible and direct transformation, but the problem is in the transformation rules which for several of them do not give an adequate transformation (for example the association-classes, roles… etc).

## VI. OVERVIEW OF OUR SOLUTION

Our solution is implemented in the Eclipse platform, by using Topcased and XSLT.

Our choice is quickly related to Eclipse because it offers the services and the characteristics that we need.

We used Topcased because he is considered one of the best free solutions of modeling which provides a serialization of the models.

We used XSLT because it is simple and to guarantee a more adequate transformation.

For the ontology, the choice among profiles OWL is made on OWL DLL because it place certain constraints on the use of the structures of language OWL such as separation two has two between the classes, the types of data, the properties of the types of data, the properties of objects, the properties of annotation, the properties of ontologies, the individuals, the values of data, and the integrated vocabulary. That means, for example, that class cannot be at the same time an individual [10]. These constraints enable us to lead to our objective which is an ontology reflects well what modelled by a diagram.

For ontology, the choice among OWL profiles is made on OWL DL because it places certain con-straints on the use of the structures of OWL such as separation two to two between classes, data types, datatypes properties, objects properties, annotation properties, ontologies properties, individuals, data values, and integrated vocabulary [9]. That means, for example, that a class cannot be at the same time an individual [9]. These constraints enable us to lead to our objective which is an ontology well reflecting what is modeled in a class diagram.

### A. *Topcased*

#### 1) *Presentation*

Topcased (Toolkit in Open source for Critical Applications and SystEms Development) is an OpenSource tool making it possible to develop and maintain effectively complex systems. It privileges models engineering, that the importance is now established, and which stipulates that the development process must be centered on data-processing models, systems, software, or materials to be produced.

The tool thus comprises a whole of modeling frameworks in the standard distribution, each one being dedicated to a specific language (UML, AADL, SysML, SA, EAST-ADL…). But Topcased innovates while making it possible to create new framworks for new languages in a semi-automatic way (technique of meta-modelisation), in addition including the automatic production of a standardized format file (XML) offering of this fact the easy settling of internal or external interfaces with other tools [2].

#### 2) *Editor UML*

To help the users to create UML models through diagrams in conformity with UML specification of OMG. UML Editor provides the following diagrams: class diagram, components diagram, diagram, deployments diagram, composite structure diagram, use case diagram, activity diagram, state-transition diagram, sequence diagram. Figure 4 represents UML editor of Topcased.

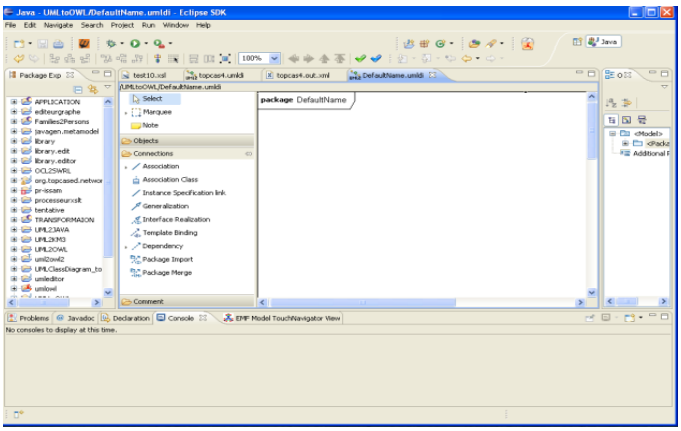


Figure 4. UML editor of Topcased .

The format of the Topcased models is based on plug-in Eclipse UML2 which is compatible with XMI. That means that each compatible XMI file exported from a specialized UML tool can be read by the editor Eclipse UML2 (available in Topcased).

Topcased separates the graphic data and data of a model, it stores the data of model in a file with the extension (.uml), although it stores the graphic data in a different file with the extension (.umldi).

Figure 5 shows a file (.umldi), while figure 6 illustrates the file (.uml) corresponding to it

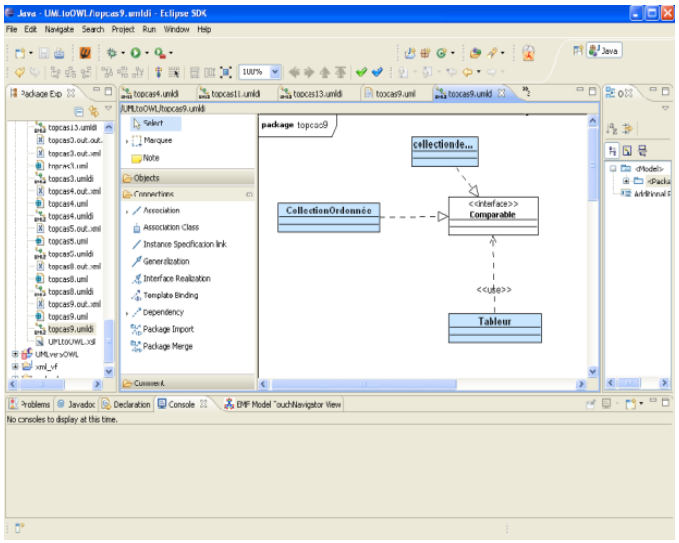



Figure 5. The file with extention (.umldi).

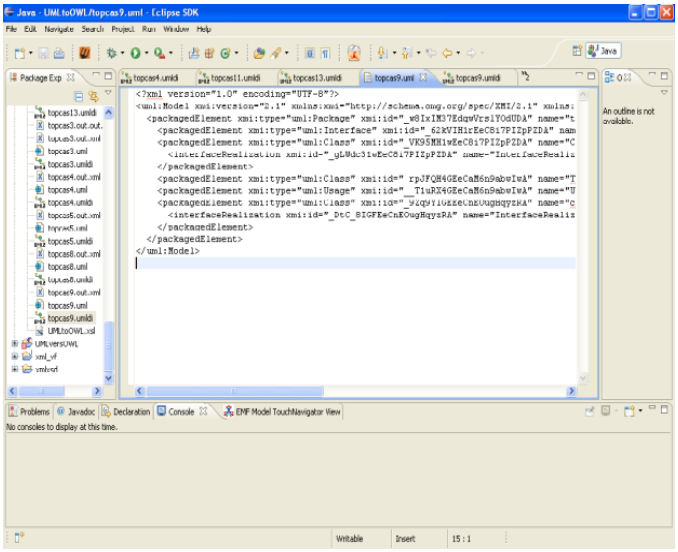


Figure 6. The file with extention (.uml).

This separation is benefic for us, because we want worked only on the data, i.e. what is interest us is the contained information in the model not the representation. In more, it allows a simple, flexible, and clear handling of the data contained in the model such as classes, associations… etc

### *B. XSLT*

XSLT (extensible Stylesheet Language for Transformations) is a language allows the transformation of XML documents. The transformations consist either of a modification of a XML with safeguard of the modifications, or in a transformation into another type of document (HTML, pdf, for example).

In our case we use XSLT for the transformation of the (.uml) file of an UML diagram (we choose the class diagram) described in the UML editor of Topcased towards an OWL ontology represented into RDF/XML.

To deal with XML we use JAXP (API Java for XML Processing) the standard API Java for the handling of XML format. This API is available under Eclipse, and it allows the reading, the transformation and the writing of files or XML flow.

The XSLT API is a component of JAXP in addition to the SAX, DOM API and it is sufficient to complete our work.

Figure 7 shows a piece of our stylesheet (in Eclipse) which generates the heading of ontology.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
      xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:owl="http://www.w3.org/2002/07/owl#"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

    <xsl:output method="xml" encoding="UTF-8"/>

    <xsl:template match="/">

<!-- ////////////////////////////////////////////////////////////////////////////////////// -->
    <rdf:RDF

            xmlns ="http://uml.vers.owl#"
            xml:base ="http://uml.vers.owl#"
            xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
            xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML"
            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#" >

<!-- ////////////////////// FIN ENTETE ONTOLOGIE ////////////////////////////////////////////// -->
```

Figure 7. The Stylesheet XSL developed.

## VII. TRANSFORMATION SENARIO

This transformation is done by an automatic generation of an ontology OWL represented in RDF/XML from an UML class diagram describes in Topcased, this solution is implemented into Eclipse, It proceeds in several steps (figure below):

*1) Graphic Description of the class diagram in Topcased by using the file (nom_diagramme.umldi).*

*2) Save the class diagram: it is automatically stored in two files (nom_diagramme.umldi) and (nom_diagramme.uml).*

*3) By using a processor XSLT (JAXP) and a stylesheet XSL (UMLtoOWL.xsl), apply the latter to the file (nom_diagramme.uml) to transform it into an OWL document represented in RDF/XML.*

*4) Visualization of OWL ontology by using special tools (Protégé, Swoop…).*

*5) Use of OWL ontology.*

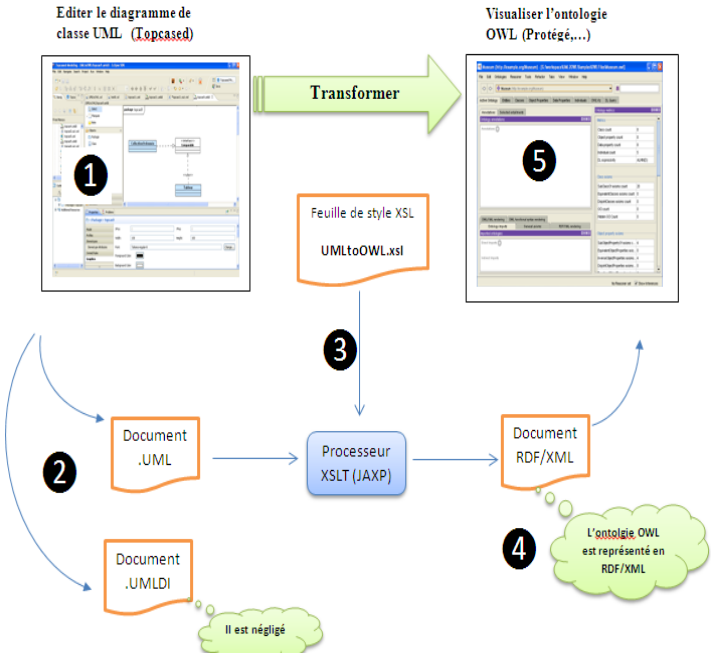
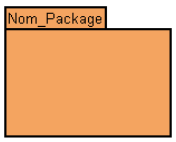


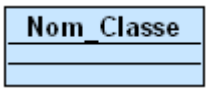
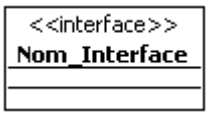
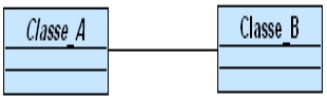

Figure 8. The transformation scenario.

## VIII. RULES OF TRANSFORMATION

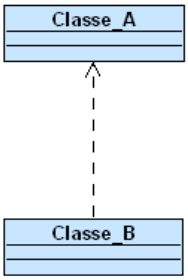
Our approach is realized according to suggested transformation rules (Table 1). We propose a set of rules in particular for classes' transformation, enumerations, associations, roles, dependences, association-classes, and almost all the elements of a class diagram. The level of abstraction in those rules is close to the application in order to have usable ontologies. For lack of space, we have not presented all the rules, we chose between them some rules.

TABLE I. TRANSFORMATION RULES.

| **UML to OWL** | |
|---|---|
| **Package** | |
|  | Direct transformation, the name of the package becomes the name of ontology. |
| <owl:Ontology rdf:about="**Nom_Package**"> | |

| **Class** | |
|---|---|
|  | An UML class is transformed into an OWL class, the name of the class is preserved. |
| <owl:Class rdf:ID="**Nom_Classe**"/> | |
| **Interface** | |
|  | We can transform an interface into an OWL class named (**Interface-nom_interface**). The class which implements the interface is defined as sub-class of the interface. The class which uses the interface is defined as sub-class of the interface. |
| <owl:Class rdf:ID="**Interface-Nom_Interface**/"> | |
| **Bidirectionnel Association** | |
|  | Associations are transformed into object properties. An inverse object property is generated automatically named (**Inverse - nomassociation**) |
| <owl:ObjectProperty rdf:ID="**Nom_Association**"><br><rdfs:domain rdf:resource="#**Classe_A**"/><br><rdfs:range rdf:resource="#**Classe_B**"/><br></owl:ObjectProperty><br><owl:ObjectProperty rdf:ID="**Inverse- Nom_Association**"><br><owl:inverseOf rdf:resource= "#**Nom_Association**"/><br></owl:ObjectProperty> | |

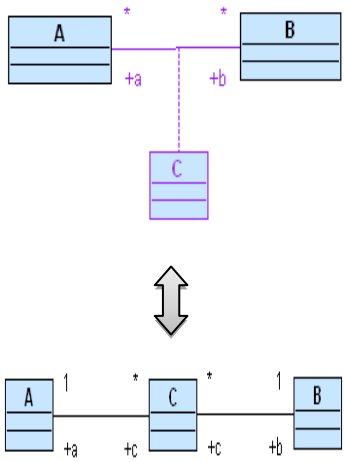
## Dependence




The dependences are transformed into object properties with the name (DependencyN).
The domain and the range are given according to the direction of association.

```
<owl:ObjectProperty rdf:ID="DependencyN">
<rdfs:domain rdf:resource="#Classe_B"/>
<rdfs:range rdf:resource="#Classe_A"/>
</owl:ObjectProperty>
```

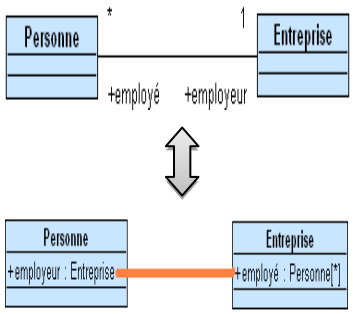
## Association-Class




An association-class is transformed to OWL class (implementation level), named (ac-ssociationclassname). The latter is connected to the left part by a relation named (AG_AC-associationclassname), and to the right part by a relation named (AD_AC-associationclassname). We named also the two new roles on the two new association ends (RG_AC-associationclassname) and (RD_AC-associationclassname). After these transformations on the association class we find ourselves on the situation of transformation of binary associations (which is treated previously).

```
owl:Class rdf:ID="AC-C"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#A.AC-C"/>
<owl:mincardinality
rdf:datatype=="http://www.w3.org/2001/XMLSchema#
nonNegativeInteger">0 </owl:mincardinality>
</owl:Restriction>
</rdfs:subClassOf> …
<owl:FunctionalProperty rdf:ID="AC-C.A">
<rdfs:domain rdf:ressource="#AC-C"/>
<rdfs:range rdf:ressource="#A"/>
</owl:FunctionalProperty> …
```

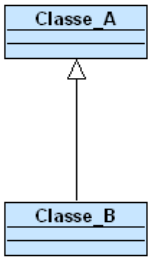
## Roles




Roles transformation is based on the representation by attributes (implementation level). Thus the situation of transformation of attributes.

```
<owl:ObjectProperty  rdf:ID="Personne.employeur">
<rdfs:subPropertyOf rdf:resource=" #travailler" />
<rdfs:domain rdf:ressource="#Personne"/>
<rdfs:range rdf:ressource="#Entreprise"/>
<owl:ObjectProperty />…
<owl:Classrdf:ID="Personne">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onPropertyrdf:resource= "#Personne.employeur"/>…
```

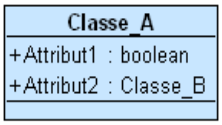
## Héritage




La classe spécialisée est définie comme sous-classe de la classe généralisée.

```
<owl:Classrdf:ID="Classe_B"/>
<rdfs:subClassOfrdf:resource="#Classe_A" />
</owl:Class>
```

## Attribut




Un attribut est transformé en une propriété, et la transformation s'effectue suivant le type de l'attribut.
Si le type de l'attribut est un type primitif, l'attribut est transformé en propriété de type de données. Si La valeur de l'attribut est une classe, il est transformé en propriété d'objet.

```
<owl:Classrdf:ID="Classe_B"/>
<rdfs:subClassOfrdf:resource="#Classe_A" />
</owl:Class>
```

UML datatypes are transformed into XML schema (XSD) datatypes because OWL uses the majority of the datatypes integrated into XML schema. The calls of these datatypes are done through datatype URI address reference http://www.w3.org/2001/XMLSchema [9]. The in-stances of the primitive types used in UML itself in-clude: Boolean, Integer, String, and UnlimitedNatural [7]. Count 2 presents the UML primitive datatypes and theirs transformations.

We present in the table2 some datatypes and their transformations.

TABLE II. DATATYPES TRANSFORMATION.

| UML | XSD |
|---|---|
| Boolean | xsd:boolean |
| Float | xsd:float |
| Integer | xsd:integer |
| Char, Character, String | xsd:string |
| Short | xsd:short |
| Long | xsd:long |
| Decimal | xsd:decimal |
| Date | xsd:date |

## IX. CONCLUSION

We saw in this article how to implement an application which makes a transformation from a class diagram towards an OWL ontology, the process is based on transformation rules which are proposed in order to maintain all what modeled in an UML model in an OWL ontology.

To implement this application we use a stylesheet XSL written for allows the transformation of a document (.uml) towards a document OWL represented by (RDF/XML) format, and the UML editor of Topcased based on Eclipse is used to describe an UML diagram to generate a file (.uml), and the JAXP processor XSLT makes it possible to treat the stylesheet written to make the transformation easily.

## REFERENCES


[1] Laurent AUDIBERT, “UML2”, http://www.lipn.univparis13.fr /audibert/pages/enseignemen t/cours.htm, 2007.

[2] P. Farail, P. Gaufillet, “TOPCASED Un environnement de développement Open Source pour les systèmesembarqués”, Airbus France, 2005.

[3] Fowler, Martin,“UML Distilled - Third Edition - A Brief Guide to the Standard Object Modeling Language”, 2003.

[4] Sebastian Leinhos, http://diplom.ooyoo.de, 2006.

[5] OMG, “Ontology Definition Metamodel”, V1.0, http://www.omg.org/spec/ODM/1.0, May 2009.

[6] OMG, “OMG Unified Modeling Language,Infrastructure,v2.3”, http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF, May 2010.

[7] SIDo Group, “ATL Use Case - ODM Implementation (Bridging UML and OWL)”, http://www.eclipse.org/m2m/atl/usecases/ ODMImplementation/, 2007.

[8] Deborah L. McGuinness et Frank van Harmelen, “OWL Web Ontology Language-Overview”, http://www.w3.org/TR/2004/REC-owl-features-20040210/. Recommandation du W3C du 10 février 2004, Traduction française : Vue d'ensemble du langage d'ontologie Web OWL, http://www.yoyodesign.org/doc/w3c/owl-features-20040210/.

[9] Michael K. Smith, Chris Welty et Deborah L. McGuinness, “OWL Web Ontology Language–Guide”, http://www.w3.org/TR/2004/REC-owl-guide-20040210/.Recommandation du W3C du 10 février 2004, Traduction française : Le langage d'ontologie Web OWL - Guide, http://www.yoyodesign.org/doc/w3c/owl-guide-20040210/.

[10] Mike Dean, Guus Schreiber, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider et Lynn Andrea Stein, “OWL Web Ontology Language-Reference”, http://www.w3.org/TR/2004/REC-owl-ref-20040210/ (Recommandation du W3C du 10 février 2004). Traduction française : La référence du langage d'ontologie Web OWL, http://www.yoyodesign.org/doc/w3c/owl-ref-20040210/.

[11] W3C OWL Working Group, “OWL 2 Web Ontology Language Document Overview”. http://www.w3.org/TR/2009/REC-owl2-overview-20091027/(W3C Recommendation 27 October 2009).