

An Approach based AToM3 for the Generation of OWL Ontologies from UML Diagrams

Aissam Belghiat
Computing Department
University of Md Boudiaf M'sila, 28000, Algeria

Mustapha Bourahla
Computing Department
University of Md Boudiaf M'sila, 28000, Algeria

ABSTRACT

The models are placed by modeling paradigm at the center of development process. These models are represented by languages, like UML the language standardized by the OMG which became necessary for development. Moreover the ontology engineering paradigm places ontologies at the center of development process, in this paradigm we find OWL the explicitation language adopted by a great community of users like the principal language of knowledge representation. The bringing between UML and OWL appeared on several regards such as the classes and associations. In this paper, we propose an approach based graph transformation and registered in the MDA architecture for the automatic generation of usable OWL ontology from UML class diagrams. The transformation is based on transformation rules make it possible to achieve our aim. This approach is illustrated by an example.

General Terms

Ontology engineering, software engineering, model transformation.

Keywords

UML, Ontologie, OWL, ATOM3, MDA.

1. INTRODUCTION

UML is unified object oriented modeling language which becomes an important standard. In the other side, the ontologies became the backbone of the se-mantic web which described formally using description logics implemented in a standard language called OWL. In this work we propose a set of rules for trans-forming classes diagram into ontologies described in OWL language in the order to profit from the power of ontologies so that the information described by those diagrams can be shared and linked with other information and we could start dealing with the over-laps, gaps, and integration barriers between modeling languages and get greater value out of the information capture in them. These rules will be implemented in a software to automate this transformation. Thus, we benefit from UML in order to have models on ontologies to make preliminary analyzes and OWL implementations to test ontologies consistencies.

The rest of the paper is organized as follows: In setion 2, we present some related works. In section 3, we recall some basic notion about UML, OWL, and their bridging. In section 4, we recall some concepts about model transformation and graph transformation and give an overview of the AToM3 tool [1]. In section 5, we describe our approach that transforms UML class

diagrams models to OWL ontologies models. In section 6, we illustrate our generated tool through an example. Finally concluding remarks are drawn from the work and perspectives for further research are presented in section 7.

2. RELATED WORK

The idea of our work is not innovating, indeed several work exist in the literature tackle this subject. In [2] the authors proposed a transformation of UML towards DAML at the end of the Nineties, by showing similarities and differences between the two languages. In [3] the work of “Converting UML to OWL Ontologies” proposed a transformation of Ontology UML Profile (OUP) towards an ontology OWL. In [4] the OMG notices the interest of such subject and proposed in its turn the ODM which provides a profile for writing RDF and OWL within UML, it also includes partial mappings between UML and OWL as well as mappings amongst RDF, RDFS, Common Logic and Topic Maps, it should be noted that several work are carried out like answer to the call of the OMG and gathered in the ODM and thus we do not evoke here.

In [5] the author presented an implementation of the ODM using ATL language. In [6] the author used a style sheet “OWLfromUML.xsl” applied to a file XMI (UML model) to generate an ontology OWL DL represented in RDF/XML. And finally in [7] the authors proposed a detailed comparison between UML and OWL carried out in 2008. In the other side Atom3 has been proven to be a very powerful tool allowing the meta-modeling and the transformations between formalisms, in [1] we can found treatment of class diagrams, activity , and other diagrams UML, in these works the meta-modelisation allows visual modeling and the graph grammars allows the transformation.

Obviously, the heart of our work is articulated on transformation rules and theirs implementation. In preceding work, the transformation rules are more specific and reflect a general opinion of the author often related to a specific field which he works on him (transformation on measure). In this paper we propose another vision different from that approached in preceding works neither in the proposition of transformation rules, or in theirs implementation, this vision is to propose the transformation during the implementation of class diagrams, in order to obtain a usable ontology, because more the selected level of abstraction is close to the application minus ontology is reusable, but more it is usable.

3. BRIDGING UML AND OWL

UML (Unified Modeling Language) is a language to visualize, specify, build and document all the aspects and artifacts of a software system [8]. UML defines thirteen diagrams; some of them represent the system statically while others show the functionalism of system. 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 [9].

OWL (Ontology Web Language), was recommended by the W3C in 2004, and its version 2 in 2009, it is designed for use by applications that need to process the content of information instead of just presenting information to humans. It allows an interpretation of the Web contents by the machines higher than that offered by the languages XML, RDF and diagram RDF, by providing an additional vocabulary with a formal semantics. OWL1 offers three sublanguages with increasing expression intended for specific communities of developers and users: OWL Lite, OWL DL, and OWL Full [10] whereas OWL2 defines three new profiles: OWL2 EL, OWL2 QL, OWL2 RL [11].

UML and OWL comprise some components which are similar in several regards, like: classes, associations, properties, packages, types, generalization and instances [4]. UML and OWL have different goals and approaches; however they have some overlaps and similarities, especially for representation of structure (class diagrams). UML is a notation for modeling the artifacts of objects oriented software, whereas OWL is a notation for knowledge representation, but both are modeling languages.

4. MODEL TRANSFORMATION

4.1 Overview

Modeling and model transformation play an essential role in the MDA “Model Driven Architecture”. MDA recommends the massive use of models in order to allow a flexible and iterative development, thanks to refinements and enrichments by successive transformations.

A model transformation is a set of rules that allows passing from a meta-model to another, by defines for each one of elements of the source their equivalents among the elements of the target. These rules are carried out by a transformation engine; this last read the source model which must be conform to the source meta-model, and apply the rules defined in the model transformation to lead to the target model which will be itself conforms to the target meta-model. The principle of model transformation is illustrated by fig. 1:

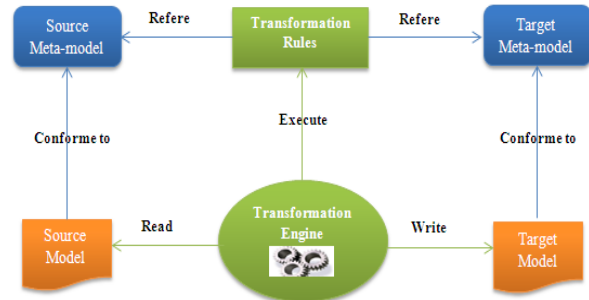


Fig 1: Model transformation principle.

4.2 Graph Transformation

Graph transformation was largely used for the expression of model transformation [12]. Particularly transformations of visual models can be naturally formulated by graph transformation, since the graphs are well adapted to describe the fundamental structures of models.

The set of graph transformation rules constitutes what is called the model of graph grammar. A graph grammar is a generalization, for graphs, of Chomsky grammars. Each rule of a graph grammar is composed of a graph of left side (LHS) and of a graph of right-sided (RHS).

Therefore, the graph transformation is the process to choose a rule among the graph grammar rules, apply this rule on a graph and reiterate the process until no rule can be applied [12].

4.3 ATOM3

ATOM3 [1] “A Tool for Multi-formalism and Meta-Modeling” is a visual tool for model transformation, written in Python [13] and is carried out on various platforms (Windows, Linux,...). It implements various concepts like multi-paradigme modeling, meta-modelisation and graph grammars. It can be also used for simulation and code generation.

ATOM3 provides visual models those are conform to a specific formalism, and uses the grammar of graph to go from a model to another.

In the next sections, we will discuss how we use ATOM3 to meta-model class diagrams and how to generate OWL models by Applying a graph grammar.

5. OUR APPROACH

5.1 Overview

Our solution is implemented in ATOM3. Our choice is quickly related to ATOM3 because of the advantages which it presents like its simplicity, and its availability.

For the realization of this application we have to propose and to develop a meta-model of class diagram, this meta-model allows us to edit visually and with simplicity class diagrams on the canvas. In addition to meta-model proposed we develop a graph grammar made up of several rules which allows transforming progressively all what is modeled on the canvas towards an OWL ontology stored in a disk file. The graph grammar is based on transformation rules; those rules try to transform the class diagram in the implementation level, always in order to obtain in final a usable ontology.

For ontology, the choice among OWL profiles is made on OWL DL because it places certain constraints 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 [14]. That means, for example, that a class cannot be at the same time an individual [15]. These constraints enable us to lead to our objective which is an ontology well reflecting what is modeled in a class diagram.

5.2 Transformation scenario

The transformation proceeds in several steps (fig. 2):

1. Graphic description of class diagram in ATOM3.
2. This class diagram is conform to the meta-model of class diagram developed in ATOM3.
3. Apply the graph grammar (ClassDiagram2OWLontology) on the class diagram.
4. A file (owlcode.owl) is generated automatically contains the OWL ontology represented in RDF/XML, it is stored on disc.
5. Visualization and use of OWL ontology by using special tools (Protected, Swoop...).

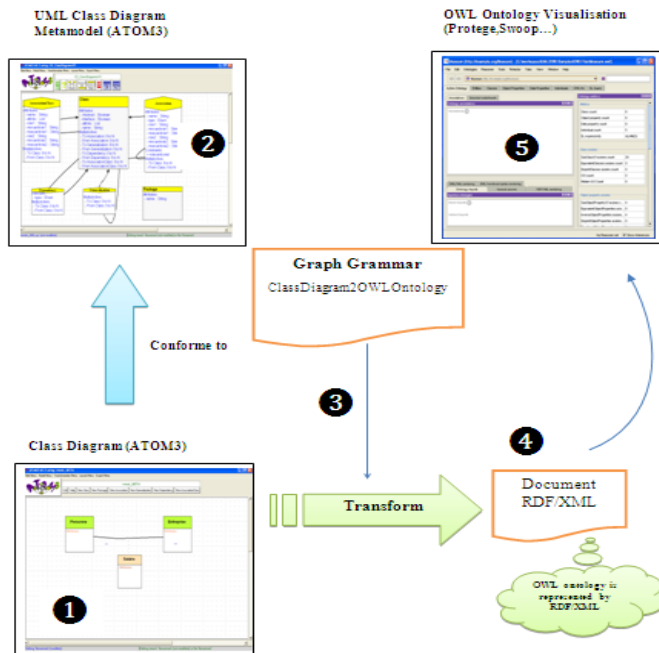


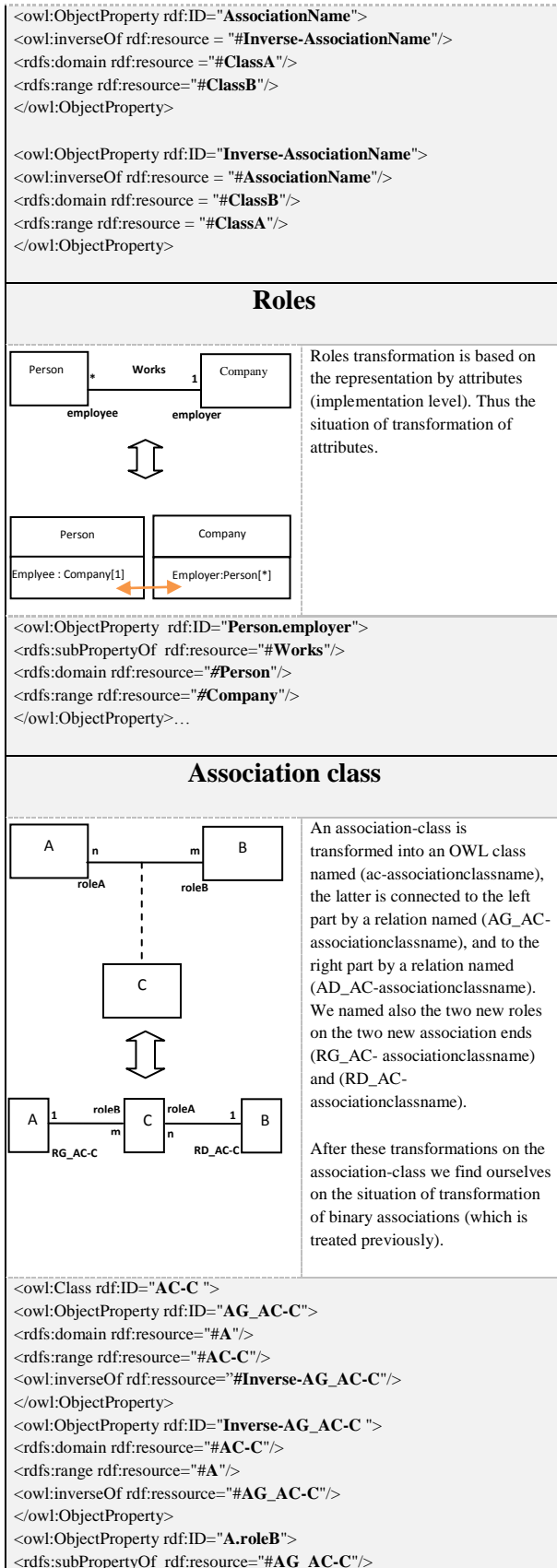
Fig 2: Transformation scenario.

5.3 Transformation rules

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 1. Transformation rules.

UML to OWL	
Class	
<div>ClassName</div>	<p>An UML class is transformed to an OWL class; the name of the class is preserved.</p> <pre><owl:Class rdf:ID="ClassName"/></pre>
Heritage	
<div>ClassA</div> <div>ClassB</div>	<p>The specialized class is defined subclass of the generalized class.</p> <pre><owl:Class rdf:ID="ClassB"/> <rdfs:subClassOf rdf:resource="#ClassA" /> </owl:Class></pre>
Attribute	
<div>ClassA</div> <div>Attribute1 : Boolean</div> <div>Attribute2 : ClassB</div>	<p>An attribute is transformed into a property, and the transformation is carried out according to the type of attribute.</p> <p>If the type of the attribute is a primitive type, the attribute is transformed into datatype property. If the value of the attribute is a class, it is transformed into object property.</p> <pre><owl:DatatypeProperty rdf:ID="ClassA.Attribute1"> <rdfs:domain rdf:resource="#ClassA" /> <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema# boolean"/> </owl:DatatypeProperty></pre> <p>OR</p> <pre><owl:ObjectProperty rdf:ID="ClassA.Attribute2"> <rdfs:domain rdf:resource="#ClassA"/> <rdfs:range rdf:resource="#ClassB"/> </owl:ObjectProperty></pre>
Bidirectionnel association	
<div>ClassA</div> <div>AssociationName</div> <div>ClassB</div>	<p>Associations are transformed into object properties. An inverse object property is generated automatically named (Inverse-AssociationName)</p>



```

<rdfs:domain rdf:resource="#A"/>
<rdfs:range rdf:resource="#AC-C"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="AC-C.RG_AC-C">
<rdfs:subPropertyOf rdf:resource="#Inverse-AG_AC-C"/>
<rdfs:domain rdf:resource="#AC-C"/>
<rdfs:range rdf:resource="#A"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="A">
<rdfs:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#A.roleB"/>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#
nonNegativeInteger">m</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="AC-C">
<rdfs:subClassOf><owl:Restriction>
<owl:onProperty rdf:resource="#AC-C.RG_AC-C"/>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#
nonNegativeInteger">1</owl:cardinality>
</owl:Restriction></rdfs:subClassOf>
</owl:Class>

```

... After that we proceed to transform the right part of association-class...

5.4 Datatypes transformation

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> [14]. The instances of the primitive types used in UML itself include: Boolean, Integer, String, and UnlimitedNatural [8]. Count 2 presents the UML primitive datatypes and theirs transformations.

Table 2. Datatypes Transformation.

UML	XSD
Integer	xsd:integer
Boolean	xsd:boolean
String	xsd:string
UnlimitedNatural	xsd:nonNegativeInteger xsd:positiveInteger

5.5 UML Class diagram Meta-model

To build UML class diagrams models in ATOM3, we have to define a meta-model for them. Our meta-model is composed of 2 classes and 4 associations developed by the meta-formalism (CD_classDiagramsV3), and the constraints are expressed in Python [Python] code (fig.3):

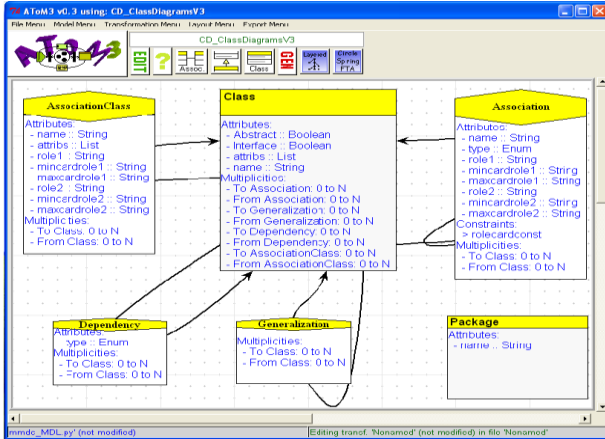


Fig 3: Class diagram meta-model.

After we build our meta-model, there remains only its generation. The generated meta-model comprises the set of classes modeled in the form of buttons which are ready to employ for a possible modeling of a class diagram.

5.6 The Graph grammar proposed

To perform the transformation between class diagrams and OWL ontologies, we have proposed a graph grammar called (ClassDiagram2OWLOntology) composed of an initial action, ten rules, and a final action. For lack of space, we have not presented all the rules.

Initial Action: Ontology header.

Role: In the initial action of the graph grammar, we created a file named (owlcode) with sequential access in order to store generated OWL code. In Python, access to files is ensured via a file-object called "obFichier". In our case, it was created using the internal function 'open ()'. After having called this function, we can write in the file using 'Write ()' then to close it again by 'close ()'. We start to write the ontology header which is fixed for all our generated ontologies (fig. 4).

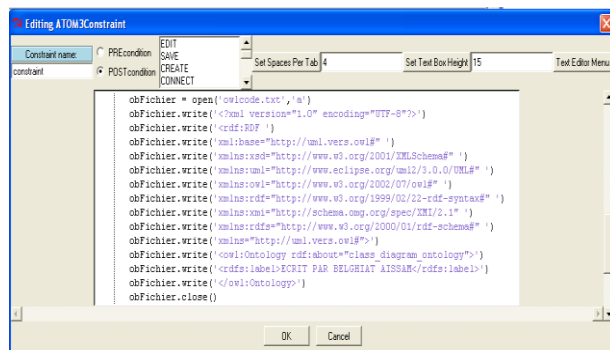


Fig 4: Ontology header definition.

Rule 1 : Class transformation

Name : class2class

Priority : 2

Role : This rule transforms an UML class towards an OWL class (cf. Table 3). In the condition of the rule we test if the class

is already transformed, if not, in the action of the rule we open the file 'owlcode' and adds the class in OWL code.

Table 3. Class transformation.

Condition	
<pre>node = self.getMatched(graphID, self.LHS.nodeWithLabel(1)) return not hasattr(node, "rule executed")</pre>	
<pre>node = self.getMatched(graphID, self.LHS.nodeWithLabel(1)) classname = node.name.getValue() node.rule_executed = True abst = node.Abstract.getValue()[1] interf = node.Interface.getValue()[1] if abst == 1: self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.setValue('Abstract-'+classname) elif interf == 1: self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.setValue('Interface-'+classname) obFichier = open('owlcode.txt','a') node = self.getMatched(graphID, self.LHS.nodeWithLabel(1)) classname = node.name.getValue() obFichier.write('<owl:Class rdf:ID="'+classname+'"/>') obFichier.close()</pre>	

Rule 2 : Generalization transformation

Name : gener2sclass

Priority : 1

Role : This rule transforms a generalization by indicating that the specialized class is defined as subclass of the generalized class (Table 4).

Table 4. Generalization transformation.

Condition	
<pre>node = self.getMatched(graphID, self.LHS.nodeWithLabel(3)) gen = self.graphRewritingSystem.parent.ASGroot.listNodes['Generalization'] return gen != [] and not hasattr(node, "rule4 executed")</pre>	
<pre>node = self.getMatched(graphID, self.LHS.nodeWithLabel(3)) node.rule4_executed = True node = self.getMatched(graphID, self.LHS.nodeWithLabel(1)) classname = node.name.getValue() node = self.getMatched(graphID, self.LHS.nodeWithLabel(2)) class2name = node.name.getValue() obFichier = open('owlcode.txt','a') obFichier.write('<owl:Class rdf:ID="'+classname+'"/>') obFichier.write('<rdof:subClassOf rdf:resource="'+class2name+'"/>') obFichier.write('</owl:Class>') obFichier.close()</pre>	


Rule 3: Attributes transformation

Name : attrib2prop

Priority: 2

Role : This rule transforms an attribute of a class of the class diagram towards an OWL object property or an OWL datatype property according to the type of the attribute (cf. Table 5).

Table 5. Attribute transformation.

Condition
<pre>node = self.getMatched(graphID, self.LHS.nodeWithLabel(1)) return not hasattr(node, "rule6_executed")</pre>

Action
<pre>node = self.getMatched(graphID, self.LHS.nodeWithLabel(1)) classname = node.name.getValue() enum = node.Enumeration.getValue()[1] node.rule6_executed = True # Si c'est une enumeration listattrib = node.attrs.getValue() if enum == 1: obFichier = open('owlcode.txt', 'a') obFichier.write('<owl:Class rdf:about="#" + classname + ">') obFichier.write('<owl:oneOf rdf:parseType="Collection">') for t in listattrib: elm = t.toString() elmList = elm.split() debut = elmList[0] final = elmList[1] final = final[5:] obFichier.write('<owl:Thing rdf:about="#" + debut + ">') obFichier.write('</owl:oneOf>') obFichier.write('</owl:Class>') else: # Si c'est une classe avec des attributs listattrib = node.attrs.getValue() for e in listattrib: elm = e.toString() elmList = elm.split() debut = elmList[0] final = elmList[1] final = final[5:] init = elmList[2] init = init[11:] obFichier = open('owlcode.txt', 'a') #si le type de l'attribut est une autre classe if final == "Text": obFichier.write('<owl:ObjectProperty') obFichier.write('<owl:Class rdf:about="#" + classname + ">') obFichier.write('<rdfs:domain rdf:resource="#" + classname + ">') obFichier.write('<rdfs:range rdf:resource="#" + init + ">') obFichier.write('</owl:ObjectProperty>') #si le type de l'attribut est un type primitif else: obFichier.write('<owl:DatatypeProperty') obFichier.write('<owl:Class rdf:about="#" + classname + ">') obFichier.write('<rdfs:domain rdf:resource="#" + classname + ">') if final == "String": obFichier.write('<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>') elif final == "Integer": obFichier.write('<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>') elif final == "Boolean": obFichier.write('<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>') elif final == "Float": obFichier.write('<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>') else: obFichier.write('<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>') obFichier.write('</owl:DatatypeProperty>') obFichier.close()</pre>

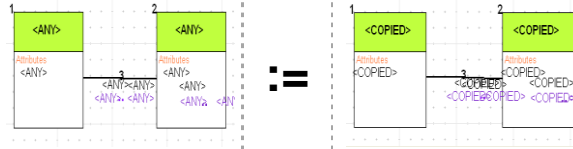
Rule 4 : Binary association transformation

Name : asso2prop

Priority : 2

Role : This rule transform an association of the class diagram towards an OWL object property, it allows also the transformation of the roles and cardinalities of this association (cf Table 6).

Table 6. Association transformation.

Condition
<pre>node = self.getMatched(graphID, self.LHS.nodeWithLabel(3)) return not hasattr(node, "rule4_executed")</pre>

Action
<pre>node = self.getMatched(graphID, self.LHS.nodeWithLabel(3)) assoname = node.name.getValue() typ = node.type.getValue()[1] node.rule4_executed = True role1name = node.role1.getValue() role2name = node.role2.getValue() role1min = node.mincardrole1.getValue() role1max = node.maxcardrole1.getValue() role2min = node.mincardrole2.getValue() role2max = node.maxcardrole2.getValue() node = self.getMatched(graphID, self.LHS.nodeWithLabel(1)) classname = node.name.getValue() node = self.getMatched(graphID, self.LHS.nodeWithLabel(2)) class2name = node.name.getValue() ##### Transformation des associations ##### if typ == 0 or typ == 2: # association bidirectionnelle ou aggregation obFichier = open('owlcode.txt', 'a') obFichier.write('<owl:ObjectProperty rdf:ID="#" + assoname + ">') obFichier.write('<rdfs:domain rdf:resource="#" + classname + ">') obFichier.write('<rdfs:range rdf:resource="#" + class2name + ">') obFichier.write('<owl:inverseOf rdf:resource="#" + assoname + ">') obFichier.write('</owl:ObjectProperty>') obFichier.write('<owl:ObjectProperty rdf:ID="Inverse-' + assoname + ">') obFichier.write('<rdfs:domain rdf:resource="#" + class2name + ">') obFichier.write('<rdfs:range rdf:resource="#" + classname + ">') obFichier.write('<owl:inverseOf rdf:resource="#" + assoname + ">') obFichier.write('</owl:ObjectProperty>') obFichier.close() elif typ == 1 or typ == 3: # association unidirectionnelle ou composition</pre>

Rule 5 : Association-class transformation

Name : ac2class

Priority : 1

Role : This rule allows the promotion of association-class to a full class (cf Table 7), that reflects what we show in the transformation rules. This class takes as name the name of the LHS class-association preceded by (AC-). Two binary associations are created in the RHS named AG_AC, AD_AC, thus two new roles RG_AC, RD_AC as illustrated in the transformation rules.

Table 7. Association-class transformation.

Condition	
No condition	
Action	
No action	

Final Action : Definition of the ontology end

Role : In the final action of the graph grammar, we finish our ontology, So that becomes possible, we will have to open our file and to add “</rdf:RDF>” (cf fig. 5).

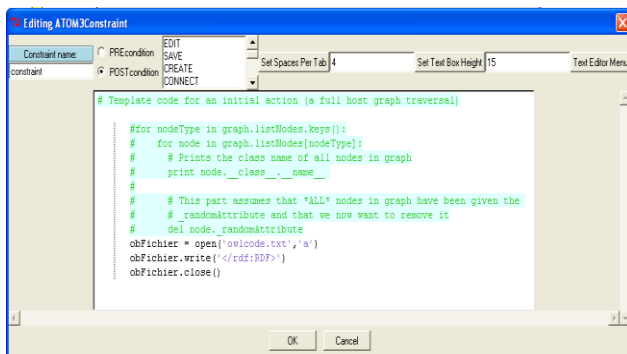


Fig 5: Ontology end definition.

6. EXAMPLE

Let us apply our approach on the example illustrated in figure 6, which models the situation that a person occupies a job in a company:

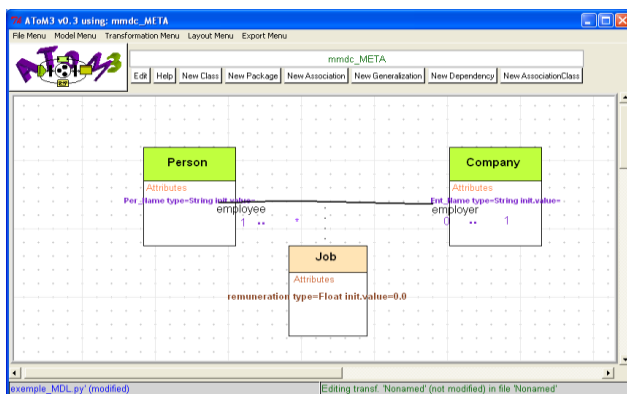


Fig 6: Example class diagram.

We start the execution of our graph grammar we obtain the following intermediate graph:

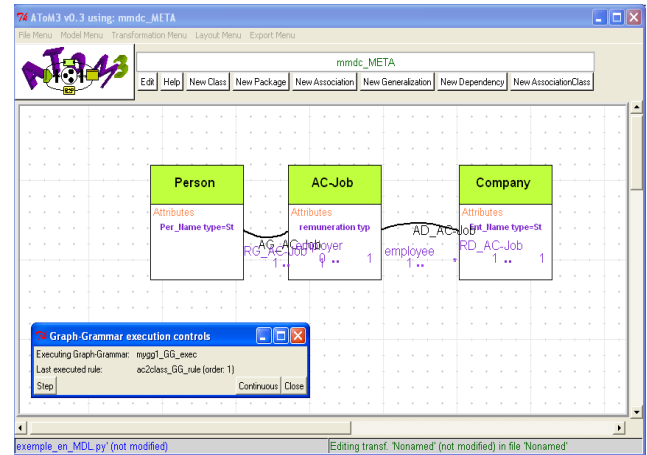


Fig 7: Intermediate graph.

After the execution of the graph grammar on our example we obtain the diagram illustrated in fig. 8:

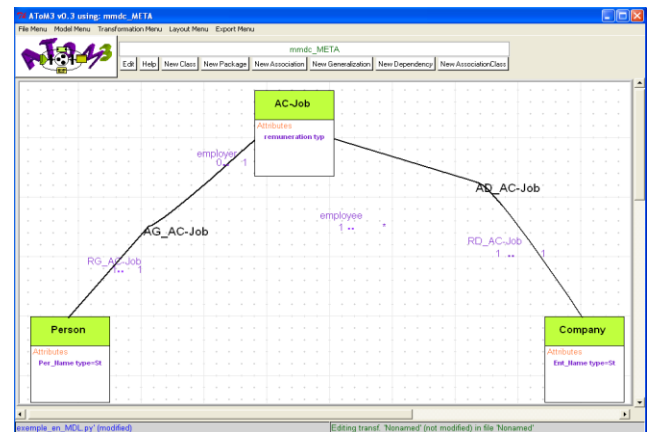


Fig 8: Class diagram after execution.

In parallel, there is an automatic generation of the file which contains OWL code stored on disc (fig. 9):

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xml:base="http://uml.vers.owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
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:xmi="http://schema.omg.org/spec/XMI/2.1"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns="http://uml.vers.owl#"><owl:Ontology
rdf:about="class_diagram_ontology"><rdfs:label>Written by BELGHIAI
AISSAM</rdfs:label> </owl:Ontology>
<owl:Class rdf:ID="Person"/>
<owl:Class rdf:ID="AC-Job"/>
<owl:Class rdf:ID="Company"/>
<owl:ObjectProperty rdf:ID="AG-AC-Job">
<rdfs:domain rdf:resource="#Person"/>
<rdfs:range rdf:resource="#AC-Job"/>
<owl:inverseOf rdf:resource="#Inverse-AG-AC-Job"/>
</owl:ObjectProperty>
```

```

<owl:ObjectProperty rdf:ID="Inverse-AG_AC-Job">
<rdfs:domain rdf:resource="#AC-Job"/>
<rdfs:range rdf:resource="#Person"/>
<owl:inverseOf rdf:resource="#AG_AC-Job"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Person.employer">
<rdfs:subPropertyOf rdf:resource="#AG_AC-Job"/>
<rdfs:domain rdf:resource="#Person"/>
<rdfs:range rdf:resource="#AC-Job"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="AC-Job.RG_AC-Job">
<rdfs:subPropertyOf rdf:resource="#Inverse-AG_AC-Job"/>
<rdfs:domain rdf:resource="#AC-Job"/>
<rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Person">
<rdfs:subClassOf><owl:Restriction> <owl:onProperty
rdf:resource="#Person.employer"/> <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0
<owl:minCardinality><owl:Restriction></owl:Restriction></owl:Class>
<owl:Class rdf:ID="AC-Job">
<rdfs:subClassOf><owl:Restriction> <owl:onProperty rdf:resource="#AC-
Job.RG_AC-Job"/><owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1
<owl:cardinality><owl:Restriction></owl:Restriction></owl:Class>
<owl:ObjectProperty rdf:ID="AD_AC-Job">
<rdfs:domain rdf:resource="#AC-Job"/>
<rdfs:range rdf:resource="#Company"/>
<owl:inverseOf rdf:resource="#Inverse-AD_AC-Job"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Inverse-AD_AC-Job">
<rdfs:domain rdf:resource="#Company"/>
<rdfs:range rdf:resource="#AC-Job"/>
<owl:inverseOf rdf:resource="#AD_AC-Job"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="AC-Job.RD_AC-Job">
<rdfs:subPropertyOf rdf:resource="#AD_AC-Job"/>
<rdfs:domain rdf:resource="#AC-Job"/>
<rdfs:range rdf:resource="#Company"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Company.employee">
<rdfs:subPropertyOf rdf:resource="#Inverse-AD_AC-Job"/>
<rdfs:domain rdf:resource="#Company"/>
<rdfs:range rdf:resource="#AC-Job"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="AC-Job">
<rdfs:subClassOf><owl:Restriction> <owl:onProperty rdf:resource="#AC-
Job.RD_AC-Job"/> <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1<owl:
cardinality><owl:Restriction></owl:Restriction></owl:Class>
<owl:Class rdf:ID="Company">
<rdfs:subClassOf><owl:Restriction> <owl:onProperty
rdf:resource="#Company.employee"/> <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1<owl:
minCardinality><owl:Restriction></owl:Restriction></owl:Class>
<owl:DatatypeProperty rdf:ID="Person.Per_Name">
<rdfs:domain rdf:resource="#Person"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="AC-Job.remuneration">
<rdfs:domain rdf:resource="#AC-Job"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Company.Ent_Name">
<rdfs:domain rdf:resource="#Company"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
</rdf:RDF>

```

Fig 9: Generated OWL ontology.

7. CONCLUSION

We saw in this paper how to implement an application which makes a transformation from a UML class diagram towards an OWL ontology based on graph transformation and by using

ATOM3 in order to obtain a usable ontology for needs in shared information. For the realization of this application we developed a meta-model for class diagrams, and a graph grammar composed of several rules which enables us to transform all what is modeled in our ATOM3 generated environment toward an OWL ontology stored in a disk file.

In future work, we will try to implement the transformation by choose the level of abstraction far to the application in order to have reusable ontologies.

8. REFERENCES

- [1] ATOM3. Home page: <http://atom3.cs.mcgill.ca.2002>.
- [2] Kenneth Baclawski², Mieczyslaw K. Kokar², Paul A. Kogut¹, Lewis Hart⁵, Jeffrey Smith³, William S. Holmes III¹, Jerzy Letkowski⁴, and Michael L. Aronson¹ "Extending UML to Support Ontology Engineering for the Semantic Web".
- [3] Dragan Gašević, Dragan Djurić, Vladan Devedžić, Violeta Damjanović "Converting UML to OWL Ontologies", 2004
- [4] OMG, "Ontology Definition Metamodel", V1.0, <http://www.omg.org/spec/ODM/1.0>, May 2009.
- [5] SDO Group, "ATL Use Case - ODM Implementation (Bridging UML and OWL)", <http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/>, 2007.
- [6] Sebastian Leinhos, <http://diplom.ooyoo.de>, 2006.
- [7] Kilian Kiko, Colin Atkinson, "A Detailed Comparison of UML and OWL", 2008.
- [8] OMG, "OMG Unified Modeling Language, Infrastructure, v2.3", <http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF>, May 2010.
- [9] Fowler, Martin, "UML Distilled - Third Edition - A Brief Guide to the Standard Object Modeling Language", 2003.
- [10] Deborah L. McGuinness et Frank van Harmelen, "OWL Web Ontology Language-Overview", <http://www.w3.org/TR/2004/REC-owl-features-20040210/>. (W3C Recommendation 10 February 2004).
- [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).
- [12] G. Karsai, A. Agrawal, "Graph Transformations in OMG's Model-Driven Architecture", Lecture Notes in Computer Science, Vol 3062, 243-259, Springer Berlin /Heidelberg, juillet 2004.
- [13] Python. Home page: <http://www.python.org>.
- [14] Michael K. Smith, Chris Welty et Deborah L. McGuinness, "OWL Web Ontology Language-Guide", <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>. (W3C Recommendation 10 February 2004).
- [15] 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/> (W3C Recommendation 10 February 2004).