World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:6, No:8, 2012

Automatic Generation of OWL Ontologies from UML Class Diagrams Based on Meta-Modelling and Graph Grammars

Aissam Belghiat, Mustapha Bourahla

***Abstract*—**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 principal language for knowledge representation. Building ontologies from scratch is generally a difficult task. The bridging between UML and OWL appeared on several regards such as the classes and associations. In this paper, we have to profit from convergence between UML and OWL to propose an approach based on Meta-Modelling and Graph Grammars and registered in the MDA architecture for the automatic generation of OWL ontologies from UML class diagrams. The transformation is based on transformation rules; the level of abstraction in these rules is close to the application in order to have usable ontologies. We illustrate this approach by an example.

***Keywords*—**ATOM3, MDA, Ontology, OWL, UML

I. Introduction

UML is the unified object oriented modeling language which became an important standard. In the other side, the ontologies became the backbone of the semantic web which described formally using a standard language called OWL (Ontology Web Language). In this work we propose an approach for transforming UML class diagrams into ontologies described in OWL language in order to profit from the visual expressivity of the notation language UML and 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 overlaps, gaps, and integration barriers between modeling languages and get greater value out of the information capture. In addition to that, we benefit from UML in order to have models on ontologies to make preliminary analyzes and OWL implementations to test ontologies consistencies. This approach is based on the combined use of Meta-Modelling and Graph Grammars to automatically generate OWL ontologies from UML class diagrams. We use the meta-modelling tool AToM3 to propose and implement a class diagram meta-model, after that we generate automatically a visual modelling tool to process class diagrams. We also define a graph grammar to translate the models created in the generated tool to OWL ontologies in RDF/XML format.

Aissam Belghiat is with the Department of Computer Science, Faculty of Engineering, University Md Boudiaf Msila, Algeria (e-mail: belghiatissam@gmail.com).

Mustapha Bourahla is with the Department of Computer Science, Faculty of Engineering, University Md Boudiaf Msila, Algeria (e-mail: m_bourahla@hotmail.fr).

The rest of the paper is organized as follows: In Section 2, we present some related works. In Section 3, we present some basic notions about UML, OWL, and their bridging. In Section 4, we present concepts about model transformation and graph transformation, and then we 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 tool through an example. Finally concluding remarks drawn from the work and perspectives for further research are presented in Section 7.

II. Related Works

The idea of our work is not innovating, indeed several works exist in the literature tackle this subject. In [14] the authors proposed a transformation of UML to-wards DAML at the end of the Nineties, by showing similarities and differences between the two languages. In [15] the work of "Converting UML to OWL Ontologies" proposed a transformation of Ontology UML Profile (OUP) towards an ontology OWL. In [6] the OMG notices the interest of such subject and pro-posed 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 works are carried out like answer to the call of the OMG and gathered in the ODM that we do not evoke here. In [9], the author presented an implementation of the ODM using ATL language. In [5], the author used a style sheet "OWLfromUML.xsl" applied to an XMI file (intermediate format of UML model) to generate an ontology OWL DL represented as RDF/XML format. And finally in [16], the authors proposed a detailed comparison between UML and OWL that 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 [1,17,18] we can found treatment of class diagrams, activity, and other UML diagrams. In these works the Meta modeling allows visual modeling and graph grammar allows the transformation. Obviously, the heart of our work is articulated on transformation rules and their implementation. In preceding works, the transformation rules are more specific and reflect a general opinion of the author often related to a specific field which he works on (specific transformation). In this paper we propose another vision different from that approached in preceding works either in the proposition of transformation rules, or in theirs implementation, this vision is to propose the transformation rules in a level of abstraction close to the application in order to obtain usable ontologies, because more the selected level of abstraction is close to the application minus ontology is reusable, but more it is usable. Then we propose a graph grammar implementation for these rules.

III. 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 [7]. UML defines thirteen diagrams; some of them represent the system statically while others show the functionalism of the system. The class diagram is considered as the most important of object oriented modeling, it shows the internal structure of a system and makes it possible to provide an abstract representation of its objects [2]. OWL (Ontology Web Language), was recommended by the W3C in 2004, and its version 2 in 2009, 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, and OWL2 RL [13]. UML and OWL have different goals and approaches; however they have some overlaps and similarities, especially for representation of structure (class diagrams). UML and OWL comprise some components which are similar in several regards, like: classes, associations, properties, packages, types, generalization and instances [6]. 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.

IV. Model Transformation

A. 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 defining 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 reads the source model which must be conform to the source meta-model, and applies the rules defined in the model transformation to lead to the target model which will be itself conform to the target meta-model. The principle of model transformation is illustrated by fig. 1.

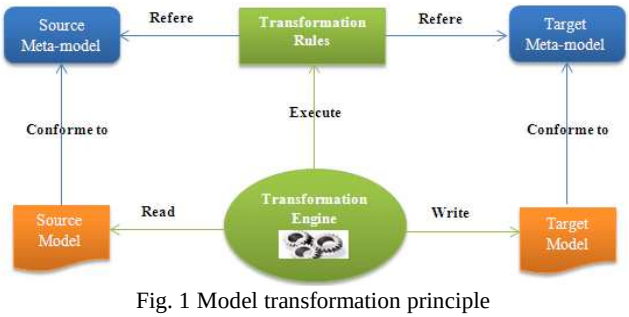


Fig. 1 Model transformation principle

B. Graph Transformation

Graph transformation was largely used for the expression of model transformation [4]. 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 left hand side (LHS) pattern and of a right-hand sided (RHS) pattern.

Therefore, the graph transformation is the process to choose a rule among the graph grammar rules, apply this rule on a graph pattern that is matched with the LHS pattern to produce the RHS pattern, and reiterate the process until no rule can be applied [4].

C. AToM3

AToM3 [1] "A Tool for Multi-formalism and Meta-Modeling" is a visual tool for model transformation, written in Python [8] and is carried out on various platforms (Windows, Linux, …) [18]. It implements various concepts like multi-paradigm modeling, meta-modeling 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 graph grammar 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.

V. Our Approach

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 AToM3 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 at the end a usable description of 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, datatypes, datatype properties, objects properties, annotation properties, ontologies properties, individuals, data values, and integrated vocabulary [11]. That means, for example, a class cannot be at the same time an individual [12]. These constraints enable us to lead to our objective which is an ontology well reflecting what is modeled in a class diagram.

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 on the class diagram.
4) An OWL file is generated automatically which contains the OWL ontology represented in RDF/XML format.
5) Visualization and use of OWL ontology by using special tools (Protégé, Swoop…).

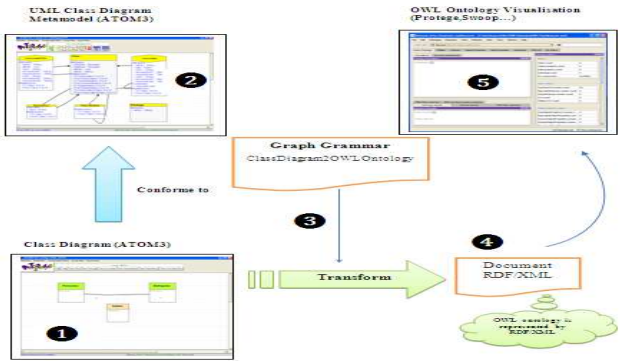


Fig. 2 Transformation sequence

A. Transformation rules

Our approach is realized according to suggested transformation rules (Table I). We propose a set of rules for all 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 presented some of the rules.

TABLE I
UML TO OWL TRANSFORMATION RULES

Class
An UML class is transformed to an OWL class; the name of the class is preserved.
Inheritance
The specialized class is defined subclass of the generalized class.
Class Attributes
An attribute is transformed into a property, and the transformation is carried out according to the type of attribute. 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.
Bidirectional association
Associations are transformed into object properties. An inverse object property is generated automatically named (Inverse-associationname)
Roles
Roles transformation is based on the representation by attributes (implementation level). Thus the situation of attributes transformation.
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).

B. Datatypes transformation

UML data types are transformed into XML schema (XSD) data types because OWL uses the majority of the datatypes integrated into XML schema. The instances of the primitive types used in UML itself include: Boolean, Integer, String, and UnlimitedNatural [7]. Table II presents the UML primitive datatypes and their transformations.

TABLE II
DATATYPES TRANSFORMATION

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

C. Meta-model of UML Class diagram

To build UML class diagram models in AToM3, we have to define a meta-model for them. Our meta-model is developed by the meta-formalism (CD_classDiagramsV3), and the constraints are expressed in Python [8] code (see fig.3):

We have proposed to meta-model class diagrams two Classes to describe packages and classes, and four associations to describe association relations, association class relations, generalization relations, and dependency relations.

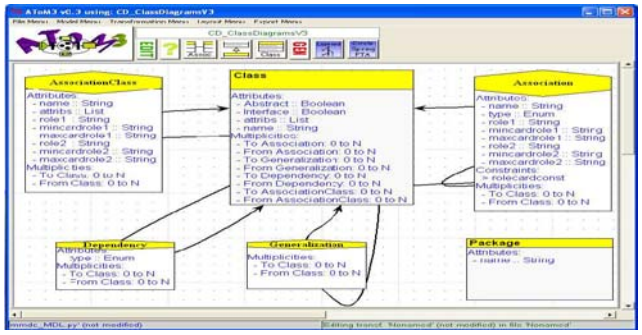


Fig. 3 Class diagram meta-model

After we built our meta-model, it remains only its generation. The generated meta-model comprises the set of classes and associations modeled in the form of buttons which are ready to be employed for a possible modeling of a class diagram.

Fig.4 shows the generated class diagram tool and a dialog box to edit a class. Each class has a name, and a list of attributes, it can be also abstract, interface or enumeration. All these attributes are defined in the proposed Meta-model (fig.3).

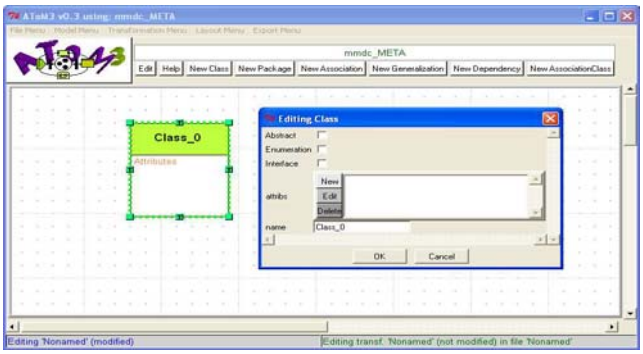


Fig. 4 Generated class diagram tool

D.The Proposed Graph grammar

To perform the transformation between class diagrams and OWL ontologies, we have proposed a graph grammar 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 with sequential access in order to store generated OWL code. We begin by writing the ontology header which is fixed for all our generated ontologies (see fig. 5).

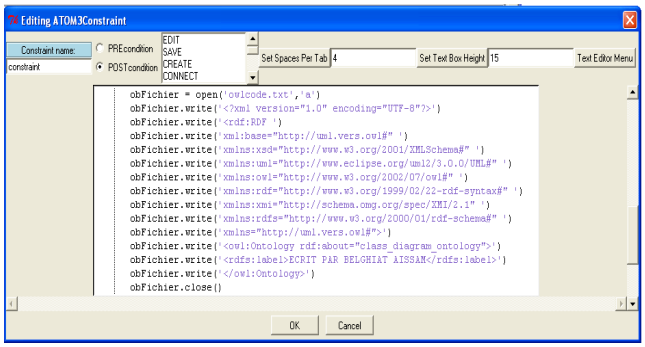
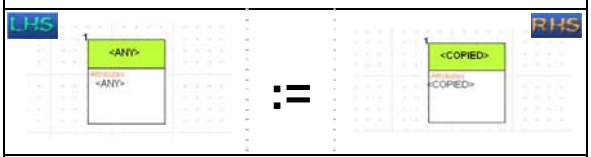


Fig. 5 Ontology header definition

Rule 1: Class transformation
Name: class2class
Priority: 1
Role: This rule transforms an UML class towards an OWL class (see Table III). In the condition of the rule we test if the class is already transformed, if not, in the action of the rule we reopen the OWL file to add the OWL code of this class.

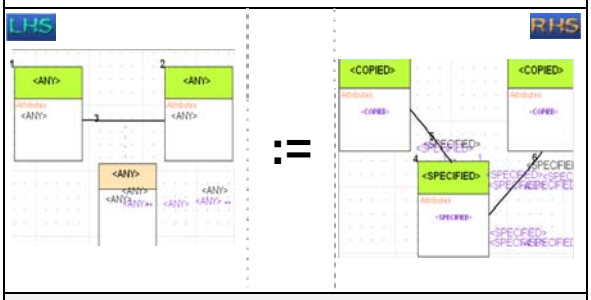
TABLE III
CLASS TRANSFORMATION

Condition
`node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))` `return not hasattr(node, "rule executed")`

Action


```
    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()
```

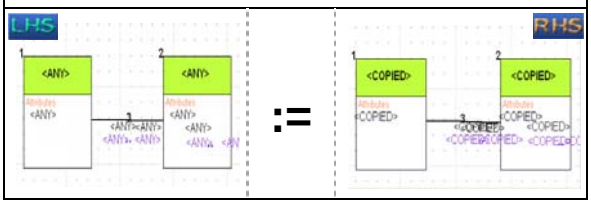
Rule 2: Association-class transformation
Name: ac2class
Priority: 2
Role: This rule allows the promotion of association class to a full class (see Table IV), that reflects what we show in the transformation rules. This class takes as name the name of the LHS class-association preceded by (AC-). Two binary associations are created in the RHS named AG_AC, AD_AC, thus two new roles RG_AC and RD_AC as illustrated in the transformation rules.

TABLE IV
ASSOCIATION-CLASS TRANSFORMATION

Condition
No condition

Action
No action

Rule 3: Binary association transformation
Name: asso2prop
Priority: 3
Role: This rule transform an association of the class diagram towards an OWL object property, it transforms also roles and cardinalities of the association (see Table V).

TABLE V
ASSOCIATION TRANSFORMATION

Condition
`node = self.getMatched(graphID, self.LHS.nodeWithLabel(3))` `return not hasattr(node, "rule4 executed")`


Action

```
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))
class1name = 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 bidirectionelle ou aggregation
    obFichier = open('owlcode.txt','a')
    obFichier.write('<owl:ObjectProperty rdf:ID="'+assoname+'">')
    obFichier.write('<rdfs:domain rdf:resource="#'+class1name+'"/>')
    obFichier.write('<rdfs:range rdf:resource="#'+class2name+'"/>')
    obFichier.write('<owl:inverseOf rdf:resource="#Inverse-'+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="#'+class1name+'"/>')
    obFichier.write('<owl:inverseOf rdf:resource="#'+assoname+'"/>')
    obFichier.write('</owl:ObjectProperty>')
    obFichier.close()
elif typ == 1 or typ == 3:# association unidirectionelle ou composition
```

Final Action: Definition of the end of ontology

Role: In the final action of the graph grammar, we end our ontology, So that becomes possible, we will have to open our file and to add '</rdf:RDF>' (see fig. 6).

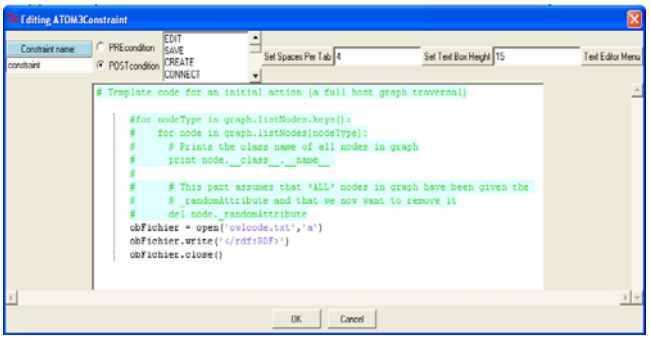


Fig. 6 End of ontology

VI. Example

Let us apply our approach on the example illustrated in figure 7, which models the situation that a person occupies a job in a company. To model this situation, we use two classes, "person" and "company", and an association class "Job".

A person has a full name "Per_Name", a company has also a name "Ent_Name". Moreover a person occupies only one work at the same time and a company employed several persons. Furthermore each person occupies a job must have a remuneration "remuneration" according to occupied work.

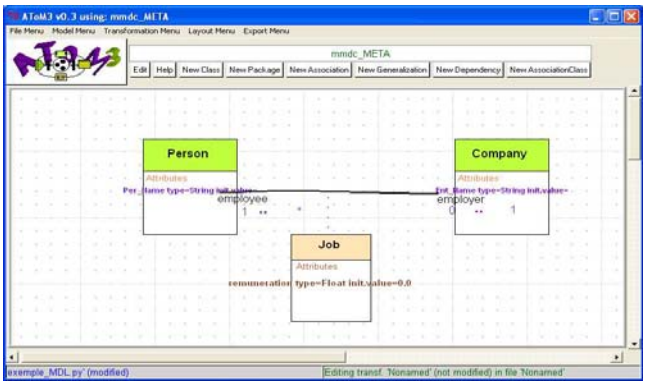


Fig. 7 Class diagram of our example

We start the execution of our graph grammar; we obtain the following intermediate graph (see fig. 8):

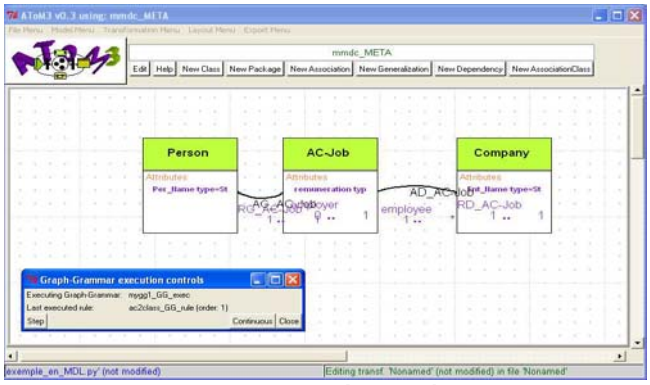


Fig. 8 Intermediate graph

After the execution of the graph grammar on our example we obtain the diagram illustrated in figure 9:

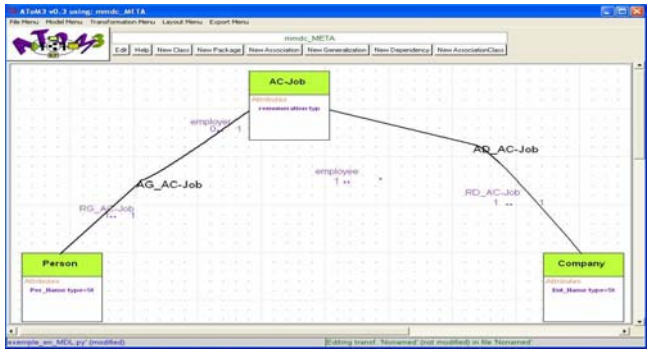


Fig. 9 Class diagram after execution

In parallel, there is an automatic generation of the file which contains OWL code stored on hard disc (see fig. 10):

Fig. 10 Generated OWL ontology

VII. Conclusion

We saw in this paper how to implement an application which makes a transformation from an UML class diagram to an OWL ontology based on graph transformation and by using the tool AToM3.

For the realization of this application we developed a meta-model for UML class diagrams, and a graph grammar composed of several rules which enables us to transform all what is modeled in our AToM3 generated environment to an OWL ontology stored in a hard disk file.

In future work, we plan to extend the transformation of semantic rules models towards the language of rules SWRL (Semantic Web Rule Language).

References

[1] AToM3. Home page: http://atom3.cs.mcgill.ca.2002.
[2] Laurent AUDIBERT, "UML2", http://www.lipn.univparis13.fr/audibert/pages/enseignement/cours.htm, 2007.
[3] Fowler, Martin, "UML Distilled - Third Edition - A Brief Guide to the Standard Object Modeling Language", 2003.
[4] 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.
[5] Sebastian Leinhos, http://diplom.ooyoo.de, 2006.
[6] OMG, "Ontology Definition Metamodel", V1.0, http://www.omg.org/spec/ODM/1.0, May 2009.
[7] OMG, "Unified Modeling Language (OMG UML) Superstructure", version 2.3, http://www.omg.org/spec/UML/2.3/Superstructure. 2010.
[8] Python. Home page: http://www.python.org.
[9] SIDo Group, "ATL Use Case - ODM Implementation (Bridging UML andOWL)",http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/, 2007.
[10] Deborah L. McGuinness and Frank van Harmelen, "OWL Web Ontology Language-Overview", http://www.w3.org/TR/2004/REC-owl-features-20040210/. W3C Recommendation 10 February 2004.
[11] Michael K. Smith, Chris Welty and Deborah L. McGuinness, "OWL Web Ontology Language–Guide", http://www.w3.org/TR/2004/REC-owl-guide-2004 0210. W3C Recommendation 10 February 2004.
[12] Mike Dean, Guus Schreiber, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, "OWL Web Ontology Language-Reference",http://www.w3.org/TR/2004/REC-owl-ref-20040210. W3C Recommendation 10 February 2004.
[13] 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.
[14] 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".
[15] Dragan Gašević, Dragan Djurić, Vladan Devedžić, Violeta Damjanović "Converting UML to OWL Ontologies", 2004.
[16] Kilian Kiko, Colin Atkinson, "A Detailed Comparison of UML and OWL",2008.
[17] Bardohl, R., H. Ehrig, J. De Lara and G. Taentzer (2004). "Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation". Lecture Notes in Computer Science 2984, pp.: 214-228.
[18] A. Chaoui, R. Elmansouri, Wafa Saadi, and E. Kerkouche, From UML Sequence Diagrams to ECATNets: a Graph Transformation based Approach for modelling and analysis, 2008.

Aissam Belghiat is a Student in the department of Computer science, University of Msila, Algeria. His research field is model transformation and ontology engineering.

Mustapha Bourahla is Professor in the department of Computer science, University of Msila, Algeria. His research field is formal methods.