From UML class diagrams to OWL ontologies: a Graph transformation based Approach

Aissam BELGHIAT, Mustapha BOURAHLA

Department of Computer Science, University of Md Boudiaf, Msila, 28000, Algeria

Belghiatissam@gmail.com
mbourahla@hotmail.com

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 description language adopted by a great community of users) the principal language for knowledge representation. The bridging 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 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.

Keywords: UML, Ontology, OWL, ATOM3, MDA.

1 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 a set of rules for transforming classes diagrams into OWL ontologies 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 overlaps, gaps, and integration barriers between modeling languages and get greater value out of the information capture. These rules will be implemented within AToM3 to automate this transformation.

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. In Section 4, we present concepts about model and graph transformation. In Section 5, we describe our approach. Finally concluding remarks drawn from the work and perspectives for further research are presented in Section 6.

2 Related Works

The idea of our work is not innovating, indeed several works exist in the literature tackle this subject. In [6] 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. 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 to generate an ontology OWL DL represented as RDF/XML format. 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] and other works 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 that transformation rules are in a level of abstraction close to the application in order to obtain usable ontologies.

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 [7].

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 [10].

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 [2], whereas OWL is a notation for knowledge representation, but both are modeling languages.

4 Graph Transformation

Model transformation play an essential role in the MDA. MDA recommends the massive use of models in order to allow a flexible and iterative development.

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 (see fig. 1).

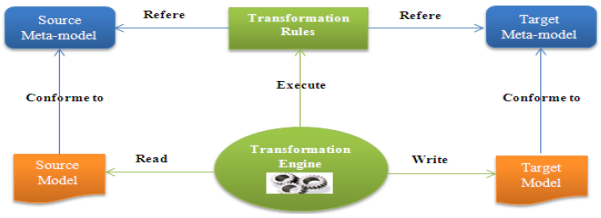


Fig. 1. Model transformation principle.

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, each rule of a graph grammar is composed of a left hand side (LHS) pattern and of a right-hand sided (RHS) pattern.

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

5 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 (fig.2), 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 (fig.2). 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 [10][11].

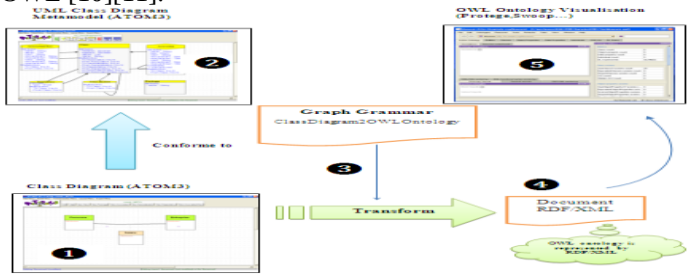


Fig. 2. Transformation sequence.

5.1 Transformation rules

Our approach is realized according to suggested transformation rules (Table 1). We propose a set of rules for all elements of a class diagram. The level of abstraction of rules is close to the application. For lack of space, we have presented one rule.

Table 1. UML to OWL Transformation rules.

Class
An UML class is transformed to an OWL class; the name of the class is preserved.
<owl:Class rdf:ID="**ClassName**"/>

5.2 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 composed of two classes and four associations developed by the meta-formalism (CD_classDiagramsV3), and the constraints are expressed in Python [8] code (fig.3):

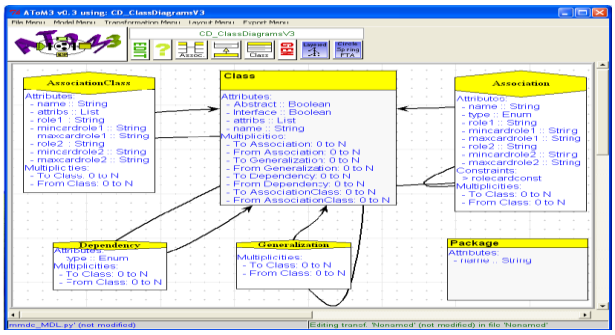


Fig. 3. Class diagram meta-model.

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

5.3 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. Then we begin by writing the ontology header which is fixed for all our generated ontologies (fig. 4).

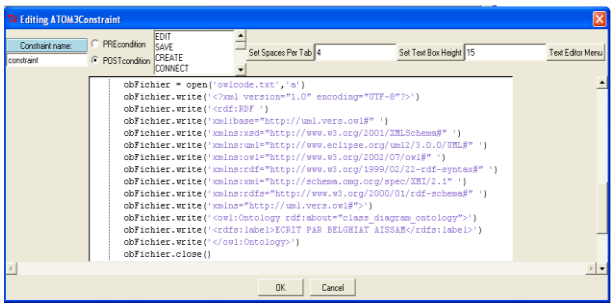


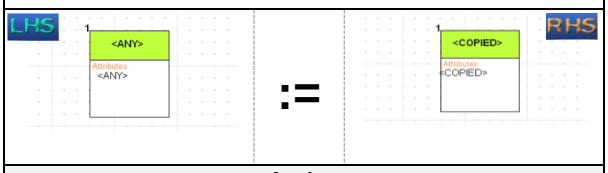
Fig. 4. Ontology header definition.

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 reopen the OWL file to add the OWL code of this class.

Table 2. 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()
```

Final Action: **Definition of the end of ontology**
Role: In the final action of the graph grammar, we end our ontology, we will have to open our file and to add '</rdf:RDF>' (cf. fig. 5).

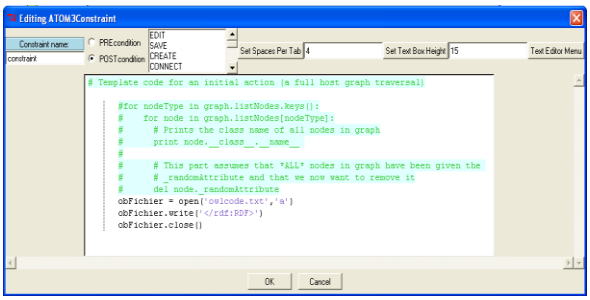


Fig. 5. End of ontology.

6 Conclusion

We saw in this paper how to implement an application which makes a transformation from a 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).

7 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, Springer, juillet 2004.
5. Sebastian Leinhos, http://diplom.ooyoo.de, 2006.
6. OMG, “Ontology Definition Metamodel”, http://www.omg.org/spec/ODM/1.0, May 2009.
7. OMG, “OMG Unified Modeling Language, Infrastructure,v2.3”,http://www.omg.org/spec/UML/2.1.2/Infra structure/PDF, May 2010.
8. Python. Home page: http://www.python.org.
9. SIDo Group, “ATL Use Case - ODM Implementation (Bridging UML and OWL)”,http://www.eclipse.org /m2m/atl/usecases/ODMImplementation/, 2007.
10. W3C OWL Working Group, “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 Web Ontology Language–Guide”, http://www.w3.org/TR/2004/REC-owl-guide-20040210. W3C Recommendation 10 February 2004.