Appling MDA approach to solve the interoperability problem in modelio and Argouml tools

Wafae Lahlayl, Naoual Berbiche, Fatima Guerouate, Mohamed Sbihi

Abstract: ***The modeling of computer systems is applied by the approach called MDA (Model Driven Architecture) which recommends presenting the computer system in the form of models; these are described by UML modeling software. These models are serialized by another standard published by the OMG called XML Metadata Interchange (XMI) which is recommended to ensure interoperability between UML modeling tools. However, the independent evolutions of these standards; namely UML and XMI; have caused interoperability problems between UML modeling softwares. In the article "Solving the Interoperability Problem between UML Modeling Tools: Modelio and ArgoUML", we solved the problem of interoperability at the level of the XMI files of class diagrams of UML language and mainly between Modelio and ArgoUML using the transformation by XSLT template. This paper presents, on the one hand, a hybrid transformation approach via ATL to transform XMI files to solve the interoperability problem and, on the other hand, a comparison of this solution to the previous transformation by XSLT models.***

Index Terms: ***MDA, Interoperability, UML, XMI, ATL, Modelio, ArgoUML, class diagram, XSLT.***

I. INTRODUCTION

IT development is an essential element in the organization of our companies and of our society in general. Today, computer systems have been built from the aggregation of several applications that need to be maintained and scaled flexibly and without difficulty. The objectives are, among other things, to improve the quality of the services offered while maintaining the autonomy of the actors, the setting up of the computer systems, a coherent management of information, reduced production times and a better control of maintenance costs. However, setting up interoperability of computer systems is difficult at both the conceptual and technical levels.

This paper is a continuation of the previous article in which the interoperability issues related to the phase of design in computer development were presented. And more precisely the interoperability of data between modeling tools. These allow presenting the architecture of computer software in the form of models. This approach is called Model Driven Architecture (MDA). The preferred language for describing MDA is UML. The storage and exchange of UML models is done by a standard published by the OMG (Object Management Group, http://www.omg.org/): XML Metadata Interchange (XMI).

However, the UML and XMI standards have evolved separately. The models can therefore be described in several versions of UML and serialized in several versions of XMI. This situation produces to non-interoperable models.

In Section 2, we present a state of the art where we give insights on model transformation approaches in the MDA framework and mainly the hybrid approach through the ATL language. In section 3, we present the motivations that led us to do this work. Section 4 discusses the approach used and presents the meta-models and ATL transformation rules executed on XMI files. And finally, section 5 presents a comparison between the XSLT and ATL transformation.

II. STATE OF ART

A. *Models Transformation Approaches*

In the MDA context, there are different approaches to performing a model transformation, each providing a very specific way for the elaboration of transformation rules. We can divide the approaches into two types of transformation: the Model to Text (M2T) transformation and the Model to Model (M2M) transformation.

• M2T (Model to Text) transformations: these are transformations that aim for generating codes or documentations. M2T transformations are the subject of the MOFM2T project, which is one of the parts of the MDA project [1].

• Model to Model (M2M) transformations: These are model-to-model transformations; this type of transformations affects all the tasks to be executed in order to achieve a model that meets the technical specifications of the target environment starting from another model. The technological base that technically represents this type of transformation in the MDA approach is the MOF 2.0 QVT standard.

The following figure provides an overview of the MDA [2] transformation.

Manuscript published on 30 June 2019.
* Correspondence Author (s)
Wafae Lahlayl*, LASTIMI Laboratory, Superior School of Technologies of Sale, Mohammadia School of engineering, Mohamed V University city of Rabat, Morocco.
Naoual Berbiche, LASTIMI Laboratory, Superior School of Technologies of Sale, Mohammadia School of engineering, Mohamed V University city of Rabat, Morocco.
Fatima Guerouate, , LASTIMI Laboratory, Superior School of Technologies of Sale, Mohammadia School of engineering, Mohamed V University city of Rabat, Morocco.
Mohamed Sbihi, , LASTIMI Laboratory, Superior School of Technologies of Sale, Mohammadia School of engineering, Mohamed V University city of Rabat, Morocco.

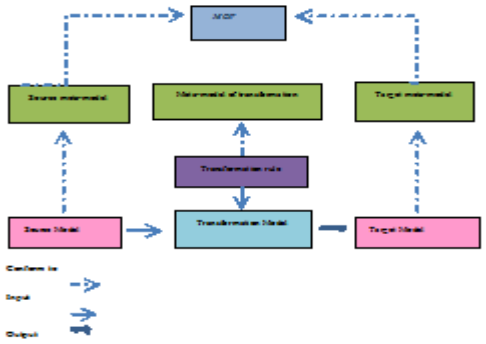

Fig. 1. Overview of MDA transformation

There are different approaches to perform a transformation of M2M type models, each offering a specific way for the elaboration of transformation rules.

✓The programming approach:

This approach is based on the use of programming languages in order to describe the transformation rules as computer programs, and so the model transformation is a computer application that manages models. This approach is considered a direct manipulation approach [3]. The programming approach has two different visions:

1) The imperative approach: Similar to imperative programming, this approach relies on the foundations of structured programming and defines the transformations as instruction sequences contained in functions and procedures in order to change the state of the model at runtime. The advantage of such an approach lies in the fact that it remains relatively familiar to developers, the syntax and semantics it uses is close to the Object Constraint Language(OCL) [4].

2) The API approach: Some transformation APIs (Application Programming Interface) are described in the imperative programming languages and are then provided as libraries allowing the description of the transformation process according to the syntax of the language used. And so the results are of a considerable performance. However, the developer is responsible for the organization and description of all the steps explicitly in terms of mandatory statements [4].

✓The modeling approach:

Also known as the rule transformation approach, this approach consists in modeling the transformations themselves in order to make them durable and productive by applying on them model engineering. It can be concretized by one of two approaches:

1) The graph approach: This is a mathematical formalism that applies the theory of graphs on the transformation of models, considering these ones as graphs. The transformation strategy in this approach is to replace and match the source and target model, which uses graphs rule syntax to take a Left Hand Side Graph (LHSG) and transform it into a RHSG. (Right Hand Side Graph) [3]. the complexity of this approach lies in the non-deterministic aspect in the strategy of application of transformation rules [3], which implies that solutions based on this paradigm are used very little in the concrete.

2) The declarative approach: In this approach, a rule maps a set of concepts invoked at the source model to a set of concepts that should be adopted at the target model. The implementation is performed by an inference engine. However, one of the main disadvantages of this approach is the high workload that must be performed by the developer, who must specify all the support constraints for the transformation, a task that is considered tedious [5].

✓The template approach:

It consists of defining parametric patterns for the target models. The parameters are provided by the values contained in the source models so that the output model can be mounted. The reference implementation representing this approach is the XSLT approach that implements the XSLT language (eXtensible Stylesheet Language Transformations). This language was originally designed for the transformation of XML documents (eXtensible Markup Language) into other formats. In an MDA framework, models are serialized in XMI format (XML Metadata Interchange) [6] which is a variant of the XML language; this explains why such an approach suits perfectly the model transformations in such a context. Nevertheless, this approach has disadvantages of performance and efficiency once the models to be transformed reach a certain level of complexity.

✓The hybrid approach:

This approach is the most recent of the other transformative approaches; it combines the declarative and imperative approaches. The declarative approach is generally used for the definition and selection of transformations that can be applied, while the imperative approach is well adapted to the description of the transformation strategy [4].

Figure 2 shows the hierarchy of the M2M transformation

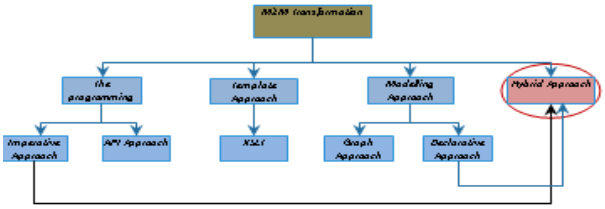

Fig. 2. Different approaches of M2M model transformations

Our work consists of two model transformations (of the M2M type): the first step is to adapt the structure of the models obtained by the serialization process from the ArgoUML tool[7] to the format recommended by Modelio and in a second step, to make a transformation in the opposite way, namely to adapt the models obtained by Modelio[8] to the format used by ArgoUML. In order to achieve this, we are interested in the hybrid approach through the ATL language.

B. *Atlas Transformation Languages (ATL)*

The ATL transformation language (ATLAS Transformation Language)[9] is a response from the INRIA and LINA research group to OMG MOF (Meta Object Facilities (MOF, 2003)) / QVT (Query View Transformation (QVT, 2010)). ATL is a model transformation language in the field of Model Driven Engineering (MDE). It provides developers with a mean to specify how to produce a number of target models from source models [10].

ATL is also a Plugin of Eclipse whose metamodels conform to Ecore [11].

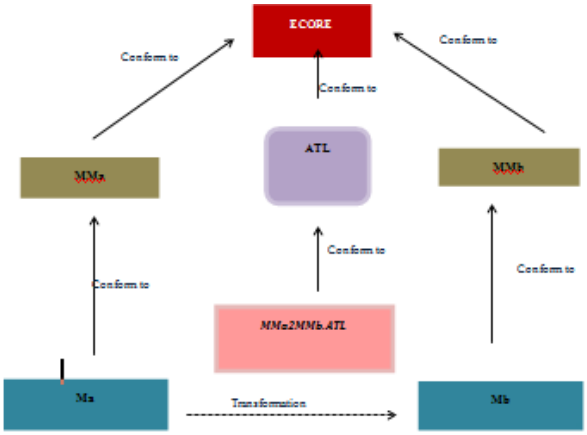

Fig. 3. ATL Operational Framework

III. MOTIVATIONS

Before applying the MDA approach on Modelio and ArgoUML, we observed the XMI files produced by each UML tool. The table I shows the XMI import and export versions of each tool and the UML version.

Table I. XMI Import and Export

Tools	ArgoUml	Modelio
XMI Import version	1 1.1 1.2	2.1
XMI export version	1.2	2.1
UML version	1.4	2.1.1 2.2 2.3 2.4.1
Used Version	Version 0.34	Version 3.6.0
Current Version	Version 0.34	Version 3.6.0
category	Free	Free

We can note that there is a correspondence between XMI and UML. Going back to the specifications of the OMG, we could detail all the different existing version of UML and XMI [12]. The table II presents a part of each XMI file of the argouml and modelio modeling tools.During the observation of the XMI files of Modelio and Argouml, we noticed also that there is a syntactic difference in the tags [13]. The table III provides an overview of this finding by comparing the tags of each XMI file [13].

The problem of interoperability of the UML tools is spotted in the tags of the XMI file; we propose an MDA approach to solve this problem based on the use of the ATL hybrid transformation approach.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

Table II. The difference between Modelio and ArgoUML

Tools	ArgoUML	Modelio
XMI Files	<UML:Class xmi.id = '-64--88-1-101--205ea92b:1645c2b0855:-8000:0000000000000962' name = 'Personne' visibility = 'public' isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false' isActive = 'false'> <UML:Classifier.feature> <UML:Attribute xmi.id = '-64--88-1-101--205ea92b:1645c2b0855:-8000:0000000000000963' name = 'cin' visibility = 'public' isSpecification = 'false' ownerScope = 'instance' changeability = 'changeable' targetScope = 'instance'> <UML:StructuralFeature.multiplicity> <UML:Multiplicity xmi.id = '-64--88-1-101--205ea92b:1645c2b0855:-8000:0000000000000964'> <UML:Multiplicity.range> <UML:MultiplicityRange xmi.id = '-64--88-1-101--205ea92b:1645c2b0855:-8000:0000000000000965' lower = '1' upper = '1'/> </UML:Multiplicity.range> </UML:Multiplicity> </UML:StructuralFeature.multiplicity> </UML:StructuralFeature.type> </UML:Attribute>	<packagedElement xmi:type="uml:Class" xmi:id="_vG5x8H4jEeiMnbkorutwtA" name="Personne"> <ownedAttribute xmi:type="uml:Property" xmi:id="_vG5x8X4jEeiMnbkorutwtA" name="cin" visibility="public" isUnique="false"> <type xmi:type="uml:PrimitiveType" href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#String"/> </ownedAttribute>

Table III. Comparison of the tags describing the elements of the class diagram

Tools	ArgoUML	Modelio
Package	<**UML:Package** xmi.id=" " name="Package" >	<packagedElement xmi:type="**uml:Package**" xmi:id=" " name="Package">
Class	<**UML:Class** xmi.id=" " name="Membre">	<packagedElement xmi:type="**uml:Class**" xmi:id="" name="Membre">
Attribute	<**UML:Attribute** xmi.id=" " name="nom" visibility="private">	<**ownedAttribute** xmi:type="**uml:Property**" xmi:id=" " name="nom" visibility="private">
Method	<**UML:Operation** xmi.id=" " name="setNom" visibility="public">	<**ownedOperation** xmi:type="**uml:Operation**" xmi:id=" " name="setNom" visibility="public"/>
Association	<**UML:Association** xmi.id=" " name="">	<packagedElement xmi:type="**uml:Association**" xmi:id=" " >
Aggregation	<UML:AssociationEnd xmi.id=" " name="" visibility="public" **aggregation="aggregate"** >	<ownedAttribute xmi:type="uml:Property" xmi:id=" " name=" " visibility="public" type=" " **aggregation="shared"** association=" ">
Composition	<UML:AssociationEnd xmi.id=" " name="" visibility="public" **aggregation="composite"** >	<ownedAttribute xmi:type="uml:Property" xmi:id=" " name=" " visibility="public" type="" **aggregation="composite"** association="">
Generalization	<**UML:Generalization** xmi.id=""> <UML:Generalization.child> <UML:Class xmi.idref=""/> </UML:Generalization.chil> <**UML:Generalization.parent**> <UML:Class xmi.idref=""/> </UML:Generalization.parent> </UML:Generalization>	<generalization xmi:type="**uml:Generalization**" xmi:id=" " **general**=" "/>
Multiplicity	<UML:MultiplicityRange xmi.id=" " **lower**="" **upper**=""/>	<**upperValue** xmi:type=" " xmi:id=" " value=""/> <**lowerValue** xmi:type="" xmi:id="" value=""/>

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

IV. METAMODELS AND TRANSFORMATION RULES

To begin the transformation, we made a class diagram containing all the elements and the main relations of the class diagram as described in the following figure:

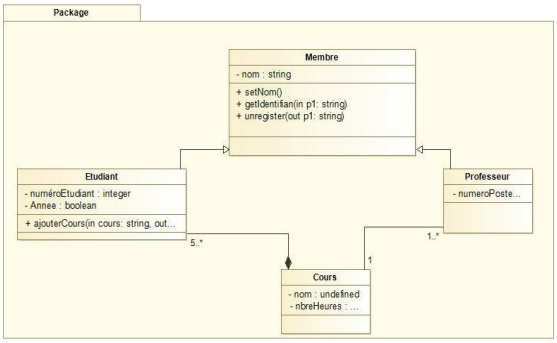

Fig. 4. The used class diagram

Then we imported the two XMI files from ArgoUML and Modelio. And as it is mentioned previously, it was necessary to create the Ecore model at the level of these two tools.
To achieve this, we created the XML schema of each XMI file. The following table provides an overview of the XML schema for each XMI file:

Table IV. XML schema of Modelio and ArgoUML

Tools	ArgoUML	Modelio
XML schema	<xs:complexType> <xs:sequence> <xs:element name="Namespace.ownedElement"> <xs:complexType> <xs:sequence> <xs:element name="Class"> <xs:complexType> <xs:sequence> <xs:element name="Classifier.feature"> <xs:complexType> <xs:sequence> <xs:element maxOccurs="unbounded" name="Attribute"> <xs:complexType> <xs:sequence> <xs:element name="StructuralFeature.multiplicity"> <xs:complexType> <xs:sequence> <xs:element name="Multiplicity"> <xs:complexType> <xs:sequence> <xs:element name="Multiplicity.range">	<xs:element name="packagedElement"> <xs:complexType> <xs:sequence> <xs:element name="ownedAttribute" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="type"> <xs:complexType> <xs:attribute name="xmi:type" type="xs:string"></xs:attribute> <xs:attribute name="href" type="xs:string"></xs:attribute> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="xmi:type" type="xs:string"></xs:attribute> <xs:attribute name="xmi:id" type="xs:string"></xs:attribute> <xs:attribute name="name" type="xs:string"></xs:attribute> <xs:attribute name="visibility" type="xs:string"></xs:attribute> <xs:attribute name="isUnique" type="xs:string"></xs:attribute> </xs:complexType> </xs:element>

Once the XSD (XML Schema) file was created, it was easy to make the two ECORE meta-models which are presented as class diagrams as it is shown below.
The first figure shows the Ecore meta-model of the XMI Modelio file and the second presents the Ecore meta-model of the ArgoUML XMI file.

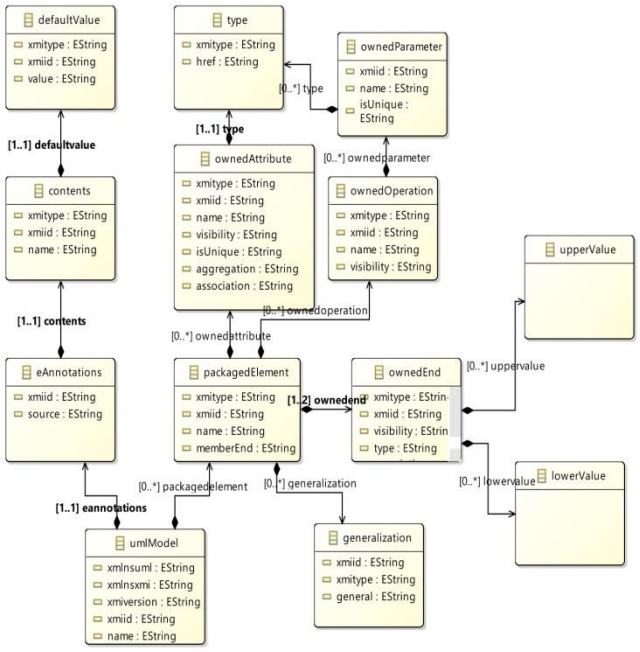

Fig.5. Eore meta-model of Modelio

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

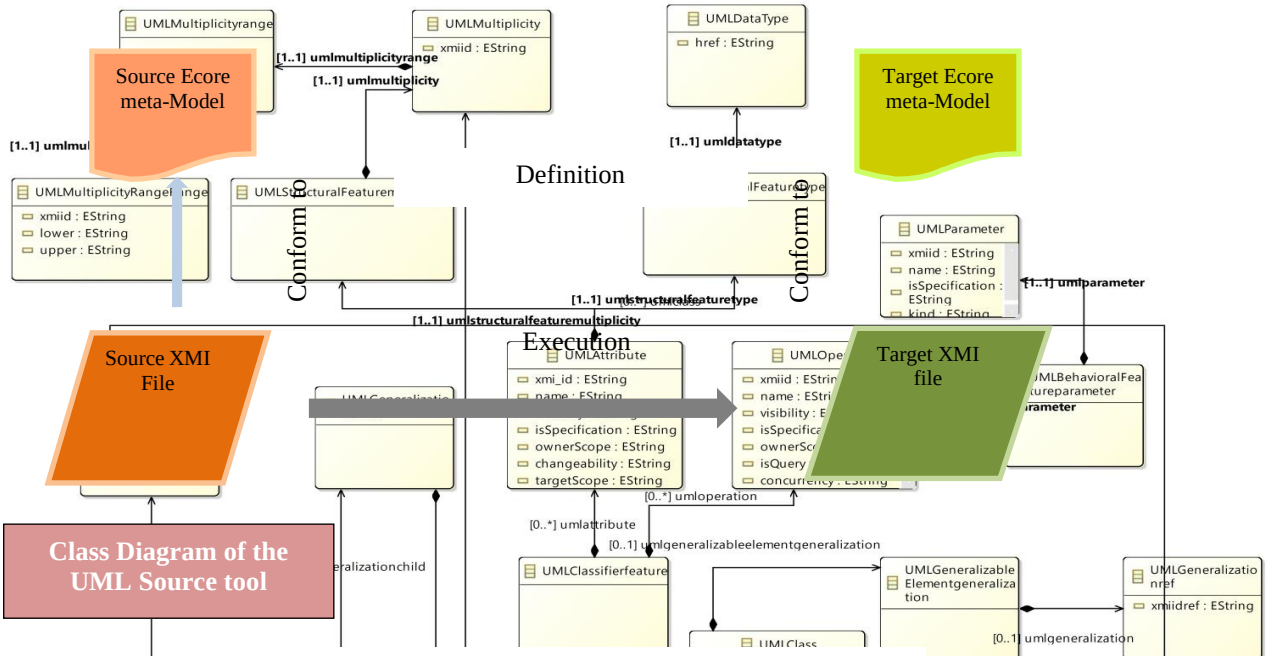

Fig.7. Architecture of transformation

Fig.6. Ecore meta-model of ArgoUML

Architecture and transformation rules:

To perform the ATL transformation, several steps were followed. These steps are presented in the figure 7.

We have created the corresponding Ecore meta-models for our source and target meta-models. However, it was necessary to correct before the XMI files since they contain special characters that will not be accepted in the ECORE meta-models.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

The process of implementing the solution via eclipse is shown in the figure12: this process corrects the XMI file exported by the source tool in order to establish the source ECORE metamodel and transform it into a destination ECORE metamodel then the target XMI file.

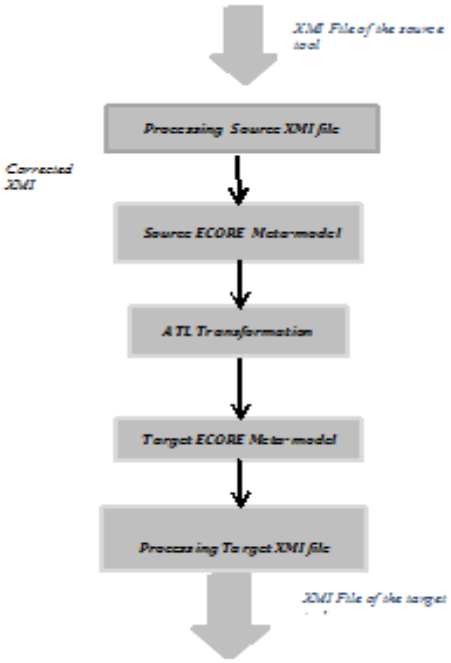

Fig. 8. Process of the transformation via eclipse

XSD File

Class Diagram of the UML Target tool

The correction of XMI files is established by a Java code, part of whose code is presented in the following figure:

```
private                static                void
updateElementsValue(Document doc) {
        NodeList              XMI1              =
doc.getElementsByTagName("XMI.header");
        Element xh = null;
       xh= (Element) XMI1.item(0);
doc.renameNode(xh, null, "xmiheader");
          // documentation

          NodeList              XMId              =
doc.getElementsByTagName("XMI.documentation")
;
          Element xd = null;
         xd= (Element) XMId.item(0);

            doc.renameNode(xd,           null,
"xmidocumentation");
          //
          //exporter
          NodeList              XMIe              =
doc.getElementsByTagName("XMI.exporter");
          Element xex = null;
```

For ATL Mod2Arg and Arg2Mod transformations, each ATL file has its own transformation rules. For example, the transformation of a class and its attributes from Argouml to Modelio and vice versa. Parts of these transformation files are presented in the following:

- *A part of the ATL Mod2Arg file*

```
        ---@path
Mod=/ATlModArg/metamodels/Mod.ecore
    ---@path
Argo=/ATlModArg/metamodels/argTest.ecore

module Mod2Argo;

          ---module template
    create OUT : Argo from IN: Mod ;


   rule xmiroote{
          from M : Mod!umlModel,
     GM : Mod!generalization
           to
     A : Argo!XMIRoot(

                 xmiversion<-'1.2',

xmlnsUML<-'org.omg.xmi.namespace.UML',
      xmicontent<-C,
      xmiheader<-H
            ),
     H : Argo!XMIheader(
      xmidocumentation<-DOC,
             xmimetamodel<-Met
           ),

          C : Argo!XMIcontent
               (
                umlmodel<-Mod
          ),

     Met : Argo!XMImetamodel (
       xminame<-'UML',
       xmiversion<-'1.4'
     ),

     Mod : Argo!UMLModel (
          xmiid<-'-'+ M.xmiid,
         name<-M.name,
         isSpecification<-'false',
         isRoot<-'false',
         isLeaf<-'false',
        isAbstract<-'false',
       umlnamespaceownedelement<-Nam
           )
```

A part of the ATL Arg2Mod file

```
          ---@path
Argo=/ATlArgMOD/metamodels/argTest.ecore
     ---@path
```


Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

```
Mod=/ATlArgMOD/metamodels/Mod.ecore

module Arg2Mod;
 ---module template
    create OUT : Mod from IN: Argo ;

rule UMLModel2umlModel {
  from
       A : Argo!UMLModel,
     C : Argo!XMIdocumentation,
     D : Argo!XMIexporterVersion,
     E : Argo!UMLClass,
     CL : Argo!UMLClassifierfeature,
     AS : Argo!UMLAssociation,
     ASE : Argo!UMLAssociationEnd,
     cl : Argo!UMLClass2,
     GA : Argo!UMLGeneralization
      to
    Cont : Mod!contents(
  xmitype<-'uml:Property',
  xmiid<-'_vG5K4n4jEeiMnbkorutwtA',
  name<-'exporterVersion',
  defaultvalue<-Df
    ),

  P : Mod!packagedElement(
    name<-E.name,
    xmiid<-E.xmiid,
    xmitype<-'uml:Class',
  ownedattribute<-CL.umlattribute->colle
ct(e |
thisModule.UMLAttribute2ownedAttribute(e
)),
  ownedoperation<-CL.umloperation->colle
ct(e |
thisModule.UMLOperation2ownedOp(e))
        )
```

After implementing these transformation rules and syntax correction, we obtained an XMI file, this file is then imported into the target tool and the class diagram is here viewable.

V. COMPARISON BETWEEN ATL AND XSLT

Comparing the transformation of XMI files with the XSLT Template model approach - object of our previous article[13] - with the hybrid transformation via the ATL language. We noticed that the XSLT transformation is more specific. In fact, the transformation rules are executed by reading each XMI tag, which makes the XSLT transformation more difficult in case we have a complex XMI file. Indeed, the process of writing XSLT stylesheets would be tedious. Unlike the ATL transformation, the transformation rules are more general and easy to define because the XMI file is metamodeled as a simple and clear ECORE model. It can be concluded that the ATL transformation is more practical and more efficient in model transformations between the models produced by the UML modeling tools.

VI. CONCLUSION

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, building, and documenting software system architectures. UML modeling tools use XMI files for storage and exchange of UML models. The interoperability of UML modeling tools can make UML models reusable and extensible. However, we found that the syntactic incompatibility of the XMI tags of these tools was the cause of the interoperability problem of UML diagrams.

In this paper, we carried out the hybrid transformation approach using the ATL language on the XMI files generated by the UML modeling tools Modelio and ArgoUML to solve the interoperability problem. We also compared the ATL transformation to the XSLT transformation that was the subject of our previous article. We concluded that the ATL transformation was more appropriate for solving the interoperability problem between UML modeling tools.

For future work, we will perform the ATL transformation on other modeling tools in order to extract all the common rules. This extraction will help us to develop a platform that will guarantee the interoperable exchange of UML diagrams.

REFERENCES

1. "Model To Text." [Online]. Available: http://www.eclipse.org/modeling/m2t/.
2. X. Blanc, MDA en action : Ingénierie logicielle guidée par les modèles. 1st edition, 270 pages, 2005.
3. Quyet Thang, "Model Transformation Reuse : A Graph-based Model Typing Approach," Universit é europé enne de Bretagne, Université de Rennes 1, 2012.
4. N. Cuong and X. Qafmolla, "Model transformation in web engineering and automated model driven development," Int. J. Model. Optim., vol. 1, no. 1, 2011.
5. M. Dehayni and K. Barbar, "Some Model Transformation Approaches: a Qualitative Critical Review," J. Appl. …, vol. 5, no. 11, pp. 1957–1965, 2009.
6. Number, OMG Document. XML Metadata Interchange (XMI) Specification. no. June,2015.
7. ''ArgoUML User Manual : A tutorial and reference description'' by Alejandro Ramirez, Philippe Vanpeperstraete, Andreas Rueckert, Kunle Odutola, Jeremy Bennett, Linus Tolke, and Michiel van der Wulp
8. Modeliosoft, the open source modeling environment, "Modelio." [Online]. Available: https://www.modelio.org/.
9. Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, "ATL: A model transformation tool" Science of computer programming elsevier Volume 72, n.1-2: pp. 31-39, 2008
10. F.jouault, "Contribution à l'étude des langages de transformation de modèles",PhD thesis, University of Nantes, 2006
11. Eclipse.org. ATLAS Transformation Language (ATL). http://www.eclipse.org/m2m/atl/.
12. Number, OMG Document. OMG Unified Modeling Language TM (OMG UML) Version 2.5. no. March, 2015.
13. W. Lahlayl, N. Berbiche, F. Guerouate and M.Sbihi "Solving the Interoperability problem between UML modeling tools: Modelio and ArgoUML", International Journal of Applied Engineering Research Volume 12, Number 19 (2017), pp. 8632-8641
14. A. Srai, F. Guerouate, N. Berbiche and H. Drissi "An MDA approach for the development of data warehouses from Relational Databases Using ATL Transformation Language ",International Journal of Applied Engineering Research Volume 12, Number 12 (2017), pp. 3532-3538
15. A. Srai, F. Guerouate, N. Berbiche and H. Drissi "Applying MDA approach for Spring MVC Framework ",International Journal of Applied Engineering Research Volume 12, Number 14 (2017) , pp. 4372-4381
16. Frederic Jouault , Ivan Kurtev, "On the interoperability of model-to-model transformation languages", Science of computer programming elsevier (2007)
17. Gaurav Bansal, Deepak Vijayvargiya, Siddhant Garg, and Sandeep Kumar Singh., "An Approach to Identify and Manage Interoperability of Class Diagrams in Visual Paradigm and MagicDraw Tools"
18. S. El Idrissi, N.Berbiche, F.Guerouate and M.Sbihi, "Performance evaluation of web application security scanners for prevention and protection against vulnerabilities", International Journal of Applied Engineering Research Volume 12, Number 21(2017) pp.11068-11076